

Jeffrey D. Ullman - Jennifer Widom

ADATBÁZIS- RENDSZEREK

Alapvetés

Második, átdolgozott kiadás

www.panem.hu

4. Sőt, ha a képek ennyi területet használnak fel, akkor a filmek még ennél is többet. Egy egyórás videó legalább egy gigabájtot foglal el. A YouTube-hoz hasonló oldalak filmek százezreinek vagy millióinak tárolását és könnyű elérését teszik lehetővé.
5. A *peer-to-peer* fájlmegosztó rendszerek hagyományos számítógépek nagy hálózatait használják az adatok tároláshoz és többféle megosztásához. Ha-bár egy hálózati csúcs csak néhány száz gigabájtnyi adat tárolását teszi lehetővé, a többi csúccsal együtt viszont már hatalmas mennyiséget tárolhatnak.

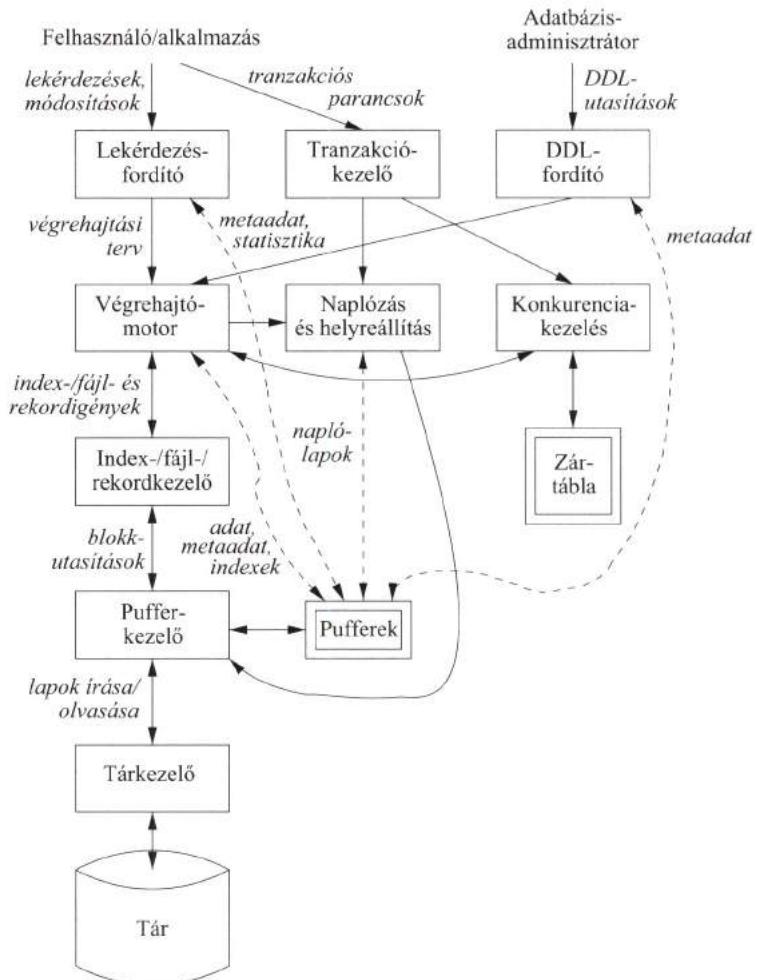
1.1.5. Információk egyesítése

Ahogy az információ szerepe egyre alapvetőbbé válik a munkában és a szórakozásban, azt figyelhetjük meg, hogy a meglevő információforrásokat egyre többféle módon hasznosítjuk. Például egy nagyvállalatnak sok részlege van, amelyek mindegyike külön adatbázissal rendelkezik a saját termékeiről, illetve alkalmazottairól. Természetesen néhány ilyen részleg maga is független vállalat, amelynek megvan a saját módszere a dolgok intézésére. Ezek a részlegek akár különböző ABKR-t, illetve adatstruktúrát is használhatnak. Használhatnak eltérő kifejezéseket ugyanazon dolognak a leírására, vagy azonos kifejezések különböző dolgok leírására. A helyzetet tovább rontja az olyan örökölt alkalmazások létezése, melyek az összes adatbázist használják. Így lehetetlenné válik ezen adatbázisok elvetése.

Ennek az eredményeként egyre gyakrabban szükségessé vált egy, a létező adatbázisok feletti struktúra kiépítése. Ezeknek az a célja, hogy a megosztott információkat egységesített formára hozzák. Az egyik ilyen népszerű megközelítés az *adattárházak* létrehozása, amelyekben a több adatbázisból örökölt információkat a megfelelő átalakítások használatával időszakosan kimásoljuk egy központi adatbázisba. Egy másik megközelítés egy olyan *közvetítő* (vagy „*middleware*”) megvalósítása, aminek a feladata a különböző adatbázisokból származó adatok egyesített modelljének támogatása ezen modell és az egyes adatbázisok aktuális modellje közötti átfordítással.

1.2. Az adatbázis-kezelő rendszerek áttekintése

Az 1.1. ábrán egy ABKR vázlatát láthatjuk. Az egyvonás téglalapok a rendszer komponenseit jelölik, a kétvonások pedig memóriabeli adatszerkezeteket. A folyamatos vonalak a vezérléseket és az adatáramlásokat jelölik, a szaggatott vonalak pedig azt, hogy az adott irányban csak adatok áramlanak. Mivel az ábra meglehetősen bonyolult, a részleteket több lépésben vizsgáljuk. Először tekintsük az ábra tetején az ABKR felé kiadható utasításokat, amelyek két különböző helyről érkezhetnek:



1.1. ábra. Egy adatbázis-kezelő rendszer részei

1. Hagyományos felhasználóktól és alkalmazói programuktól, amelyek lekérdezik vagy módosítják az adatokat.
2. *Adatbázis-adminisztrátortól*. Ez olyan személyt vagy személyeket jelent, akik felelősek az adatbázis szerkezetéért vagy más szóval *sémájáért*.

1.2.1. Adatdefiníciós nyelvi utasítások

A második fajta utasításokat egyszerűbb feldolgozni, először az ilyen utasítások feldolgozási útvonalát mutatjuk be az 1.1. ábra jobb felső sarkából kiindulva. Tegyük fel például, hogy egy egyetemi nyilvántartó adatbázis-adminisztrátora vagy röviden *DBA*-ja (a DBA az angol Database Administratorból származó rövidítés) elhatározza, hogy létre kell hozni egy olyan táblát, amelynek oszlopában a hallgatót, a hallgató által felvett kurzusokat és a kurzuson szerzett érdemjegyet tároljuk. A DBA azt is eldöntheti, hogy a lehetséges érdemjegyek köre A, B, C, D és F. Ezek a szerkezetre és megszorításra vonatkozó információk minden adatbázis sémájának részét képezik. Azt láthatjuk az 1.1. ábrán, ahogyan a DBA elküldi az utasításokat a rendszernek. A DBA-nak speciális jogosultságokkal kell rendelkeznie ahhoz, hogy a sémamódosító utasításokat végrehajthassa, hiszen ezek alapvető hatással lehetnek az adatbázisra. Ezeket a sémamódosító utasításokat DDL-utasításoknak nevezzük (a rövidítés az angol Data Definition Language-re utal, amely adatdefiníciós nyelvet jelent). Egy DDL-feldolgozó elemzi őket, majd továbbítja a végrehajtómotorhoz, ahonnan az index-/fájl-/rekordkezelőhöz kerülnek, majd ezután megtörténik a *metaadatok* módosítása, amely információk az adatbázis sémájának leírására szolgálnak.

1.2.2. A lekérdezések végrehajtásának áttekintése

Az ABKR-nek küldött utasítások többsége az 1.1. ábra bal oldalán látható útvonal mentén hajtódik végre. Egy felhasználó vagy egy alkalmazói program művelet kezdeményezésére az adatmanipulációs nyelvet (vagy az angol Data Manipulation Language alapján a DML-t) használják. Ez az utasítás nem érinti az adatbázis sémáját, de érintheti a tartalmát (amennyiben módosító utasításról van szó), vagy adatot kér az adatbázisból (lekérdező utasítás esetén). A DML-utasításokat két különálló alrendszer kezeli, amelyeket a következőkben mutatunk be.

Lekérdezések megválaszolása

A lekérdezéseket a *lekérdezésfordító* elemzi és optimalizálja. Az eredményül kapott *végrehajtási terv*, ami az ABKR által végrehajtandó elemi műveletek sorozata, a végrehajtómotorhoz kerül. A végrehajtómotor olyan utasítások sorozatát adja ki az erőforrás-kezelő felé, amelyek kis adatmennyiségekre, általában a relációk néhány sorára vonatkoznak. Az erőforrás-kezelő ismeri a relációkat tároló *adatfájlok*at, a fájlokban belüli rekordok méretét és formátumát, valamint az *indexfájlok*at, amelyek segítségével az adatfájlok elemei gyorsan megtalálhatók.

Az adatkéréseket a *pufferkezelő* kapja meg. A pufferkezelő szerepe az, hogy a másodlagos tárolón (ez általában a lemez) tartósan tárolt adatokat a memóriabeli pufferekbe megfelelő adagokban beolvassa. Általában lapnak vagy „lemezblokknak” nevezük a lemez és a memóriapufferek között mozgatott adatok mennyiségi egységét.

A pufferkezelő a tárkezelővel kommunikál az adatok lemezről való beolvasásának céljából. A tárkezelő kiadhat operációs rendszer szintű utasításokat is, de gyakoribb megoldás, hogy az ABKR közvetlenül ad utasításokat a lemezvezérőlönök.

Tranzakciók feldolgozása

A lekérdezések és más DML-műveletek tranzakciókba vannak csoportosítva. Ezek olyan egységeket jelölnek, amelyeket atomosan és egymástól elkülönítve kell végrehajtani. Bármely lekérdezés vagy módosító művelet önmagában alkot egy tranzakciót. Ezenkívül a tranzakciók végrehajtásának *tartósnak* kell lennie, ami annyit jelent, hogy egy befejezett tranzakció eredményének meg kell őriződnie még abban az esetben is, ha a tranzakció befejeződése után közvetlenül valamiféle rendszerhiba fordul elő. A tranzakciófeldolgozót két nagyobb részre osztjuk, amelyek a következők:

1. *konkurenciakezelő* vagy más néven *ütemező*, amely a tranzakciók atomoságáért és elkülönítéséért felelős, valamint
2. *naplázás- és helyreállítás-kezelő*, amely a tranzakciók tartósságáért felelős.

1.2.3. A tárkezelő és a pufferkezelő

Az adatbázis adatai általában másodlagos tárolón találhatók, ami a mai számítógépes rendszerek esetében többnyire mágneslemezt jelent. Ahhoz azonban, hogy az adatokon bármilyen műveletet végezhessünk, azoknak a memóriában kell lenniük. A *tárkezelő* feladata az adatok lemezen való elhelyezkedésének, valamint a lemez és memória közötti adatmozgatásoknak a felügyelete.

Egy egyszerű adatbázisrendszerben a tárkezelő lehetne az operációs rendszer fájlkezelő része is, de a hatékonyúság érdekében az adatbázis-kezelők általában közvetlenül felügyelik az adatok lemezen való tárolását, legalábbis bizonyos körfülmények között. A *tárkezelő* tartja nyilván a fájlok lemezen való elhelyezkedését, és a pufferkezelő igénye szerint ő állítja elő egy adott fájl blokkját vagy blokkjait.

A *pufferkezelő* feladata a memóriának pufferekre való felosztása. A pufferek lapmátrix tartományok, amelyekbe a lemezblokkok beolvashatók. Így az ABKR-nek minden komponense, amelynek lemezen lévő információra van szüksége, kapcsolatba kerül a pufferekkel és a pufferkezelővel, vagy közvetlenül, vagy a végrehajtómotoron keresztül. A különböző komponenseknek a következő típusú információkra lehet szükségük:

1. *Adat*: Magának az adatbázisnak a tartalma.
2. *Metaadat*: Az adatbázis sémája, ami leírja az adatbázis szerkezetét, valamint a rajta definiált megsorításokat.
3. *Napló rekordok*: Az adatbázison végzett módosításokról szóló információkat tartalmazzák. Az adatbázis tartósságának megőrzését támogatják ezekkel.
4. *Statisztikák*: Az ABKR által összegyűjtött és tárolt információk az adatok tulajdonságairól, például a relációk méretéről és a bennük előforduló értékekről, valamint az adatbázis további komponenseiről.
5. *Indexek*: Olyan adatszerkezetek, amelyek a hatékony adatelérést támogatják.

1.2.4. Tranzakciók feldolgozása

Teljesen általános dolog, hogy egy vagy több adatbázis-műveletet *tranzakcióba* csoportosítunk, ami olyan egységet alkot, amit a rendszernek atomosan kell végrehajtania, valamint az egyes tranzakciókat látszólag egymástól függetlenül kell futtatnia. Ezenkívül az ABKR garantálja még a tartósságot, vagyis azt, hogy egy befejezett tranzakció eredménye nem fog elveszni. A *tranzakciókezelő* tehát fogadja az alkalmazástól a tranzakció utasításait, ami megmondja, hogy egy tranzakció mikor kezdődik és mikor ér véget, továbbá információkat kap az alkalmazás egyéb elvárásairól (előfordulhat például, hogy nem igényli az atomosságot). A tranzakciókezelő a következő feladatokat végzi:

1. *Naplózás*: A tartósság biztosításának érdekében minden adatbázis-módosítás külön naplózódik a lemezen. A *naplókezelő* egy olyan eljárást használ, amelynek segítségével bármikor következzék is be egy rendszerhiba, a *helyreállítás-kezelő* képes lesz a változások naplóbejegyzéseinek vizsgálatával az adatbázist egy konzisztens állapotba visszaállítani. A naplókezelő először a bejegyzéseket pufferbe írja, majd a pufferkezelővel egyezteti, hogy a puffer a megfelelő időpontokban lemezre legyenek írva (ahol túlélik a rendszerösszeomlást).
2. *Konkurenciakezelés*: A tranzakcióknak úgy kell tűnniük, mintha elkülönítve futnának. A legtöbb rendszerben azonban a valóságban sok tranzakció hajtódiagram végre egy időben. Ezért az ütemezőnek (konkurenciakezelőnek) kell azt biztosítania, hogy a tranzakciók egyedi műveletei olyan sorrendben hajtódjanak végre, hogy azok összhatása ugyanaz legyen, mintha a tranzakciók valóban egymás után futottak volna le. Az ütemezők ezt a feladatot jellemzően úgy végezik el, hogy az adatbázis bizonyos részeire zárákat helyeznek el. A zárák megakadályozzák azt, hogy két tranzakció úgy férjen hozzá egy adatelemhez, ahogyan az nem megengedett. A zárák

A tranzakcióktól elvárt tulajdonságok^a

A helyesen megvalósított tranzakcióktól elvárjuk, hogy teljesítsék az alábbi követelményeket:

- *Atomosság*: Ez azt jelenti, hogy a tranzakciónak vagy az összes utasítása hajtódjon végre, vagy egyetlen utasítása se hajtódjon végre.
- *Elkülönítés*: Ez azt jelenti, hogy az egyes tranzakcióknak látszólag úgy kell végrehajtódniuk, mintha a lefutásuk ideje alatt egyetlen más tranzakció sem futna.
- *Tartósság*: Ez azt jelenti, hogy ha egy tranzakció befejeződött, akkor az általa végrehajtott utasítások eredménye már semmilyen körülmenyek között nem veszhet el.

Szokás még egy tulajdonságot említeni, nevezetesen a „*konziszenciát*”. minden adatbázisban vannak konziszenciára vonatkozó megszorítások, amelyek az adatelemek közötti kapcsolatokra vonatkozó elvárásainkat fejezik ki. Ilyen lehet például, hogy a számlaegyenlegek ne lehessenek negatívak. A tranzakcióktól elvárjuk, hogy ezeket a konziszenciafeltételeket megőrizzék.

^a Az angol szakirodalom ezt ACID-tulajdonságoknak nevezi (A = atomicity, C = consistency, I = isolation, D = durability). (A szerkesztő megjegyzése.)

általában a memoriában tárolódnak egy úgynevezett zárttáblában, ahogyan azt az 1.1. ábrán is láthatjuk. Az ütemező oly módon befolyásolja a lekérdezések és egyéb adatbázis-műveletek végrehajtását, hogy nem engedi a végrehajtómotor számára a hozzáférést az adatbázis zárt részeihez.

3. *Holtpontfeloldás*: Miközben a tranzakciók az ütemező által biztosított zárák segítségével versengenek az erőforrásokért, előállhat egy olyan helyzet, hogy egyik tranzakció sem tud továbblépni, mert olyan erőforrásra van szüksége, amit valamelyik másik tranzakció zárol. Ilyenkor a tranzakciókezelő feladata, hogy közelbelépjen és megszakítson egy vagy több tranzakciót annak érdekében, hogy a többiek tovább tudjanak dolgozni.

1.2.5. A lekérdezésfeldolgozó

Az ABKR-nek az a része, amelyik a legnagyobb hatással van a felhasználó által érzékelhető hatékonyságra, a lekérdezésfeldolgozó. Az 1.1. ábrán a lekérdezés-feldolgozó két összetevőjét ábrázoltuk:

1. A lekérdezésfordító fordítja le a lekérdezést egy belső formára, amit végrehajtási tervnek nevezünk. Ez a terv az adatokon végrehajtandó műveletek sorozata. Általában ezek a műveletek a „relációs algebra” műveleteinek megvalósításai, amely műveletekről majd a 2.4. szakaszban lesz szó. A lekérdezésfordító három főbb részből áll:

- egy lekérdezéselemzőből, ami a lekérdezés szövegéből egy fa struktúrát épít fel;
- egy lekérdezés-előfeldolgozóból, ami szemantikai ellenőrzéseket végez a lekérdezésen (például ellenőrzi, hogy a lekérdezésben előforduló relációk valóban léteznek-e), majd átalakításokat végez az elemzőfán úgy, hogy abból egy algebrai műveleteket tartalmazó kezdeti vérehajtási terv legyen;
- egy lekérdezés-optimalizálóból, ami a kezdeti vérehajtási tervet átalakítja a lehető legjobb tervvé, vagyis műveletsorozattá, a tényleges adatok figyelembevételével.

A lekérdezésfordító a várhatóan leggyorsabb műveletsorozat megtalálásához az adatokra vonatkozó metaadatokat és statisztikákat használja. Például egy *index* létezése az egyik tervet sokkal gyorsabbá teheti egy másik tervnél. Az index egy speciális adatszerkezet, ami lehetővé teszi az adatok gyors elérését, feltéve, hogy egyes adatelemeket ismerünk.

2. A vérehajtómotor felelős a kiválasztott vérehajtási terv egyes lépéseinak vérehajtásáért. A vérehajtómotor az ABKR legtöbb komponensével valamilyen kapcsolatba lép, vagy közvetlenül, vagy a puffereken keresztül. Ahhoz, hogy bármit tehessen az adatokkal, azokat az adatbázisból a pufferkbe kell bekérnie. Egyeztetni kell az ütemezővel, nehogy zárolt adatokhoz férjen hozzá, továbbá a naplókezelővel, hogy biztos lehessen benne, hogy minden adatbázisbeli változás megfelelően naplózva lett.

1.3. Adatbázisrendszerrel kapcsolatos ismeretek áttekintése

Az adatbázisok tanulmányozását öt fő részre osztottuk.³ Ez az alfejezet áttekintést ad arról, hogy mikre lehet számítani az egyes részeknél:

1. rész. Relációs adatbázisok modellezése

A relációs modell elhanyagolhatatlan jelentőségű az adatbázisrendszerek vizsgálatánál. Az alapvető fogalmak vizsgálata után elmélyedünk a relációs adatbázisok elméletében. Ez a tanulmány magában foglalja a *funkcionális függő-*

³ A 4. és az 5. rész tárgyalása a következő könyvben található: H. Garcia-Molina, J. D. Ullman, J. Widom: *Adatbázisrendszerek megvalósítása*, Panem, Budapest, 2001, 2008.

még a kényelmes használat is, vagy másnéppen szólva az adatot feldolgozó programozók termelékenysége. Meglepő módon egy modell, nevezetesen a relációs modell teljesít minden célkitűzést, azaz:

1. Egy egyszerű, korlátozott megközelítést ad az adatok strukturálására, ami ésszerűen sokoldalú. Azaz bármilyen modellezhető vele.
2. Egy korlátozott, ám hasznos, az adatokon végrehajtható műveletgyűjteményt tesz lehetővé.

Ez a két korlátozás együtt ad egy környezetet, és lehetővé teszi nyelvek megvalósítását, mint például az SQL-t, ami biztosítja a programozóknak, hogy kérdéseiket nagyon magas szinten megfogalmazhassák. Egy néhány soros SQL-utasítás elérheti ugyanazt az eredményt, mint egy több ezer soros C-program, vagy mint egy több száz soros kód, amelyet a korábbi modellekben (mint a hierarchikus vagy a hálós) írtak az adatokhoz történő hozzáférésre. Mivel a használható műveletek halmaza erősen korlátozott, a rövid SQL-programok még optimalizálhatóak is azért, hogy gyorsan vagy más választható nyelvben írt kódoknál gyorsabban futhassanak.

2.2. A relációs modell alapjai

A relációs modellben az adatok egyszerűen reprezentálhatók: kétdimenziós táblákban, ún. *relációkban*. A 2.1. ábrán egy példát mutattunk relációra, amelyet most a 2.3. ábrán megismertünk. A reláció neve **Filmek**, és a már többször hivatkozott, állandó példánk **Filmek** egyedhalmazának elemeiről tárolunk benne információt. minden sor egy filmnek felel meg, az oszlopok pedig egy-egy film tulajdonságának. Ebben az alfejezetben bevezetjük a legfontosabb jelöléseket és fogalmakat a relációkhöz, és ezeket a **Filmek** relációt fogjuk szemléltetni.

<i>filmcím</i>	<i>év</i>	<i>hossz</i>	<i>műfaj</i>
Elfújta a szél	1939	231	dráma
Csillagok háborúja	1977	124	sci-fi
Wayne világa	1992	95	vígjáték

2.3. ábra. A **Filmek** relációja

2.2.1. Attribútumok

A reláció oszlopait az *attribútumok* látják el névvel. A 2.3. ábrán az attribútumok a következők: *filmcím*, *év*, *hossz* és *műfaj*. Az attribútumok az oszlopok fejrészében láthatók. Általában az attribútumok megadják az abban az oszlopból szereplő adatok jelentését. Például a *hossz* attribútummal ellátott oszlop minden egyes filmhez megadja annak hosszát percekben.

Relációk és attribútumok szokásos jelölései

Általában követendő példa, hogy a relációk neveit nagybetűvel, az attribútumneveket pedig kisbetűvel kezdjük. A könyv hátralévő részében azonban a relációkról absztrakt értelemben tárgyalunk, ahol nem számítanak az attribútumnevek. Ezen esetekben mind a relációkra, mind az attribútumakra nagybetűket használunk, mint például az $R(A, B, C)$, ami egy általános 3-attribútumos relációt jelöl.

2.2.2. Sémák

A reláció nevét és a relációattribútumok halmazát együtt nevezzük *relációsémának*. A relációsémát a reláció nevével és az attribútumainak zárójelek közötti felsorolásával adjuk meg. Tehát a 2.3. ábra *Filmek* relációsémája a következő:

Filmek(filmcím, év, hossz, műfaj)

A relációséma attribútumai halmazt alkotnak és nem listát. Mégis, amikor általában a relációkról van szó, meg kell határoznunk az attribútumok valamelyen szabványos sorrendjét. Tehát amikor attribútumok listájával vezetjük be a relációsémát, ugyanezt a szabványos sorrendet vesszük akkor is, amikor megjelenítjük a relációt vagy annak a sorait.

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

2.2.3. Sorok

A reláció azon sorait, amelyek különböznek az attribútumokból álló fejléc sorától, *sorknak* (tuple) nevezzük. A reláció minden egyes attribútumához tartozik a sorban egy *komponens*. Például a 2.3. ábra három sora közül az első sor négy komponense: *Elfújta a szél, 1939, 231, dráma*, amelyek ebben a sorrendben a *filmcím, év, hossz* és *műfaj* attribútumokhoz tartoznak. Amikor külön írjuk le a sorokat, nem pedig valamely reláció részeként, akkor vesszőkkel választjuk el a komponenseket, és a sort zárójelek közé tesszük. Például

(*Elfújta a szél, 1939, 231, dráma*)

a 2.3. ábra relációjának első sora. Megjegyezzük, hogy amikor egy sor magában van, az attribútumokat nem láthatjuk, így meg kell adnunk valamilyen hivatkozást, hogy melyik relációhoz tartozik az adott sor, és minden ugyanabban a sorrendben írjuk fel a komponenseket, ahogyan az attribútumokat felsoroltuk a relációsémában.

2.2.4. Értéktartományok

A relációs modellben követelmény, hogy minden sor minden komponense atomi, azaz elemi típusú legyen, például egész vagy karaktersorozat. Nem megengedett az értékekhez a rekordszerkezet, halmaz, lista, tömb vagy bármely más olyan típus, amelynek az értékei kisebb komponensekre felbonthatók.

Feltesszük továbbá, hogy a reláció minden attribútumához tartozik egy **értéktartomány**, azaz egy elemi típus. A reláció bármely sorában szereplő komponensek értékének a megfelelő oszlop értéktartományából kell származnia. Például a 2.3. ábrán szereplő *Filmek* reláció sorainak az első komponense karaktersorozat, a második és harmadik komponense egész szám, a negyedik komponens értéke pedig karaktersorozat.

Olyan megadás is lehetséges, amely tartalmazza az értéktartományt (vagy típust) a relációsémában szereplő összes attribútumra. Ezt megtehetjük az attribútumot követő kettőspont és típus hozzáírásával. Azaz a *Filmek* reláció sémáját reprezentálhatjuk például az alábbi módon:

```
 Filmek(filmcím:string, év:integer, hossz:integer, műfaj:string)
```

2.2.5. Relációk egyenértékű ábrázolási módjai

Egy reláció sorokból álló halmaz, nem pedig lista. Emiatt lényegtelen, hogy minden sorrendben jelenítjük meg azokat. Például, ha a 2.3. ábra három sorát a hat lehetséges sorrend bármelyikében is írjuk fel, a reláció „ugyanaz”, mint ami a 2.3. ábrán található.

Továbbá a reláció attribútumainak a sorrendjét is felcserélhetjük, a relációt ez nem változtatja meg. Mégis, amikor a relációsémát átrendezzük, ügyelnünk kell az oszlopok fejrészében szereplő attribútumokra. Azaz, amikor megváltoztatjuk az attribútumok sorrendjét, akkor ezzel megváltoztatjuk az oszlopok sorrendjét is. Ha az oszlopokat felcseréljük, akkor vele együtt a sorok komponenseinek a sorrendje is megváltozik. Az eredmény, hogy mindegyik sorban a komponensek ugyanúgy permutálódnak, mint ahogyan az attribútumok vannak permutálva.

Például a 2.4. ábrán látható reláció a 2.3. ábrán szereplő relációból a sorok és oszlopok permutációjával kapható egyik lehetséges reláció. A két relációt „azonosnak” tekintjük. Pontosabban ez a két tábla ugyanannak a relációnak két különböző megjelenítése.

<i>év</i>	<i>műfaj</i>	<i>filmcím</i>	<i>hossz</i>
1977	sci-fi	Csillagok háborúja	124
1992	vígjáték	Wayne világa	95
1939	dráma	Elfújta a szél	231

2.4. ábra. A *Filmek* reláció másik megjelenítése

2.2.6. Relációk előfordulásai

A filmeket tartalmazó reláció nem állandó, sőt a relációk többször is változnak az idők során. A változások egy része várhatóan a reláció soraira fog vonatkozni, mint például új sorok beszúrása, azaz az új filmek adatbázisba vétele, a létező sorok megváltoztatása, ha újabb vagy pontosabb információt kapunk a filmekről, és esetleg sorok törlése, ha filmeket valamilyen ok miatt eltávolítunk az adatbázisból.

A relációséma megváltoztatása kevésbé általános. Bár előfordulhatnak olyan helyzetek, amikor attribútumokat szeretnénk felvenni vagy törölni. A forgalomban levő adatbázisrendszerek lehetővé teszik, hogy a sémát megváltoztassuk, ám ez igen költséges, ugyanis előfordulhat, hogy milliónyi sor mindegyikét át kell frni, hogy hozzávegyünk vagy töröljünk komponenseket. Ha új attribútummal bővítettünk, akkor az is nehézséget okoz, vagy egyáltalán nem lehetséges, hogy a sorok új komponenseihez megfelelő értékeket találjunk.

Az adott reláció sorainak halmazát *reláció-előfordulásnak* nevezzük. Például a 2.3. ábrán található három sor a Filmek reláció egy előfordulása. Feltehetően a Filmek reláció változott már a múltban és változni fog a jövőben. Például 1990-ben a Filmek nem tartalmazta a Wayne világa sort. A hagyományos adatbázisrendszerek bármely relácionak csak egyetlen változatát kezelik: csak azokat a sorokat, amelyek „most” vannak a relációban. Ezt a reláció-előfordulást *aktuális előfordulásnak*¹ nevezzük.

2.2.7. A reláció kulcsai

A relációs modell jó néhány – a relációra vontkozó – megszorítás megfogalmazását lehetővé teszi számunkra az adatbázissémán belül. A megszorítások tárgyalását a 7. fejezetre hagyjuk. Van viszont egy olyan alapvető megszorítás, amelyet már itt be kell vezetnünk: a *kulcs* megszorítás. Az attribútumok egy halmaza egy kulcsot alkot egy relációra nézve, ha a reláció előfordulásaiban nincs két olyan sor, amelyek a kulcs összes attribútumának értékein megegyeznének.

2.1. példa. Megadhatjuk a Filmek relációt egy kulccsal is, amelyet a *filmcím* és *év* attribútumok alkotnak. Azaz nem feltételezzük azt, hogy valaha is lehetne két olyan film, amelyeknek a címe is és a gyártási éve is megegyezne. Figyeljük meg, hogy a „felújított” filmek létezése miatt a *filmcím* önmagában nem alkothat kulcsot. Példaként három King Kong című film is van, de minden más évben készült. Az is látszik, hogy az év sem lehet kulcs, hiszen több film is készül egy adott évben. □

¹ Azokat az adatbázisokat, amelyek az adat időben korábbi verzióit is nyilvántartják, *temporális adatbázisoknak* nevezzük.

Az attribútum vagy attribútumok aláhúzásával jelöljük, hogy a szóban forgó attribútum kulesattribútum. Ezért például a *Filmek* reláció sémáját a következőképpen is írhatnánk:

Filmek(filmcím, év, hossz, műfaj)

Emlékezzünk, hogy az az állítás, hogy az attribútumok egy halmaza alkot kulcsot a relációra nézve, egyúttal egy, a reláció összes előfordulására vonatkozó állítás is. Például tekintsük a 2.3. ábra apró relációját. Ekkor állíthatnánk azt is, hogy a **műfaj** önmagában kulcsot fog alkotni, hiszen nem látunk két olyan sort, amelyek **műfaj** komponensükön megegyeznének. Viszont az is könnyen el-képzelhető, hogy ha a reláció előfordulásai több filmet tartalmaznának, akkor sok dráma, sok vígjáték és sok más egyéb kategóriájú film is lenne. Azaz több olyan sor is lehetne, amelyek a **műfaj** komponensükön megegyeznek. Következésképpen a *Filmek* relációnak a **műfaj** nem lehet kulcsa.

Miközben mi megbizonyosodtunk arról, hogy a **filmcím** és az **év** együtt lehetnek a *Filmek* kulcsa, addig a legtöbb valós adatbázisban mesterségesen előállított kulcsokat használnak, mivel kételkednek annak a biztonságos voltában, hogy nem teljes kontroll alatt lévő attribútumértékekkel feltevéseket tegyenek. Például a cégek általában minden egyes alkalmazottukhoz egy olyan egyedi alkalmazottazonositót rendelnek, amelyeket nagy körültekintéssel egyedi számnak választanak meg. Ezeknek az azonosítóknak célja, hogy a cég adatbázisán belül minden egyes dolgozót meg tudjanak különböztetni a többitől, még akkor is, ha van két azonos nevű alkalmazott. Azaz az alkalmazottazonosító attribútum lehet kulcsa az alkalmazottak relációjának.

Az amerikai cégeknél minden egyes alkalmazottnak van társadalombiztosítási száma. Ha az adatbázisukban van egy társadalombiztosítási számra vonatkozó attribútum, akkor ez is lehet az alkalmazottak relációjának a kulcsa. Megjegyezzük, hogy az nem jelent problémát, ha több kulcsválasztás is létezik, mint ahogyan az alkalmazottakra nézve is mind az alkalmazottazonosító, mind a társadalombiztosítási szám kulcs volt.

A kulcsként használható attribútum létrehozásának az elve széles körben elterjedt. Az alkalmazottazonosítók mellett az egyetemek is hallgatói azonosítóval különböztetik meg a diákjait. Vannak még jogosítványszámok és rendszámok a sofőrök, illetve az autók közötti különbségtételhez. Az olvasó minden kétséget kizárában magától is több olyan attribútumokra vonatkozó példát tud felsorolni, amelyeket azért hoztak létre, hogy kulcsként szolgáljanak.

2.2.8. Példa egy adatbázissémára

Az alfejezetet egy teljes adatbázissémát tartalmazó példa megadásával zárjuk. A szóban forgó példa témája a *Filmek* lesznek. Az egész a korábban már szerepettelte *Filmek* relációra fog épülni. Az adatbázis sémáját a 2.5. ábrán láthatjuk. A séma alapjainak megértéséhez tekintsük át részleteiben az egyes részeit:

```

Filmek(
    filmcím:string,
    év:integer,
    hossz:integer,
    műfaj:string,
    stúdióNév:string,
    producerAzon:integer
)
FilmSzínész(
    név:string,
    cím:string,
    nem:char,
    születésiDátum:date
)
SzerepelBenne(
    filmCím:string,
    filmÉv:integer,
    színészNév:string
)
GyártásIrányító(
    név:string,
    cím:string,
    azonosító:integer,
    nettóBevétel:integer
)
Stúdió(
    név:string,
    cím:string,
    elnökAzon:integer
)

```

2.5. ábra. Példa egy adatbázissémára a filmek tárolásához

Filmek

Ez a reláció a korábban tárgyalt példareláció kiterjesztése. Emlékezzünk rá, hogy a filmcím és az év együttesen kulcsot alkotnak a relációra nézve. Két új attribútum keletkezett. A stúdióNév szolgál a tulajdonos gyártóazonosítójaként. A producerAzon pedig egy olyan egész szám lesz, amely a film producerét úgy fogja azonosítani, mint ahogyan a GyártásIrányító bemutatásánál majd megbeszéljük.

FilmSzínész

Ez a reláció határozza meg a színészekre vonatkozó adatokat. A kulcs a név attribútum lesz, amely a színész nevét tartalmazza. Általában viszont a név

nem feltétlenül egyedi, ezért nem is biztos, hogy kulcs lesz. A filmszínészek viszont elternek ettől abban az értelemben, hogy nem használnak olyan nevet, amelyet már valamelyik másik színész használt volna. Ezért is használjuk azt a feltevést, hogy a filmszínészek nevei egyediek. Egy általánosabban használható megközelítés egy olyan azonosító sorszám használata, mint a társadalombiztosítási számé, amely minden személyhez egy egyedi sorozatszámot rendel. Így ez használható kulcsattribútumként. Ezt a megközelítést alkalmazzuk a filmgyártókra, mint ahogy azt látni fogjuk. A másik érdekesség a **FilmSzínész** relációval kapcsolatban, hogy két új adattípus is tartalmaz: a nemet, amely lehet egy F vagy egy N karakter, illetve a születési dátumot, amely „dátum” típusú, speciális formátumú karaktert sorozatot takar.

SzerepelBenne

Ez a reláció szolgál a film és a színészek kapcsolatának leírására azáltal, hogy összekapcsolja a színészt azokkal a filmekkel, melyekben a színész szerepel. Figyeljük meg, hogy a filmeket a **Filmek** tábla kulcsával (**filmcím** és **év**) reprezentáljuk annak ellenére, hogy más attribútumneveket választottunk hozzájuk, nevezetesen **filmCím**, **filmÉv**. A színészeket ehhez hasonlóan a **FilmSzínész színészNév** kulcsattribútumával ábrázoljuk. Végül figyeljük meg, hogy itt a három attribútum együttesen alkot kulcsot. Ez pedig pontosan megfelel annak az elkövetésnek, hogy a **SzerepelBenne** relációnak lehet két olyan különböző sor, amelyek valamely két attribútumukon megegyeznek. Például, ha egy adott színész egy adott évben két filmben is szerepelt, akkor lesz két olyan sor, amelyek a **filmÉv**, illetve **színészNév** komponensükben megegyeznek, de a **filmCím** komponenseiken eltérnek.

GyártásIrányító

Ez a reláció a filmgyártók adatait tartalmazza: a gyártó nevét, címét és a gyártó nettó bevételét. A filmgyártókat, beleértve a producereket (ahogy a **Filmek** relációban fel van tüntetve) és a stúdió vezetőit is, akik csak a következő **Stúdió** relációban szerepelnek, egy „azonosító számmal” láttuk el, amely kulcsként fog szolgálni. Ezek az azonosítók egész számok, és minden gyártónak különbözők.

Stúdió

Ez a reláció a filmstúdiókról szól. Feltételezzük, hogy két stúdiónak nem lehet azonos a neve, így a **név** kulcs lesz a relációra nézve. A másik két mező: a stúdió címe, illetve a stúdió vezetőjéhez rendelt azonosító szám. Feltételezzük, hogy a stúdió vezetője egy valódi gyártásirányítót takar, és így szerepel a **GyártásIrányító** relációban is.

2.2.9. Feladatok

2.2.1. feladat. Legyen a 2.6. ábrán található két reláció-előfordulás egy banki adatbázis része. Mutassuk meg a következőket:

- a) Mindegyik reláció attribútumait.
- b) Mindegyik reláció sorait.
- c) Mindegyik reláció egy sorának a komponenseit.
- d) Mindegyik reláció relációsémáját.
- e) Az adatbázissémát.
- f) Mindegyik attribútumhoz egy megfelelő értéktartományt.
- g) Mindegyik relációhoz egy másik, vele ekvivalens megjelenítését.

számlaSzám	típus	egyenleg
12345	Betétszámla	12000
23456	Folyószámla	1000
34567	Betétszámla	25

A Számlák reláció

vezetéknév	keresztnév	azonosítószám	számla
Balogh	Róbert	901-222	12345
Kovács	Léna	805-333	12345
Kovács	Léna	805-333	23456

A Vevők reláció

2.6. ábra. Egy banki adatbázis két relációja

2.2.2. feladat. Több példát is javasoltunk a 2.2.7. alfejezetben az olyan attribútumok kiválasztására, amelyek a relációk kulcsaként szolgálhatnak. Adjunk néhány további példát!

!! 2.2.3. feladat. Hány különböző módon reprezentálható egy reláció-előfordulás (az attribútumok és sorok sorrendjét tekintve), ha az előfordulás az alábbiakkal rendelkezik:

- a) Hárrom attribútum és három sor, mint a 2.6. ábra Számlák relációjában?
- b) Négy attribútum és öt sor?
- c) n attribútum és m sor?

2.3. Relációsémák definiálása SQL-ben

Az SQL a legfontosabb nyelv, amely lehetővé teszi relációs adatbázisok leírását illetve módosítását is. A legutóbbi SQL-szabvány az SQL-99. A forgalomban lévő adatbázis-kezelő rendszerek túlnyomó része a szabványhoz hasonló – de nem teljesen azonos – dolgokat valósít meg. Az SQL-nek két szempontnak megfelelő része van:

1. az *adatdefinícos* résznyelv az adatbázissémák megadásához, illetve
2. az *adatmanipulációs* résznyelv az adatbázis *lekérdezéséhez* (vagy az adatbázisnak adható kérdések megfogalmazásához), illetve az adatbázis módosításához.

A legtöbb programozási nyelvben megvan egy ilyen típusú megkülönböztetés. Például a *C*-nek és a *Javának* is van deklarációs része, illetve futtatható kód része. Ez pedig pontosan megfelel az adatdefiníciós, illetve adatmanipulációs kettősnek.

Ebben a részben az adatdefiníciós rész tárgyalását kezdjük el, de részletesebben a 7. fejezetben térünk vissza rá, főként az adatokon tehető megszorítások témaörénél. Az adatmanipulációs részzel a 6. fejezet részletesen foglalkozik.

2.3.1. SQL-relációk

Az SQL három típusú relációt ismer:

1. Tárolt relációk vagy más néven *táblák*. Általában ilyen relációkkal foglalkozunk. Ezek benne vannak az adatbázisban, soraik változtatásával megváltozthatatók és soraik is lekérdezhetők.
2. A *Nézetek* számításokból kapott relációk. Ezeket nem tároljuk, de részben vagy teljes egészben szerkeszthetjük őket, ha ez szükséges. Ezek lesznek a 8.1. alfejezet fő témai.
3. Ideiglenes táblák, amelyeket az SQL nyelvi feldolgozója készít, amikor valamilyen lekérdezéseket és adatmódosításokat végez el. Ezeket a relációkat eldobja és nem tárolja sehol az SQL feldolgozója.

Ebben a részben a táblák megadásának módjait sajátítjuk el. Itt nem foglalunk a nézetek meghatározásával és definiálásával sem. Az ideiglenes táblák pedig sohasem deklaráltak. Az SQL CREATE TABLE utasításával határozható meg egy tárolt reláció sémája. Megadja a tábla nevét, az attribútumait, illetve az attribútumok típusát. Emellett kulcs, illetve kulcsok megadását is lehetővé teszi. A CREATE TABLE utasításnak az eddig említetteken túl több lehetősége is van. Ilyenek például a különféle megszorítások, amelyeket szintén meghatározhatunk vele. Illetve ilyenek az *indexek* (vagy olyan adatstuktúrák, amelyek a táblán végezhető műveletek felgyorsítására szolgálnak) megadása is, de most halasszuk ezeket egy alkalmasabb időpontra.

2.3.2. 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 n hosszúságú karaktersort jelöl. A VARCHAR(n) egy legfeljebb n hosszúságú karaktersort jelöl. A különbözők közöttük implementációfüggő, ugyanis a CHAR általában azt jelenti, hogy a rövidebb karaktersorozatokat kiegészítjük n hosszúvá, míg a VARCHAR esetén használnak egy lezáró jelet vagy hosszértéket is. Az SQL ésszerű típuskényszerítést is megenged a két típus értékei között. Normál körülmények között a stringet a végére fűzött üres karakterekkel egészítjük ki, amennyiben egy annál hosszabb, rögzített méretű karaktersorozatnak adjuk értékül. Tekintsük a 'foo' karaktersorozatot, ha ezt egy CHAR(5) típusú attribútumnak adjuk értékül, akkor ez a 'foo' értéket jelenti (azaz két szóközzel kiegészítve az eredetit).²
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. A BOOLEAN egy logikai értékű attribútumot jelöl. Az ilyen attribútumok lehetséges értékei: TRUE, FALSE vagy – ami még George Boole-t is meglepné – UNKNOWN.
4. 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).
5. A lebegőpontos értékeket többféleképpen is tárolhatjuk. Használhatjuk a FLOAT és a REAL típusokat (amelyek megegyeznek). Nagyobb pontossággal

² Megjegyezzük, hogy sok más egyéb programozási nyelvhez hasonlóan SQL-ben is a karakterláncokat idézőjelek, illetve macskakörök közé írjuk.

Dátumok és időpontok SQL-ben

A különböző SQL-megvalósításokban eltérő módokon ábrázolhatják a dátumot, illetve az időt, de a most következőkben mi az SQL szerinti reprezentálását vizsgáljuk majd. 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 kötőjel, majd két számjegy jelképezi a hónapot. Végül következik egy újabb kötőjel és két számjegy, mely a napot jelképezi. Megjegyezzük, hogy mind a hónapok, mind a napok esetében az egy számjegyből álló számokat kiegészítjük egy 0-val.

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 diákok elhagyta az osztálytermet egy olyan óra után, mely délután 3-kor ért véget, tehát két és fél másodperccel három után.

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). A NUMERIC is a DECIMAL egyik szinonimája, habár lehetnek különbségek a megvalósítástól függően.

6. Dátumokat és időket a DATE és TIME típusok jelképeznek. Lásd bővebben a „Dátumok és időpontok SQL-ben” keretes résznél. 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.

2.3.3. Egyszerű táblalétrehozások

A legegyszerűbb módja egy tábla létrehozásának a CREATE TABLE kulcsszavakból, a relációnevűből, az attribútumoknak és típusainak zárójelek közé tett listájából áll.

2.2. példa. A 2.5. ábrán látható Filmek relációt a 2.7. ábrán látható módon adhatjuk meg. A filmcím 100 hosszú karakterláncként deklarált, az év és hossz attribútumok egész számok, a műfaj pedig 10 hosszú karakterlánc.

```
CREATE TABLE Filmek (
    filmcím      CHAR(100),
    év           INT,
    hossz        INT,
    műfaj         CHAR(10),
    stúdióNév   CHAR(30),
    producerAzon INT
);
```

2.7. ábra. A *Filmek* tábla meghatározása SQL-ben

A cím hosszának választása tetszőleges, most éppen 100 karaktert engedtünk meg azért, mert nem akarjuk túlságosan korlátozni vagy éppen megcsönkítani a címek értékét. Feltettük azt is, hogy 10 karakter megfelel a műfaj tárolására. Ez megint egy véletlenszerű választás, amit meg is bánnhatunk, ha lesz egy hosszú nevű műfajunk. A stúdiónevek hasonló okokból 30 karakterből állnak majd. A producerek azonosító száma pedig egy egész szám lesz. □

2.3. példa. A 2.5. ábra *FilmSzínész* reláció sémáját a 2.8. ábrán látható SQL-utasítással hozhatjuk létre. Néhány új lehetőségét is látjuk az adattípusoknak. A tábla neve *FilmSzínész*, és négy attribútuma lesz. Az első két attribútum, a név és a cím típusa karaktersistor. A különbözőség abban áll, hogy a név attribútum egy 30 karakterből álló rögzített hosszúságú karaktersistor, amelyet szükség esetén üres karakterekkel töltünk ki, illetve lecsónkítjuk a végét, ha hosszabb. Ezzel szemben a címet egy változó hosszúságú karaktersistorba helyeztük, melynek maximális hossza 255 karakter.³ Nem biztos, hogy ez a két választás a legjobb, csak azért használtuk ezeket, hogy bemutassuk a kétféle karaktersistor használatát.

```
CREATE TABLE FilmSzínész (
    név          CHAR(30),
    cím          VARCHAR(255),
    nem          CHAR(1),
    születésiDátum DATE
);
```

2.8. ábra. A *FilmSzínész* reláció sémájának definiálása

A nem attribútum értékei lehetnek 'N' és 'F', tehát egyetlen karakter. Így nyugodtan használhatjuk az egy hosszúságú karaktersist sort típusként. Végül a születésiDátum attribútum értéke természetesen DATE. □

³ A 255 nem valamelyen bűvös szám, hogy minden címnek ilyen hosszúnak kell lennie. Egy bájton 0 és 255 közötti számokat lehet tárolni. Tehát egy változó hosszúságú karaktersist sort, amelynek maximális hossza 255, úgy lehet tárolni, hogy egy bájton tároljuk a karakterek számát, és magát a karaktersist sort annyi bájton, amennyi szükséges. A kereskedelmi rendszerek általában hosszabb változó hosszúságú karaktersistereket is megengednek.

2.3.4. Relációsémák módosítása

Most már tudjuk, hogyan lehet megadni egy relációt. De mi van akkor, ha meg akarjuk változtatni egy olyan tábla sémáját, amelyet már régóta használtunk, és amelynek túl sok sora is van emiatt? Eltávolíthatjuk az egész táblát az összes aktuális sorával együtt, vagy módosíthatjuk a sémát új attribútum hozzáadásával, illetve törlésével.

A következő SQL-utasítás törli a táblát:

```
DROP TABLE R;
```

Végrehajtása után az R reláció nem lesz benne a relációsémában, és így a sora-ihoz sem férhetünk hozzá a továbbiakban. 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, majd egy attribútumnév és annak adattípusa.
2. DROP és egy attribútumnév.

2.4. példa. Módosítsuk a FilmSzínész relációt, hozzáadva egy telefonszám attribútumot:

```
ALTER TABLE FilmSzínész ADD telefonszám CHAR(16);
```

Az utasítás eredményeképpen a FilmSzínész relácionak öt attribútuma lesz, az első négyet a 2.8. ábra bemutatta, az ötödik pedig a telefonszám attribútum, amely egy rögzített 16 bájt hosszúságú karaktorsor. A megváltozatott relációban minden sorban szerepelni fog a telefonszám komponens, de mivel még nincsenek telefonszámok megadva, ezért ezen komponensek értéke egy speciális, NULL nevű nullérték lesz. A 2.3.5. alfejezetben ismertetni fogjuk, hogyan lehet egy másik alapértelmezett értéket beállítani a NULL érték helyett.

Egy másik példaként töröljük ki a születésiDátum attribútumot:

```
ALTER TABLE FilmSzínész DROP születésiDátum;
```

Ennek eredményeként ez az attributum nem lesz benne a sémában a továbbiakban, és a FilmSzínész összes sorának a születésiDátum komponense is törölni fog. □

2.3.5. Alapértelmezés szerinti értékek

Amikor létrehozunk vagy módosítunk sorokat, néha egyes komponensek értékét nem ismerjük. A 2.4. példában a relációsémához hozzáadtunk egy új oszlopot, és a létező sorokban nincs meg az új oszlop értéke. Ott azt javasoltuk, hogy a komponens értéke NULL érték legyen. Vannak esetek, amikor *kezdeti* érték ilyen

választása nem megfelelő, hanem helyette valamilyen rögzített értéket használunk az oszlopra, ha nem ismert az értéke. Általában, amikor egy attribútumot és az adattípusát definiáljuk, hozzáadhatjuk a DEFAULT kulcsszót és egy megfelelő értéket. Az érték vagy NULL érték, vagy egy konstans. Egyéb olyan értékeket is használhatunk, amelyeket a rendszer rendelkezésünkre bocsát, mint például az aktuális idő.

2.5. példa. Tekintsük a 2.3. példát. Szeretnénk alapértelmezésként a '?' karaktert használni a nem attribútum esetében és a '0000-00-00' dátumot a születésiDátum esetében. Így a 2.8. ábra megfelelő sorait a következőre kell kicserélni:

```
nem CHAR(1) DEFAULT '?',
születésiDátum DATE DEFAULT DATE '0000-00-00',
```

Egy másik példaként az 2.4. példában a telefonszám új attribútum alapértelmezett értéke legyen a 'nem ismert' szöveg. Ekkor a következő ALTER TABLE utasítást kell használnunk:

```
ALTER TABLE Filmszínész
ADD telefonszám CHAR(16) DEFAULT 'nem ismert';
```

□

2.3.6. Kulcsok megadása

Két módszer van, amivel egy attribútumot vagy attribútumoknak egy halmazt kulcsként adhatjuk meg egy CREATE TABLE utasítás során (tárolt reláció megadásánál):

1. Egy attribútumot kulcsként adhatjuk meg a relációséma attribútumlistájának megadásánál.
2. A sémában megadott lista elemeit (vagyis az attribútumokat) használhatjuk egy olyan deklarációban, amely azt fejezi ki, hogy a szóban forgó attribútum vagy attribútumhalmaz kulcsot alkot a relációra nézve.

Ha a kulcs egynél több attribútumból áll, akkor a 2. módszert használjuk. Ha pedig egy attribútum alkotja, akkor bármelyik alkalmazható.

A kulcsjelleget kétféleképpen fejezhetjük ki:

- a) PRIMARY KEY, vagy
- b) UNIQUE.

Ha az *S* attribútumhalmazt az *R* reláció kulcsaként értelmezzük, akkor (akár PRIMARY KEY, akár UNIQUE kulcsszót használtunk) a hatás a következő lesz:

- Az R -nek nem lehet két olyan sora, ami az S attribútumok értékein meggyezne, még akkor sem, ha az értékek NULL értékek. Bármely olyan beszúrási vagy módosítási próbálkozás, amely sérti ezt a szabályt, azt okozza, hogy az ABKR visszautasítja a szabálysértő művelet végrehajtását.

Ha a PRIMARY KEY kulcsszót használjuk, akkor S attribútumaira nem szabad, hogy a NULL engedélyezve legyen. A rendszer itt is visszautasít minden próbálkozást, amely megséríti a szabályt. Ezzel szemben a UNIQUE esetén S -re megengedettek a NULL értéket. Az ABKR meg tudja különböztetni ezt a két feltételt, ha szükséges.

2.6. példa. Tekintsük újra a FilmSzínész reláció sémát. Mivel egyik filmszínész sem használja a másik nevét, ezért a név önmagában alkothat kulcsot a relációra nézve. Azaz ezt a tényt hozzáadhatjuk a név meghatározásához. A 2.9. ábra szemlélteti a 2.8. ábra változásait. A UNIQUE helyettesíthetné a PRIMARY KEY-t a megadásnál. Ez esetben viszont kettő vagy több sornak is lehetne a név értéke NULL, de ezenkívül más ismétlődő érték nem lesz erre az attribútumra nézve.

```
CREATE TABLE FilmSzínész (
    név CHAR(30) PRIMARY KEY,
    cím VARCHAR(255),
    nem CHAR(1),
    születésiDátum DATE
);
```

2.9. ábra. A név legyen kulcs

A kulcsokat külön is megadhatjuk. Ebben az esetben az eredményséma a 2.10. ábrán látható. Itt is kicserélhető a PRIMARY KEY UNIQUE-ra. □

```
CREATE TABLE FilmSzínész (
    név CHAR(30),
    cím VARCHAR(255),
    nem CHAR(1),
    születésiDátum DATE,
    PRIMARY KEY (név)
);
```

2.10. ábra. A kulcs külön történő megadása

2.7. példa. A 2.6. példában mind a 2.9. ábra, mind a 2.10. ábra formalizmusa megfelelő volt, hiszen a kulcs egy attribútumból állt. Ha több attribútum alkotja a kulcsot, akkor a 2.10. ábrán mutatott megadást kell használnunk. Például a

Filmek relációnál, ahol a `filmcím` és az `év` attribútumpár alkotott kulcsot, ez a 2.11. ábrán látható módon adható meg. Ahogy azt már megszokhattuk, a `UNIQUE` felcserélhető a `PRIMARY KEY`-vel. □

```
CREATE TABLE Filmek (
    filmcím      CHAR(100),
    év           INT,
    hossz        INT,
    műfaj         CHAR(10),
    stúdióNév    CHAR(30),
    producerAzon INT,
    PRIMARY KEY (filmcím, év)
);
```

2.11. ábra. A `filmcím` és `év` a `Filmek` kulcsa

2.3.7. Feladatok

2.3.1. feladat. Ebben a feladatban bemutatjuk az egyik, a relációs adatbázis-sémáknál gyakran használt példánkat. Az adatbázisséma négy relációt tartalmaz, amelyeknek a sémája a következő:

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

A `Termék` reláció megadja a gyártót, a modellszámot és a termékek típusát (`PC`, laptop vagy nyomtató). Az egyszerűség kedvéért a modellszámot egyedinek tekintjük az összes gyártóra és terméktípusra nézve. Ez a feltételezés általában nem helytálló, hiszen egy valós adatbázisban kellene egy kód a gyártókra is, amely a modellszámnak a része. A `PC` reláció megadja minden modellszámhöz a `PC` sebességét (a processzorét gigahertzben), a RAM-értéket (megabajtban), a merevlemez méretét (gigabajtban) és az árat. A `Laptop` reláció hasonló, a különbség csak az, hogy a képernyőméret (inch-ben) is meg van adva. A `Nyomtató` reláció minden nyomtatóhoz tárolja, hogy a nyomtató színes nyomtató-e (igaz, ha igen), illetve a nyomtató típusát (általában lézer- vagy tintasugaras) és az árat.

Adjuk meg a következő deklarációkat:

- A `Termék` reláció megfelelő sémáját.
- A `PC` reláció sémáját.
- A `Laptop` reláció sémáját.

- d) A Nyomtató reláció sémáját.
- e) A d) pontbeli Nyomtató séma módosítását úgy, hogy a színes attribútumot töröljük.
- f) A c) pontbeli séms módosítását úgy, hogy tartalmazzon egy cd mezőt, melynek alapértelmezett értéke legyen 'nincs', abban az esetben, ha a laptop nem tartalmaz CD-olvasót.

2.3.2. feladat. Ebben a feladatban bemutatjuk egy másik gyakran használt példánkat a II. világháború hadihajóiról. A következő relációk lesznek benne:

Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)

A hajók „osztályonként” azonos módon épülnek, és az osztály általában az osztály első hajójáról van elnevezve. A Hajóosztályok reláció tárolja az osztály nevét, típusát ('hh' a hadihajó, 'hc' a hadicirkáló), az építettő ország nevét, a fegyvereinek számát, a kaliberüket (az ágyúcső átmérőjét inch-ben) és a vízkiszorításukat (súly tonnában). A Hajók reláció tartalmazza a hajó nevét, az osztályának nevét, illetve a vízre bocsátásának dátumát. A Csaták reláció a hajók közötti csaták nevét és dátumát rögzíti. A Kimenetelek reláció pedig megadja az összes hajóra az összes csatájának az eredményét (elsüllyedt, megsérült vagy ép).

Adjuk meg a következő deklarációkat:

- a) Egy megfelelő sémát a Hajóosztályok reláció számára.
- b) A Hajók reláció sémáját.
- c) A Csaták reláció sémáját.
- d) A Kimenetelek reláció sémáját.
- e) Az a) pontban megadott Hajók reláció módosítását úgy, hogy töröljük ki a kaliber attribútumot.
- f) A b) pontban megadott Hajók reláció sémájának módosítását úgy, hogy tartalmazza a készült attribútumot, mely megadja azt a helyet, ahol a hajót készítették.

2.4. Egy algebrai lekérdező nyelv

Ebben a részben bevezetjük a relációs modell adatmanipulációs részeit. Emlékezzünk, hogy az adatmodell nem csupán egy struktúra, hanem kellenek módszerek az adat lekérdezésére és módosítására is. A relációkon értelmezett műveletek tárgyalása előtt be kell vezetnünk egy speciális algebrát, a *relációs algebrát*, amely egyszerű (de hatékony) lehetőségeket tartalmaz az adott relációkból új relációk létrehozására. Ha a relációk tárolt adatok, akkor az előállított reláció lehet egy, az adatokon történt lekérdezésre a válaszreláció. A relációs algebrát a forgalomban lévő ABKR-ek manapság már nem használják lekérdező nyelvként annak ellenére, hogy a kezdeti próbaverziók viszont direkt módon használták ilyen célokra. Az SQL, a „valódi” lekérdező nyelv, a relációs algebrára épül, és sok SQL-program valójában „szintaktikailag fűszerezett” relációs algebra. Sőt amikor az ABKR egy lekérdezést dolgoz fel, akkor az első dolog, ami az SQL-lekérdezéssel történik, hogy lefordítják vagy relációs algebrába, vagy egy hozzá nagyon hasonló belső formára. Azaz rengeteg oka van, hogy a relációs algebra tanulmányozásával kezdjük a tanulást.

2.4.1. Miért kell egy speciális lekérdező nyelv?

A relációs algebra műveleteinek bevezetése előtt megkérdezhetnénk, hogy miért is van szükségünk egy új típusú programozási nyelvre az adatbázisokhoz. Nem lenne elegendő a legelterjedtebb (mint C, illetve Java) nyelvek használata, hogy kérdéseket fogalmazhassunk meg, illetve hogy kiszámoljuk a kérdéshez tartozó relációt? Mindemellett a reláció egy sora kifejezhető struktúráként (C-ben) vagy objektumként (Javában), és így a relációt megadhatnánk ilyen elemek tömbjeként is. A meglepő válasz, hogy a relációs algebra azért hasznos, mivel *kevésbé* kifejezőbb, mint a C vagy a Java. Vagyis vannak számítások, amelyek megvalósíthatóak ezen nyelvekben, de a relációs algebrában nem. Egy példa erre, ha meg akarjuk határozni, hogy egy relácionak páros vagy páratlan számú sora van-e. Azzal, hogy korlátozzuk, hogy mit tudunk kifejezni, illetve tenni a lekérdező nyelvünkben, két nagy előnyre teszünk szert: kényelmes programozást és a fordító által magas szinten optimalizált kódok előállítását. Ezeket a 2.1.6. alfejezetben tárgyaltuk.

2.4.2. Mit nevezünk algebrának?

Egy algebra általában műveleteket és atomi operandusokat tartalmaz. Például egy számtani algebra esetén az x -hez hasonló változók és a 15-höz hasonló konstansok lesznek az atomi operandusok. A műveletek általában számtani műveletek lesznek: összeadás, kivonás, szorzás és osztás. Az algebra lehetővé teszi *kifejezések* megfogalmazását az atomi operandusokon és/vagy algebrai kifejezésekben végzett műveletek alkalmazásával. Gyakran zárójeleket is kell használnunk a műveletek operandusainak különválasztásához, mint például a következő kifejezéseknel: $(x + y) * z$ vagy $((x + 7)/(y - 3)) + x$.

A relációs algebra is példa egy algebrára. Az atomi operandusok a következők:

1. A relációkhöz tartozó változók.
2. Konstansok, amelyek véges relációt fejeznek ki.

A következőkben megmutatjuk a relációs algebra műveleteit.

2.4.3. A relácós algebra áttekintése

A hagyományos relációs algebrai műveletek négy osztályba sorolhatók:

- a) A hagyományos halmazműveletek – egyesítés, metszet és különbség – relációkra alkalmazva.
- b) Műveletek, amelyek a reláció egyes részeit eltávolítják: a „kvívalasztás” kihagy bizonyos sorokat, a „vetítés” bizonyos oszlopokat hagy ki.
- c) 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.
- d) Egy művelet, az „átnevezés”, amelyik nem befolyásolja a reláció sorait, de megváltoztatja a reláció sémáját, azaz az attribútumok neveit és/vagy a reláció nevét.

A relációs algebrai kifejezésekre általában *lekérdezések*ként hivatkozunk.

2.4.4. Relációkon értelmezett halmazműveletek

Halmazokon a három leggyakoribb művelet az egyesítés, a metszet és a 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 R -ben is és 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.

Ezen műveletek relációkra történő alkalmazásakor néhány feltételt kell szabnunk:

1. Az R és S relációk sémájának ugyanazt az attribútumhalmazt kell tartalmazniuk, illetve a típusoknak (értéktartományoknak) az összes megfelelő attribútumpárra meg kell egyezniük R -ben és S -ben.
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 minden két 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 minden két reláció sémájának megváltoztatásához használhatjuk a 2.4.11. alfejezetben bemutatott átnevezés operátort, és így ugyanazt az attribútumhalmazt tudjuk adni a relációknak.

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

Az R reláció

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

Az S reláció

2.12. ábra. Két reláció

2.8. példa. Tegyük fel, hogy van két relációból, R és S , amelyek a 2.2.8. alfejezetben található FilmSzínész reláció előfordulásai. R és S aktuális előfordulásait a 2.12. ábrán láthatjuk. R és S egyesítése, $R \cup S$ a következő reláció:

<i>név</i>	<i>cím</i>	<i>nem</i>	<i>születésiDátum</i>
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 minden két relációban szerepel.

Az $R \cap S$ metszet:

<i>név</i>	<i>cím</i>	<i>nem</i>	<i>születésiDátum</i>
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:

<i>név</i>	<i>cím</i>	<i>nem</i>	<i>születésiDátum</i>
Mark Hamill	456 Oak Rd., Brentwood	F	8/8/88

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

□

2.4.5. Vetítés

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

<i>filmcím</i>	<i>év</i>	<i>hossz</i>	<i>műfaj</i>	<i>stúdióNév</i>	<i>producerA</i>
Csillagok háborúja	1977	124	sci-fi	Fox	12345
Galaktikos küldetés	1999	104	vígjáték	DreamWorks	67890
Wayne világa	1992	95	vígjáték	Paramount	99999

2.13. ábra. A Filmek reláció

2.9. példa. Tekintsük a 2.2.8. alfejezetben megadott sémájú Filmek relációt. A reláció egy előfordulását a 2.13. ábrán láthatjuk. Ezt a relációt a következő kifejezés segítségével vetíthetjük az első három attribútumára:

$$\pi_{\text{filmcím}, \text{év}, \text{hossz}}(\text{Filmek})$$

Az eredményül kapott reláció a következő:

<i>filmcím</i>	<i>év</i>	<i>hossz</i>
Csillagok háborúja	1977	124
Galaktikos küldetés	1999	104
Wayne világa	1992	95

Megjegyzés az adatok minőségéről :-)

Miközben felettesebb odafigyeltünk arra, hogy a példáinkban minél pontosabb adatokat adjunk meg, ennek ellenére fiktív értékeket adtunk a lakcímekre és személyes adatokra nézve azért, hogy az előadóművészkek személyes jogait ne sértsük meg. Hiszen sokan közülük érzékenyek lehetnek a magánszférájukra.

Másik példaként levetíthetjük a `Filmek` relációt a `műfaj` attribútumra a $\pi_{műfaj}(Filmek)$ kifejezéssel. Ekkor az eredményül kapott reláció csak egyetlen oszlopot tartalmaz:

<i>műfaj</i>
<hr/>
sci-fi
vígjáték

Figyeljük meg, hogy az eredményben csak két sor található, mivel a 2.13. ábrán látható utolsó két sor `műfaj` attribútumának értéke ugyanaz. \square

2.4.6. Kiválasztás

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

A C egy olyan feltételkifejezés, amilyet megszoktunk a hagyományos programozási nyelveknél. Például az `if` kulcsszót feltételkifejezés követi mind a C, mind pedig a Java programozási nyelvekben. Az egyetlen különbség az, hogy a C feltételben levő operandusok vagy konstansok, vagy az R attribútumai. Alkalmazzuk a C feltételt 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 C értéke igaz, akkor a t sor egyike azoknak a soroknak, amelyek megjelennek a $\sigma_C(R)$ eredményében; egyébként t nincs az eredményben.

2.10. példa. Tekintsük a 2.13. ábrán látható *Filmek* relációt. Ebben az esetben a $\sigma_{\text{hossz} \geq 100}(\text{Filmek})$ kifejezés értéke a következő:

filmcím	év	hossz	műfaj	stúdióNév	producerA.
Csillagok háborúja	1977	124	sci-fi	Fox	12345
Galaktitkos küldetés	1999	104	vígjáték	DreamWorks	67890

Az első sor kielégíti a $\text{hossz} \geq 100$ feltételt, hiszen ha behelyettesítjük a hossz helyébe az első sor megfelelő komponensét, a 124-et, akkor a feltétel így néz ki: $124 \geq 100$. Ez utóbbi feltétel igaz, ezért az első sort elfogadjuk. Ugyanilyen indokok magyarázzák, hogy a 2.13. ábra második sora miért kerül be az eredménybe.

A harmadik sor hossz komponense 95. Ezért, amikor behelyettesítünk a hossz attribútumba, a $95 \geq 100$ hamis feltételt kapjuk. Ennél fogva a 2.13. ábra utolsó sora nincs az eredményben. \square

2.11. példa. Tegyük fel, hogy a *Filmek* relációból azon sorok halmazát szeretnék 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:

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

Az eredmény:

filmcím	év	hossz	műfaj	stúdióNév	producerAzon
Csillagok háborúja	1977	124	sci-fi	Fox	12345

Az eredményrelációnak egyetlenegy sora van. \square

2.4.7. Descartes-szorzat

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

Az eredményreláció sémája 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 tételehez, amelyik mind az R , mind az S sémájában szerepel, a nevek megkülönböztetésére az $R.A$, illetve $S.A$ jelölésekkel használjuk attól függően, hogy az R reláció A attribútumáról vagy az S reláció A attribútumáról van szó.

	A	B
1		2
3		4

(a) Az R reláció

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

(b) 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

(c) Az $R \times S$ eredménye

2.14. ábra. Két reláció és a Descartes-szorzatuk

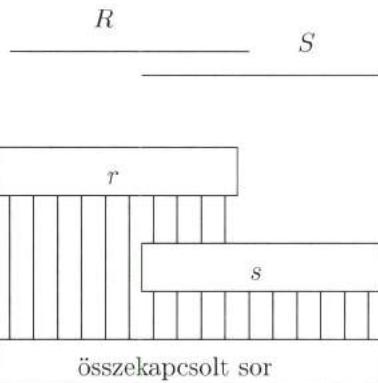
2.12. példa. A tömörség kedvéért használjunk egy absztrakt példát a szorzás-művelet szemléltetésére. Tekintsük a 2.14 (a), illetve a 2.14 (b) ábrán látható R és S relációkat, az ábrán megadott sémákkal és sorokkal. Amint az a 2.14 (c) ábrán is látható, az $R \times S$ szorzat hat sorból áll. Figyeljük meg, hogyan párosítottuk R két sorának mindegyikét S három sorának mindegyikével. Mivel a B minden sémának attribútuma, ezért az $R.B$ és $S.B$ jelöléseket használtuk az $R \times S$ sémájában. A többi attribútum egyértelmű, ezeknek a nevei változatlanok az eredmény sémájában. \square

2.4.8. Természetes összekapcsolás

Két reláció szorzásánál jóval gyakrabban van szükségünk arra, hogy *összekapcsoljunk* relációkat oly módon, hogy csak azokat a sorokat párosítjuk, amelyek valamilyen módon összeillesznek. 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 R és S sémájának összes közös attribútumán. Még pontosabban: legyenek

A_1, A_2, \dots, A_n azok az attribútumok, amelyek megtalálhatók mind az R , mind az S sémájában. R egy r sorának és S egy s sorának párosítása akkor és csak akkor sikeres, ha r és s megfelelő értékei megegyeznek az összes A_1, A_2, \dots, 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 az S összes attribútumán. Mivel r és s összekapcsolása sikeres volt, így tudjuk, hogy 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 sorral a minden sémában szereplő attribútumokon. Az összekapcsolt sorok szerkezetét szemlélteti a 2.15. ábra. Az attribútumok sorrendjének viszont nem feltétlenül kell megegyezni R és S attribútumainak a sorrendjével.



2.15. ábra. Sorok összekapcsolása

2.13. példa. A 2.14. ábra (a) és (b) részén látható R és S relációk természetes összekapcsolásának eredménye:

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

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 (R -ből), a B attribútumhoz (R -ből vagy S -ből), a C attribútumhoz (S -ből) és a D attribútumhoz (S -ből).

Ebben a példában R első sora csak az S első sorával párosítható sikeresen; minden sor B attribútumának értéke 2. Ennek a párosításnak az eredménye az első sor: $(1, 2, 5, 6)$. R második sora csak az S második sorával párosítható sikeresen, és ez a párosítás a $(3, 4, 7, 8)$ sort eredményezi. Megfigyelhetjük, hogy S harmadik sora nem párosítható 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 nevezünk. \square

2.14. 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 2.16. 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 U első sora sikeresen párosítható a V első két sorával, viszont U második és harmadik sora csak a V harmadik sorával párosítható sikeresen. A négy párosítás eredménye a 2.16 (c) ábrán látható. \square

2.4.9. 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, 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 szorzásműveletnél, az eredmény séma-ja itt is az R és S séma-janak egyesítése. Ha szükséges megjelölni, hogy az attribútumok melyik sémából származnak, akkor használjuk az attribútumokhoz az „ $R.$ ”, illetve „ $S.$ ” előtagokat.

2.15. példa. Tekintsük az $U \bowtie_{A < D} V$ műveletet, ahol U és V a 2.16. ábra (a) és (b) részében látható két reláció. Figyelembe kell vennünk a relációk sorai-nak mind a kilenc lehetséges párosítását, és meg kell néznünk, hogy vajon U sorának A komponense kisebb-e, mint V sorának D komponense. U első sorában az A komponens értéke 1, és ez a sor sikeresen párosítható V összes sorával.

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

(a) Az U reláció

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

(b) A V reláció

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

(c) Az $U \bowtie V$ eredménye**2.16. ábra.** Relációk természetes összekapcsolása

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 2.17. ábrán látható.
□

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

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

Figyeljük meg, hogy a 2.17. 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.

2.16. 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 U sorának A komponense kisebb legyen, mint V sorának D komponense, hanem annak is teljesülnie 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. \square

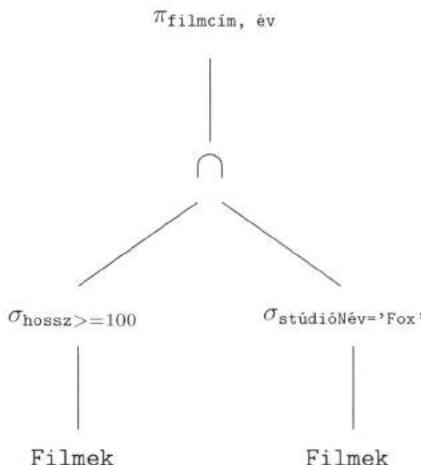
2.4.10. Lekérdezések megfogalmazása műveletek segítségével

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 tételere. Lehetséges a kifejezések kifejezésfával történő megadása is: ez utóbbit nekünk könnyebb olvasni, de géppel kevésbé olvasható.

2.17. példa. Tekintsük az általunk megadott `Filmek` relációt. 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 *Filmek* relációból azokat a sorokat, amelyekre $\text{hossz} \geq 100$.
2. Kiválasztjuk a *Filmek* relációból azokat a sorokat, amelyekre a $\text{stúdióNév} = \text{'Fox'}$.
3. Kiszámoljuk az 1. és 2. metszetét.
4. A 3. lépésben megkapott relációt levetítjük a *filmcím* és *év* attribútumokra.



2.18. ábra. Egy relációs algebrai kifejezés kifejezésfája

A 2.18. ábrán a fenti lépéseknek megfelelő kifejezésfát láthatjuk. A kifejezésfa kiértékelése alulról felfelé történik. A belső csúcsokban az argumentumokra (a gyerekekben kapott eredményekre) alkalmazzuk a megfelelő műveleteket. Az alulról felfelé kiértékelésnél az argumentumok a megfelelő részknél minden rendelkezésre fognak állni. 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.

$$\pi_{\text{filmcím, év}}(\sigma_{\text{hossz} \geq 100}(\text{Filmek}) \cap \sigma_{\text{stúdióNév} = \text{'Fox'}}(\text{Filmek})).$$

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

$$\pi_{\text{filmcím, év}}(\sigma_{\text{hossz} \geq 100 \text{ AND } \text{stúdióNév} = \text{'Fox'}}(\text{Filmek}))$$

ekvivalens az előző kifejezéssel. \square

Ekvivalens kifejezések és 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 1.2.5. alfejezetben vázlatosan tárgyalt lekérdezés-válaszolónak egyik fontos feladata éppen az, hogy egy relációs algebrai kifejezést olyan ekvivalens kifejezéssel helyettesítsen, amely hatékonyabban értékelhető ki.

2.4.11. Elnevezés és átnevezés

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

2.18. példa. A 2.12. példában kiszámoltuk a 2.14. ábra (a) és (b) részében látható két reláció, R és S szorzatát. Megállapodtunk abban is, hogy ha egy attribútum minden operandusban szerepel, akkor ezeket az attribútumokat átnevezzük oly módon, hogy az új névben az attribútumokhoz tartozó relációk neve is jelenjen meg előtagként. 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 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 2.14. 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 2.14 (c) ábrán látható $R \times S$ reláció, kivéve, hogy az öt attribútum neve balról jobbra: A, B, X, C és D . A 2.19. ábrán ez a reláció látható.

Egy másik lehetőség az, hogy amint a 2.12. példában is tettük, kiszámoljuk a szorzatot átnevezés nélkül, ezt követően pedig átnevezzük az eredményt. A

$$\rho_{RS(A,B,X,C,D)}(R \times S)$$

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

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

kifejezés eredményének ugyanazok az attribútumai, mint a 2.19. ábrán látható relációnak, viszont a neve RS , míg a 2.19. ábrán látható relációnak nincs neve.
 \square

2.4.12. Műveletek közötti kapcsolatok

A 2.4. 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 minden sorban a R -ben és a S -ben is benne vannak, de T -ben nem sorai. Azaz kivonjuk R -ból S -et. Ezután kivonjuk R -ból 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:

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

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

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

2.19. példa. A 2.16. 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 végezzünk 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 2.16. 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. \square

Az ebben az alfejezetben említett 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.

2.4.13. Egy lineáris jelölési mód az algebrai kifejezésekhez

A 2.4.10. alfejezetben a kifejezsfákat használtuk az összetett relációs algebrai kifejezések leírására. Egy másik megközelítés az, hogy a belső csúcsoknak megfelelő ideiglenes relációknak nevet adunk, és egy értékadási sorozatot adunk meg az értékeik meghatározására. Az értékadások sorrendje rugalmas abban az értelemben, hogy ha az N csúcs összes gyerekének értéke már adott, akkor már kiszámíthatjuk N értékét is.

Az értékadó utasításokat a következőképpen jelölhetjük:

1. Először szerepel a reláció neve és a reláció attribútumainak zárójelezett listája. A **Válasz** nevet az utolsó lépés eredményének jelölésére használjuk, azaz a kifejezésfa gyökeréhez tartozó reláció neveként.
2. Ezt követi a $::=$ értékadási szimbólum.
3. A jobb oldalon pedig egy algebrai kifejezés áll. minden értékadásnál használhatunk akár egyetlen műveletet is, ekkor tulajdonképpen a kifejezésfa minden csúcsának pontosan egy értékadás felel meg. Viszont kombinálhatjuk is a jobb oldalon a relációs algebrai műveleteket, ami szintén egy elfogadott módszer.

2.20. példa. Vegyük a 2.18. ábra kifejezésfáját. Az értékkadások egy lehetséges sorozata a kiértékeléshez a következő lehetne:

$$\begin{aligned} R(fc, \acute{e}, h, m, s, p) &:= \sigma_{\text{hossz} \geq 100}(\text{Filmek}) \\ S(fc, \acute{e}, h, m, s, p) &:= \sigma_{\text{st\'udi\'on\'ev} = 'Fox'}(\text{Filmek}) \\ T(fc, \acute{e}, h, m, s, p) &:= R \cap S \\ \text{V\'alasz(filmcím, \acute{e}v)} &:= \pi_{fc, \acute{e}}(T) \end{aligned}$$

Az első lépésben a 2.18. ábra $\sigma_{\text{hossz} \geq 100}$ címkéjű belső csúcsát számítjuk ki. A második lépésben pedig a $\sigma_{\text{st\'udi\'on\'ev} = 'Fox'}$, értékét számítjuk. Megjegyezzük, hogy „ingyen” átnevezést kaphatunk, hiszen az értékkadás bal oldalán tetszőleges relációnevet és attribútumnevet is használhatnánk. A fennmaradó két lépésben természetesen a metszetet és a vetítést végezzük el.

A lépések összevonása is megengedett. Például az utolsó két lépést össze is vonhatnánk:

$$\begin{aligned} R(fc, \acute{e}, h, m, s, p) &:= \sigma_{\text{hossz} \geq 100}(\text{Filmek}) \\ S(fc, \acute{e}, h, m, s, p) &:= \sigma_{\text{st\'udi\'on\'ev} = 'Fox'}(\text{Filmek}) \\ \text{V\'alasz(filmcím, \acute{e}v)} &:= \pi_{fc, \acute{e}}(R \cap S) \end{aligned}$$

Akár R -et és S -et is beírhatnánk az utolsó sorba, és így egy egysoros kifejezést kaphatnánk. \square

2.4.14. Feladatok

2.4.1. feladat. Ez a feladat a 2.3.1. feladatban szereplő termékek sémáján alapszik. Itt megismételjük azt az adatbázissémát, amely négy táblán alapult. Ezek sémája az alábbi volt:

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

A Termék reláció néhány mintaadata a 2.20. ábrán látható. A másik három reláció mintaadatai a 2.21. ábrán láthatók. A gyártók, illetve a modellszámok fiktív adatok, de a többi adat az 2007. év elejének piacát tükrözi.

<i>gyártó</i>	<i>modell</i>	<i>típus</i>
A	1001	pc
A	1002	pc
A	1003	pc
A	2004	laptop
A	2005	laptop
A	2006	laptop
B	1004	pc
B	1005	pc
B	1006	pc
B	2007	laptop
C	1007	pc
D	1008	pc
D	1009	pc
D	1010	pc
D	3004	nyomtató
D	3005	nyomtató
E	1011	pc
E	1012	pc
E	1013	pc
E	2001	laptop
E	2002	laptop
E	2003	laptop
E	3001	nyomtató
E	3002	nyomtató
E	3003	nyomtató
F	2008	laptop
F	2009	laptop
G	2010	laptop
H	3006	nyomtató
H	3007	nyomtató

2.20. ábra. A Termék reláció mintaadatai

<i>modell</i>	<i>sebesség</i>	<i>memória</i>	<i>merevlemez</i>	<i>ár</i>
1001	2.66	1024	250	2114
1002	2.10	512	250	995
1003	1.42	512	80	478
1004	2.80	1024	250	649
1005	3.20	512	250	630
1006	3.20	1024	320	1049
1007	2.20	1024	200	510
1008	2.20	2048	250	770
1009	2.00	1024	250	650
1010	2.80	2048	300	770
1011	1.86	2048	160	959
1012	2.80	1024	160	649
1013	3.06	512	80	529

(a) A PC reláció mintaadatai

<i>modell</i>	<i>sebesség</i>	<i>memória</i>	<i>merevlemez</i>	<i>képernyő</i>	<i>ár</i>
2001	2.00	2048	240	20.1	3673
2002	1.73	1024	80	17.0	949
2003	1.80	512	60	15.4	549
2004	2.00	512	60	13.3	1150
2005	2.16	1024	120	17.0	2500
2006	2.00	2048	80	15.4	1700
2007	1.83	1024	120	13.3	1429
2008	1.60	1024	100	15.4	900
2009	1.60	512	80	14.1	680
2010	2.00	2048	160	15.4	2300

(b) A Laptop reláció mintaadatai

<i>modell</i>	<i>színes</i>	<i>típus</i>	<i>ár</i>
3001	igen	tintasugaras	99
3002	nem	lézer	239
3003	igen	lézer	899
3004	igen	tintasugaras	120
3005	nem	lézer	120
3006	igen	tintasugaras	100
3007	igen	lézer	200

(c) A Nyomtató reláció mintaadatai

2.21. ábra. A 2.4.1. feladat relációinak mintaadatai

Írjuk fel a következő lekérdezésekhez tartozó relációs algebrai kifejezéseket. A 2.4.13. alfejezetben bemutatott lineáris jelölésmód használható. A 2.20. és a 2.21. 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.

- a) Melyek azok a PC-modellek, amelyek sebessége legalább 3.00?
- b) Mely gyártók készítenek legalább száz 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, amelyek laptopot árulnak, PC-t viszont nem?
- ! f) Melyek azok a merevlemezretek, amelyek legalább két PC-ben megtalálhatók?
- ! g) Adjuk meg azokat a PC-modell párokat, amelyek ugyanolyan gyorsak és a memoriájuk is ugyanakkora. Egy pár csak egyszer jelenjen meg, azaz ha (i, j) már szerepel, akkor (j, i) ne jelenjen meg.
- !! h) Melyek azok a gyártók, amelyek gyártanak legalább két, egymástól különböző, legalább 2.80 gigahertzen 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, amelyek pontosan három típusú PC-t forgalmazznak?

2.4.2. feladat. A 2.4.1. feladatban megfogalmazott lekérdezéseknek rajzoljuk fel a kifejezésfáit is.

2.4.3. feladat. Ez a példa, a 2.3.2. feladatban megfogalmazott témával, a II. világháború csatahajójival foglalkozik, és a következő sémájú relációkat tartalmazza:

Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)

osztály	típus	ország	agyúkSzáma	kaliber	vízkiszorítás
Bismarck	bb	Németország	8	15	42000
Iowa	bb	USA	9	16	46000
Kongo	bc	Japán	8	14	32000
North Carolina	bb	USA	9	16	37000
Renown	bc	Nagy-Brittannia	6	15	32000
Revenge	bb	Nagy-Brittannia	8	15	29000
Tennessee	bb	USA	12	14	32000
Yamato	bb	Japán	9	18	65000

(a) A Hajóosztályok reláció mintaadatai

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

(b) A Csaták reláció mintaadatai

hajó	csata	eredmény
Arizona	Pearl Harbor	elsüllyedt
Bismarck	Denmark Strait	elsüllyedt
California	Surigao Strait	ép
Duke of York	North Cape	ép
Fuso	Surigao Strait	elsüllyedt
Hood	Denmark Strait	elsüllyedt
King George V	Denmark Strait	ép
Kirishima	Guadalcanal	elsüllyedt
Prince of Wales	Denmark Strait	megsérült
Rodney	Denmark Strait	ép
Scharnhorst	North Cape	elsüllyedt
South Dakota	Guadalcanal	megsérült
Tennessee	Surigao Strait	ép
Washington	Guadalcanal	ép
West Virginia	Surigao Strait	ép
Yamashiro	Surigao Strait	elsüllyedt

(c) A Kimenetelek reláció mintaadatai

2.22. ábra. A 2.4.3. feladat adatai

<i>név</i>	<i>osztály</i>	<i>felavatva</i>
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

2.23. ábra. A Hajók reláció mintaadatai

A 2.22. és a 2.23. ábra a fenti négy reláció mintaadatait tartalmazza.⁴ Figyeljük meg, hogy a 2.4.1. feladat adataival ellentétben ezek az adatok tartalmaznak „lőgó 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 2.4.13. alfejezetben bemutatott lineáris jelölésmód használható. A 2.22. és a 2.23. ábrán látható adatok alapján számoljuk ki a lekérdezések eredményeit. A válaszul felírt kifejezéseknek más adatokra is a helyes választ kell adniuk, nem csak az ábrán látható adatokra.

- 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 előtt avattak fel?
- c) Adjuk meg a Denmark Strait-i csatában elsüllyedt hajók nevét.

⁴ 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.

- 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 guadalcanali csatában részt vett hajók nevét, vízkiszorítását és ágyúnak számát.
- f) Adjuk meg az adatbázisban szereplő összes hadihajó nevét. (Ne feledjük, hogy a Hajók relációban nem feltétlenül szerepel az összes hajó!)
- ! g) Adjuk meg azokat az osztályokat, amelyekbe csak egyetlenegy hajó tartozik.
- ! h) Melyek azok az országok, amelyeknek csatahajójuk is és cirkálóhajójuk is voltak?
- ! i) Adjuk meg azokat a hajókat, amelyek „újjáéledtek”, azaz egyszer már meg-sérültek egy csatában, de egy későbbi csatában újra harcoltak.

2.4.4. feladat. Az előző feladatban kapott kifejezések mindegyikéhez rajzoljuk fel a megfelelő kifejezésfát.

2.4.5. 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?

! 2.4.6. feladat. Egy relációkon értelmezett operátor akkor *monoton*, ha bárminelyik argumentumrelációhoz egy újabb sort hozzávéve az eredmény tartalmazza az összes olyan sort, amelyet addig tartalmazott és esetleg újabb sorokat is. Az ebben a fejezetben felsorolt operátorok közül melyek monotonok? Mindegyik operátorra indokoljuk, hogy miért monoton, illetve ha nem monoton, adjunk ellenpéldát.

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

- $R \cup S$.
- $R \bowtie S$.
- $\sigma_C(R) \times S$, tetszőleges C feltétel esetén.
- $\pi_L(R) - S$, tetszőleges L attribútumlista esetén.

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

! 2.4.9. feladat. Az R és S relációk *nem félíg-összekapcsolása*, $R \not\bowtie S$ az R azon t sorainak halmaza, amely sorok R és S összes attribútumain nem eggyeznek meg S egyetlen sorával sem. Adjunk meg olyan három különböző relációs algebrai kifejezést, amelyek ekvivalensek az $R \not\bowtie S$ kifejezéssel.

!! 2.4.10. feladat. Az R reláció sémája

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

és az S reláció sémája (B_1, B_2, \dots, B_m) ; azaz S összes attribútuma benne van R attribútumainak halmazában. Az R és S *hányadosa*, $R \div S$ megadja azon A_1, A_2, \dots, 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.

2.5. Relációkra vonatkozó megszorítások

Most térjünk rá az adatmodell harmadik legfontosabb részére: az adatbázisban tárolható adatok megszorításainak a képességére. Ezeket is az adatbázisban tároljuk le. Eddig csak egyetlen típusú megszorítást láthattunk, mégpedig azt, mikor egy vagy több attribútum kulcsot alkothat (lásd 2.3.6. alfejezet). Ez utóbbi, illetve sok egyéb megszorítás is kifejezhető a relációs algebrában. Ennek az alfejezetnek a során láttni fogjuk, hogyan fejezhetjük ki minden a kulcsmegszorítást, minden a „hivatkozási épseg” megszorítást. Ez utóbbi elvárás az, hogy ha egy érték egy reláció egy oszlopában előfordul, akkor ez szerepelni fog a reláció vagy egy másik reláció valamelyik oszlopában is. A 7. fejezetben láttni fogjuk, hogyan kényszeríti ki ezt a relációs algebrában kifejezhető megszorítást az SQL adatbázisrendszere.

2.5.1. Megszorítások megadása relációs algebra segítségével

Megszorításokat két módon fejezhetünk ki relációs algebrai kifejezésekkel:

- Ha R egy relációs algebrai kifejezés, akkor $R = \emptyset$ egy olyan megszorítás, amelynek jelentése: „ R -nek üresnek kell lennie” vagy másképpen fogalmazva „ R eredményében egyetlen sor sincs”.
- Ha R és S relációs algebrai kifejezések, akkor $R \subseteq S$ egy olyan megszorítás, melynek jelentése: „ R eredményének minden sora benne kell legyen S eredményében”. Természetesen S eredménye tartalmazhat R sorain kívül más sorokat is.

A megszorítások megadásának ez a két módja a kifejezőről 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 $R - S$ üres. Fordítva, ha $R - S$ egyetlen sort sem tartalmaz, akkor R minden sorának benne kell lennie S -ben, máskülönben $R - S$ 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 7. 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.

2.5.2. Hivatkozási épség

A megszorítások egy gyakori formája a *hivatkozási épség*, mely szerint ha egy érték megjelenik valahol egy környezetben, akkor ugyanez az érték egy másik, az előzővel összefüggő környezetben is megjelenik. Például a filmeket nyilvántartó adatbázisunkban elvárhatjuk, hogy a **SzerepelBenne** egy sorában lévő p **színészNév** komponens esetén egy olyan színésznek a nevéről beszélünk, aki benne van a **FilmSzínész** relációban is. Ellenkező esetben ugyanis feltehetnénk a kérdést, hogy az illető „színész” valóban színész-e.)

Általánosságban, ha az R reláció egy sorának A attribútumában szerepel egy v érték, akkor tervezési szándékaink miatt elvárhatjuk, hogy ez a v érték egy másik S reláció valamely sorának egy bizonyos komponensében (például B -ben) is megjelenjen. A hivatkozási épséget relációs algebrában kifejezhetjük a $\pi_A(R) \subseteq \pi_B(S)$ vagy az (ezzel ekvivalens) $\pi_A(R) - \pi_B(S) = \emptyset$ kifejezések valamelyikével.)

2.21. példa. Vegyük a filmkről szóló szokásos példánkból a következő két reláció sémáját:

```
Filmek(filmcím, év, hossz, műfaj, 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 **Filmek** 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:

$$\pi_{\text{producerAzon}}(\text{Filmek}) \subseteq \pi_{\text{azonosító}}(\text{GyártásIrányító})$$

A bal oldali kifejezés megadja a Filmek 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. \square

2.22. példa. Egynél több attribútum értékére vonatkozó hivatkozási épseget hasonló módon fejezhetünk ki. Például biztosítani akarjuk, hogy a

$$\text{SzerepelBenne}(\text{filmCím}, \text{filmÉv}, \text{színészNév})$$

relációban szereplő mindegyik film szerepeljen a

$$\text{Filmek}(\text{filmcím}, \text{év}, \text{hossz}, \text{műfaj}, \text{stúdióNév}, \text{producerAzon})$$

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

$$\pi_{\text{filmCím}, \text{filmÉv}}(\text{SzerepelBenne}) \subseteq \pi_{\text{filmcím}, \text{év}}(\text{Filmek})$$

a relációk megfelelő komponenseinek levetítésével nyert filmcím-év párok össze-hasonlításával fejezi ki a hivatkozási épseget. \square

2.5.3. Kulcsmegszorítás

Ugyanezen megszorítási jelöléssel többet is kifejezhetünk, mint a hivatkozási épseg. A most következőkben látni fogjuk, hogyan fejezhető ki algebrai úton az a megszorítás, hogy egy konkrét attribútum vagy egy attribútumhalmaz kulcsot kell alkossan a relációra nézve.

2.23. példa. Emlékezzünk, hogy a név a

$$\text{FilmSzínész}(\text{név}, \text{cím}, \text{nem}, \text{születésiDátum})$$

reláció kulcsa volt. Ezért egyetlen sorpárnak sem egyezik meg a név komponense a másikéval. Ezt a megszorítást algebrailag egy implikációval fejezhetjük ki, azaz: ha két sor megegyezik a név komponensében, akkor a cím értékükben is meg kell egyezniük. Megjegyezve, hogy ezen „két” sorra vonatkozó állítás (azaz megegyeznek a név komponensben), csak akkor lehetséges, ha a két sor megegyezik, azaz minden attribútumukban azonosak.

Az elv az, hogy ha a FilmSzínész összes (t_1, t_2) sorpárját képezzük, akkor nem lehet olyan pár, amely a név komponensen megegyezik, de a cím komponensen eltér. Az összes pár előállítására képezzük a direkt szorzatot, és a feltételt megsértő párok meghatározására kiválasztást használunk. A megszorítás megadható úgy, hogy a kiértékelés eredménye egyezzen meg \emptyset -val.

Mivel a relációt önmagával szorozzuk meg, ezért legalább az egyik másolatainak az attribútumneveit át kell neveznünk. A tömörseg kedvéért használjuk az FSZ1 és FSZ2 új neveket a FilmSzínész relációra való hivatkozásnál. Ekkor az elvárás kifejezhető az alábbi algebrai megszorítással:

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

A fenti kifejezésben az $\text{FSZ1} \times \text{FSZ2}$ szorzatban szereplő FSZ1 a következő átnevezésnek a rövidítése:

$$\rho_{\text{FSZ1(név, cím, nem, születésiDátum)}}(\text{FilmSzínész})$$

Az FSZ2 szintén a FilmSzínész ehhez hasonló átnevezése lesz. \square

2.5.4. További példák megszorításokra

Többféle megszorítás van, amelyet kifejezhetünk relációs algebrában, és amelyek hasznosak lesznek az adatbázis korlátozására nézve is. A megszorítások egy nagyobb családját bizonyos értékek tiltása alkotja. Például, hogy minden attribútumnak van egy típusmegszorítása az attribútum értékeire nézve. Gyakran elég egyértelműek ezek a megszorítások, mint például: az attribútum típusa legyen egész szám vagy 30 hosszúságú karakterlánc. Más esetekben azt szeretnénk, ha az attribútumban lévő értékek egy kisebb felsorolható halmaz értékeire korlátozódnának. Megint más esetekben viszont elvárhatjuk, hogy összetett korlátozásokat adhassunk meg az előforduló értékekre. Két egyszerű példát fogunk adni: az első egy attribútum egyszerű *tartománymegszorítása*, a másik pedig egy bonyolultabb elvárás lesz.

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

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

Ez azt jelenti, hogy a FilmSzínész reláció azon sorainak halmaza, amelyekben a nem komponens értéke sem nem 'N', sem nem 'F', üres. \square

2.25. példa. Tegyük fel, hogy valaki csak akkor lehet egy filmstúdió elnöke, ha a nettó bevétele legalább 10 000 000 \$. Ezt a következő módon fejezhetjük ki algebraileg. Először is szükségünk van a következő két reláció thétaösszekapcsolására:

$$\begin{aligned} &\text{GyártásIrányító(név, cím, azonosító, nettóBevétel)} \\ &\text{Stúdió(név, cím, elnökAzon)} \end{aligned}$$

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, olyan halmazt kapunk, amely a megszorításunk értelmében üres. Tehát a megszorítást a következőképpen fejezhetjük ki:

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

Ennek a megszorításnak a kifejezésére egy másik lehetőség az, ha a stúdió elnökeinek azonosítóból kapott halmazt hasonlíttuk össze azon gyártásirányítók azonosítójának halmzával, akiknek a nettó bevétele legalább 10 000 000 \$. Ez előző részhalmaza kell legyen az utóbbinak. A következő tartalmazás éppen ezt fejezi ki:

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

□

2.5.5. Feladatok

2.5.1. feladat. Írjuk fel a következő megszorításokat a 2.3.1. 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 2.4.1. 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 2.00, nem árulhatják 500 \$-nál drágábban.
- b) A 15,4 hüvelyknél kisebb képernyőjű laptopoknak a merevlemeze legalább 100 gigabájtos kell legyen, ellenkező esetben csak 1000 \$-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.
- ! e) Ha egy laptopnak nagyobb memóriája van, mint egy PC-nek, akkor a laptop drágább kell legyen, mint a PC.

2.5.2. feladat. Fejezzük ki relációs algebrában a következő megszorításokat. A megszorítások a 2.3.2. feladat relációira vonatkoznak.

Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)

A megszorításokat tartalmazással és üres halmaz segítségével is kifejezhetjük. A 2.4.3. 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 kettőnél több hajót.
 - ! d) Egy országnak sem lehet csatahajója és cirkálóhajója is egyszerre.
 - !! e) A több mint 9 ágyúval rendelkező hajó nem csatázhat olyan 9-nél kevesebb ágyúval rendelkező hajóval, amelyik elsüllyedt.
- ! 2.5.3. feladat.** Tegyük fel, hogy van két relációból, R és S . Legyen C egy olyan hivatkozási épseg, amely azt állítja, hogy ha az R relációban szerepel egy olyan sor, amelynek az A_1, A_2, \dots, A_n attribútumokon vett értéke rendre v_1, v_2, \dots, v_n , akkor az S relációban szerepelnie kell egy olyan sornak, amelynek a B_1, B_2, \dots, B_n attribútumokon vett értéke rendre v_1, v_2, \dots, v_n . Írjuk fel ezt a C megszorítást relációs algebrában.
- ! 2.5.4. feladat.** Másik algebrai módszer egy megszorítás kifejezésére az $E_1 = E_2$ alak, ahol E_1 és E_2 relációs algebrai kifejezések. Tud-e többet kifejezni a megszorításnak ez a formája, mint az ebben a fejezetben megbeszélt két társa?

2.6. Összefoglalás

- ◆ *Adatmodellek:* Az adatmodell egy jelölésmód egy adatbázis adatszerkezeteinek a leírására, beleértve az adatra vonatkozó megszorításokat is. Az adatmodell általában az adatokon végezhető műveletek leírására alkalmas jelölést is magába foglalja. A műveletek lehetnek lekérdezések, illetve adatmódosítások.
- ◆ *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.

3. fejezet

Relációs adatbázisok tervezésének elmélete

Egy alkalmazáshoz kapcsolódó relációs adatbázisséma megtervezésekor nagyon sokféle módon járhatunk el. A 4. fejezetben számos magas szintű adatszerkezetet leíró módszerrel ismerkedhetünk meg, illetve azokkal a módszerekkel, amelyekkel ezek a magas szintű leírások sémákká alakíthatók. Továbbá tanulmányozzuk az adatbázisokra vonatkozó követelményeket és megvizsgáljuk, hogyan definiálhatunk relációkat a magas szintű köztes lépés kihagyásával. Bármilyen megközelítést is használunk, leggyakrabban egy kiinduló sémát készítünk, amelyet javítani kell, például a redundancia kiküszöbölésével. Gyakran az adatbázissámkal kapcsolatos problémákat az okozza, hogy túl sok minden szeretnénk egyetlen sémába összekombinálni.

Szerencsére az adatbázisok tervezésének elméleti háttere jól kidolgozott: a „függőségek” és azok következményei segítségével jó sémákat állíthatunk elő, illetve segítenek megoldani azokat a helyzeteket, amikor egy séma hibát kell kijavítani. Ebben a fejezetben először azonosítjuk azokat a helyzeteket, amelyekben bizonyos függőségek jelenléte a sémában problémát okoz. Ezeket a problémákat „anomáliáknak” nevezzük.

A részletes tárgyalást a „funkcionális függőségek” vizsgálatával kezdjük, amely a relációk kulesának általánosítása. Ezután a funkcionális függőség fogalmát felhasználva normálformákat állítunk elő az adatbázisokhoz. Ennek az elméletnek hatása a „normalizáció”, melynek során egyes relációkat két vagy több részre vágunk, így megszüntetve az anomáliákat. Ezután bemutatjuk a „többértékű függőségeket”, amelyek azt fogalmazzák meg, hogy egy reláció egy vagy több attribútuma független egy vagy több másik attribútumtól. Ezek a függőségek szintén normálformákhoz, illetve relációk felbontásához vezetnek a redundancia kiküszöbölése érdekében.

3.1. Funkcionális függőségek

A funkcionális függőségek elmélete egy olyan tervezési elvet szolgáltat, amellyel alaposan tanulmányozhatjuk a relációkat, és néhány fontos alapelve segítségével javíthatjuk őket. Első lépésként megállapítjuk azokat a megszorításokat, amelyeket alkalmazni kell a relációra. A leggyakoribb megszorítás a „funkcionális függőség”, amely a 2.5.3. alfejezetben tárgyalt kulcsfogalom általánosításával fogalmaz meg állításokat. A fejezet későbbi részében látni fogjuk, hogy ez az elmélet egy egyszerű eszközöt ad a relációk „dekompozíciójának” tökéletesítésére: egy relációt helyettesítünk több másikkal, melyek összes attribútumának halmaza tartalmazza az eredeti reláció összes attribútumát.

3.1.1. A funkcionális függőség definíciója

A funkcionális függőség (FD¹) egy R reláción a következő formájú állítás: „Ha R két sora megegyezik az A_1, A_2, \dots, A_n attribútumokon (azaz ezen attribútumok mindenekélez megfeleltetett komponensnek ugyanaz az értéke a két sorban), akkor meg kell egyezniük más attribútumok egy B_1, B_2, \dots, B_m sorozatán.” Ezt a függőséget formálisan $A_1A_2 \dots A_n \rightarrow B_1B_2 \dots B_m$ -mel jelöljük, és azt mondjuk, hogy

„ A_1, A_2, \dots, A_n funkcionálisan meghatározza B_1, B_2, \dots, B_m -et”.

A 3.1. ábrán láthatjuk, mit jelent a funkcionális függőség az R reláció bármely két, t és u sorára. Ugyanakkor az A -k és a B -k bárhol elhelyezkedhetnek, nem szükséges hogy az A -k és a B -k egymást kövessék, vagy hogy az A -k megelőzzék a B jelűeket.



3.1. ábra. A funkcionális függőség két soron vett hatása

¹ FD – Functional Dependency, a funkcionális függőség angol nyelvű megfelelőjének rövidítése. (A lektor megjegyzése.)

Ha biztosak lehetünk abban, hogy az R reláció minden előfordulása olyan, amelyen az adott FD igaz, akkor azt mondhatjuk, hogy R kielégíti FD-t. Fontos megjegyezni, hogy amikor azt állítjuk, hogy R kielégíti az f FD-t, akkor ez egy megszorítást jelent R -re, nem csupán egy állítást R valamely egyedi előfordulásáról.

Gyakori, hogy a függőség jobb oldala egyetlen attribútum. Valójában az $A_1A_2 \dots A_n \rightarrow B_1B_2 \dots B_m$ függősségre úgy fogunk tekinteni, mint FD-k egy halmazára:

$$\begin{array}{l} A_1A_2 \dots A_n \rightarrow B_1 \\ A_1A_2 \dots A_n \rightarrow B_2 \\ \dots \\ A_1A_2 \dots A_n \rightarrow B_m \end{array}$$

filmcím	év	hossz	műfaj	stúdióNév	színészNév
Csillagok háborúja	1977	124	sci-fi	Fox	Carrie Fisher
Csillagok háborúja	1977	124	sci-fi	Fox	Mark Hamill
Csillagok háborúja	1977	124	sci-fi	Fox	Harrison Ford
Elfújta a szél	1939	231	dráma	MGM	Vivien Leigh
Wayne világa	1992	95	vígjáték	Paramount	Dana Carvey
Wayne világa	1992	95	vígjáték	Paramount	Mike Meyers

3.2. ábra. A Filmek1(filmcím, év, hossz, műfaj, stúdióNév, színészNév) reláció egy előfordulása

3.1. példa. Tekintsük a

`Filmek1(filmcím, év, hossz, műfaj, stúdióNév, színészNév)`

relációt és ennek a 3.2. ábrán található előfordulását. Kapcsolódunk a filmekről szóló példánkhöz, de ez a reláció tartalmaz további attribútumokat, így „Filmek” helyett „Filmek1”-nek nevezzük. Megjegyezzük, hogy ez a reláció túl sok minden akar egyszerre magába foglalni. Megpróbálja egyesíteni mindenzt, amit az eddigi példákban három külön relációban tároltunk: `Filmek`, `Stúdió` és `SzerepelBenne`. Ahogy látni fogjuk, a `Filmek1` reláció sémája nincs jól meghatározva. Ahhoz, hogy lássuk, miért rossz ez a terv, meg kell határoznunk a relációra érvényes funkcionális függőségi megszorításokat. Azt állítjuk, hogy a

`filmcím év → hossz műfaj stúdióNév`

FD fennáll.

Informálisan ez azt jelenti, hogy ha két sorban a `filmcím` mezőben ugyanazz az érték szerepel, és szintén megegyeznek az `év` attribútumon, akkor ezekben a sorokban azonos értéknek kell szerepelnie a `hossz`, `műfaj` és `stúdióNév` komponensekben is. Ezt az állítást úgy kell érteni, hogy nincs két olyan film, amely

ugyanabban az évben azonos címmel jelent meg (ugyanakkor lehetséges, hogy vannak azonos című filmek, amelyek különböző években jelentek meg). Erről a 2.1. példában volt szó. Tehát azt várjuk, hogy a filmcím és az év megadása egy egyedi filmet határoz meg. Azaz a filmnek egyedi hossza, műfaja és stúdiója van.

Másrészt vegyük észre, hogy a

$$\text{filmcím} \text{ } \text{év} \rightarrow \text{színészNév}$$

állítás hamis, ez a függőség nem érvényes a filmekre. Egy filmnek ugyanis több szereplője is lehet, így egy adott filmhez több színész is szerepelhet az adatbázisban. Jegyezzük meg, hogy ha lusták lettünk volna felsorolni több szereplőt a *Csillagok háborújához* és a *Wayne világához* (ahogyan azt tettük az *Elfújta a szél* esetében), ez a függőség akkor sem lenne azonnal igaz a *Filmek1* relációra. Ennek az az oka, hogy az FD a lehetséges előfordulásokról fogalmaz meg állitást, nem pedig az aktuális előfordulásról. Az a tény, hogy *lehetne* olyan előfordulásunk, amelyben több színész tartozhatna egy mozifilmhez, kizártja azt, hogy a *filmcím* és az *év* meghatározza a *színészNév* értékét. □

3.1.2. Relációk kulcsai

Azt mondjuk, hogy az egy vagy több attribútumból álló $\{A_1, A_2, \dots, A_n\}$ halmaz az *R* kulcsa, ha:

1. Ezek az attribútumok funkcionálisan meghatározzák a reláció minden más attribútumát, azaz nem lehet *R*-ben két olyan különböző sor, amely minden egyik A_1, A_2, \dots, A_n -nen megegyezne.
2. Nincs olyan valódi részhalmaza $\{A_1, A_2, \dots, A_n\}$ -nek, amely funkcionálisan meghatározná *R* összes többi attribútumát, azaz a kulcsnak *minimálisan* kell lennie.

Amikor csak egyetlenegy *A* attribútumból áll a kulcs, akkor gyakran azt mondjuk, hogy *A* (ahelyett hogy $\{A\}$) kulcs.

3.2. példa. A $\{\text{filmcím}, \text{év}, \text{színészNév}\}$ attribútumok a 3.2. ábrán szereplő *Filmek1* reláció egy kulcsát alkotják. Először azt kell megmutatnunk, hogy ezek az attribútumok funkcionálisan meghatározzák az összes többi attribútumot. Azaz vegyük két sort, amelyek megegyeznek a három attribútumon: *filmcím*, *év* és *színészNév*. Mivel ezek megegyeznek a *filmcím*-en és *év*-en, ezért meg kell egyezniük a *hossz*, *műfaj* és a *stúdióNév* attribútumokon, mint korábban, a 3.1. példában már láttuk. Tehát nincs két olyan különböző sor, amely minden *három* – *filmcím*, *év* és *színészNév* – attribútumon megegyezne, hiszen ez tulajdonképpen ugyanazt a sort jelentené.

Most azt kell megmutatnunk, hogy $\{\text{filmcím}, \text{év}, \text{színészNév}\}$ attribútumhalmaznak nincs olyan valódi részhalmaza, amely funkcionálisan meghatározná az összes többi attribútumot. Ahhoz, hogy ezt belássuk, először figyeljük meg,

Mitől „funkcionális” a funkcionális függőség?

$A_1 A_2 \cdots A_n \rightarrow B$ -t „funkcionális” függőségnek nevezzük, mivel elméletben van egy olyan függvény (funkció), ami olyan értéklistán van értelmezve, amely az A_1, A_2, \dots, A_n minden egyes attribútumához hozzárendel egy-egy értéket, és előállítja a B egyértelmű értékét (vagy egyáltalán semmilyen értéket). Például a Filmek1 relációhoz elképzelhetünk egy olyan függvényt, amely a "Csillagok háborúja" sztringhez és az 1977 egészhez rendeli hozzá a hossz egyértelmű értékét, nevezetesen a 124-et, ami a Filmek1 relációban megtalálható. Habár ez a függvény, mint leképezés, általános értelemben vett függvény, mégsem olyan, mint amivel a matematikában gyakran találkozunk, mivel nem tudjuk hogyan kiszámolni az értékét az argumentumértékekből. Azaz nincs olyan művelet, amelyet ha a "Csillagok háborúja" sztringen és az 1977 egészen végrehajtanánk, eredményül a helyes filmhosszt adná vissza. A függvény kiszámolása inkább csupán a relációból való visszakeresést jelenti. Megkeressük az adott filmcím-hez és év-hez tartozó sort és megnézzük, hogy milyen érték található a hossz-ra abban a sorban.

hogy a filmcím és év nem határozza meg a színészNév attribútumot, mivel több filmben is szerepel egynél több színész. Tehát a {filmcím, év} nem kulcs.

Az {év, színészNév} szintén nem kulcs, mivel egy színész két filmben is játszhat ugyanabban az évben, és így

$$\text{év színészNév} \rightarrow \text{filmcím}$$

nem teljesülő funkcionális függőség. Végül azt állítjuk, hogy {filmcím, színészNév} sem kulcs, hiszen két olyan filmnek, amelyeknek ugyanaz a címük, de más évben készültek, lehetséges, hogy van közös szereplője.² □

Előfordulhat, hogy a relációnak egynél több kulcsa van. Ha így van, akkor általában kijelöljük az egyik kulcsot mint elsődleges kulcsot. A kereskedelmi adatbázisrendszerknél az elsődleges kulcs megválasztása az implementációt is befolyásolhatja, például azt, hogyan tároljuk a relációt a lemezen. Ugyanakkor a funkcionális függőségek elmélete nem kezeli speciálisan az „elsődleges kulcsokat”.

² Egy korábbi könyvben erre nem tudtunk valós példát hozni, de azóta többen is megmutatták, hogy van ilyen. Érdekes kihívás olyan színészeket keresni, akik ugyanannak a filmnek kétféle változatában is szerepeltek.

Kulcsok eltérő terminológiája

Egyes cikkekben és könyvekben más szóhasználattal találkozhatunk a kulcsokra vonatkozóan. A „kulcs” (key) fogalmat ugyanis tágabb értelemben, a mi fogalmaink szerinti „szuperkulcsra” használják, azaz olyan attribútumhalmazt jelölnek vele, melytől funkcionálisan függ a séma összes attribútuma, és ez nem szükségszerűen minimális. Ebben az esetben a „minimális kulcs” (vagy „kulcsjelölt”, azaz „candidate key”) felel meg az általunk definiált „kulcsnak”.

3.1.3. Szuperkulcsok

Azokat az attribútumhalmazokat, amelyek tartalmaznak kulcsot, *szuperkulcsoknak* nevezzük, ez a rövidítése a „kulcsnál bővebb halmazknak” (kulcsok szuperhalmazának). Tehát minden kulcs egyben szuperkulcs is. Vannak azonban olyan szuperkulcsok, amelyek nem (minimális) kulcsok. Megjegyezzük, hogy minden szuperkulcs eleget tesz a kulcsdefiníció első feltételének: funkcionálisan meghatározzák a reláció összes többi attribútumát. A szuperkulcs azonban nem feltétlenül tesz eleget a második feltételnek, a minimalitásnak.

3.3. példa. A 3.2. példa relációjában több szuperkulcs van. Nemcsak a

$\{\text{filmcím, év, színészNév}\}$

kulcs szuperkulcs, hanem ennek az attribútumhalmaznak bármely hatványhalmaza is az, mint például

$\{\text{filmcím, év, színészNév, hossz, stúdióNév}\}$

is szuperkulcs. □

3.1.4. Feladatok

3.1.1. feladat. Tekintsünk egy relációt az Egyesült Államokban élő emberekről, amely tartalmazza a nevüket, a társadalombiztosítási számukat, a lakcím utcaját, városát, államát, irányítószámát, területi kódját és egy (7 számjegyű) telefonszámot. Milyen funkcionális függőségekről várhatsuk, hogy érvényesek? Melyek a reláció kulcsai? Ahhoz, hogy megválaszoljuk ezeket a kérdéseket, tudunk kell, hogyan történik ezeknek a számoknak a megadása. Például egy területi kód kiterjedhet-e két államra? Az irányítószám kiterjedhet-e két területi kódra? Lehet-e két embernek ugyanaz a társadalombiztosítási száma? Lehet-e ugyanaz a lakcímük vagy telefonszámuk?

3.1.2. feladat. Tekintsünk egy zárt konténerben található molekulák jelenlegi helyzetét leíró relációt. Az attribútumok a molekulaazonosító, a molekulák x , y és z koordinátái és a sebességek az x , y és z irányokban. Milyen funkcionális függőségekre várhatjuk, hogy érvényesek? Melyek a kulcsok?

!! **3.1.3. feladat.** Legyen R reláció, és A_1, A_2, \dots, A_n az attribútumai. Adjuk meg n függvényeként, hány szuperkulcsa van R -nek, ha:

- a) csak A_1 kulcs;
- b) csak A_1 és A_2 kulcsok;
- c) csak $\{A_1, A_2\}$ és $\{A_3, A_4\}$ kulcsok;
- d) csak $\{A_1, A_2\}$ és $\{A_1, A_3\}$ kulcsok.

3.2. Funkcionális függőségekre vonatkozó szabályok

Ebben a fejezetben azt tanuljuk meg, milyen *levezetési szabályok* érvényesek a funkcionális függőségekre. Tegyük fel, tudjuk, hogy a reláció minden függőségi halmazt elégít ki. Ebből gyakran vezethetünk le további függőségeket, melyeket a relációnak szintén ki kell elégítenie. Az a képesség, hogy fel tudjuk tárni a további függőségeket, a jó relációsémák tervezésénél válik majd jelentőssé a 3.3. alfejezetben.

3.2.1. Funkcionális függőségek levezetése

Kezdjük egy példával, amelyben megmutatjuk, hogyan lehet szabályokat levezetni más adott FD-k felhasználásával.

3.4. példa. Ha azt tudjuk, hogy az $R(A, B, C)$ reláció eleget tesz az $A \rightarrow B$ és a $B \rightarrow C$ funkcionális függőségeknek, akkor ebből levezethető, hogy R megfelel az $A \rightarrow C$ függőségnek is. Nézzük az érvelést! Ahhoz, hogy bebizonyítsuk $A \rightarrow C$ érvényességét, meg kell vizsgálnunk R bármely két olyan sorát, amelyek megegyeznek A -n, és be kell bizonyítanunk, hogy ezek megegyeznek C -n is.

Legyen az A attribútumon megegyező két sor (a, b_1, c_1) és (a, b_2, c_2) . Felte tessük, hogy a sorokban az attribútumok sorrendje A, B, C . Mivel R eleget tesz $A \rightarrow B$ -nek, és ezek a sorok megegyeznek A -n, így meg kell egyezniük B -n is. Azaz $b_1 = b_2$, és a sorok valójában (a, b, c_1) és (a, b, c_2) , ahol $b = b_1$ -et és b_2 -t is jelenti egyben. Hasonlóan: mivel R eleget tesz $B \rightarrow C$ -nek, és a sorok megegyeznek B -n, így megegyeznek C -n is. Azaz $c_1 = c_2$, tehát a sorok tényleg megegyeznek C -n, és így teljesül az $A \rightarrow C$ funkcionális függőség.³ □

³ A bizonyításból igazolódott az is, hogy az A, B és C attribútumokból álló R relációban A kulcs, tehát nem is lehet a két függőséget kielégítő relációban két olyan sor, amelyik A -ban megegyezik. A bizonyítás valójában indirekt volt. (A fordítói megjegyzése.)

A funkcionális függőségeket gyakran többféleképpen is megadhatjuk anélkül, hogy változna a reláció érvényes előfordulásainak a halmaza. Ilyenkor azt mondjuk, hogy:

- Két funkcionális függőségi halmaz, S és T *ekvivalensek*, ha S -et pontosan ugyanazok a reláció-előfordulások elégítik ki, mint amelyek T -t.
- Általánosabban azt mondjuk, hogy az S funkcionális függőségi halmaz *következik* a T funkcionális függőségi halmazból, ha minden olyan reláció-előfordulás, amely eleget tesz az összes T -beli függőségnak, eleget tesz minden S -beli függőségnak is.

A fentiek értelmében az S és T függőségi halmazok ekvivalensek, ha S következik T -ből, és T következik S -ből.

Ebben a fejezetben a funkcionális függőségekre vonatkozó több hasznos szabályt láthatunk. Általában ezekkel a szabályokkal valamely függőségi halmazt tudunk helyettesíteni egy, vele ekvivalens halmazzal, vagy kibővíteni az eredeti halmazból következő függőségekből álló halmazzal. Erre egy példa a *tranzitív szabály*, amellyel funkcionális függőségek láncolatait tudjuk követni, mint azt korábban a 3.4. példában láttuk. Megadunk egy algoritmust is, amelynek segítségével válaszolhatunk arra az általános kérdésre, hogy egy funkcionális függőség következik-e egy vagy több függőségből.

3.2.2. A szétvághatósági és összevonhatósági szabály

Idézzük fel, hogy az alábbi, a 3.1.1. alfejezetben már látott funkcionális függőség:

$$A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$$

ekvivalens az következő függőséghalmazzal:

$$A_1 A_2 \cdots A_n \rightarrow B_1, \quad A_1 A_2 \cdots A_n \rightarrow B_2, \dots, A_1 A_2 \cdots A_n \rightarrow B_m$$

Azaz szétvághatjuk a jobb oldalon szereplő attribútumokat úgy, hogy csak egy attribútum legyen mindegyik funkcionális függőség jobb oldalán. Hasonlóan tudjuk helyettesíteni azonos bal oldalú függőségek készletét egyetlen függőséggel, amelynek a bal oldala ugyanaz, és a jobb oldala pedig az összes jobb oldali attribútum egy attribútumhalmazzá való összevonása. Mindkét esetben az új függőségi halmaz ekvivalens a régivel. A fenti ekvivalencia kétféleképpen alkalmazható.

- Az $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ függőséget helyettesíthetjük az $A_1 A_2 \cdots A_n \rightarrow B_i$ ($i = 1, 2, \dots, m$) funkcionális függőségekből álló halmazzal. Ezt az átalakítást *szétvághatósági szabálynak* nevezzük.
- Az $A_1 A_2 \cdots A_n \rightarrow B_i$ ($i = 1, 2, \dots, m$) funkcionális függőségekből álló halmazt helyettesíthetjük az $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ egyetlen függőséggel. Ezt az átalakítást *összevonhatósági szabálynak* nevezzük.

3.5. példa. A 3.1. példában már látott függősséghalmaz:

$$\begin{aligned} \text{filmcím } \text{év} &\rightarrow \text{hossz} \\ \text{filmcím } \text{év} &\rightarrow \text{műfaj} \\ \text{filmcím } \text{év} &\rightarrow \text{stúdióNév} \end{aligned}$$

ekvivalens az alábbi egyetlen függősséggel:

$$\text{filmcím } \text{év} \rightarrow \text{hossz műfaj stúdióNév}$$

ahogyan azt már akkor is állítottuk. \square

A szétvághatósgági és összevonhatósági szabályok bizonyítása nyilvánvalónak tűnik. Tegyük fel, hogy van két sorunk, melyek az A_1, A_2, \dots, A_n attribútumokon megegyeznek. Egyetlen szabályban azt mondhatnánk, hogy „ezeknek a soroknak meg kell egyezniük a B_1, B_2, \dots, B_m attribútumokon”. Ehelyett különálló függősségekként azt állítjuk, hogy „ezek a sorok megegyeznek B_1 -en, és megegyeznek B_2 -n, …, és megegyeznek B_m -en.” Ez a két megfogalmazás pontosan ugyanazt jelenti.

Azt gondolhatnánk, hogy a szétvághatósgág a funkcionális függősségek bal oldalára ugyanúgy alkalmazható, mint a jobb oldalakra, azonban ez téves, ugyanis nincs bal oldalakra vonatkozó szétvághatósgági szabály, ahogyan azt a következő példában is láthatjuk:

3.6. példa. Tekintsük az alábbi függősséget, amely a 3.1. példa Filmek1 relációjára érvényes:

$$\text{filmcím } \text{év} \rightarrow \text{hossz}$$

Ha megpróbálnánk a bal oldalt szétvágni:

$$\begin{aligned} \text{filmcím} &\rightarrow \text{hossz} \\ \text{év} &\rightarrow \text{hossz} \end{aligned}$$

akkor hibás függősségeket kapnánk. Ugyanis a **filmcím** nem határozza meg funkcionálisan a **hossz**-t, mivel lehet két azonos című film (például *King Kong*), amelyeknek a hossza különbözik. Hasonlóan az **év** sem határozza meg funkcionálisan a **hossz**-t, hiszen biztosan több különböző hosszságú film készült ugyanabban az évben. \square

3.2.3. Triviális funkcionális függősségek

Egy megszorításra azt mondjuk, hogy *triviális*, ha fennáll a reláció minden előfordulására attól függetlenül, hogy milyen más megszorításokat fogalmazunk meg. Ha a megszorítások funkcionális függősségek, könnyen meghatározhatjuk, hogy egy FD triviális. Ezek azok az $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ függősségek, melyekre

$$\{B_1, B_2, \dots, B_m\} \subseteq \{A_1, A_2, \dots, A_n\}$$

Azaz triviális függősségről beszélhetünk, ha a jobb oldal a bal oldal részhalmaza. Például a

$$\text{filmcím } \text{év} \rightarrow \text{filmcím}$$

triviális függőség csakúgy, mint a

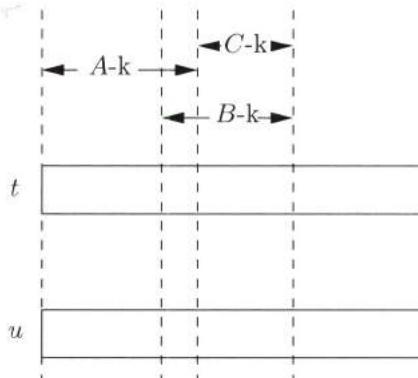
$$\text{filmcím} \rightarrow \text{filmcím}$$

Minden olyan triviális függőseg minden adatbázisban igaz, amely azt mondja, hogy „ha két sor megegyezik az A_1, A_2, \dots, A_n attribútumok mindegyikén, akkor ezek egy részhalmazán is megegyeznek”.

Abban az esetben, amikor néhány jobb oldali attribútum szerepel a függőseg bal oldalán, de nem az összes, akkor ez a függőseg nem triviális. De egyszerűsíthető úgy, hogy a jobb oldalról eltávolítjuk a bal oldalon előforduló attribútumokat. Azaz:

- $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ ekvivalens az $A_1A_2 \cdots A_n \rightarrow C_1C_2 \cdots C_k$ függősséggel, ahol C -kkel jelöltük azokat a B -ket, amelyek nem fordulnak elő az A -k között.

Ezt a szabályt, amelyet a 3.3. ábrán is láthatunk, *triviális függősségi szabálynak* nevezzük.

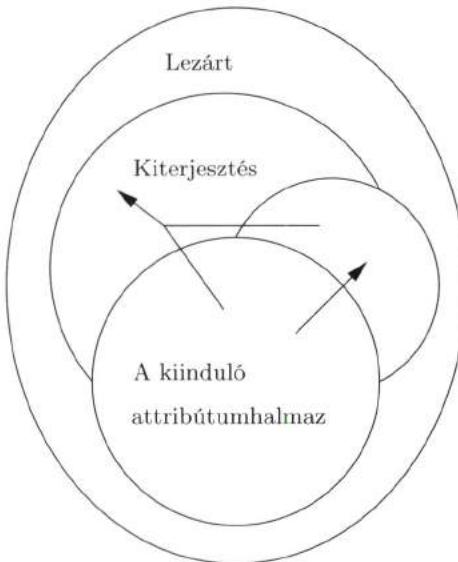


Ha t és u akkor meg kell
megegyezik egyezniük
az A -kon, tehát biztosan
megegyeznek a C -ken.

3.3. ábra. A triviális függősségi szabály

3.2.4. Attribútumhalmazok lezártjának kiszámítása

Mielőtt a többi szabállyal folytatnánk, megadunk egy általános elvet, amelyből az összes érvényes szabály következik. Legyen $\{A_1, A_2, \dots, A_n\}$ egy attribútumhalmaz, S pedig funkcionális függőségeknek egy halmaza. Az $\{A_1, A_2, \dots, A_n\}$ halmaz S -beli függőségek szerint vett lezártja azoknak a B attribútumoknak a halmaza, amelyekre minden olyan reláció, amely eleget tesz az összes S -beli függőségnak, eleget tesz $A_1 A_2 \dots A_n \rightarrow B$ -nek is. Azaz $A_1 A_2 \dots A_n \rightarrow B$ az S -beli függőségekből következik. Jelöljük az $A_1 A_2 \dots A_n$ attribútumhalmaz lezártját $\{A_1, A_2, \dots, A_n\}^+$ -szal. Ahhoz, hogy egyszerűsítsük a lezártak kiszámolásának a tárgyalását, megengedjük a triviális függőségeket, így A_1, A_2, \dots, A_n mindenbenne van $\{A_1, A_2, \dots, A_n\}^+$ -ban.



3.4. ábra. Egy attribútumhalmaz lezáráásának kiszámítása

A 3.4. ábra szemléletesen mutatja a lezáráás folyamatát. Kiindulunk az adott attribútumhalmazból, és többször ismételten növeljük ezt a halmazt azoknak a funkcionális függőségeknek a jobb oldali attribútumaival, amely függőségeknek a bal oldalát már tartalmazza az attribútumhalmaz. Nyilvánvalóan eljutunk egy pontig, amikor a halmaz már nem bővíthető tovább. Ez az eredményhalmaz lesz a lezárt.

3.7. algoritmus. Attribútumhalmaz lezártja

BEMENET: Egy $\{A_1, A_2, \dots, A_n\}$ attribútumhalmaz és funkcionális függőségek egy halmaza.

KIMENET: Az $\{A_1, A_2, \dots, A_n\}^+$ lezártja.

1. Ha szükséges, vágjuk szét a függőségeket úgy, hogy a jobb oldalukon egyetlen attribútum maradjon.
2. Legyen X az az attribútumhalmaz, amely végül a lezárt lesz. Inicializáláskor legyen $\{A_1, A_2, \dots, A_n\}$.
3. Ismételten keresünk olyan $B_1 B_2 \dots B_m \rightarrow C$ funkcionális függőséget S -ből, amelyre minden B_1, B_2, \dots, B_m benne van az X attribútumhalmazban, de a C nincs. Ekkor C -t hozzávesszük az X halmazhoz. Mivel X minden lépében csak nőhet, és minden reláció attribútumainak száma véges, tehát lesz olyan lépés, hogy X -hez már nem lehet elemet hozzávenni. Ekkor az eljárás befejeződik.
4. Az az X halmaz lesz $\{A_1, A_2, \dots, A_n\}^+$ -nak a helyes értéke, amelyet már nem tudunk tovább bővíteni.

□

3.8. példa. Tekintsünk egy relációt, amelynek attribútumai A, B, C, D, E és F . Legyenek ehhez a relációhoz tartozó funkcionális függőségek $AB \rightarrow C$, $BC \rightarrow AD$, $D \rightarrow E$ és $CF \rightarrow B$. Mi az $\{A, B\}$ lezártja, azaz $\{A, B\}^+?$

Először vágjuk szét a $BC \rightarrow AD$ függőséget $BC \rightarrow A$ és $BC \rightarrow D$ függőségekre. Induljunk ki $X = \{A, B\}$ -ből. Figyeljük meg, hogy az $AB \rightarrow C$ funkcionális függőség bal oldalán szereplő minden attribútum benne van az X -ben, így hozzávehetjük a függőség jobb oldali attribútumát, azaz C -t X -hez. Tehát a 3. lépés első ciklusában X egyenlő lesz $\{A, B, C\}$ -vel.

Ezután látjuk, hogy $BC \rightarrow A$ és $BC \rightarrow D$ bal oldalai benne vannak az X -ben, így hozzávehetjük az A és D attribútumokat X -hez. A már benne van, D nincs, ezzel bővíjtük X -et, így az egyenlő lesz $\{A, B, C, D\}$ -vel. Most a $D \rightarrow E$ függőséget használva E -vel bővíjtük X -et, így az $\{A, B, C, D, E\}$ -re bővül. Ezután X -en már nem tudunk tovább változtatni. Ebben a példában a $CF \rightarrow B$ funkcionális függőséget nem vettük figyelembe, mivel X soha nem tartalmazta a függőség bal oldalát. Tehát $\{A, B\}^+ = \{A, B, C, D, E\}$. □

Attribútumhalmazok lezártjának előállításával el tudjuk dönteni, hogy egy $A_1 A_2 \dots A_n \rightarrow B$ függőség következik-e egy funkcionális függőséget tartalmazó S halmazból. Először számoljuk ki az $\{A_1, A_2, \dots, A_n\}$ halmaz lezártját az S -beli szabályok felhasználásával. Ha B benne van $\{A_1, A_2, \dots, A_n\}^+$ -ban, akkor $A_1 A_2 \dots A_n \rightarrow B$ következik S -ből, ha B nincs benne $\{A_1, A_2, \dots, A_n\}^+$ -ban, akkor ez az FD nem következik S -ből. Általánosabban: $A_1 A_2 \dots A_n \rightarrow B_1 B_2 \dots B_m$ következik egy S szabályhalmazból, akkor és csak akkor, ha B_1, B_2, \dots, B_m benne van az $\{A_1, A_2, \dots, A_n\}^+$ halmazban.

3.9. példa. Vegyük a 3.8. példában szereplő relációt és funkcionális függőségeket. Tegyük fel, hogy ellenőrizni szeretnénk, vajon az $AB \rightarrow D$ következik-e ezekből a függőségekből. Számoljuk ki $\{A, B\}^+$ -t, amely $\{A, B, C, D, E\}$, ahogyan az előző példában láttuk. Mivel D eleme a lezártnak, ezért azt kapjuk,

hogy $AB \rightarrow D$ következik az adott függőségekből.

Másrészt vegyük a $D \rightarrow A$ funkcionális függőséget. Ahhoz, hogy ellenőrizzük, vajon ez a függőség következik-e az adott függőségekből, először számoljuk ki $\{D\}^+$ -t. Ehhez kezdjük $X = \{D\}$ -vel. Felhasználhatjuk a $D \rightarrow E$ függőséget, hogy az X halmazt bővítsük E -vel. Ezután leragadtunk. Nem tudunk találni más függőséget, amelynek a bal oldalát X tartalmazná, így $\{D\}^+ = \{D, E\}$. Mivel A nem eleme $\{D, E\}$ -nek, azt kapjuk, hogy $D \rightarrow A$ nem következik. \square

3.2.5. Miért működik a lezárási algoritmus?

Ebben az alfejezetben megmutatjuk, hogy a 3.7. algoritmus miért tudja helyesen eldönteni, hogy egy $A_1 A_2 \cdots A_n \rightarrow B$ funkcionális függőség következik-e avagy sem az adott S funkcionális függőségi halmazból. A bizonyítás két részből áll:

1. Bizonyítanunk kell, hogy a 3.7. algoritmus nem követel túl sokat, azaz meg kell mutatnunk, hogy ha $A_1 A_2 \cdots A_n \rightarrow B$ a lezárási próba alapján bekerült (vagyis B benne van $\{A_1, A_2, \dots, A_n\}^+$ -ban), akkor $A_1 A_2 \cdots A_n \rightarrow B$ fennáll bármilyen olyan reláció esetén, amely S összes funkcionális függőségét kielégíti.
2. Be kell látnunk, hogy a lezárási algoritmus nem fog kihagyni egyetlen funkcionális függőséget sem, ami tényleg következne az S funkcionális függőségi halmazból.

Miért csak az igaz funkcionális függőségeket fogadja el a lezárási algoritmus?

Indukcióval bizonyíthatjuk be – aszerint, hogy az X -ben szereplő egyes D attribútumokra a harmadik lépés bővítési műveletét hányszor kellett alkalmazzuk –, hogy az $A_1 A_2 \cdots A_n \rightarrow D$ funkcionális függőség teljesül. Eszerint minden olyan R reláció, amely eleget tesz az összes S -beli függőségnek, eleget tesz az $A_1 A_2 \cdots A_n \rightarrow D$ -nek is.

KIINDULÁS: Az indukció 0. lépése. Ekkor D csak az A_1, A_2, \dots, A_n közül lehet az egyik, és az $A_1 A_2 \cdots A_n \rightarrow D$ funkcionális függőség biztosan fennáll tetszőleges relációban, mivel ez triviális függőség.

INDUKCIÓ: Az indukcióhoz tegyük fel, hogy D -vel akkor bővítünk, amikor felhasználjuk a $B_1 B_2 \cdots B_m \rightarrow D$ függőséget az S halmazból. Az indukciós feltétel miatt R eleget tesz az $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ függőségnek. Tekintsünk két R -beli sort, melyek megegyeznek az összes A_1, A_2, \dots, A_n -en. Mivel R megfelel az $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ függőségnek, a két sornak meg kell egyeznie az összes B_1, B_2, \dots, B_m -en is. Mivel R eleget tesz $B_1 B_2 \cdots B_m \rightarrow D$ -nek, azt is tudjuk, hogy ez a két sor megegyezik D -n. Tehát R eleget tesz $A_1 A_2 \cdots A_n \rightarrow D$ -nek is.

Miért talál meg minden igaz funkcionális függőséget a lezárási algoritmus?

Tegyük fel, hogy $A_1A_2 \cdots A_n \rightarrow B$ volt az a függőség, amely nem következik S halmazból a 3.7. lezárási algoritmus szerint. Azaz S funkcionális függőségi halmazát felhasználva $\{A_1, A_2, \dots, A_n\}$ halmaz lezárása nem tartalmazza B -t. Be kell látnunk, hogy $A_1A_2 \cdots A_n \rightarrow B$ valójában nem következik S -ből. Tehát mutatnunk kell legalább egy olyan relációt, amelyre S minden funkcionális függősége teljesül, és még így sem teljesíti $A_1A_2 \cdots A_n \rightarrow B$ -t.

Egy ilyen I előfordulást viszonylag könnyű készíteni. A 3.5. ábrán is egy ilyet láthatunk. Az I -nek csak két sora van: t és s . A két sor $\{A_1, A_2, \dots, A_n\}^+$ minden attribútumában megegyezik, de a többiben eltérnek egymástól. Először meg kell mutatnunk, hogy I kielégíti S összes függőségét, és azt, hogy $A_1A_2 \cdots A_n \rightarrow B$ -t pedig nem elégíti ki.

	$\{A_1, A_2, \dots, A_n\}^+$	Más attribútumok
$t:$	1 1 1 … 1 1	0 0 0 … 0 0
$s:$	1 1 1 … 1 1	1 1 1 … 1 1

3.5. ábra. Egy I előfordulás, amely kielégíti S -et, de $A_1A_2 \cdots A_n \rightarrow B$ -t nem

Tegyük fel, hogy létezik néhány $C_1C_2 \cdots C_k \rightarrow D$ függőség S -ben (a szabályok szétvágása után), amelyeket I nem elégít ki. Mivel I -nek összesen két sora van, s és t , így ezek azok a sorok, amelyek megsértik a $C_1C_2 \cdots C_k \rightarrow D$ szabályt. Ez azt jelenti, hogy s és t megegyeznek a $\{C_1, C_2, \dots, C_k\}$ attribútumokon, de eltérnek D -n. A 3.5. ábra alapján látható, hogy C_1, C_2, \dots, C_k az $\{A_1, A_2, \dots, A_n\}^+$ halmazban vannak, mert csak olyan attribútumok, amelyeken t és s megegyezik. Hasonlóan D a másik attribútumhalmazba tartozik, mert t és s azokon az attribútumokon különbözik.

Ekkor viszont nem jól számítottuk ki a lezárást. $C_1C_2 \cdots C_k \rightarrow D$ -t fel kellett volna használnunk, hogy D -t X -hez adjuk, amikor $X = \{A_1, A_2, \dots, A_n\}$ volt. Így arra következtethetünk, hogy $C_1C_2 \cdots C_k \rightarrow D$ nem létezhet, vagyis az I előfordulása kielégíti S -et.

Másrészt be kell látnunk, hogy I nem elégíti ki $A_1A_2 \cdots A_n \rightarrow B$ -t. Ez a rész viszont egyszerű. A_1, A_2, \dots, A_n valójában azok az attribútumok, amelyeken t és s megegyeznek. Azt is tudjuk még, hogy B nincs benne $\{A_1, A_2, \dots, A_n\}^+$ -ban, hiszen B olyan attribútumokat jelent, ahol t és s eltérnek, vagyis I nem elégíti ki $A_1A_2 \cdots A_n \rightarrow B$ -t. Így levonhatjuk azt a következtetést, hogy a 3.7. lezárási algoritmus nem állítja sem több, sem kevesebb funkcionális függőségek a teljesülését, csak pontosan azokat a függőségeket állítja igaznak, amelyek S -ből következnek.

Lezáráskulcsok és kulcsok

Megjegyezzük, hogy $\{A_1, A_2, \dots, A_n\}^+$ akkor és csak akkor az összes attribútumból álló halmaz, ha A_1, A_2, \dots, A_n a szóban forgó reláció szuperkulcsa. Csak ekkor határozza meg funkcionálisan A_1, A_2, \dots, A_n az összes attribútumot. Ezzel ellenőrizni tudjuk, hogy A_1, A_2, \dots, A_n a reláció kulcsa-e: először ellenőrizzük, hogy $\{A_1, A_2, \dots, A_n\}^+$ tartalmazza-e az összes attribútumot, majd ellenőrizzük, hogy nincs olyan X halmaz, amelyet $\{A_1, A_2, \dots, A_n\}$ -ból kapnánk az egyik attribútum törlésével, és amelyre X^+ az összes attribútumot tartalmazná.

3.2.6. A tranzitivitási szabály

A tranzitivitási szabállyal két funkcionális szabályt kapcsolhatunk össze, általánosítva a 3.4. példa megállapítását:

- Ha $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ és $B_1 B_2 \cdots B_m \rightarrow C_1 C_2 \cdots C_k$ teljesül az R relációban, akkor $A_1 A_2 \cdots A_n \rightarrow C_1 C_2 \cdots C_k$ szintén teljesül R -ben.)

Ha a C -k között vannak olyanok, amely az A -k között is megtalálhatók, azokat a triviális függőségi szabállyal kiküszöbölhetjük a jobb oldalról.

Ahhoz, hogy belássuk, miért teljesül a tranzitivitási szabály, felhasználjuk a 3.2.4. alfejezet ellenőrzési algoritmusát. Ahhoz, hogy ellenőrizzük, $A_1 A_2 \cdots A_n \rightarrow C_1 C_2 \cdots C_k$ teljesül-e, ki kell számolnunk $\{A_1, A_2, \dots, A_n\}^+ \rightarrow \{C_1, C_2, \dots, C_k\}$ megadott függőségekre.

Az $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ funkcionális függőség miatt az összes B_1, B_2, \dots, B_m benne van $\{A_1, A_2, \dots, A_n\}^+$ -ban. Ezután alkalmazhatjuk a $B_1 B_2 \cdots B_m \rightarrow C_1 C_2 \cdots C_k$ függőséget, hogy C_1, C_2, \dots, C_k -t hozzávegyük $\{A_1, A_2, \dots, A_n\}^+$ -hoz. Mivel az összes C benne van $\{A_1, A_2, \dots, A_n\}^+$ -ban, megkapjuk, hogy $A_1 A_2 \cdots A_n \rightarrow C_1 C_2 \cdots C_k$ fennáll minden olyan relációban, amely eleget tesz mind az $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$, mind a $B_1 B_2 \cdots B_m \rightarrow C_1 C_2 \cdots C_k$ függőségeknek.

3.10. példa. Az alábbi példában a Filmek reláció egy újabb verzióját láthatjuk, amely tartalmazza a filmstúdiót és információkat is róla.

filmcím	év	hossz	műfaj	stúdióNév	stúdióCím
Csillagok háborúja	1977	124	sci-fi	Fox	Hollywood
Kutyahideg	2005	120	dráma	Disney	Buena Vista
Wayne világa	1992	95	vígjáték	Paramount	Hollywood

Feltehetjük, hogy fennáll az alábbi két függőség:

$\text{filmcím } \text{év} \rightarrow \text{stúdióNév}$

$\text{stúdióNév} \rightarrow \text{stúdióCím}$

Az első azért indokolt, mert csak egy film lehet adott címmel egy évben, és csak egy stúdióhoz tartozhat egy adott film. A másodikat az igazolja, hogy a stúdióknak egyértelmű címük van.

A tranzitív szabállyal a fenti két függőség kombinálásával kapunk egy új függőséget:

$$\text{filmcím év} \rightarrow \text{stúdióCím}$$

Ez a függőség azt jelenti, hogy a filmcím és az év (azaz a film) meghatározza a címet – a filmet gyártó stúdió címét. \square

3.2.7. Funkcionális függőségi halmazok lezárása

Előfordulhat, hogy választhatunk, milyen funkcionális függőségeket használunk, amelyek reprezentálják egy reláció teljes FD-halmazát. Ha megadunk egy S függőséghalmazt (olyan függőségekből, amelyek fennállnak a relációban), akkor bármely S -sel ekvivalens függőséghalmazt S bázisának nevezzük. Hogy elkerüljük a lehetséges halmazok számának megugrását, olyan függőségek vizsgálatára szorítkozunk, amelyeknek a jobb oldalán egyetlen attribútum szerepel. Ha van egy bázisunk, akkor a szétvágási szabályt alkalmazva a jobb oldalak ilyen alakúra hozhatók. A *minimális bázis* egy relációhoz az a B bázis, amely az alábbi három feltételt elégíti ki:

1. B összes függőségének jobb oldalán egy attribútum van.
2. Ha bármelyik B -beli függőséget elhagyjuk, a fennmaradó halmaz már nem bázis.
3. Ha bármelyik B -beli funkcionális függőség bal oldaláról elhagyunk egy vagy több attribútumot, akkor az eredmény már nem marad bázis.

Jegyezzük meg, hogy egyetlen triviális funkcionális függőség sem lehet a minimális bázisban, mert azt a 2. szabály szerint eltávolíthatjuk.

3.11. példa. Tekintsük az $R(A, B, C)$ relációt, amelynek minden attribútumától funkcionálisan függ a másik két attribútuma. A származtatott függőségek teljes halmaza hat függőséget tartalmaz, egyetlen attribútummal a jobb oldalon: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow A$, $B \rightarrow C$, $C \rightarrow A$ és $C \rightarrow B$. Ezenkívül van három nem triviális funkcionális függőség, két attribútummal a bal oldalon: $AB \rightarrow C$, $AC \rightarrow B$ és $BC \rightarrow A$. Továbbá vannak olyan függőségek, amelyeknek több attribútum van a jobb oldalán, például $A \rightarrow BC$, vagy triviális függőségek, mint például $A \rightarrow A$.

Az R relációnak számos minimális bázisa van. Ezek közül az egyik:

$$\{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B\}$$

Egy másik az $\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$. Ezeken kívül még több minimális bázisa van R -nek, amelyek megkeresését az olvasóra bízzuk. \square

A levezetési szabályok egy teljes halmaza

Ha meg akarjuk tudni, hogy egy funkcionális függőség következik-e néhány adott függőségből, a 3.2.4. alfejezetben szereplő lezárás kiszámolását minden használhatjuk. Nem árt tudni, hogy létezik a szabályoknak egy olyan halmaza, az úgynevezett *Armstrong-axiómák*, amelyek segítségével egy adott halmazból következő bármely funkcionális függőség levezethető. Ezek az axiómák:

1. *Reflexivitás.* Ha $\{B_1, B_2, \dots, B_m\} \subseteq \{A_1, A_2, \dots, A_n\}$, akkor $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$. Ezeket neveztük triviális függőségeknak.
2. *Bővítés.* Ha $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$, akkor

$$A_1 A_2 \cdots A_n C_1 C_2 \cdots C_k \rightarrow B_1 B_2 \cdots B_m C_1 C_2 \cdots C_k$$

bármely $\{C_1, C_2, \dots, C_k\}$ attribútumhalmazra. Mivel tudjuk, hogy a C -k lehetnek A -k és B -k is, a bal oldalról ki kell szűrnünk a duplikált attribútumokat, majd ugyanezt kell tennünk a jobb oldalakkal is.

3. *Tranzitivitás.* Ha

$$A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m \text{ és } B_1 B_2 \cdots B_m \rightarrow C_1 C_2 \cdots C_k,$$

akkor $A_1 A_2 \cdots A_n \rightarrow C_1 C_2 \cdots C_k$.

3.2.8. Funkcionális függőségek vetítése

Egy relációséma tervének vizsgálatánál a következő, funkcionális függőségekkel kapcsolatos kérdéseket is meg kell válaszolnunk. Tegyük fel, hogy adott az R reláció egy S funkcionális függőséghalmazzal, ahol elkészítjük R egy vetítését: $R_1 = \pi_L(R)$ R néhány attribútumára. Mely függőségek állnak fenn R_1 -ben?

A választ elvileg az S -beli funkcionális függőségek vetületének kiszámításával nyerjük, mely függőségek:

- a) S -ből levezethetők és
- b) csak R_1 attribútumait tartalmazzák.

Mivel nagyon nagy számú ilyen funkcionális függőség lehet, és ezek közül sok redundáns is (azaz más ugyanilyen típusú függőségekből következik), a függősséghalmazt szabadon egyszerűsíthetjük. Általában viszont az R_1 funkcionális függőségeinek kiszámítási ideje a legrosszabb esetben az R_1 -beli attribútumok

számának exponenciális függvénye. Az alábbiakban egy egyszerű algoritmus összefoglalása látható.

3.12. algoritmus. Funkcionális függőséghalmaz vetületének kiszámítása

BEMENET: Egy R reláció, és egy másik R_1 reláció, amelyet az $R_1 = \pi_L(R)$ vetítéssel számítottunk ki. Továbbá adott egy S függőséghalmaz, mely fennáll R -ben.

KIMENET: Az R_1 -ben fennálló funkcionális függőségek halmaza.

ELJÁRÁS:

1. Legyen T a végül előálló funkcionális függőségek halmaza. Kezdetben T üres.
2. minden olyan X attribútumhalmazra, amely R_1 -nek része, számítsuk ki X^+ -t. Ezt a kiszámítást az S -beli funkcionális függőségeket figyelembe véve végezzük, és előfordulhatnak olyan attribútumok is, amelyek R -ben szerepelnek, de R_1 -ben nem. Adjuk hozzá T -hez az összes nem triviális függőséget, amelyek $X \rightarrow A$ formátumúak, ahol A eleme az X^+ és az R_1 attribútumhalmaznak is.
3. Ezután a kapott T bázisa az R_1 -beli funkcionális függőségeknek, de nem feltétlen minimális bázis. A minimális bázist előállíthatjuk T -ből az alábbi módosítások végrehajtásával:
 - a) Ha szerepel egy F funkcionális függőség T -ben, amely más T -beli függőségekből következik, akkor töröljük F -et a T halmazból.
 - b) Legyen $Y \rightarrow B$ egy T -beli funkcionális függőség, ahol Y legalább két attribútumot tartalmaz, és legyen Z az a halmaz, amelyet úgy kapunk, hogy Y -ból egy attribútumot elhagyunk. Ha $Z \rightarrow B$ függőség következik a T -beli funkcionális függőségekből (beleértve $Y \rightarrow B$ -t is), akkor cseréljük ki $Y \rightarrow B$ -t $Z \rightarrow B$ -re.
 - c) Ismételjük az előző lépéseket addig, amíg már nem lehet semmilyen módosítást végrehajtani T -n.

□

3.13. példa. Vegyük az $A \rightarrow B$, $B \rightarrow C$ és $C \rightarrow D$ funkcionális függőségekkel rendelkező $R(A, B, C, D)$ relációt. Tegyük fel azt is, hogy a B attribútumot akarjuk elhagyni, hogy vetítéssel egy $R_1(A, C, D)$ relációt kapunk. Elviekben az R_1 funkcionális függőségeinek megtalálásához vennünk kell az $\{A, C, D\}$ minden nyolc részhalmazának a lezártját az összes funkcionális függőség figyelembe-vételével (beleértve a B -t érintőket is). Van viszont néhány nyilvánvaló egyszerűsítési lehetőség, amelyeket elvégezhetünk.

- Az üres, illetve a minden attribútumot tartalmazó halmaz lezárása nem vezet nem triviális funkcionális függőséghez.

- Ha tudjuk, hogy egy X halmaz lezárása tartalmazza az összes attribútumot, akkor nem tudunk újabb funkcionális függőséghoz jutni az X szuperhalmazainak lezáráásával.

Azaz kezdhetünk az egyértékű halmazok lezártjával, majd utána folytatjuk a kétértékű halmazokkal, ha szükséges. minden X halmaz lezártjára hozzá tesszük az $X \rightarrow E$ funkcionális függőséget az R_1 séma minden olyan E attribútumára, amely X^+ -ban benne van, de nincs benne X -ben.

Először $\{A\}^+ = \{A, B, C, D\}$, azaz $A \rightarrow C$ és $A \rightarrow D$ fennáll R_1 -ben. Figyeljük meg, hogy $A \rightarrow B$ fennáll R -ben, viszont R_1 -ben nem értelmezhető, hiszen B nem attribútuma R_1 -nek.

Majd tekintsük $\{C\}^+ = \{C, D\}$ -t, amelyből a $C \rightarrow D$ függőséget kapjuk. Mivel $\{D\}^+ = \{D\}$, nem tudunk további FD-ket felvenni, és az egyelemű halmazokkal végeztünk.

Mivel $\{A\}^+$ tartalmazza R_1 minden attribútumát, ezért nem érdemes $\{A\}$ szuperhalmazával sem foglalkozni. Ennek oka a megtalálható funkcionális függőségek típusa. Például az $AC \rightarrow D$ függőség következik abból a függőségből, amelynek bal oldalán csak A szerepel, azaz most $A \rightarrow D$ -ből. Az egyetlen két-elemű halmaz, amelyet meg kell vizsgálni, a $\{C, D\}^+ = \{C, D\}$ halmaz. Ez a vizsgálat sem nyújt semmi újat. A lezártak előállításával végeztünk, a megtalált funkcionális függőségek: $A \rightarrow C$, $A \rightarrow D$ és $C \rightarrow D$.

Ha szeretnénk, akkor azt is megfigyelhetjük, hogy $A \rightarrow D$ a másik két szabályból a tranzitivitással már következik. Ezért egy egyszerűbb, ekvivalens funkcionális függőségi halmazt kapunk R_1 -re, amelyben $A \rightarrow C$ és $C \rightarrow D$. Ez a halmaz valójában az R_1 -beli funkcionális függőségek minimális bázisa. \square

3.2.9. Feladatok

3.2.1. feladat. Tekintsünk egy relációt $R(A, B, C, D)$ sémával és $AB \rightarrow C$, $C \rightarrow D$ és $D \rightarrow A$ funkcionális függőségekkel.

- Melyek az összes nem triviális függőségek, amelyek az adott függőségekből következnek? Csak olyan függőségekkel foglalkozzunk, amelyek jobb oldala egy attribútumból áll.
- Melyek az R összes kulcsai?
- Melyek az R összes olyan szuperkulcsai, amelyek nem kulcsok?

3.2.2. feladat. Ismételjük meg a 3.2.1. feladatot az alábbi sémákra és függőségekre:

- $S(A, B, C, D)$, az $A \rightarrow B$, $B \rightarrow C$ és $B \rightarrow D$ funkcionális függőségekkel.
- $T(A, B, C, D)$, az $AB \rightarrow C$, $BC \rightarrow D$, $CD \rightarrow A$ és $AD \rightarrow B$ funkcionális függőségekkel.

- iii) $U(A, B, C, D)$, az $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$ és $D \rightarrow A$ funkcionális függőségekkel.

3.2.3. feladat. Mutassuk meg, hogy az alábbi szabályok érvényesek, használva a 3.2.4. alfejezet lezárási algoritmusát.

- a) *Bővítés a bal oldalon.* Ha $A_1A_2 \cdots A_n \rightarrow B$ funkcionális függőség és C egy másik attribútum, akkor $A_1A_2 \cdots A_nC \rightarrow B$ fennáll.
- b) *Teljes bővítés.* Ha $A_1A_2 \cdots A_n \rightarrow B$ funkcionális függőség és C egy másik attribútum, akkor $A_1A_2 \cdots A_nC \rightarrow BC$ fennáll. Megjegyezzük, hogy ebből a szabályból könnyen belátható a „bővítési” szabály, amit a 3.2.7. alfejezet keretes írásában („A vezetési szabályok egy teljes halmaza”) állítottunk.
- c) *Pszeudotranzitivitás.* Tegyük fel, hogy $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ és $C_1C_2 \cdots C_k \rightarrow D$ funkcionális függőségek fennállnak, és B -k mindegyike a C -k között előfordul. Ekkor

$$A_1A_2 \cdots A_nE_1E_2 \cdots E_j \rightarrow D$$

fennáll, ahol E -k éppen azon C -k közül valók, amelyek nincsenek B -k között.

- d) *Additivitás.* Ha

$$A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m \quad \text{és} \quad C_1C_2 \cdots C_k \rightarrow D_1D_2 \cdots D_j$$

funkcionális függőségek fennállnak, akkor

$$A_1A_2 \cdots A_nC_1C_2 \cdots C_k \rightarrow B_1B_2 \cdots B_mD_1D_2 \cdots D_j$$

is fennáll. Ebben a függősségen ki kellene törölünk azoknak az attribútumoknak az egyik másolatát, amelyek mind A -k és C -k, valamint B -k és D -k között előfordulnak.

! 3.2.4. feladat. Mutassuk meg, hogy az alábbiak nem érvényes szabályok a funkcionális függősségekre, úgy, hogy adjunk példát olyan relációra, amely eleget tesz a „Ha...” részben adott függősségeknek, de nem tesz eleget az állítás szerinti következménynek.

- a) Ha $A \rightarrow B$, akkor $B \rightarrow A$.
- b) Ha $AB \rightarrow C$ és $A \rightarrow C$, akkor $B \rightarrow C$.
- c) Ha $AB \rightarrow C$, akkor $A \rightarrow C$ vagy $B \rightarrow C$.

! 3.2.5. feladat. Mutassuk meg, hogy ha egy relációt nincs olyan attribútuma, amelyet funkcionálisan meghatározna az összes többi attribútum, akkor a relációt egyáltalán nincs nem triviális függőisége.

! 3.2.6. feladat. Legyenek X és Y attribútumokból álló halmazok. Mutassuk meg, hogy ha $X \subseteq Y$, akkor $X^+ \subseteq Y^+$, ahol a lezárásokat ugyanarra a funkcionális függőségekből álló halmazra számoljuk.

! 3.2.7. feladat. Bizonyítsuk be, hogy $(X^+)^+ = X^+$.

!! 3.2.8. feladat. Azt mondjuk, hogy X attribútumhalmaz *zárt* (a funkcionális függőségek egy adott halmazára nézve), ha $X^+ = X$. Tekintsünk egy $R(A, B, C, D)$ sémájú relációt és funkcionális függőségek egy ismeretlen halmazát. Ha tudjuk, mely attribútumhalmazok zártak, fel tudjuk tárni a funkcionális függőségeket. Melyek a funkcionális függőségek, ha:

- a négy attribútum összes részhalmaza zárt;
- a zárt halmazok csak az \emptyset és az $\{A, B, C, D\}$;
- a zárt halmazok az \emptyset , az $\{A, B\}$ és az $\{A, B, C, D\}$.

! 3.2.9. feladat. Keressük meg a 3.11. példában szereplő relációhoz és függősségekhez az összes minimális bázist.

! 3.2.10. feladat. Tegyük fel, hogy adott egy $R(A, B, C, D, E)$ relációt funkcionális függőségekkel, melyeket vetíténi szeretnénk az $S(A, B, C)$ relációra. Adjuk meg az S -ben érvényes függőségeket, ha R -ben a következők érvényesek:

- $AB \rightarrow DE, C \rightarrow E, D \rightarrow C$ és $E \rightarrow A$.
- $A \rightarrow D, BD \rightarrow E, AC \rightarrow E$ és $DE \rightarrow B$.
- $AB \rightarrow D, AC \rightarrow E, BC \rightarrow D, D \rightarrow A$ és $E \rightarrow B$.
- $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$ és $E \rightarrow A$.

Mind a négy esetben adjuk meg az S -ben érvényes függőségek egy minimális bázisát.

!! 3.2.11. feladat. Mutassuk meg, hogy ha F funkcionális függőség következik adott funkcionális függőségekből, akkor F -et be tudjuk bizonyítani az adott függőségekből az Armstrong-axiómák (lásd a 3.2.7. alfejezet „A levezetési szabályok egy teljes halmaza” című keretes frását) segítségével. **Útmutatás:** Vizsgáljuk meg az attribútumhalmaz lezárását előállító 3.7. algoritmust, és lássuk be, hogy az algoritmus minden lépését helyettesíthetjük az Armstrong-axiómákból következő funkcionális függőségekkel.

3.3. Relációs adatbázissémák tervezése

Egy relációs adatbázisséma nem kellően gondos megtervezése redundanciához és anomáliákhoz vezet. Vizsgáljuk meg például a 3.2. ábrán látható relációt, amelyet most a 3.6. ábraként idemásolunk. Megjegyezzük, hogy a *Csillagok háborúja* és a *Wayne világa* filmek hossz és műfaj attribútumai minden egyes színész sorában megismétlődnek. Az információ ilyen megismétlése redundanciához vezet. Ahogy látni fogjuk, ez további hibalehetőségekhez vezet.

Ebben a fejezetben a jó relációsémák tervezésének a problémáit a következő lépésekben tárgyaljuk:

1. Először részletesebben feltárjuk, milyen problémák merülhetnek fel hibás séma esetén.
2. Ezután bevezetjük a „felbontás” (dekompozíció) ötletét, mellyel a relációsémát (attribútumok halmozát) széttördeljük kisebb sémákra.
3. A következő lépében bevezetjük a „Boyce–Codd normálformát” vagy „BCNF”-et, amely a relációsémára jelent olyan feltételeket, amellyel kiküszöböljük ezeket a problémákat.
4. Ezután összekapcsoljuk a két téma kört, és megnézzük, hogyan biztosíthatjuk a BCNF feltételét a relációsémák felbontásával.

filmcím	év	hossz	műfaj	stúdióNév	színészNév
Csillagok háborúja	1977	124	sci-fi	Fox	Carrie Fisher
Csillagok háborúja	1977	124	sci-fi	Fox	Mark Hamill
Csillagok háborúja	1977	124	sci-fi	Fox	Harrison Ford
Elfújta a szél	1939	231	dráma	MGM	Vivien Leigh
Wayne világa	1992	95	vígjáték	Paramount	Dana Carvey
Wayne világa	1992	95	vígjáték	Paramount	Mike Meyers

3.6. Ábra. A Filmek1 reláció az anomáliák bemutatására

3.3.1. Anomáliák

Azokat a problémákat, amelyek a redundanciához hasonlóan akkor fordulnak elő, amikor túl sok információt próbálunk egyetlenegy relációba belegyömöszölni, anomáliáknak nevezzük. A számításba jövő anomáliák alapvető fajtái a következők:

1. *Redundancia*. Az információk feleslegesen ismétlődhetnek több sorban. A 3.6. ábrán a filmek hossza és a műfaj erre példa.

2. *Módosítási anomáliák.* Lehet, hogy megváltoztatjuk az egyik sorban tárolt információt, miközben ugyanaz az információ változatlanul marad egy másik sorban. Például, ha kiderül, hogy a *Csillagok háborúja* valójában 125 perces, – ha nem vagyunk elég gondosak – a 3.6. ábra első sorában változtatjuk meg a hosszt és a második vagy harmadik sorokban pedig nem. Persze mondhatjuk, hogy nem szabadna ilyen figyelmetlennek lenni, de majd látni fogjuk, hogy a *Filmek1* relációt át lehet úgy tervezni, hogy az ilyen hibák veszélye egyáltalán ne merülhessen fel.
3. *Törlési anomáliák.* Ha az értékek halmaza üres halmazzá válik, akkor ennek mellékhatásaként más információt is elveszíthetünk. Például ha törlőnk kell az *Elfújta a szél*-ben szereplő színészek közül Vivien Leigh-t, akkor az adatbázisban ehhez a filmhez nem maradna több színész, így a *Filmek1* relációban az *Elfújta a szél* című filmre vonatkozó sor eltűnne azzal az információval együtt, hogy a film egy 231 perces dráma.

3.3.2. Relációk felbontása

Az anomáliák megszüntetésének az elfogadott útja a *relációk felbontása (dekompozíciója)*. R felbontása azt jelenti, hogy R attribútumait szétosztjuk úgy, hogy két új reláció sémáját alakítjuk ki belőlük. A felbontási eljárás megadása után meghatározzuk, hogyan válasszunk ki egy olyan felbontást, amellyel megszüntetjük az anomáliákat.

Legyen adott az R reláció az $\{A_1, A_2, \dots, A_n\}$ sémával. R -t felbonthatjuk $S(B_1, B_2, \dots, B_m)$ és $T(C_1, C_2, \dots, C_k)$ relációra úgy, hogy

1. $\{A_1, A_2, \dots, A_n\} = \{B_1, B_2, \dots, B_m\} \cup \{C_1, C_2, \dots, C_k\}$.
2. $S = \pi_{B_1, B_2, \dots, B_m}(R)$.
3. $T = \pi_{C_1, C_2, \dots, C_k}(R)$.

3.14. példa. Bontsuk fel a 3.6. ábrán látható *Filmek1* relációt. A választott felbontás, amelynek az előnyeit a 3.3.3. alfejezetben fogjuk látni, az alábbi:

1. A *Filmek2* séma tartalmazza az összes attribútumot, kivéve a **színészNév** nevűt.
2. A *Filmek3* séma pedig tartalmazza a **filmcím**, **év** és **színészNév** attribútumokat.

A *Filmek1* reláció vetítése erre a két sémára a 3.7. ábrán látható. \square

Figyeljük meg, hogyan szünteti meg ez a felbontás a 3.3.1. alfejezetben említett anomáliákat. A redundanciát kizártuk, például minden film hossza csak egyszer fordul elő a *Filmek2* relációban. A módosítási anomália kockázata megszűnt. Például, mivel csak a *Filmek2* egy sorában kell megváltoztatnunk a *Csillagok háborúja* film hosszát, ezért nem fordulhat elő, hogy ennek a filmnek két

filmcím	év	hossz	műfaj	stúdióNév
Csillagok háborúja	1977	124	sci-fi	Fox
Elfújta a szél	1939	231	dráma	MGM
Wayne világa	1992	95	vígjáték	Paramount

(a) A Filmek2 reláció

filmcím	év	színészNév
Csillagok háborúja	1977	Carrie Fisher
Csillagok háborúja	1977	Mark Hamill
Csillagok háborúja	1977	Harrison Ford
Elfújta a szél	1939	Vivien Leigh
Wayne világa	1992	Dana Carvey
Wayne világa	1992	Mike Meyers

(b) A Filmek3 reláció

3.7. ábra. A Filmek1 reláció vetületei

különböző hosszát tároljuk véletlenül. Végül a törlési anomália kockázata is megszűnt. Ha töröljük az *Elfújta a szél* című filmben szereplő összes színészt, akkor ez a film kiesik a Filmek3 relációból, de az összes többi információ erről a filmről még megtalálható lesz a Filmek2-ben.

Még mindig előfordulhat redundancia a Filmek3-ban, mivel egy film címe és éve többször is szerepelhet. Jóllehet ez a két attribútum a filmeknek egy kulcsát alkotja, mégsem tudjuk másképpen szabatosan reprezentálni a filmet. Továbbá a Filmek3-ban mégsem merül fel a módosítási anomália lehetősége. Ha megváltoztatjuk a *Csillagok háborúja* évét 2008-ra Carrie Fisher sorában, de a másik két sorban nem, akkor lehetne módosítási anomália. Jóllehet egyik feltételezett funkcionális függőség sem akadályozza meg, hogy ne készüljön *Csillagok háborúja* című másik film 2008-ban, és Carrie Fisher szerepelhet abban is. Tehát nem kívánjuk megakadályozni, hogy a *Csillagok háborúja* egyik sorában az évet megváltoztassuk, sőt még csak azt sem akarjuk állítani, hogy egy ilyen változás biztosan hibás eredményre vezetne.

3.3.3. Boyce–Codd normálforma

A felbontás célja, hogy egy relációt többel helyettesítsünk úgy, hogy ezzel megszüntessük az anomáliákat. Kiderült, hogy van egy egyszerű feltétel, ami biztosítja, hogy a korábban tárgyalt anomáliák ne fordulhassanak elő. Ezt a feltételt *Boyce–Codd normálformának* vagy *BCNF-nek* nevezzük.

- Az R reláció BCNF-ben van akkor és csak akkor, ha minden olyan esetben, ha R -ben érvényes egy $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ nem triviális függőség, akkor az $\{A_1, A_2, \dots, A_n\}$ halmaz R szuperkulcsa.

Azaz minden nem triviális funkcionális függőség bal oldalának szuperkulcsnak kell lennie. Emlékeztetünk arra, hogy a szuperkulcsnak nem kell minimálisnak lennie. Emiatt a BCNF feltétel egy ekvivalens megfogalmazása az, hogy minden nem triviális funkcionális függőség bal oldalának tartalmaznia kell egy kulcsot.

3.15. példa. A 3.6. ábrán látható *Filmek1* reláció nincs BCNF-ben. Ahhoz, hogy belássuk, miért nem, először meg kell határoznunk, mely attribútumhalmazok alkotják a kulcsokat. A 3.2. példában láttuk, hogy a $\{\text{filmcím}, \text{év}, \text{színészNév}\}$ miért alkot kulcsot. Tehát bármely olyan attribútumhalmaz, amely tartalmazza ezt a hármat, szuperkulcs. A 3.2. példa megfontolásaihoz hasonlóan beláthatjuk azt is, hogy nem lehet szuperkulcs az olyan attribútumhalmaz, amely nem tartalmazza minden hármat a *filmcím*, *év* és *színészNév* közül. Tehát azt kaptuk, hogy a $\{\text{filmcím}, \text{év}, \text{színészNév}\}$ az egyetlen kulcs a *Filmek1* relációhoz.

Tekintsük a következő funkcionális függőséget:

$$\text{filmcím } \text{év} \rightarrow \text{hossz } \text{műfaj } \text{stúdióNév}$$

amelyről a 3.2. példából tudjuk, hogy fennáll a *Filmek1* relációban.

Sajnos a fenti függőség bal oldala nem szuperkulcs, hiszen tudjuk, hogy a *filmcím* és *év* funkcionálisan nem határozza meg a hatodik attribútumot, a *színészNév* attribútumot. Azaz ennek a függőségnak a létezése megséríti a BCNF feltételt, és emiatt a *Filmek1* nincs BCNF-ben. \square

3.16. példa. Másrészt a 3.7. ábrán látható *Filmek2* reláció BCNF-ben van. Mivel

$$\text{filmcím } \text{év} \rightarrow \text{hossz } \text{műfaj } \text{stúdióNév}$$

fennáll ebben a relációban, és korábban láttuk, hogy sem a *filmcím* sem az *év* önmagában nem határozza meg funkcionálisan a többi attribútum egyikét sem, ezért a *Filmek2* reláció egyetlen kulcsa a $\{\text{filmcím}, \text{év}\}$. Továbbá a nem triviális funkcionális függőségek legalább a *filmcím* és *év* attribútumokat tartalmazzák a bal oldalon. Emiatt a bal oldalak szuperkulcsok, tehát a *Filmek2* BCNF-ben van. \square

3.17. példa. Azt állítjuk, hogy bármely két attribútumból álló reláció BCNF-ben van. Azokat a lehetséges nem triviális függőségeket kell megvizsgálnunk, amelyeknek jobb oldala egyetlen attribútumból áll. Nincs túl sok eset, amit figyelembe kell vennünk, ezért nézzük meg ezeket sorban. Tegyük fel, hogy az attribútumok A és B .

1. Nincs nem triviális függőség. Ekkor a BCNF feltétel mindenkorban érvényes, hiszen csak a nem triviális függőségek sérthetik meg a feltételt. Megjegyezzük egyébként, hogy $\{A, B\}$ az egyetlen kulcs ebben az esetben.

2. $A \rightarrow B$ fennáll, de $B \rightarrow A$ nem áll fenn. Ebben az esetben A az egyedüli kulcs, és minden nem triviális függőség tartalmazza A -t a bal oldalon (valójában a bal oldal csak A lehet). Tehát nem sérül a BCNF feltétel.
3. $B \rightarrow A$ fennáll, de $A \rightarrow B$ nem áll fenn. Ez az eset szimmetrikus az előző, 2. esettel.
4. Mindkét $A \rightarrow B$ és $B \rightarrow A$ fennáll. Ekkor minden attribútum, A és B is kulcs. Bármelyik függőséget nézzük, a két attribútum közül az egyik a bal oldalon áll, amiatt nem sértheti meg a BCNF feltételt.

Érdemes megjegyeznünk a 4. eset kapcsán, hogy egy relációnak több kulcsa lehet. Továbbá a BCNF feltétel csak azt követeli meg, hogy bármely nem triviális függőség bal oldala tartalmazzon valamilyen kulcsot, de nem kell minden kulcsot tartalmaznia a bal oldalnak. Azt is megjegyezzük, hogy a kétattribútumú relációnál az az eset, amikor bármelyik attribútum funkcionálisan meghatározza a másikat, nem teljesen valószínűtlen. Például egy vállalat a dolgozóinak egyértelmű alkalmazottazonosítót adhat meg, és a személyi számukat is bejegyzik. Abban a relációban, amelyiknek a **alkalmazottAzonosító** és a **személyiSzám** az attribútumai, bármelyik attribútum funkcionálisan meghatároz a másikat. Másképpen kifejezve, minden attribútum külön-külön kulcs, mivel várhatóan nem találunk két olyan sort, ahol az egyes attribútumértékek megegyeznének. □

3.3.4. Boyce–Codd normálformájú felbontás

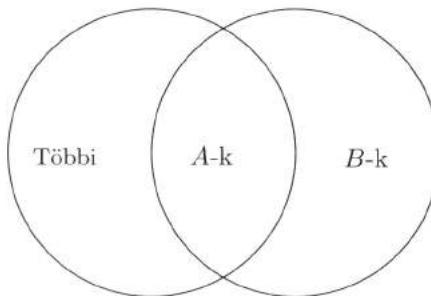
Alkalmas felbontások ismétlődő választásával bármely relációsémát fel tudunk bontani az attribútumaiból álló részhalmazok összességére, amelyre az alábbi fontos tulajdonságok teljesülnek:

1. Ezek a részhalmazok BCNF-ben levő relációsémák.
2. Az eredeti relációban tárolt adatokat pontosan reprezentálják a felbontás eredményeként nyert relációk adatai. A 3.4.1. alfejezetben pontosítjuk, hogyan értjük ezt. Durván megfogalmazva arra van szükségünk, hogy a felbontással kapott relációk előfordulásaiból képesek legyünk pontosan visszaállítani az eredeti reláció előfordulását.

A 3.17. 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 minden teljesítik a 2. feltételt, ahogyan később a 3.4.1. alfejezetben 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 $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ nem triviális funkcionális függőséget, amely megsérti a BCNF-et, azaz $\{A_1, A_2, \dots, A_n\}$ nem szuperkulcs. A jobb oldalhoz hozzáadjuk az összes

$\{A_1, A_2, \dots, A_n\}$ által funkcionálisan meghatározott attribútumot. Ez a lépés nem kötelező, de többnyire csökkenti a végrehajtás összköltségét, ezért be fogjuk építeni az algoritmusunkba. A 3.8. ábrán látható, hogyan bonthatók az attribútumok két, egymást részben fedő relációsémára. Az egyik az összes olyan attribútum, amely a feltételt megsértő függőségen előfordul, a másik pedig a függőség bal oldala és azok az attribútumok, amelyek nem szerepelnek a függőségen, például az összes attribútum, kivéve azokat a B -ket, amelyek nem A -k.



3.8. ábra. BCNF megsértésén alapuló relációséma felbontása

3.18. példa. Vizsgáljuk meg a futó példánkat, a Filmek1-et a 3.6. ábrán. A 3.15. példában láttuk, hogy a

filmcím év → hossz műfaj stúdióNév

függőség megséri a BCNF feltételt. Ebben az esetben a jobb oldal már tartalmazza azokat az attribútumokat, amelyeket a filmcím és az év funkcionálisan meghatároz, tehát a BCNF megsértése miatt a Filmek1 relációt kétfelé bontjuk:

1. A $\{\text{filmcím}, \text{év}, \text{hossz}, \text{műfaj}, \text{stúdióNév}\}$ reláció tartalmazza az összes attribútumot a függőség mindkét oldaláról.
2. A $\{\text{filmcím}, \text{év}, \text{színészNév}\}$ séma pedig tartalmazza a függőségek bal oldalát és a Filmek1 azon attribútumait, amelyek nem fordulnak elő a függőségen (ebben az esetben csak a színészNév).

Megjegyezzük, hogy ezek a sémák a 3.14. példában szereplő Filmek2 és Filmek3 sémák. A 3.16. példában láttuk, hogy a Filmek2 BCNF-ben van. A Filmek3 szintén BCNF-ben van, nincs egyetlen nem triviális függősége sem. □

A 3.18. példában elég volt a dekompozíciós szabályt józan ésszel egyszer alkalmazni ahhoz, hogy BCNF-ben lévő relációk egy csoportját kapjuk. Általában nem ez a helyzet, ahogy a következő példa is mutatja.

3.19. példa.

Vizsgáljuk meg a

$$\{filmcím, év, stúdióNév, elnök, elnökCím\}$$

sémát. Azaz minden sorban tároljunk egy filmet, a stúdiót, ahol készült, a stúdió vezetőjét és az ő lakkímét. Feltételezzük az alábbi függőségek teljesülését:

$$\begin{aligned} filmcím &\rightarrow stúdióNév \\ stúdióNév &\rightarrow elnök \\ elnök &\rightarrow elnökCím \end{aligned}$$

Az öt attribútumból álló halmaz lezárással azt tapasztalhatjuk, hogy ennek a relációnak az egyetlen kulcsa a $\{filmcím, év\}$. Így a fent említett függőségek közül az utolsó kettő megséríti a BCNF feltételt. Tegyük fel, hogy a szétvágást a

$$stúdióNév \rightarrow elnök$$

függőséggel kezdjük. Először hozzáadjuk ennek a függőségnek a jobb oldalához az összes attribútumot, ami a $stúdióNév$ lezártjában szerepel. A lezárt tartalmazza az $elnökCím$ attribútumot, tehát a szétvágáshoz választott függőség végül:

$$stúdióNév \rightarrow elnök elnökCím$$

A dekompozíció, amelyet e szerint a függőség szerint hajthatunk végre, az alábbi két relációsémához vezet:

$$\begin{aligned} &\{filmcím, év, stúdióNév\} \\ &\{stúdióNév, elnök, elnökCím\} \end{aligned}$$

Ha a 3.12. algoritmust használjuk a függőség vetületének elkészítéséhez, akkor megállapíthatjuk, hogy az első reláción a függőségek bázisa

$$filmcím \rightarrow stúdióNév$$

míg a másodikon

$$\begin{aligned} stúdióNév &\rightarrow elnök \\ elnök &\rightarrow elnökCím \end{aligned}$$

Az egyetlen kulcs az első relációban a $\{filmcím, év\}$, és ennek következtében BCNF-ben van. Ugyanakkor a másodiknak a $\{stúdióNév\}$ az egyetlen kulcsa, de érvényes benne az

$$elnök \rightarrow elnökCím$$

függőség, ami a BCNF feltétel megsértése. Azaz ismét dekompozíciót kell végrehajtanunk, ezúttal az előbbi függőséggel. A kapott három relációséma minden egyike BCNF-ben van:

$\{filmcím, év, stúdióNév\}$
 $\{stúdióNév, elnök\}$
 $\{elnök, elnökCím\}$

□

Általában addig kell alkalmazni a dekompozíciós szabályt, amíg minden kapott séma a BCNF feltételnek megfelelő nem lesz. Biztosak lehetünk abban, hogy az eljárás véget ér, ugyanis minden alkalommal a dekompozíciós szabályt alkalmazva R -re a kapott relációk kevesebb attribútumot fognak tartalmazni, mint R -ben volt. Ahogy láttuk a 3.17. példában, amint elérünk a két attribútumig, már biztosan BCNF-sémához jutunk; gyakran a nagyobb attribútumhal-mazú relációk is BCNF-ben vannak. A módszer az alábbiakban látható:

3.20. algoritmus. BCNF-dekompozíció algoritmusa

BEMENET: Az R_0 reláció az S_0 függőséghalmazzal.

KIMENET: Az R_0 dekompozíciója relációk csoportjára, melyek mindegyike BCNF-ben van.

ELJÁRÁS: A következő lépések bármely R relációra és S függőséghalmazra alkalmazhatók rekurzívan. Kezdetben legyen $R = R_0$ és $S = S_0$.

1. Ellenőrizzük, R BCNF-ben van-e. Ha igen, akkor készen vagyunk, és $\{R\}$ a válasz.
2. Ha vannak BCNF-et megsértő függőségek, akkor $X \rightarrow Y$ legyen egy ilyen. Használjuk a 3.7. algoritmust X^+ kiszámításához. Legyen $R_1 = X^+$ az egyik relációséma, továbbá R_2 -ben legyenek benne X attribútumai és azok az R -beli attribútumok, amelyek nincsenek X^+ -ban.
3. Használjuk a 3.12. algoritmust az R_1 és R_2 függőségeinek meghatározásához, a kapottak legyenek rendre S_1 és S_2 .
4. Rekurzívan bontsuk fel R_1 -et és R_2 -t ennek az algoritmusnak a használataival. A végeredmény a dekompozíciók eredményeinek uniója lesz.

□

3.3.5. Feladatok

3.3.1. feladat. A következő relációsémákra és funkcionális függőségi hal-mazokra:

- a) $R(A, B, C, D)$ sémára és $AB \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$ funkcionális függőségekre;
- b) $R(A, B, C, D)$ sémára és $B \rightarrow C$, $B \rightarrow D$ funkcionális függőségekre;

- c) $R(A, B, C, D)$ sémára és $AB \rightarrow C$, $BC \rightarrow D$, $CD \rightarrow A$, $AD \rightarrow B$ funkcionális függőségekre;
- d) $R(A, B, C, D)$ sémára és $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$ funkcionális függőségekre;
- e) $R(A, B, C, D, E)$ sémára és $AB \rightarrow C$, $DE \rightarrow C$, $B \rightarrow D$ funkcionális függőségekre;
- f) $R(A, B, C, D, E)$ sémára és $AB \rightarrow C$, $C \rightarrow D$, $D \rightarrow B$, $D \rightarrow E$ 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, amelyeket szükséges, BCNF-ben levő relációkra.

3.3.2. feladat. A 3.3.4. alfejezetben említettük, hogy gyakoroljuk annak a lehetőségnek a használatát, hogy a függőségek jobb oldalát kiterjesztjük, amíg egy BCNF-megsértés lehetséges. Vizsgáljuk az R relációt, melynek sémája $\{A, B, C, D\}$, az érvényes függőségei pedig $A \rightarrow B$ és $A \rightarrow C$. Mindkettő egy-egy BCNF-megsértés, mert az R egyetlen kulcsa $\{A, D\}$. Tegyük fel, hogy az R dekompozícióját $A \rightarrow B$ szerint végezzük. Ugyanazt kapjuk végeredményként, ha kiterjesztjük a BCNF-et sértő függőséget $A \rightarrow BC$ -re? Miért? Miért nem?

! 3.3.3. feladat. Legyen R ugyanaz a reláció, mint a 3.3.2. feladatban, de most az $A \rightarrow B$ és $B \rightarrow C$ funkcionális függőségek legyenek érvényesek. Újból hasonlítsuk össze R felbontását, amikor az $A \rightarrow B$ funkcionális függőségek legyenek érvényesek. Újból hasonlítsuk össze R felbontását, amikor az $A \rightarrow BC$ -t vesszük először.

! 3.3.4. feladat. Tegyük fel, hogy van egy $R(A, B, C)$ relációsémánk az $A \rightarrow B$ funkcionális függőséggel. Azt is tegyük fel, hogy felbontottuk ezt $S(A, B)$ és $T(B, C)$ sémáakra. Adjunk példát az R reláció egy olyan előfordulására, amelynek az S -re és T -re vett vetítéseit újból összekapcsolva (mint a 3.4.1. alfejezetben), nem ugyanazz a reláció-előfordulást adja vissza, azaz $\pi_{A,B}(R) \bowtie \pi_{B,C}(R) \neq R$.

3.4. Dekompozíció: a jó, a rossz és a csúf

Eddig láttuk, hogy a BCNF-re bontás előtt a relációsémában lehetségesek az anomáliák; a dekompozíció után a kapott eredményekben nem fordulhatnak elő anomáliák. Ez a „jó”. A dekompozíciónak ugyanakkor van néhány rossz, sőt kimondottan „csúf” következménye is. Ebben a részben megvizsgálunk három különböző tulajdonságot, amelyekről szeretnénk, ha a dekompozícióra teljesülne.

1. *Anomáliák kiküszöbölése* dekompozícióval, ahogyan a 3.3. alfejezetben láttuk.
2. *Információ-visszaállíthatóság*. Vissza tudjuk-e állítani az eredeti relációt a dekompozícióval kapott relációk rekordjaiból?
3. *Függőségek megőrzése*. Ha ellenőrizzük a függőségek vetületét a dekompozícióval kapott relációkon, akkor garantálni tudjuk-e, hogy ha felépítjük az eredeti relációt a komponensek összekapcsolásával, akkor az eredmény kielégíti az eredeti függőségeket?

Kiderül, hogy a 3.20. BCNF-algoritmus teljesíti az 1. és 2. feltételeket, de nem szükségszerűen minden relációt teljesíti. A 3.5. alfejezetben látni fogunk egy másik dekompozíciós módszert, amely biztosítja a 2. és 3. feltételeket, de nem feltétlenül minden relációt teljesíti. Valójában nincs olyan módszer, amivel minden relációt teljesíti az eredeti függőségeket.

3.4.1. Információ visszanyerése a komponensekből

Mivel láttuk, hogy minden kétattribútumú reláció BCNF-ben van, felmerülhet, hogy miért kell a 3.20. algoritmus nehézségével foglalkoznunk? Miért nem mondjuk azt minden relációra, hogy felbontjuk olyan relációkat, melyek minden attribútumának a sémája egy-egy attribútumpár R -ból? A válasz az, hogy az adatok a felbontással kapott relációkban nem biztos, hogy lehetővé teszik a relációk összekapcsolását és R visszaállítását, még akkor sem, ha minden attribútumnak a sémája egy-egy attribútumpár R -ból. A válasz az, hogy az adatok a felbontással kapott relációkban nem biztos, hogy lehetővé teszik a relációk összekapcsolását és R visszaállítását, még akkor sem, ha minden attribútumnak a sémája egy-egy attribútumpár R -ból. A válasz az, hogy az adatok a felbontással kapott relációkban nem biztos, hogy lehetővé teszik a relációk összekapcsolását és R visszaállítását, még akkor sem, ha minden attribútumnak a sémája egy-egy attribútumpár R -ból.

Ugyanakkor, ha a 3.20. algoritmussal végezzük a felbontást, ahol minden dekompozíció egy BCNF-et megsértő függőségből indul, akkor az eredeti reláció vetületeiként kapott rekordok összekapcsolhatók úgy, hogy csak az eredeti sorokat adják vissza. Itt minden rekordot megmutatjuk, miért. Majd a 3.4.2. alfejezetben megadunk egy „chase” nevű algoritmust, amellyel ellenőrizhetjük, hogy a relációk vetülete bármely felbontásra lehetővé teszi-e a reláció visszaállítását újra-összekapcsolással.

A helyzet egyszerűsítése érdekében vizsgáljuk az $R(A, B, C)$ relációt és a $B \rightarrow C$ függőséget, ami a BCNF megsértése. A $B \rightarrow C$ függőség kettéosztja az attribútumokat az $R_1(A, B)$ és $R_2(B, C)$ relációkba.

Legyen t egy R -beli rekord. Felfrjuk t -t $t = (a, b, c)$ alakban, ahol a , b és c a t rekord attribútumai rendre az A , B és C attribútumokhoz. A t rekord t vetülete (a, b) az $R_1(A, B) = \pi_{A,B}(R)$ -ben és (b, c) az $R_2(B, C) = \pi_{B,C}(R)$ -ben. Amikor az $R_1 \bowtie R_2$ természetes összekapcsolást számítjuk ki, akkor ezek a rekordok összekapcsolódnak, mert megegyeznek a közös B attribútumon (mivel minden attribútumnak a sémája egy-egy attribútumpár R -ból). Megadják a $t = (a, b, c)$ rekordot, azaz a kiinduló rekordot a kapcsolásban. Vagyis függetlenül attól, hogy milyen sorokból indultunk ki, a vetületek minden rekordot megkapnak, hogy visszakapjuk a kiinduló rekordot.

Azonban a kiinduló sorok visszanyerése nem elegendő ahhoz, hogy biztosak lehessünk abban, hogy a dekompozícióval kapott relációk valóban az eredeti R relációt reprezentálják. Gondoljuk meg, mi történik akkor, amikor van két sorunk R -ben, például $t = (a, b, c)$ és $v = (d, b, e)$. Amikor levetítjük t -t $R_1(A, B)$ -re, akkor $u = (a, b)$ -t kapunk, és amikor levetítjük v -t $R_2(B, C)$ -re, akkor $w = (b, e)$ -t kapunk. Ezek a sorok szintén összeillenek a természetes összekapcsoláskor, és a kapott sor $x = (a, b, e)$ lesz. Lehetséges, hogy x egy hamis sor? Azaz lehet, hogy (a, b, e) nem az R egy sora?

Mivel feltettük, hogy a $B \rightarrow C$ érvényes R -re, a válasz „nem”. Idézzük fel, hogy ez a függőség azt jelenti, hogy minden olyan sor, amely megegyezik a B komponensen, megegyezik a C komponensen is. Ez azt jelenti, hogy $c = e$, azaz a két attribútum, amelyet különbözőnek feltételeztünk, valójában ugyanaz. Tehát az (a, b, e) sor R -ben valójában (a, b, c) , azaz $x = t$.

Mivel t szerepel R -ben, x -nek is benne kell lennie. Másként mondva, mivel a $B \rightarrow C$ függőség fennáll, a két rekord összekapcsolása nem vezethet hamis rekordhoz. Illetve minden természetes összekapcsolással megkapott rekord sora lesz az R relációnak.

Ez a megállapítás általánosan igaz. Feltettük, hogy A , B és C mind külön attribútumok voltak. De ugyanez az okoskodás mondható el akkor is, ha tetszőleges X , Y és Z attribútumhalmazról van szó. Azaz, ha $Y \rightarrow Z$ fennáll R -ben, melynek attribútumai $X \cup Y \cup Z$, akkor $R = \pi_{X \cup Y}(R) \bowtie \pi_{Y \cup Z}(R)$.

Összefoglalva:

- Ha a 3.20. algoritmus szerint végezzük a reláció dekompozíjóját, akkor az eredeti reláció pontosan előállítható természetes összekapcsolással.

Hogy megértsük miért, a fenti érvelésről elmondhatjuk, hogy a rekurzív dekompozíció bármelyik lépése során a reláció ekvivalens a két komponensére vett vetületeinek összekapcsolásával. Ha ezeket a komponenseket tovább bontjuk, szintén visszaállíthatók lesznek a felbontással kapott relációk természetes összekapcsolásával. Azaz egy egyszerű indukcióval a bináris dekompozíciókon, lépésenként elmondhatjuk, hogy az eredeti reláció minden azoknak a relációknak a természetes összekapcsolása, amelyekre felbontottuk. Bizonyíthatjuk továbbá, hogy a természetes összekapcsolás asszociatív és kommutatív, azaz a felbontott komponensek természetes összekapcsolásának sorrendje nem számít.

Az $Y \rightarrow Z$ függőség vagy a vele szimmetrikus $Y \rightarrow X$ alapvető. Ezek valamelyike nélkül nem tudnánk helyreállítani az eredeti relációt. Lássunk erre is példát.

3.21. példa. Vizsgáljuk a fenti $R(A, B, C)$ relációt, melyre a $B \rightarrow A$ és a $B \rightarrow C$ függőségek egyike sem teljesül. Továbbá R tartalmazza az alábbi két sort:

A	B	C
1	2	3
4	2	5

Az R reláció vetületei az $\{A, B\}$ és $\{B, C\}$ sémájú relációkra az alábbi
 $R_1 = \pi_{AB}(R) =$

A	B
1	2
4	2

és $R_2 = \pi_{BC}(R) =$

B	C
2	3
2	5

Mivel mind a négy rekord azonos B -értékkal rendelkezik (amely 2), minden két reláció mindegyik sora összekapcsolható a másik reláció minden két sorával. Amikor megpróbáljuk visszaállítani R -et, akkor a vetületrelációk természetes összekapcsolásával egy $R_3 = R_1 \bowtie R_2 =$

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

relációt kapunk. Az eredmény „túl sok”, kapunk két hamis sort, az $(1, 2, 5)$ -öt és a $(4, 2, 3)$ -at, amelyek nem voltak benne az eredeti R relációban. \square

3.4.2. Chase-teszt a veszteségmentes összekapcsoláshoz

A 3.4.1. alfejezetben megállapítottuk, hogy az $R(A, B, C)$ reláció dekompozíciójának az $\{A, B\}$ és $\{B, C\}$ sémákra az egyetlen $B \rightarrow C$ függőséggel van veszteségmentes összekapcsolása. Most tekintsünk egy általánosabb esetet. Felbontjuk az R relációt olyanokra, amelyek attribútumhalmazai S_1, S_2, \dots, S_k . Adott egy F függősséghalmaz, amely fennáll R -ben. Igaz-e, hogy ha levetítjük R -et a dekompozícióval kapott relációkra, akkor visszakaphatjuk R -et a kapott relációk természetes összekapcsolásával? Azaz, igaz-e, hogy

$$\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \cdots \bowtie \pi_{S_k}(R) = R?$$

Három fontos dologra kell emlékeznünk:

- A természetes összekapcsolás asszociatív és kommutatív. Nem számít, hogy milyen sorrendben kapcsoljuk össze a vetületeket, eredményként ugyanazt a relációt fogjuk kapni. Pontosabban az eredmény egy olyan t sorokból álló sorhalmaz, melyre minden $i = 1, 2, \dots, k$ -ra, t vetülete az S_i attribútumhalmazokra egy sor a $\pi_{S_i}(R)$ -ben.

Az összekapcsolás az egyetlen módja az adatvisszányerésnek?

Feltessük, hogy az egyetlen lehetséges módszer az eredeti reláció felépítésének a vetületekből az, ha a természetes összekapcsolást használjuk. Talán lehetnek más algoritmusok is, amelyekkel előállíthatjuk az eredeti relációt, amely működhet olyan esetben is, amikor a természetes összekapcsolás sikertelenül végződik? Igazság szerint nincs másik út. A 3.21. példában látottuk, hogy az R és R_3 relációk különböző előfordulások, de mégis ugyanaz a vetületük az $\{A, B\}$ és $\{B, C\}$ attribútumhalmazokra, nevezetesen az R_1 -nek és R_2 -nek nevezett relációkra. Azaz adott R_1 és R_2 relációk esetén egyetlen algoritmus sem tud semmit mondani arról, hogy az eredeti reláció R vagy R_3 .

Ráadásul ez a példa nem egyedi. Tekintsük az $X \cup Y \cup Z$ attribútumokból álló reláció bármely olyan dekompozícióját az $X \cup Y$ és az $Y \cup Z$ sémákra, ahol sem $Y \rightarrow X$, sem $Y \rightarrow Z$ nem áll fenn. Ehhez konstruálhatunk egy példát, amely hasonlít a 3.21. példában láttotthoz, ahol az eredeti előfordulás nem határozható meg a vetületekből.

- Bármely t sor R -ben biztosan benne van $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \dots \bowtie \pi_{S_k}(R)$ -ben. Ennek oka az, hogy t vetületei az S_i halmazokra biztosan benne vannak $\pi_{S_i}(R)$ -ben minden i -re, és ennek következtében, ahogy az előbb láttuk, t a kapcsolás eredménye.
- Következmény, hogy amikor az F -beli függőségek fennállnak R -en, $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \dots \bowtie \pi_{S_k}(R) = R$ fennáll akkor és csak akkor, ha az összekapcsolás minden sora R -ben is benne van. Tehát csak ezt kell tesztelnünk, hogy bebizonyítsuk, a dekompozíció veszteségmentes összekapcsolású.

A *chase*-teszt a veszteségmentes kapcsoláshoz csupán egy szervezett módszer arra, hogy ellenőrizzük, egy $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \dots \bowtie \pi_{S_k}(R)$ -beli t sorról bizonyítható-e az F -beli függőségek használatával, hogy az R egy sora. Ha t szerepel az összekapcsolásban, akkor R -ben lenniük kell t_1, t_2, \dots, t_k soroknak úgy, hogy t az egyes S_1, S_2, \dots, S_k attribútumokra vett vetületek összekapcsolása minden $i = 1, 2, \dots, k$ -ra. Emiatt tudjuk, hogy t_i megegyezik t -vel az S_i attribútumain, de t_i -nek vannak ismeretlen értékei azokon a komponenseken, amelyek nincsenek S_i -ben.

Készíthetünk egy képet, amit *tablónak* nevezünk, és amelyen ábrázolhatjuk, hogy mit tudunk. Feltételezve, hogy R attribútumai A, B, \dots , az a, b, \dots komponenseket használjuk t -hez. A t_i -khez ugyanazt a betűt használjuk, mint t -ben, hogy leírjuk az S_i -beli komponenseket, de alsó indexben i betűvel jelöljük azt a komponensem, amely nincs S_i -ben. Ezzel a módszerrel t_i megegyezik t -vel az S_i

attribútumain, de egyedi értéke van – azaz mindegyik csak egyszer fordulhat elő a tablóban – a többi helyen.

3.22. példa. Tegyük fel, hogy egy $R(A, B, C, D)$ relációt van, amely felbon-tásra kerül az $S_1 = \{A, D\}$, $S_2 = \{A, C\}$ és $S_3 = \{B, C, D\}$ attribútumhalma-zokra. Ennek a dekompozíciójának a tablója a 3.9. ábrán látható.

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

3.9. ábra. Tabló az R reláció felbontására $\{A, D\}$, $\{A, C\}$ és $\{B, C, D\}$ -re

Az első sor megfelel az A és D attribútumhalmaznak. Jegyezzük meg, hogy az A és D attribútumokon szereplő komponensek a nem indexelt a és d betűk. Emellett a többi attribútumnak – b és c – adunk egy 1-es alsó indexet, hogy je-lezzük, ezek tetszőleges értékek. A választás magyarázata, hogy az (a, b_1, c_1, d) R egy sorát jelenti, amely a $t = (a, b, c, d)$ sorból készült az $\{A, D\}$ -re való vetítéssel, majd más sorokkal való összekapcsolással. Mivel a rekord B és C komponenseit a vetítéskor eldobtuk, nem tudunk semmit arról, hogy a sor mi-lyen értékekkel rendelkezett ezeken az attribútumokon.

Hasonlóan a második sor index nélküli betűi az A és C attribútumokhoz tartoznak, míg a többi attribútumot 2-vel indexeljük. Az utolsó sor index nél-küli elemei a $\{B, C, D\}$ komponensekben vannak, és a-t 3-mal indexeljük. Mivel minden sor saját számot használ az indexhez, a többször előforduló jelek csak indexeletlen betűk lehetnek. \square

Emlékeztetünk rá, hogy a célunk az F függőséghalmaz felhasználásával bizo-niytani, hogy t valóban szerepel R -ben. Ennek érdekében „kivadásszuk” („chase-eljük”) a tablót az F -beli függőségeket alkalmazva, egyenlővé téve a szimbólumokat, amikor csak lehet. Ha egyszer azt kapjuk, hogy a sorok egyike t -vel egyezik meg (azaz a sorból eltűnik minden index nélküli szimbólum), akkor látjuk, hogy bármely t rekord, amely a vetületek összekapcsolásában szerepel, éppen egy rekordja R -nek.

A zűrzavar elkerülése érdekében két szimbólum egyenlővé tételekor, ha az egyik index nélküli, akkor a másik is azt az értéket kapja meg. Ha két inde-xelt szimbólumot kell egyenlővé tenni, akkor bármelyik indexet választhatjuk és megadhatjuk a másiknak. Mindemellett emlékezzünk rá, hogy szimbólumok egyenlővé tételekor az egyes jelek minden előfordulását le kell cserélni, nem csak néhány előfordulását.

3.23. példa. Folytassuk a 3.22. példát, és tegyük fel, hogy az $A \rightarrow B$, $B \rightarrow C$ és $CD \rightarrow A$ függőségek fennállnak. Kezdjük a 3.9. ábrán látható tablóval. Mivel az első két sor megegyezik az A attribútumon, az $A \rightarrow B$ függőség szerint a

B attribútumon is megegyeznek. Tehát $b_1 = b_2$. Bármelyiket lecserélhetjük a másikra, mivel minden kettő indexelt. Cseréljük most le b_2 -t b_1 -re. A kapott tabló:

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

Most láthatjuk, hogy az első két sor megegyezik a B attribútumon is, tehát felhasználhatjuk a $B \rightarrow C$ függőséget, amellyel levezethetjük, hogy a C komponensei, c_1 és c megegyeznek. Mivel c indexeletlen, c_1 -et cseréljük le c -re, így adódik:

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

Következőként figyeljük meg, hogy az első és a harmadik sorok mind a C , mind a D oszlopban megegyeznek, azaz alkalmazhatjuk a $CD \rightarrow A$ függőséget, hogy levezessük, ezeknek a soroknak az A -értéke is azonos, azaz $a = a_3$. Lecseréljük a_3 -t a -ra, mellyel adódik:

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

Ezen a ponton láthatjuk, hogy az utolsó sor t -vel egyenlő lett, azaz (a, b, c, d) . Ezzel beláttuk, hogy amennyiben R megfelel az $A \rightarrow B$, $B \rightarrow C$ és $CD \rightarrow A$ függőségeknek, ha levetítjük az $\{A, D\}$, $\{A, C\}$ és $\{B, C, D\}$ sémákra, majd újra összekapcsoljuk, akkor a kapott eredménynek benne kell lennie R -ben. Sőt pontosan megegyezik R -nek azzal a sorával, amelyet $\{B, C, D\}$ -re vetítettünk.

□

3.4.3. Miért működik a chase?

Két kérdést kell megválaszolni:

- Amikor a chase előállít egy t -vel megegyező sort (azaz a tablóban van indexmentes sor), akkor ebből miért következik, hogy a kapcsolás veszteségmentes?
- Ha az összes lehetséges függőség alkalmazása után sem találunk olyan sort, amelyben csak index nélküli változók vannak, miért nem lehet a kapcsolás veszteségmentes?

Az első kérdés megválaszolása egyszerű. A chase-eljárás maga egy bizonyítása annak, hogy a vetületsoroknak R -ból mindenképpen elő kell állítniuk t -t a függőségekkel. Másrészt tudjuk, hogy minden R -beli sor elkészül, ha elvégezzük a vetítést és az összekapcsolást. Tehát a chase bebizonyítja, hogy a vetítés és az összekapcsolás eredménye pontosan R .

A második kérdéshez tegyük fel, hogy végül is egy olyan tablót állítunk elő, melyben nincs indexmentes sor, és ebben a tablóban már nem tudunk újabb függőségeket felhasználva szimbólumokat egyenlővé tenni. Gondoljunk a tablóra úgy, mint R egy előfordulására. Ez nyilvánvalóan kielégíti a megadott függőségeket, mert többet már nem lehet alkalmazni szimbólumok egyenlővé tételehez. Tudjuk, hogy az i . sorban az indexmentes elemek S_i attribútumai között vannak, a dekompozíció i . relációjában. Így amikor levetítjük ezt a relációt S_i -re, majd vesszük a természetes összekapcsolást, akkor megkapjuk az index nélküli elemekből álló sort. Ez a sor nincs benne R -ben, tehát megállapíthatjuk, hogy az összekapcsolás nem veszteségmentes.

3.24. példa. Vizsgáljuk meg az $R(A, B, C, D)$ relációt a $B \rightarrow AD$ függőséggel és a javasolt felbontással $\{A, B\}$, $\{B, C\}$ és $\{C, D\}$ sémákra. Itt látható a kiinduló tabló:

A	B	C	D
a	b	c_1	d_1
a_2	b	c	d_2
a_3	b_3	c	d

Amikor alkalmazzuk az egyetlen függőséget, akkor levezethetjük, hogy $a = a_2$ és $d_1 = d_2$. Tehát az eredménytábló az alábbi:

A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a_3	b_3	c	d

További módosítást nem tudunk végrehajtani, mert nincs más függőség meghatározva, és nincsen olyan sor, amely teljesen indexmentes. Így ennek a dekompozícióknak nincs veszteségmentes összekapcsolása. Ezt az állítást beláthatjuk, ha felhasználjuk a fenti tablót, mint egy relációt három sorral. Amikor $\{A, B\}$ -re vetítünk, $\{(a, b), (a_3, b_3)\}$ adódik. A vetület $\{B, C\}$ -re $\{(b, c_1), (b, c), (b_3, c)\}$, és a vetület $\{C, D\}$ -re $\{(c_1, d_1), (c, d_1), (c, d)\}$. Ha összekapcsoljuk az első két vetületet, akkor $\{(a, b, c_1), (a, b, c), (a_3, b_3, c)\}$ adódik. Ha ezt a relációt a harmadik vetüettel is összekapcsoljuk, akkor $\{(a, b, c_1, d_1), (a, b, c, d_1), (a, b, c, d), (a_3, b_3, c, d_1), (a_3, b_3, c, d)\}$ adódik. Megjegyezzük, hogy ez az összekapcsolás R -hez képest két további sort tartalmaz, sőt tartalmazza az (a, b, c, d) sort, ahogy szükséges. \square

3.4.4. Függőségek megőrzése

Említettük, hogy néhány esetben lehetetlen a relációt BCNF-relációkra bontani úgy, hogy a kapott eredmény rendelkezzen veszteségmentes összekapcsolással és megőrizze a függőségeket is. Az alábbiakban egy olyan példa következik, melyben kompromisszumot kellett kötnünk a függőségek megőrzése és a BCNF között.

3.25. példa. Tegyük fel, hogy az alábbi *Vetítések* relácionál adott, a következő attribútumokkal:

1. **filmcím**, a film neve.
2. **mozi**, a mozi neve, ahol a filmet vetítik.
3. **város**, a város neve, ahol a mozi megtalálható.

Az (m, t, c) rekord jelentése, hogy az m című filmet jelenleg a t moziban játsszák, c városban.

Érthető, hogy feltételezhetjük az alábbi függőségeket:

$$\begin{aligned} \text{mozi} &\rightarrow \text{város} \\ \text{filmcím} \text{ } \text{város} &\rightarrow \text{mozi} \end{aligned}$$

Az első azt jelenti, hogy a mozi egyetlen városban van. A második nem teljesen nyilvánvaló, de azon a gyakorlaton alapul, hogy egyazon város két mozijában nem kötjük le ugyanazt a filmet egy időben.

Először keressük meg a kulcsokat. Egyetlen egyedülálló attribútum sem kulcs. Például a **filmcím** nem kulcs, mert a filmet játszhatják egyazon időben különböző városokban és különböző mozikban.⁴ A **mozi** sem kulcs, mert bár a **mozi** funkcionálisan meghatározza a **város**-t, vannak többvásznas mozik, így egyszerre több filmet vetítenek. Végül a **város** sem kulcs, mert a városokban általában egynél több mozi van, és egynél több filmet játszanak.

Másrészt a három kétattribútumos részhalmazból kettő is kulcs. Világos, hogy a **{filmcím, város}** kulcs, mert a megadott függőség szerint ezek az attribútumok meghatározzák a **mozi**-t.

Ugyanígy igaz, hogy a **{mozi, filmcím}** is kulcs, mert a lezártja tartalmazza a **város**-t a **mozi** \rightarrow **város** függőség következtében. A fennmaradó attribútumpár, a **város** és a **mozi** nem határozza meg funkcionálisan a **filmcím**-et a többvásznas mozik miatt, így nincs több kulcs. Összefoglalva, a két lehetséges kulcs:

$$\begin{aligned} \{\text{filmcím}, \text{város}\} \\ \{\text{mozi}, \text{filmcím}\} \end{aligned}$$

⁴ Ebben a példában feltesszük, hogy nincs egyszerre két „aktuális” film, amelyeknek azonos a címe, annak ellenére, hogy korábban elismertük, lehet két film azonos címmel, amelyek különböző évben készültek.

Azonnal látható a BCNF megsértése. Megadtunk egy funkcionális függőséget, a $\text{mozi} \rightarrow \text{város}$ -t, de a bal oldali mozi nem szuperkulcs. Ez arra csábít bennünket, hogy ezt a BCNF-et sértő függőséget felhasználva felbontsuk a relációt az alábbi két sémára:

$$\begin{array}{l} \{\text{mozi}, \text{város}\} \\ \{\text{mozi}, \text{filmcím}\} \end{array}$$

Ezzel a felbontással nincs semmi gond az alábbi függőséget illetően:

$$\text{filmcím város} \rightarrow \text{mozi}$$

Lehetnek az aktuális relációk a felbontott sémákban olyanok, hogy kielégítik a $\text{mozi} \rightarrow \text{város}$ függőséget (amely ellenőrizhető a $\{\text{mozi}, \text{város}\}$ relációban), de amikor elvégezzük az összekapcsolást, az olyan relációt eredményez, amely nem elégíti ki a $\text{filmcím város} \rightarrow \text{mozi}$ függőséget. Például az alábbi relációkról:

mozi	város
Guild	Menlo Park
Park	Menlo Park

és

mozi	filmcím
Guild	Z, a hangya
Park	Z, a hangya

azt gondolnánk, hogy megfelelnek a fenti relációra alkalmazottnak, de amikor összekapcsoljuk őket, két sort kapunk:

mozi	város	filmcím
Guild	Menlo Park	Z, a hangya
Park	Menlo Park	Z, a hangya

amelyek megsértik a $\text{filmcím város} \rightarrow \text{mozi}$ függőséget. \square

3.4.5. Feladatok

3.4.1. feladat. Adott az $R(A, B, C, D, E)$ és relációkra való bontása a következő attribútumhalmazokkal: $\{A, B, C\}$, $\{B, C, D\}$ és $\{A, C, E\}$. Az alábbi függősséghalmazok mindenekkel végezzük el a chase-tesztet, hogy eldönthessük, veszteségmentes-e a dekompozíció. Azokra, amelyek nem veszteségmentesek, adjunk példát az R olyan előfordulására, melynél a dekompozíció és az újraösszekapcsolás az R -nél több sort eredményez.

- a) $B \rightarrow E$ és $CE \rightarrow A$;
- b) $AC \rightarrow E$ és $BC \rightarrow D$;
- c) $A \rightarrow D$, $D \rightarrow E$ és $B \rightarrow D$;
- d) $A \rightarrow D$, $CD \rightarrow E$ és $E \rightarrow D$.

! 3.4.2. feladat. A 3.4.1. feladat összes függőséghalmazai közül melyiket őrzi meg a dekompozíció?

3.5. Harmadik normálforma

A 3.25. példában látott problémát megoldhatjuk úgy, hogy enyhítünk a BCNF feltételen annak érdekében, hogy alkalmanként megengedhető legyen, hogy ha egy relációsémát nem tudunk felbontani BCNF-relációkra, akkor ne végezzük el, vagy enyhítsük az FD-k ellenőrzését. Ennek az enyhítő feltételnek a neve „harmadik normálforma”. Ebben a részben megadjuk a harmadik normálforma követelményeit, aztán megadunk egy algoritmust, amellyel elvégezhetjük a dekompozíciót, a 3.20. algoritmustól lényegesen eltérő módon annak érdekében, hogy megkapjuk a harmadik normálformát és megtartsuk mind a veszteségmentes összekapcsolást, mind pedig a függőségmegőrző tulajdonságokat.

3.5.1. A harmadik normálforma definíciója

Az R reláció *harmadik normálformában* (3NF) van akkor és csak akkor, ha:

- Valahányszor létezik R -ben egy $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ nem triviális függőség, akkor vagy az

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

halmaz az R szuperkulcsa, vagy azokra az attribútumokra B_1, B_2, \dots, B_m közül, amelyek nincsenek az A -k között, teljesül, hogy egy kulcsnak az elemei (nem feltétlenül ugyanannak a kulcsnak).

Ha egy attribútum szerepel valamelyik kulcsban, akkor gyakran *elsődleges attribútumnak* (vagy más néven *prímattribútumnak*) nevezzük. Így a 3NF feltételét átfogalmazhatjuk: „minden nem triviális függőségre igaz, hogy bal oldala szuperkulcs, vagy jobb oldala csak elsődleges attribútumokat tartalmaz”.

Megjegyezzük, hogy a 3NF feltétel és a BCNF feltétel közötti különbség a „vagy jobb oldala csak elsődleges attribútumokat tartalmaz” mondatrészben van, ami „felmenti” a $\text{mozi} \rightarrow \text{város}$ -hoz hasonló függőséget a 3.25. példában, mivel a jobb oldali **város** egy elsődleges attribútum.

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. Van „negyedik normálforma” is, ezzel később, a 3.6. alfejezetben találkozunk.

3.5.2. 3NF-szintetizáló algoritmus

Most kifejtjük és megindokoljuk, hogyan bontható fel egy R reláció olyan relációk halmazává, amelyekre teljesülnek az alábbiak:

- a) A felbontásban szereplő relációk mindegyike 3NF-ben van.
- b) A felbontásnak veszteségmentes összekapcsolása van.
- c) A felbontásnak függőségmegőrző tulajdonsága van.

3.26. algoritmus. Harmadik normálformában lévő relációk előállítása veszteségmentes összekapcsolással és függőségmegőrzéssel

BEMENET: Egy R reláció és az R -re vonatkozó funkcionális függőségek F halmaza.

KIMENET: Az R felbontása relációk gyűjteményére, melyek mindegyike 3NF-ben van. A felbontás rendelkezik veszteségmentes összekapcsolással és függőségmegőrző tulajdonsággal.

ELJÁRÁS: Hajtsuk végre a következő lépéseket:

1. Keressük meg F egy minimális bázisát, amit hívunk G -nek.
2. A G -ben szereplő összes $X \rightarrow A$ funkcionális függőségre használjuk XA -t sémaként a felbontás egy relációjához.
3. Amennyiben a 2. lépésben kapott relációk attribútumhalmazának egyike sem szuperkulcs R -ben, adjunk még egy relációt az eredményhez, melynek sémája kulcs lesz R -hez.

□

3.27. példa. Tekintsük az $R(A, B, C, D, E)$ relációt az $AB \rightarrow C$, $C \rightarrow B$ és $A \rightarrow D$ funkcionális függőségekkel. Első lépésként figyeljük meg, hogy a megadott funkcionális függőségek a minimális bázist jelentik. Ennek ellenőrzésére

egy kis munkát kell végeznünk. Először ellenőrizzük, hogy nem tudunk egyetlen függőséget sem elhagyni. Ennek megmutatására a 3.7. algoritmust használjuk, azaz funkcionális függőségek egyik párosa sem implikálja a harmadikat. Például vegyük az $\{A, B\}$ lezárását. Az első funkcionális függőség bal oldala csak a második és harmadik funkcionális függőségekben – azaz a $C \rightarrow B$ -ben és $A \rightarrow D$ -ben – kerül felhasználásra. Így a lezárás tartalmazza D -t, de nem tartalmazza C -t, így levonhatjuk azt a konklúziót, hogy az $AB \rightarrow C$ funkcionális függőséget nem implikálja a második és a harmadik funkcionális függőség. Hasonló konklúzióra jutunk, ha a második vagy a harmadik funkcionális függősséget próbáljuk elhagyni.

Azt is ellenőriznünk kell, hogy nem tudunk egyetlen attribútumot sem elhagyni a bal oldalról. Ebben az egyszerű esetben az egyetlen lehetőség az, hogy az első funkcionális függőségből az A -t vagy a B -t hagyjuk el. Például, ha az A -t hagyjuk el, akkor $B \rightarrow C$ marad. Azt kell megmutatni, hogy ez a $B \rightarrow C$ nem vezethető le a három eredeti funkcionális függőségből, amelyek az $AB \rightarrow C$, $C \rightarrow B$ és $A \rightarrow D$. Ezekkel a funkcionális függőségekkel a $\{B\}$ lezárása csak a B , így a $B \rightarrow C$ nem következik belőle. Így megvan a minimális bázisunk.

A 3NF előállítását azzal kezdjük, hogy vesszük az egyes funkcionális függőségek attribútumait, mint egy-egy sémát. Így az $S_1(A, B, C)$, $S_2(B, C)$ és $S_3(A, D)$ relációkat kapjuk. Sosem szükséges olyan relációt használnunk, amelynek sémája egy másik reláció részhalmaza, így az S_2 sémát eldobjuk.

Azt is meg kell vizsgálnunk, hogy szükséges-e egy olyan reláció hozzáadása, amelynek sémája egy kulcs. Ebben a példában R két kulccsal rendelkezik: $\{A, B, E\}$ és $\{A, C, E\}$, így ezeket tudjuk ellenőrizni. Ezen kulcsok egyike sem részhalmaza az eddig kiválasztott sémáknak. Így az egyiket hozzá kell adnunk, mondjuk az $S_4(A, B, E)$ sémát. Így R végső felbontása a következő: $S_1(A, B, C)$, $S_3(A, D)$ és $S_4(A, B, E)$. \square

3.5.3. Miért működik a 3NF-szintetizáló algoritmus?

Három dolgot kell megmutatnunk: azt, hogy a veszteségmentes összekapcsolási, valamint a függősegmegőrző tulajdonságok megmaradtak, és azt, hogy a felbontás relációi 3NF-ben vannak.

1. *Veszteségmentes összekapcsolás.* Kezdjük egy olyan relációval, amely attribútumainak egy K halmaza szuperkulcs. Tekintsük a funkcionális függőségek szekvenciáját, ahogy azt a 3.7. algoritmusban használtuk a K kiterjesztéséhez K^+ -ra. Mivel K egy szuperkulcs, így tudjuk, hogy K^+ az összes attribútumot jelenti. A funkcionális függőségeket a chase-tablóra alkalmazva azt kapjuk, hogy az indexelt szimbólumok a K -hoz tartozó sorban egyenlők lesznek a nem indexelt szimbólumokkal, ugyanabban a sorrendben, ahogyan az attribútumok a lezártba kerültek. Így ennek a tesztnek az eredményeként levonhatjuk azt a következtetést, hogy a felbontás veszteségmentes.

2. *Függősegmegőrzés.* A minimális bázis összes funkcionális függősége rendelkezik az összes attribútummal a felbontás valamelyik relációjában. Így az összes függőség ellenőrizhető a felbontás relációiban.
3. *Harmadik normálforma.* Amikor egy olyan relációt adunk hozzá a felbontáshoz, amelynek sémája egy kulcs, akkor ez a reláció biztosan 3NF-ben van. Ennek az az oka, hogy ennek a relációnak az összes attribútuma előfordulnak a kulcsban, így nem sérti meg a 3NF-et erre a relációra. A minimális bázis funkcionális függőségeiből származó relációk esetében annak bizonyítása, hogy 3NF-ben vannak, túlmutat ennek a könyvnek a keretein. Az érvelés annak bemutatására szorítkozik, hogy a 3NF megsértése maga után vonná, hogy a bázis nem minimális.

3.5.4. Feladatok

3.5.1. feladat. minden relációsémára és funkcionális függősegħalmażra, amely a 3.3.1. feladatban szerepelt:

- i) Derítsük fel a 3NF-sértéseket.
- ii) Ha szükséges, bontsuk fel a relációkat 3NF-ben lévő relációk halmazára.

3.5.2. feladat. Vizsgáljuk meg a Kurzusok(K, O, I, T, D, Cs) relációt, amelynek az attribútumai informálisan az alábbiak: kurzus, oktató, időpont, terem, diákok, csoport. Legyenek a funkcionális függőségek: $K \rightarrow O$, $IT \rightarrow K$, $IO \rightarrow T$, $ID \rightarrow T$ és $KD \rightarrow Cs$. Intuitívan az első állítás azt mondja, hogy egy kurzusnak egyedi oktatója van, a második azt jelenti, hogy egy időpontban egy teremben egy kurzus lehet. A harmadik azt mondja, hogy egy tanár egy időpontban csak egy teremben lehet, a negyedik pedig ugyanezt mondja a diákokról. Az utolsó pedig azt jelenti, hogy egy diáknak csak egy kurzus egyetlen csoportjába jár.

- a) Melyek a Kurzusok reláció kulcsai?
- b) Igazoljuk, hogy a megadott funkcionális függőségek minimális bázist alkotnak.
- c) Használjuk a 3NF-szintetizáló algoritmust, hogy megtaláljuk R veszteségmentes összekapcsolási és függősegmegőrző tulajdonságú dekompozíciót 3NF-relációkra. Van ezek között olyan reláció, amelyik nincs BCNF-ben?

3.5.3. feladat. Vizsgáljuk meg a Részvények(B, I, Be, R, M, O) relációt, melynek az attribútumai informálisan az alábbiak: bróker, iroda (a brókeré), befektető, részvény, mennyiségek (a befektető által birtokolt részvényé) és osztályok (a részvényé). Legyenek a Részvények-re érvényes függőségek $R \rightarrow O$, $Be \rightarrow B$, $BeR \rightarrow M$ és $B \rightarrow I$. Ismételjük meg a 3.5.2. feladatot a Részvények relációra.

3.5.4. feladat. Igazoljuk a chase használatával, hogy a 3.27. példában látott dekompozíció veszteségmentes.

!! 3.5.5. feladat. Tegyük fel, hogy módosítjuk a 3.20. algoritmust (BCNF-dekompozíció) úgy, hogy csak akkor bontjuk fel R -et, amikor nincs 3NF-ben, ahelyett hogy minden olyan esetben felbontanánk, amikor R nincs BCNF-ben. Adjunk ellenpéldát, amely igazolja, hogy a módosított algoritmus nem feltétlenül állítja elő a függősegmegőrző 3NF-felbontást.

3.6. Többértékű függőségek

A „többértékű függőség” olyan állítás, amely azt fejezi ki, hogy két attribútum vagy attribútumhalmaz független egymástól. Lájtuk 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 az alfejezetben feltárjuk a többértékű függőség okait, és látni fogjuk, hogyan használhatjuk azokat az adatbázisséma tervezésében.

3.6.1. Attribútumfüggetlenségből származó redundancia

Előfordulhatnak olyan helyzetek, hogy olyan relációsémát tervezünk, amely bár BCNF-ben van, mégis marad benne egyfajta redundancia, amely nem a funkcionális függőségezhez kapcsolódik. Leggyakrabban ez abból ered, hogy a kulcs két vagy több halmazértékű tulajdonságát is egyetlen relációval próbálunk reprezentálni.

3.28. példa. Ebben a példában feltesszük, hogy egy színésznek lehet több lakcím is. A lakcímeket kétkomponensűnek tekintjük: a várost különválasztjuk az utca-házszámtól. A színésekhez és lakcímekhez a szokásos attribútumokat is (a filmek címe és éve, amelyekben szerepelnek) hozzávesszük a relációhoz. A 3.10. ábra ennek a relációnak egy tipikus előfordulását mutatja.

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

3.10. ábra. A lakcímek halmaza független a filmektől

A 3.10. ábra középpontjában Carrie Fisher két vélt lakcíme és a három legismertebb filmje áll. Nincs értelme összekapcsolni egy lakcímét 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 3.10. ábrán háromszor megismételjük Carrie Fisher lakcímét (minden filmjéhez külön-külön) és kétszer minden filmjét (minden lakcímhez egyszer).

A 3.10. ábrán javasolt reláció 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 **város-on** kívül minden más attribútumon megegyezne, a **város-on** pedig különbözne. Így a

$$\text{név} \text{ } \text{utca} \text{ } \text{filmcím} \text{ } \text{év} \rightarrow \text{város}$$

funkcionális függőség nem érvényes a **Színészek** relációnkban. Az olvasóra bízunk, hogy ellenőrizze a többi esetben is, hogy az öt attribútumból egyiket sem határozza meg funkcionálisan a többi négy. Mivel nincs nem triviális funkcionális függőség, ebből következik, hogy az egyetlen kulcsot az öt attribútum együttesen alkotja, és nem sértheti semmi a BCNF-et. \square

3.6.2. Többértékű függőségek definíciója

A *többértékű függőség* (angolul multivalued dependency, MVD-vel rövidítjük) 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

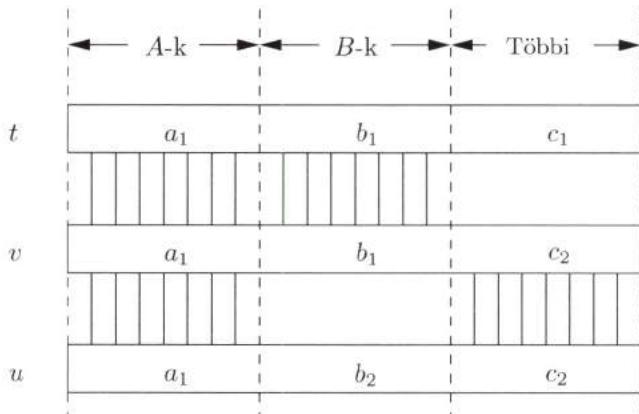
$$A_1 A_2 \cdots A_n \rightarrow\!\!\! \rightarrow B_1 B_2 \cdots B_m$$

többértékű függőség fennáll 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 azt találjuk, hogy a B -ken felvett értékek halmaza független R -nek az A -któl és B -ktől eltérő összes attribútumán felvett értékeket 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 R -ben olyan v sort, amely megegyezik:

1. t -vel és u -val az A -kon,
2. t -vel a B -ken és
3. u -val R minden olyan attribútumán, amely nincs az A -kban vagy a B -kben.

Megjegyezzük, hogy ebben a szabályban t -t és u -t felcserélhetjük, amelynek az a következménye, hogy létezik egy negyedik w sor, amely a B -ken meggyezik u -val és a többi eltérő attribútumokon pedig t -vel. Végeredményben az A -knak valamely rögzített értékeire a B -khez és a többi attribútumhoz rendelt értékeknek az összes lehetséges kombinációban elő kell fordulni a különböző sorokban. A 3.11. ábrában felvázoltuk, v hogyan viszonyul t -hez és u -hoz, amikor fennáll a többértékű függőség. Az A -k és a B -k nem feltétlenül egymást követően fordulnak elő az oszlopokban.



3.11. ábra. A többértékű függőség biztosítja v létezését

Általában feltehetjük, hogy egy többértékű függőségen az A -k és B -k (a függőség bal 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.

3.29. példa. A 3.28. példában láttunk egy többértékű függőséget, amelyet a jelöléseinkkel az alábbi formában tudunk kifejezni:

$$\text{név} \rightarrow\!\!\!-\!\!\!> \text{város} \text{ utca}$$

Azaz minden színésznevre a lakcímek halmaza kapcsolatban van a színész minden egyik filmjével. Például nézzük meg, hogyan alkalmazhatjuk a többértékű függőség formális definícióját a 3.10. ábra első és negyedik sorára:

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

Ha ezt a két sort ebben a sorrendben t -vel és u -val jelöljük, akkor a többértékű függőség azt mondja ki, hogy 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 (`filmcím` és `év`) megegyezik az u -val. Valóban van ilyen sor, mégpedig a 3.10. ábra harmadik sora.

Hasonlóan vegyük most a fenti t sort másodiknak és u legyen az első. Ekkor a többértékű függőség azt mondja ki, hogy R -ben lennie kell olyan sornak, amely a `név`, `város`, `utca` attribútumokon megegyezik a másodikkal, és a `név`, `filmcím` és `év` attribútumokon megegyezik az elsővel. Ez a sor is létezik, mégahozzá ez a 3.10. ábra második sora. \square

3.6.3. Többértékű függőségekre vonatkozó szabályok

A 3.2. alfejezetben a funkcionális függőségekre tanult levezetési szabályokhoz hasonló számos szabály van a többértékű függőségekre is. Például a többértékű függőségre teljesülnek a következők:

- *Triviális többértékű függőség.* Az

$$A_1 A_2 \cdots A_n \rightarrow\!\!\! \rightarrow B_1 B_2 \cdots B_m$$

többértékű triviális függőség fennáll bármely relációban, ha

$$\{B_1, B_2, \dots, B_m\} \subseteq \{A_1, A_2, \dots, A_n\}$$

teljesül.

- *A tranzitivitási szabály,* amely szerint ha érvényesek

$$A_1 A_2 \cdots A_n \rightarrow\!\!\! \rightarrow B_1 B_2 \cdots B_m \quad \text{és} \quad B_1 B_2 \cdots B_m \rightarrow\!\!\! \rightarrow C_1 C_2 \cdots C_k$$

többértékű függőségek egy bizonyos relációra, akkor

$$A_1 A_2 \cdots A_n \rightarrow\!\!\! \rightarrow C_1 C_2 \cdots C_k$$

is érvényes. Mindazon C -ket, amelyek egyben A -k is, törölhetjük a jobb oldalról.

A többértékű függőség viszont nem tesz eleget a szétvághatósági/összenvonhatósági szabály szétvághatóságra vonatkozó részének, mint ahogy a következő példa mutatja.

3.30. példa. Nézzük meg újból a 3.10. ábrát, amelyben korábban már láttuk, hogy teljesül az alábbi többértékű függőség:

$$\text{név} \rightarrow\!\!\! \rightarrow \text{város } \text{utca}$$

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

$$\text{név} \rightarrow\!\!\! \rightarrow \text{utca}$$

is teljesül. E többértékű függőség szerint minden színész lakkímében az *utca* független a többi attribútumtól, beleértve a *város*-t is. Azonban ez az állítás hamis. Tekintsük például a 3.10. ábra első két sorát. A feltételezett többértékű függőség arra engedne következtetni, hogy a sorok az utcákban váltakozhatnak, azaz a

<i>név</i>	<i>város</i>	<i>utca</i>	<i>filmcím</i>	<i>év</i>
C. Fisher	Hollywood	5 Locust Ln.	Csillagok háborúja	1977
C. Fisher	Malibu	123 Maple St.	Csillagok háborúja	1977

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

Vannak új szabályok is a többértékű függőségekre. Először:

- *Funkcionális függőség előléptetése.* minden funkcionális függőség egyben többértékű függőség is. Azaz, amennyiben

$$A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$$

akkor

$$A_1 A_2 \cdots A_n \rightarrow\! \rightarrow B_1 B_2 \cdots B_m$$

A bizonyításhoz legyen R egy tetszőleges reláció, amelyben

$$A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$$

fennáll, és legyenek t és u az R -nek olyan sorai, amelyek az A -kon megegyeznek. Ahhoz, hogy megmutassuk $A_1 A_2 \cdots A_n \rightarrow\! \rightarrow B_1 B_2 \cdots B_m$ érvényességét, azt kell megmutatnunk, hogy R tartalmaz egy olyan v sort is, amely az A -kon megegyezik t -vel és u -val, a B -ken t -vel, 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 feltétük, hogy ez a két sor megegyezik az A -kon. Az $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ funkcionális függőség biztosítja, hogy u megegyezik a B -ken t -vel. Természetesen a többi attribútumon pedig u megegyezik önmagával. Így ha egy funkcionális függőség fennáll, akkor a megfelelő többértékű függőség is fennáll.

- *Komplementer-szabály.* Ha $A_1 A_2 \cdots A_n \rightarrow\! \rightarrow B_1 B_2 \cdots B_m$ többértékű függőség az R relációban, akkor R kielégíti az $A_1 A_2 \cdots A_n \rightarrow C_1 C_2 \cdots C_k$ 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.

Ez azt jelenti, hogy a B -k közül két olyan sornak a cseréje, amelyek megegyeznek az A -kkal, ugyanolyan hatást vált ki, mint a C -k közötti csere.

3.31. példa. Nézzük meg újból a 3.10. ábrán látható relációt, amelyre feltétük, hogy teljesíti az alábbi többértékű függőséget:

név → város utca

A komplementer-szabály szerint a

név → filmcím év

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

Az olyan többértékű függőség, amelynek jobb oldala a bal oldal egy rész-halmaza, triviálisan fennáll minden reláció esetén. A komplementer szabály egy érdekes következménye azonban, hogy vannak egyéb olyan triviális többértékű függőségek is, amelyek nem néznek ki triviálisnak.

- *További triviális többértékű függőségek.* Amennyiben egy R reláció összes attribútuma az

$$\{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m\}$$

halmaz, akkor $A_1 A_2 \cdots A_n \rightarrow\!\!\! \rightarrow B_1 B_2 \cdots B_m$ fennáll R -en.

Annak bizonyításához, hogy ezek a további triviális többértékű függőségek fennállnak, figyeljük meg, hogy ha veszünk két olyan sort, amelyek meggyeznek az A_1, A_2, \dots, A_n attribútumokon, és a komponenseiket felcseréljük a B_1, B_2, \dots, B_m attribútumokon, akkor ugyanazt a két sort kapjuk, bár ellenkező sorrendben.

3.6.4. Negyedik normálforma

Azokat a többértékű függőségek által okozott redundanciákat, amelyeket korábban a 3.6.1. alfejezetben találtunk, megszüntethetjük, ha ezeket a függőségeket felhasználjuk a felbontásokban. Ebben az alfejezetben bevezetünk egy új normálformát, az ún. „negyedik normálformát”. Ebben a normálformában az összes „nem triviális” többértékű függőséget megszüntetjük úgy, ahogy tettek az összes olyan funkcionális függőséggel, amelyek megsértették a BCNF-et. Az eredményül kapott, felbontott relációkban nincs sem a funkcionális függőségekből származó olyan redundancia, mint amilyen a 3.3.1. alfejezetben szerepelt, sem a többértékű függőségekből származó redundancia, mint amilyent a 3.6.1. alfejezetben láttunk.

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

$$A_1 A_2 \cdots A_n \rightarrow\!\!\! \rightarrow B_1 B_2 \cdots B_m$$

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

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

3.32. példa. A 3.10. ábra relációja megséríti a 4NF feltételt. Például, a

$$\text{név} \rightarrow \text{város utca}$$

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

A negyedik normálforma valóban a BCNF általánosítása. A 3.6.3. alfejezetben láttuk, hogy minden funkcionális függőség egyben többértékű függőség is. Így minden, ami megséríti a BCNF-et, megséríti a 4NF-et is. Másképpen fogalmazva, minden 4NF-ben levő reláció emiatt BCNF-ben van.

Vannak olyan BCNF-ben levő relációk, amelyek nem 4NF-beliek. A 3.10. 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 3.32. példában láttuk, hogy nincs 4NF-ben.

3.6.5. Negyedik normálformára bontás

A 4NF dekompozíciós algoritmusa meglehetősen hasonlít a BCNF dekompozíciós algoritmusához.

3.33. algoritmus. Negyedik normálformára bontás

BEMENET: Az R_0 reláció és az S_0 funkcionális és többértékű függőségeket tartalmazó halmaz.

KIMENET: Az R_0 dekompozíciója olyan relációkra, amelyek mindegyike 4NF-ben van. A dekompozíció veszteségmentes összekapcsolási tulajdonsággal rendelkezik.

ELJÁRÁS: Végezzük el a következő lépéseket $R = R_0$ -lal és $S = S_0$ -lal:

1. Keressük a 4NF feltétel egy megsértését R -ben. Legyen ez

$$A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$$

ahol $\{A_1, A_2, \dots, A_n\}$ nem szuperkulcs. Megjegyezzük, hogy ez a többértékű függőség lehet egy valódi többértékű függőség, vagy levezethető egy megfelelő $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ S -beli funkcionális függőségből, mivel tudjuk, hogy minden funkcionális függőség egyben többértékű függőség is. Ha nincs ilyen, akkor készen vagyunk, R önmagában egy megfelelő dekompozíció.

2. Ha találunk 4NF-sértést, akkor osszuk fel a 4NF feltételt megsértő R relációt két sémára:
- R_1 , melynek a sémája tartalmazza az A -kat és a B -ket;
 - R_2 , melynek a sémája az A -kból és a sem A -ban, sem B -ben nem szereplő attribútumokból áll.
3. Keressük meg az R_1 -ben és R_2 -ben fennálló funkcionális és többértékű függőségeket (a 3.7. alfejezet bemutatja, hogyan kell ezt általános esetben végrehajtani, de gyakran a függőségek „vetítése” a legegyszerűbb). Rekurzívan dekomponáljuk R_1 -et és R_2 -t a vetített függőségeikkel.

□

3.34. példa. Folytassuk a 3.32. példát. Tapasztaltuk, hogy a

$$\text{név} \rightarrow\!\!\! \rightarrow \text{utca} \text{ } \text{város}$$

a 4NF megsértése. A dekompozíciós szabály azt mondja, hogy helyettesítsük ezt az ötattribútumú sémát egy sémával, amely a fenti függőség attribútumait tartalmazza, és egy olyannal, amely tartalmazza a bal oldali **név** attribútumot és azokat, amelyek nem fordulnak elő a függőségben. Ezek az attribútumok a **filmcím** és az **év**, tehát a dekompozíció eredményeként a következő sémák adódnak:

$$\begin{aligned}\{\text{név}, \text{utca}, \text{város}\} \\ \{\text{név}, \text{filmcím}, \text{év}\}\end{aligned}$$

Ezekben a sémákban nincs nem triviális többértékű (vagy funkcionális) függő-ség, tehát 4NF-ben vannak. Megjegyezzük, hogy a **{név, utca, város}** sémájú relációban a

$$\text{név} \rightarrow\!\!\! \rightarrow \text{utca} \text{ } \text{város}$$

többértékű függőség triviális, mivel tartalmazza az összes attribútumot. Hason-lóan a **{név, filmcím, év}** relációban a

$$\text{név} \rightarrow\!\!\! \rightarrow \text{filmcím} \text{ } \text{év}$$

többértékű függőség triviális. Ha egyik vagy minden kapott reláció nem 4NF-ben lenne, akkor folytatnunk kellene a nem 4NF-ben lévő sémák(k) felbontásával.

□

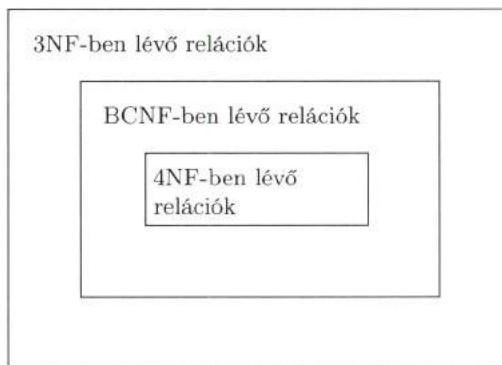
Ahogy a BCNF-felbontásnál, a felbontás minden egyes lépésében egyre keve-sebb attributumból álló sémat kapunk, mint amilyenből kiindultunk. Ily módon végül olyan sémákhoz jutunk, melyeket már nem kell tovább felbontani, mivel már 4NF-ben vannak. A 3.4.1. alfejezetben adott felbontás helyességének iga-zolása átvihető a többértékű függőségekre is. Amikor felbontunk egy relációt az

$A_1 A_2 \cdots A_n \rightarrow\!\!\! \rightarrow B_1 B_2 \cdots B_m$ többértékű függőség miatt, ez a függőség elegendő ahhoz, hogy az eredeti reláció a felbontásban szereplő relációkból visszaállítható.

A 3.7. alfejezetben megadunk egy olyan algoritmust, melyet a többértékű függőségre alkalmazva 4NF-dekompozíciót szolgáltat, és igazolja, hogy a dekompozíció veszteségmentes. Ugyanezen alfejezetben megmutatjuk ennek végrehajthatóságát, megjegyezve azonban, hogy ez időigényes, mert az MVD-knek a dekompozícióval kapott relációkra való vetítését kell végrehajtani. Ez a vetítés ahhoz szükséges, hogy eldöntsük, vajon szükséges-e további felbontás.

3.6.6. Normálformák közötti kapcsolatok

Említettük korábban, hogy a 4NF-ből következik a BCNF, amelyből következik a 3NF. A 3.12. ábrán szemléltetjük, hogy a normálformákat kielégítő függőségekkel rendelkező relációsémák halmazai között mi a kapcsolat. Ha egy reláció adott függőségekkel 4NF-ben van, akkor az egyben BCNF-ben és 3NF-ben is van. Ha egy reláció BCNF-ben van, akkor 3NF-ben is van.



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

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

3.13. ábra. A normálformák és felbontásai jellemzői

A normálformák összehasonlításának másik módja az, hogy az adott normálformára való felbontás eredményéül kapott relációk halmaza mit biztosít. Ezt a 3.13. 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 többértékű függőségek okoznak, amelyek nem funkcionális függőségek. Gyakran elegendő a 3NF ilyen redundancia megszüntetésére, de vannak olyan esetek, amikor nem. 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.

3.6.7. Feladatok

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

3.6.2. feladat. Vegyük azt a relációt, amelyben feljegyezzük a személyek nevét, személyi számát és születési dátumát. Továbbá a személy összes gyerekének a nevét, személyi számát és születési dátumát, és amennyiben autóval vagy autókkal rendelkezik a személy, az autóinak a sorozatszámát és gyártmányát. Pontosabban ennek a relácionak a sorai az összes olyan

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

sor, amelyre

1. *n* a személy neve, akinek a személyi száma *szsz*;
2. *szd* az *n* születési dátuma;
3. *gyn* az *n* egyik gyerekének neve;
4. *gysz* a *gyn* személyi száma;
5. *gyszd* a *gyn* születési dátuma;
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-ben való felbontására.

3.6.3. feladat. A következő relációsémákra és funkcionális függőségi halma-zokra nézve:

- a) $R(A, B, C, D)$ sémára és $A \rightarrow\!\!\! \rightarrow B$, $A \rightarrow\!\!\! \rightarrow C$ többértékű függőségekkel;
- b) $R(A, B, C, D)$ sémára és $A \rightarrow\!\!\! \rightarrow B$, $B \rightarrow\!\!\! \rightarrow CD$ többértékű függőségekkel;
- c) $R(A, B, C, D)$ sémára, $AB \rightarrow\!\!\! \rightarrow C$ többértékű függőséggel és $B \rightarrow D$ funk-cionális függőséggel;
- d) $R(A, B, C, D, E)$ sémára és $A \rightarrow\!\!\! \rightarrow B$ és $AB \rightarrow\!\!\! \rightarrow C$ többértékű függőségek-
kel, továbbá $A \rightarrow D$ 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 relációsémáakra.

3.6.4. feladat. Adjunk informális érveket arra, miért nem vártuk a 3.28. példa öt attribútumának egyikétől sem, hogy a másik négy funkcionálisan meghatá-rozza.

3.7. Egy algoritmus többértékű függőségek megkeresésére

Az érvelések, melyek többértékű függőségekre vagy többértékű függőségek és funkcionális függőségek együttesére vonatkoznak, sokkal bonyolultabbak, mint azok, amelyek csak funkcionális függőségekre vonatkoznak. Funkcionális függő-ségekre a 3.7. algoritmust használhatjuk, és eldönthetjük vele, hogy egy füg-gőség következik-e több megadott függőségből vagy nem. Ebben az alfejezetben először megmutatjuk, hogy a lezárási algoritmus valójában azonos a chase-eljárással, amelyet a 3.4.2. alfejezetben tanulmányoztunk. A chase mögötti ötlet kiterjeszthető többértékű függőségekre úgy, ahogyan a funkcionális függőségek-re. Lesz egy eszközünk, amivel megoldhatjuk az összes olyan problémát, amely a többértékű függőségekkel és a funkcionális függőségekkel kapcsolatos, mint például annak eldöntése, hogy egy többértékű függőség következik-e megadott függőségekből, vagy mi lesz a többértékű és funkcionális függőségek vetülete a dekompozícióval kapott relációra.

3.7.1. A chase és a lezárás

A 3.2.4. alfejezetben láttuk, hogyan kell egy adott X attribútumhalmazból ki-számítani a lezártját, X^+ -t, amely az összes olyan attribútumot tartalmazza, amely funkcionálisan függ X -től. Így tesztelhetjük, hogy egy $X \rightarrow Y$ funkcio-nális függőség következik funkcionális függőségek egy F halmazából, az X lezá-rásával F szerint, illetve megnézhetjük, hogy $Y \subseteq X^+$ teljesül-e. Tekinthetjük

a lezárást a chase egy variációjával, ahol a kiinduló tabló és a cél különböznek attól, amit a 3.4.2. alfejezetben láttunk.

Tegyük fel, hogy egy olyan tablóval kezdünk, amelynek két sora van. Ezek a sorok megegyeznek X attribútumain és eltérnek az összes többi attribútumon. Ha alkalmazzuk az F -beli funkcionális függőségeket, akkor pontosan azokban az oszlopokban fogjuk egyenlővé tenni a szimbólumokat, amelyek $X^+ - X$ -ben vannak. Így a chase-alapú teszt annak eldöntésére, hogy $X \rightarrow Y$ következik-e F -ből, az alábbiakban foglalható össze:

1. Kezdjük egy olyan tablóval, aminek két olyan sora van, amely megegyezik X attribútumain és különbözik a többin.
2. Hajtsuk végre a chase-eljárást az F függőséghalmazt használva.
3. Ha a végső tabló megegyezik Y összes oszlopán, akkor $X \rightarrow Y$ fennáll, különben pedig nem.

3.35. példa. Ismételjük meg a 3.8. példát, ahol az alábbi relációink volt:

$$R(A, B, C, D, E, F)$$

az $AB \rightarrow C$, $BC \rightarrow AD$, $D \rightarrow E$ és $CF \rightarrow B$ függőségekkel. Azt szeretnénk eldönteni, hogy $AB \rightarrow D$ fennáll-e. Indulunk ki ebből a tablóból:

A	B	C	D	E	F
a	b	c_1	d_1	e_1	f_1
a	b	c_2	d_2	e_2	f_2

Alkalmazhatjuk az $AB \rightarrow C$ -t $c_1 = c_2$ bizonyítására; legyen mondjuk mindkettő c_1 . Az eredménytábló:

A	B	C	D	E	F
a	b	c_1	d_1	e_1	f_1
a	b	c_1	d_2	e_2	f_2

Következőnek alkalmazzuk a $BC \rightarrow AD$ -t $d_1 = d_2$ bizonyítására, majd utána $D \rightarrow E$ -t, hogy belássuk: $e_1 = e_2$. Ezen a ponton a tabló:

A	B	C	D	E	F
a	b	c_1	d_1	e_1	f_1
a	b	c_1	d_1	e_1	f_2

Innen nem tudunk továbblépni. Mivel a két sor már megegyezik a D oszlopon, tudjuk, hogy $AB \rightarrow D$ következik a megadott függőségekből. \square

3.7.2. A chase kiterjesztése többértékű függőségekre

A chase során funkcionális függőségeknél alkalmazott eljárás alkalmazható többértékű függőségek bizonyítására is. Amikor megpróbálunk belátni egy funkcionális függőséget, azt kérdezzük, hogy következtethetünk-e arra, hogy két nem feltétlenül azonos értéknek valójában meg kell egyeznie. Amikor alkalmazzuk az $X \rightarrow Y$ funkcionális függőséget, olyan sorpárokat keresünk a tablóban, amelyek megegyeznek X összes oszlopán, és arra törekszünk, hogy Y minden oszlopában az értékek megegyezőek legyenek.

Azonban a többértékű függőségek nem jelentik azt, hogy bizonyos szimbólumok azonosak. $X \rightarrow Y$ inkább azt állítja, hogy ha találunk két olyan sort a tablóban, amelyek megegyeznek X -en, akkor készíthetünk két újabb sort, megcsérélve az Y attribútumokon elhelyezkedő komponenseiket; a kapott két új sornak a relációban szerepelnie kell, ezáltal a tablóban is. Hasonlóan, ha funkcionális és többértékű függőségekből le szeretnénk vezetni egy $X \rightarrow Y$ többértékű függőséget, egy kétsoros tablóval kell kezdenünk, amely két olyan sort tartalmaz, amely megegyezik az X attribútumokon és különbözik azokon, amelyek X -en kívüliek. Alkalmazzuk a megadott funkcionális függőségeket, hogy egyenlővé tegyük szimbólumokat, majd alkalmazzuk a többértékű függőségeket is, azaz megcséréljük az értékeket bizonyos attribútumokon a két előforduló sorban annak érdekében, hogy új sorokat adjunk a tablóhoz. Ha észrevesszük, hogy az eredeti sorok egyikében az Y attribútumokat kicseréljük egy másik eredeti sorból ugyanazokkal, akkor beláttuk a függőséget.

Eljött a figyelmeztetés helye ebben a lényegesen összetettebb chase-eljárásban. Mivel szimbólumokat egyenlővé tudunk tenni, és ki tudjuk cserélni más szimbólumokkal, nem biztos, hogy felismerjük, hogy elkészítettünk egyet a kívánt sorból, mert néhány eredeti szimbólumot esetleg újakra cserélünk. A legegyszerűbb út ennek a problémának a megelőzésére, ha a célsort az inicializáláskor definiáljuk, és később nem változtatjuk a szimbólumait. Azaz legyen a célsor egy olyan sor, amelynek minden egyes komponense egy nem indexelt betű. Legyen két kezdősor az $X \rightarrow Y$ függőség teszteléséhez, amely nem indexelt betűket tartalmaz X attribútumain. Legyen az első sor olyan, hogy nincs indexük az Y attribútumain és legyen a második olyan, hogy nem indexelt betűk vannak az X -en és Y -on kívüli összes attribútumon. A két sorban fennmaradó helyeket töltük ki új szimbólumokkal, amelyek mindegyike csak egyszer fordul elő. Amikor egyenlővé teszünk indexelt és nem indexelt szimbólumokat, akkor az indexelt helyére tegyük a nem indexeltet, ahogyan a 3.4.2. alfejezetben is tettük. Ezután, ha alkalmazzuk a chase-eljárást, csak azt a kérdést kell megválaszolnunk, hogy előfordul-e az a sor a tablóban, amelynek minden eleme indexletlen.

3.36. példa. Tegyük fel, hogy adott az $R(A, B, C, D)$ reláció és $A \rightarrow B$, $B \rightarrow C$ függőségek. Azt szeretnénk beláttni, hogy $A \rightarrow C$ fennáll R -ben. Kezdjük egy kétsoros tablóval, amely $A \rightarrow C$ -t reprezentálja:

A	B	C	D
a	b_1	c	d_1
a	b	c_2	d

Megjegyezzük, hogy a célsorunk az (a, b, c, d) . A tabló minden sorában az A attribútumon index nélküli elemet tartalmaz. Az első sorban index nélküli elem van a C oszlopból, míg a második sorban a fennmaradó helyeken.

Először az $A \rightarrow B$ függőséget fogjuk alkalmazni annak bizonyítására, hogy $b = b_1$. Tehát le kell cserélnünk az indexelt b_1 -et az index nélküli b -re. A keletkező tabló:

A	B	C	D
a	b	c	d_1
a	b	c_2	d

Következőnek a $B \rightarrow C$ többértékű függőséget alkalmazzuk, mivel a két sor megegyezik a B oszlopon. A C oszlopokat cserélgetjük, hogy új sorokat adhassunk a tablóhoz, ami az alábbihoz vezet:

A	B	C	D
a	b	c	d_1
a	b	c_2	d
a	b	c_2	d_1
a	b	c	d

Ezzel megkapjuk a csupa index nélküli szimbólumokból álló sort, ami bizonyítja, hogy $A \rightarrow C$ fennáll az R relációban. Nézzük meg, hogy a tablómanipulációk tényleg bizonyítják, hogy $A \rightarrow C$ fennáll. A bizonyítás a következő: „Adott két sor, amelyek megegyeznek A -n. $A \rightarrow B$ miatt meg kell egyezniük B -n is. Mivel megegyeznek B -n, a $B \rightarrow C$ miatt megcserélhetjük a C -komponenseket, és a kapott sorok R -beliek lesznek. Azaz, ha két sor megegyezik A -n, akkor azok a sorok, amelyeket a C attribútumok megcserélésével kapunk, szintén R -beliek lesznek, azaz $A \rightarrow C$.“ \square

3.37. példa. A funkcionális és többértékű függőségekkel kapcsolatosan van egy meglepő szabály, amely azt mondja ki, hogy ha az $X \rightarrow Y$ többértékű függőség fennáll, továbbá van egy olyan funkcionális függőség, amelynek a jobb oldala Y nem feltétlenül teljes részhalmaza, amit nevezünk Z -nek, akkor $X \rightarrow Z$ is fennáll. A chase-eljárást fogjuk használni arra, hogy egy egyszerű példáját bizonyítsuk ennek a szabálynak. Legyen a megadott relációink $R(A, B, C, D)$ az $A \rightarrow BC$ többértékű függőséggel és a $D \rightarrow C$ funkcionális függőséggel. Azt állítjuk, hogy $A \rightarrow C$.

Mivel egy funkcionális függőséget próbálunk belátni, az index nélküli elemeket tartalmazó célrekord elérésével nem kell foglalkoznunk. Bármely két olyan rekordból kiindulhatunk, amelyek megegyeznek A -n, és különböznek az összes többi összetevőben. Például:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

A célunk annak belátása, hogy $c_1 = c_2$.

Az egyetlen dolog, amivel kezdhetünk, hogy alkalmazzuk az $A \rightarrow BC$ többértékű függőséget, mivel a két sor megegyezik A -n, de semmilyen más oszlopon nem. Amikor megcseréljük a B és C attribútumokat ebben a két sorban, két újabb sort kell felvennünk:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2
a	b_2	c_2	d_1
a	b_1	c_1	d_2

Most olyan párokat kaptunk, amelyek megegyeznek a D oszlopban, tehát alkalmazhatjuk a $D \rightarrow C$ funkcionális függőséget. Például az első és a harmadik sornak is d_1 a D -értéke, tehát alkalmazhatjuk a függőséget $c_1 = c_2$ levezetésére. Ez volt a célunk, azaz bizonyítani, hogy $A \rightarrow C$ fennáll. A tábló az alábbi:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_1	d_2
a	b_2	c_1	d_1
a	b_1	c_1	d_2

Végül semmilyen más módosítás nem lehetséges a megadott függőségekkel. Ugyanakkor ez nem is probléma, mert a szükséges bizonyítást már elvégeztük.
 \square

3.7.3. Miért működik a chase a többértékű függőségekre?

Az okoskodás lényegében most is ugyanaz, mint amit az előbb láttunk. A chase minden lépése, legyen az szimbólumok egyenlővé tétele vagy új sorok generálása, egy igaz észrevétel az R sorairól, amelyet az adott lépésben alkalmazott funkcionális vagy többértékű függőséggel tudunk igazolni. Azaz a chase pozitív eredménye minden bizonyítja, hogy a megadott funkcionális és többértékű függőségek fennállnak R -ben.

Amikor a chase hibával végződik – azaz a célsor (többértékű függőségeknél) vagy az elvárt egyenlőségek (funkcionális függőségeknél) nem állnak elő –, akkor a befejező lépéskor létrejött tábló egy ellenpélda. Kielégíti a megadott függőségeket, különben tudtunk volna még változtatásokat végrehajtani. Ugyanakkor nem elégíti ki a belátni próbált függőséget.

Van egy másik probléma is, ami nem jött elő akkor, amikor csak funkcionális függőségekkel használtuk a chase-t. Amióta a többértékű függőségek új sorokat

adnak a tablóhoz, honnan tudhatjuk, hogy a chase-eljárás be fog fejeződni? Adhatunk örökké hozzá új sorokat úgy, hogy soha nem érjük el a célunkat, de nem lehetünk biztosak abban, hogy még néhány lépés múlva elérhetjük? Szerencsére ez nem történhet meg. Ez azért van így, mert sosem készítünk új szimbólumokat. Induláskor legfeljebb két különböző szimbólumunk van a k oszlop mindegyikében. Azaz sosem lehet 2^k -nál több sorunk a tablóban, ha k az oszlopok száma. A chase többértékű függőségekkel eltarthat exponenciális ideig, de sosem fut a végtelenséggel.

3.7.4. Többértékű függőségek vetítése

Idézzük fel, hogy a többértékű függőségek bizonyítása azért kellett, mert ez vezetett el a relációk 4NF-felbontásához. A feladat befejezéséhez képesnek kell lennünk arra, hogy a megadott függőségeket levetítsük a két relációsémára, amelyet a dekompozíció első lépésében kapunk. Csak ekkor tudhatjuk meg, hogy a relációk 4NF-ben vannak, vagy esetleg újabb dekompozícióra van szükség.

Legrosszabb esetben ki kell próbálnunk minden lehetséges funkcionális és többértékű függőséget a felbontott relációkra. A chase-tesztet alkalmaztuk az eredeti reláció összes attribútumán. Mivel a cél egy többértékű függőség esetében egy olyan sor előállítása a tablóban, amely index nélküli betűket tartalmaz a dekompozíció egyik relációjának összes oszlopában, ebben a sorban a többi helyen bármilyen szimbólum szerepelhet. A funkcionális függőség esetében a cél ugyanez: a szimbólumok egyezése egy megadott oszlopanban.

3.38. példa. Adott az $R(A, B, C, D, E)$ reláció, amit felbontunk, és az egyik, dekompozícióval létrejövő reláció legyen $S(A, B, C)$. Tegyük fel, hogy az $A \rightarrow\!\! \rightarrow CD$ függőség fennáll R -ben. Implikál ez a függőség bármilyen függőséget S -ben? Azt állítjuk, hogy $A \rightarrow\!\! \rightarrow C$ fennáll S -ben csakúgy, mint $A \rightarrow\!\! \rightarrow B$ is (a komplementer-szabály miatt). Lássuk be, hogy $A \rightarrow\!\! \rightarrow C$ fennáll S -ben. Az alábbi tablóval kezdünk:

A	B	C	D	E
a	b_1	c	d_1	e_1
a	b	c_2	d	e

Alkalmazzuk az $A \rightarrow\!\! \rightarrow CD$ függőséget, és cseréljük meg a C és D komponenseket ebben a két sorban, hogy két új sort kapunk:

A	B	C	D	E
a	b_1	c	d_1	e_1
a	b	c_2	d	e
a	b_1	c_2	d	e_1
a	b	c	d_1	e

Megjegyezzük, hogy az utolsó sor az S attribútumain, azaz A -n, B -n és C -n, csak indexeletlen szimbólumokat tartalmaz. Ez elegendő ahhoz, hogy belássuk, $A \rightarrow\!\! \rightarrow C$ fennáll S -re. \square

Gyakran a funkcionális és többértékű függőségek megkeresésének a vetületrelációkban nem kell teljesen kimerítőnek lennie. Alább láthatunk néhány egyszerűsítést.

1. Biztosan nem kell ellenőrizni a triviális funkcionális és többértékű függőségeket.
2. Funkcionális függőségek esetében nem kell azokra szorítkoznunk, amelyeknek egy attribútuma van a jobb oldalon, a kombinációs szabály miatt.
3. Egy funkcionális vagy egy többértékű függőség, amelynek bal oldala nem tartalmazza egyetlen adott függőség bal oldalát sem, biztosan nem áll fenn, mivel seholgyan sem tudjuk elkezdeni vele a chase-eljárást. Azaz a két sor, melyekkel a tesztet kezdenénk, nem változik a megadott függőségekkel.

3.7.5. Feladatok

3.7.1. feladat. Használjuk a chase-tesztet annak előntésére, hogy az alábbi függőségek fennállnak-e az $R(A, B, C, D, E)$ reláión az $A \rightarrow\!\!\! \rightarrow BC$, $B \rightarrow D$ és $C \rightarrow\!\!\! \rightarrow E$ függőségek mellett?

- a) $A \rightarrow D$
- b) $A \rightarrow\!\!\! \rightarrow D$
- c) $A \rightarrow E$
- d) $A \rightarrow\!\!\! \rightarrow E$

! 3.7.2. feladat. Ha a 3.7.1. feladatban látható R relációt levetítjük az $S(A, C, E)$ -re, akkor mely nem triviális funkcionális és többértékű függőségek érvényesek S -ben?

! 3.7.3. feladat. Lássuk be a következő szabályokat a többértékű függőségekre. minden egyes esetben a bizonyítás megadható a chase-teszttel, de egy kicsit általánosabban kell gondolkozni, mint a látott példákban, mivel az X, Y, Z téteszüleges attribútumhalmazok, továbbá vannak egyéb, név nélküli attribútumok a relációban, amelyekre a függőségek érvényesek.

- a) *Az unió szabálya.* Ha X, Y és Z attribútumhalmazok, továbbá $X \rightarrow\!\!\! \rightarrow Y$ és $X \rightarrow\!\!\! \rightarrow Z$, akkor $X \rightarrow\!\!\! \rightarrow (Y \cup Z)$.
- b) *A metszet szabálya.* Ha X, Y és Z attribútumhalmazok, továbbá $X \rightarrow\!\!\! \rightarrow Y$ és $X \rightarrow\!\!\! \rightarrow Z$, akkor $X \rightarrow\!\!\! \rightarrow (Y \cap Z)$.
- c) *A különbség szabálya.* Ha X, Y és Z attribútumhalmazok, továbbá $X \rightarrow\!\!\! \rightarrow Y$ és $X \rightarrow\!\!\! \rightarrow Z$, akkor $X \rightarrow\!\!\! \rightarrow (Y - Z)$.

- d) A közös attribútumok elhagyása a jobb oldalról. Ha $X \rightarrow\!\!\! \rightarrow Y$ fennáll, akkor $X \rightarrow\!\!\! \rightarrow (Y - X)$ is fennáll.

! 3.7.4. feladat. Adjunk ellenpéldaként olyan relációkat, amelyek megmutatják, hogy a következő szabályok a többértékű függőségekre *nem* igazak. Útmutatás: alkalmazzuk a chase-tesztet, és figyeljük meg, mi történik.

- Ha $A \rightarrow\!\!\! \rightarrow BC$, akkor $A \rightarrow\!\!\! \rightarrow B$.
- Ha $A \rightarrow\!\!\! \rightarrow B$, akkor $A \rightarrow B$.
- Ha $AB \rightarrow\!\!\! \rightarrow C$, akkor $A \rightarrow\!\!\! \rightarrow C$.

3.8. Összefoglalás

- ◆ *Funkcionális függőségek:* A funkcionális függőség egy állítás, amely azt mondja, hogy ha egy reláció két sora megegyezik néhány megadott attribútumon, akkor megegyezik attribútumok egy másik halmazán is.
- ◆ *Relációk kulcsa:* Egy reláció szuperkulcsa egy olyan attribútumhalmaz, amely funkcionálisan meghatározza a reláció többi attribútumát. A kulcs olyan szuperkulcs, melynek egyik valódi részhalmaza sem szuperkulcs.
- ◆ *Funkcionális függőségek bizonyítása:* Számos olyan szabály van, amellyel beláthatjuk, hogy egy $X \rightarrow A$ funkcionális függőség fennáll minden olyan előfordulásban, amely megfelel bizonyos funkcionális függőségeknek. Az $X \rightarrow A$ igazolásához számoljuk ki X lezártját úgy, hogy a funkcionális függőségek felhasználásával addig bővítjük X -et, amíg nem tartalmazza A -t.
- ◆ *Minimális bázis funkcionális függőségek egy halmazához:* Bármely függősséghalmazhoz létezik legalább egy minimális bázis, amely az eredetivel ekvivalens funkcionális függőségekből álló halmaz (azaz egyik halmazból következik a másik). A függőségek jobb oldalain egyetlen attribútum áll, egyetlen szabály sem hagyható el az ekvivalencia megtartásával, és a bal oldalakról sem hagyható el egyetlen attribútum sem az ekvivalencia megtartása mellett.
- ◆ *Boyce–Codd normálforma:* Egy reláció BCNF-ben van, ha a nem triviális függőségek csak olyanok, melyekben egy szuperkulcs meghatároz egy vagy több más attribútumot. A BCNF legnagyobb előnye, hogy kiküszöböli azokat a redundanciákat, amelyeket a funkcionális függőségek jelenléte okoz.
- ◆ *Veszeségmentes összekapcsolással rendelkező dekompozíció:* A dekompozíció egy másik hasznos tulajdonsága, hogy az eredeti reláció a felbontásban szereplő relációk természetes összekapcsolásával előállítható. Bármely

4. fejezet

Magas szintű adatbázismodellek

Vizsgáljuk meg azt a folyamatot, ahogy egy új adatbázist – mint például a filmadatbázisunkat – létrehozunk. Ezt a folyamatot a 4.1. ábra szemlélteti. A tervezési fázissal kezdjük, amelyben a következő kérdéseket tesszük fel és válaszoljuk meg: milyen információkat kell tárolni, mely információelemek kapcsolódnak egymáshoz, milyen megszorításokat (például kulcs vagy a hivatkozási épség megszorítása) kell figyelembe venni és így tovább. Ebben a fázisban kell a lehetőségeket értékelni, az elképzeléseket összeegyeztetni – ez sokáig tarthat. A 4.1. ábra azt a folyamatot mutatja be, ahogy az elgondolásokat magas szintű tervvé alakítjuk.



4.1. ábra. Az adatbázis-modellezés és implementálás eljárása

Mivel a kereskedelmi adatbázisrendszerek nagy többsége a relációs modellt használja, élhetünk azzal a feltevéssel, hogy a tervezési fázisban ezt a modellt is figyelembe vegyük. A gyakorlatban azt tapasztaljuk, hogy gyakran egyszerűbb, ha a magas szintű modell megalkotásával kezdjük, majd ezt alakítjuk át a relációs modell tervévé. Annak, hogy így járunk el, az elsődleges oka az, hogy a relációs modell csak egyetlen – a reláció – fogalmát tartalmazza, és nem foglalkozik egy sor másik, a valós világhoz sokkal közelebb álló modellel. A relációs koncepció egyszerűsége a modell nagy erőssége, amely az adatbázis-műveletek megvalósításának hatékonyságában jelenik meg. Ez az erősség az előzetes tervezéskor viszont gyengeséggé válik, és ez az, amiért általában hasznos, ha a magas szintű tervezéssel kezdjük.

A tervezési módszerek megnevezésében gyakran utalnak az alkalmazott módszerre is. Az első és legrégebbi módszer az „egyed-kapcsolat diagram”, a 4.1. alfejezetben mi is ezzel kezdjük a módszerek bemutatását. Az aktuális irányzat az UML (Unified Modeling Language – Egységesített Modellező Nyelv) használata, melyet eredetileg az objektumorientált szoftvertervezéshez szántak, de az adatbázissémák leírására is alkalmassá tették. Ezt a modellt a 4.7. alfejezetben fogjuk bemutatni. Végül a 4.9. alfejezetben az ODL-t (Object Description Language – Objektum Leíró Nyelv) tanulmányozzuk, amely az adatbázisoknak osztályok és ezek objektumai összességeből álló rendszerként való leírására alkalmas.

A 4.1. ábrán látható következő fázis a magas szintű terv átalakítása relációs tervvé. Erre a lépésre csak akkor kerül sor, ha a magas szintű tervet már elfogadtuk. Bármelyik magas szintű modellt is használtuk, teljesen mechanikus lehetőségünk van arra, hogy a magas szintű tervet olyan relációs adatbázissémává alakítsuk, amely a kereskedelmi adatbázisrendszerekben alkalmazható lesz. Az E/K-diagramok relációs adatbázissémává való átalakításával a 4.5. és 4.6. alfejezetekben foglalkozunk. Az UML átalakítását a 4.8., az ODL használatát a 4.10. alfejezetekben tárgyaljuk.

4.1. Az egyed-kapcsolat (E/K) modell elemei

Az *E/K-modellben* az adatok szerkezetét grafikusan ábrázoljuk, ún. egyed-kapcsolat diagramként. Három alapvető eleme lehet egy ilyen diagramnak:

1. Egyedhalmazok,
2. Attribútumok,
3. Kapcsolatok.

A következőkben ezeket részletezzük.

4.1.1. Egyedhalmazok

Az *egyed* valamilyen típusú absztrakt objektum. Hasonló egyedekek összessége *egyedhalmazt* alkot. Az egyed fogalma bizonyos értelemben hasonlít az objektumorientált programozás objektumfogalmához; hasonló viszony mondható el egy egyedhalmaz és objektumok egy osztálya között. Ugyanakkor az E/K-modell statikus fogalom, mely csupán az adatok szerkezetéről szól, a rajtuk végezhető műveletekről nem. Így tehát az osztályuktól eltérően, az egyedhalmazokhoz nem tartoznak metódusok.

4.1. példa. Példaként tekintsük a filmekről szóló példaadatbázisunkat. minden film egy-egy egyed, és a filmek összessége egyedhalmazt alkot. Hasonlóan a színészek is egyedek, összességük egyedhalmazt alkot. Más típusú egyed a stúdió, és így a stúdiók halmaza lesz a harmadik egyedhalmaz, mely a további példákban szerepelni fog. □

Az E/K-modell változatai

Az E/K egyes változataiban az attribútumok típusai lehetnek:

1. atomiak, mint ahogy az itt bemutatott változatban,
2. rekordok, adott számú atomi komponensből felépítve (a C nyelvű „struct”-hoz hasonlóan),
3. atomi vagy rekord típusú elemekből álló halmazok, adott elemtípus-sal.

Például egy attribútum típusa lehetne párok halmaza, melyek első eleme egész szám, második eleme karakterSORozat.

4.1.2. Attribútumok

Az egyedhalmazokhoz *attribútumok* tartoznak, melyek az egyedek tulajdonságait írják le. A *Filmek* egyedhalmaz attribútuma lehet például a *filmek címe* vagy percekben mért *hossza*. Abban semmi meglepő nincs, hogy a *Filmek* egyedhalmaz attribútumai hasonlítanak a *Filmek* reláció attribútumaihoz. Az egyedhalmazokat általában relációkkal valósítjuk meg, azonban a végső relációs terv nem mindegyik relációja származik egyedhalmazból.

Az E/K-modell itt bemutatott változatában az attribútumok értékeit atomiaknak tekintjük (például karakterSORozatok, egész, illetve lebegőpontos számok). Az E/K-modell egyes változataiban korlátozott mértékben lehetőség van arra, hogy az attribútumoknak belső struktúrát adjunk meg (lásd az ide vonatkozó keretes frást).

4.1.3. Kapcsolatok

A *kapcsolatok* két vagy több egyedhalmazt kötnek össze egymással. Például a *Filmek* és a *Színészek* egyedhalmazok között felírható a *SzerepelBenne* kapcsolat: ha egy *m* filmben egy *s* színész szerepel, akkor az *m* filmegyed az *s* színészegyeddrel részt vesz a *SzerepelBenne* kapcsolatban. A kapcsolatok leggyakrabban binárisak, azaz két egyedhalmazt kötnek össze. Az E/K-modell azonban lehetővé teszi, hogy tetszőleges számú egyedhalmazt kapcsolunk össze egyetlen kapcsolattal. Az ilyen, többágú kapcsolatokkal majd a 4.1.7. alfejezetben foglalkozunk.

4.1.4. Egyed-kapcsolat diagramok

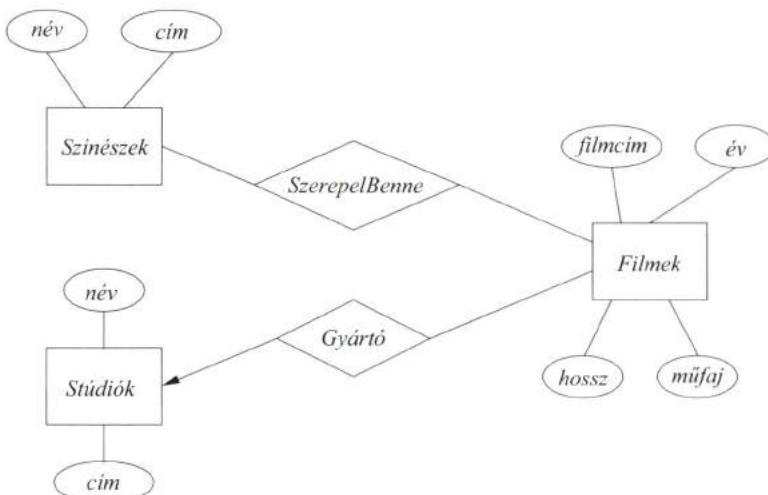
Az *egyed-kapcsolat diagram* egy gráf, melynek csúcsPontjai egyedhalmazoknak, attribútumoknak és kapcsolatoknak felelnek meg. E különböző típusú elemeket

különféle alakzatokkal ábrázoljuk, az alábbiak szerint:

- az egyedhalmazt téglalappal,
- az attribútumot oválissal,
- a kapcsolatot pedig rombusszal jelöljük.

Az egyedhalmazokat összekötjük attribútumaikkal, a kapcsolatokat pedig a benne részt vevő egyedhalmazokkal. Ezek az összekötések a gráf élei.

4.2. példa. A 4.2. ábrán egy egyszerű, filmkről szóló adatbázis E/K-diagramja látható. Az egyedhalmazok: *Filmek*, *Színészek*, *Stúdiók*.



4.2. ábra. A filmadatbázis egyed-kapcsolat diagramja

A *Filmek* egyedhalmaznak négy attribútuma van: *filmcím*, *év* (készítés ideje), *hossz*, *műfaj*. A másik két egyedhalmaz (*Színészek* és *Stúdiók*) két-két attribútuma megegyezik: *név* és *cím*, jelentésük értelemszerű. Ezenkívül két kapcsolat is látható a diagramon:

1. A *SzerepelBenne* kapcsolat hozzárendeli a filmekhez a benne szereplő színészeket. Másik irányból tekintve egy-egy színészhez rendeli hozzá azokat a filmeket, melyekben az illető szerepel.
2. A *Gyártó* kapcsolat az egyes filmeket összekapcsolja a gyártójukkal. A 4.2. ábrán a *Stúdiók* egyedhalmaz irányába mutató nyíl azt fejezi ki, hogy minden filmhez csak egy gyártó stúdió tartozik. Az ehhez hasonló egyértelműségi megszorítást a 4.1.6. alfejezetben tárgyaljuk.

4.1.5. Az E/K -diagram előfordulásai

Az egyed-kapcsolat diagram az adatbázis *sémájának*, azaz szerkezetének jelölésére szolgál. Egy E/K -diagram által leírt adatbázist aktuális adataival együtt az adatbázis előfordulásának nevezük. Minthogy az E/K -diagram az adatbázisnak csak a terve, nem pedig a megvalósítása, így az előfordulás nem egy létező dolog abban az értelemben, mint ahogy az adatbázisrendszerben a relációk előfordulása létezik. Gyakran mégis hasznos, ha úgy ábrázoljuk, mintha létező dolog lenne.

Egy előfordulásban például egy adott egyedhalmazra nézve konkrét egyedek véges halmaza szerepel. Az egyedek mindenike konkrét értékkel rendelkezik az összes attribútumon. Egy n ágú R kapcsolatnak, mely az E_1, E_2, \dots, E_n egyedhalmazokat köti össze, az előfordulásban (e_1, e_2, \dots, e_n) sorok véges halmaza felel meg, melyekben minden e_i -t a neki megfelelő E_i egyedhalmaz aktuális előfordulásából választunk ki. minden ilyen sort úgy értünk, hogy R „összekapcsolja” a benne szereplő egyedeket.

E sorok halmaza az R aktuális előfordulásának *kapcsolathalmaza*. Gyakran segít, ha a kapcsolathalmazt táblázatként ábrázoljuk. A táblázat oszlopait a kapcsolatban részt vevő egyedhalmazok neveivel címkézzük meg, és a táblázat egy-egy sorát az előfordulásban szereplő egy-egy sor alkotja. Látjuk majd, hogy amikor a kapcsolatot relációvá alakítjuk, akkor az eredményreláció nem pontosan az, mint a kapcsolathalmaz.

4.3. példa. A *SzerepelBenne* kapcsolat egy előfordulását alkotó párok, táblázatos formában például az alábbiak szerint ábrázolhatók:

Filmek	Színészek
Elemi ösztön	Sharon Stone
Emlékmás	Arnold Schwarzenegger
Emlékmás	Sharon Stone

A kapcsolathalmaz elemei a táblázat sorai. Például a (Elemi ösztön, Sharon Stone) pár a *SzerepelBenne* kapcsolat aktuális előfordulásának egy eleme. \square

4.1.6. Bináris E/K -kapcsolatok típusai

Általában egy bináris kapcsolat az egyik egyedhalmaz egy eleméhez tetszőleges számú egyedet kapcsolhat a másik egyedhalmazból. Ugyanakkor erre a „számos-ságra” (multiplicitásra) vonatkozóan gyakran teszünk megszorítást. Tegyük fel, hogy R egy kapcsolat az E és F egyedhalmazok között. Ekkor:

- Ha E minden eleme R -en keresztül legfeljebb egy F -beli elemhez kapcsolódhat, akkor R -et *sok-egy* kapcsolatnak nevezzük E -ből F -be. Vegyük észre, hogy ez esetben F egyedei tetszőleges számú E -beli egyedhez kapcsolódhatnak. Hasonlóan, ha F minden egyede R -en keresztül legfeljebb egy E -beli egyedhez kapcsolódhat, akkor R sok-egy kapcsolat F -ből E -be (más szóval *egy-sok* kapcsolat E -ből F -be).

- Ha R sok-egy kapcsolat egyszerre E -ből F -be és F -ből E -be is, akkor R -et *egy-egy* kapcsolatnak nevezzük. Az egy-egy kapcsolat bármely egyedhalmazának egy eleme a másik egyedhalmaz legfeljebb egy eleméhez kapcsolódhat.
- Ha R nem sok-egy kapcsolat E -ból F -be és F -ből E -be sem az, akkor R -et *sok-sok* kapcsolatnak nevezzük.

Mint a 4.2. példában említettük, a kapcsolatok típusát nyilak használatával jelölhetjük az E/K-diagramban. Ha egy kapcsolat sok-egy típusú az E egyedhalmaz és az F egyedhalmaz között, akkor egy nyíl mutat az F -be. A nyíl azt jelenti, hogy minden E -beli egyedhez legfeljebb egy F -beli egyed kapcsolódik. Ha E felé viszont nem nyíl, hanem egyszerű vonal vezet, akkor egy F -beli egyedhez sok E egyed kapcsolódhat.

4.4. példa. A fentiek értelmében, ha egy-egy kapcsolat van E és F egyedhalmazok között, akkor nyíl mutat minden egyedhalmazba. Például a 4.3. ábrán látható két egyedhalmaz a *Stúdiók* és *Elnökök*, illetve az *Irányít* kapcsolat (az attribútumokat elhagytuk). Feltételezhetjük, hogy egy elnök csak egy stúdiót vezet és egy stúdiónak csak egy elnöke van, így ez egy-egy kapcsolat, amelyet a két nyíl jelez.



4.3. ábra. Példa egy-egy kapcsolatra

Emlékeztetünk, hogy a nyíl „legfeljebb egy”-et jelent, azaz nem garantálja, hogy minden egyed kapcsolatban is áll a másik egyedhalmaz valamelyikével, amerre a nyíl mutat. A 4.3. ábrán például egy elnök bizonyára minden hozzá van rendelve egy stúdióhoz, mivel máskülönben nem lehetne elnök. Fordított irányban azonban lehetséges, hogy egy stúdiónak egy adott időszakban nincs elnöke, így az *Irányít*tól az *Elnökök* felé mutató nyíl valóban „legfeljebb egy”-et jelent, nem pedig „pontosan egy”-et. Ezt a különbséget részletesebben a 4.3.3. alfejezetben tárgyaljuk. □

4.1.7. Sokágú kapcsolatok

Az E/K-modellben lehet olyan kapcsolatokat definiálni, amelyek több egyedhalmazt kapcsolnak össze. A gyakorlatban a hármas és magasabb fokú kapcsolatok ritkák, bár néha szükségesek ahhoz, hogy kifejezzük a dolgok valódi állását. A sokágú kapcsolatokat az E/K-diagramban a kapcsolatot reprezentáló rombuszból kiinduló és a kapcsolatban részt vevő egyedekhez vivő vonalakkal jelöljük.

4.5. példa. A 4.4. ábrán a *Szerződik* kapcsolatban részt vesznek a stúdiók, a színészek és a filmek. Ez a kapcsolat reprezentálja, hogy egy stúdió mely színéssel, melyik filmre kötött szerződést. Mint korábban a 4.1.5. alfejezetben

Következtetési lehetőségek kapcsolattípusok viszonya alapján

Figyeljük meg, hogy a sok-egy kapcsolat speciális esete a sok-sok kapcsolatnak, és az egy-egy kapcsolat speciális esete a sok-egy kapcsolatnak. Azaz a sok-sok kapcsolatok hasznos tulajdonságaival rendelkeznek a sok-egy kapcsolatok, és a sok-egy kapcsolatok hasznos tulajdonságaival pedig rendelkeznek az egy-egy kapcsolatok. Például a sok-egy kapcsolatokhoz alkalmazott adatstruktúra ugyanúgy működik az egy-egy kapcsolatknál, habár ugyanez nem biztos, hogy működik a sok-sok kapcsolatok esetén.



4.4. ábra. Egy háromágú kapcsolat

tárgyaltuk, egy E/K-kapcsolat tekinthető egy kapcsolathalmaznak, amelynek az elemei a részt vevő egyedhalmazok egyedeiből képzett n -esek. Így a *Szerződik* kapcsolat a következő alakú hármasokkal írható le: (stúdió, színész, film).

A sokágú kapcsolatokban, ha egy nyíl egy E egyedhalmazra mutat, akkor az azt jelenti, hogy ha kiválasztunk a többi egyedhalmazból egy-egy egyedet, akkor a kapcsolatban ezekhez az egyedekhez legföljebb egy egyed tartozik az E egyedhalmazból. (Megjegyezzük, hogy ez a szabály általánosítja a kétágú kapcsolatoknál bevezetett jelölést.) Ezt funkcionális függőségnek is tekinthetjük, a függőség jobb oldalán E , bal oldalán az összes többi egyedhalmazok állnak.

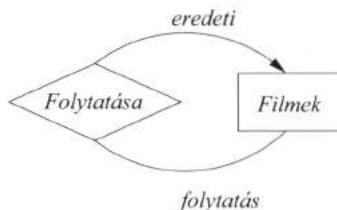
A 4.4. ábrán a *Stúdiók* egyedhalmazra mutat egy nyíl, ami azt jelenti, hogy csak egy stúdió létezik, amely egy színéssel egy adott filmre szerződést kötött. Azonban nem mutat nyíl a *Színészek* egyedhalmazra vagy a *Filmek* egyedhalmazra sem. Azaz egy stúdió szerződést köthet több színéssel is egy filmre, illetve egy színéssel több filmre is szerződhet. □

4.1.8. Szerepek a kapcsolatokban

Lehetséges, hogy egy egyedhalmaz kétszer vagy többször is szerepel egy kapcsolatban. Ha így van, akkor annyi vonalnak kell jönnie a kapcsolatból az egyedhalmazhoz, ahányszor az részt vesz a kapcsolatban. minden vonal az egyedhalmaz egy másik szerepét mutatja a kapcsolatban. Ezért meg kell címkézni a kapcsolat és az egyedhalmaz közötti éleket, ezeket nevezzük „szerepeknek”.

Korlátozások a nyíl jelölésre a sokágú kapcsolatokban

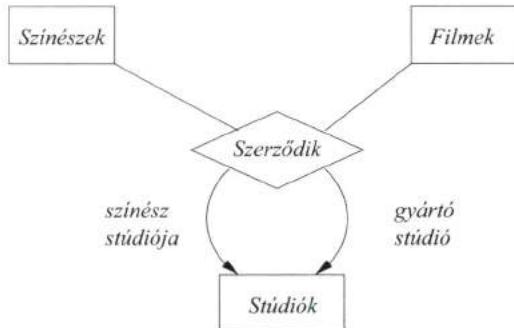
Nincs elegendő lehetőség a nyíl és nem nyíl választásban a három vagy több résztvevők kapcsolatok esetében. Így nem tudjuk leírni az összes lehetőséget nyíakkal. Például, a 4.4. ábrán a stúdió valójában csak a filmnek függvénye, és nem a színésznek és a filmnek együtt, mivel csak egy stúdió gyárt egy adott filmet. Azonban a jelölésünkben ezt nem tudjuk megkülönböztetni. Egy háromágú kapcsolat esetén az egyik egyedhalmazra mutató nyíl azt jelenti, hogy ennek az egyedhalmaznak az egyedei a másik két egyedhalmaz egyedeinek függvényei. Amennyiben minden lehetséges szituációt kezelni akarunk, funkcionális függőségek halmazát kell megadnunk a kapcsolatban részt vevő egyedhalmazok között.



4.5. ábra. Egy kapcsolat szerepekkel

4.6. példa. A 4.5. ábrán a *Folytatása* kapcsolat a *Filmek* egyedhalmaznak önmagával való kapcsolata. Két film között lehet kapcsolat úgy, hogy az egyik folytatása a másiknak. A kapcsolatban álló két film megkülönböztetésére az éleket a szerepekkel megcímkezzük. Az egyik, amely az *Eredeti* szerepben van és az eredeti filmet mutatja, a másik, amely *Folytatás* szerepben van és a folytatást reprezentálja. Feltesszük, hogy egy filmnek lehet több folytatása, de minden folytatásnak csak egy eredetije van. Így kapunk a 4.5. ábrán egy sok-egy kapcsolatot. □

4.7. példa. Záró példánkban egyaránt szerepel sokágú kapcsolat, és egy egyedhalmaz több szerepben. A 4.6. ábrán a 4.5. példában bevezetett *Szerződik* kapcsolat egy bonyolultabb verziója látható. Most a *Szerződik* kapcsolatban kétszer szerepel a stúdió, egyszer a színész és egyszer a film. Ez azt reprezentálja, hogy egy stúdió, amelyiknek van egy színéssel szerződése (általánosságban, nem egy adott filmre vonatkozóan), megengedi, hogy a színésze szerződést kössön egy másik stúdióval egy filmre. Így a kapcsolat a következő alakú négyesekkel írható le: (stúdió1, stúdió2, színész, film), amely azt jelenti, hogy a stúdió2 szerződést köt a stúdió1-gel arra, hogy stúdió2 szerepelteti a stúdió1 színészét az általa gyártott filmben.



4.6. ábra. Egy négyágú kapcsolat

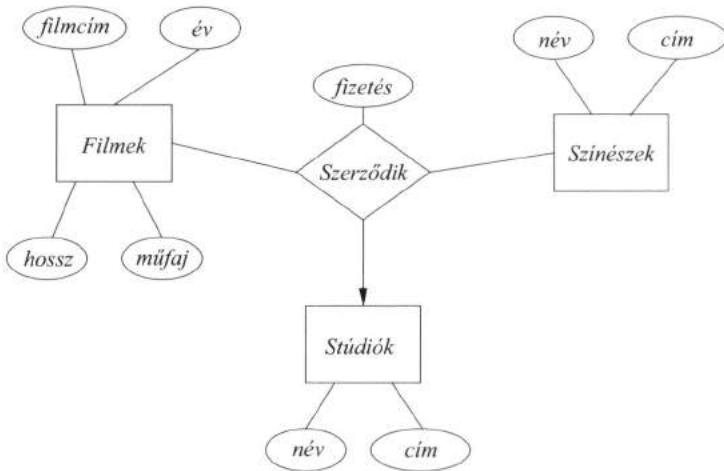
A 4.6. ábrán láthatjuk, hogy a nyilak minden két szerep esetében a *Stúdiók* egyedhalmazra mutatnak, akkor is, amikor a színész „tulajdonosa”, és akkor is, amikor filmgyártó szerepben van a *Stúdiók* egyedhalmaz. Azonban továbbra sincsenek nyilak a *Színészek* és a *Filmek* egyedhalmazokra. Az ok a következő: Ha adott egy színész, egy film és egy stúdió, amely a filmet készíti, akkor csak egy stúdió lehet, amely a „tulajdonosa” a színésznek. (Feltessük, hogy egy színész pontosan egy stúdióval kötött szerződést.) Hasonlóan csak egy stúdió lehet a gyártó, ha adott a színész, a film és a színész stúdiója. Megjegyezzük, hogy minden két esetben, a többi egyedekek közül egy is elegendő ahhoz, hogy egyértelműen meghatározzuk a stúdió egyedét. Például a gyártó stúdió meghatározásához elegendő a filmet ismerni. Ez azonban nem változtat a sokágú kapcsolat specifikációján.

Tehát a *Színészek* és a *Filmek* egyedhalmazokra nem mutatnak nyilak. Ha adott egy színész, a stúdiója és a gyártó stúdió, akkor ezekhez több szerződés is tartozhat más-más filmekkel. Így ez a három komponens nem határozza meg egyértelműen a filmet. Hasonlóan egy gyártó stúdió szerződést köthet egy másik stúdióval több színészre is egy adott filmhez. Így a színészt sem határozza meg egyértelműen a másik három komponens. □

4.1.9. Kapcsolatok attribútumai

Néha kényelmes, sőt elengedhetetlen, ha attribútumokat rendelhetünk egy kapcsolathoz, és nem a benne részt vevő valamelyik egyedhalmazhoz. Például vizsgáljuk meg a 4.4. ábrát, amelyen modelleztük a színészek és a stúdiók szerződéseit valamelyen filmre.¹ Szeretnénk nyilvántartani az adott szerződéshez tartozó fizetést is. Azonban ez nem rendelhető a színészhez, mert a színész valószínűleg különböző filmek esetében különböző összeget kap. Hasonlóan, a stúdióhoz sem rendelhető (különböző színészeknek valószínűleg különböző összeget fizet) és a

¹ Itt visszatérünk a 4.5. példában szereplő, háromágú *Szerződik* kapcsolatra a 4.7. példa négyágú változata helyett.



4.7. ábra. Egy kapcsolat attribútummal

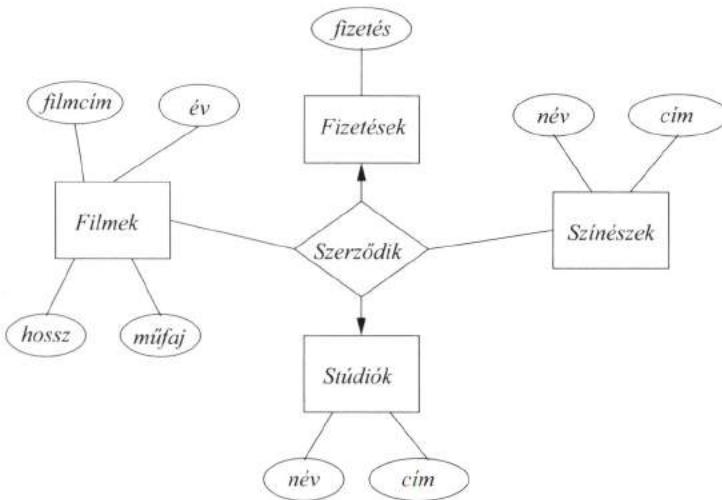
filmhez sem rendelhető (különböző színészek egy adott filmben is különböző fizetést kaphatnak).

Azonban, ha a fizetést a (színész, film, stúdió) hármashoz rendeljük a *Szerződik* kapcsolat kapcsolathalmazában, akkor az éppen jó lesz. A 4.7. ábra ugyanaz, mint a 4.4. ábra, kiegészítve attribútumokkal. A kapcsolatnak van *fizetés* attribútuma, az egyedhalmazoknak pedig ugyanazok az attribútumai, mint a 4.2. ábrán.

Általában bármely kapcsolathoz attribútumot vagy attribútumokat rendelhetünk. Ezen attribútumok értékét a kapcsolathalmazhoz rendelt reláció sora funkcionálisan meghatározza. Egyes esetekben az attribútumok a relációban szereplő egyedhalmazok egy részhalmaza által meghatározottak lehetnek, de feltehetően egyik egyszerű egyedhalmaz által sem meghatározott (vagy az attribútumnak az adott egyedhalmazhoz való rendelése nagyobb zavart okozna). A 4.7. ábrán például a fizetés valójában a film és a színész egyedeik által meghatározott, a stúdió egyedet pedig a film határozza meg.

Nem szükséges a kapcsolatokhoz attribútumokat rendelni. Helyette felvehetünk egy új egyedhalmazt, amely egyedeinek attribútumai éppen a kapcsolathoz rendelt attribútumok. Ha ezt az egyedhalmazt is bele vesszük a kapcsolatba, akkor elhagyhatjuk a kapcsolat attribútumait. Ugyanakkor hasznos konvenció a kapcsolatoknak attribútumokat adni, ezért ezt a jövőben is fogjuk alkalmazni, ahol indokolt.

4.8. példa. Tekintsük a 4.7. ábra E/K-diagramját, amelyen a fizetés a *Szerződik* kapcsolat attribútuma. Hozzunk létre egy *Fizetések* egyedhalmazt fizetés attribútummal. A *Fizetések* lesz a negyedik egyedhalmaz a *Szerződik* kapcsolatban. A teljes E/K-diagram a 4.8. ábrán látható. Megjegyezzük, hogy a 4.8. ábrán a *Fizetések* egyedhalmazhoz nyíl vezet. Ez a nyíl azt fejezi ki, hogy tudjuk, hogy



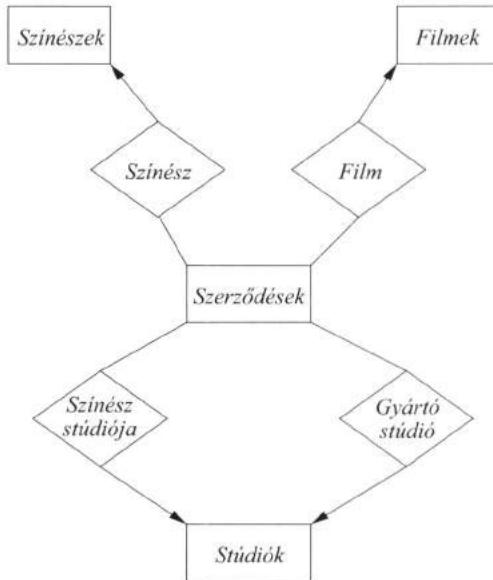
4.8. ábra. Kapcsolat attribútumának megszüntetése egy új egyed bevezetésével

a fizetés a kapcsolatban szereplő többi egyed által meghatározott. Általában, amikor a kapcsolat attribútumát átalakítjuk egy újonnan bevezetett egyedhalmaz attribútumává, akkor ehhez az egyedhalmazhoz nyíl vezet. □

4.1.10. Sokágú kapcsolatok átalakítása binárissé

Egyes adatmodellek csak bináris kapcsolatokat engednek meg. Ilyenek például az UML (4.7. alfejezet) és az ODL (4.9. alfejezet). Annak ellenére tehát, hogy az E/K-modellben nincs ilyen megszorítás, érdemes megemlíteni, hogy bármely sokágú kapcsolat átalakítható sok-egy típusú bináris kapcsolatok gyűjteményére egy új egyedhalmaz bevezetésével, melynek az egyedei tulajdonképpen a sokágú kapcsolat kapcsolathalmazának elemei. Ezt az egyedhalmazt *kapcsoló egyedhalmaznak* nevezzük. Ezután sok-egy kapcsolatot készítünk a kapcsoló egyedhalmaz és az eredeti kapcsolatban részt vevő egyedhalmazok között. Ha egy egyedhalmaz több szerepben is előfordult az eredeti kapcsolatban, akkor most minden szerep egy új kapcsolat lesz.

4.9. példa. A 4.6. ábrán lévő négyágú *Szerződik* kapcsolatot kicsérélhetjük egy *Szerződések* egyedhalmazra. Mint a 4.9. ábrán látható, négy kapcsolatban vesz részt. Ha a *Szerződik* kapcsolat kapcsolathalmazának van egy (stúdió1, stúdió2, színész, film) eleme, akkor a *Szerződések* egyedhalmaznak is van egy e egyede. Ez az egyed a *Színész* kapcsolaton keresztül kapcsolódik a *Színészek* egyedhalmaz fenti színész egyedéhez. Ugyanígy a *Film* kapcsolaton keresztül éri el a *Filmek* egyedhalmaz a fenti film egyedét. A *Stúdiók* egyedhalmaz stúdió1 és stúdió2 egyedét pedig *Színész stúdiója* és a *Gyártó stúdió* kapcsolatokon keresztül azonosítja.



4.9. ábra. Egy sokágú kapcsolat kiváltása egy egyedhalmazzal és bináris kapcsolatokkal

Feltettük, hogy a *Szerződések* egyedhalmaznak nincs attribútuma, a többi egyedhalmaz attribútumait pedig a 4.9. ábrán nem tüntettük fel. Természetesen vehetnénk fel attribútumokat a *Szerződések* egyedhalmazhoz is, mint például az aláírás dátuma stb. □

4.1.11. Alosztályok az E/K-modellben

Gyakran előfordul, hogy egy egyedhalmaz egyedei között egyeseknek olyan speciális tulajdonságaik vannak, amelyekkel nem rendelkezik a halmaz minden egyede. Így célszerű speciális egyedhalmazokat, ún. *alosztályokat* definiálni, ahol minden alosztálynak vannak speciális attribútumai és/vagy kapcsolatai. Ha egy egyedhalmazt alosztályival speciális kapcsolat köt össze, azt „az-egy” (angolul *isa*) kapcsolatnak nevezzük (például „egy *A* az egy *B*” állítás kifejezi a speciális „az-egy” kapcsolatot *A*-ból *B*-be). Ezt a kapcsolatot más néven *öröklési kapcsolatnak* is nevezzük, és a továbbiakban így hivatkozunk rá. (*A* *ford.*)

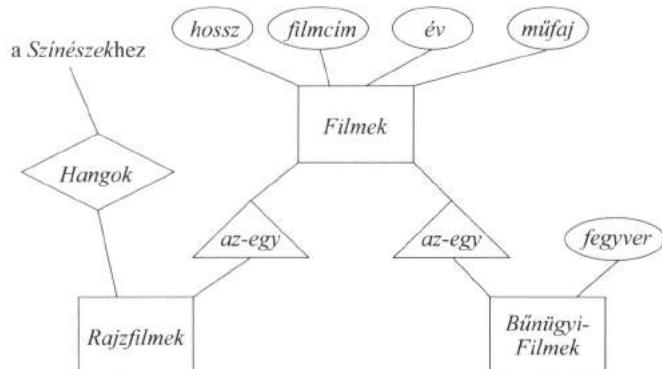
Az öröklési (az-egy) kapcsolatot a hagyományostól eltérően háromszöggel jelöljük, ezzel is kifejezve e kapcsolattípus különlegességét. A háromszög egyik oldalát az alosztályal kötjük össze, ellenkező oldali csúcsát pedig az ősosztályval (szuperosztályval). minden öröklési kapcsolat egy-egy kapcsolat, de az ezt kifejező nyilakat nem tüntetjük fel külön a diagramon.

4.10. példa. Azok közöt a filmek között, amelyeket tárolunk a példa adatbázisunkban, vannak rajzfilmek, bűnügyi filmek, kalandfilmek, vígjátékok és sok

A párhuzamos kapcsolatok különbözhetnek

A 4.9. ábrán láthatunk egy finomságot is a kapcsolatokra vonatkozóan. A Szerződések és a *Stúdiók* egyedhalmazt kétféle kapcsolat is összeköti egymással: a *Színész stúdiója* és a *Gyártó stúdió*. Ez nem jelenti azt, hogy a két kapcsolat tartalmilag megegyezik, ugyanazok a párok fordulnának elő bennük. Ez esetben ennek épp az ellenkezője igaz: nem valószínű, hogy egy szerződést minden két kapcsolat ugyanahoz a stúdióhoz rendelne, mivel akkor a stúdió saját magával kötne szerződést.

Általánosságban nem baj, ha egy E/K-diagramon két egyedhalmaz között többféle kapcsolatot definiálunk. Normális esetben e kapcsolatok tartalmilag is különbözni fognak az adatbázisban, jelentésüknek megfelelően.



4.10. ábra. Alosztályok az E/K-diagramban

egyéb speciális típusú film. Ezen filmtípusok mindegyike definiálható a *Filmek* egyedhalmaz alosztályaként. Például legyen az alábbi két alosztály: *Rajzfilmek* és *BüdügyiFilmek*. Egy rajzfilmnek a *Filmek* egyedhalmaznál megadott attribútumokon és kapcsolatokon kívül van még egy *Hangok* nevű kapcsolata, mely hozzárendeli azokat a színészeket, akik a filmben beszélnek, de maguk nem jelennek meg. Ilyen színészek csak a rajzfilmekhez rendelhetők. A büdügyi filmekhez pedig tartozik egy további attribútum: a *fegyver*. A *Filmek*, *Rajzfilmek* és *BüdügyiFilmek* egyedhalmazok viszonyait a 4.10. ábra mutatja. □

Elméletileg az öröklési kapcsolattal összekötött egyedhalmazok tetszőleges struktúrát alkothatnak, de mi teszünk egy megszorítást: az ilyen leszármazási struktúrák fát kell alkossanak, melynek egy *gyökéreleme* van (például ilyen a 4.10. ábrán a *Filmek* egyedhalmaz). Ez a legáltalánosabb egyedhalmaz, a gyökértől távolodva pedig egyre speciálisabbak következnek.

Alosztályok objektumorientált rendszerekben

Lényeges hasonlóság van az E/K öröklési („az-egy” típusú) kapcsolata és az objektumorientált nyelvek alosztályai között. Az E/K-modellben az öröklési kapcsolat bizonyos értelemben egy alosztályt kapcsol össze ökosztályával (szuperosztályával). Ugyanakkor alapvető különbség is van: míg egy egyednek lehetnek reprezentánsai több egyedhalmazban, amelyek fát alkothatnak, addig egy objektum minden csakis csak egy osztályba vagy alosztályba tartozik bele.

A különbséget abból érzékelhetjük, ahogyan a 4.11. példában a *Roger nyúl a páccban* című filmet kezeltük. Objektumelvű megközelítésben egy negyedik egyedosztályra volna szükségünk, a „*BűnügyiRajzFilmek*”-re, amely örökölné a *Filmek*, a *Rajzfilmek* és a *BűnügyiFilmek* attribútumait és kapcsolatait. Az E/K-modellben erre a negyedik osztályra nincs szükség, mivel épp ezt érjük el, ha az említett filmet besoroljuk a *Rajzfilmek* és a *BűnügyiFilmek* egyedhalmazba is, és a film összetevőit a megfelelő egyedhalmazban adjuk meg.

Tegyük fel, hogy van egy egyedhalmazokból álló fa, amely öröklési kapcsolatokkal van felépítve. Ekkor egy egyed komponensekből tevődik össze, mivel a fában egy elemet valójában felmenőivel együtt kell értenünk (egészen a gyökérig). Pontosabban, ha egy *e* egyednek van egy *E* egyedhalmazhoz tartozó *c* komponense, és a fában *E* szülője az *F* egyedhalmaz, akkor *e*-nek kell legyen egy *d* komponense is az *F* egyedhalmazban. Továbbá a *c-d* párnak szerepelnie kell az *E-t F-fel* összekötő öröklési kapcsolatban. Az *e* egyed attribútumait a komponensek attribútumainak összessége alkotja, és *e* részt vesz mindenekben a kapcsolatokban, melyekben valamely komponense részt vesz.

4.11. példa. Tekintsük a 4.10. ábrát. Egy film, amely nem rajzfilm és nem is bűnügyi film, csak egy komponensből áll, mely a *Filmek* egyedhalmazhoz tartozik, és ennek megfelelően csak négy attribútuma lesz (valamint két kapcsolatban vehet részt: *SzerepelBenne* és *Gyártó* – ezeket nem tüntettük fel a 4.10. ábrán).

Egy rajzfilm, mely nem bűnügyi film, két komponensből áll: egyik a *Filmek*, másik a *Rajzfilmek* egyedhalmaznak az eleme. Egy ilyen film rendelkezik az előbb említett négy attribútummal, és a *Filmek* kapcsolatai mellett a *Hangok* kapcsolat is hozzátarozik. Hasonlóan, egy bűnügyi film két komponensből áll (*Filmek* és *BűnügyiFilmek*), és így összesen öt attribútuma van, beleértve a *fegyver-t* is.

Egy olyan egyed, mint a *Roger nyúl a páccban*, amely egyszerre rajzfilm és bűnügyi film, a komponenseit három egyedhalmazból, a *Filmek*, a *Rajzfilmek* és a *BűnügyiFilmek* egyedhalmazokból gyűjtí össze. A három komponens az öröklődési kapcsolaton keresztül kapcsolódik egy egyedben. Együtt a három komponens adja a *Roger nyúl a páccban* egyed attribútumait és kapcsolatait.

Négy attribútumot és két kapcsolatot ad a *Filmek* egyedhalmaz, a *fegyver* attribútumot adja a *BűnűgyiFilmek* egyedhalmaz és a *Hangok* kapcsolatot adja a *Rajzfilmek* egyedhalmaz. □

4.1.12. Feladatok

4.1.1. feladat. Tervezzünk egy bank részére adatbázist, amely tartalmazza az ügyfeleket és azok számláit. Az ügyfelek rövidített nevüket, címüket, telefonszámukat és TAJ-számukat. A számlák legyenek számlaszámuk, típusuk (például takarékbetét-számla, folyószámla stb.) és egyenlegük. Továbbá meg kell jelölni azokat az ügyfeleket, akiknek van számlájuk. Adjuk meg az E/K-diagramját ennek az adatbázisnak. Alkalmazzunk nyilakat a kapcsolatokban a multiplicitások jelölésére.

4.1.2. feladat. Módosítsuk a 4.1.1. feladatra adott megoldásunkat az alábbiak szerint:

- a) Változtassuk meg a diagramot úgy, hogy egy számlának csak egy tulajdonosa lehessen.
- b) Változtassuk tovább a diagramot úgy, hogy egy ügyfélnek csak egy számlája lehessen.
- c) A 4.1.1. feladat eredeti diagramját változtassuk meg úgy, hogy egy ügyfélhez rendelhessünk lakcímeknek (ország-város-utca hármasal megadva) és telefonszámoknak egy halmazát. Ügyeljünk arra, hogy az E/K-modellben nem megengedett, hogy attribútumoknak valamelyen összetett típusa (például halmaz) legyen.
- d) Módosítsuk tovább a diagramunkat úgy, hogy minden ügyfélhez tartozhasson lakcímek egy halmaza, és minden egyes lakcímhez telefonszámoknak egy halmaza.

4.1.3. feladat. Adjuk meg az E/K-modelljét egy olyan adatbázisnak, amely csapatokat, játékosokat és azok szurkolói tartja nyilván:

1. minden csapatról tároljuk a nevét, játékosait, csapatkapitányát (ő is egy játékos) és a mezük színét.
2. minden játéknak legyen neve.
3. minden rajongóról tartsuk nyilván a nevét, kedvenc csapatát, kedvenc játékosát és kedvenc színét.

Vigyázzunk, a színek halmaza nem lehet a csapatok egy attribútumának típusa. Hogyan lehet ezzel a megszorítással együtt megfelelő modellt készíteni?

4.1.4. feladat. Tegyük fel, hogy szeretnénk hozzávenni a 4.1.3. feladat E/K-diagramjához a *Vezeti* kapcsolatot, ami két játékos közötti viszonyt fejez ki egy csapaton belül. Ez a kapcsolat (játékos1, játékos2, csapat) hármasok halmaza úgy, hogy a játékos2 volt a csapat kapitánya, amikor együtt játszott a játékos1-gyel.

- Rajzoljuk meg ezt a módosítást az E/K-diagramon.
- Cseréljük ki új egyedhalmazra és bináris kapcsolatokra a fenti hármas kapcsolatot.
- Az új bináris kapcsolatok ugyanolyanok, mint a korábban meglévő kapcsolatok? Feltételezhetjük, hogy a két játékos különböző, azaz egy kapitány nem vezeti saját magát.

4.1.5. feladat. Módosítsuk a 4.1.3. feladatot úgy, hogy a játékosokról jegyezzük fel, korábban mely csapatokban játszottak, beleértve a belépés és kilépés dátumát is.

! 4.1.6. feladat. Tegyük fel, hogy családfa-adatbázist tervezünk az egyetlen, *Személyek* egyedhalmazzal. Egy személyről a következő információkat kívánjuk tárolni: neve (attribútumként), anyja, apja és gyerekei (kapcsolatokként).

! 4.1.7. feladat. Módosítsuk a 4.1.6. feladatban lévő adatbázistervet a következő speciális személytípusokkal:

- Nők.
- Férfiak.
- Szülők.

Megkülönböztethetünk más típusokat is, hogy a kapcsolatok a megfelelő alosztályokhoz kapcsolódjanak.

4.1.8. feladat. Alternatív megoldás lehet a 4.1.6. feladatban meghatározott információk reprezentálásának a *Család* hármas kapcsolat. Ez a kapcsolat (személy, anya, apa) hármasok halmaza, ahol egy személyt és szüleit tároljuk. Természetesen minden adat a *Személyek* egyedhalmazból kerül ki.

- Rajzoljuk meg ezt a diagramot. Helyezzük el a nyilakat a megfelelő helyekre.
- Cseréljük ki a *Család* hármas kapcsolatot egy egyedhalmazra és bináris kapcsolatokra. Ismét helyezzük el a nyilakat a kapcsolatok típusának megfelelően.

4.1.9. feladat. Tervezzünk adatbázist egy tanulmányi osztály számára. Ez az adatbázis tartalmazza a hallgatókat, oktatókat, tanszékeket és kurzusokat. Ezenkívül tartsuk nyilván, hogy a hallgatók milyen kurzusokat vettek fel, az adott kurzust mely oktató oktatja, a hallgatók jegyeit, a kurzusoknál az oktató munkáját segítő hallgatókat, egy adott kurzust mely tanszék ajánlotta, és minden olyan információt, amely a fentiek megvalósításához szükséges. Megjegyezzük, hogy ez a feladat nagy szabadságot enged a korábbiakhoz képest. Dönteni kell a kapcsolatok típusáról (sok-sok, sok-egy vagy egy-egy), az alkalmas típus megválasztásáról, illetve arról, hogy milyen segédinformációkat használunk.

! 4.1.10. feladat. Informálisan két E/K-diagramról azt mondhatjuk, hogy „ugyanazt az információt hordozzák”, ha bármely valós helyzetre igaz, hogy az egyik diagram előfordulásából kiszámítható a másik diagram előfordulása, mely ugyanazt a helyzetet írja le. Tekintsük a 4.6. ábrát. A négyágú kapcsolat szétbontható egy háromágú és egy kétágú kapcsolattá, figyelembe véve, hogy minden egyes filmhez egy gyártó stúdió tartozik. Adjunk meg egy E/K-diagramot, melyben nem szerepel négyágú kapcsolat, és ugyanazt az információt hordozza, mint a 4.6. ábra.

4.2. Tervezési alapelvek

Még nagyon sok részletet kell megismernünk az E/K-modellel kapcsolatban, de már eleget tudunk ahhoz, hogy elkezdjük megvizsgálni a lényeges pontjait annak, hogy mitől jó egy terv és mit kell elkerülni. Ebben az alfejezetben néhány hasznos tervezési elvet ajánlunk az Olvasó figyelmébe.

4.2.1. Valósághű modellezés

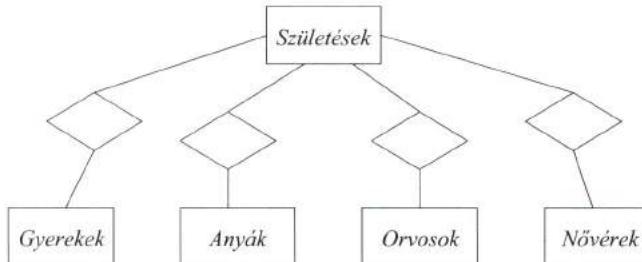
Az első és legfontosabb, hogy a tervnek pontosan meg kell felelnie az alkalmazás specifikációjának. Azaz, az egyedhalmazoknak és attribútumaiknak tükrözniük kell a valóságot. Nem lehet a *Színészek* egyedhalmaznak attribútuma a *hengerekSzáma*, hiszen ez inkább az *Autók* egyedhalmazhoz tartozik. Akármilyen kapcsolatok jönnek is létre, azoknak értelmeseknek kell lenniük az alapján, amit ismerünk a valós világ modellezendő részéről.

4.12. példa. Ha definiálunk egy *SzerepelBenne* kapcsolatot a *Színészek* és a *Filmek* között, annak sok-sok kapcsolatnak kell lennie. Az ok az, hogy a valós világban megfigyelhetjük, hogy színészek több filmben is szerepelhetnek, és a filmekben egynél több színész is szerepelhet. Tehát nem lenne korrekt a *SzerepelBenne* kapcsolatot sok-egy vagy egy-egy kapcsolatnak definiálni. □

4.13. példa. Másfelől nem minden magától értetődő, hogyan modellezzük a valóságot E/K-diagrammal. Tegyük fel, hogy van egy *Tantárgyak* és egy *Oktatók* egyedhalmazunk, közöttük az *Oktat* kapcsolattal. Kérdés, hogy e kapcsolat sok-egy típusú-e a *Tantárgyak* felől az *Oktatók*hoz. A válasz az adatbázist létrehozó szervezet szándékain és működési szabályain múlik. Elképzelhető, hogy

- c) Az a) és b) feltételeket egészítsük ki azzal, hogy minden születés pontosan egy orvoshoz tartozik.

Milyen tervezési hibákkal terheltek az egyes esetek?



4.16. ábra. A születéseket egy egyedhalmazzal adjuk meg

!! **4.2.7. feladat.** Tegyük fel, hogy megváltoztatjuk a nézőpontunkat és megengedjük, hogy egy szülés alatt több gyereket is szülhessen egy anya. Hogyan lehetne ekkor megvalósítani azt, hogy minden gyereknek pontosan egy anyja van a 4.2.5. feladatban leírt és a 4.2.6. feladatban megadott megközelítések esetében külön-külön?

4.3. Megszorítások modellezése

Az E/K-modell számos lehetőséget ad arra, hogy a tervezés alatt álló adatbázis adataira vonatkozó szokásos típusú előírásokat adjunk meg. A relációs modellhez hasonlóan itt is meg tudjuk adni, hogy egy attribútum vagy attribútumok egy halmaza az egyedhalmaz kulcsa. Láttuk azt is, hogy egy kapcsolatból az egyedhalmazhoz vezető nyíl a „funkcionális függőség” kifejezésére szolgál. Arra is van mód, hogy kifejezzük a hivatkozásiépség-megszorítást, amikor egy egyedhalmaz egy egyéneknél másik egyedhalmazbeli egyedhez kell kapcsolódnia.

4.3.1. Kulcsok az E/K-modellben

Egy E egyedhalmaz *kulcsa* egy vagy több attribútum K halmaza úgy, hogy tetszőleges két, egymástól különböző e_1 és e_2 egyed nem egyezhet meg a K -beli attribútumok mindegyikén. Ha K több attribútumot is tartalmaz, akkor e_1 és e_2 megegyezhet azok egy részén, de nem mindegyiken. Hárrom fontos szempontra hívjuk fel a figyelmet:

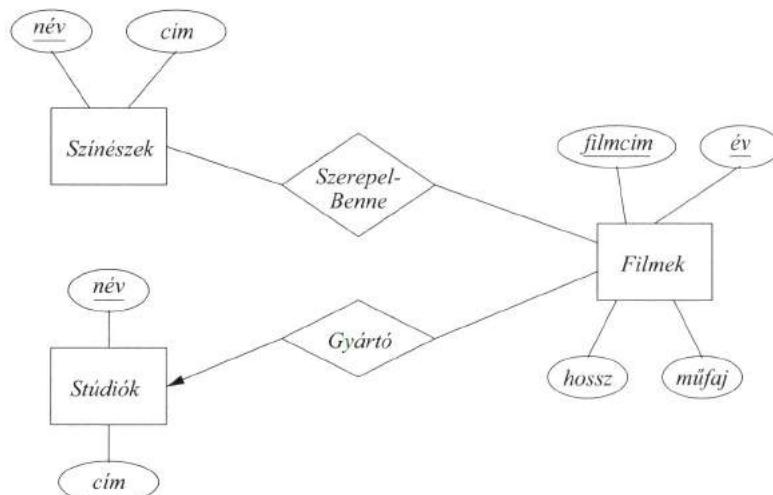
- minden egyedhalmaznak kell legyen kulcsa, még az az-egy-hierarchiában részt vevő és a gyenge egyedhalmazok (lásd 4.4. alfejezet) esetében is, ilyenkor a kulcs más egyedhalmazhoz tartozik.

- Egy egyedhalmaz több lehetséges kulccsal is rendelkezhet. Szokás szerint ilyenkor választunk közülük egy „elsődleges kulcsot” és úgy tekintjük, mintha csak az volna kulcs.
- Ha egy egyedhalmaz egy öröklési („az-egy” típusú) kapcsolatok által meghatározott hierarchiában szerepel, akkor a gyökérben lévő egyedhalmaznak kell rendelkeznie a kulcshoz szükséges attribútumokkal; és minden egyed kulesát a gyökérhalmazbeli komponense határozza meg, függetlenül attól, hogy milyen további komponensei vannak a hierarchiában.

A filmes példánkban a *filmcím* és *év* attribútumok halmazát használjuk a *Filmek* kulcsának, arra a megfigyelésre alapozva, hogy valószínűtlen az, hogy ugyanabban az évben két ugyanolyan című film készülne. Azt is eldöntöttük, hogy a *Színészek* esetében a *név* kellően biztonságos kulcs, feltehetjük ugyanis, hogy egyik színész sem kívánja másik színész nevét használni (inkább „művésznevet” választanak maguknak).

4.3.2. Kulcsok jelölése az E/K-modellben

Az E/K-diagramokban aláhúzással jelöljük az egyedhalmazok kulcsattribútumait. Például a 4.17. ábrán a 4.2. ábra diagramja látható, a *Filmek*, *Színészek* és *Stúdiók* kulcsainak aláhúzásával. A *név* a *Színészek* egyattribútumos kulcsa, és hasonlóan a *Stúdiók* kulcsa is a saját *név* attribútuma. A *Filmek* kulcsa pedig két attribútumból áll: *filmcím* és *év*.



4.17. ábra. Kulcsok jelölése aláhúzással az E/K-diagramon

Ha egy egyedhalmaznak több kulcsa is van, azt nem jelöljük, csak az elsődleges kulcsot. Ha egyszerre több attribútum van aláhúzva (mint ahogyan a

4.17. ábrán a *Filmek* esetén), akkor azok együtt alkotják a kulcsot. Érdemes megemlíteni, hogy egyes, szokatlan esetekben előfordulhat, hogy a kulcsnak nem minden attribútuma tartozik magához az egyedhalmazhoz. Erre a 4.4. alfejezetben térünk vissza („gyenge” egyedhalmazok).

4.3.3. Hivatkozások épsége

Gondoljunk vissza a 2.5.2. alfejezetben tárgyalt hivatkozásiépség-megszorításokra. Ez a megszorítás azt mondja, hogy az egyik egyedhalmazban szereplő értéknek elő kell fordulnia egy (adott) másik egyedhalmazban is. Vizsgáljuk meg például a 4.2. ábra *Gyártó* kapcsolatát, mely sok-egy típusú a *Filmek* írányából a *Stúdiók* felé. A sok-egy kapcsolat pusztán annyit követel meg, hogy ne lehessen olyan film, amelyet egynél több stúdió gyárt. Azt viszont nem, hogy minden filmhez tartozzon gyártó stúdió, és ha tartozik is, benne kell legyen az adatbázisban, a *Stúdiók* egyedhalmaz elemeként. A hivatkozások épsége azt követeli meg a *Gyártó* esetében, hogy a (kapcsolat által hivatkozott) stúdió egyed minden filmhez létezzen az adatbázisban.

Az E/K-diagramokban használt nyíl jelölés alkalmas arra is, hogy mutassa meg egy kapcsolatról, hogy az elvárja a hivatkozás épségének megőrzését. Tegyük fel, hogy R egy kapcsolat E és F egyedhalmazok között. Kerek nyílvéget fogunk használni annak jelölésére, hogy egy kapcsolat nemcsak sok-egy vagy egy-egy kapcsolat, hanem megköveteli az egy oldalon a hivatkozás épségét, azaz E minden egyedéhez kell, hogy létezzen egy vele kapcsolatban álló F egyedhalmazbeli egyed. Ugyanez az ötlet használható, amikor R több egyedhalmaz közötti kapcsolat.

4.19. példa. A 4.18. ábra mutatja a hivatkozásiépség-megszorításokat a *Filmek*, *Stúdiók* és *Elnökök* egyedhalmazok között. Ezeket az egyedhalmazokat és kapcsolatokat a 4.2. és 4.3. ábrákon vezettük be. Láthatjuk, hogy a *Stúdiók* egyedhalmazhoz megy egy kerek nyíl a *Gyártó* kapcsolatban. Ez azt jelenti, hogy minden filmhez kell hogy tartozzon egy és csak egy gyártó stúdió, melynek benne kell lennie a *Stúdiók* egyedhalmazban.



4.18. ábra. Hivatkozásiépség-megszorításokat tartalmazó E/K-diagram

Hasonlóan kerek nyílat találunk a *Irányít* kapcsolatban szintén a *Stúdiók* oldalán. Ez azt jelenti, hogy minden elnök vezet egy stúdiót, amely létezik a *Stúdiók* egyedhalmazában.

Megjegyezzük, hogy a *Irányít* kapcsolat *Elnökök* oldala nem kerek nyíl maradt. Ennek oka a következő ésszerű feltevés a stúdiók és elnökeik közötti kapcsolatról. Ha egy stúdió megszűnik, akkor az elnököt nem lehet többé (stúdió)elnöknek nevezni, így elvárható, hogy ez az elnök töröljön az *Elnökök* egyedhalmazból. Ezért van a kerek nyíl a *Stúdiók* oldalon. Már részről, ha egy

elnök törlődik az adatbázisból, akkor a stúdió még tovább létezhet. Ezért helyeztünk el hagyományos nyilat az *Elnökök* oldalon, ami azt jelenti, hogy minden stúdiónak legfeljebb egy elnöke van, de lehet olyan időszak, amikor éppen nincs elnöke. \square

4.3.4. Egyéb megszorítások

Az E/K-modellben a kapcsolat fokára vonatkozó megszorítással korlátozhattuk a kapcsolatban részt vevő egyedek számát. Például egy film egyedhez nem kapcsolódhat 10-nél több színész egyed a *SzerepelBenne* kapcsolatban. Az E/K-modellben a kapcsolat éleire írhatjuk rá a korlátot, ami azt jelenti, hogy a kapcsolatban részt vevő egyedek száma korlátozott az egyedhalmazra nézve. A 4.19. ábra mutatja, hogyan lehet E/K-modellben megadni ezt a megszorítást. Másik példaként tekinthetjük, hogy a nyíl annak a megszorításnak a szinonimája, hogy „ ≤ 1 ”, a kerek nyíl pedig, amelyet a 4.18. ábrán láttunk, az „ $= 1$ ” szinonimája.



4.19. ábra. A filmenkénti maximális szereplők számát szabályozó megszorítás

4.3.5. Feladatok

4.3.1. feladat. Adjuk meg a következő feladatokhoz készült E/K-diagramokban *i)* a kulcsokat és *ii)* a hivatkozásiépség-megszorításokat:

- a)* 4.1.1. feladat,
- b)* 4.1.3. feladat,
- c)* 4.1.6. feladat.

! 4.3.2. feladat. Az E/K-modell kapcsolatainak ugyanúgy lehetnek kulcsai, mint az egyedhalmazoknak. Legyen R egy kapcsolat az E_1, E_2, \dots, E_n egyedhalmazok között. Ekkor R kulcsa az E_1, E_2, \dots, E_n egyedhalmazok attribútumainak olyan K halmaza, hogy bárholgy választunk két különböző elemet R kapcsolathalmazából, ezek az összes K -beli attribútumokon nem egyezhetnek meg. Tegyük fel, hogy $n = 2$, azaz R egy bináris kapcsolat. minden i -re, K_i attribútumhalmaz legyen az E_i egyedhalmaz kulcsa. Adjuk meg R legkisebb lehetséges kulcsát E_1 és E_2 segítségével az alábbi feltevések mellett:

- a)* R sok-sok kapcsolat.
- b)* R sok-egy kapcsolat úgy, hogy E_2 az egy oldal.

- c) R sok-egy kapcsolat úgy, hogy E_1 az egy oldal.
- d) R egy-egy kapcsolat.

!! 4.3.3. feladat. Vizsgáljuk meg ismét a 4.3.2. feladatban lévő problémát, de n bármilyen szám lehessen, ne csak 2. Használva azt az információt, hogy R milyen nyilakkal kapcsolódik az egyedhalmazokhoz, adjuk meg, hogyan lehetne megtalálni R legkisebb K lehetséges kulcsát K_i -k segítségével.

4.4. Gyenge egyedhalmazok

Esetenként előfordulhat, hogy egy egyedhalmaz kulcsában szereplő attribútumok közül néhány, vagy esetleg az összes, más egyedhalmaznak attribútuma. Az ilyen egyedhalmazokat *gyenge egyedhalmazoknak* nevezzük.

4.4.1. A gyenge egyedhalmazok bevezetésének okai

Két elvi oka lehet a gyenge egyedhalmazok bevezetésének. Egyik, amikor az egyedhalmazok az-egy-hierarchiát alkotnak (a 4.1.11. alfejezetben tárgyalta). Ha például az E egyedhalmaz egyedei részei az F egyedhalmaz egyedeinek, akkor lehetséges, hogy az E -beli egyedek nevei nem egyértelműek; ha viszont az E -beli egyedhez tartozó (azt tartalmazó) F -beli egyed nevét hozzávesszük, akkor már igen. Néhány példán keresztül érzékelhetjük a problémát.

4.20. példa. Tegyük fel, hogy egy filmstúdiónál több csoport van, amelyek filmkészítéssel foglalkoznak. Egy adott stúdió esetében ezeket a csoportokat egy-egy szám jelöli, azaz 1. csoport, 2. csoport stb. Azonban egy másik stúdió is ugyanezt a jelölést használhatja csoportjaira, így a *szám* attribútum nem kulcsa a csoportoknak. Tehát ahhoz, hogy egyértelmű legyen egy csoport neve, szükség van a stúdió nevére is és a csoport számára is. Ezt ábrázolja a 4.20. ábra. A duplán keretezett téglalap jelzi a gyenge egyedhalmazt, a duplán keretezett rombusz egy sok-egy típusú kapcsolat, és a gyenge egyedhalmaz kulcsának megállapítását segíti. A jelölésekkel részletesebben a 4.4.3. alfejezetben foglalkozunk. A *Csoportok* gyenge egyedhalmaz, kulcsa a saját *szám* attribútuma és a *Stúdiók* egyedhalmaz *név* attribútuma, amelyhez a *Csoportok* egyedhalmaz a *Része* része sok-egy kapcsolaton keresztül kapcsolódik. \square



4.20. ábra. A csoportok gyenge egyedhalmaza és kapcsolatai

4.21. példa. A fajokat nemzetsegükkel és fajukkal jelölik. Például az emberek a *Homo sapiens* fajhoz tartoznak, ahol a *Homo* a nemzetseg neve, a *sapiens* a faj neve. Általában egy nemzetseg fajokat tartalmaz, amelyeknek a neve a nemzetseg nevével kezdődik és a faj nevével fejeződik be. Sajnos, maguk a fajok nevei nem egyértelműek; két vagy több nemzetsegben is lehet ugyanolyan fajnév. Így egy fajt egyértelműen a nemzetseg és a faj neve jelöli. A nemzetseghez a faj egy *Tagja* kapcsolaton keresztül kapcsolódik, a 4.21. ábrán javasolt módon. Tehát a *Fajok* egy gyenge egyedhalmaz, amelynek a kulcsa a nemzetseggel egészül ki. □



4.21. ábra. A fajok gyenge egyedhalmaza

A gyenge egyedhalmazok másik forrása a 4.1.10. alfejezetben bemutatott eljárás, amelynek segítségével a sokágú kapcsolatokat átalakítjuk binárisá³. Ezeknek az egyedhalmazoknak gyakran nincs is attribútuma. A kulcsuk olyan attribútumokból áll, amelyek a hozzá kapcsolódó egyedhalmazok kulcsattribútumai.

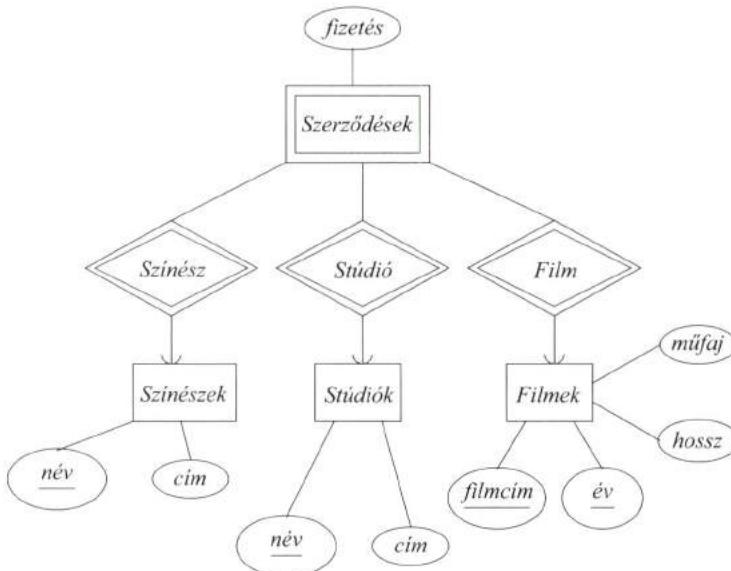
4.22. példa. A 4.22. ábrán láthatjuk a *Szerződések* kapcsoló egyedhalmazt, amely a 4.5. példában lévő *Szerződik* háromágú kapcsolatot helyettesíti. A *Szerződések* egyedhalmaznak van egy *fizetés* attribútuma, de ez az attribútum nem vesz részt a kulcsában. Egy szerződés kulcsa a stúdió nevéről, a színész nevéről, a film címéről és évéről áll. □

4.4.2. Gyenge egyedhalmazokra vonatkozó követelmények

Egy gyenge egyedhalmaz kulcsa nem származhat akárhonnan. Ha *E* egy gyenge egyedhalmaz, akkor kulcsa az alábbiakból tevődik össze:

1. Nulla vagy opcionálisan néhány saját attribútuma;
2. Más egyedhalmazok kulcsattribútumai, melyeknek *E*-hez sok-egy kapcsolattal kell kapcsolódniuk. E kapcsolatokat az *E* kiteljesítő kapcsolatainak nevezhetjük (supporting relationships).

³ Megjegyezzük, hogy bár az E/K-modellben nem kötelező megszüntetni a sokágú kapcsolatokat, egyes adatbázis-tervezési modellekben a sokágú kapcsolatokat át kell alakítani binárisá.



4.22. ábra. A kapcsoló egyedhalmaz gyenge

Ahhoz, hogy egy E és F egyedosztályt összekötő R kapcsolat E egyik kiteljesítő kapcsolata legyen, az alábbi feltételeknek kell teljesülniük:

- R -nek bináris sok-egy kapcsolatnak⁴ kell lennie, amelynek az „egy” oldala az F egyedhalmaznál van.
- R -ben hivatkozási épségnek kell teljesülnie E -től F irányába, azaz minden E -beli egyedhez az adatbázisban léteznie kell R -en keresztül kapcsolódó F -beli egyednek. Más szavakkal, R -ből F -be lekerekített nyílnak kell vezetnie.
- F azon attribútumai, amelyek benne vannak E kulcsában, benne vannak F kulcsában is.
- Ha F maga is gyenge egyedhalmaz, akkor vegyük F azon kulcsattribútumait, amelyek benne vannak E kulcsában. Ezek kulcsattribútumai kell legyenek egy vagy több olyan G egyedhalmaznak is, amelyhez F kiteljesítő kapcsolattal kötődik. Rekurzíve, ha G gyenge, akkor egyes kulcsattribútumait G is máshonnan kapja és így tovább.
- Ha több kiteljesítő kapcsolat is vezet E -ból F -be, akkor mindegyiken keresztül hozzájárulnak F kulcsának attribútumai E kulcsának kialakításához. Megjegyezzük, hogy egy E -beli e egyed más-más F -beli egyeddel

⁴ Emlékezzünk arra, hogy az egy-egy kapcsolat a sok-egy kapcsolat speciális esete. Tehát amikor azt mondjuk, hogy sok-egy kapcsolat, akkor ebbe minden beleérjük az egy-egy kapcsolatot is.

kapcsolódhat a különböző támogató kapcsolatokon keresztül. Így több különböző F -beli egyed kulcsa segíthet azonosítani az e E -beli egyedet.

Az intuitív ok, amiért ezek a feltételek szükségesek, a következő: vizsgálunk meg egy egyedet egy gyenge egyedhalmazból, mondjuk egy csoportot a 4.20. példából. minden csoport egyedi. Elvben meg tudjuk különböztetni a csoportokat egymástól még akkor is, ha ugyanaz a számuk, de különböző stúdióhoz tartoznak. Ez azonban elég nehéz, mert a számuk önmagában nem elegendő. Az egyetlen mód, hogy további információt kapcsolunk a csoportokhoz, ha van egy determinisztikus eljárás, amivel további értékeket kapunk a csoport egyértelműsítéséhez. Csak a következő értékek összessége lesz egyedi egy absztrakt csoport egyedre vonatkozóan:

1. A *Csoportok* egyedhalmaz attribútumainak értéke;
2. A csoport egyed egy kapcsolatán keresztül elérhetők másik, egyedi egyedből származó, szintén egyedi értékek. Az említett kapcsolat sok-egy típusú (vagy speciális esetben egy-egy típusú) kell legyen, a vett érték pedig a másik egyedhalmaz kulcsának része.

4.4.3. Gyenge egyedhalmazok jelölése

A következő jelöléseket használjuk gyenge egyedhalmazokkal kapcsolatban.

1. Ha egy egyedhalmaz gyenge, akkor dupla kerettel jelezzük. Ez a jelölés látható a 4.20. és 4.22. ábrákon (*Csoportok*, illetve *Szerződések*).
2. A gyenge egyedhalmazok sok-egy típusú kiteljesítő kapcsolatait dupla kerettel jelöljük. Ez látható a *Része* kapcsolat esetében a 4.20. ábrán és a 4.22. ábra összes kapcsolata esetében.
3. Ha egy egyedhalmaz bármely attribútuma része a kulcsnak, akkor aláhúzással jelöljük. Például a 4.20. ábrán a csoportok száma részt vesz a kulcsban, bár nem a teljes kulcs.

Az előző jelöléseket a következő szabályban összegezhetjük:

- Ha egy E egyedhalmazt dupla kerettel jelöltünk, akkor az gyenge. Ha E aláhúzott attribútumaihoz (ha vannak) hozzávesszük az E -hez dupla keretű kapcsolattal kötődő egyedhalmazok kulcsattribútumait, akkor ezek összessége E kulcsát képezi.

Ne feledjük, hogy a kettős rombusz csak a kiterjesztő kapcsolatok jelölésére szolgál. Egy gyenge egyedhalmaznak lehetnek olyan sok-egy kapcsolatai, melyek nem kiterjesztő kapcsolatok, ezeket egyszerű rombusz jelöli.

4.23. példa. A 4.22. ábrán a *Stúdió* kapcsolat nem feltétlenül kiterjesztő kapcsolata a *Szerződéseknek*, ugyanis minden filmhez egyedi gyártó stúdió tartozik, melyet – az ábrán fel nem tüntetett – sok-egy kapcsolat határoz meg a *Filmek* és a *Stúdiók* között. Így tehát egy film és egy színész neve már egyértelműen azonosít egy szerződést, mely a filmbeli szereplésre vonatkozik valamely stúdióval. Jelöléseinkben ez annak felel meg, hogy a 4.22. ábrán a *Stúdió* kapcsolatot elegendő lenne hagyományos (szimpla vonalas) rombusszal feltüntetni. □

4.4.4. Feladatok

4.4.1. feladat. Egyik módja, hogy nyilvántartsuk az egyetemi hallgatókat és az általuk szerzett különböző érdemjegyeket, ha veszünk három egyedhalmazt: egyet a hallgatóknak, egyet a kurzusoknak és egyet a „kurzusfelvételeknek”. A kurzusfelvételnek megfelelő egyedhalmaz egy kapcsoló egyedhalmaz a hallgatók és a kurzusok között, és nincsak azt reprezentálja, hogy egy hallgató mely kurzusokat vette fel, hanem a kurzusra kapott érdemjegyet is. Rajzolunk E/K-diagramot a fenti esethez, jelöljük be a gyenge egyedhalmazokat és a kulcsokat. Az érdemjegy része-e a kurzusfelvételt reprezentáló egyedhalmaz kulcsának?

4.4.2. feladat. Módosítsuk a 4.4.1. feladat megoldását úgy, hogy a hallgató összes részteljesítési érdemjegyét tartsuk nyilván, ami egy adott kurzus elvégzéséhez kapcsolódik. Ismét jelöljük a gyenge egyedhalmazokat és a kulcsokat.

4.4.3. feladat. A 4.2.6 a)–c) feladatokhoz készült E/K-diagramokban jelöljük be a gyenge egyedhalmazokat, kiterjesztő kapcsolataikat és a kulcsokat.

4.4.4. feladat. Rajzolunk E/K-diagramot a következő esetekre. Az E/K-diagramokban jelöljük a gyenge egyedhalmazokat és a kulcsokat.

- a) Egyedhalmazok: *Kurzusok*, *Tanszékek*. Egy kurzust egy tanszék hirdet meg, de csak egy számmal azonosítja. A különböző tanszékek adhatják ugyanazokat a számokat a kurzusaiknak. minden tanszék neve egyedi.
- ! b) Egyedhalmazok: *Ligák*, *Csapatok*, *Játékosok*. Ligák nevei egyediek. Nincs olyan liga, amelyben két egyforma nevű csapat lenne. Nincs olyan csapat, amelyben két egyforma kódszámú játékos lenne. Azonban különböző csapatok esetében lehetnek ugyanolyan kódszámú játékosok és különböző ligákon belül lehetnek ugyanolyan nevű csapatok.

4.5. E/K-diagram átírása relációs modellé

Első megközelítésben egy E/K-terv átalakítása egy relációs adatbázissára egyszerűen a következő:

- minden egyedhalmazt írunk át az attribútumaival definiált relációvá;

- Egy kapcsolatot pedig helyettesítsünk egy olyan relációval, amelynek az attribútumai a kapcsolatban álló egyedhalmazok kulcsainak felelnek meg.

Annak ellenére, hogy ez a két szabály az esetek nagy többségét kezeli, van még néhány speciális eset, amelyekkel foglalkoznunk kell, beleértve az alábbiakat:

1. A gyenge egyedhalmazokat nem lehet simán relációkká alakítani.
2. Az öröklési („az-egy”) kapcsolatokat és alosztályokat óvatosan kell kezelni.
3. Néha jól tesszük, ha két relációt összevonunk, különösen egy E egyedhalmazhoz tartozó relációt egy olyan relációval, mely az E és néhány egyedhalmaz közötti sok-egy kapcsolathoz tartozik.

4.5.1. Egyedhalmazok átírása relációkká

Először vegyük egy olyan egyedhalmazt, amely nem gyenge. A gyenge egyedhalmazok kezelésére szükséges módosításokat később a 4.5.4. alfejezetben fogjuk megnézni. minden nem gyenge egyedhalmazhoz létrehozunk egy relációt ugyanezzel a névvel és ugyanezzel az attribútumhalmazzal.⁵ Ebben a relációban nincsenek jelölve, hogy mely kapcsolatokban vesz részt az egyedhalmaz, a kapcsolatokat külön relációkban fogjuk kezelni, ahogyan ezt a 4.5.2. alfejezetben tárgyaljuk.

4.24. példa. Vegyük a 4.17. ábra *Filmek*, *Színészek* és *Stúdiók* egyedhalmazait, amelyet itt újból megismétünk a 4.23. ábrán. A *Filmek* egyedhalmaz attribútumai a *filmcím*, *év*, *hossz*, *műfaj*. A relációra átírt eredmény, a *Filmek* reláció olyan, mint a 2.1. ábra *Filmek* relációja, amellyel a 2.2. alfejezetben kezdtünk foglalkozni.

Most vegyük a 4.23. ábra *Színészek* egyedhalmazát. Két attribútuma van, a *név* és a *cím*. Így a megfelelő *Színészek* reláció sémája *Színészek(név, cím)* lesz és egy lehetséges előfordulását az alábbi formában kapjuk:

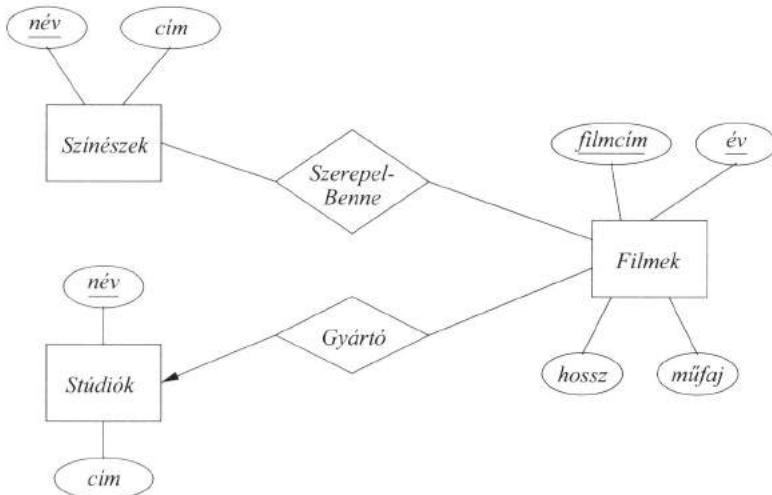
<i>név</i>	<i>cím</i>
Carrie Fisher	123 Maple St., Hollywood
Mark Hamill	456 Oak Rd., Brentwood
Harrison Ford	789 Palm Dr., Beverly Hills

□

4.5.2. E/K-kapcsolatok átírása relációkká

Az E/K-modellben a kapcsolatokat szintén relációkkal reprezentáljuk. Egy adott kapcsolathoz rendelt R reláció a következő attribútumokat tartalmazza:

⁵ Az egyedeknek megfeleltetjük az attribútumaihoz rendelt értékekből álló sort. Az így keletkező reláció-előfordulás az (erős) egyedhalmaz kulesa miatt nem tartalmazhat két azonos sort, tehát korrekt. (A fordító megjegyese.)



4.23. ábra. A filmadatbázis E/K-diagramja

1. Az R reláció sémájába belevesszük az R kapcsolatban részt vevő minden egyedhalmaz kulcsát képező attribútumot vagy attribútumokat.
2. Ha a kapcsolatnak vannak attribútumai, akkor ezek szintén az R reláció attribútumai lesznek.

Ha valamelyik egyedhalmaz többszörösen benne van a kapcsolatban, különböző szerepben, akkor minden kulcsattribútuma annyiszor kell szerepeljen, ahányfélé szerepe van az egyedhalmaznak. Ilyenkor át kell neveznünk ezen attribútumokat, hogy elkerüljük a nevek ismétlődését egy sémán belül. Általánosságban, ha ugyanaz az attribútumnév kétszer vagy többször szerepel magának az R -nek és az R kapcsolatban részt vevő egyedhalmazoknak a kulcsattribútumai között, akkor át kell neveznünk, hogy elkerüljük az ismétlődést.

4.25. példa. Most vegyük a 4.23. ábra *Gyártó* kapcsolatát. Ez a kapcsolat a *Filmek* és a *Stúdiók* egyedhalmazokat kapcsolja össze. Így a *Gyártó* relációsémájában használjuk a *Filmek* kulcsát, azaz a *filmcím* és *év* attribútumokat és a *Stúdiók* kulcsát, amely a *név* attribútum:

Gyártó(filmcím, év, stúdióNév)

Az egyértelműség kedvéért itt a *stúdióNév* attribútumot választottuk, ami megfelel a *Stúdiók* *név* attribútumának. A reláció előfordulására egy példa a következő:

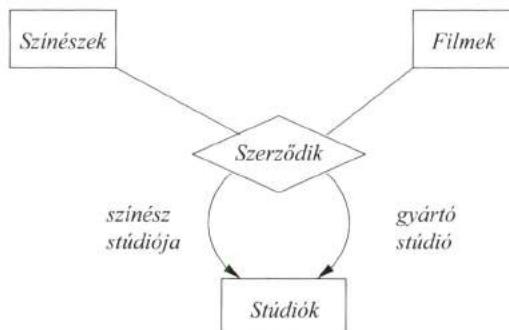
<i>filmcím</i>	<i>év</i>	<i>stúdióNév</i>
Csillagok háborúja	1977	Fox
Elfújta a szél	1939	MGM
Wayne világa	1992	Paramount

□

filmcím	év	színészNév
Csillagok háborúja	1977	Carrie Fisher
Csillagok háborúja	1977	Mark Hamill
Csillagok háborúja	1977	Harrison Ford
Elfújta a szél	1939	Vivien Leigh
Wayne világa	1992	Dana Carvey
Wayne világa	1992	Mike Meyers

4.24. ábra. A *SzerepelBenne* kapcsolat relációja

4.26. példa. Hasonlóan írjuk át a 4.23. ábra *SzerepelBenne* kapcsolatát relációvá, amelynek attribútumai *filmcím* és *év* (a *Filmek* kulcsa) és a *színészNév*, amely a *Színészek* egyedhalmaz kulcsa. A 4.24. ábrán a *SzerepelBenne* relációra találunk egy példát. □

4.25. ábra. A *Szerződik* kapcsolat

4.27. példa. A többirányú kapcsolatok szintén könnyen átírhatók relációkká. Tekintsük a 4.6. ábra négyirányú *Szerződik* kapcsolatát, amelyet itt a 4.25. ábrán újból bemutatunk. Ez a kapcsolat tartalmazza a színészt, a filmet és két stúdiót, az első stúdió, amelyikkel a színész szerződött, a második pedig, ahova a színész szerződést kötött a filmre. Ezt a kapcsolatot a *Szerződések* relációval reprezentáljuk, amelynek a sémáját az alábbi négy egyedhalmaz kulcsaiban szereplő attribútumok alkotják:

1. *színészNév*, a színész kulcsa.
2. *filmcím* és *év*, a film két attribútumából álló kulcsa.

3. színészStúdió kulcs jelzi az első stúdió nevét; korábban feltettük, hogy a Stúdiók egyedhalmaz kulcsa a stúdió neve.
4. gyártóStúdió kulcs jelzi annak a stúdiónak a nevét, amelyik elkészíti a filmet a szóban forgó színész közreműködésével.

Így tehát a séma a következőképpen alakul:

Szerződések(színészNév, filmcím, év, színészStúdió, gyártóStúdió)

Megjegyezzük, hogy találékonyan kellett megválasztanunk az attribútumneveket a relációsémánkban, hogy egyik attribútumként se használjuk azt, hogy „név”, ugyanis akkor nem lett volna nyilvánvaló, hogy az a színészre vagy a stúdióra, és az utóbbi esetben pedig melyik stúdióra vonatkozik. Ha a *Szerződések* egyedhalmazhoz is tartoznának további attribútumok, mint például *fizetés*, akkor ezeket is hozzá kellene adnunk a *Szerződések* relációsémához. □

4.5.3. Relációk kombinációja

A relációk, melyeket az egyedhalmazok és kapcsolatok relációkká alakítása során kapunk, néha nem a legmegfelelőbbek a rendelkezésre álló adatokhoz. Egy gyakori esethez juthatunk, ha egy E egyedhalmazhoz van egy R sok-egy kapcsolat E -ből F -be. Ekkor az E -hez, illetve az R -hez tartozó mindegyik reláció sémája tartalmazni fogja az E kulcsát. Továbbá, az E reláció sémája az E nem kulcs típusú attribútumait is tartalmazza. R relációja pedig magában foglalja R összes attribútumát és az F kulcsattribútumait is. Mivel R sok-egy kapcsolat, ezért E kulcsa egyértelműen meghatározza minden attribútumának az értékét, és így a relációkat egyesíthetjük egyetlen relációba egy olyan séma használatával, amely tartalmazza:

1. E minden attribútumát,
2. F kulcsattribútumait és
3. az R kapcsolathoz tartozó összes attribútumot.

Azon E -beli e egyedek, amelyek F egyetlen egyedéhez sem kapcsolódnak, a hozzájuk tartozó sorban, a 2. és 3. típusú attribútumaikban nullértéket tartalmaznak.

4.28. példa. A 4.25. példában relációvá alakított *Gyártó* egy sok-egy kapcsolat a *Filmek* és *Stúdiók* között az általunk használt filmes példában. A *Filmek* egyedhalmazból kapott relációt már megtárgyalunk a 4.24. példában. Ezen két relációt egyesíthetjük egyetlen relációba, az összes attribútumuk egy relációsémába foglalásával. Ezt a helyzetet szemlélteti a 4.26. ábrán szereplő reláció. □

filmcím	év	hossz	műfaj	stúdióNév
Csillagok háborúja	1977	124	sci-fi	Fox
Elfújta a szél	1939	239	dráma	MGM
Wayne világa	1992	95	vígjáték	Paramount

4.26. ábra. A Filmek reláció kombinációja a Gyártó relációval

Az, hogy ilyen módon egyesítsük-e vagy sem a relációkat, valójában döntés kérdése. Van viszont néhány előnye annak, hogy az E egyedhalmaz kulcsától függő összes attribútum egy helyen fordul elő, még akkor is, ha van néhány sok-egy kapcsolat E -ből különböző egyedhalmazokba. Jóval hatékonyabb például, ha a lekérdezés válaszának attribútumai egy relációban szerepelnek, mintha több relációból származnának. Néhány E/K-terven alapuló tervezőrendszer automatikusan egyesíti az ilyen relációkat.

Másrészről elköpzelhető, hogy valaki az E relációt egyesíteni szeretné egy olyan R kapcsolathoz tartozó relációval, ami ugyan tartalmazza E -t, de nincs sok-egy kapcsolat az E -ből valamilyen más egyedhalmazba. Ez azonban kockázatos próbálkozás, mivel gyakran redundanciához vezethet, ahogy azt a következő példa is mutatja.

4.29. példa. Annak megértéséhez, hogy mi romolhat el, tekintsük a 4.26. ábrán lévő reláció egyesítését a *SzerepelBenne* sok-sok kapcsolathoz kapott relációval, amely a 4.24. ábrán szerepelt. Ekkor az egyesített reláció úgy néz ki, mint a 3.2. ábrán látható reláció, melyet itt a 4.27. ábrán megismételtünk. Ahogy a 3.3.1. alfejezetben már tárgyaltuk, ez a reláció anomáliákkal terhelt, melyeket a normalizálás folyamán kell megszüntetnünk. □

filmcím	év	hossz	műfaj	stúdióNév	színészNév
Csillagok háborúja	1977	124	sci-fi	Fox	Carrie Fisher
Csillagok háborúja	1977	124	sci-fi	Fox	Mark Hamill
Csillagok háborúja	1977	124	sci-fi	Fox	Harrison Ford
Elfújta a szél	1939	231	dráma	MGM	Vivien Leigh
Wayne világa	1992	95	vígjáték	Paramount	Dana Carvey
Wayne világa	1992	95	vígjáték	Paramount	Mike Meyers

4.27. ábra. A Filmek reláció a szereplőkre vonatkozó információval kiegészítve

4.5.4. Gyenge egyedhalmazok kezelése

Ha egy E/K-diagramban gyenge egyedhalmaz van, akkor három dolgot kell eltérő módon tennünk.

1. Magához a W gyenge egyedhalmazhoz tartozó relációt tartalmaznia kell nem csupán W attribútumait, hanem más egyedhalmazok kulcsatttribútumait is, amelyek segítik W kulcsának kialakítását. Ezeket a segítő

egyedhalmazokat könnyen felismerhetjük arról, hogy a W -hez kitéjesítő (dupla keretes rombusszal jelölt) kapcsolattal kötődnek.

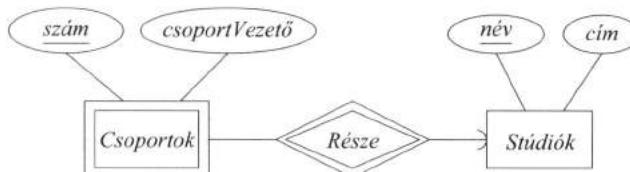
2. Bármely olyan kapcsolat relációjában, amelyben a W gyenge egyedhalmaz részt vesz, W kulcsának az összes kulcsattribútumát használjuk, azokat az egyéb egyedhalmazokbeli attribútumokat is beleérte, amelyek hozzájárultak W kulcsához.
3. A kitéjesítő (dupla keretes) kapcsolatokat a W gyenge egyedhalmaztól valamely más egyedhalmazba, vagyis amelyek hozzájárulnak W kulcsához, egyáltalán nem kell relációkká átírnunk. Ez – mint ahogyan a 4.5.3. alfejezetben tárgyalunk – amiatt van így, mert egy R sok-egy kapcsolat relációjának attribútumai vagy már eleve W relációjának attribútumai, vagy pedig (R saját attribútumai esetén) kombinálhatók W relációsémájával.

Természetesen amikor további attribútumokat veszünk fel egy gyenge egyedhalmaz kulcsába, akkor vigyáznunk kell, hogy ne használjuk ugyanazt a nevet kétszer. Ha szükséges, nevezzük át az összes ilyen attribútumot.

4.30. példa. Tekintsük a 4.20. ábrán szereplő *Csoporthok* gyenge egyedhalmazt, amelyet a 4.28. ábrán megismétlik. Ebből a diagramból három relációt kapunk, amelyeknek a sémai:

Stúdiók(név, cím)
Csoporthok(szám, stúdióNév, csoportVezető)
Része(szám, stúdióNév, név)

Az első relációt, a *Stúdiók*-at természetes módon írtuk át az azonos nevű egyedhalmazból. A második reláció, a *Csoporthok* gyenge egyedhalmazból származik. Ennek a relációjának az attribútumai a *Csoporthok* kulcsattribútumai, és egy nem kulcs attribútum, a *csoportVezető*. A *Csoporthok* relációjában a *stúdióNév*-et választották attribútumként, amely megfelel a *Stúdiók* egyedhalmaz *név* attribútumának.



4.28. ábra. A *Csoporthok*, avagy példa gyenge egyedhalmazra

A harmadik reláció, az *Része*, az azonos nevű kapcsolatból származik. Mint minden, egy E/K-kapcsolatot a relációs modellben olyan relációval reprezentálunk, amelynek sémája a kapcsolt egyedhalmazok kulcsaiból áll. Ebben az esetben az *Része* attribútumai a *szám* és *stúdióNév*, a *Csoporthok* gyenge egyedhalmaz kulcsa, és a *név* attribútum, amely a *Stúdiók* kulcsa. Megjegyezzük, hogy

mivel az *Része* sok-egy kapcsolat, a stúdiót azonosító **stúdióNév** biztos, hogy megegyezik ugyanahhoz a stúdióhoz tartozó **név**-vel.

Például tegyük fel, hogy a hármas számú csoport (#3) a Disney stúdió csoportjainak egyike. Ekkor az *Része* kapcsolathalmaz tartalmazza a következő párt:

$$(Disney \text{ csoport } \#3, Disney)$$

Ez a pár az alábbi sort adja az *Része* relációhoz:

$$(3, Disney, Disney)$$

Vegyük észre, hogy e sor **stúdióNév** és **név** attribútumai megegyeznek, mint ahogy meg is kell egyezniük. Következésképpen „egybeolvashatjuk” az *Része* reláció **stúdióNév** és **név** attribútumait, így a következő, egyszerűbb sémát kapjuk:

$$\mathbf{Része}(\mathbf{szám}, \mathbf{név})$$

Tulajdonképpen el is hagyhatjuk az *Része* relációt, mivel attribútumhalmaza így része a *Csoportok* reláció attribútumhalmazának. \square

A 4.30. példában megfigyelt jelenség, vagyis hogy a kiteljesítő kapcsolathoz nem szükséges relációt megadni, általában is igaz a gyenge egyedhalmazokra. A gyenge egyedhalmazokra vonatkozóan a következő módosított szabályokat állíthatjuk fel.

- Ha W egy gyenge egyedhalmaz, akkor készítsünk W számára egy olyan relációt, amelynek sémaja a következőkből tevődik össze:
 1. W attribútumaiból,
 2. W kiteljesítő kapcsolatainak attribútumaiból,
 3. azon E egyedhalmazok kulcsattribútumaiból, melyekhez W kiteljesítő kapcsolattal kötődik (ezek sok-egy típusú kapcsolatok).

A névkonfliktusok elkerülése céljából, ha szükséges, nevezzük át az attribútumokat.

- Ne készítsünk relációkat W kiteljesítő kapcsolataihoz.

4.5.5. Feladatok

4.5.1. feladat. Alakítsuk át a 4.29. ábra E/K-diagramját relációs adatbázis-sémává.

Relációk részhalmazsémákkal

A 4.30. példa alapján azt gondolhatnánk, hogy amennyiben egy R relációt van egy olyan attribútumhalmaza, amely egy másik S reláció attribútumainak részhalmaza, akkor R eltávolítható lesz. Ez viszont nem teljesen igaz. R tartalmazhat S -ben nem szereplő információkat, ugyanis S további attribútumai nem teszik lehetővé számunkra, hogy egy R -beli sorból S -beli sorra következzünk.

Példaként tekintsük azt, hogy az adóhatóság a potenciális adófizetők neveit és társadalombiztosítási számaikat tartalmazó **Emberek**(név, TAJ#) relációt akarja fenntartani még azokra is, akiknek nincs bevételre, illetve adó-visszaigénylést sem adtak be. Ehhez szükségük lehet még egy **AdóFizetők**(név, TAJ#, összeg) relációra, ami tartalmazza minden olyan személy befizetett adójának mértékét, aki adott be visszaigénylést az adott évben. Az **Emberek** séma részhalmaza az **AdóFizetők** sémájának, mégis lehetnek olyan értékek, nevezetesen olyan társadalombiztosítási számokkal rendelkező egyének, akik szerepelnek az **Emberek**-ben, de az **AdóFizetők**-ben nem.

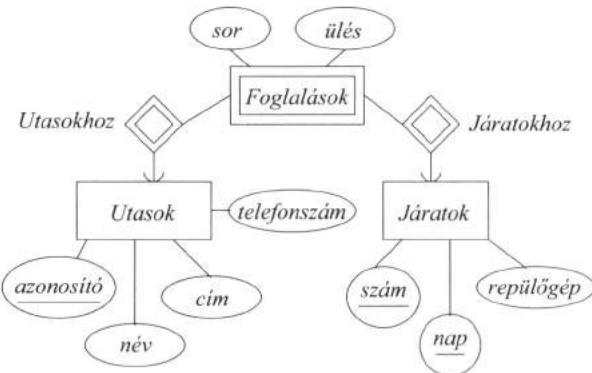
Valójában még azonos attribútumhalmazok esetén is eltérő lehet a szemantika, ezért ezeknek a sorait sem fésülhetjük össze. A példa kedvéért vegyük a **Színészek**(név, cím) és **Stúdiók**(név, cím) relációkat. A séma ugyan egyformának tűnik, mégsem rakhatjuk a színészek sorait a stúdiók sorai közé (vagy fordítva).

Másrésztől, ha a két reláció egy gyenge egyedhalmaz konstrukciójából származik, akkor nem lehetnek a szűkebb attribútumhalmazzal bíró relációknak további értékei. Ennek oka, hogy a kiteljesítő kapcsolatból származó relációsorok egy az egyben megegyeznek a gyenge egyedhalmazból származó relációsorokkal. Ilyenkor rutinszerűen az egyik relációt eltávolítjuk.

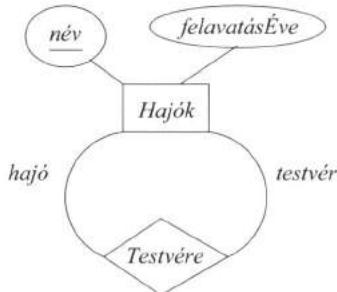
! 4.5.2. feladat. A 4.29. ábrán látható *Foglalások* gyenge egyedhalmazhoz más E/K-diagram is elképzelhető. Vegyük észre, hogy egy foglalást egyértelműen azonosít a járat száma, dátuma, a sor és az ülés száma. Így tehát az utas nem játszik szerepet egy foglalás azonosításában.

- Módosítsuk a 4.29. ábrát a fenti nézőpont szerint.
- Alakítsuk át az a) feladat megoldását relációkká. Ugyanazt az eredményt kapjuk-e, mint a 4.5.1. feladatban?

4.5.3. feladat. A 4.30. ábra E/K-diagramja hajókat reprezentál. *Testvéreknek* hívjuk azokat a hajókat, amelyeket ugyanazon terv alapján építettek. Alakítsuk át ezt a diagramot relációs adatbázissémává.



4.29. ábra. A repülőgépes helyfoglalás E/K-diagramja



4.30. ábra. A testvérhajók egy lehetséges E/K-diagramja

4.5.4. feladat. Alakítsuk át a következő E/K-diagramokat relációs adatbázis-sémákra:

- a) 4.22. ábra
- b) 4.4.1. feladat válasza
- c) 4.4.4 a) feladat válasza
- d) 4.4.4 b) feladat válasza

4.6. Osztályhierarchia átalakítása relációkká

Egyedhalmazok öröklési („az-egy” típusú) kapcsolattal megadott hierarchiája esetén több stratégia közül választhatunk a relációkká alakításhoz. Emlékeztetőül a következőket feltételezzük:

- A hierarchiának van egy gyökéregyedhalmaza.
- Ez az egyedhalmaz rendelkezik kulccsal, amely a hierarchiában reprezentált összes egyed azonosítására szolgál.
- Egy adott egyednek lehetnek *komponensei*, amelyek a hierarchiában különféle egyedhalmazokhoz tartozhatnak. Ezek az egyedhalmazok a struktúra tetszőleges részfáját alkothatják, melynek a gyökéregyedhalmazt tartalmaznia kell.

Az átalakítási stratégiák elvei az alábbiak:

1. *Az E/K szempontjainak követése.* A hierarchiában lévő összes E egyedhalmazra külön-külön hozunk létre egy relációt, amely tartalmazza a gyökérben lévő kulcsattribútumokat és E minden attribútumát.
2. *Az egyedek egy egyszerű osztályhoz tartozó objektumkénti kezelése.* Az összes, a gyökeret tartalmazó lehetséges részfára hozunk létre egy relációt, melynek a sémaja magában foglalja a részfában szereplő összes egyedhalmaznak az attribútumait.
3. *Nullértékek használata.* A hierarchiában szereplő összes egyedhalmaz összes attribútumával hozunk létre egyetlen relációt. minden egyedet egy sor fog reprezentálni, amely nullértéket vesz fel azon oszlopokon, ahol az egyed nem rendelkezik attribútumértékkel. (A nullérték az érték hiányát jelzi.)

Lépésenként az összes megközelítést megvizsgáljuk.

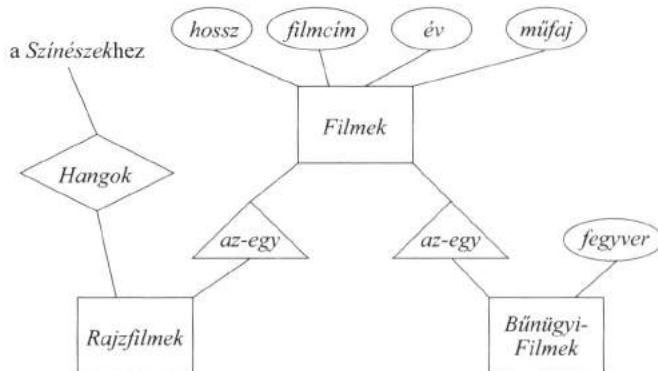
4.6.1. E/K-típusú átalakítás

Az első megközelítésünkben a megszokott módon minden egyedhalmazhoz készítünk egy relációt. Ha az E egyedhalmaz nem a hierarchia gyökérkcsúcsa, akkor az E -hez tartozó reláció tartalmazza a gyökér kulcsattribútumait is. Ha E emellett szerepel egy kapcsolatban is, akkor a kapcsolatban is ezeket a kulcsattribútumokat fogjuk használni E egyedeinek azonosításához.

Megjegyezzük, hogy annak ellenére, hogy az öröklési (az-egy) kapcsolatot is kapcsolatként kezeltük, mégsem hasonlít a többi kapcsolatra abban az értelemben, hogy egy egyszerű egyed komponenseit kapcsolja össze, nem pedig egyedeket egymással. Az öröklési kapcsolathoz tehát nem készíthető reláció.

4.31. példa. Tekintsük a 4.10. ábrán lévő hierarchiát, amelyet a 4.31. ábrán újra bemutatunk. A relációknak a hierarchiában szereplő négy különböző egyedet kell reprezentálniuk:

1. **Filmek(filmcím, év, hossz, műfaj).** A 4.24. példában már megállapított reláció, ahol minden film egy sorral van ábrázolva.



4.31. ábra. A filmek hierarchiája

2. **BűnögyiFilmek(filmcím, év, fegyver).** Az első két attribútum a filmek kulcsa, az utolsó pedig az egyedhalmaz saját attribútuma. A bűnögyi filmekhez mind itt, mind a *Filmek*-ben is tartozik egy-egy sor.
3. **Rajzfilmek(filmcím, év).** Ez a reláció a rajzfilmek egy halmazát jelenti. A filmek kulcsán kívül nincs más attribútuma, mivel a rajzfilmekhez tartozó kiegészítő információt a *Hangok* kapcsolat tartalmazza. A rajzfilmekhez itt is és a *Filmek*-ben is tartozik egy-egy sor.

Megjegyezzük, hogy a negyedik típusú filmnek – amely rajzfilm és egyben bűnögyi film is – mind a három relációban van sora.

Szükségünk lehet ezeken kívül még egy *Hangok(filmcím, év, színész-Név)* relációra, amely a *Színészek* és a *Rajzfilmek* közötti kapcsolatot valósítja meg. Ebben az utolsó attribútum a *Színészek* kulcsa lesz, az első kettő pedig a *Rajzfilmek* kulcsát alkotja majd.

A *Roger nyúl a páccban* című filmhez például minden relációban tartozik sor. Az alapinformációi a *Filmek* relációban, a fegyver a *BűnögyiFilmek* relációban, a hangjukat adó színészek pedig a *Hangok* relációban találhatók.

Vegyük észre, hogy a *Rajzfilmek* reláció sémája a *Hangok* reláció sémájának részhalmaza. A legtöbb esetben egy *Rajzfilmek*-hez hasonló relációt eltávolítanánk, mivel nem szerepel benne olyan információ, amit a *Hangok* ne tartalmazna. Ugyanakkor lehetnek néma rajzfilmek is az adatbázisban, melyekhez nem tartoznak hangok, és így elveszítenénk azt az információt, hogy ezek rajzfilmek.

4.6.2. Egy objektumorientált megközelítés

Egy másik stratégia lehet az öröklési hierarchia relációvá alakítása során, ha fel soroljuk a hierarchia összes lehetséges részfáját. Ezek mindegyikére létrehozunk egy-egy relációt, amely azon egyedeiket reprezentálja, amelyeknek pontosan az

adott részfában vannak komponensei. A reláció sémája az összes, a részfában szereplő egyed komponenseit tartalmazza. Erre objektumorientált megközöltsésként hivatkozunk, mivel az a feltevés motiválta, hogy az egyedek egy és csak egy osztályhoz tartozó „objektumok” legyenek.

4.32. példa. Tekintsük a 4.31. ábra hierarchiáját. A gyökeret is beleértve négy lehetséges részfa lehetséges:

1. A *Filmek* önmagában.
2. Csak a *Filmek* és a *Rajzfilmek*.
3. Csak a *Filmek* és a *BűnügyiFilmek*.
4. Mindhárom egyedhalmaz.

Mind a négy „osztályhoz” relációt kell készítenünk. E sémákban most van némi azonosság, mivel csak a bűnügyi filmekhez tartozik saját speciális attribútum:

```
Filmek(filmcím, év, hossz, műfaj)
FilmekRF(filmcím, év, hossz, műfaj)
FilmekBF(filmcím, év, hossz, műfaj, fegyver)
FilmekRFBF(filmcím, év, hossz, műfaj, fegyver)
```

Ha a *Rajzfilmek*nek saját attribútumai lennének, akkor minden reláció különböző attribútumhalmazzal rendelkezne. Mivel jelen esetben nem ez a helyzet, ezért kombinálhatjuk a *Filmek*-et a *FilmekRF*-fel (azaz létrehozhatunk egy relációt a nem bűnügyi filmekre), illetve a *FilmekBF*-et a *FilmekRF*-fel (azaz létrehozhatunk egy relációt minden bűnügyi filmre), habár így elveszítjük azt az információt, hogy melyik film lesz rajzfilm.

Azt is meg kell még fontolnunk, hogyan kezeljük a *Rajzfilmek* és *Színészek* közötti *Hangok* kapcsolatot. Ha a *Hangok* sok-egy kapcsolatban álltak volna a *Rajzfilmek*kel, akkor a *FilmekRF*-hez és a *FilmekRFBF*-hez hozzáadhattunk volna egy *hang* attribútumot, amely ábrázolhatná a *Hangok* kapcsolatot, és ebből adódóan minden reláció különböző lenne. Viszont a *Hangok* sok-sok típusú, ezért önálló relációkat kell létrehoznunk ehhez a kapcsolathoz. Ennek a sémája – mint minden reláció – egy kulcsattribútumon keresztül kapcsolódik az egyedhalma-

zokhoz, és így a következő séma alkalmas lesz erre:

```
Hangok(filmcím, év, színészNév)
```

Felmerülhet, hogy szükség van-e két ilyen reláció létrehozására, ahol az egyik a nem bűnügyi témájú rajzfilmekkel, a másik pedig a bűnügyi témájú rajzfilmekkel kapcsolja össze a szinkronhangokat. Ennek viszont jelen esetben semmi hasznára sem lenne. □

4.6.3. Nullértékek használata relációk egyesítéséhez

Még egy megközelítés maradt arra, hogy egyedhalmazok hierarchiáját relációs modellben ábrázoljuk. Ha megengedjük a relációkban (az SQL nullértékéhez hasonló) NULL érték használatát, akkor az egyedhalmazok teljes hierarchiáját egyetlen egyszerű reláció segítségével ábrázolhatjuk. Ez a reláció a hierarchia összes egyedhalmazához tartozó minden attribútumot tartalmaz. Egy egyedet pedig egy sorral ábrázolunk. Ennek a sornak azon értékei, amelyek nem definiáltak az adott egyedre, NULL értéket vesznek fel.

4.33. példa. Ha ezt a megközelítést alkalmazzuk a 4.31. ábra diagramjára, akkor a következő sémájú relációt kapjuk:

Filmek(filmcím, év, hossz, műfaj, fegyver)

A nem bűnügyi filmek sorai a **fegyver** komponensben tartalmazhatnak NULL értéket. Szükségünk lehet még a 4.32. példához hasonlóan egy **Hangok** relációra is, amely a rajzfilmeket kapcsolja össze a szinkronizálást végző színészekkel. □

4.6.4. A megközelítések összehasonlítása

Mindhárom tárgyalt megközelítésnek, amelyekre a továbbiakban „E/K-elvű”, „objektumelvű”, illetve „nullértékes”-ként hivatkozunk, vannak előnyei és hátrányai is. A most következőben listaszerűen felsoroljuk a legfontosabb tényezőket:

1. A válaszadás költséges a sok relációt érintő lekérdezésekre, ezért inkább azt részesítjük előnyben, amikor egy lekérdezés eredményéhez minden szükséges attribútum egyetlen relációban szerepel. Mivel a nullértékes megközelítés csak egy, minden attribútumot magában foglaló relációt használ, így ebből a szempontból ez rendelkezik ezzel az előnyivel. A másik két megközelítésnek más típusú lekérdezések esetén van előnye. Például:
 - a) A „Mely 2008-as filmek voltak 150 percnél hosszabbak?” típusú kérdést a 4.31. példa E/K-elvű megközelítésből származó **Filmek** relációjából közvetlenül megválaszolhatjuk. A 4.32. példa objektumelvű megközelítése esetén viszont a **Filmek**, a **FilmekRF**, a **FilmekBF** és a **FilmekRFBF** vizsgálatára is szükségünk lesz, mivel egy hosszú film ezek bármelyikében benne lehet.
 - b) Másrészről viszont a „Milyen fegyvereket használtak a 150 percenél hosszabb rajzfilmekben?” típusú kérdés esetén az E/K-elvű megközelítés használata gondot okozhat. Hozzá kell férnünk a **Filmek**-hez a 150 percnél hosszabb filmek meghatározásához, a **Rajzfilmek**-hez annak ellenőrzésére, hogy egy film rajzfilm-e, illetve a **BűnügyiFilmek**-hez a fegyver megkereséséhez. Az objektumelvű megközelítés esetén pedig csak a **FilmekRFBF**-hez kell hozzáfernni, ahol minden szükséges információ megtalálható.

2. Nem szeretnénk túl sok relációt használni. Így itt újra a nullértékes módszer jeleskedik, mivel annál csak egy relációra van szükség. A másik két módszer között viszont vannak különbségek, ugyanis míg az E/K-elvű megközelítésnél a hierarchiában szereplő minden egyedhalmaz csak egyetlen relációt használt, addig az objektumelvű megközelítésnél, ha egy gyökér és n darab gyerek csúcs (összesen $n + 1$ darab egyedhalmaz) adott, akkor 2^n különböző egyedesztályt kapunk, és ennyi relációra lenne szükségünk.
3. Minimalizálni szeretnénk a területet és elkerülni az információk ismétlődését. Mivel az objektumelvű megközelítés minden egyedhez egyetlen sort használ, és ez a sor csak az egyeden értelmezett attribútumokat tartalmazza, így ez a megközelítés biztosítja a lehetséges minimális helyfoglalást. A nullértékes megközelítés is hasonló módon egyedenként egy sort tárol, viszont ezek a sorok „hosszúak” abban az értelemben, hogy minden attribútumra van komponensük, akár értelmezettek az adott egyedre, akár nem. Ha sok egyedhalmaz van a hierarchiában, illetve ezen egyedhalmazok attribútumai sokban különböznek egymástól, akkor a nullértékes megközelítés használata esetén nagy, kihasználatlan tárterület lesz lekötve. Az E/K-elvű módszer több sort rendel egy-egy egyedhez, viszont csak a kulcsattribútumokat ismétli. Azaz ez a módszer akár kevesebb vagy több területet is használhat, mint a nullértékes.

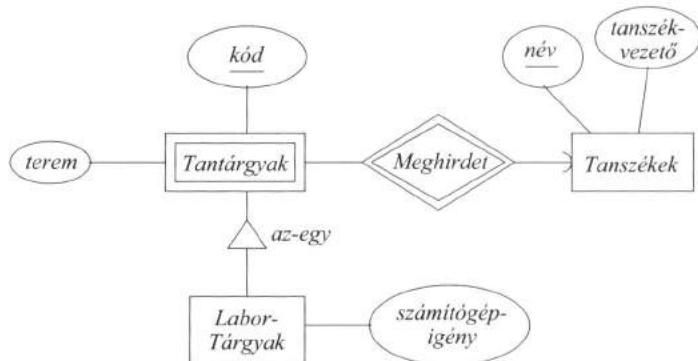
4.6.5. Feladatok

4.6.1. feladat. Alakítsuk át a 4.32. ábra E/K-diagramját relációs adatbázis-sémává, a különböző megközelítésekkel:

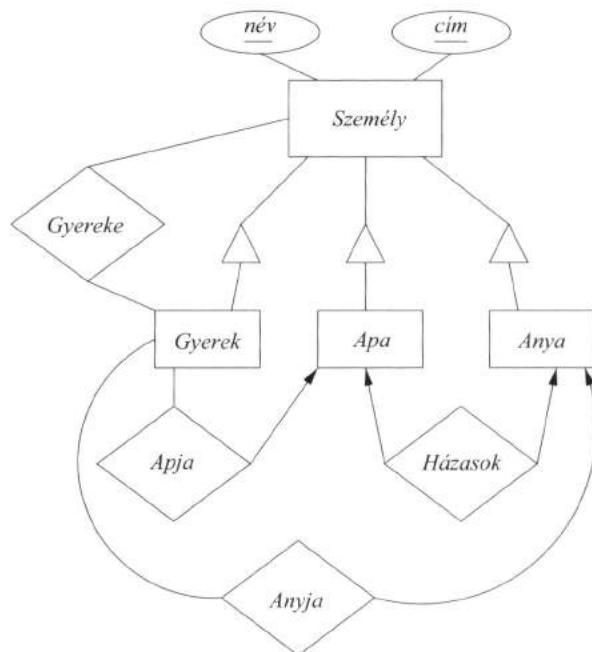
- a) E/K-elvű,
- b) objektumelvű,
- c) nullértékes módszerrel.

! 4.6.2. feladat. Alakítsuk át a 4.33. ábra E/K-diagramját relációs adatbázis-sémává, a különböző megközelítésekkel:

- a) E/K-elvű,
- b) objektumelvű,
- c) nullértékes módszerrel.



4.32. ábra. A 4.6.1. feladathoz tartozó E/K-diagram



4.33. ábra. A 4.6.2. feladathoz tartozó E/K-diagram

4.6.3. feladat. A 4.1.7. feladat megoldásaként kapott E/K-diagramot alakít-suk át relációs adatbázissémává:

- a) E/K-elvű,
- b) objektumelvű,
- c) nullértékes módszerrel.

! 4.6.4. feladat. Tegyük fel, hogy adott e darab egyedhalmaz öröklési hierarchiája. minden egyedhalmaznak van a darab attribútuma és ezek közül k darab a gyökérben az összes egyedhalmaz kulcsát alkotja. Adjunk képletet, hogy relációsémák kódolására i) legalább, illetve legfeljebb hány reláció keletkezik, ii) összességében legalább, illetve legfeljebb hány komponensből álló sor(ok)ra képződik egy egyed, ha az alábbi átalakítási módszert használjuk:

- a) E/K-elvű,
- b) objektumelvű,
- c) nullértékes módszerrel.

4.7. Bevezetés az UML-be

Az UML-t (*Unified Modeling Language – Egységesített Modellező Nyelv*) eredetileg az objektumorientált szoftvertervezési stílushoz igazodó grafikus jelölőkészletnek fejlesztették ki. Később kibővítették olyan módosításokkal, amelyekkel az adatbázisok leírásának népszerű eszköze lett. Az UML-nek ezt a részét fogjuk tanulmányozni. A többszörös kapcsolatok kivételével az UML lehetőségei sokban hasonlítanak az E/K-modell lehetőségeire. Az UML is lehetőséget ad az egyedhalmazok – mint valódi osztályok – kezelésére, azok metódusaival és adataival. A 4.34. ábrán összefoglaljuk az E/K- és az UML-modellek közös koncepcióját az általuk használt terminológiai különbségekkel.

UML	E/K-modell
Osztály	Egyedhalmaz
Társítás	Bináris kapcsolat
Társításosztály	A kapcsolat attribútumai
Alosztály	Osztályhierarchia
Aggregáció (összesítés)	Sok-egy kapcsolat
Kompozíció (összeállítás)	Sok-egy kapcsolat hivatkozásiépség-megszorítással

4.34. ábra. Az UML- és az E/K-terminológiák összehasonlítása

5. fejezet

Algebrai és logikai lekérdező nyelvek

A jelen fejezet során a relációs adatbázisok modellezése helyett a programozásra fektetjük a hangsúlyt. A tárgyalást két absztrakt programozási nyelv leírásával kezdjük: egy algebraival, illetve egy logikán alapulóval. Az algebrai programozási nyelvet (vagy relációs algebrát) már a 2.4. alfejezetben bemutattuk, amely során láthattuk, milyen műveletek szerepelnek a relációs modellben. Most azonban többről van szó, mint az ott bemutatott algebráról, ezért a 2.4. alfejezetben tárgyalt halmaz alapú algebrát kiterjesztjük multihalmazokra. Ez a megfogalmazás már jobban fogja tükrözni a relációs modell gyakorlatban használt megvalósítási módjait. A kiterjesztés miatt a korábbiakon felül lesz még néhány újabb művelet is, mint például a reláció oszlopainak összegzése (például átlag).

A fejezetet egy másik, logikán alapuló lekérdező nyelvnek a formalizmusával zárjuk le. Ezt *Datalognak* fogjuk nevezni. Ebben a nyelvben már megfogalmazhatunk a kívánt eredménynek megfelelő lekérdezéseket is anélkül, hogy az eredményt kiszámító olyan algoritmust adnánk, amely a relációs algebrai megközelítésnél elhanyagolhatatlan volt.

5.1. Relációs műveletek multihalmazokon

Ebben az alfejezetben a relációkat inkább multihalmaznak tekintjük, mint halmaznak. Éppen ezért ugyanaz a sor többször is megjelenhet egy adott relációban. Néhány relációs műveletet azonban át kell fogalmaznunk, amennyiben a relációkra multihalmazokként tekintünk. Első lépésben tekintsük az alábbi egyszerű példát, amelyben a reláció egy valódi multihalmaz, azaz nem egy halmaz.

5.1. példa. Az 5.1. ábrán látható reláció tulajdonképpen sorokból álló multihalmaz, amelyben az (1, 2) sor háromszor és a (3, 4) sor egyszer szerepel. Ha az 5.1. 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. \square

A	B
1	2
3	4
1	2
1	2

5.1. ábra. Egy multihalmaz

5.1.1. Mire jók a multihalmazok?

A forgalomban lévő ABKR-ek a relációkat multihalmazként valósítják meg (nem pedig halmazként), mint ahogy azt már korábban is említettük. A relációk hatékony megvalósítása szempontjából a relációk multihalmazokként való kezelése több módon is gyorsíthatja a relációs műveleteket. Például:

1. Ha két multihalmazként értelmezett reláció unióját vesszük, akkor az egyik reláció sorait egyszerűen lemásoljuk, és ehhez a másolathoz hozzáadjuk a másik reláció összes sorát. Ebben az esetben nem kell megszüntetnünk azon ismétlődő sorokat sem, amelyek mind a két relációban szerepelnek.
2. Ha a reláció vetítését halmazként fogjuk fel, akkor minden vetített sort össze kell hasonlítanunk az összes többi vetített sorral azért, hogy meggyőződjünk arról, hogy a vetítésben minden sor csak egyszer szerepel. Ezzel szemben, hogyha az eredményt multihalmaznak tekintjük, akkor minden sort levetítünk, és egyszerűen hozzáadjuk ezeket az eredményhez. Így egyetlen összehasonlitást sem kell végeznünk a vetítésből származó sorok között.

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

5.2. ábra. Az 5.2. példában szereplő multihalmaz

5.2. példa. Az 5.1. ábrán látható multihalmaz, akár az 5.2. 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, annak ellenére, hogy a multihalmazként kezelt eredmény nagyobb méretű, mégis gyorsabban ki lehet számolni, hiszen nem kell összehasonlítsuk az összes $(1, 2)$, illetve $(3, 4)$ sort a megelőző lépésekben megkapott sorokkal.

□

Egy másik motiváció a relációk multihalmazként történő kezelésére: előfordulhat olyan helyzet, mikor a kívánt választ csak úgy kaphatjuk meg, ha legalábbis átmenetileg multihalmazos megközelítést használunk. A következő példa ezt mutatja be.

5.3. példa. Ha például az 5.2. ábrán lévő halmazként kezelt reláció A oszlopára vonatkozó átlagot akarjuk venni, akkor nem használhatjuk a halmaz modell A -ra vonatkozó vetítését, mivel ez esetben az átlagra 2-t kapunk. Ugyanis az 5.2. ábrán az A -nak csak két különböző értéke van (3 és 1), ezeknek az átlaga pedig 2 . Ha viszont az A oszlopra mint a $\{1, 3, 1, 1\}$ multihalmazra tekintünk, akkor a valódi átlagot kapjuk, amelynek értéke – az 5.2. ábra 4 sorának felhasználásával számolva – $1,5$ lesz. □

5.1.2. Multihalmazok egyesítése, metszete, különbsége

Mind a három műveletnek új értelmezése lesz multihalmazok esetén. Tegyük fel, hogy R és S multihalmazok, és legyen t az R n -szer, illetve az S m -szer előforduló sora. Megengedjük azt is, hogy akár n , akár m , akár mindenkor 0 legyen. Ekkor:

- Az $R \cup S$ multihalmazon értelmezett egyesítésben a t sor $n + m$ -szer fog előfordulni.
- Az $R \cap S$ multihalmazon értelmezett metszetben a t sor $\min(n, m)$ -szer fog előfordulni.
- Az $R - S$ multihalmazon értelmezett különbségben a t sor $\max(0, n - m)$ -szer fog előfordulni. Azaz ha a t többször fordul elő R -ben, mint S -ben, akkor t az $R - S$ -ben annyiszor fordul elő, mint ahányszor előfordult R -ben minusz ahányszor előfordul S -ben. Amennyiben viszont t legalább annyiszor előfordul S -ben, mint R -ben, akkor t nem lesz benne $R - S$ különbségben. Informálisan tehát t minden egyes előfordulása S -ben „semlegesít” t egy előfordulását R -ben.

5.4. példa. Legyen R az 5.1. ábrán látható reláció, amely valójában egy multihalmaz, amelyben az $(1, 2)$ sor kétszer és a $(3, 4)$ sor egyszer szerepel. Legyen 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 $(1, 2)$ négyeszer (háromszor r miatt és egyszer 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 R -ben és egyszer S -ben és $\min(3, 1) = 1$; hasonló módon a $(3, 4)$ sorra $\min(1, 2) = 1$. Az $(5, 6)$ sor egyszer szerepel S -ben és nem szerepel 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 R -ben és egyszer S -ben, $\max(0, 3 - 1) = 2$, ezért az $R - S$ különbségen az a sor kétszer jelenik meg. A $(3, 4)$ sor egyszer szerepel R -ben és kétszer szerepel S -ben, $\max(0, 1 - 2) = 0$, tehát ez a sor nem szerepel az $R - S$ különbségen. Mivel R -nek nincs más sora, ezért az $R - S$ eredmény sem tartalmazhat más sort.

Az $S - R$ 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. \square

Multihalmaz-műveletek halmazokon

Képzeljük el, hogy van két halmazunk, R és 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 -et multihalmaznak tekintjük, és az ennek megfelelő szabályt alkalmazzuk. Ebben az esetben ugyanazt az eredményt kapjuk, mintha az R -et és S -et halmaznak tekintettük volna. Ha R -et és S -et 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 minden, hogy halmaz- vagy multihalmazszabályal 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 -et halmaznak vagy multihalmaznak tekintjük. Ha multihalmazszabályt alkalmazunk az $R \cup S$ kiszámításához, akkor előfordulhat, hogy az eredmény nem halmaz lesz annak ellenére, hogy minden az R , minden az S halmaz. Azaz, ha a t sor az R -ben és az S -ben is szerepel, akkor multihalmazszabályt alkalmazva a t sor kétszer jelenik meg az $R \cup S$ eredményben, ha viszont halmazszabályt alkalmazunk, akkor a t sor csak egyszer jelenik meg az $R \cup S$ eredményben.

5.1.3. Multihalmazok vetítése

A multihalmazok vetítéséről már volt szó. Amint az 5.2. példában láthattuk, a vetítés során minden sort külön kezeljük. Ha az R reláció az 5.2. ábrán látható multihalmaz, akkor $\pi_{A,B}(R)$ eredménye az 5.1. 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 az 5.2. ábra R relációjának (1, 2, 5), (1, 2, 7) és (1, 2, 8) sorait levetítjük az A és B attribútumokra, akkor minden sor ugyanazt az (1, 2) sort eredményezi. Ez azt jelenti, hogy a multihalmaz-eredményben az (1, 2) sor háromszor szerepel, míg a hagyományos vetítéssel csak egyszer szerepelne.

Algebrai azonosságok multihalmazok esetén

Az algebrai azonosság két relációs algebrai kifejezés közötti ekvivalencia. A kifejezések argumentumai relációk. Az ekvivalencia szerint minden egy, hogy milyen relációkat helyettesítünk be, a két kifejezés ugyanazt az eredményt adja. Egy jól ismert példa az egyesítésre vonatkozó azonosság: $R \cup S = S \cup R$. 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: $(R \cup S) - T = (R - T) \cup (S - T)$. Ez az azonosság érvényes halmazokra, viszont nem érvényes multihalmazokra. Hogy lássuk, miért is nem igaz multihalmazok esetén, tegyük fel, hogy a t sor egyszer szerepel mindenhol relációban, R -ben, S -ben és 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 halmazszabályt alkalmazunk, akkor egyik oldal eredményében sem szerepel a t sor. Az 5.1.4. és 5.1.5. feladatokban megvizsgáljuk további algebrai azonosságok érvényességét multihalmazok esetén.

5.1.4. 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.

5.5. példa. Legyen R a következő multihalmaz:

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

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

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

Azaz, az első sor kivételével minden egyik sor teljesíti a feltételt. A két utolsó sor R -ben is megegyezik, így minden kettő szerepel az eredményben is. \square

5.1.5. Multihalmazok szorzata

A multihalmazok Descartes-szorzatára vonatkozó szabály pontosan az, amit vártunk: az első reláció minden egyes sorát össze kell párosítanunk a második reláció minden 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 az $R \times S$ szorzat eredményében.

5.6. példa. Legyen az R és S reláció az 5.3. ábrán látható két reláció. Ekkor az $R \times S$ szorzat hat sorból áll, amint az az 5.3. (c) ábrán is látható. Megjegyezzük, hogy a hagyományos szorzatnál bevezetett, attribútumnevekre 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. \square

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) Az $R \times S$ szorzat**5.3. ábra.** Multihalmazok szorzatának kiszámítása

5.1.6. Multihalmazok összekapcsolása

A multihalmazok összekapcsolása szintén nem szolgál különösebb meglepetéssel. Az első reláció minden egyes sorát összehasonlítjuk a második relációval amennyi sorával, és eldöntjük, hogy az így kapott sorpárok sikeresen összekapcsolhatók-e vagy sem. Ha az összekapcsolás elvégezhető, akkor az eredményből nem kell kiküszöbölni a megegyező sorokat.

5.7. példa. Az 5.3. 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 minden sor kétszer szerepel a megfelelő relációkban, ezért az összekapcsolt sor $2 \times 2 = 4$ alkalommal jelenik meg az eredményben. Ezenkívül még összekapcsolhatnánk az R (1, 2) sorát az S (2, 3) sorával, de ez a sorpár nem teljesíti a feltételt, íly módon az eredményben sem jelenik meg. \square

5.1.7. Feladatok

5.1.1. feladat. Tekintsük a 2.21 (a) ábrán szereplő PC relációt. Számítsuk ki a $\pi_{\text{sebesség}}(\text{PC})$ 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?

5.1.2. feladat. Végezzük el az 5.1.1. feladatot a $\pi_{\text{merevlemez}}(\text{PC})$ kifejezéssel.

5.1.3. feladat. Tekintsük a 2.4.3. feladatban szereplő „Csatahajó” relációt.

- a) Tekintsük a $\pi_{\text{kaliber}}(\text{Hajóosztályok})$ egyoszlopos relációt adó kifejezést, amely az egyes osztályok kaliberét adja meg. Mi lesz ennek az értéke a 2.4.3. 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 használunk multihalmazokat, azaz annyiszor szerepeljen a b mint kaliber, ahány b kaliberű hajó van.

! 5.1.4. 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: $(R \cup S) \cup T = R \cup (S \cup T)$.
- b) A metszet asszociatív tulajdonsága: $(R \cap S) \cap T = R \cap (S \cap T)$.
- c) A természetes összekapcsolás asszociatív tulajdonsága:

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

- d) Az egyesítés kommutatív tulajdonsága: $R \cup S = S \cup R$.
- e) A metszet kommutatív tulajdonsága: $R \cap S = S \cap R$.
- f) A természetes összekapcsolás kommutatív tulajdonsága:

$$R \bowtie S = S \bowtie R$$

- g) $\pi_L(R \cup S) = \pi_L(R) \cup \pi_L(S)$, ahol L az attribútumok tetszőleges listája
- h) Az egyesítés és a metszet disztributív tulajdonsága:

$$R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$$

- i) $\sigma_C \text{ AND } D(R) = \sigma_C(R) \cap \sigma_D(R)$, ahol C és D tetszőleges feltételek az R soraira.

!! 5.1.5. 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) $(R \cap S) - T = R \cap (S - T)$.
- b) A metszet disztributív tulajdonsága az egyesítés felett:

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$$

- c) $\sigma_C \text{ OR } D(R) = \sigma_C(R) \cup \sigma_D(R)$, ahol C és D tetszőleges feltételek az R soraira.

5.2. Kiterjesztett műveletek a relációs algebrában

A klasszikus relációs algebrát a 2.4. alfejezetben már bemutattuk, illetve az olyan módosításokat is megtettük az 5.1. alfejezetben, amelyek elengedhetetlenek voltak ahhoz, hogy a korábbi halmazos szemlélettel szemben, most a relációkra mint sorok multihalmazára tekinthessünk. Ennek a két résznek az elképzelései képezik a legtöbb korszerű lekérdező nyelv alapjait. Habár az olyan nyelvek, mint az SQL, számos olyan, ezektől eltérő műveletet is tartalmaznak, amelyek alkalmazások írásánál nagyon fontosak lehetnek. Ezért a relációs műveletek egy teljesebb kezelési megközelítésében szerepelni kell még néhány más műveletnek is, amelyeket ezen alfejezet során be is fogunk vezetni. A kiegészítések a következők:

- 1. Az ismétlődések megszüntetésének művelete: δ .* Ez a művelet a multihalmaz halmazzá alakítja a sorok másolatainak megszüntetésével.
- 2. Az összesítő műveletek.* Ilyen például az összegzés, illetve az átlag, amelyek ugyan nem a relációs algebra műveletei, de csoportosító műveletekkel használhatók (ezt később részletezzük). Az összesítő műveletek egy reláció attribútumaira (oszlopaira) vannak értelmezve. Például egy oszlopra vonatkozó összegzés értéke azt a számot reprezentálja, amelyet az oszloban szereplő értékek összeadása által kapunk meg.
- 3. Csoportosítás.* A sorok csoportosítása egy reláció sorainak „csoportokba” történő besorolása a reláció egy vagy több attribútumának értékétől függően. Ezek után már az egyes csoportok oszlopaira összesítési művelet is végezhető, amely lehetővé teszi számunkra néhány olyan lekérdezés megfogalmazását is, amelyeket a klasszikus relációs algebrával nem tudtunk kifejezni. A γ csoportosítási művelet egy olyan művelet, amely a csoportosítás és az összesítés hatását kombinálja.
- 4. Kiterjesztett vetítés művelet.* Ez kiterjeszti a π művelet hatását azzal, hogy a néhány oszlopra történő vetítés mellett azt is lehetővé teszi, hogy az érintett oszlopok valamilyen összesítési relációja alapján új oszlopok kiszámítását is elvégezhessük.
- 5. Rendezési művelet.* A τ egy relációt a sorainak egy vagy több attribútumtól függő rendezett listájává alakít. Megfontoltan kell bánni ezzel az operátorral, hiszen a relációs algebra néhány művelete nincs értelmezve listára. Ezzel szemben a vetítés és a kiválasztás elvégezhető listákra is, sőt az eredményben a lista elemeinek sorrendje is megkövetelhető.
- 6. Külső összekapcsolás.* Az összekapcsolás egyik fajtája, amely a lógó sorokat is megőrzi. A lógó sorok NULL értékekkel „egészülnek ki” a külső összekapcsolás eredményében, tehát a lógó sorok is szerepelnek a végeredményben.

5.2.1. Ismétlődések megszüntetése

Néhány esetben szükségünk lehet olyan műveletre, amely a multihalmazból halmazt állít elő. A $\delta(R)$ kifejezést használjuk arra, hogy olyan halmazt kapunk, amely R relácionak csak a különböző sorait tartalmazza.

5.8. példa. Tekintsük az 5.1. ábra R relációját:

A	B
1	2
3	4
1	2
1	2

Ekkor $\delta(R)$ a következő:

A	B
1	2
3	4

Figyeljük meg, hogy az (1, 2) sor, amely $\delta(R)$ -ben csak egyszer fordul elő, R -ben háromszor is szerepelt. \square

5.2.2. Összesítési műveletek

Több olyan művelet is létezik, amely alkalmazható számok vagy karakterláncok halmazaira vagy multihalmazaira. Ezek a műveletek összegzik vagy összesítik a reláció egy oszlopában szereplő értékeket. Ezen tulajdonságuk miatt ezeket összefoglalóan **összesítési műveleteknek** nevezzük. A legáltalánosabb műveletek ezek közül az alábbiak:

1. SUM, az oszlop értékeinek összegét határozza meg.
2. AVG, az oszlop értékeinek átlagát határozza meg.
3. MIN, illetve MAX az oszlop értékeinek minimumát, illetve maximumát határozza meg. Amennyiben karakterláncokat tartalmazó oszlopra alkalmazzuk, akkor a lexikografikusan (ábécé szerinti) legelső, illetve legutolsó értéket határozzák meg.
4. COUNT, az oszlopan található (nem feltétlenül különböző) elemek számát határozza meg. Más szavakkal a COUNT egy reláció bármely attribútumára alkalmazva megadja a reláció sorainak a számát, beleértve az ismétlődéseket is.

5.9. példa. Tekintsük az alábbi relációt:

A	B
1	2
3	4
1	2
1	2

Tekintsünk néhány példát az adott reláció attribútumain történő összesítésekre:

1. $\text{SUM}(B) = 2 + 4 + 2 + 2 = 10$.
2. $\text{AVG}(A) = (1 + 3 + 1 + 1)/4 = 1,5$.
3. $\text{MIN}(A) = 1$.
4. $\text{MAX}(B) = 4$.
5. $\text{COUNT}(A) = 4$.

□

5.2.3. Csoportosítás

Néha nem egyszerűen egy oszlop összesítésére van szükségünk, hanem a reláció sorait kell csoportosítanunk egy vagy több oszlop értékei szerint, és így csoportonként összesíthetünk. Például szeretnénk meghatározni minden egyik stúdió által előállított filmpercek összegét stúdióinként, azaz egy, az alábbihoz hasonló relációt:

stúdióNév	hossz
Disney	12345
MGM	54321
...	...

Induljunk ki a 2.2.8. alfejezetben szereplő adatbázissémából:

`Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)`

Ezért csoportosítanunk kell a `Filmek` reláció sorait a `stúdióNév` szerint, majd csoportonként ki kell számolni a `hossz` összegét. Azaz tegyük fel, hogy a `Filmek` sorai az 5.4. ábrán szereplő alakban vannak, és csoportonként alkalmazzuk a `SUM(hossz)` összesítést.

	<i>stúdióNév</i>	
	Disney	
	Disney	
	Disney	
	MGM	
	MGM	
	○	
	○	
	○	

5.4. ábra. Egy reláció az elképzelt csoportfelosztásról

5.2.4. A csoportosítási művelet

Most már bevezethetünk egy olyan műveletet, ami lehetővé teszi egy reláció csoportokra osztását, illetve néhány oszlopra vonatkozó összesítést. Ha van csoportosítás, akkor az összesítés az egyes csoportokon belül értendő.

A γ alsó indexeként szereplő L elemeknek egy listája, ahol egy elem az alábbiak közül bármelyik lehet:

- a) Az R reláció egy attribútuma, ahol γ R -re van alkalmazva. Azaz egyike R olyan attribútumainak, amelyre a csoportosítást végeztük. Ezt az elemet *csoportosítási attribútumnak* nevezzük.
- b) A reláció valamelyik attribútumára vonatkozó összesítési művelet. Ha az összesítés eredményére névvel szeretnénk hivatkozni, akkor egy nyilat és egy új nevet kell utána írnunk. A kiindulásul szolgáló attribútumot *összesítési attribútumnak* nevezzük.

Az $\gamma_L(R)$ kifejezés eredményrelációjának felépítése a következő:

1. Osszuk R sorait *csoportokba*. Egy csoport azokat a sorokat tartalmazza, amelyeknek az L listán szereplő csoportosítási attribútumokhoz tartozó értékei megegyeznek. Ha nincs csoportosítási attribútum, akkor az egész R reláció egy csoportot képez.
2. minden csoporthoz hozunk létre olyan sort, amelyik tartalmazza:
 - i) A szóban forgó csoport csoportosítási attribútumait.
 - ii) Az L lista összesítési attribútumaira vonatkozó összesítéseket.
(Az adott csoport összes sorára.)

5.10. példa. Tegyük fel, hogy adott az alábbi reláció:

SzerepelBenne(filmCím, filmÉv, színészNév)

A δ speciális esete γ -nak

Technikailag a δ művelet redundáns. Ha $R(A_1, A_2, \dots, A_n)$ egy reláció, akkor $\delta(R)$ megegyezik a $\gamma_{A_1, A_2, \dots, A_n}(R)$ kifejezéssel. Azaz az ismétlések megszüntetéséhez csoportosítást végezünk az attribútumokra, de nem összegezzük azokat. Ekkor minden csoportnak egy sor felel meg, amelyek R -ben egynél többször is előfordulhattak. Mivel a γ eredménye valójában csak egy sort tartalmaz a csoportokra, ezért a „csoportosításunk” eredményeként az ismétlődések is megszűnnek. A δ viszont egy elterjedt és fontos művelet, ezért célszerű továbbra is külön vizsgálnunk a műveletek algebrai szabályokkal és algoritmusokkal történő megvalósítása során.

Azt is láthatjuk, hogy γ a halmazon alkalmazott vetítésművelet kiterjesztése. Azaz $\gamma_{A_1, A_2, \dots, A_n}(R)$ eredménye ugyanaz, mint a $\pi_{A_1, A_2, \dots, A_n}(R)$ kifejezésé, ha R halmaz. Ha viszont R multihalmaz, akkor a γ megszünteti az ismétlődéseket, míg a π nem.

Meg szeretnénk találni minden olyan színészt, aki már szerepelt legalább három filmben, illetve a hozzá tartozó első film dátumát is. Az első lépésben csoportosítanunk kell a **színészNév** mezővel, mint csoportosítási attribútummal. Mindegyik csoportra ki kell számolnunk a **MIN(filmÉv)** összesítést. Emellett viszont ki kell számítanunk a **COUNT(filmCím)** összesítést is minden csoportra ahhoz, hogy eldönthessük, mely csoport elégíti ki a legalább három filmben való szereplés feltételeit.

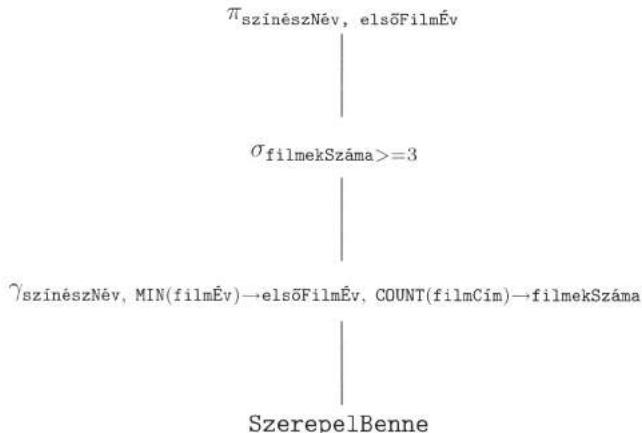
Elsőként írjuk fel a csoportosító kifejezést:

$$\gamma_{\text{színészNév}, \text{MIN(filmÉv)} \rightarrow \text{elsőFilmÉv}, \text{COUNT(filmCím)} \rightarrow \text{filmekSzáma}}(\text{SzerepelBenne})$$

A kifejezés első két oszlopa kelleni fog a végeredményben is. A harmadik oszlop egy segédattribútum (amelyet **filmekSzámá**-nak nevezünk) annak meghatározására, hogy az adott színész szerepelt-e már legalább három filmben. Azaz a lekérdezésnek megfelelő algebrai kifejezést egy **filmekSzáma** ≥ 3 feltéttel történő kiválasztással és az első két oszlopra történő vetítéssel kell kiegészítünk. A lekérdezés kifejezésfáját az 5.5. ábrán tekinthetjük meg. □

5.2.5. A vetítés művelet kiterjesztése

Tekintsük át újra a 2.4.5. alfejezetben bevezetett $\pi_L(R)$ vetítési műveletet. A klasszikus relációs algebrában L az R reláció néhány attribútumának listájaként értelmezhető. Most pedig terjesszük ki a vetítés műveletét úgy, hogy lehetővé tegye számítások elvégzését is a sorok választott komponenseivel. A *kiterjesztett vetítés* – amelynek jelölése szintén $\pi_L(R)$ – vetítési listájában a következő típusú elemek szerepelhetnek:



5.5. ábra. Az 5.10. példa lekérdezésének algebrai kifejezésfája

1. Az R reláció egy attribútuma.
2. Egy $x \rightarrow y$ kifejezés, ahol x és y attribútumneveket jelölnek. Az L lista $x \rightarrow y$ eleme lekérdezi az R x attribútumát és y -ra nevezi át az oszlop nevét, azaz az eredmény sémájában ezen attribútum neve y lesz.
3. Egy $E \rightarrow z$ kifejezés, ahol E az R reláció attribútumaira vonatkozó (konsztansokat, aritmetikai műveleteket, illetve karakterlánc-műveleteket tartalmazó) kifejezés, z pedig az E kifejezés alapján számolt, az eredményekhez tartozó új attribútumnak a nevét jelöli. Példaként tekintsük az $a + b \rightarrow x$ kifejezést a lista elemének, ekkor ez az a és b attribútumok összegét jelöli, amelynek a neve x lesz. A $c \mid d \rightarrow e$ elem jelentése pedig az, hogy c és d karakterláncként értelmezett attribútumokat összefűzzük, majd az eredményül kapott oszlopot e -nek nevezzük el.

A vetítés kiszámítása R összes sorának figyelembenve történik. Az L lista kiértékelése a sor azon komponencinek behelyettesítésével történik, amelyek szerepeltek L attribútumai között. A behelyettesítéskor az L -ben szereplő műveleteket is elvégezzük. Az eredmény egy olyan reláció lesz, amelynek a sémáját az L listában szereplő nevek (illetve átnevezések) alkotják. R minden egyes sora egy sort eredményez a végeredményben. Ezért az R relációban többször előforduló sorok az eredményben is többször fordulnak elő. A végeredményben viszont akkor is lehetnek ismétlődések, ha R -ben eredetileg nem voltak.

5.11. példa. Legyen az R reláció:

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

Ekkor $\pi_{A,B+C \rightarrow X}(R)$ értéke a következő:

A	X
0	3
0	3
3	9

Az eredmény sémájának két attribútuma van. Az első az A , ami R attribútuma volt ugyanezzel a névvel. A második az X , ami R második és harmadik attribútumának összegét reprezentálja.

Másik példaként tekintetjük $\pi_{B-A \rightarrow X,C-B \rightarrow Y}(R)$ értékét, azaz:

X	Y
1	1
1	1
1	1

Figyeljük meg, hogy ennek a vetítéslistának a kiszámításánál a $(0, 1, 2)$ és a $(3, 4, 5)$ sorokat ugyanazon $(1, 1)$ sorba képezzük le. Ezért ez a sor jelenik meg háromszor az eredményben. \square

5.2.6. Rendezési művelet

Jó néhány esetben elközelhető, hogy egy reláció sorait egy vagy több attribútuma alapján szeretnénk rendezni. Lekérdezéseink során gyakran megköveteljük az eredményreláció rendezettségét. Tegyük fel például, hogy Sean Connery filmjeit szeretnénk lekérdezni, és elvárjuk, hogy a kapott lista filmcím szerint rendezett legyen azért, hogy egy konkrét filmet gyorsabban megtalálhassunk. A lekérdezések optimalizálásánál vizsgálatánál majd láthatjuk, hogyan válik gyakran hatékonyabbá az ABKR-ben egy lekérdezés végrehajtása, ha a relációt először rendezzük.

A $\tau_L(R)$ kifejezés, ahol R egy relációt, L pedig az R reláció attribútumait tartalmazó listát jelenti, magát az R relációt határozza meg, csak már az L lista alapján meghatározott rendezett alakban. Ha az L lista tartalma A_1, A_2, \dots, A_n , akkor R sorait először A_1 értékei szerint fogja rendezni. A maradt csomókat A_2 értéke szerint tovább bontja, majd az olyan soroknál, amelyek mind A_1 , minden A_2 szerint is azonos rendezettséggűek, az A_3 értéke szerint rendezzi és így tovább. Azokat a csomókat, amelyek A_n értéke szerinti rendezés után is megmaradtak, tetszőleges sorrendben helyezzük el.

5.12. példa. Ha tekintjük az $R(A, B, C)$ sémájú R relációt, akkor $\tau_{C, B}(R)$ az R sorait C érték szerint rendezi, majd az ugyanazon C értékkel rendelkező sorokat B szerint továbbrendezzi. Ha minden a C , minden a B érték megegyezik, akkor tetszőleges lesz a sorrend. \square

Amennyiben a τ által már rendezett eredményre egy olyan másik műveletet is elvégzünk, mint az összekapcsolás, akkor a korábbi rendezettség az esetek nagy többségében jelentését veszti, és ezután az eredményre is inkább multihalmazként érdemes tekinteni, nem pedig listaként. Ezzel szemben a multihalmazokon értelmezett vétítések megőrzik a rendezettséget. Sőt a listára alkalmasztott kiválasztás eredményében, amely a kiválasztás feltételének nem megfelelő sorokat eldobja, a megmaradó sorok még mindig az eredeti rendezés szerinti sorrendjükben szerepelhetnek.

5.2.7. Külső összekapcsolások

Az összekapcsolás műveletének egyik tulajdonsága, hogy lehetnek lógó sorok, amelyek nem kapcsolhatóak össze más sorokkal, azaz nem lesz egyezés ezen sorok és a másik reláció sorai között a közös attribútumokon. A lógó sorokhoz nem tartozik sor az összekapcsolás eredményében, így az összekapcsolás nem biztos, hogy az eredeti reláció adatait hiánytalansul tükrözi. Az ilyen esetekben az összekapcsolás egy változatát, nevezetesen a „külső összekapcsolást” ajánlott alternatívaként használni. A külső összekapcsolás megtalálható a különféle, forgalomban lévő rendszerekben.

Először is tekintsük a „természetes” esetet, amelyben az összekapcsolás a benne szereplő két reláció közös attribútumaiban lévő értékeknek az egyenlőségén alapult. A $R \bowtie S$ alakú *külső összekapcsolás* először elvégzi az $R \bowtie S$ természetes összekapcsolást, majd ehhez hozzáadja az R és az S lógó sorait is. Az előbbi módon hozzáadott sort minden olyan attribútumában ki kell egészíteni egy speciális *null* szimbólummal (\perp), amellyel a sor ugyan nem rendelkezett, de az összekapcsolás eredményében már szerepel. Megjegyezzük, hogy \perp szimbólum a *NULL* (a 2.3.4. alfejezetben szereplő) értéknek felel meg SQL-ben.

5.13. példa. Vegyük az 5.6 (a) ábra, illetve az 5.6 (b) ábra U és V relációját. Az U (1, 2, 3) sora így V -nek mind a (2, 3, 10), mind a (2, 3, 11) sorával összekapcsolható. Így ez a három sor nem lógó. Viszont a fennmaradó V -beli és U -beli sorok lógók: a (4, 5, 6), illetve a (7, 8, 9) az U -ból és a (6, 7, 12) a V -ból. Azaz, ezen három sor egyikének sincs olyan sorpárja a másik relációban, amellyel a B és C komponenseken is megegyezne. Ezért az 5.6 (c) ábrán szereplő $U \bowtie V$ eredményében a lógó sorok ki vannak egészítve egy \perp szimbólummal ott, ahol nem rendelkeznek értékkal: a D attribútumnál az U sorainál és az A attribútumnál a V sorai esetén. \square

Az alap (természetes) külső összekapcsolás elvének létezik több variánsa is. A $R \bowtie_L S$ bal oldali külső összekapcsolás annyiban különbözik a külső összekapcsolástól, hogy csak a bal oldalon szereplő R argumentum lógó sorainak \perp szimbólummal kiegészített változatát adjuk hozzá az eredményhez. A $R \bowtie_R S$ jobb oldali külső összekapcsolás annyiban különbözik a külső összekapcsolástól, hogy csak a jobb oldalon szereplő S argumentum lógó sorainak \perp szimbólummal kiegészített változatát adjuk hozzá az eredményhez.

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

(a) Az U reláció

B	C	D
2	3	10
2	3	11
6	7	12

(b) A V reláció

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	\perp
7	8	9	\perp
\perp	6	7	12

(c) Az $U \bowtie V$ eredményreláció**5.6. ábra.** Relációk különböző összekapcsolása**5.14. példa.** Ha vesszük az 5.6. ábra U és V relációját, akkor $U \bowtie_L V$:

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

Az $U \bowtie_R V$ pedig:

A	B	C	D
1	2	3	10
1	2	3	11
\perp	6	7	12

□

A három természetes összekapcsolási műveleten túl a théta-összekapcsolás még hátramaradt, amely során először egy théta-összekapcsolást végzünk, majd az eredményéhez hozzáadjuk azokat a \perp szimbólummal kiegészített sorokat is, amelyeket a théta-összekapcsolás feltételének ellenőrzése során nem tudunk a másik reláció egyetlen sorával sem társítani. A \bowtie_C kifejezést használjuk a C feltéttel rendelkező théta-összekapcsolás jelölésére. Ez a művelet is módosítható L vagy R segítségével bal vagy jobb oldali külső összekapcsolássá.

5.15. példa. Legyenek U és V az 5.6. ábra relációi, és tekintsük a következőt:

$$U \bowtie_{A>V,C} V$$

Az U $(4, 5, 6)$ és $(7, 8, 9)$ sorai, illetve a V $(2, 3, 10)$ és $(2, 3, 11)$ sorai is kielégítik a feltételt. Ezért ezen négy sor egyike sem lesz nem társítható. Ezzel szemben a maradék két sor lógó lesz: az U $(1, 2, 3)$ sora és a V $(6, 7, 12)$ sora. Ezért kiegészítve fognak szereplni az 5.7. ábrán szereplő eredményben. \square

A	$U.B$	$U.C$	$V.B$	$V.C$	D
4	5	6	2	3	10
4	5	6	2	3	11
7	8	9	2	3	10
7	8	9	2	3	11
1	2	3	\perp	\perp	\perp
\perp	\perp	\perp	6	7	12

5.7. ábra. A théta-összekapcsolás eredménye

5.2.8. Feladatok

5.2.1. feladat. Tekintsük az alábbi két relációt:

$$R(A, B): \{(0, 1), (2, 3), (0, 1), (2, 4), (3, 4)\}$$

$$S(B, C): \{(0, 1), (2, 4), (2, 5), (3, 4), (0, 2), (3, 4)\}$$

Számítsuk ki a következőket: a) $\pi_{A+B, A^2, B^2}(R)$; b) $\pi_{B+1, C-1}(S)$; c) $\tau_{B, A}(R)$; d) $\tau_{B, C}(S)$; e) $\delta(R)$; f) $\delta(S)$; g) $\gamma_{A, \text{sum}(B)}(R)$; h) $\gamma_{B, \text{AVG}(C)}(S)$; i) $\gamma_A(R)$; j) $\gamma_{A, \text{MAX}(C)}(R \bowtie S)$; k) $R \bowtie_L S$; l) $R \bowtie_R S$; m) $R \bowtie_S S$; n) $R \bowtie_{R.B < S.B} S$.

! 5.2.2. feladat. Egy f egyértékű műveletet *idempotensnek* nevezünk, ha bár-mely R relációra teljesül, hogy $f(f(R)) = f(R)$, azaz f többszöri alkalmazása ugyanazt az eredményt adja, mintha egyszer alkalmaztuk volna. A következő operátorok közül melyik lesz idempotens? Válaszához adjon számolási példát vagy indokolja meg.

- a) δ ; b) π_L ; c) σ_C ; d) γ_L ; e) τ .

! 5.2.3. feladat. Az egyik doleg, amit az eredeti 2.4.5. alfejezetben definiált vértései művelettel szemben elérhetünk a kiterjesztett vetítési művelet segítségével, hogy megduplázzunk oszlopokat. Például, ha $R(A, B)$ egy reláció, akkor $\pi_{A,A}(R)$ létrehoz egy (a, a) sort minden R -beli (a, b) sorra. Elvégezhető-e ez a művelet a klasszikus relációs algebra 2.4. alfejezetben leírt műveleteinek a segítségével? Indokolja válaszát.

5.3. Logika a relációkhoz

Az algebrán alapuló absztrakt lekérdező nyelvek alternatívjaként logikai forma is használható a lekérdezések leírására. A logikai lekérdezések nyelve a *Datalog* („database logic”), amely *if-then* szabályokat tartalmaz. minden szabály azt az elvet fejezi ki, hogy az egyes relációkban a sorok bizonyos kombinációjából következtethetünk arra, hogy valamilyen másik sornak szerepelnie kell valamilyen másik relációban vagy egy lekérdezés eredményében.

5.3.1. Predikátumok és atomok

Datalogban a relációkat *predikátumokkal* reprezentáljuk. minden predikátum rögzített számú argumentummal rendelkezik és egy predikátumot, amelyet az argumentumai követnek, *atomnak* nevezünk. Az atomok szintaxisa ugyanolyan, mint egy függvényhívás a hagyományos programozási nyelvekben; például $P(x_1, x_2, \dots, x_n)$ egy atom, amely tartalmazza a P predikátumot az x_1, x_2, \dots, x_n argumentumokkal.

Egy predikátum tulajdonképpen egy függvénynek a neve, ami logikai értékkel tér vissza. Ha R egy reláció valamelyen rögzített sorrendben lévő n argumentummal, akkor R -et használhatjuk, mint erre a relációra hivatkozó predikátumnevet. Az $R(a_1, a_2, \dots, a_n)$ atom értéke IGAZ (TRUE), ha (a_1, a_2, \dots, a_n) az R egy sora; egyébként az értéke HAMIS (FALSE).

5.16. példa. Legyen R egy reláció:

A	B
1	2
3	4

Ekkor $R(1, 2)$ és $R(3, 4)$ is igaz. Viszont x és y bármilyen más értékére $R(x, y)$ hamis. \square

Egy predikátumnak lehetnek változói, valamint konstansai argumentumként. Ha egy atomnak egy vagy több argumentuma változik, akkor az egy logikai visszatérési értékű függvény, amely ezen változókra vett értékekre IGAZ vagy HAMIS értékkel tér vissza.

5.17. példa. Ha R az 5.16. példában használt predikátum, akkor $R(x, y)$ egy függvény, amely azt fejezi ki bármely x -re és y -ra, hogy (x, y) az R reláció

egy sora-e vagy sem. Az R jelen előfordulására a 5.16. példában $R(x, y)$ IGAZ értékkel tér vissza, ha

1. $x = 1$ és $y = 2$, illetve
2. $x = 3$ és $y = 4$;

és HAMIS-sal egyébként. Egy másik példa az $R(1, z)$ atom, amely IGAZ-zal tér vissza, ha $z = 2$, és HAMIS-sal egyébként. \square

5.3.2. 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 az 5.3.1. alfejezetben bevezetett atomokat *relációs atomoknak* nevezzük (mindkettőt „atomoknak” nevezzük).

Figyeljük meg, hogy mind az aritmetikai és mind a relációs atomok argumentumaikban kaphatnak értéket, ezek az atomban előforduló változók értékei, és az atomok logikai visszatérési értékkel rendelkeznek. Valójában az aritmetikai összehasonlítások, mint a $<$ vagy a \geq , tulajdonképpen olyan relációk, amelyek tartalmazzák az összes igaz párt. Azaz a „ $<$ ” relációt úgy képzelhetjük el, mint amely tartalmaz minden olyan sort – mint például az $(1, 2)$ vagy a $(-1, 5, 65, 4)$ –, amelynek az első komponense kisebb a másodiknál. Ne feledjük, hogy az adatbázis-relációk minden végesek, és általában időről időre változnak. Ezzel ellentétben az olyan aritmetikai összehasonlításból származó relációk, mint például a $<$, végtelenek és tartalmuk is változatlan.

5.3.3. Datalog-szabályok és lekérdezések

A Datalogban az 5.2. alfejezetben bemutatott klasszikus relációs algebrához hasonló műveletek leírása *szabályokkal* történik, amelyek a következőkből épülnek fel:

1. Egy *fej*nek nevezett relációs atom;
2. Ezt követi a \leftarrow szimbólum, amelyet gyakran „if”-nek („ha”) olvasunk ki;
3. Ez után következik 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.

5.18. példa. A következő Datalog-szabály:

```
HosszúFilm(fc, é) ← Filmek(fc, é, h, m, s, p) AND h ≥ 100
```

a „hosszú” filmek halmazát határozza meg (a legalább 100 percnél hosszabbakat). A fenti szabályban a standard *Filmek* relációt hivatkozunk, amelynek sémája a következő:

$$\text{Filmek}(\text{filmcím}, \text{év}, \text{hossz}, \text{műfaj}, \text{stúdióNév}, \text{producerAzon})$$

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

1. Az első részcél a *Filmek* nevű predikátumból, mégpedig a *Filmek* reláció hat attribútumának megfelelő hat argumentumból áll. Az argumentumok különbözők: *fc* a *filmcím* komponens, *é* az *év* komponens, *h* a *hossz* komponens stb. „Ezt a részcélt tekinthetjük úgy is, mint a *Filmek* reláció aktuális előfordulásának (*fc, é, h, sz, s, p*) sorát.” Pontosabban fogalmazva, a *Filmek(fc, é, h, sz, s, p)* akkor igaz, ha a hat változó értéke megegyezik a *Filmek* reláció egy sorának hat komponensével.
2. A második részcél, $h \geq 100$, akkor igaz, ha a *Filmek* reláció megfelelő sorában a *hossz* komponens értéke legalább 100.

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

- a) A *filmcím* és *év* attribútumoknak megfelelő két első komponens értéke *fc* és *é*.
- b) A *hossz* attribútumnak megfelelő harmadik komponens – azaz 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:

$$\text{HosszúFilm} := \pi_{\text{filmcím}, \text{év}}(\sigma_{\text{hossz} \geq 100}(\text{Filmek}))$$

a fenti megfeleltetés jobb oldala egy relációs algebrai kifejezés. \square

Datalogban egy *lekérdezés* egy vagy több szabály együttese. Ha a szabály fej részében csak egy reláció van, akkor ezen reláció kiértékelése lesz a lekérdezés eredménye. Ekképpen az 5.18. példában megfogalmazott lekérdezés eredménye a *HosszúFilm* reláció. Ha egynél több relációt tartalmaznak a szabályfejek, akkor ezen relációk egyike lesz a lekérdezés eredménye, míg a maradék az eredmény megfogalmazásában segít. Meg kell jelölnünk, hogy melyik reláció adja meg a lekérdezésre a választ. Ezt megtehetjük például úgy, hogy ennek a relációnak speciális nevet adunk, például azt, hogy *Válasz*.

Név nélküli (anonymus) változók

A Datalog-szabályokban gyakran találhatók olyan változók, amelyek csak egyszer szerepelnek az adott szabályban. Éppen ezért teljesen mindegy, hogy ezeknek a változóknak mi a nevük. A változó neve akkor fontos, ha a változó többször is megjelenik a szabályban, hiszen ekkor a változó második, illetve további előfordulásakor is tudnunk kell, hogy ugyanarról a változóról van-e szó. Ezért megegyezés szerint azokat a változókat, amelyek csak egyszer jelennek meg, jelölhetjük egy aláhúzás jelleggel. Tehát egy atomban használhatunk $_$ jelet argumentumként. Az $_$ többszöri előfordulása különböző változóra utal, soha nem utalhat ugyanarra a változóra. Például, az 5.18. példában használt szabály felirható a következő módon:

```
HosszúFilm(fc,é) ← Filmek(fc,é,h,_,_,_) AND h ≥ 100
```

Az sz , s , p változók csak egyszer szerepelnek a szabályban, ezért ezek helyettesíthetők az aláhúzás jelleggel. A többi változót nem helyettesíthetjük, hiszen mindegyik kétszer szerepel a szabályban.

5.3.4. A Datalog-szabályok jelentése

Az 5.18. példa útmutatásul szolgált a Datalog-szabályok jelentésére vonatkozóan. Ha pontosabb képet akarunk kapni, képzeljük el, hogy a szabályban szereplő változók felveszik az összes lehetséges értékét. Amikor a változók értékei igazzá teszik az összes részcélét, akkor megkapjuk, hogy mi a fej értéke az aktuális változókra, és hozzáadjuk a kapott sort a fejben szereplő predikáturnak megfelelő relációhoz.

Képzeljük el például, hogy az 5.18. példának mind a hat változója felveszi az összes lehetséges értékét. Az összes részcél csak akkor lehet igaz, ha a $(fc, é, h, sz, s, p)$ értékek a megadott sorrendben a *Filmek* reláció egy sorát képezik. Sőt, mivel a $h \geq 100$ részcélnak is teljesülni kell, ezért a keresett sorban a *hossz* komponensnek megfelelő h változó értéke legalább 100 kell legyen. Ha ilyen értékkombinációt találunk, akkor a $(fc, é)$ sort betesszük a fejnek megfelelő *HosszúFilm* relációba.

Ha biztosítani szeretnénk, hogy egy szabály kiértékeléskor kapott reláció véges legyen, illetve ha értelmezni szeretnénk az aritmetikai és a negált részcélokat is, akkor bizonyos megszorításokat kell tennünk a változókra nézve. (Negált részcélnak nevezzük azt a részcélt, amely előtt NOT szerepel.) Az ilyen megszorítást *biztonságossági feltételnek* nevezzük, és a következőképpen fogalmazhatjuk meg:

- A szabályban szereplő valamennyi változónak szerepelnie kell valamely nem negált, relációs részcélban is.

Ez azt jelenti, hogy a fejben, a negált relációs részcélokban és az aritmetikai részcélokban szereplő valamennyi változónak szerepelnie kell valamely nem negált relációs részcélban is.

5.19. példa. Tekintsük az 5.18. példában megfogalmazott szabályt:

$$\text{HosszúFilm}(\text{fc}, \text{é}) \leftarrow \text{Filmek}(\text{fc}, \text{é}, \text{h}, _, _, _) \text{ AND } \text{h} \geq 100$$

Az első részcél egy nem negált relációs részcél, és ráadásul tartalmazza a szabályban szereplő összes változót. A fejben szereplő két változó (fc , é) szerepel az első részcél törzsében is. Ehhez hasonlóan, az aritmetikai részcélban szereplő h változó pedig megjelenik az első részcélban is. \square

5.20. példa. Az alábbi szabály három dologban is megséríti a biztonságossági feltételeket:

$$\text{P}(\text{x}, \text{y}) \leftarrow \text{Q}(\text{x}, \text{z}) \text{ AND NOT R}(\text{w}, \text{x}, \text{z}) \text{ AND } \text{x} < \text{y}$$

1. Az y változó szerepel a fejben, de nem jelenik meg egyetlen nem negált relációs részcélban sem. Figyeljük meg, hogy az y ugyan megjelenik az $x < y$ aritmetikai részcélban, de ettől a lehetséges y értékek halmaza nem lesz véges. Ha találunk a w , x és z változókhöz olyan a , b és c értékeket, amelyekre az első két részcél teljesül, a fejhez rendelt P relációhoz még minden végtelen sok olyan (b, d) sor lesz hozzáadható, amelyre igaz, hogy $d > b$.
2. A w változó megjelenik egy negált relációs részcélban, viszont nem jelenik meg egyetlen nem negált relációs részcélban sem.
3. Az y változó megjelenik egy aritmetikai részcélban, viszont nem jelenik meg egyetlen nem negált relációs részcélban sem.

Látható tehát, hogy ez a szabály nem biztonságos, éppen ezért Datalogban nem használható. \square

A szabályok jelentését más módon is meghatározhatjuk. Ahelyett hogy minden a változók összes lehetséges értékét, a nem negált relációs részcélok relációinak soraiból álló halmazt vegyük csak. A sorok olyan megfeleltetését, amely valamennyi nem negált részcélra nézve ugyanazt az értéket felelteti meg az ugyanolyan nevű változóknak, *konzisztens* megfeleltetésnek nevezzük. Figyeljük meg, hogy a biztonságosság miatt konzisztens megfeleltetéskor minden egyes változóhoz egy érték tartozik.

Minden egyes konzisztens megfeleltetéskor kiértékeljük a negált relációs részcélokat és az aritmetikai részcélokat, hogy megbizonyosodhassunk arról, hogy a változókhoz hozzárendelt értékek kielégítik-e ezeket a részcélokat. Ne feledjük, hogy egy negált részcél akkor igaz, ha a részcél atomja hamis. Ha mindegyik részcél igaz, akkor vesszük a fejben szereplő változók értékeiből álló sort, és ezt a sort hozzáadjuk a fej predikátumához rendelt relációhoz.

5.21. példa. Tekintsük a következő Datalog-szabályt:

$$P(x, y) \leftarrow Q(x, z) \text{ AND } R(z, y) \text{ AND NOT } Q(x, y)$$

Legyen a Q relácionak két sora, az $(1, 2)$ és az $(1, 3)$. Az R reláció sorai legyenek $(2, 3)$ és $(3, 1)$. Két nem negált relációs részcélunk van, a $Q(x, z)$ és az $R(z, y)$. Ezekhez a részcélokhoz vennünk kell a Q és R relációk sorainak összes lehetséges megfeleltetési kombináját. Az 5.8. ábrán látható táblázat tartalmazza minden négy lehetséges kombinációt.

	A $Q(x, z)$ sorai	Az $R(z, y)$ sorai	Konzisztens a megfeleltetés?	Igaz a NOT $Q(x, y)$ részcel?	A fej eredménye
1.	$(1, 2)$	$(2, 3)$	Igen	Nem	—
2.	$(1, 2)$	$(3, 1)$	Nem; $z = 2, 3$	Irreleváns	—
3.	$(1, 3)$	$(2, 3)$	Nem; $z = 3, 2$	Irreleváns	—
4.	$(1, 3)$	$(3, 1)$	Igen	Igen	$P(1, 1)$

5.8. ábra. A sorok összes lehetséges megfeleltetése a $Q(x, z)$ és $R(z, y)$ részcélokhoz

Az 5.8. ábrán látható második és harmadik megfeleltetés nem konzisztens, mert minden két különböző értéket feleltet meg a z változónak. Ezért ezzel a két megfeleltetéssel nem kell tovább foglalkoznunk.

Az első megfeleltetésben a $Q(x, z)$ részcélhoz az $(1, 2)$ sort feleltettük meg, az $R(z, y)$ részcélhoz pedig a $(2, 3)$ sort. Ez a megfeleltetés konzisztens, hiszen akármelyik változót tekintjük, a megfeleltetés során egyetlen értéket kapott. Az x változó az 1, az y változó a 3 és a z változó a 2 értéket kapta. Ekképpen hozzákezdhetünk a nem negált, relációs részcéloktól különböző részcélok vizsgálatához. Csak egyetlen ilyen részcélunk van, a NOT $Q(x, y)$. A változók értékét behelyettesítve a NOT $Q(1, 3)$ kifejezést kapjuk, ami hamis, hiszen az $(1, 3)$ sor benne van a Q relációban. Tehát ez a részcél hamis, és így nem keletkezett olyan sor, amelyre a fej igaz lenne (1.).

Az utolsó megfeleltetés (4.) szintén konzisztens, az x, y, z változók rendre az 1, 1, 3 értékeket kapták. A NOT $Q(x, y)$ részcél ezúttal NOT $Q(1, 1)$ kifejezéssé alakul. Mivel az $(1, 1)$ nem sora a Q relácionak, tehát ez a részcél ezúttal igaz. Ezután a fejben szereplő változókba behelyettesítjük a megfelelő értékeket, és a $P(1, 1)$ kifejezést kapjuk. Az $(1, 1)$ sort hozzáadjuk a P predikátumhoz rendelt relációhoz. Ily módon a sorok összes lehetséges megfeleltetését megvizsgáltuk, és a P relációba csak ez az egy sor került be. \square

5.3.5. Extenzionális és intenzionális predikátumok

Ajánlatos különbséget tennünk az alábbi kétfajta predikátum között:

- *Extenzionális* predikátumok, amelyekhez rendelt relációk megtalálhatók az adatbázisban;

- *Intenzionális* predikátumok, amelyekhez rendelt relációkat egy vagy több Datalog-szabály kiértékelése alapján kapjuk meg.

Ez a különbség ugyanolyan, mint a relációs algebrai kifejezések operandusai és a relációs algebrai kifejezéssel számított relációk közötti eltérés. A relációs algebrai kifejezések operandusai „extenzionálisak”, mivel a „hozzájuk tartozó” relációk aktuális előfordulásával (*extenzióival*) vannak definiálva. A relációs algebrai kifejezéssel számított relációk pedig – legyenek akár néhány részki-fejezéshez tartozó végeles vagy köztes eredmények – „intenzionálisak”, hiszen tulajdonképpen a program frója hozta létre őket.

Datalog-szabályok esetén az extenzionális, illetve intenzionális predikátumokhoz rendelt relációkat extenzionális, illetve intenzionális relációknak nevezzük. Az intenzionális predikátumokra, illetve relációkra történő hivatkozás esetén használatos az *IDB* rövidítés (a rövidítés az „intensional database” angol kifejezésből ered). Az *EDB* rövidítést az extenzionális predikátumok, illetve relációk esetén használjuk (a rövidítés az „extensional database” angol kifejezésből ered).

Így az 5.18. példában a *Filmek* reláció egy EDB-reláció és a *Filmek* predikátum szintén EDB-predikátum. A *HosszúFilm* reláció és a *HosszúFilm* predikátum pedig egyaránt intenzionális.

Egy EDB-predikátum soha nem jelenhet meg egy szabály fejében sem, csak a törzsében. Az IDB-predikátumok szerepelhetnek a szabály fejében, a törzsében vagy mindenkorban egyaránt. Az is megszokott, hogy több szabály fejében is szerepeljen ugyanaz a predikátum. Az 5.24. példában két reláció egyesítését számoljuk ki ily módon.

Több intenzionális predikátum használatával az EDB-relációkból egyre bonyolultabb kifejezéseket építhetünk fel. Ez a folyamat hasonló a relációs algebrai kifejezések operátorokkal történő felépítéséhez.

5.3.6. Multihalmazokra vonatkozó Datalog-szabályok

A Datalog magában foglal egy halmazlogikát. Viszont amíg nincs negált relációs részcél, addig a Datalog-szabályok kiértékelésének elve elfogadható multihalmazokra nézve is, ha a relációk halmazok. Fogalmi szinten könnyebben használható a Datalog-szabályok kiértékelésének második megközelítése (amelyet az 5.3.4. alfejezetnél mutattunk be), amikor a relációk multihalmazok. Emlékeztetőként, ezen technika magában fogalta az összes nem negált relációs részcél vizsgálatát, illetve a reláció összes sorának a megfelelő részcél predikátumába történő behelyettesítését. Ha a részcélok ból kiválasztott sorok minden változóra konzisztens értéket adnak és az aritmetikai részcélok is teljesülnek¹, akkor megnézhetjük, hogy ezen helyettesítési értékekre a fej mit ad. Az eredménysorokat hozzáadjuk a fejhez tartozó relációhoz.

¹ Megjegyezve, hogy a szabályban nem szerepelhet negált, relációs részcél. Ugyanis a multihalmazos modellben nincs világosan definiálva a negált, relációs részcél tartalmazó Datalog-szabályok jelentése.

Mivel most multihalmazokkal foglalkozunk, most nem kell a fejben szereplő duplikátumokat megszüntetnünk. Sőt, mivel az összes részcélra a sorok minden lehetséges kombinációját figyelembe vesszük, ezért egy sor n -szeres előfordulását egy részcélhoz tartozó relációban n -szer tekintettük a részcél sorának a sorok összes kombinációjával együtt a többi részcélra nézve.

5.22. példa. Tekintsük az alábbi szabályt:

$$H(x, z) \leftarrow R(x, y) \text{ AND } S(y, z)$$

ahol $R(A, B)$ reláció sorai:

A	B
1	2
1	2

és $S(B, C)$ sorai:

B	C
2	3
4	5
4	5

Az egyetlen eset, amikor konzisztens sor helyettesítését kapjuk a részcélokra (egy helyettesítés, amelyben y értéke minden részcélra ugyanaz), amikor az első részcél helyettesítése az R -ból származó $(1, 2)$ sor, a második részcél helyettesítése pedig az S -ból származó $(2, 3)$ sor. Mivel $(1, 2)$ kétszer fordul elő R -ben, $(2, 3)$ pedig egyszer S -ben, így a soroknak két helyettesítését kapjuk, melyek az $x = 1, y = 2, z = 3$ változóhelyettesítést szolgáltatják. A fejhez tartozó (x, z) sor minden behelyettesítésre $(1, 3)$. Az $(1, 3)$ sor tehát kétszer fordul elő a fejhez tartozó H relációban, és más sora nem is lesz. Azaz az alábbi reláció lesz a szabályból kapott fejhez tartozó reláció:

1	3
1	3

Általánosan úgy lehet megfogalmazni, hogy ha az $(1, 2)$ sor n -szer fordul elő R -ben, a $(2, 3)$ sor pedig m -szer fordul elő S -ben, akkor az $(1, 3)$ sor nm -szer fordul elő a H -ban. □

Ha egy relációt több szabály ír le, akkor az eredmény a szabályok által előállított összes sorból képzett multihalmazok uniója.

5.23. példa. Tekintsük a H relációt, amelyet két szabállyal definiálunk:

$$H(x, y) \leftarrow S(x, y) \text{ AND } x > 1$$

$$H(x, y) \leftarrow S(x, y) \text{ AND } y < 5$$

ahol az $S(B, C)$ reláció olyan, mint az 5.22. példában: $S = \{(2, 3), (4, 5), (4, 5)\}$. Az első szabály S minden sorát H -ba teszi, mivel minden sor első komponense nagyobb 1-nél. A második szabály alapján csak a $(2, 3)$ sor kerül be H -ba, mivel $(4, 5)$ -re nem teljesül, hogy $y < 5$. Azaz a H eredményreláció két darab $(2, 3)$, illetve két darab $(4, 5)$ sort tartalmaz. \square

5.3.7. Feladatok

5.3.1. feladat. Írjuk fel Datalogban a 2.4.1. feladat valamennyi lekérdezését. Kizárolag biztonságos szabályokat használunk. A bonyolultabb relációs algebrai részkifejezések megfogalmazásához használunk IDB-predikátumokat.

5.3.2. feladat. Írjuk fel Datalogban a 2.4.3. feladat valamennyi lekérdezését. Itt is használhatunk IDB-predikátumokat, és most is kizárolag biztonságos szabályokkal dolgozzunk.

!! 5.3.3. feladat. Ahhoz, hogy a fejhez rendelt reláció véges legyen abban az esetben, ha minden 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, hogy a relációs predikátumokhoz véges relációkat hozzárendelve a fejhez rendelt reláció is véges.

5.4. A relációs algebra és a Datalog

A 2.4. alfejezetben szereplő relációs algebrai operátorok mindenike kifejezhető egy vagy több Datalog-szabály segítségével. Ebben az alfejezetben valamennyi operátort egyenként megvizsgáljuk. Meg kell vizsgálnunk, hogyan kombinálhatjuk a Datalog-szabályokat összetettebb algebrai kifejezések leírására. Az is igaz lesz, hogy bármely biztonságos Datalog-szabály kifejezhető lesz relációs algebrában is, habár mellőzni fogjuk ennek a bizonyítását. Ezek ellenére a Datalog-lekérdezések nagyobb kifejezőerővel bírnak, mint a relációs algebra, ha megengedjük néhány szabály kölcsönhatását is. Ebben az esetben kifejezhetjük a rekurziót is, amelyre a relációs algebrában nem volt lehetőségünk (lásd az 5.35. példánál).

5.4.1. Boole-műveletek

A relációs algebra Boole-műveletei (az unió, a metszet és a halmazkülönbség) könnyen kifejezhetők Datalogban. Használjuk a következő három módszert a leíráshoz. Az R és S relációkat ugyanannyi attribútummal rendelkezőnek tekintjük, számuk legyen n . A szükséges szabályok leírását minden esetben a **Válasz** nevű fejpredikátum használatával oldjuk meg. Tegyük ezt annak ellenére, hogy bármilyen nevet használhatnánk az eredmény leírására, és hogy különböző műveletek eredményéhez különböző predikátumokat kellene választanunk.

- Vegyük az $R \cup S$ uniót, használunk a leíráshoz két szabályt és n különböző változót:

$$a_1, a_2, \dots, a_n$$

Az egyik szabály csak az $R(a_1, a_2, \dots, a_n)$ részcélból áll, míg a másik csak az $S(a_1, a_2, \dots, a_n)$ részcélból. Legyen minden két szabálynak a fejérésze $\text{Válasz}(a_1, a_2, \dots, a_n)$. Így az eredmény az, hogy R összes sora és S összes sora is benne lesz a válaszrelációban.

- Vegyük az $R \cap S$ metszetet. Egyetlen szabály lesz, amelynek a törzse az alábbi:

$$R(a_1, a_2, \dots, a_n) \text{ AND } S(a_1, a_2, \dots, a_n)$$

A fej legyen a $\text{Válasz}(a_1, a_2, \dots, a_n)$. Ebben az esetben egy sor pontosan akkor lesz benne a válaszrelációban, ha minden R -ben, minden S -ben benne van.

- Vegyük az $R - S$ különbséget. Egyetlen szabály lesz, amelynek a törzse az alábbi:

$$R(a_1, a_2, \dots, a_n) \text{ AND NOT } S(a_1, a_2, \dots, a_n)$$

A fej legyen a $\text{Válasz}(a_1, a_2, \dots, a_n)$. Ekkor egy sor pontosan akkor lesz benne a válaszrelációban, ha benne lesz R -ben, de nem lesz benne S -ben.

5.24. példa. Legyenek az $R(A, B, C)$ és $S(A, B, C)$ sémák a két relációt séma. A félreértések elkerülése végett itt különböző predikátumneveket fogunk használni a végeredményekre ahelyett, hogy minden Válasz -nak neveznénk.

Az $R \cup S$ meghatározásához a következő két szabályt használjuk:

1. $U(x, y, z) \leftarrow R(x, y, z)$
2. $U(x, y, z) \leftarrow S(x, y, z)$

Az 1. szabály fejezi ki, hogy R minden sora az U IDB-reláció egy sorát képezi. Hasonlóan a 2. szabály azt fejezi ki, hogy S minden sora benne van U -ban.

Az $R \cap S$ kiszámításához használjuk az alábbi szabályt:

$$I(a, b, c) \leftarrow R(a, b, c) \text{ AND } S(a, b, c)$$

Végül a

$$D(a, b, c) \leftarrow R(a, b, c) \text{ AND NOT } S(a, b, c)$$

szabály az $R - S$ halmazkülönbség kiszámítására való. \square

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.^a Írjuk át például az 5.24. példában felírt második szabályt a következő módon:

$$U(a,b,c) \leftarrow S(a,b,c)$$

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

^a Ezt a kiértékelést azért lehető meg, mert a Datalog-program is Horn-klöz program, amelyben a szabályok egymástól függetlenek, így a szabályok kiértékelésének sorrendje tetszőleges, mely Prolog-programokra nem érvényes. (A lektor megjegyzése.)

5.4.2. Vetítés

Az R reláció vetítésének kiszámításához 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.

5.25. példa. Tegyük fel, hogy a

`Filmek(filmcím, év, hossz, műfaj, stúdiónév, producerAzon)`

relációt szeretnénk levetíteni az első három attribútumára. A

`P(fc,é,h) ← Filmek(fc,é,h,m,s,p)`

szabály pontosan a vetítés eredményének megfelelő P relációt adja meg. □

5.4.3. Kiválasztás

A kiválasztás operátor kifejezése Datalogban már bonyolultabb dolog, mint az eddigi operátorok átfírása. A legegyszerűbb eset, amikor a kiválasztás feltétele egy vagy több aritmetikai összehasonlítás AND szócskával történő összekap-

csolásából 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 megadunk 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 lépésekben a relációs részcélt megadtuk.

5.26. példa. Az alábbi kiválasztást

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

a következő Datalog-szabállyal fejezhetjük ki:

$$S(fc, é, h, m, s, p) \leftarrow \text{Filmek}(fc, é, h, m, s, p) \text{ AND } h \geq 100 \text{ AND } s = 'Fox'$$

A kiértékelés eredménye a K reláció. Figyeljük meg, hogy minden részcélban a h és s változók a Filmek reláció hossz és stúdióNév attribútumainak felelnek meg. \square

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, azonos fejű szabállyal. Az i -edik szabály a kiválasztás i -edik feltételének felel meg.

5.27. példa. Cseréljük ki az 5.26. példa kiválasztásában az AND szócskát OR szócskára:

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

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

1. $S(fc, é, h, m, s, p) \leftarrow \text{Filmek}(fc, é, h, m, s, p) \text{ AND } h \geq 100$
2. $S(fc, é, h, m, s, p) \leftarrow \text{Filmek}(fc, é, h, m, s, p) \text{ AND } s = 'Fox'$

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

Az AND, OR és NOT logikai operátorok tetszőleges alkalmazásával már meg lehetősen bonyolult kiválasztási feltételek is megfogalmazhatók. Egy jól ismert módszer segítségével (amelynek az ismertetésétől most eltekintünk), ezek a logikai kifejezések „diszjunktív normálformára” hozhatók. A diszjunktív normálforma „konjunkciók” OR szócskával történő összekapcsolása. Egy *konjunkció* literálok AND szócskával történő összekapcsolása, egy *literál* pedig egy negált vagy egy nem negált összehasonlítás.²

A literálokat részcélokkal tudjuk reprezentálni, amelyek előtt szerepelhet egy NOT szócska is. Negált aritmetikai részcél esetén a NOT szócska bevhető az összehasonlításba. Például a NOT $x \geq 100$ feltétel felírható $x < 100$ alakban. Bármely konjunkció megfelel egyetlen olyan Datalog-szabálynak, amelynek részcéljai maguk az összehasonlítások. Végül, bármely diszjunktív normálformájú kifejezés felírható a konjunkcióinak megfelelő Datalog-szabályok segítségével. A szabályok kiértékelésének eredménye tulajdonképpen a konjunkciók eredményeinek egyesítése.

5.28. példa. Az 5.27. példában a fenti algoritmus egy egyszerű példáját láthatunk. Ha az előző példa feltételét negáljuk, akkor egy jóval bonyolultabb kifejezést kapunk:

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

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:

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

A NOT operátort bevhetjük az összehasonlítások belsejébe:

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

Ez a kifejezés Datalogban a következő szabállyal írható fel:

$$S(fc, é, h, m, s, p) \leftarrow \text{Filmek}(fc, é, h, m, s, p) \text{ AND } h < 100 \text{ AND } s \neq 'Fox'$$

□

5.29. példa. Vegyük egy hasonló példát, ahol a kiválasztás feltételében AND operátorral összekapcsolt feltételek negáltja szerepel. Ezúttal a DeMorgan-szabály második formáját alkalmazzuk, amely szerint az AND negálja a negációk OR-ja. Kezdjük a relációs algebrai kifejezéssel:

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

² Lásd: A. V. Aho, J. D. Ullman, *Foundations of Computer Science*, Computer Science Press, New York, 1992.

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

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

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

Ezután bevíssük a NOT operátort az összehasonlítások belsejébe:

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

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

1. $S(fc, é, h, m, s, p) \leftarrow \text{Filmek}(fc, é, h, m, s, p) \text{ AND } l < 100$
2. $S(fc, é, h, m, s, p) \leftarrow \text{Filmek}(fc, é, h, m, s, p) \text{ AND } s \neq 'Fox'$

□

5.4.4. 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.

5.30. példa. Tekintsük az 5.24. példa relációit. Az R és S egyaránt három attribútummal rendelkezik. Az $R \times S$ szorzatot a következő Datalog-szabállyal fejezhetjük ki:

$$P(a, b, c, x, y, z) \leftarrow R(a, b, c) \text{ AND } S(x, y, z)$$

Az R részcél változónak nevét az ábécé elejéről vettük, míg az S részcél változónak nevét az ábécé végéről. Az R és S részcélok minden a hat változója megjelenik a szabály fejében. □

5.4.5. Összekapcsolás

Két reláció természetes összekapcsolásának kifejezése Datalogban nagyon hasonlít a két reláció szorzatának megfelelő szabályához. 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.

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

$$\text{TÖ}(a, b, c, d) \leftarrow R(a, b) \text{ AND } S(b, c, d)$$

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

A théta-összekapcsolás hasonló módon írható fel Datalogban. A 2.4.12. 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.

5.32. példa. Tekintsük 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:

$$\begin{aligned} \text{Ö}(a, ub, uc, vb, vc, d) &\leftarrow U(a, ub, uc) \text{ AND } V(vb, vc, d) \text{ AND} \\ &a < d \text{ AND } ub \neq vb \end{aligned}$$

Az U reláció B attribútumához rendelt változónak az ub nevet adtuk, és ugyanígy kapták nevüket a vb , uc , vc változók is. A két relációt 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. \square

Ha a théta-összekapcsolás feltétele nem egy konjunkció, akkor a feltételt az 5.4.3. alfejezetben tárgyalt módon átalakítjuk diszjunktív normálformára. 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 relációt összesen van.

5.33. példa. Módosítsunk egy kicsit az 5.32. példában megadott relációs algebrai kifejezésen. Cseréljük ki az AND operátort OR operátorra. Ily módon a feltétel rögtön diszjunktív normálformában van, hiszen nem szerepel benne negáció. Két konjunkció van, minden kettő egyetlen literálból áll. A kifejezés:

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

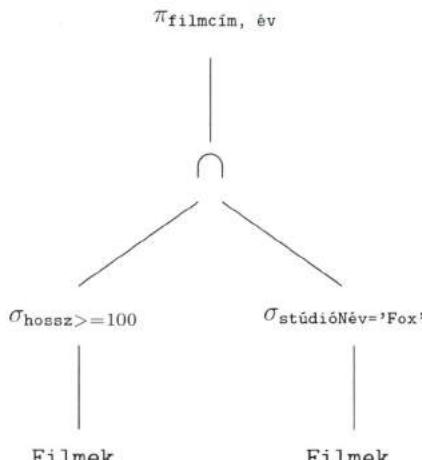
Ha a változókat ugyanúgy nevezzük, mint az 5.32. példában, akkor a következő két szabályt kapjuk:

1. $\ddot{\cup}(a, ub, uc, vb, vc, d) \leftarrow U(a, ub, uc) \text{ AND } V(vb, vc, d) \text{ AND } a < d$
2. $\ddot{\cup}(a, ub, uc, vb, vc, d) \leftarrow U(a, ub, uc) \text{ AND } V(vb, vc, d) \text{ AND } ub \neq vb$

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

5.4.6. Kifejezések megadása Datalogban

Datalog-szabályokkal nem csak elemi relációs algebrai operátorokat fejezhetünk ki, hanem alapjában véve bármilyen relációs algebrai kifejezést. Az ötlet minden össze annyi, hogy felrajzoljuk a relációs algebrai kifejezéshez tartozó kifejezésfát, és minden egyes belső csúcsra megadjuk a megfelelő IDB-predikátumot. Az IDB-predikátumhoz tartozó szabályt vagy szabályokat úgy kapjuk, hogy a fa megfelelő csúcsában található operátorra alkalmazzuk. A kifejezésfa extenzionális operandusaira (amelyek tulajdonképpen az adatbázis relációi) a hozzájuk tartozó predikátumokkal hivatkozhatunk. A belső csúcsok operandusaira a megfelelő IDB-predikátumokkal hivatkozhatunk. Az algebrai kifejezés eredménye egy olyan reláció, amely a kifejezésfa gyökerében szereplő predikátumnak felel meg.



5.9. ábra. A kifejezésfa

5.34. példa. Vegyük a 2.17. példa relációs algebrai kifejezését.

$$\pi_{\text{filmcím, év}}(\sigma_{\text{hossz}\geq 100}(\text{Filmek}) \cap \sigma_{\text{stúdiónév}='Fox'}(\text{Filmek}))$$

A megfelelő kifejezésfát a 2.18. ábrán láthattuk, de a könnyebb követhetőség érdekében az 5.9. ábrán is ugyanez látható. Négy IDB-predikátumot kell felírnunk a négy belső csúcsnak megfelelően. Mindegyik predikátumhoz egyetlen szabály fog tartozni, amint az az 5.10. ábrán látható.

A két alsó belső csúcs egyszerű kiválasztás a *Filmek* EDB-relációra alkalmazva. A hozzájuk tartozó IDB-predikátumok a W és az X , amelyeket az 5.10. ábra első két szabályában írtunk fel. Az első szabály például a *Filmek* reláció azon sorait adja meg, amelyekben a film hosszúsága legalább 100 perc.

1. $W(fc, \acute{e}, h, m, s, p) \leftarrow \text{Filmek}(fc, \acute{e}, h, m, s, p) \text{ AND } h \geq 100$
2. $X(fc, \acute{e}, h, m, s, p) \leftarrow \text{Filmek}(fc, \acute{e}, h, m, s, p) \text{ AND } s = 'Fox'$
3. $Y(fc, \acute{e}, h, m, s, p) \leftarrow W(fc, \acute{e}, h, m, s, p) \text{ AND } X(fc, \acute{e}, h, m, s, p)$
4. $\text{Válasz}(fc, \acute{e}) \leftarrow Y(fc, \acute{e}, h, m, s, p)$

5.10. ábra. Összetett relációs algebrai kifejezésnek megfeleltetett Datalog-szabályok

A harmadik szabály a W és X metszetének megfelelő Y predikátumot adja meg az 5.4.1. alfejezetben tanult szabályforma használatával. Végezetül a negyedik szabály fogalmazza meg a kérdésre adott választ, az Y *filmcím* és *év* attribútumaira történő levétítése által. A vetítés leírására az 5.4.2. alfejezetben bemutatott módszert használtuk.

Vegyük észre, hogy Y -t egyetlen szabály írja le, ezért az 5.10. ábrán lévő 4. szabály Y részcéljának helyébe behelyettesíthetjük a szabály törzsét. Ezek után a W és az X részcélok is helyettesíthetjük az 1., illetve 2. szabályok törzsével. Mivel a *Filmek* részcél minden törzsben szerepel, ezért az egyiket elhagyhatjuk. Eredményként így egyetlen szabályt kapunk.

$\text{Válasz}(fc, \acute{e}) \leftarrow \text{Filmek}(fc, \acute{e}, h, m, s, p) \text{ AND } h \geq 100 \text{ AND } s = 'Fox'$

□

5.4.7. A relációs algebra és a Datalog összehasonlítása

Az 5.4.6. alfejezetnek megfelelően minden alap relációs algebrai kifejezés kifejezhető Datalog-lekérdezésként. A kiterjesztett relációs algebrának viszont vannak olyan műveletei, mint például az 5.2. alfejezetben leírt csoportosítás vagy összesítés, amelyeknek nincs megfelelő Datalog-változata. Hasonlóan a Datalog nem támogatja a multihalmaz-műveleteket sem, mint például az ismétlődések megszüntetését.

Másrészről igaz az, hogy tetszőleges Datalog-szabály kifejezhető relációs algebrában. Azaz, az alap relációs algebrában írhatunk olyan lekérdezést, amely ugyanazt a sorhalmazt adja, mint amelyet a szabály feje eredményez.

Amikor viszont Datalog-szabályok gyűjteményét vizsgáljuk, ez a helyzet megváltozik, hiszen Datalog-szabályokkal a rekurziót is tudjuk kezelni, amelyet a relációs algebrában viszont már nem tudunk megtenni. Ennek az oka, hogy az IDB-predikátumokat szabályok törzsében is használhatjuk, és így egy szabály fejéhez meghatározott sorokat tovább használhatjuk szabályok törzsében is. Ezáltal pedig több sort kaphatunk a fejrészben. Ezen a ponton nem tárgyaljuk, hogy milyen bonyodalmakat okozhat ez a megközelítés, főként azokban az

esetekben, mikor negált részcél is van. A következő példával szemléltetjük a rekurzív Datalogot.

5.35. példa. Tegyük fel, hogy adott egy $\text{Él}(X, Y)$ reláció, ami irányított éleket jelent az X és Y csúcsok között. Megfogalmazhatjuk az él reláció tranzitív lezártját, azaz az $\text{Út}(X, Y)$ relációt, ami azt fejezi ki, hogy X csúcsból Y csúcsba van egy legalább 1 hosszú út, azaz:

$$\begin{aligned} 1. \text{ Út}(X, Y) &\leftarrow \text{Él}(X, Y) \\ 2. \text{ Út}(X, Y) &\leftarrow \text{Él}(X, Z) \text{ AND } \text{Út}(Z, Y) \end{aligned}$$

Az első szabály azt jelenti, hogy minden él egyben út is. A második szabály pedig azt fejezi ki, hogyha van él az X csúcsból valamelyen Z csúcsba, illetve Z csúcsból van út Y csúcsba, akkor X -ből is van út Y -ba. Ha tekintjük a két szabályt, akkor 2 hosszú utakat kapunk. Ha vesszük az eddigi eljárásból nyert Út tényeket, és felhasználjuk azokat egy újabb alkalmazáshoz, akkor már 3 hosszú utakat kapunk. Ha ezekkel az Út tényekkel újra alkalmazzuk a szabályainkat, akkor 4 hosszú utakat nyerünk és így tovább. Végül megtaláljuk az összes utat, és a következő menetben már nem kapunk újabb tényeket. Ennél a pontnál befejezhetjük az eljárást. Ha nem kaptuk meg az $\text{Út}(a, b)$ tényt, akkor valójában nincs is út a gráfban az a és b csúcsok között. \square

5.4.8. Feladatok

5.4.1. 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 segítsével:

- 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) $\pi_{a,b}(R) \cap \rho_{U(a,b)}(\pi_{b,c}(S))$.

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

- a) $x = y$.
- b) $x < y \text{ AND } y < z$.

- c) $x < y \text{ OR } y < z$.
- d) $\text{NOT } (x < y \text{ OR } x > y)$.
- ! e) $\text{NOT } ((x < y \text{ OR } x > y) \text{ AND } y < z)$.
- ! f) $\text{NOT } ((x < y \text{ OR } x < z) \text{ AND } y < z)$.

5.4.3. 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.)

5.4.4. 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 az 5.4.2. példában felsorolt feltételek egyike. A feltételekben található aritmetikai összehasonlításokat tekintsük úgy, mint amelyik sorrendben az R egy attribútumát hasonlíta össze az S egy attribútumával. Az $x < y$ alakú összehasonlítás értelmezése tehát $R.x < S.y$.

! 5.4.5. 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 kifejezésekkel, amelyek eredménye megegyezik a fejnek megfelelő relációval.

- a) $P(x, y) \leftarrow Q(x, z) \text{ AND } R(z, y)$
- b) $P(x, y) \leftarrow Q(x, z) \text{ AND } Q(z, y)$
- c) $P(x, y) \leftarrow Q(x, z) \text{ AND } R(z, y) \text{ AND } x < y$

5.5. Összefoglalás

- ◆ *Relációk multihalmazokként:* A forgalomban lévő adatbázisrendszerben a relációkat multihalmaznak tekintik, amelyben ugyanazon sor többször is előfordulhat. A relációs algebra halmazokon értelmezett műveletei kiterjeszthetők multihalmazokra is, de van néhány algebrai azonosság, amely sérülni fog.

6. 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 Lekérdező Nyelv) rövidítése. Az SQL lekérdező lehetőségei nagyon hasonlóak a relációs algebrának az 5.2. alfejezetben bemutatott kiterjesztéséhez. Ezenfelül az SQL-nek vannak az adatbázist módosító utasításai (például sorok beszúrása és sorok törlése), valamint az adatbázisséma megadását lehetővé tevő utasításai. Így az SQL adatmanipulációs és adatleíró nyelvként is szolgál. Az SQL-szabványban további adatbázis-kezelő utasítások is szerepelnek, ezekkel a 7. és 9. fejezetben foglalkozunk.

Az SQL-nek számos különböző verziója van. Először is három fő szabványa ismert. Az ANSI (American National Standards Institute – Amerikai Nemzeti Szabvány Intézet) SQL és egy 1992-ben elfogadott módosított szabvány, az SQL-92 vagy SQL2. A jelenleg SQL-99-nek (korábban SQL3-nak) nevezett szabványra kiterjeszti az SQL2-t az objektumrelációs környezetre, és számos új lehetőséggel bővíti. Az SQL-99-nek is vannak már kiterjesztései, ezeket együtt SQL:2003-nak nevezik. Ezenkívül léteznek az SQL-nek a nagy adatbázis-kezelő rendszereket forgalmazó cégek által készített verziói is. Ezek mindegyike kielégíti az eredeti ANSI-szabványt. Nagyrészt összhangban vannak az SQL2-vel is, mindegyikük az SQL2 egy változatának és kiterjesztésének tekinthető, megvalósítják az SQL-99 és az SQL:2003 szabványokban rögzített lehetőségek egy részét is.

Ez a fejezet az SQL alapjaival, a lekérdező nyelvvel és az adatbázis-módosító utasításokkal foglalkozik. Az adatbázisrendszerek alapvető programozási egységeként bevezetjük a „tranzakció” fogalmát is. Az áttekintésünk egyszerűsített, csak érzékeltetjük, hogy az adatbázis-műveletek hogyan hathatnak egymásra és milyen buktatóik vannak.

A következő fejezetben a megszorításokkal és a triggerekkel foglalkozunk, ezek az adatbázis tartalma feletti felhasználói ellenőrzés lehetőségeit bővítenek. A 8. fejezetben olyan lehetőségeket mutatunk be, melyekkel az SQL-lekérdezéseinket tudjuk hatékonyabbá tenni, elsősorban az indexek és hasonló struktúrák deklarálásával. A 9. fejezet az adatbázisok programokból való hasz-

nálatával foglalkozik. E programok olyan nagy rendszerek részei, mint például a weben közösen használt nagy (szerver) kiszolgáló rendszerek. Láttni fogjuk, hogy az SQL-lekérdezéseket és más SQL-műveleteket túlnyomó részben sosem önmagukban használjuk, hanem a hagyományos programozási nyelven írt programokba beágyazottan, így ezeknek együtt kell működniük.

Végül a 10. fejezet egy sor fejlett adatbázis-programozási koncepciót magyaráz el. Ilyenek a rekurzív SQL, az SQL hozzáférés-ellenőrzési és biztonsági lehetőségei, az objektumrelációs SQL, és az adatok adatkocka modellje.

Ebben és a következő fejezetekben a célunk az SQL megismertetése az olvasóval, mégpedig bevezetés, mintsem kézikönyv szintjén. Így csak a leggyakrabban használt részekre koncentrálunk, és olyan példákat mutatunk, melyek nemcsak szabványosak, hanem a kereskedelmi adatbázisrendszerekkel is összhangban vannak. Az irodalomjegyzékbeli hivatkozások olyan publikációkat tartalmaznak, amelyekben megtalálható a nyelv részletesebb leírása és a különböző verziók.

6.1. Egyszerű lekérdezések az SQL-ben

Az SQL legegyszerűbb lekérdezései azon sorokra vonatkoznak, melyek egy bizonos 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.

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

6.1. ábra. Minta adatbázisséma, ismétlés

6.1. példa. A következő példákban a film adatbázis 2.2.8. alfejezetben leírt adatbázissémáját fogjuk alkalmazni. Ezt az adatbázissémát a 6.1. ábrán ismét bemutatjuk.

Első lekérdezésként a

Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
relációból kérdezzük le az összes olyan filmet, melyet a Disney stúdió 1990-ben készített. A megfelelő SQL-utasítás:

```
SELECT *
FROM Filmek
WHERE stúdióNév = 'Disney' AND év = 1990;
```

Hogyan is használjuk az SQL-t?

Ebben a fejezetben feltételezzük, hogy van egy általános lekérdező programunk, melynek begépelhetjük az SQL-lekérdezéseket és más SQL-utasításokat, és a lekérdező program ezeket végrehajtja. A gyakorlatban ilyen általános lekérdező programot ritkán használnak. Sokkal inkább hagyományos programozási nyelveken – mint a C vagy a JAVA (*befogadó nyelvek*) – írott programokkal dolgozunk. Ezen programokban az adott befogadó nyelv speciális könyvtárait használva az adatbázisra vonatkozó SQL-utasításokat is kiadhatnak. Az adatokat a befogadó nyelv változóiból az SQL-utasításokba, ezen utasítások végrehajtásának eredményeit pedig az adatbázisból a befogadó nyelv változóiba mozgatják át. Ezt a 9. fejezetben sokkal részletesebben tárgyaljuk.

Ez a lekérdezés az SQL-lekérdezések jellegzetes, select-from-where alakját mutatja be.

- A **FROM** záradék azon relációt vagy relációkat adja meg, melyekre a lekérdezés vonatkozik. A példánkban a lekérdezés a **Filmek** relációra vonatkozik.
- A **WHERE** záradék egy feltétel, nagyon hasonlít a relációs algebrában használt kiválasztási feltételhez. A lekérdezés válaszába azon sorok kerülnek, melyek kielégítik az adott feltételt. A példában a feltétel az, hogy a **stúdióNév** attribútum értéke '**Disney**' és az **év** attribútum értéke 1990 legyen. Azon sorok, melyek minden feltételt kielégítik, megjelennek a lekérdezés eredményében, 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 teljes sort tartalmazni fogja a válasz. A lekérdezés eredménye az a reláció, amely tartalmazza az összes sort, melyeket ez az eljárás előállít.

A lekérdezés feldolgozásának egyik módja az, hogy a **FROM** záradékbeli relációknak 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 **Filmek**-beli 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:

<i>filmcím</i>	<i>év</i>	<i>hossz</i>	<i>műfaj</i>	<i>stúdióNév</i>	<i>producerAzon</i>
Micsoda nő!	1990	119	vígjáték	Disney	999

Hogyan olvassunk és írunk lekérdezéseket?

A select-from-where lekérdezések értelmezésének általában legegyszerűbb módja, ha először a FROM záradékot keressük meg, innen megtudjuk, hogy a lekérdezésben mely relációra hivatkozunk. Aztán a WHERE záradékot keressük, belőle megtudhatjuk, hogy a lekérdezés szempontjából mely sorok számítanak fontosnak. Végül a SELECT záradékból látjuk, milyen lesz az eredmény. Ugyanez a sorrend – from, majd where, aztán select – gyakran hasznosnak bizonyulhat akkor is, amikor a lekérdezést mi magunk írjuk.

(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 év = 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. □

6.1.1. 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

6.2. példa. Tételezzük fel, hogy a 6.1. példa lekérdezését szeretnénk úgy módosítani, hogy csak a film címét és hosszát adja vissza. A megfelelő utasítás:

```
SELECT filmcím, hossz
  FROM Filmek
 WHERE stúdióNév = 'Disney' AND év = 1990;
```

Az eredmény egy kétoszlopos tábla, a **filmcím** 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:

¹ Igy a SELECT kulcsszó főleg a relációs algebra vetítés műveletének, míg az algebra kiválasztás művelete az SQL-lekérdezés WHERE záradékának felel meg.

<i>filmcím</i>	<i>hossz</i>
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 eredeti oszlopnév helyett fog szerepelni. Az AS nem kötelező. A másodnév, minden elválasztójel (pont, vessző) nélkül, rögtön azon attribútum után következik, melynek neveként fog szereálni.

6.3. példa. A 6.2. példát módosíthatjuk úgy, hogy olyan relációt eredményezzen, melynek attribútumai *név* és *időtartam*, az eredeti *filmcím* és *hossz* helyett:

```
SELECT filmcím AS név, hossz AS időtartam
FROM Filmek
WHERE stúdióNév = 'Disney' AND év = 1990;
```

Az eredmény megegyezik a 6.2. példa eredményével, de az oszlopnevek *név* és *időtartam*. Például az eredményreláció kezdődhet így:

<i>név</i>	<i>időtartam</i>
Micsoda nő!	119
...	...

□

További lehetőség a SELECT záradékban a kifejezés használata az attribútum helyett, valamint, hogy az 5.2.5. alfejezetben tárgyalt kiterjesztett vetítés listájához hasonlót adjunk meg a SELECT listában is. A 6.4. alfejezetben láthatjuk, hogy a SELECT lista ugyanúgy tartalmazhat összesítéseket, mint ahogy az 5.2.4. alfejezetben bemutatott γ művelet is.

6.4. példa. Tételezzük fel, hogy ugyanazt az eredményt szeretnénk kapni, mint a 6.3. példában, de a hossz órákban legyen kifejezve. A SELECT záradékot ki-cserélhetjük a következőre:

```
SELECT filmcím AS név, hossz*0.016667 AS hosszÓrákban
```

Így ugyanazokat a filmeket adja, de az időtartamot órákban kapjuk meg, és a második oszlop neve *hosszÓrákban* lesz:

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. mindenkorban 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 karakterek, melyek egyike sem egyezik meg a `FROM` kulcsszóval.

<i>név</i>	<i>hosszÓrákban</i>
Micsoda nő!	1.98334
...	...

□

6.5. 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 filmcím, hossz*0.16667 AS hossz, 'óra' AS Órákban
FROM Filmek
WHERE stúdióNév = 'Disney' AND év = 1990;
```

olyan sorokat generál, mint:

<i>filmcím</i>	<i>hossz</i>	<i>Órákban</i>
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 olyan benyomást kelt, mintha a második oszlopbeli érték egységeként szerepelne. □

6.1.2. Kiválasztás az SQL-ben

Az SQL WHERE záradéka kiterjeszti a relációs algebra kiválasztás operátorát. A WHERE-t ahhoz hasonló feltételkifejezések követhetik, mint amelyeket a közismert C vagy Java nyelvek is használnak.

A hat alapvető összehasonlítási operátor: `=`, `<>`, `<`, `>`, `<=`, `>=` segítségével értékeket hasonlíthatunk össze, s ezekkel építhetjük fel a feltételkifejezéseket. Ezeknek az operátoroknak jelentése megegyezik a C-ben használtakkal, de az

SQL „ $<>$ ” (jelentése: „nem egyenlő”) a C-ben használt „!=“ -vel azonos, az „=“ (jelentése: egyenlő) a C-ben használt „==“ -vel egyezik meg.

Az összehasonlítható értékek között lehetnek konstansok és a FROM utáni relációk attribútumai. Az értékekre alkalmazhatjuk a szokásos matematikai alapműveleteket is, mint például a +, * stb., mielőtt összehasonlítjuk őket. Például az $(\text{év} - 1930) * (\text{év} - 1930) < 100$ kifejezés igaz az 1930-tól legfeljebb 9 év távolságra levő évekre. Alkalmazhatjuk az összekapsolás műveletet is: (||) karakterláncokra; például ’labda’ || ’rúgás’ értéke ’labdarúgás’.

Egy példát láthattunk az összehasonlításra a 6.1. példában:

```
stúdióNév = 'Disney'
```

A Filmek 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: -12.34 vagy 1.23E45.

Az összehasonlítás eredménye egy igazságérték: TRUE (igaz) vagy FALSE(hamis).² A logikai értékeket kombinálhatjuk az AND, OR és NOT (az „és”, „vagy” és „nem”) logikai műveletek segítségével, amelyeket a szokásos jelentsükkel használunk. A 6.1. példában láthattuk, hogyan lehet két feltételt az AND műveettel 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.

6.6. példa. Tekintsük a következő lekérdezést:

```
SELECT filmcím
FROM Filmek
WHERE (év > 1970 OR hossz < 90) AND stúdióNév = 'MGM' ;
```

Ez a lekérdezés azon filmcímeket adja meg, amelyeket az MGM Stúdió készített, és amelyek 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 ugyanolyan, mint a legtöbb programozási nyelvben: az AND megelőzi az OR műveletet, a NOT pedig megelőzi mindenkitőjüket. □

6.1.3. Karakterláncok összehasonlítása

Két karakterlánc egyenlő, ha a karaktereknek ugyanabból az egymás után következő sorozatából állnak. A 2.3.2. alfejezetből más ismerjük, hogy a karakterláncokat fix hosszú karakterláncok (CHAR), vagy változó hosszúságú karakterláncokban (VARCHAR) tároljuk. Ha különbözően deklarált karakterláncokat

² Valójában többféle igazságérték is létezik, lásd a 6.1.7. alfejezetet.

SQL-lekérdezések és a relációs algebra

Az eddig látott egyszerű SQL-lekérdezések minden az alábbi alakúak:

```
SELECT L  
FROM R  
WHERE C
```

ahol *L* kifejezések listája, *R* reláció, *C* pedig feltétel. Az ilyen lekérdezés jelentése megegyezik a relációs algebrai

$$\pi_L(\sigma_C(R))$$

kifejezés jelentésével. Ez az oka annak, hogy az értelmezést a FROM záradék relációjával kezdtük, melynek minden sorára ellenőriztük, hogy kielégíti-e a WHERE záradékban megadott feltételt, majd vettük fel a SELECT záradékban megadott attribútumokra és/vagy kifejezésekre.

hasonlítunk össze, akkor csak az aktuális karakterláncok hasonlítódnak, azaz az SQL figyelmen kívül hagyja a „kitöltő” („pad”) karaktereket, amelyeket azért használ, hogy az adatbázisban a karakterlánc elérje a deklarációjában megkívánt hosszát.

Amikor két karakterláncot összehasonlítunk egy aritmetikai összehasonlító operátorral, mint a $<$ vagy a \geq , azt szeretnénk tudni, hogy az egyik megelőzi-e a másikat lexikografikus rendezésben (azaz ábécérend szerint). Ha $a_1a_2 \dots a_n$ és $b_1b_2 \dots b_m$ 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. $a_1a_2 \dots a_n < b_1b_2 \dots b_m$ akkor is, ha $n < m$ és $a_1a_2 \dots a_n = b_1b_2 \dots b_n$, 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.

6.1.4. Mintával való összehasonlítás SQL-ben

Az SQL lehetővé teszi, hogy mintákkal is összehasonlíthassunk karakterláncokat. Az ilyen jellegű összehasonlítás formája:

s LIKE *p*

ahol *s* egy karakterlánc, *p* pedig egy *minta*. A minta egy olyan 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

Bitláncok megadása

Egy bitláncot egy B karakter és két idézőjel közötti 0 és 1 karakterek sorozata jelképezi. Így a B'011' egy három bites 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 jelképezi, és a számkezdő 0-kat nem hagyhatjuk el.

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.

6.7. 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 filmcím  
FROM Filmek  
WHERE filmcím LIKE 'Halálos _____';
```

Ez a lekérdezés olyan filmcímeket keres, melyek 15 karakter hosszúak, az első 7 karakter: 'Halálos', a nyolcadik üres, az utolsó 7 karakter pedig bármí lehet. A lekérdezés eredménye a feltételnek megfelelő filmcímek, mint például a *Halálos fegyver* vagy a *Halálos játszma*. □

6.8. példa. Azokat a filmeket keressük, melyeknek a címében megtalálható az 's' karaktersor. A megfelelő lekérdezés:

```
SELECT filmcím  
FROM Filmek  
WHERE filmcím LIKE '%''s%';
```

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 ''s' karaktersor egy szimpla idézőjelből és azt követő s-ből álló karaktersorozatra illeszkedik.

A minta két végén a % karakterek bármilyen karaktersornak megfelelnek, így a válasz olyan filmeket fog tartalmazni, mint például a *Wayne's World* (Wayne világa). □

Escape karakterek a LIKE típusú kifejezésekben

Mi történik, ha az a karaktorsor típus, melyre a LIKE kifejezéssel szeretnénk keresni, tartalmazza a % vagy _ karaktereket? Szükségiünk 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 karaktorsorban, és nem egy bármilyen karakterekből álló karaktorsorozatnak, illetve egy bármilyen karakternek. Például az

```
s LIKE 'x%/%x%' ESCAPE 'x'
```

kifejezésben x az escape karakter az x%/%x% mintában. Az x% karaktorsor egyetlen %-nak fog megfelelni. Így a minta minden olyan karaktorsorra illeszkedik, mely a % karakterrel kezdődik és fejeződik be. Jegyezzük meg, hogy csak a középső % bír „bármilyen karaktorsorozat” jelentéssel.

6.1.5. Dátumok és időpontok

Az SQL különböző megalósításai általában speciális típusként kezelik a dátum- és időtípusokat. Ezek az adatok aztán sokféleképpen tárolhatók, mint például 05/14/1948 vagy 14 May 1948. Ebben a részben csak az SQL-szabványt adjuk meg, amely aprólékosan leírja ezeket a formátumokat.

Egy *dátum* típusú konstanst a DATE kulcsszóval, utána pedig egy idézőjelek közötti speciális karaktorsorral 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, amely a hónapot jelképezi. Figyeljük meg, hogy egy egyszámjegyű hónap ki van egészítve elől egy 0 karakterrel. Végül következik egy újabb gondolatjel és két számjegy, amely a napot jelképezi. Ugyanúgy, mint a hónapok esetében, ha szükséges, a napot is kiegészítjük egy kezdő 0-val, hogy kétszámjegyű számot kapunk.

Egy *idő* konstans értékét hasonlóan a TIME kulcsszó és egy idézőjelek közötti karakterlánc segítségével lehet ábrázolni. A karaktorsorban az órának két számjegy felel meg a 24 órás rendszerben. Ezután kettőspont következik, két számjegy a perceknak, 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 diákok elhagyta az osztálytermet egy olyan óra után, mely délután 3-kor ért véget, tehát két és fél másodperccel három után.

Az időt a greenwich-i időhöz (GMT: Greenwich Mean Time) képesti + vagy – eltéréssel is megadhatjuk. Például a TIME '12:00:00-8:00' (az USA) nyugati parti zónaidőben delet jelent, mely 8 órával kevesebb a greenwich-i időnél.

A dátum és az idő együtt jelenik meg a TIMESTAMP típusú értékben. A TIMESTAMP értékben a dátumot egy üres karakter, majd az időpont követi. Így 1948. május 14-én déli 12 órát a TIMESTAMP '1948-05-14 12:00:00' érték reprezentálja.

A dátum- és időértékeket ugyanúgy összehasonlíthatjuk, mint a karakterlánc- vagy numerikus értékeket. A < jel dátumok esetén azt jelenti, hogy az első korábbi dátum mint az utóbbi, időértékek esetében pedig azt jelenti, hogy az első korábbi időpont (egy napon belül), mint a második.

6.1.6. A nullérték és műveletek nullértékekkel

Az SQL lehetővé teszi, hogy az attribútum értéke egy speciális NULL legyen, amit *nullértéknek* nevezünk. 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 egy nem ismert születésnap.
2. *Alkalmazhatatlan érték*: „Nincs olyan érték, aminek itt értelme lenne.” Például, ha a FilmSzínész relációnak lenne egy **hitves** attribútuma is, akkor az egyedi álló színészknél ennek az attribútumnak nullértéke lenne, hiszen nem tudhatjuk valaki 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.

Az 5.2.7. alfejezetben láttuk, hogy a külső összekapcsolás során is kerülhetnek egyes sorok bizonyos komponenseibe nullértékek. Az SQL is alkalmazza a külső összekapcsolás műveletet, így ha egy lekérdezés kiváltja a külső összekapcsolás művelet végrehajtását, akkor itt is keletkezhetnek nullértékek, bővebben lásd a 6.3.8. alfejezetben. Az SQL más esetekben is produkál nullértékeket. Például sorok beszűrásakor is keletkezhetnek nullértékek, amint azt a 6.5.1. alfejezetben láthatjuk.

A WHERE záradékban fel kell készülnünk annak lehetőségére, hogy valamely sor éppen felhasznált komponensének értéke NULL is lehet. Két fontos szabályt kell figyelembe vennünk, amikor NULL értékekkel dolgozunk.

1. Amikor egy aritmetikai műveletben, mint a \times vagy $a +$, legalább az egyik tag NULL, akkor az eredmény is NULL.

Nullértékekkel kapcsolatos csapdák

Feltételezhettünk, hogy a NULL egy olyan érték, amit nem ismerünk, de amely biztosan létezik. A valóságban azonban ez a feltételezés nem helytálló. Például tételezzük fel, hogy x egy sor egy bizonyos komponense, melynek az értelmezési tartománya egész számokból áll. Úgy érvelhetnénk, hogy $0 * x$ értéke biztosan 0, mivel függetlén attól, hogy mennyi az x értéke, a 0-val való szorzás eredménye 0. Mégis, ha x értéke NULL, akkor a 6.1.6. 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.

- Amikor egy NULLértéket hasonlítunk össze bármely más értékkel, beleértve a NULL-t is, 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* tekintetű konstansnak. Míg a fenti szabályok olyan esetben alkalmazandók, amikor egy kifejezés értéke NULL, a NULL-t nem használhatjuk direkt módon egy kifejezésben.

6.9. példa. Legyen x értéke NULL. Ekkor $x + 3$ értéke is NULL. Ennek ellenére $\text{NULL} + 3$ nem egy szabályos SQL-kifejezés. Hasonlóképpen $x = 3$ logikai értéke ISMERETLEN, hiszen nem tudjuk eldönteni, hogy az x értéke (amely NULL) egyenlő-e 3-mal. A $\text{NULL} = 3$ nem egy szabályos összehasonlítás az SQL-ben.
□

A szabványos útja annak, hogy megtudjuk, egy kifejezés értéke NULL-e vagy sem, az „ $x \text{ IS NULL}$ ” kifejezés segítségével történik. A kifejezés értéke IGAZ, ha x értéke NULL, illetve HAMIS, ha nem. Hasonlóképpen „ $x \text{ IS NOT NULL}$ ” értéke IGAZ, ha x nem NULL.

6.1.7. Az ISMERETLEN igazságérték

A 6.1.2. 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 előző szakaszban 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 (vagyis valahol az igaz és a hamis között). Ekkor:

- Két logikai értékre alkalmazott AND eredménye a két érték minimuma. Tehát $x \text{ AND } y$ értéke HAMIS, ha legalább az egyik hamis; ISMERETLEN, ha egyik sem HAMIS, de legalább az egyik ISMERETLEN; és IGAZ, ha mindenkető IGAZ.
- Két logikai értékre alkalmazott OR eredménye a két érték maximuma. Tehát $x \text{ OR } y$ értéke IGAZ, ha legalább az egyik IGAZ; ISMERETLEN, ha egyik sem IGAZ és legalább az egyik ISMERETLEN; és HAMIS, ha mindenkető HAMIS.
- A v logikai érték tagadásának értéke $1 - v$. Tehát NOT x értéke IGAZ, ha x HAMIS; HAMIS, ha x IGAZ; és ISMERETLEN, ha x értéke ISMERETLEN.

A 6.2. 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.

x	y	$x \text{ AND } y$	$x \text{ OR } y$	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

6.2. ábra. Igazságértékek táblázata a háromértékű logika esetén

A select-from-where utasítások WHERE záradékában szereplő SQL-feltételeket a rendszer minden egyes sorra ellenőrzi, 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 vagy ISMERETLEN, kizártuk az eredményből. Ez a helyzet – amint a következő példa is szemlélteti – egy újabb meglepő eredményhez vezet a nullértékekkel kapcsolatban, hasonlóan a „Nullértékekkel kapcsolatos csapdák” című bekeretezett részhez.

6.10. példa.

Tételezzük fel, hogy a következő relációra

Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)

kipróbáljuk a következő lekérdezést:

```
SELECT *
FROM Filmek
WHERE hossz <= 120 OR hossz > 120;
```

Ösztönösen arra gondolhatnánk, hogy a **Filmek** reláció egy másolatát fogjuk kapni, mivel mindegyik film hossza kielégíti a feltételeit.

Vizsgáljuk meg azonban azt az esetet, amikor léteznek olyan sorok, amelyekben a **hossz** attribútum értéke **NULL**. Így a **hossz > 120** és a **hossz <= 120** feltételek kiértékelésének eredménye **ISMERETLEN** lesz. A 6.2. ábra szerint két **ISMERETLEN** érték **OR** műveettel történő összekapcsolása **ISMERETLEN** értéket eredményez. Tehát minden olyan sor esetén, amelyben 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** hosszáértékű **Filmek** sort”. □

6.1.8. 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 és így tovább, ugyanúgy, ahogy az 5.2.6. alfejezet τ műveletében is. 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, ezt az attribútumlistában az érintett attribútum mellett a **DESC** kulcsszó megadásával jelezzük (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.

Az **ORDER BY** záradék a **WHERE** és minden más záradék (mint a **GROUP BY** és a **HAVING**, melyekkel a 6.4. alfejezetben foglalkozunk) után következik. A rendezés a lekérdezés eredményére vonatkozik, csak annak megjelenítésében van szerepe.

6.11. példa. Írjuk át a 6.1. példát, amely az 1990-es Disney-filmeket kérdezte le a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
relációból. A filmeket hossz szerint szeretnénk lekérdezni növekvően, és az egyenlő hosszúakat ábécérendben. A megfelelő utasítás:
```

```
SELECT *
FROM Filmek
WHERE stúdióNév = 'Disney' AND év=1990
ORDER BY hossz, filmcím;
```

A rendezéskor a **Filmek** összes attribútuma rendelkezésünkre áll akkor is, ha az eredményben nem jelennek meg. Így ha a „**SELECT ***” kifejezést „**SELECT produceAzon**”-ra cseréljük, a lekérdezés továbbra is érvényes marad. □

A rendezés további lehetősége, hogy az ORDER BY záradékot követő listában kifejezést is használhatunk éppen úgy, mint a SELECT záradékában is. Például az $R(A, B)$ reláció sorait rendezhetjük a sorok két komponensének összege szerint csökkenő sorrendbe is:

```
SELECT *
FROM R
ORDER BY A+B DESC;
```

6.1.9. Feladatok

6.1.1. 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?

6.1.2. feladat. Adjuk meg SQL-ben a következő lekérdezéseket, a fejezet elején említett film adatbázisra vonatkozóan:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

- a) Keressük meg az MGM stúdió címét.
- b) Keressük meg Sandra Bullock születési dátumát.
- c) Keressük meg az összes olyan filmsztárt, akik vagy egy 1980-as filmben szerepeltek, vagy egy olyanban, amelynek 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).

6.1.3. feladat. Adjuk meg a következő SQL-lekérdezéseket, amelyek a 2.4.1. feladat adatbázisára vonatkoznak:

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

A lekérdezések eredményeit szemléltessük a 2.4.1. feladat adataival.

- a) Keressük meg a modellszámát, sebességét, merevlemez-kapacitását azoknak a PC-knek, melyek ára 1000 \$ alatt van.
- b) Ugyanaz, mint az a) pontban, de nevezzük át a **sebesség** oszlopot **gigahertz**-re, a **merevlemez** oszlopot pedig **gigabajt**-ra.
- c) Keressük meg a nyomtatók gyártóit.
- d) Keressük meg azon laptopok modellszámát, memóriakapacitását és képernyőnagyságát, melyek több mint 1500 \$-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 és merevlemezméretét, melyek sebessége 3.2 és 2000 \$-nál olcsóbbak.

6.1.4. feladat. Adjuk meg SQL-ben a következő lekérdezéseket, melyek a 2.4.3. feladat adatbázissémájára alapulnak:

Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetek(hajó, csata, eredmény)

és a lekérdezések eredményeit szemléltessük a 2.4.3. példa adataival.

- a) Keressük meg az osztálynévét és országát azon hajóosztályoknak, melyek legalább 10 ágyúval rendelkeznek.
- b) Keressük meg az 1918 előtt felavatott hajókat, de az eredményoszlop neve legyen **hajóNév**.
- c) Keressük meg a csatákban elsüllyesztett hajók nevét és azon csaták nevét, melyben elsüllyedtek.
- d) Keressük meg azon hajókat, amelyek neve megegyezik a hajóosztályuk nevével.
- e) Keressük meg az „R” betűvel kezdődő hajók nevét.
- f) Keressük meg az olyan hajónéveket, melyek három vagy több szóból állnak (például „Nagy Lajos király”).

6.1.5. feladat. Legyenek a és b egész szám típusú attribútumok, melyek NULL értéket is felvethetnek. Az alábbi feltételek mindegyikére (ilyenek fordulhatnak elő például a WHERE záradékban) adjuk meg pontosan a feltételt kielégítő (a, b) -k halmazát, vegyük figyelembe azokat az eseteket is, amikor a és/vagy b NULL értékű.

- a) $a = 10 \text{ OR } b = 20$
- b) $a = 10 \text{ AND } b = 20$
- c) $a < 10 \text{ OR } a \geq 10$
- ! d) $a = b$
- ! e) $a \leq b$

! 6.1.6. feladat. A 6.10. példában a

```
SELECT *
FROM Filmek
WHERE hossz <= 120 OR hossz > 120;
```

lekérdezéssel foglalkoztunk, amely a NULL hosszú filmek esetén össztöneinkkel ellentétesen viselkedett. Keressünk egyszerűbb, ezzel ekvivalens lekérdezést, amelynek WHERE záradékában egyszerű feltétel áll (olyan feltétel, amelyben nincs sem AND sem OR).

6.2. Több relációra vonatkozó lekérdezések

A relációs algebra legfőbb erőssége az, hogy két vagy több relációt tud kombinálni összekapcsolásokon, szorzatokon, egyesítésekben, metszeteken és különbségeken keresztül. Ezen műveletek mindegyikét alkalmazhatjuk az SQL-ben. A halmazműveletek – egyesítés, metszet és különbség – direkt módon jelennek meg az SQL-ben, ahogy a 6.2.5. alfejezetben szemléltetni fogjuk. Először azonban vizsgáljuk meg, hogy a select-from-where utasítás milyen módon teszi lehetővé szorzatok és összekapcsolások képzését.

6.2.1. 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.

6.12. példa. Tegyük fel, hogy szeretnénk megtudni a *Csillagok háborúja* című film producerének nevét. A következő két relációra van ehhez szükség:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A producer azonosító száma a Filmek relációban megtalálható, így ez a Filmek relációján végrehajtott egyszerű lekérdezéssel megszerezhető. Ezután egy második, a GyártásIrányító reláción végrehajtott lekérdezéssel megtaláljuk a producerazonosítóhoz tartozó személy nevét.

Ezt a két lépést összevonhatjuk egyetlen lekérdezésbe, mely a Filmek és a GyártásIrányító relációkra vonatkozik:

```
SELECT név
FROM Filmek, GyártásIrányító
WHERE Filmek.filmcím = 'Csillagok háborúja',
      AND producerAzon = azonosító;
```

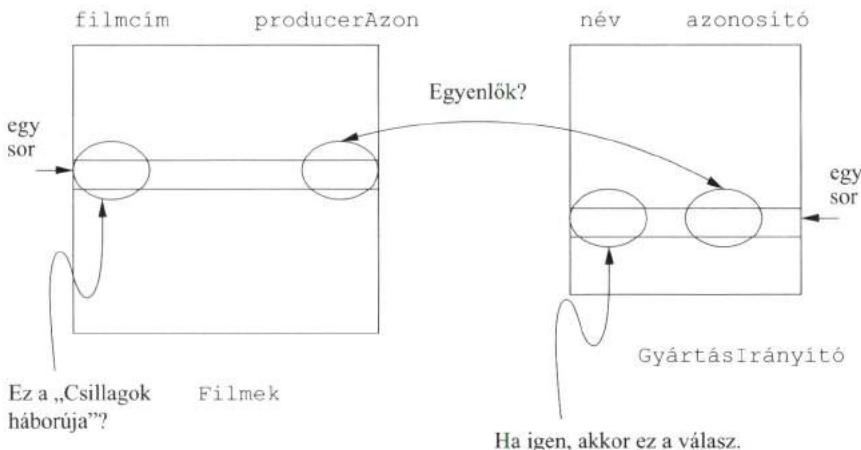
Ez a lekérdezés az összes Filmek és GyártásIrányító sorpárt megvizsgálja, és ezekre ellenőrzi a WHERE feltételt:

1. A Filmek reláció sorának filmcím attribútuma 'Csillagok háborúja', értékű kell legyen.
2. A Filmek 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, amely minden feltételnek eleget tesz, a GyártásIrányító sor név attribútumértéke belekerül az eredménybe. Mindkét feltétel csak az általunk adatok esetén teljesül, nevezetesen amikor a Filmek közül éppen a *Csillagok háborúja* sort, a GyártásIrányító-k közül pedig George Lucas sorát vizsgáljuk. Ekkor és csak ekkor lesz a filmcím megfelelő és az azonosítók (producerAzon és azonosító) megegyezők. Így George Lucas lesz az egyetlen válasz a lekérdezésre. Ezt a folyamatot a 6.3. ábra mutatja. A 6.2.4. alfejezetben írjuk le részletesebben, hogyan is interpretáljuk a több relációra hivatkozó lekérdezéseket. □

6.2.2. Attribútumok megkülönböztetése

Előfordulhatnak olyan lekérdezések, amelyekben két vagy több reláció szerepel, és ezekben a relációkban két vagy több attribútumnak ugyanaz a neve. Ilyen esetben valamilyen módon jelezniük 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.



6.3. ábra. A 6.12. példa lekérdezése a Filmek reláció minden sorát párosítja a GyártásIrányító összes sorával, és ellenőrzi a két feltétel teljesülését

6.13. példa. Tekintsük a következő két relációt:

FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)

Mindkettőben szerepelnek a név és cím attribútumok. Tételezzük fel, hogy olyan filmszínész-gyártásirányító párosokat keresünk, akiknek azonos a lakcímiük. A megfelelő lekérdezés a következő:

```
SELECT FilmSzínész.név, GyártásIrányító.név
FROM FilmSzínész, GyártásIrányító
WHERE FilmSzínész.cím = GyártásIrányító.cím;
```

Ebben a lekérdezésben olyan FilmSzínész és GyártásIrányító sorpárokat keresünk, melyekben a cím attribútumérték megegyezik. A WHERE záradék fejezi ki a cím attribútumokra vonatkozó feltételt. Mindegyik olyan sorpár esetén, amely megfelel a feltételnek, minden két név attribútumot hozzáadjuk az eredményhez. Így az eredményben olyan névpárok lesznek, mint:

<i>FilmSzínész.név</i>	<i>GyártásIrányító.név</i>
Jane Fonda	Ted Turner
...	...

□

A relációnevet és a pontot olyan esetben is az attribútum elé írhatjuk, amikor nincs névazonosságból származó kétérterműség. Például a 6.12. példa lekérdezését így is írhatnánk:

```
SELECT GyártásIrányító.név
FROM Filmek, GyártásIrányító
WHERE Filmek.filmcím = 'Csillagok háborúja'
      AND Filmek.producerAzon = GyártásIrányító.azonosító;
```

A relációnév és pont kombinációt a lekérdezésben szereplő bármely attribútumra alkalmazhatjuk.

6.2.3. Sorváltozók

A relációnev előtagként történő alkalmazása jó megoldás az attribútumok névütközésének elkerülésére addig, amíg több különböző reláció szerepel a lekérdezésben. Néha azonban olyan lekérdezésre van szükségünk, amely ugyanazon reláció két vagy több sorát kombinálja össze. Az *R* relációt annyiszor sorolhatjuk fel a FROM 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; az AS kulcsszó általában elhagyható.

R attribútumait a SELECT és WHERE záradékokban megkülönböztethetjük egy előtag segítségével, mely a megfelelő sorváltozóból és egy pontból áll. Tehát a sorváltozó az *R* reláció másodneveként szerepel, és bárhol használható az *R* helyett.

6.14. példa. Míg a 6.13. példában olyan filmszínészeket és gyártásirányítókat kerestünk, akiknek azonos a lakcímük, hasonlóképpen kereshetünk olyan színészpárokat is, akiknek megegyezik a lakcímük. 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 (Színész1- és Színész2-beli) FilmSzínész sor cím attribútumának ugyanaz az értéke.

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 $R \text{ AS } R$ helyett. Továbbá, ahogy már láttuk, ha egy attribútum egyértelműen egy relációhoz tartozik, akkor a reláció (sorváltozóként használt) neve elhagyható.

A WHERE záradék második feltétele, $\text{Színész1.név} < \text{Színész2.név}$, 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 összehasonlízási operátorként, akkor az eredmény minden színész házaspárt kétszer tartalmazna, például:

Színész1.név	Színész2.név
Paul Newman	Joanne Woodward
Joanne Woodward	Paul Newman
...	...

□

6.2.4. 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 relációt értelmezhető saját másodneveként, amint azt a „sorváltozók és relációnevek” bekeretezett részben említettük.

³ Ugyanaz a probléma előfordulhat a 6.13. példában is, ha egy személy filmszínész és gyártásirányító is. A problémát hasonlóan tudjuk megoldani, ha feltételként szabjuk, hogy a két név legyen különböző.

Ha több sor változónk van, mindegyikhez rendelhetünk egy-egy ciklust, melyben a változó végigmegy a neki megfelelő reláció összes során. A ciklusok egymásba vannak ágyazva. A sor változók mindenekig 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 a 6.4. ábra érzékelteti.

```

Legyenek a FROM záradékban megadott relációk  $R_1, R_2, \dots, R_n$ ;
FOR  $t_1$  sorra az  $R_1$  relációban DO
    FOR  $t_2$  sorra az  $R_2$  relációban DO
        ...
        FOR  $t_n$  sorra az  $R_n$  relációban DO
            IF a where záradék igaz, amikor az attribútumokban
                 $t_1, t_2, \dots, t_n$  megfelelő értékei találhatóak
            THEN
                 $t_1, t_2, \dots, t_n$ -nek megfelelően kiértékeljük a
                select záradék attribútumait és az értékekből
                alkotott sort az eredményhez adjuk

```

6.4. ábra. Egy egyszerű SQL-lekérdezés kiértékelése

Párhuzamos értékadás

Ebben az esetben nem kell explicit módon beágyazott ciklusokat létrehozni. Ehelyett tekintsük tetszőleges sorrendben vagy párhuzamosan a sor változók összes lehetséges értékadását a megfelelő relációkból. minden egyes értékadásra megvizsgáljuk, hogy a WHERE feltétel igaz-e. minden egyes olyan értékadás, amelyre 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 sor változóiból indulunk ki, és tekintjük a hozzájuk tartozó relációk Descartes-szorzatát. 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 feltételekkel, amelyet alkalma-zunk az elkészített szorzatra. A WHERE záradékbeli minden attribútumtalálat kicséréljük arra a szorzatbeli attribútumra, amelynek megfelel. Végül a SELECT záradék alapján létrehozzuk a kifejezések listáját, a záró (kiterjesztett) vetí-tési művelet számára. Mint ahogyan a WHERE záradéknál is tettük, a SELECT

Az SQL-szemantika egy meglepő következménye

Tételezzük fel, hogy R , S és T unáris (egyoszlopos) relációk, és mindenik egyedüli attribútuma az A . Szeretnénk megkeresni azokat az elemeket, amelyek az R -ben és vagy az S -ben, vagy a T -ben (vagy mindenben) megtalálhatók. 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 várunk, hogy az eredmény $R \cap S$ legyen. Mégis, a 6.2.4. 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 a 6.4. ábrán bemutatott beágyazott ciklus szemantikát használjuk, észrevehető, hogy a T változó ciklus 0 hosszúságú, mivel annak a relációnak, mely fölött a T változik, nincs egy sora sem. Így a legbelől ciklusbeli if utasítás egyszer sem kerül végrehajtásra, tehát az eredmény üres lesz. Hasonlóképpen, ha a sorváltozók lehetséges értékkedései 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.

záradékban minden attribútumot kicserélünk a szorzat neki megfelelő attribútumára.

6.15. példa. Írjuk át a 6.14. példát relációs algebrára. Először is a FROM záradékban két sorváltozó van, mindenik a FilmSzínész relációra vonatkozik. Ezért az algebrai kifejezés így kezdődik:

$\text{FilmSzínész} \times \text{FilmSzínész}$

Az eredményrelációt nyolc attribútuma van, az első négy a FilmSzínész reláció első másolatának név, cím, nem és születésiDátum attribútumainak felel meg, a következő négy pedig a FilmSzínész reláció második másolatának ugyanazon attribútumainak. Az attribútumokat jelölhetnénk a sorváltozóval és egy ponttal – például Színész1.név –, de a tömörseg kedvéért nevezzük át az attribútumokat A_1, A_2, \dots, A_8 -nak. Így A_1 megfelel Színész1.név -nek, A_5 pedig Színész2.név -nek és így tovább.

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:

$$\pi_{A_1, A_5} \left(\sigma_{A_2 = A_6 \text{ AND } A_1 < A_5} \left(\rho_{M(A_1, A_2, A_3, A_4)}(\text{FilmSzínész}) \times \rho_{N(A_5, A_6, A_7, A_8)}(\text{FilmSzínész}) \right) \right)$$

□

6.2.5. Egyesítés, metszet és különbség az SQL-ben

Néha a relációs algebra halmazműveleteit: az egyesítést, a metszetet és a különbséget kell használnunk relációk kombinálására. Az SQL biztosítja a megfelelő operátorokat, amelyeket lekérdezések eredményeire lehet alkalmazni, feltéve, hogy a lekérdezések ugyanolyan attribútumhalmazú (megegyező nevű és típusú attribútumhalmazok) relációkat eredményeznek. A \cup , \cap és – műveleteknek megfelelő kulcsszavak: UNION, INTERSECT és EXCEPT. Ezeket a kulcsszavakat két zárójel közé írt lekérdezés közé kell helyezni.

6.16. 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, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A lekérdezést a 6.5. ábra mutatja be. Az 1–3. sorok egy olyan relációt eredményeznek, melynek sémája (név, cím), és sorai a színésznők nevét és címét tartalmazzák.

- 1) (SELECT név, cím
- 2) FROM FilmSzínész
- 3) WHERE nem = 'N')
- 4) INTERSECT
- 5) (SELECT név, cím
- 6) FROM GyártásIrányító
- 7) WHERE nettóBevétel > 10000000);

6.5. ábra. Színésznők és gazdag gyártásirányítók metszete

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 sémája csak a név és cím attribútumokból áll. Mivel a két relációséma megegyezik, alkalmazhatjuk rá a metszet műveletet, melyet a 4. sor ad meg. □

SQL-lekérdezések olvashatósága

Általában az SQL-lekérdezéseket úgy írjuk, hogy az olyan fontos kulcsszavak, mint a FROM vagy WHERE új sorban kezdődjenek. Ez a stílus sokkal jobban láthatóvá teszi a lekérdezés struktúráját. Egy rövid lekérdezést azonban egy sorba is írhatunk, mint ahogy a 6.17. példában tettük. A lekérdezés tömörsegét megtartva, ez a stílus is jó olvashatóságot biztosít.

6.17. példa. Hasonló módon meghatározhatjuk a két személyhalmaz különbségét. A következő lekérdezés:

```
(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 – amint a 6.3. példában láttuk – átnevezhetjük az attribútumokat, hogy azonos attribútumhalmazokat kapunk.

6.18. példa. Keressük meg azon filmcímeteket és éveket, amelyek a Filmek vagy a SzerepelBenne relációkban megtalálhatók:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
```

Ideális esetben a két relációban megtalálható filmhalmazoknak meg kellene egyezniük. Gyakorlatilag azonban könnyen megtörténhet, hogy van olyan film, amelyhez nincsenek felsorolva a benne szereplő színészek vagy olyan SzerepelBenne sor is létezhet, amelynek megfelelő sor nincs a film relációban⁴. Így a lekérdezés:

```
(SELECT filmcím, év FROM Filmek)
UNION
(SELECT filmCím AS filmcím, filmÉv AS év FROM SzerepelBenne);
```

Az eredményreláció attribútumai: filmcím és év, azokat a filmeket fogja tartalmazni, amelyek legalább az egyik relációban szerepelnek. □

⁴ Léteznek módszerek az ilyen eltérések megakadályozására, lásd a 7.1.1. alfejezetben.

6.2.6. Feladatok

6.2.1. feladat. Az aktuális adatbázist használva:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

adjuk meg SQL-ben a következő lekérdezéseket:

- a) Keressük meg a *Titanic* férfi szereplőit.
- 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?

6.2.2. feladat. A 2.4.1. feladatbeli adatbázis:

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

alapján adjuk meg a megfelelő lekérdezéseket és a 2.4.1. feladat adataival szemléltessük az eredményt.

- a) Keressük meg azon laptopok gyártói és sebességét, amelyeknek legalább harminc gigabajtos 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, amelyek legalább két PC-ben előfordulnak.
- ! e) Keressük meg azon PC-modellpárokat, amelyeknek ugyanakkora a sebessége és a memoriája. Egy párt csak egyszer listázzunk ki, azaz ha (i, j) az eredményben van, akkor (j, i) ne kerüljön bele.
- !! f) Keressük meg azokat a gyártókat, akik legalább két különböző, 3.0-nál nagyobb sebességű számítógépet gyártanak (PC-t vagy laptopot).

6.2.3. feladat. A 2.4.3. 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 guadalcanali 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, amelyeknek csatahajói és cirkálói is vannak.
- ! e) Keressük meg azokat a hajókat, amelyek megsérültek egy csatában, de részt vettek később egy másikban.
- ! f) Keressük meg azokat a csatákat, amelyekben legalább három azonos országbeli hajó vett részt.

! 6.2.4. feladat. A relációs algebra egyik leggyakoribb kifejezése a következő:

$$\pi_L(\sigma_C(R_1 \times R_2 \times \cdots \times R_n))$$

ahol L az attribútumok tetszőleges listája, míg C egy tetszőleges feltétel. Az R_1, R_2, \dots, R_n lista tartalmazhatja ugyanazt a relációt többször is, ilyenkor az attribútumokat átnevezzük. Mutassuk meg, hogyan lehet kifejezni egy ilyen lekérdezést az SQL-ben.

! 6.2.5. feladat. A relációs algebra egy másik tipikus lekérdezése:

$$\pi_L(\sigma_C(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n))$$

Hasonló feltevésekkel élünk, mint a 6.2.4. feladatban azzal a különbséggel, hogy szorzat helyett természetes összekapcsolást használunk. Mutassuk meg, hogyan lehet egy ilyen lekérdezést SQL-ben megfogalmazni.

6.3. Alkérdések

Az SQL-ben egy lekérdezés kiértékelését egy másik lekérdezés különféle módonon elősegítheti. Az olyan lekérdezést, amely egy másik lekérdezés része, alkérdésnek nevezzük. Az alkérdéseknek is lehetnek alkérdéseik, és így tovább a megkívánt mélységgig. Az alkérdések egy példájával már találkoztunk a 6.2.5. alfejezetben, amikor két alkérdés eredményein végrehajtott egyesítéssel, metszetképzéssel vagy különbségképzéssel állítottuk elő a teljes lekérdezés eredményét. Számos egyéb helyzetben is használhatunk alkérdéseket:

1. Ha az alkérdés egy egyszerű konstans eredményt szolgáltat, akkor ezt használhatjuk a WHERE záradékban, más értékekkel való összehasonlításokban.
2. Ha az alkérdés eredménye reláció, akkor a WHERE záradékban ezt különböző módon használhatjuk.
3. Ha az alkérdés eredménye reláció, akkor ezt az eredményrelációt a FROM záradékban – nevet is adva neki – ugyanúgy használhatjuk, mint ahogy bármely más relációt is.

6.3.1. Skalár értéket adó alkérdések

Az olyan atomi értékeket, amelyek egy sor egyik komponenseként előfordulhatnak, *skalárnak* nevezzük. Egy select-from-where kifejezés bármilyen oszlopszámu relációt eredményezhet és bármennyi sor lehet a relációban. Néha viszont csak egy bizonyos attribútum értékeire van szükségünk. Ezenkívül a kulcsok vagy más információk 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.

6.19. példa. Térjünk vissza a 6.12. példához és keressük meg a *Csillagok háborújának* gyártásirányítóját. A következő két relációt kell lekérdeznünk:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

mert csak az első tartalmazza a filmcímre vonatkozó információkat és csak a második a gyártásirányító neveket. Az információk az azonosító számon kapcsolódnak össze. Ezek a számok egyértelműen azonosítják a gyártásirányítókat. A megfelelő lekérdezés:

```
SELECT név
FROM Filmek, GyártásIrányító
WHERE Filmek.filmcím = 'Csillagok háborúja',
      AND producerAzon = azonosító;
```

A WHERE záradékban a filmcím attribútum elő írtuk a reláció nevét és a pontot, mivel az attribútum minden relációban megtalálható.

Másképpen is vizsgálhatjuk ezt a lekérdezést. A Filmek relációból megtudhatjuk a *Csillagok háborúja* című film gyártásirányítójának azonosítóját. A GyártásIrányító relációból az azonosító alapján megtudhatjuk a megfelelő személy nevét. Az azonosító meghatározása egy alkérdésként fogalmazható meg. Az alkérdés eredményét, amely várhatóan egy érték lesz, a „fő” lekérdezésben használhatjuk fel. A lekérdezést a 6.6. ábra szemlélteti.

```
1) SELECT név
2) FROM GyártásIrányító
3) WHERE azonosító =
4)   (SELECT producerAzon
5)     FROM Filmek
6)     WHERE filmcím = 'Csillagok háborúja'
    );
```

6.6. 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, amely várhatóan csak egy sort fog tartalmazni. A sor (12345) alakú lesz, azaz egyetlen oszlop, amelyben az az egész szám található, amely 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 az alábbi:

```
SELECT név
FROM GyártásIrányító
WHERE azonosító = 12345;
```

A lekérdezés eredménye George Lucas kell legyen. □

6.3.2. Relációkat tartalmazó feltételek

Léteznek olyan SQL-operátorok, amelyeket alkalmazhatunk egy R relációra, és az eredmény logikai érték lesz. Az R relációt egy alkérdéssel kell előállítani. Ha például a Foo tárolt táblára akarjuk a most tárgyalandó operátorokat alkalmazni, akkor erre a célra a (SELECT * FROM Foo) alkérdést használjuk. Ugyanez a trükk használható a relációk uniójának, metszetének és különbségének eseteire is.

Az operátorok közül néhánynak – IN, ALL és ANY – először olyan egyszerű használatát mutatjuk be, amelyben használunk még egy s skaláris értéket is, ebben az esetben R egyoszlopos reláció kell legyen. Az operátorok definíciói:

1. **EXISTS** R egy olyan feltétel, amely akkor és csak akkor igaz, ha R nem üres.
2. $s \text{ IN } R$ akkor és csak akkor igaz, ha s egyenlő valamelyik R -beli értékkel. Hasonlóképpen, $s \text{ NOT IN } R$ akkor és csak akkor igaz, ha s egyetlen R -beli értékkel sem egyenlő. Itt feltételeztük, hogy R egyoszlopos reláció. A 6.3.3. alfejezetben vizsgálni fogjuk az **IN** és **NOT IN** operátorok kiterjesztését, amikor R -nek több attribútuma van és s egy sor.
3. $s > \text{ALL } R$ akkor és csak akkor igaz, ha s nagyobb mint az R egyoszlopos reláció minden értéke. Hasonlóképpen, a $>$ operátort analóg módon kicsérélhetjük bármelyikkel a többi öt összehasonlítási operátor közül. Például, $s < \text{ALL } R$ ugyanazt eredményezi, mint az $s \text{ NOT IN } R$.
4. $s > \text{ANY } R$ 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, $s = \text{ANY } R$ ugyanaz mint az $s \text{ IN } R$.

Hasonlóan más logikai kifejezésekhez, az **EXISTS**, **ALL** és **ANY** operátorokat tagadhatjuk, ha a **NOT** kulcsszót az egész kifejezés elő helyezzük. Így a **NOT EXISTS** R akkor és csak akkor igaz, ha R üres. **NOT** $s > \text{ALL } R$ akkor és csak akkor igaz, ha s nem nagyobb minden R -beli értéknél, míg a **NOT** $s > \text{ANY } R$ akkor és csak akkor igaz, ha s nem nagyobb egyetlen R -beli értéknél sem. Hamarosan példákon keresztül fogjuk bemutatni az operátorok használatát.

6.3.3. Sorokat tartalmazó feltételek

Az SQL-ben egy sor: skalárértékek zárójelek közötti felsorolása. Ilyen például az (123, 'labda') és a (név, cím, nettóBevétele). Az első példában a sor komponensei konstansok, a másodikban attribútumok. Megengedett a konstansok és attribútumok kombinálása.

Ha egy t sornak és R relációjának ugyanannyi komponense van, akkor a 6.3.2. alfejezetben leírt kifejezésekben hasonlíthatjuk össze azokat. Ilyen lehet például a $t \text{ IN } R$ vagy a $t < \text{ANY } R$. A második kifejezés azt jelenti, hogy létezik az R -ben t -től különböző sor. Jegyezzük meg, hogy amikor egy sort összehasonlitunk egy R reláció soraival, a komponenseket az attribútumok R -beli sorrendjének megfelelően kell összehasonlítani.

6.20. példa. A 6.7. ábrán látható SQL-lekérdezés a következő három relációra vonatkozik:

```
Filmek(filmcím, év, hossz, műfaj, stúdiónév, producerAzon)
SzerepelBenne(filmcím, filmÉv, színészNév)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

```

1) SELECT név
2) FROM GyártásIrányító
3) WHERE azonosító IN
4)     (SELECT producerAzon
5)         FROM Filmek
6)         WHERE (filmcím, év) In
7)             (SELECT filmCím, filmÉv
8)                 FROM SzerepelBenne
9)                 WHERE színész = 'Harrison Ford'
)
);

```

6.7. ábra. Így találjuk meg Harrison Ford filmjeinek gyártásirányítóit

A lekérdezés azon filmek gyártásirányítóit keresi, amelyekben Harrison Ford szerepelt. Egy „fő” lekérdezésből áll, azon belül két beágyazott lekérdezésből.

A beágyazott lekérdezéseket belülről kifelé értékeljük ki. Indulunk ki a legbelő beágyazott lekérdezésből, amely a 7–9. sorok között található. A lekérdezés a SzerepelBenne reláció sorait vizsgálja és azokat találja meg, amelyekben a színészNév komponens értéke 'Harrison Ford'. Ezeknek a filmeknek a címét és gyártási évét tartalmazza az eredmény. Emlékezzünk vissza, hogy a filmcím és az év együtt alkotja a Filmek kulcsát, amely alapján pontosan azonosítani lehet. Így a 7–9. sorok közötti lekérdezés eredménye a 6.8. ábrán bemutatotthoz hasonló sorokat fog tartalmazni.

filmcím	év
Csillagok háborúja	1977
Az elveszett frigyláda fosztogatói	1981
A szökevény	1993
...	...

6.8. ábra. Filmcím-év párok, amelyeket a legbelő alkérdés eredményez

Tekintsük a 4–6. sorok közötti középső alkérdést. Olyan Filmek sorokat keres, melyek címe és gyártási éve a 6.8. ábrán szemléltetett relációban található. 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 nézzük meg az 1–3. sorok közötti „fő” lekérdezést. Ez megvizsgálja a GyártásIrányító reláció sorait, hogy megtalálja azokat, amelyek azonosító komponense a középső lekérdezés által eredményezett halmazban van.

Minden egyes megfelelő sor esetén a gyártásirányító neve az eredménybe kerül, amely így azon filmek gyártásirányítói fogja tartalmazni, amelyekben Harrison Ford szerepelt. □

A 6.7. 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, amely a 6.7. ábra fő lekérdezésében vagy valamelyik alkérdésben megjelenik. Az IN kapcsolat helyett egyenlőségek lesznek a WHERE záradékban. Így a 6.9. ábrán látható lekérdezés lényegében megegyezik a 6.7. ábra lekérdezésével. Különbségek csak a gyártásirányítók ismétlődésének kezelésében vannak, ahogyan a 6.4.1. alfejezetben látni fogjuk.

```
SELECT név
FROM GyártásIrányító, Filmek, SzerepelBenne
WHERE azonosító = producerAzon AND
      Filmek.filmcím = SzerepelBenne.filmCím AND
      Filmek.év = SzerepelBenne.filmÉv AND
      színészNév = 'Harrison Ford';
```

6.9. ábra. Ford gyártásirányítói beágyazott lekérdezések nélkül

6.3.4. Korrelált alkérdések

A legegyszerűbb alkérdések 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, amely az alkérdésen kívüli sorváltozóból származik. Egy ilyen típusú alkérdést *korrelált alkérdésnek* nevezünk. Kezdjük egy példával:

6.21. példa. Keressük meg azokat a filmcímeket, amelyek két vagy több filmhez is tartoznak. Kezdjük a megoldást egy külső lekérdezéssel, amely a következő reláció minden sorát megvizsgálja:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

Minden egyes sor esetén egy alkérdésen keresztül megvizsgáljuk, hogy létezik-e egy ugyanolyan című film, amelyet később gyártottak (feltessük, hogy azonos című filmek nem készültek ugyanabban az évben). Az egész lekérdezést a 6.10. ábra szemlélteti.

Mint a többi beágyazott lekérdezés esetén is, kezdjük a vizsgálatot a 4–6. sorok közötti belső lekérdezéssel. Ha a Régi.filmcím 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, mi a Régi.filmcím értéke. Amikor azonban az 1–3. sorok közötti külső lekérdezés folyamán a Filmek sorokon haladunk végig, a Régi.filmcím-nek mindenkor van valamilyen értéke. Ezt az értéket használva végrehajtjuk a 4–6. sorok közötti lekérdezést azért, hogy a 3–6. sorok közötti WHERE feltétel igazságéértékét meghatározzuk.

```

1) SELECT filmcím
2) FROM Filmek Régi
3) WHERE év < ANY
4)   (SELECT év
5)     FROM Filmek
6)   WHERE filmcím = Régi.filmcím
    );

```

6.10. Ábra. A több mint egyszer előforduló filmcímek megkeresése

A 3. sorbeli feltétel akkor lesz igaz, ha létezik olyan film, amelynek a címe megegyezik a Régi.filmcím értékével és később gyártották, mint a Régi sorváltozó aktuális értékéhez tartozó filmet. Ez a feltétel igaz lesz mindenkor, amíg a Régi sorváltozó év értéke nem az utolsó év, amelyben egy olyan című filmet gyártottak. Tehát az 1–3. sorok közötti lekérdezés egyetlen kevesebb sor fog kiírni egy filmcímet, mint ahányszor ilyen című filmet gyártottak. Egy kétszer gyártott filmet egyszer fog az eredmény tartalmazni, egy háromszor gyártottat kétszer stb.⁵ □

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, akkor megvizsgáljuk az őt körbevevő lekérdezést, azután az azt körbevevőt stb. Így a 6.10. ábra 4. sorában szereplő év és a 6. sorában szereplő cím annak a sorváltozonak az attribútumai, amely az 5. sorban bevezetett Filmek reláció sorain megy végig – azaz annak a Filmek relációjának 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 elő. Ezért vezettük be a külső lekérdezésben a Filmek reláció Régi másodnevét, és ezért hivatkoztunk a 6. sorban a Régi.filmcím 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 Ez a példa az első olyan alkalom, amikor fel kell idéznünk, hogy az SQL relációi nem halmazok, hanem multihalmazok. Az ismétlődések több módon is eltávolíthatók, ahogy azt a 6.4. alfejezetben vizsgálni fogjuk.

6.3.5. Alkérdések a FROM záradékban

Az alkérdéseket – pontosabban az alkérdés által létrehozott relációt – a FROM záradékban is használhatjuk. A FROM záradék listájában a tényleges reláció(k) helyett előfordulhatnak alkérdések is. Ilyenkor ezeket zárójelek között adhatjuk meg. Minthogy az alkérdés eredményrelációjának nincs neve, másodnévadás lehetőségével elve nevet kell adnunk neki. Így az alkérdés által létrehozott reláció soraira már ugyanúgy tudunk hivatkozni, mint a FROM lista többi relációinak soraira.

6.22. példa. Vizsgáljuk meg újra a 6.20. példabeli feladatot, melyben Harrison Ford filmjeinek gyártásirányítót kerestük. Tegyük fel, hogy van olyan relációink, amely tartalmazza ezen filmek gyártásirányítóinak producerAzon-jait. Ezután már egyszerű dolog megtalálni a gyártásirányítók neveit a GyártásIrányító relációban. A 6.11. ábrán ezt a lekérdezést látjuk.

```

1) SELECT név
2) FROM GyártásIrányító,
           (SELECT producerAzon
3)          FROM Filmek, SzerepelBenne
4)          WHERE Filmek.filmcím = SzerepelBenne.filmCím AND
5)                  Filmek.év = SzerepelBenne.filmÉv AND
6)                  színészNév = 'Harrison Ford'
7)          ) Prod
8) WHERE azonosító = Prod.producerAzon ;

```

6.11. ábra. Ford gyártásirányítóinak keresése, alkérdést használva
a FROM záradékban

A 2–7. sorokban látjuk a FROM záradékot, amelyben a GyártásIrányító-n kívül, a beágyazott alkérdéssel létrehozott, Prod másodnévvel ellátott reláció is szerepel. Ez a beágyazott alkérdés a 3–5. sorokban a Filmek és a SzerepelBenne összekapcsoltjából kiválogatja azon filmek producerAzon-jait (2. sor), amelyekben szerepel Harrison Ford (6. sor). A 7. sorban az alkérdéssel előállított halmaz másodneveként Prod-ot adtunk.

A 8. sorban a GyártásIrányító és az alkérdéssel létrehozott Prod relációkra vonatkozó válogatási feltételt – az azonosítók egyenlősége – használjuk. A GyártásIrányító előbbi válogatási feltételnek megfelelő soraiból a nevek adják a lekérdezés végeredményét (1. sor). □

6.3.6. Összekapcsolások az SQL-ben

Két relációra az összekapcsolás-művelet különféle változatait alkalmazva számos új relációt tudunk előállítani. A változatok közé tartozik a Descartes-szorzat, a természetes összekapcsolás, a théta-összekapcsolás és a külső összekapcsolások. Az összekapcsolás eredménye lehet, hogy maga a válasz az adott lekérdezésre,

de – minthogy az összekapcsolás eredménye is reláció – az összekapcsoló kifejezések a select-from-where lekérdezés FROM záradékában előforduló alkérdezések is tekinthetők. Az ilyen kifejezések leginkább a jóval összetettebb select-from-where kifejezéseket rövidítik (lásd 6.3.11. feladat).

Az összekapcsolási kifejezések legegyszerűbb formája a *keresztszorozat*; ez a kifejezés rokon értelmű a Descartes-szorzat vagy csak „szorzat” kifejezéssel, amelyet a 2.4.7. alfejezetben használtunk. Például, ha a következő két reláció szorzatát szeretnénk megkapni:

```
Filmek(filmcím, év, hossz, műfaj, stúdiónév, producerazon)
SzerepelBenne(filmcím, filmév, színésznév)
```

a megfelelő utasítás:

```
Filmek CROSS JOIN SzerepelBenne;
```

és az eredmény egy kilencoszlopos reláció lesz, mégpedig a Filmek és a SzerepelBenne relációk attribútumaival. minden egyes Filmek sort párosítunk minden egyes SzerepelBenne sorral, amelyek belekerülnek az eredményrelációba.

A szorzatreláció 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 szorzatművelet használata önmagában azonban csak ritkán indokolt. Sokkal megfelelőbb a théta-összekapcsolás, amelyet az ON kulcsszóval lehet elérni. Az R és S relációnevek közé tesszük a JOIN kulcsszót, utánuk pedig az ON kulcsszót és egy feltételt. A JOIN...ON jelentése, hogy az $R \times S$ szorzatot az ON utáni feltétel szerinti kiválasztás követi.

6.23. példa. Tételezzük fel, hogy a következő két relációt szeretnénk összekapcsolni:

```
Filmek(filmcím, év, hossz, műfaj, stúdiónév, producerazon)
SzerepelBenne(filmcím, filmév, színésznév)
```

azzal a feltétellel, hogy csak az ugyanarra a filmre vonatkozó sorok legyenek összekapcsolva. Azaz a cím és a gyártási év minden sorban meg kell egyezzen. Ezt az eredményt a következő lekérdezéssel érhetjük el:

```
Filmek JOIN SzerepelBenne ON
    filmcím = filmCím AND Filmek.év = SzerepelBenne.filmév;
```

Az eredmény ismét egy kilencoszlopos reláció a nyilvánvaló attribútumnevekkel. Most azonban egy Filmek 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 `filmcím` és a `filmCím`, illetve a `Filmek.év` és a `SzerepelBenne.filmÉv` attribútumok értéke minden sorban ugyanaz lesz.

Ha zavar bennünket, hogy az előbbi példa eredményében két felesleges oszlop van, az egész kifejezést behelyezhetjük a `FROM` záradékba, és a megfelelő `SELECT` záradék segítségével eltüntethetjük a felesleges oszlopokat. Így a következő utasítás:

```
SELECT filmcím, Filmek.év, hossz, műfaj, stúdióNév,
       producerAzon, színészNév
  FROM Filmek JOIN SzerepelBenne ON
    filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

egy hétoszlopos relációt eredményez, mely a `Filmek` reláció sorait tartalmazza, kibővítve az összes lehetséges módon a filmben szereplő színészekkel. □

6.3.7. Természetes összekapcsolás

A 2.4.8. alfejezetben ismertetük, 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 SQL természetes összekapcsolása pontosan így működik. A `NATURAL JOIN` kulcsszavak jelennek meg az összekapcsolandó relációk között a \bowtie művelet kifejezésére.

6.24. példa. Tételezzük fel, hogy a következő két reláció természetes összekapcsolását szeretnénk meghatározni:

```
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Az eredmény egy olyan reláció lesz, amely tartalmazza a `név` és a `cím` attribútumokat és az összes olyan attribútumot, amely 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 azonosító számot és a nettó bevételt. Ezt a relációt a következő kifejezés írja le:

```
FilmSzínész NATURAL JOIN GyártásIrányító;
```

□

6.3.8. Külső összekapcsolások

A külső összekapcsolást, amely eredménye tartalmazza a nullértékekkel feltöltött lógó sorokat is, az 5.2.7. alfejezetben tárgyaltuk. Az SQL-ben használhatunk külső összekapcsolást; a nullérték jelölésére a NULL szolgál.

6.25. példa. Tételezzük fel, hogy a következő két reláció külső összekapcsolását szeretnénk meghatározni:

```
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Az SQL a szabványos külső összekapcsolást, amely a minden kapcsolandó táblából származó nem kapcsolható sorokat is tartalmazza, *teljes* külső összekapcsolásnak (full outerjoin) nevezi. A szintaxis:

```
FilmSzínész NATURAL FULL OUTER JOIN GyártásIrányító;
```

Az eredménye egy ugyanolyan hatoszlopos reláció, mint a 6.24. példa eredménye. A relációnak háromféle sora van. Léteznek olyan sorok, amelyek azokat a színészeket tartalmazzák, akik egyben gyártásirányítók is. Ezeknek a soroknak minden attribútuma kitölthető értéket tartalmaz. Ezek a sorok megtalálhatók a 6.24. példa eredményében is.

A második típusú sorok azokat a színészeket tartalmazzák, akik nem gyártásirányítók. Ezekben a sorokban a név, cím, nem és születésiDátum attribútumok értékét a FilmSzínész reláció alapján töltjük ki, míg a kizárolag 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 születésiDátum attribútumok, amelyek csak a FilmSzínész relációhoz tartoznak, NULL értéket tartalmaznak. A 6.12. ábrán látható három sor például megfelel a három sortípusnak. □

név	cím	nem	születésiDátum	azonosító	nettóBevétel
Mary Tyler Moore	Maple St.	'N'	9/9/99	12345	100... \$
Tom Hanks	Cherry Ln.	'F'	8/8/88	NULL	NULL
George Lucas	Oak Rd.	NULL	NULL	23456	200... \$

6.12. ábra. Három sor a FilmSzínész és GyártásIrányító külső összekapcsolásából

Az 5.2.7. alfejezetben a külső összekapcsolások összes fajtát tárgyaltuk, ezek minden elérhetők az SQL-ben is. Ha bal vagy jobb oldali külső összekapcsolást kívánunk használni, akkor a megfelelő LEFT (bal) vagy RIGHT (jobb) szót kell használnunk a FULL helyett. Például a

FilmSzínész NATURAL LEFT OUTER JOIN GyártásIrányító;

összekapcsolás tartalmazza a 6.12. ábra első két sorát, de a harmadikat nem. Hasonlóképpen a

FilmSzínész NATURAL RIGHT OUTER JOIN GyártásIrányító;

tartalmazza a 6.12. ábra első és harmadik sorát, de a másodikat nem.

Tegyük most fel, hogy a természetes összekapcsolás helyett a külső théta-összekapcsolást (feltételes külső összekapcsolást) kívánjuk használni. Ekkor a NATURAL kulcsszó helyett az összekapcsolás után írhatjuk az ON kulcsszót, majd azt a feltételt, amelyet a sorok ki kell hogy elégítsenek. Ha megadjuk a FULL OUTER JOIN kulcsszavakat is, akkor az összpárosított sorokon kívül azok a sorok is bekerülnek az eredménybe, amelyek a másik relációból egy sorral sem kapcsolódtak össze, természetesen a hiányzó attribútumokban NULL értékkel.

6.26. példa. Tekintsük ismét a 6.23. példát, ahol a Filmek és a SzerepelBenne relációkat kapcsoltuk össze azzal a feltétellel, hogy a filmcím és filmCím, illetve az év és filmÉv attribútumok a két relációban megegyezzenek. Ha módosítjuk a feltételt egy külső összekapcsolás segítségével:

```
Filmek FULL OUTER JOIN SzerepelBenne ON
    filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

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.filmÉv és a színészNév attribútumokban NULL értékkel. Hasonlóképpen megkapjuk azokat a színészeket is, akik nem szerepelnek olyan filmben, amely a Filmek relációban megtalálható, és a Filmek reláció hat attribútuma ezekben a sorokban NULL értéket fog tartalmazni. □

A 6.26. példában bemutatott külső összekapcsolásokban a FULL kulcsszó kicserélhető a LEFT vagy a RIGHT kulcsszavakra. Például a

```
Filmek LEFT OUTER JOIN SzerepelBenne ON
    filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

azokat a filmeket eredményezi, amelyeknek legalább egy felsorolt színészük van és azokat, amelyeknek nincs egy sem, de azok a színészek, akik egy nem említett filmben szerepelnek, nem lesznek az eredményben. Hasonlóképpen a

```
Filmek RIGHT OUTER JOIN SzerepelBenne ON
    filmcím = filmCím AND Filmek.év = SzerepelBenne.filmÉv;
```

utasítás kihagyja azokat a filmeket, amelyekhez nem tartozik egy színész sem, de tekintetbe veszi azokat a színészeket is (a filmekből származó attribútumokat NULL értékkel feltöltve), akik a vizsgált filmek egyikében sem szerepelnek.

6.3.9. Feladatok

6.3.1. feladat. Az 2.4.1. feladat adatbázissémáját használva adjuk meg a következő lekérdezéseket:

Termék(gyártó, modell, típus)

PC(modell, sebesség, memória, merevlemez, ár)

Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)

Nyomtató(modell, színes, típus, ár)

A válaszokban használjunk legalább egy alkérdést, és minden lekérdezésre adjunk két lényegesen különböző megoldást (azaz használjunk különböző operátorokat az EXISTS, IN, ALL és ANY közül).

- a) Keressük meg a legalább 3.0 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 az olyan PC-k gyártóit, amelyek a leggyorsabbak a legkevesebb memoriával rendelkező PC-k között.

6.3.2. feladat. Adjuk meg a következő lekérdezéseket a 2.4.3. 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, amelyekből legalább egy hajót elsüllyesztettek egy csatában.
- c) Keressük meg azoknak a hajóknak a neveit, melyek ágyúinak kalibere 16 hüvelyk.

- d) Keressük meg azokat a csatákat, amelyekben a Kongó hajóosztályba tartozó hajók vettek részt.
- !! e) Keressük meg azoknak a hajóknak a neveit, melyeknek az ágyúszáma a legnagyobb a velük megegyező kaliberű ágyúkat tartalmazó hajók között.

! 6.3.3. feladat. Adjuk meg a 6.10. ábra lekérdezését alkérdések nélkül.

! 6.3.4. feladat. Tekintsük a $\pi_L(R_1 \bowtie R_2 \bowtie \dots \bowtie R_n)$ relációs algebrai kifejezést, ahol L csak az R_1 attribútumait tartalmazza. Mutassuk meg, hogy ezt a lekérdezést csak alkérdések felhasználásával át lehet írni SQL-be. Pontosabban, adjunk meg egy olyan SQL-kifejezést, amelyben minden egyik FROM záradékban csak egy sorváltozó található.

! 6.3.5. feladat. Adjuk meg a következő lekérdezéseket metszet és különbség használata nélkül:

- a) A 6.5. ábra lekérdezését.
- b) A 6.17. példa lekérdezését.

!! 6.3.6. feladat. Észrevehetünk, hogy néhány SQL-operátor felesleges abban az értelemben, hogy más operátorokkal helyettesíthető. Például láthattuk, hogy $s \text{ IN } R$ helyettesíthető az $s = \text{ ANY } R$ kifejezéssel. Mutassuk meg, hogy az EXISTS és NOT EXISTS operátorok feleslegesek, azaz olyan kifejezéssel helyettesíthetőek, amelyben nincs EXISTS. Tanács: A SELECT záradékban konstans is használható.

6.3.7. feladat. Az aktuális adatbázisunk alábbi relációira nézve:

```
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
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 NATURAL FULL OUTER JOIN FilmSzínész;
- c) SzerepelBenne FULL OUTER JOIN FilmSzínész ON név = színészNév;

! 6.3.8. feladat. Felhasználva a következő adatbázissémát, adjunk meg egy SQL-lekérdezést, amely 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).

`Termék(gyártó, modell, típus)`

`PC(modell, sebesség, memória, merevlemez, ár)`

`Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)`

`Nyomtató(modell, színes, típus, ár)`

6.3.9. feladat. A 2.4.3. feladatban szereplő adatbázis két relációját használva adjunk meg egy SQL-lekérdezést, amely 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.

`Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
kaliber, vízkiszorítás)`
`Hajók(név, osztály, felavatva)`

! 6.3.10. feladat. Ismételjük meg a 6.3.9. feladatot, de vegyük az eredménybe azokat a hajókat is, amelyeknek a neve megegyezik a *C* osztály nevével, azokra a *C* osztályokra is, amelyekhez nem tartozik hajó a Hajók relációban. Feltehetjük, hogy létezik hajó az osztály nevével akkor is, ha nincs a Hajók relációban.

! 6.3.11. feladat. Az ebben a fejezetben bemutatott összekapcsolási műveletek (a külső összekapcsolást kivéve) 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.

6.4. Relációkra vonatkozó műveletek

Ebben az alfejezetben olyan műveletekkel foglalkozunk, melyek a relációk egészére vonatkoznak, nem pedig a relációk egyes (vagy néhány) soraira. Figyelembe kell vennünk azt a tényt, hogy az SQL a relációkat nem tekinti halmaznak, azaz ugyanaz a sor többször is előfordulhat. A 6.4.1. alfejezetben bemutatjuk, hogyan lehet az ismétlődéseket megszüntetni, míg a 6.4.2. alfejezetben ismertetjük, hogyan lehet az ismétlődések megőrzését biztosítani olyan esetekben, amikor az SQL-rendszer különben megszüntetné az ismétlődéseket.

Tárgyaljuk azt is, hogy az 5.2.4. alfejezetben bevezetett γ csoportképző és összesítő operátor funkcióit hogyan valósítja meg az SQL. Az SQL-nek vanak összesítő műveletei, és GROUP-BY csoportképző záradéka. Az SQL-ben van „HAVING” záradék is, amely nem egyedi sorokra, hanem a csoportokra vonatkozó válogatást teszi lehetővé.

6.4.1. Ismétlődések megszüntetése

Ahogyan azt a 6.3.4. alfejezetben is említettük, az SQL relációi különböznek a 2.2. alfejezetben absztrakt módon definiált relációtól. A reláció, halmazként tekintve, nem tartalmazhatja ugyanazt a sort többször. Amikor azonban az SQL-lekérdezés eredményeként létrejön egy reláció, akkor az SQL nem szünteti meg automatikusan az ismétlődéseket. Így az SQL egy lekérdezésre adott válasza tartalmazhatja ugyanazt a sort többször is.

Elevenítsük fel a 6.2.4. alfejezetben említetteket: az SQL select-from-where műveletének egyik értelmezése szerint a **FROM** záradékban szereplő relációk Descartes-szorzatát számoljuk ki először. A szorzat minden egyes sorára 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 kapunk 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. Ez a lekérdezés eredményére alkalmazott – az 5.2.1. alfejezetben bemutatott – δ művelet SQL-beli megfelelője.

6.27. példa. Tekintsük a 6.9. ábra lekérdezését, melyben a Harrison Ford-filmek gyártásirányítóit kerestük, alkérdezések nélkül. A lekérdezés jelenlegi formájában George Lucas többször is szerepelni fog az eredményben, mivel több olyan filmet is gyártott, amelyben 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 kitöröklik.

Valószínűleg a 6.7. ábra lekérdezése, amelyben alkérdezést használtunk, 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. □

6.4.2. Ismétlődések kezelése halmazműveletek során

A **SELECT** utasítással ellentétben, amely csak akkor szünteti meg az ismétlődéseket, ha jelen van a **DISTINCT** kulcsszó, a 6.2.5. alfejezetben bevezetett egyesítés,

Az ismétlődések megszüntetésének költsége

Hajlamosak lennének minden **SELECT** után a **DISTINCT** kulcsszót írni, mivel hiba nem keletkezhet belőle. Valójában azonban nagyon költséges az ismétlődések megszüntetése egy relációban. A relációt sorba kell rendezni vagy szét kell osztani, 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. Ha azt kívánjuk, hogy a lekérdezések gyorsak legyenek, akkor az ismétlődések megszüntetését csak indokolt esetben használjuk.

metszet és különbség operátorok normális esetben megszüntetik az ismétlődéseket. Tehát a multihalmazok halmazzá konvertálódnak és a halmazműveletek halmaz jellegű változatait alkalmazzák. Azért, hogy az ismétlődések megmaradjanak, az **UNION**, **EXCEPT** vagy **INTERSECT** kulcsszavak után az **ALL** kulcsszót kell írni. Ebben az esetben ezek a műveletek az 5.1.2. alfejezetben tárgyalt multihalmaz-szemantika szerint fognak működni.

6.28. példa. Tekintsük ismét a 6.18. példa lekérdezését, de adjuk hozzá az **ALL** kulcsszót:

```
(SELECT filmcím, év FROM Filmek)
    UNION ALL
    (SELECT filmCím AS filmcím, filmÉv AS év FROM SzerepelBenne);
```

Így minden egyes filmcím és gyártási év annyiszor fog szerepelni, ahányszor a **Filmek** és a **SzerepelBenne** relációkban összesen megjelenik. Például, ha egy film egyszer található meg a **Filmek** relációban és három színész tartozik hozzá a **SzerepelBenne** relációban, akkor az egyesítés eredményében a film négyeszer 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 az

$$R \text{ INTERSECT ALL } S$$

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

$$R \text{ EXCEPT ALL } S$$

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 az 5.1.2. alfejezetben tárgyalt definíciókkal.

6.4.3. Csoportosítás és összesítések az SQL-ben

Az 5.2.4. alfejezetben tárgyaltuk a kiterjesztett relációs algebra γ csoportosító és összesítő műveletét. Idézzük fel, hogy – amint az 5.2.3. alfejezetben láttuk –, ez a művelet a sorok egy vagy több attribútumának értéke szerint lehetővé teszi a reláció sorainak csoportosítását. Lehetőségünk van a reláció oszlopaira összesítő műveletek alkalmazásával különböző összesítések előállítására. Ha csoportokat képeztünk, akkor az összesítések csoportonként működnek. Az SQL a **SELECT** záradék listájában megengedi összesítő függvények használatát, a csoportosítást pedig a **GROUP BY** záradékkal oldja meg, így biztosítja a γ művelet összes lehetőségét.

6.4.4. Összesítő függvények

Az SQL öt összesítő függvényével az 5.2.2. alfejezetben találkoztunk, ezek a **SUM**, **AVG**, **MIN**, **MAX** és **COUNT**. Ezek tipikusan a **SELECT** záradékban egy-egy oszlopra alkalmazva skalár értékeket szolgáltatnak. Kivétel a **COUNT(*)** kifejezés, amely a **FROM**-ban megadott reláció(k)ból a **WHERE**-ben leírt feltétel(ek)nek megfelelően megkonstruált összes sor számát adja meg.

Az eddigiekben felül a **DISTINCT** kulcsszó alkalmazásával lehetőségünk van arra, hogy az összesítések előállítása előtt az érintett oszloból a duplikációtumokat kihagyjuk. Tehát olyan kifejezéseket használhatunk, mint például a **COUNT(DISTINCT x)**, amely az x oszloban található különböző értékek számát adja meg. A **COUNT** helyett bármelyik függvényt használhatjuk, de például a **SUM(DISTINCT x)** csak ritkán ad hasznos választ, ez ugyanis például az x oszloban található különböző értékek összegét adja meg.

6.29. példa. A következő lekérdezés megadja az összes gyártásirányító átlagos nettó bevételét:

```
SELECT AVG(nettoBevétel)
  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, nettoBevétel)
```

Összeadja az oszlopan található értékeket, mindenkor 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. □

6.30. példa. A következő lekérdezés a SzerepelBenne reláció sorainak számát határozza meg:

```
SELECT COUNT(*)
FROM SzerepelBenne;
```

Az előbbihez hasonló lekérdezés:

```
SELECT COUNT(SzínészNév)
FROM SzerepelBenne;
```

a reláció SzínészNév oszlopában előforduló összes név számát adja meg. Mivel a duplikátumokat nem zártuk ki, így a két fenti kérdés válasza egymással meggyezik.

Ha a duplikátumokat nem óhajtjuk többszörösen figyelembe venni, akkor az összegzett attribútum neve előtt a kérdésben a DISTINCT kulcsszót is használnunk kell, tehát:

```
SELECT COUNT(DISTINCT SzínészNév)
FROM SzerepelBenne;
```

Így minden színészt csak egyszer veszünk figyelembe, függetlenül attól, hogy hány filmben szerepelt. □

6.4.5. Csoportosítás

A sorok csoportosítását a WHERE záradékot követő GROUP BY záradékban tudjuk megadni. A GROUP BY kulcsszót a *csoportosító* attribútumok listája követi. A legegyszerűbb esetben a FROM záradék csak egy sor változót tartalmaz, és a reláció sorait csoportosítjuk a csoportosító attribútumoknak megfelelően. A SELECT záradékban szereplő összesítési operátorokat a csoportokra kell alkalmazni.

6.31. példa. A Filmek relációból:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

szeretnénk megtudni stúdiónként a gyártott filmek hosszának percben mért összegét. A választ a következő lekérdezés szolgáltatja:

```
SELECT stúdióNév, SUM(hossz)
FROM Filmek
GROUP BY stúdióNév;
```

A lekérdezés értelmezése úgy képzelhető el, hogy a Filmek reláció sorait az 5.4. ábrán érzékelhetettem módon átszervezzük és csoportosítjuk úgy, hogy az összes sor, amely a Disney stúdióra vonatkozik, egymás mellé kerüljön, ugyanígy egymás mellé kerülnek 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 a 6.31. példában a SELECT záradékban kétféle elem található:

1. Összesítések, amelyekben egy összesítési operátort alkalmazunk egy attribútumra vagy egy attribútumot tartalmazó kifejezésre. Ezek a kifejezések csoportonként kerülnek kiértékelésre.
2. Attribútumok, amelyek a GROUP BY záradékban szerepelnek, mint a példában stúdióNév. Egy összesítést 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, amelyben valamelyik ezek közül hiányzik. Például a következő lekérdezés is helyes:

```
SELECT stúdióNév
FROM Filmek
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 Filmek;
```

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.

6.32. példa. Tételezzük fel, hogy szeretnénk meghatározni mindegyik gyártás-irányító által gyártott filmek összhosszát. A következő két relációra van szükségünk:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Tekintsük a théta-összekapcsolásukat, egyenlővé téve az azonosító számokat. Ez a lépés egy olyan relációt hoz létre, amelyben minden GyártásIrányító sort összpárosítja az összes olyan Filmek sorral, amelyet az a gyártásirányító készített. Megjegyezzük, hogy az olyan gyártásirányító, aki nem producer, nem lesz párosítva egyetlen filmmel sem, következésképpen nem fordul majd elő a relációban. Ezután a relációt csoportosíthatjuk a gyártásirányító azonosító száma szerint, és minden csoportban meghatározzuk a filmek hosszainak összegét. A lekérdezést a 6.13. ábra szemlélteti. □

```
SELECT név, SUM(hossz)
FROM GyártásIrányító, Filmek
WHERE producerAzon = azonosító
GROUP BY név;
```

6.13. ábra. Gyártásirányítónként a filmek hosszainak összege

6.4.6. Csoportosítás, összegzés és nullértékek

Ha a sorban nullérték is előfordul, akkor van néhány szabály, amire gondolnunk kell:

- A NULL értéket bármely összesítés során figyelmen kívül hagyjuk. Nem vesz részt az oszlopra vonatkozó összeg, az átlag és a darabszám képzésében, és nem befolyásolja a maximum- és minimumértékeket. Például a COUNT(*) a relációban a sorokat számlálja össze, de a COUNT(A) csak az olyan sorokat veszi figyelembe, melyekben az A attribútum nem NULL értékű.
- Más szempontból, ha egy csoportosító attribútum értéke NULL, akkor közönséges értéknek tekintjük. Ez azt jelenti, hogy a csoportosító attribútum NULL értékeihez is létrejön az ilyen sorokból álló csoport.
- Az összesítő függvények az üres csoportokra NULL értéket adnak eredményül, kivéve a COUNT, mely üres csoportra 0 eredmény ad.

6.33. példa. Tegyük fel, hogy az $R(A, B)$ relációnak egyetlen sora van, és a sor minden komponense nullértékű, azaz:

A		B
NULL		NULL

Ekkor a:

```
SELECT A, COUNT(B)
FROM R
GROUP BY A;
```

A záradékok sorrendje az SQL-lekérdezésekben

Most már megismertük mind a hat záradékot, amelyek az SQL „select-from-where” lekérdezésében szerepelhetnek: SELECT, FROM, WHERE, GROUP BY, HAVING és ORDER BY. Csak a SELECT és FROM használata kötelező. Ha a többi közül bármelyik szerepel, akkor a fenti sorrendet be kell tartani.

lekérdezés eredménye a $(\text{NULL}, 0)$ sor lesz. Azért ez az eredmény, mert amikor A szerint csoportosítottunk, akkor egyetlen, NULL értékekből álló csoportot tudtunk képezni. Ez a csoport egy sorból áll, melyben B értéke NULL . Amikor pedig megszámoltuk a csoport nem NULL értékű sorainak számát, azt 0-nak találtuk.

Ugyanakkor a

```
SELECT A, SUM(B)
FROM R
GROUP BY A;
```

lekérdezés eredménye a $(\text{NULL}, \text{NULL})$ sor lesz. A magyarázat a következő: A NULL értékhez tartozó csoport az R reláció egyetlen sorából áll. Amikor e csoportban a B értékeket adjuk össze, akkor egyetlen NULL -t találunk, amelyet nem veszünk figyelembe az összeg képzésekor. Így egy üres halmaz értékeinek összegét képezzük, amely definíció szerint NULL . □

6.4.7. HAVING záradék

Tegyük fel, hogy a 6.32. 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 6.13. ábra harmadik sorát kicsérélhetnénk a következőre:

```
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, amely a csoportra vonatkozik.

6.34. példa. Keressük meg azokat a gyártásirányítókat és filmhosszaik összegét, akik legalább egy 1930 előtti filmet is készítettek. A 6.13. ábrához csatolhatjuk a következő sort:

```
HAVING MIN(év) < 1930
```

Az így kapott kérdés a 6.14. ábrán látható. A lekérdezés csak azokat a csoportokat veszi majd figyelembe, melyekben legalább egy 1930 előtti film található.

□

```
SELECT név, SUM(hossz)
FROM GyártásIrányító, Filmek
WHERE producerAzon = azonosító
GROUP BY név
HAVING MIN(év) < 1930;
```

6.14. ábra. A korai gyártásirányítók filmhosszainak összege

Néhány szabály, amelyekre gondolnunk kell a HAVING záradék alkalmazásakor:

- A HAVING záradékban hivatkozott összesítés csak az éppen feldolgozott csoport soraira vonatkozik.
- A FROM záradékban megadott relációk bármely attribútumára képezhetünk a HAVING záradékban összesítést, összesítés nélkül a HAVING záradékban csak azok az attribútumok fordulhatnak elő, amelyek a GROUP BY listában is szerepeltek. (Ugyanaz a szabály, mint ami a SELECT záradékra is vonatkozott.)

6.4.8. Feladatok

6.4.1. feladat. Adjuk meg a 2.4.1. feladatban szereplő lekérdezéseket SQL-ben, biztosítva az ismétlődések megszüntetését.

6.4.2. feladat. Adjuk meg a 2.4.3. feladatban szereplő lekérdezéseket SQL-ben, biztosítva az ismétlődések megszüntetését.

! 6.4.3. feladat. A 6.3.1. 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 alkérdezések nélküli olyan lekérdezést, amely ugyanazt az ismétlődésmentes eredményt adja.

! 6.4.4. feladat. Ugyanaz a feladat, mint a 6.4.3. feladatban, ezúttal azonban a 6.3.2. feladat lekérdezéseire vonatkozóan.

! 6.4.5. feladat. A 6.27. példában említettük, hogy a „keressük meg Harrison Ford filmjeinek gyártásirányítóit” feladatot megvalósító különböző lekérdezések különböző válaszokat adhatnak. Ezek lehetnek multihalmazok is annak ellenére, hogy ugyanazon választ adják. Vizsgáljuk meg a 6.22. példában szereplő kérdést, melynek a FROM záradékában alkérdezést használtunk. Ez a verzió eredményezhet-e duplikátumokat, és ha igen, miért?

6.4.6. feladat. Adjuk meg a következő lekérdezéseket, a 2.4.1. feladat alábbi adatbázisára vonatkozóan:

Termék(gyártó, modell, típus)

PC(modell, sebesség, memória, merevlemez, ár)

Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)

Nyomtató(modell, színes, típus, ár)

Szemléltessük az eredményt a 2.4.1. feladat adataival.

- a) Keressük meg a PC-k átlagos sebességét.
- b) Keressük meg az 1000 \$-nál drágább laptopok átlagos sebességét.
- c) Keressük meg az „A” gyártó által gyártott PC-k átlagos árát.
- ! d) Keressük meg a „D” gyártó által gyártott PC-k és laptopok átlagos árát.
- e) Keressük meg minden egyes PC sebességéhez 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 a 2.0-nál nagyobb sebességű PC-k átlagos árát.
- !! j) Keressük meg minden olyan gyártóhoz, akik nyomtatót gyártanak, a PC-k átlagos merevlemezmeretét.

6.4.7. feladat. Adjuk meg a következő SQL-lekérdezéseket, amelyek a 2.4.3. 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 az 2.4.3. 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 hajók számával?

- ! d) Keressük meg minden hajóosztály esetén azt az évet, amikor az ebbe az osztályba tartozó első hajót felavatták.
- ! e) Keressük meg minden hajóosztály esetén a csatában elszülyesztett hajók számát.
- !! f) Keressük meg a legalább három hajóból álló osztályokra a csatában elszülyesztett hajók számát.
- !! g) Egy ágyú által kilőt golyó súlya (fontokban) körülbelül akkora, mint a kaliber (hüvelykben) köbének a fele. Keressük meg minden országra a hozzá tartozó hajók ágyúgolyónak átlagos súlyát.

6.4.8. feladat. Az 5.10. példában feltettünk egy γ formában megfogalmazott kérdést. („Keressük meg azokat a színészeket, akik első filmszerepük évében legalább három filmben szerepeltek!”) Adjuk meg ezt a lekérdezést SQL-ben.

! 6.4.9. feladat. A kibővített relációs algebra γ operátorának nincsen az SQL HAVING-jének megfelelő lehetősége. Utánozható-e a kiterjesztett relációs algebrában a HAVING záradékkal megfogalmazott SQL-lekérdezés? Ha igen, akkor milyen általános szabály szerint?

6.5. Változtatások az adatbázisban

Eddig a pontig a select-from-where lekérdező SQL-utasításra koncentráltunk. Léteznek további SQL-utasítások is, amelyek nem egy eredményt adnak vissza, hanem megváltoztatják az adatbázis állapotát. Ebben a fejezetben három utasítástípust ismertetünk, amelyek a következő hatással járnak:

1. Sorok beszűrása egy relációba.
2. Bizonyos sorok törlése egy relációból.
3. Bizonyos létező sorok meghatározott komponensei értékeinek módosítása.

Az ilyen típusú műveleteket általában *módosításoknak* nevezzük.

6.5.1. Beszúrás

A beszúrási művelet legegyszerűbb alakja:

```
INSERT INTO R(A1, ..., An) VALUES (v1, ..., vn);
```

A sor úgy keletkezik, hogy az A_i attribútumhoz v_i értéket rendelünk, $i = 1, 2, \dots, n$. Ha az attribútumlista nem tartalmazza R összes attribútumát, akkor a hiányzó attribútumok az alapértelmezés szerinti értéket kapják.

6.35. példa. Tételezzük fel, hogy Kate Winsletet szeretnénk a *Titanic* szereplőinek listájába felvenni. A megfelelő utasítás:

- 1) `INSERT INTO SzerepelBenne(filmCím, filmÉv, színészNév)`
- 2) `VALUES('Titanic', 1998, 'Kate Winslet');`

Az utasítás hatása, hogy egy sort, amelynek három komponense a 2. sorban található, beszűrunk a `SzerepelBenne` relációba. Mivel a `SzerepelBenne` reláció minden attribútumát felsoroltuk az 1. sorban, nincs szükség alapértelmezett értékekre. A 2. sorban található értékeket összpárosítjuk az 1. sorban felsorolt megfelelő attribútumokkal, így például a `filmCím` értéke `'Titanic'` lesz stb.

□

Ha a 6.35. 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.

- Ha nem vagyunk biztosak az attribútumok sorrendjében, jobb, ha fel soroljuk azokat az `INSERT` záradékban, abban a sorrendben, ahogyan a `VALUES` záradékban megadtuk az értékeiket.

Az előbb ismertetett legegyszerűbb `INSERT` utasítás csak egy sort szűr be a relációba. A sorhoz megadott értéklista helyett egy alkérdezés segítségével meg határozhatunk több beszúrandó sort is. Ez az alkérdezés helyettesíti a `VALUES` kulcsszót és az `INSERT` utáni sorkifejezést.

6.36. példa. Tételezzük fel, hogy a következő relációba be szeretnénk szűrni a `Filmek` relációban megtalálható összes olyan stúdiót, melyek azonban a `Stúdió` relációban nem szerepelnek:

`Stúdió(név, cím, elnökAzon)`

A `Filmek` reláció sémája:

`Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)`

Mivel nem tudjuk ezen stúdiók címét és elnökének azonosítóját, ezek az attribútumok `NULL` értéket kapnak. Az utasítást a 6.15. ábra szemlélteti.

Mint a legtöbb SQL-utasítást, a 6.15. ábra utasítását is legkönnyebb belülről kifelé értelmezni. Az 5–6. sorok az összes `Stúdió` relációbeli stúdiónevet eredményezik. Ezt használva a 4. sor azt ellenőrzi, hogy a `Filmek` sor stúdióneve szerepel-e ezen stúdiók között.

```

1) INSERT INTO Stúdió(név)
2)     SELECT DISTINCT stúdióNév
3)     FROM Filmek
4)     WHERE stúdióNév NOT IN
5)         (SELECT név
6)         FROM Stúdió);

```

6.15. ábra. Új stúdiók beszúrása

A beszúrások időzítése

Az SQL-szabvány előírja, hogy a lekérdezést teljes egészében ki kell értékelni, mielőtt bármely sort beszúrnánk. A 6.15. ábra példájában 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.

Ebben a példában nincs jelentősége, hogy a beszúrások késleltetve vannak-e addig, amíg a lekérdezés teljes kiértékelése megtörténik. Tegyük fel azonban azt, hogy a 6.15. 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 *Filmek* relációban, az eredményben is többször szerepelne, tehát a *Stúdió* relációba is többször kerülne be. Viszont ha minden egyes megtalált stúdiót – a szabványtól eltérően – rögtön beszúrná az adatbázis-kezelő rendszer, akkor 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.

Tehát a 2–6. sorok azokat a stúdióneveket eredményezik, amelyek a *Filmek*ben benne vannak, de a *Stúdió*-ban nincsenek benne. A 2. sorban a DISTINCT azt biztosítja, hogy ebben a halmaiban mindegyik stúdió csak egyszer szerepel függetlenül attól, hogy hány filmet gyártottak. Végül az 1. sor beszúrja ezeket a stúdiókat, NULL értéket adva a *cím* és *elnökAzon* komponenseknek. □

6.5.2. Törlés

A törlési utasítás alakja:

```
DELETE FROM R WHERE <feltétel>;
```

Az utasítás azt eredményezi, hogy az *R* relációból kitörlődik minden olyan sor, amely megfelel a feltételnek.

6.37. példa. A következő relációból:

```
SzerepelBenne(filmCím, filmÉv, színészNév)
```

a következő SQL-utasítás segítségével kitörölhetjük azt a tényt, hogy Kate Winslet játszott a *Titanic*ban:

```
DELETE FROM SzerepelBenne
WHERE filmCím = 'Titanic' AND
    év = 1998 AND
    színészNév = 'Kate Winslet';
```

Figyeljük meg, hogy a 6.35. példa beszúrási műveletével ellentétben nem tudjuk könnyen megadni a törölni kívánt sort. A WHERE feltételben kell egészen pontosan megadni, hogy melyik sort akarjuk törölni. □

6.38. példa. Egy újabb példa a törlésre. Ezúttal töröljünk ki a következő relációból

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

több olyan sort egyszerre, amelyek megfelelnek egy bizonyos feltételnek:

```
DELETE FROM GyártásIrányító
WHERE nettóBevétel < 10000000;
```

Ezzel az utasítással kitöröljük a relációból a kis bevételű – tízmillió dollár alatti – gyártásirányítókat. □

6.5.3. 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:

```
UPDATE R SET <új értékkedások> WHERE <feltétel>;
```

Mindegyik új értékkedés egy attribútumból, az egyenlőségjelből és egy kifejezésből áll. Ha több mint egy értékkedés van, vesszővel kell azokat elválasztani. Az utasítás eredményeképpen az összes olyan *R*-beli sorban, amelyek megfelelnek a feltételnek, az értékkedáslistának megfelelően a komponensek módosulnak.

6.39. példa. Módosítsuk a következő relációt:

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

a név elő téve az 'Ig.' rövidítést minden olyan gyártásirányító esetén, aki elnöke (igazgatója) egy stúdiónak. A feltétel az, hogy az azonosító szerepeljen a Stúdió reláció valamelyik sorának az `elnökAzon` komponensében. 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 elő. Az így kapott karaktersor lesz a `név` komponens új értéke. □

6.5.4. Feladatok

6.5.1. feladat. Adjuk meg a következő adatbázis-módosításokat a 2.4.1. feladat adatbázissémájára vonatkozóan. Írjuk le ezen feladat adatmódosításainak hatását.

`Termék(gyártó, modell, típus)`

`PC(modell, sebesség, memória, merevlemez, 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, 3.2 a sebessége, a memóriája 1024, merevlemeze 180 és az ára 2499 \$.
- b) Szűrjuk be az adatbázisba azt, hogy minden egyes PC esetén létezik egy vele megegyező gyártójú, sebességű, memóriájú, merevlemezű laptop, amelynek 17 hüvelykes képernyője van, modellszáma 1100-zal nagyobb és 500 \$-ral drágább.
- c) Töröljük ki a 100 gigabájtnál kisebb merevlemezű PC-ket.
- d) Töröljük ki az összes olyan laptopot, amelyeket olyan cég gyárt, amely nem gyárt nyomtatókat.
- e) Az A cég megveszi a B céget. Módosítsuk a B által gyártott termékeket olyan módon, hogy az A gyártja őket.
- f) minden egyes PC esetén kétszerrezzük meg a memória nagyságát, és adjunk 60 gigabájtot a merevlemez méretéhez. (Ne feledjük, hogy egyetlen `UPDATE` utasítás segítségével több attribútum is módosítható.)

- ! g) A B cég által gyártott laptopok esetén adjunk egy hüvelyket a képernyő méretéhez és vonjunk ki 100 \$-t az árból.

6.5.2. feladat. Adjuk meg a következő adatbázis-módosításokat a 2.4.3. feladat adatbázissémájára vonatkozóan. Írjuk le ezen feladat adatmódosításainak hatását.

Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
kaliber, vízkiszorítás)

Hajók(név, osztály, felavatva)

Csaták(név, dátum)

Kimenetelek(hajó, csata, eredmény)

- A Nelson osztályba tartozó két brit csatahajót – a Nelsont és a Rodney-t – 1927-ben avatták fel, mindenüknek 16 hüvelykes kaliberű ágyúja van, és 34 000 tonna a vízkiszorításuk. Szűrjuk be ezeket az adatokat az adatbázisba.
- A Vittorio Veneto hajóosztályba tartozó három olasz csatahajóból kettőt – a Vittorio Venetót és az Italiát – 1940-ben avattak fel; a harmadik hajót, a Rómát pedig 1942-ben. Mind a háromnak 9 db 15 hüvelykes kaliberű ágyúja van és vízkiszorításuk 41 000 tonna. Szűrjuk be ezeket az adatokat az adatbázisba.
- Töröljük ki a Hajók relációból az összes elsüllyesztett hajót.
- 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.
- Töröljük ki a háromnál kevesebb hajóból álló osztályokat.

6.6. Tranzakciók SQL-ben

Az eddigiekben feltételeztük, hogy az adatbázishoz egyidejűleg egy felhasználó fér hozzá, ezért a program által elvégzett adatbázis-műveletek egymás után realizálódnak: az egyik művelet eredményeként létrejött adatbázis-állapot szolgál a következő adatbázis-művelet kiindulási állapotaként. Továbbá feltételeztük, hogy az összes művelet teljes egészében (atomosan) lefut: sem softver-, sem pedig hardverhibák nem idézhetnek elő olyan helyzeteket, amelyekben az adatbázis állapota egy félbemaradt művelet eredménye.

A valós alkalmazásoknál sokkal bonyolultabb a helyzet. Ebben a részben először megvizsgáljuk, hogy milyen problémák származhatnak abból, hogy az adatbázis állapota nem a rajta végrehajtott műveletek eredményét tükrözi, majd megvizsgáljuk azt, hogy az SQL milyen eszközöket ad az ilyen és ehhez hasonló problémák elkerülésére.

7. fejezet

Megszorítások és triggerek

Ebben a fejezetben az SQL-nek azokat a sajátosságait tekintjük át, amelyek az aktív elemek létrehozásával kapcsolatosak. Egy aktív elem olyan kifejezés vagy utasítás, amelyet egyszer megírunk, eltárolunk az adatbázisban, és azt várjuk el tőle, hogy a megfelelő időpillanatokban lefusson. Ez az időpont lehet egy esemény bekövetkezése, mint például egy adott relációba való beszúrás, vagy lehet az adatbázisnak olyan megváltozása, amikor egy logikai értékű feltétel igazzá válik.

Az egyik komoly nehézség, amivel az alkalmazásfejlesztők szembe találják magukat, hogy az adatbázis módosításakor az új információ nagyon sokféleképpen lehet hibás. Kézzel bevitt adatok esetén például gyakran fordul elő elírás vagy másolási hiba. Az alkalmazásokat megírhatjuk úgy is, hogy minden beszúrást, törlést és módosítást a szükséges, helyességet biztosító ellenőrzésekhez rendelünk hozzá. A legjobb ezeket az ellenőrzéseket az adatbázisban tárolni, és az ABKR-re bízni az ellenőrzést. Ekkor ugyanis biztosan nem felejtődik el egyetlen ellenőrzés sem, sőt így a felesleges munkát is elkerülhetjük.

Az SQL számos lehetőséget kínál arra, hogy az *épségi megszorításokat* az adatbázisséma részeként adjuk meg. Ebben a fejezetben a legfontosabb módszereket fogjuk áttekinteni. Korábban már foglalkoztunk a kulcsmegszorításokkal, amelyeknél egy attribútum vagy egy attribútumhalmaz a reláció kulcsaként van megadva. Az SQL lehetővé teszi az attribútumokra, a sorokra vonatkozó megszorításokat és a több relációt érintő megszorításokat is. Ez utóbbiakat önálló megszorításoknak nevezzük. Végül pedig a fejezet végén a „triggereket” fogjuk bemutatni. Ezek olyan aktív elemek, amelyek bizonyos események hatására jönnek működésbe. Ilyen esemény lehet például egy adott relációba való beszúrás.

7.1. Kulcsok és idegen kulcsok

Emlékezzünk vissza a 2.3.6. alfejezetre, amelyben láthattuk, hogy az SQL lehetővé teszi a PRIMARY KEY, illetve UNIQUE kulcsszó használatával azt, hogy egy vagy több attribútum a reláció kulcsa legyen. Az SQL nyelv a „kulcs” kifejezést

használja bizonyos hivatkozásiépség-megszorítások esetén is. Ezek a megszorítások, amelyeket idegenkulcs-megszorításoknak hívunk, azt mondják ki, hogy egy relációban előforduló értéknek szerepelnie kell egy másik reláció elsődleges kulcsoszlopában vagy oszlopaiban is.

7.1.1. Idegen kulcsok megadása

Az idegenkulcs-megszorítás bizonyos attribútumok értékeinek jelentésére előírt követelmény. Vegyük például a 2.21. példát, amelyben azt néztük meg, hogyan fejezhető ki a relációs algebrában az a megszorítás, hogy a filmeknek a producer „azonosító szám” értéke néhány gyártásirányítónak is azonosító száma lesz a GyártásIrányító relációban.

Az SQL-ben egy reláció azon attribútumát vagy attribútumait *idegen kulcsnak* deklarálhatjuk, amelyek egy másik reláció (ez lehet akár ugyanaz a reláció is) bizonyos attribútumaira hivatkoznak. Ez a deklaráció két dolgot jelent egyszerre.

1. A másik reláció azon attribútumait, amelyekre hivatkozunk, elsődleges kulcsként vagy UNIQUE kulcsként kell deklarálni abban a relációban. Enélkül nem adhatjuk meg az idegenkulcs-deklarációt.
2. Az idegen kulcs értékeinek, amelyek előfordulnak az első relációban, elő kell fordulniuk a hivatkozott attribútumokban is a másik reláció valamelyik sorában. Még pontosabban kifejezve, legyen F egy idegen kulcs, amely egy másik reláció G attribútumhalmazára hivatkozik. Tegyük fel, hogy az első relációnak egy t sora az F attribútumaiban csupa nem NULL értékkel rendelkezik. Jelöljük t -nek ezen attribútumokon felvett értékeit $t[F]$ -vel. Ekkor a hivatkozott relációt kell hogy legyen olyan s sora, amelyik a G attribútumain megegyezik $t[F]$ -vel, vagyis $s[G] = t[F]$.

Az idegen kulcsot is kétféleképpen deklarálhatjuk ugyanúgy, ahogyan azt az elsődleges kulcsok esetében láttuk.

- a) Ha az idegen kulcs egyetlen attribútum, akkor az attribútum neve és típusa után adhatjuk meg, hogy az egy másik tábla egy attribútumára hivatkozik. (Annak az attribútumnak ott elsődleges kulcsnak vagy unique-nak kell lennie.) A deklaráció formája a következő:

REFERENCES <tábla>(<attribútum>)

- b) A másik lehetőség, hogy a CREATE TABLE utasításban az attribútumok listája után egy külön deklarációban adjuk meg, hogy bizonyos attribútumok idegen kulcsot alkotnak. Itt kell megadnunk azt a táblát és azokat az attribútumokat, amelyekre az idegen kulcs hivatkozik. (Ezeknek az attribútumoknak ez esetben is elsődleges vagy unique kulcsnak kell lenniük.) A deklaráció formája ekkor a következő:

```
FOREIGN KEY (<attribútumok>) REFERENCES
    <tábla>(<attribútumok>)
```

7.1. példa. Tegyük fel, hogy a következő relációt szeretnénk deklarálni:

```
Stúdió(név, cím, elnökAzon)
```

Ennek az elsődleges kulcsa a név, az elnökAzon attribútuma pedig idegen kulcs, amelyik a

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

reláció azonosító attribútumára hivatkozik.

Az elnökAzon közvetlenül deklarálható az azonosító attribútumra történő hivatkozással a következő módon:

```
CREATE TABLE Stúdió (
    név CHAR(30) PRIMARY KEY,
    cím VARCHAR(255),
    elnökAzon INT REFERENCES GyártásIrányító(azonosító)
);
```

Vagy megadhatjuk úgy is, hogy az idegen kulcsot külön deklaráljuk:

```
CREATE TABLE Stúdió (
    név CHAR(30) PRIMARY KEY,
    cím VARCHAR(255),
    elnökAzon INT,
    FOREIGN KEY elnökAzon REFERENCES
        GyártásIrányító(azonosító)
);
```

Figyeljük meg, hogy a hivatkozott azonosító attribútum a GyártásIrányító táblának valóban kulcsa, ahogyan az szükséges. Mindkét fenti deklarációt az a jelentése, hogy ha egy érték szerepel a Stúdió tábla egy sorának elnökAzon oszlopában, akkor ennek az értéknek szerepelnie kell a GyártásIrányító tábla valamely sorának azonosító oszlopában is. Kivétel az az eset, ha a Stúdió tábla egy sorában az elnökAzon oszlopban NULL érték szerepel. Ilyenkor nem követelmény, hogy a NULL érték szerepeljen a másik tábla azonosító oszlopában. Valójában az azonosító elsődleges kulcs, és így nem is szerepelhet benne NULL érték. □

7.1.2. Hivatkozási épség fenntartása

A séma tervezője három lehetőség közül választhat az idegenkulcs-megszorítás kikényszerítésére. Az általános megközelítéseket megfigyelhettük a 7.1. példa vizsgálatánál, amelyben elvárás volt, hogy a Stúdió reláció elnökAzon értéke egyben a GyártásIrányító reláció azonosító értéke is legyen. A következő műveleteket az ABKR visszautasítja, vagyis egy futási hibát generál:

- a) Ha megpróbálunk egy olyan sort beszúrni a Stúdió táblába, amelyben az **elnökAzon** értéke nem NULL, és nem egyezik meg a GyártásIrányító táblabeli **azonosító**-hoz tartozó egyik értékkel sem.
- b) Ha megpróbáljuk módosítani a Stúdió tábla egy sorát, és az **elnökAzon** attribútumot egy olyan nem NULL értékre változtatjuk, amelyik nem egyezik meg a GyártásIrányító táblabeli egyik **azonosító** értékkel sem.
- c) Ha megpróbálunk kitörölni egy sort a GyártásIrányító táblából, amelyben az **azonosító** értéke nem NULL és szerepel a Stúdió tábla **elnökAzon** oszlopában.
- d) Ha megpróbáljuk módosítani a GyártásIrányító tábla egy sorát oly módon, hogy az **azonosító** értékét is megváltoztatjuk és a régi **azonosító** értéke szerepel a Stúdió tábla **elnökAzon** oszlopában.

Az első két módosítás esetén, amelyek az idegenkulcs-megszorítással deklarált relációt akarták megváltoztatni, nincs választási lehetőségünk. A rendszer egyszerűen visszautasítja a megszorítást sértő módosítást. A hivatkozott relációra vonatkozó módosításoknál (vagyis az utolsó kettőnél) viszont a tervező három lehetőség közül választhat:

1. *Alapértelmezés szerinti eljárás (Megszorítást sértő módosítások visszautasítása).* Az SQL-ben az alapértelmezés szerinti eljárás minden olyan módosítást, amely megsérti a hivatkozásiépség-megszorítást, visszautasít.
2. *Továbbgyűrűző eljárás.* Ezzel az eljárással az idegen kulcsok hivatkozott attribútuma(i) is megváltoznak. Ha például a továbbgyűrűző eljárással egy stúdió elnökéhez tartozó GyártásIrányító sort törlünk, akkor a rendszer a hivatkozási épség megőrzéséhez a Stúdió reláció hivatkozott sorait is törli. Ha c_1 -ről c_2 -re változtatjuk egy gyártásirányító azonosítóját, és van olyan sora a Stúdió táblának, amelyben az **elnökAzon** értéke c_1 , akkor a rendszer ezt az értéket is c_2 -re módosítja.
3. *A NULL értékre állítás módszere.* Itt a hivatkozott reláció egy idegen kulcsának értékét érintő módosításánál a hivatkozott reláció megfelő értékeit NULL értékre változtatjuk. Ha például a GyártásIrányító reláció az egyik stúdió elnökéhez tartozó sorát töröljük, akkor a rendszer a szóban forgó stúdió **elnökAzon** értékeit NULL értékre kell állítsa. Ha ennek az elnöknek az azonosító számát a GyártásIrányító relációban megváltoztatjuk, akkor a Stúdió reláció **elnökAzon** értékét NULL értékre kell változtassuk.

A fenti módszerek közül az alkalmazni kívántat a törlés és módosítás esetére külön-külön, egymástól függetlenül megadhatjuk, amikor az idegen kulcsot deklaráljuk. A megadásuk úgy történik, hogy a törlés esetén használandó módszert az **ON DELETE**, a módosítás esetén alkalmazandót pedig az **ON UPDATE** kulcsszó után adjuk meg. A módszerekre vonatkozó kulcsszavak pedig **NULL** értékre állítás esetén **SET NULL**, továbbgyűrűző módszer esetén pedig **CASCADE**.

7.2. példa. Nézzük meg, hogyan kell módosítanunk a

Stúdió(név, cím, elnökAzon)

reláció 7.1. példában szereplő deklarációját, hogy a

GyártásIrányító(név, cím, azonosító, nettóBevétel)

relációra vonatkozó törlések és módosítások esetén az általunk kívánt módszert alkalmazza a rendszer. A 7.1. ábrán az említett példában szereplő első CREATE TABLE utasítást láthatjuk az ON DELETE és ON UPDATE záradékkal kiegészítve. Az 5. sor azt adja meg, hogy ha a GyártásIrányító táblából kitörlünk egy sort, akkor az összes Stúdió-beli sorban, amelyekben az éppen törölt gyártásirányító volt az elnök, az elnökAzon értékét NULL-ra kell változtatni. A 6. sor azt mondja meg, hogy ha a GyártásIrányító tábla egy sorában megváltoztatjuk az azonosítót, akkor a Stúdió tábla megfelelő soraiban ugyanarra az értékre kell változtatni az elnökAzon értékét.

```

1) CREATE TABLE Stúdió (
2)     név CHAR(30) PRIMARY KEY,
3)     cím VARCHAR(255),
4)     elnökAzon INT REFERENCES GyártásIrányító(azonosító)
5)         ON DELETE SET NULL
6)         ON UPDATE CASCADE
);

```

7.1. ábra. Módszer megadása a hivatkozási épség megőrzésére

Figyeljük meg, hogy a példában a törlés esetén a nullértékre állítás tűnik értelmes megoldásnak, míg a módosítás esetén a továbbgyűrűző módszer a logikus. Ha ugyanis egy stúdió elnöke visszavonul, attól még a stúdió tovább működik, egy darabig esetleg elnök nélkül. Ha azonban egy stúdió elnökének megváltozik az azonosítója, az valószínűleg csak valamilyen adminisztratív változást jelent. Valószínűleg az illető továbbra is elnöke marad a stúdiónak, és ezért ezt a változást abban a relációban is követni kellene. □

7.1.3. Megszorítások ellenőrzésének késleltetése

Vegyük a 7.1. példában említett szituációt, ahol a Stúdió reláció elnökAzon attribútuma idegen kulcs, amely a GyártásIrányító tábla azonosító oszlopára hivatkozik. Tegyük fel, hogy Arnold Schwarzenegger Kalifornia kormányzójaként visszavonul, és elhatározza, hogy egy filmstúdiót alapít, amelynek neve La Vista Studios lesz. A stúdió elnöke természetesen ő lesz. Ha végrehajtjuk a következő beszűrást, akkor gondban leszünk.

Lógó sorok és módosítási eljárások

Azokat a sorokat, amelyekben olyan idegen kulcs értéke szerepel, amely nincs benne a hivatkozott táblában, *lógó soroknak* nevezzük. Emlékezzünk rá, hogy azokat a sorokat is lógó soroknak neveztük, amelyek egy összekapcsolás esetén kimaradtak az eredményből. A fenti két doleg szoros kapcsolatban áll egymással. Ha egy sorbeli idegen kulcs értéke hiányzik a hivatkozott táblából, akkor az adott sor nem lesz benne a relációnak a hivatkozott relációval vett összekapcsolásában. Ha az összekapcsolás az idegen kulcs, illetve a hivatkozott kulcs egyenlőségén alapul, akkor az összekapcsolást *idegenkulcs-összekapcsolásnak* nevezzük. A lógó sorok pontosan azok a sorok lesznek, amelyek megsértik az idegenkulcs-megszorítás hivatkozásiépség-elvárását.

```
INSERT INTO Stúdió
VALUES('La Vista', 'New York', 23456);
```

Az lesz a probléma, hogy nincs a GyártásIrányító táblának 23456 azonosítójú sora (feltételezzük, hogy ez lesz Arnold Schwarzenegger új azonosítója), és így nyilvánvaló módon megsértjük az idegenkulcs-megszorítást.

Az egyik lehetséges megoldás, hogy először szúrjuk be a La Vista stúdióra vonatkozó sort az elnök azonosítója nélkül:

```
INSERT INTO Stúdió(név, cím)
VALUES('La Vista', 'New York');
```

Ez a módosítás már nem sérti meg a megsorítást, mivel a La Vistára vonatkozó sorban NULL érték kerül az elnökAzon oszlopba, és ez nem követeli meg semmilyen érték meglétét a hivatkozott oszlopból. Mielőtt azonban módosítani tudnánk az értéket az alábbi utasítással, először be kell szűrnunk egy Arnold Schwarzeneggerre vonatkozó sort a GyártásIrányító táblába a megfelelő azonosítóval.

```
UPDATE Stúdió
SET elnökAzon = 23456
WHERE név = 'La Vista';
```

Ha nem szúrjuk be előzőleg a megfelelő sort a GyártásIrányító táblába, akkor a fenti módosítás is meg fogja sérteni az idegenkulcs-megszorítást.

A jelen példában, ha először beszúrjuk Arnold Schwarzeneggert és az Ő azonosítóját a GyártásIrányító táblába, mielőtt a La Vistát is beszúrnánk, ezzel biztosan elkerülhetjük az idegen kulcs megsértését. Vannak azonban olyan esetek, amikor a *körkörös megsorítások* miatt még az adatbázis-módosítások sorrendjének megfontolt megválasztásával sem tudjuk orvosolni a problémát.

7.3. példa. Ha a gyártásirányítók csak stúdióelnökök lehetnének, akkor az azonosító oszlopot idegen kulcsként kellene deklarálnunk, ami a Stúdió(elnökAzon)-ra hivatkozik. Ez esetben az elnökAzon oszlopot UNIQUE-nak kellene deklarálnunk, aminek akkor van értelme, ha feltesszük, hogy egy ember nem lehet egyidejűleg két stúdiónak az elnöke.

Ebben az esetben lehetetlen új stúdiókat és elnököket felvinni. Nem tudunk új elnökAzon értékkel rendelkező sort beszúrni a Stúdió táblába, mert ez a sor megsértené azt az idegen kulcsot, ami az elnökAzon-ról a GyártásIrányító(azonosító)-ra mutat. Új azonosító értékkel rendelkező sort sem tudunk beszúrni a GyártásIrányító táblába, mert az pedig azt az idegen kulcsot sérti meg, amely az azonosító-ról a Stúdió(elnökAzon)-ra mutat. □

A 7.3. példában szereplő probléma másképp is megoldható:

1. Először is szükségünk van arra, hogy a két beszúrást (az egyiket a Stúdió, a másikat a GyártásIrányító táblába) egyetlen tranzakcióba foglaljunk.
2. Másrészt valahogy meg kell mondanunk az ABKR-nek, hogy a megszorításokat ne ellenőrizze, amíg az egész tranzakció be nem fejeződött. A tranzakciók nyelvén ezt a befejeződést jóváhagyásnak nevezzük.

Ahhoz hogy a 2. pontban az ABKR-t informáljuk, bármelyik megszorítás (kulcs, idegen kulcs vagy a fejezetben később előforduló megszorítástípusok) deklarációját egy DEFERRABLE (késleltethető) vagy egy NOT DEFERRABLE kulcsszó kell kövesse. Az utóbbi az alapértelmezés, és ez azt jelenti, hogy minden adatbázis-módosításkor a megszorítás közvetlenül utána ellenőrzésre kerül, ha a módosítás egyáltalán megsértheti az idegenkulcs-megszorítást. Ha azonban a megszorítást DEFERRABLE-ként deklaráljuk, akkor lehetőségünk van arra, hogy azt mondjuk a rendszernek, hogy a megszorítás ellenőrzésével várjon a tranzakció végéig.

A DEFERRABLE kulcsszót követhet egy INITIALLY DEFERRED vagy egy INITIALLY IMMEDIATE kiegészítés is. Az első esetben az ellenőrzés a tranzakció jóváhagyásáig késleltetve lesz. A második esetben az ellenőrzés minden egyes utasítás után azonnal megtörténik.

7.4. példa. A 7.2. ábra a Stúdió reláció módosított deklarációját mutatja. A módosítás az idegen kulcs késleltetését teszi lehetővé a tranzakció végeig. Az elnökAzon oszlopot UNIQUE-nak deklaráltuk, hogy más relációk idegenkulcs-megszorításai hivatkozhassanak rá.

Ha a 7.3. példában szereplő másik idegenkulcs-deklarációt is megadtuk volna a GyártásIrányító(azonosító)-ról a Stúdió(elnökAzon)-ra, akkor olyan tranzakciókat írhatnánk, amelyek két sort szúrnak be (mindegyik táblába), és a két idegen kulcs csak a két beszúrás után kerül ellenőrzésre. Így ha egy új stúdiót és az új elnököt is beszúrjuk ugyanazzal az azonosítóval, akkor nem sérül meg egyik megszorítás sem. □

```
CREATE TABLE Stúdió (
    név CHAR(30) PRIMARY KEY,
    cím VARCHAR(255),
    elnökAzon INT UNIQUE
        REFERENCES GyártásIrányító(azonosító)
        DEFERRABLE INITIALLY DEFERRED
);
```

7.2. ábra. Az `elnökAzon` egyedisége és a megszorítás késleltetése

Két további dolgot kell megemlítenünk a megszorítások késleltetésével kapcsolatban:

- Bármelyikfajta megszorításnak nevet adhatunk. Hogy ezt hogyan tehetjük, arról majd a 7.3.1. alfejezetben szólunk bővebben.
- Ha egy megszorításnak nevet adtunk (például `saját`), akkor egy késleltethető megszorítást azonnaliról késleltetettre változtathattunk a következő SQL-utasítással:

```
SET CONSTRAINT saját DEFERRED;
```

Ennek a fordítottját is megtehetjük, ha a `DEFERRED` helyett az `IMMEDIATE` kulcsszót használjuk.

7.1.4. Feladatok

7.1.1. feladat. A 2.2.8. alfejezetben bevezetett filmes adatbázisban minden relációhoz adtunk meg kulcsot.

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmcím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

Adjuk meg a következő hivatkozásiépség-megszorítások deklarációját a filmadatbázisra:

- Egy film producerének szerepelnie kell a gyártásirányítók között. A `GyártásIrányító` tábla olyan módosításait, amelyek ezt a megszorítást megsértenék, utasítsa vissza a rendszer.
- A feltétel legyen ugyanaz, mint az előbb, de a megszorítást megsértő módosítások esetén a `Filmek` reláció `producerAzon` oszlopát változtassa a rendszer `NULL`-ra.

- c) A feltétel legyen ugyanaz, mint a)-ban, de a megszorítást megsértő törlések vagy módosítások esetén törölje vagy módosítsa a rendszer a Filmek reláció megfelelő sorait is.
- d) A SzerepelBenne relációban lévő filmeknek benne kell lenniük a Filmek táblában is. A megszorítást megsértő utasításokat a rendszer utasítsa vissza.
- e) A SzerepelBenne relációban levő színészeknek benne kell lenniük a FilmSzínész táblában is. A megszorítást megsértő utasításokat a rendszer kezelje le a megfelelő sorok törlésével.

! 7.1.2. feladat. Azt a megszorítást szeretnénk megadni, hogy minden filmnek, amely a Filmek relációban szerepel, legalább egy színéssel benne kell lennie a SzerepelBenne relációban is. Megadhatjuk-e ezt egy idegen kulcs segítségével? Indokoljuk meg a választ!

7.1.3. feladat. Javasoljunk megfelelő kulcsokat és idegen kulcsokat a 2.4.1. feladat PC-adatbázisának relációihoz. Módosítsuk a 2.3.1. feladatban szereplő SQL-sémát úgy, hogy az tartalmazza ezeknek a kulcsoknak a deklarációját.

Termék(gyártó, modell, típus)

PC(modell, sebesség, memória, merevlemez, cd, ár)

Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)

Nyomtató(modell, színes, típus, ár)

7.1.4. feladat. Javasoljunk megfelelő kulcsokat a 2.4.3. feladatban szereplő Csatahajók adatbázis relációihoz. Módosítsuk a 2.3.2. feladatban szereplő SQL-sémát úgy, hogy az tartalmazza ezeknek a kulcsoknak a deklarációját.

Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma, kaliber, vízkiszorítás)

Hajók(név, hajóosztály, felavatva)

Csaták(név, dátum)

Kimenetelek(hajó, csata, eredmény)

7.1.5. feladat. Adjuk meg a következő hivatkozásiépség-megszorításokat az előző feladatban szereplő adatbázisséma vonatkozóan. Az ott általunk megadott kulcsokat használjuk, és a megszorításokat megsértő utasításokat kezeljük le úgy, hogy a megfelelő értékeket NULL-ra változtatjuk.

- a) minden Hajók táblabeli hajóosztálynak szerepelnie kell a Hajóosztályok táblában is.
- b) minden Kimenetelek táblabeli csatának szerepelnie kell a Csaták táblában is.
- c) minden Kimenetelek táblabeli hajónak szerepelnie kell a Hajók táblában is.

7.2. Attribútumokra és sorokra vonatkozó megszorítások

Egy SQL-beli CREATE TABLE utasításon belül kétféle megszorítást fogalmazhatunk meg:

1. Egy attribútumra vonatkozó megszorítást.
2. Egy sor egészére vonatkozó megszorítást.

A 7.2.1. alfejezetben bevezetünk egy egyszerű megszorítástípust az attribútum értékére vonatkozóan, amely azt mondja ki, hogy az attribútum nem veheti fel a NULL értéket. A 7.2.2. alfejezetben áttekintjük a fenti 1. módon kifejezett megszorítások alapvető formáját. Ezeket *attribútumra vonatkozó CHECK feltételeknek* fogjuk hívni. A 2. módon kifejezett sorokra vonatkozó megszorításokat a 7.2.3. alfejezetben tárgyaljuk.

Vannak további, még általánosabbfajta megszorítások, ezekről a 7.4. és 7.5. alfejezetben lesz szó. Ez utóbbiak segítségével korlátozhatjuk a teljes relációra vonatkozó módosításokat, vagy akár több relációra vonatkozó módosításokat is, de használhatjuk azokat egyetlen attribútum vagy sor értékének korlátozására is.

7.2.1. NOT NULL feltételek

Egy egyszerű megszorítás, amely egy attribútumhoz hozzárendelhető, a NOT NULL feltétel. Ez nem engedi meg olyan sorok előfordulását, amelyekben az adott attribútum értéke NULL. A megszorítást úgy deklarálhatjuk, hogy a CREATE TABLE utasításban az attribútum megadása után a NOT NULL kulcsszót szerepeljük.

7.5. példa. Tegyük fel, hogy a Stúdió relációban az eelnökAzon értéke nem lehet NULL. Ezt megadhatjuk úgy, hogy a 7.1. ábra 4. sorát a következőre változtatjuk:

4) eelnökAzon INT REFERENCES GyártásIrányító(azonosító) NOT NULL

Ennek a változtatásnak a következő következményei lesznek:

- Nem szúrhatunk be egy olyan sort a Stúdió relációba, amelyre csak a nevet és a címet adtuk meg, mert ekkor az új sorban az eelnökAzon értéke NULL lenne.
- Nem alkalmazhatjuk a NULL értékre állítás módszerét a 7.1. ábra 5. sorához hasonló módon, ahol úgy oldaná meg a rendszer az idegenkulcs-megszorítások megsértését, hogy az eelnökAzon értékét NULL-ra változtatná.



7.2.2. Attribútumra vonatkozó CHECK feltételek

Bonyolultabb megszorítások rendelhetők hozzá egy attribútumhoz, ha a deklarációban a CHECK kulcsszót használjuk. A kulcsszót egy zárójelbe tett feltétel követi, amelyet az attribútum minden értékének ki kell elégítenie. A gyakorlatban egy attribútumra vonatkozó CHECK feltétel olyan, mint egy egyszerű korlátozás az attribútum értékeire vonatkozóan. Ilyen lehet például a megengedett értékek felsorolása vagy egy aritmetikai egyenlőtlenség. Elvben azonban a feltétel bármi lehet, ami egy SQL-lekérdezésben a WHERE után állhat. Hivatkozhat például az éppen korlátozni kívánt attribútumra oly módon, hogy a feltételben ennek az attribútumnak a nevét szerepeljük. Ha azonban a feltétel másik relációra vagy más attribútumokra hivatkozik, akkor a megfelelő relációkat egy alkérdezés FROM záradékában kell megadnunk. Ez magára a korlátozni kívánt attribútum relációjára is vonatkozik.

Egy attribútumra vonatkozó CHECK feltételt akkor ellenőriz a rendszer, amikor valamelyik sorban az adott attribútum új értéket kap. Ez történhet egy módosítás útján vagy egy beszúrás alkalmával. Ha az új érték megsérti a feltételt, akkor a rendszer az adatmódosítást visszautasítja. A módosítások esetén a megszorítás ellenőrzését az új értékre végezzük, nem pedig a régire. Ha az új érték megsérti a megszorítási feltételt, akkor a rendszer visszautasítja a módosítási kísérletet.

Fontos megérteni, hogy az attribútum alapú CHECK megszorítást nem ellenőri le a rendszer, amennyiben a módosítás nem érinti a megszorított attribútum értékét. Ez a korlát viszont vezethet a megszorítás megsérüléséhez, ha a megszorításban érintett egyéb értékek megváltoznak. Először nézzünk meg egy egyszerű példát az attribútum alapú ellenőrzésre. Majd vizsgálunk meg egy alkérdezést tartalmazó megszorítást, illetve annak a következményeit is, hogy a megszorítás csak akkor kerül ellenőrzésre, ha az attribútumát módosítjuk.

7.6. példa. Tegyük fel, hogy azt követjük meg, hogy az azonosító számok legalább hatjegyűek legyenek. Ehhez a 7.1. ábra 4. sorát, amelyik a

Stúdió(név, cím, elnökAzon)

reláció sémájának egy részét deklarálja, a következőre módosíthatjuk:

4) elnökAzon INT REFERENCES GyártásIrányító(azonosító)
CHECK (elnökAzon >= 100000)

Vegyük egy másik példát. A

FilmSzínész(név, cím, nem, születésiDátum)

reláció nem attribútumát a 2.8. ábrán CHAR(1) típusúnak deklaráltuk, ami egyetlen karaktert jelent. Valójában azonban azt szeretnénk, ha ez az egy karakter csak 'F' vagy 'N' lehetne. Ezt elérhetjük, ha a 2.8. ábra 4. sorát a következővel helyettesítjük:

4) nem CHAR(1) CHECK (nem IN ('F', 'N'))

Megjegyezzük, hogy az ('F', 'N') kifejezés egy egyoszlopos reláció két sorát jelenti. A megszorítás tehát azt írja elő, hogy a **nem** komponens értéke ennek a halmaznak az eleme kell legyen. □

7.7. példa. Azt gondolhatnánk, hogy egy idegenkulcs-megszorítást szimulálni tudunk egy attribútumra vonatkozó CHECK feltétellel, amely megköveteli a hivatkozott érték létezését. A következő egy *hibás* kísérlet annak megkövetelésére, hogy a

Stúdió(név, cím, elnökAzon)

reláció soraiban szereplő **elnökAzon** értékeknek elő kell fordulniuk a

GyártásIrányító(név, cím, azonosító, nettóBevétel)

reláció azonosító attribútumában.

Tegyük fel, hogy a 7.1. ábra 4. sorát a következővel helyettesítjük:

4) elnökAzon INT CHECK

(elnökAzon IN (SELECT azonosító FROM GyártásIrányító))

Ez egy szabályos, attribútumra vonatkozó CHECK feltétel, de vizsgáljuk meg, hogy mit is eredményez pontosan. A **Stúdió** reláció a **GyártásIrányító** táblában nem szereplő **elnökAzon** értékkel történő módosításait a rendszer meg fogja tagadni. Ez majdnem ugyanolyan viselkedés, mint amit az idegen kulcsnál tapasztaltunk, azzal a kivétellel, hogy az attribútumalapú megszorítás az **elnökAzon** NULL értékeit is visszautasítja, ha nincs NULL értékű azonosító. Ennél fontosabb különbség, hogy ha megváltoztatjuk a **GyártásIrányító** relációt, például törljük az egyik stúdió elnökének a sorát, akkor a CHECK megszorítás előtt ez a módosítás rejte fog maradni. Ezért az ilyen esetekben a törlés még akkor is megengedett lesz, amikor az **elnökAzon** attribútum alapú CHECK megszorítása megsérül. □

7.2.3. Sorra vonatkozó CHECK feltételek

Egy *R* tábla soraira vonatkozó megszorítást megadhatunk úgy, hogy a CREATE TABLE utasításban az attribútumok, a kulcsok és az idegen kulcsok deklarációja után megadjuk a CHECK kulcsszót, majd zárójelek között egy feltételt. Ez a feltétel bármi lehet, ami a WHERE után szerepelhet. A rendszer a megadott feltételt az *R* egy sorára vonatkozó feltéteként értelmezi, amelyben *R* attribútumaira a nevükkel hivatkozhatunk. Hasonlóan azonban az attribútumra vonatkozó CHECK feltételek esetéhez, a feltétel itt is hivatkozhat más relációkra vagy az *R* reláció más soraira. Ezek a hivatkozások most is alkérdésben szerepelhetnek.

A sorra vonatkozó CHECK feltételeket a rendszer akkor ellenőri, amikor egy új sort szűrünk be *R*-be, vagy amikor *R* egy sorát módosítjuk. A feltétel minden

A nem teljes körű ellenőrzés hiba vagy lehetőség?

Talán csodálkozunk azon, hogy az attribútumra és a sorra vonatkozó CHECK feltételek esetén miért engedi meg a rendszer azok megsértését, ha azok másik relációra vagy az adott reláció másik soraira hivatkoznak. Ennek az az oka, hogy ezek a megszorítások hatékonyabban valósíthatók meg, mint az általánosabb megszorítások. Az attribútumra vagy a sorra vonatkozó CHECK feltételek esetén a megszorítást csak a beszúrandó vagy módosítandó sorra kell kiértékelnie a rendszernek, míg az önálló megszorításokat minden esetben ki kell értékelni, ha a bennük szereplő bármelyik reláció megváltozik. A megfontolt adatbázis-tervező csak attribútumra vagy sorra vonatkozó feltételeket használ olyan esetben, amikor ezek megsérülésének nem áll fenn a veszélye, egyéb esetekben pedig más módszereket alkalmaz, például az önálló megszorításokat (lásd 7.4. alfejezet) vagy a triggereket (lásd 7.5. alfejezet).

új, illetve módosított sorra kiértékelődik. Ha az eredmény hamis lesz, akkor a feltételt megsértő sorra vonatkozó beszúrás- vagy módosításutasítást a rendszer visszautasítja. Ha azonban a feltétel egy másik relációra hivatkozik egy alkérdezésben (ez lehet akár maga az R is) és ennek a másik relációt a módosítása okozza azt, hogy R egy sorára a feltétel hamis lesz, akkor a rendszer emiatt nem utasítja vissza azt az utasítást. Vagyis az attribútumra vonatkozó CHECK feltételekhez hasonlóan a sorra vonatkozó CHECK feltételek is láthatatlanok más relációk számára. Így még egy R -ből történő törlés is vezethet a feltétel hamissá válásához, ha az R szerepelt valamelyen alkérdezésben.

Másrészt, ha egy sorra vonatkozó CHECK feltétel csak az ellenőrizendő sor attribútumait tartalmazza, és nincs benne alkérdezés, akkor a feltétel mindenig igaz marad. Nézzünk egy egyszerű, sorra vonatkozó CHECK feltételt, amelyik egy sor több attribútumát is tartalmazza.

7.8. példa. Idézzük fel a 2.3. példában szereplő FilmSzínész tábla sémadeklarációját. A 7.3. ábra megismétli a CREATE TABLE utasítást és kiegészíti azt egy elsődleges kulcs deklarációval, valamint egy további megszorítással, amelyik egy lehetséges „következetességi feltétel”, amelyet szeretnénk ellenőrizni. Ez a feltétel azt mondja ki, hogy ha egy színész neme férfi, akkor a neve nem kezdődhet 'Ms.'-szel.

A 2. sorban a név attribútumot a reláció elsődleges kulcsaként adtuk meg, a 6. sor pedig a megszorítást deklarálja. A megszorítás feltétele igaz minden nőnemű filmszínészre és minden olyan színészre, akinek a neve nem 'Ms.'-szel kezdődik. A feltétel azokra a sorokra lesz hamis, amelyekben a nem oszlop értéke férfi és a név 'Ms.'-szel kezdődik. Éppen ezeket a sorokat szerettük volna kizárnai a relációból. □

A megszorítások helyes megfogalmazása

Sokszor van szükségünk olyan megszorításra, amely hasonlít a 7.8. példában szereplőhöz, vagyis ahol olyan sorok előfordulását szeretnénk kizární, amelyek két vagy több feltételnek is eleget tesznek. Ilyenkor a CHECK kulcsszót a feltételek tagadásainak kell követnie, közöttük az OR kulcsszót megadva. Ez az átalakítás egyike a „De Morgan-szabályoknak”: a konjunkció tagadása azonos az állítások tagadásainak diszjunkciójával. A 7.8. példában az első feltétel az lenne, hogy a színész férfi legyen. Ezért használtuk a nem = 'N' feltételt, mint ennek tagadását. (Használhattuk volna a nem <> 'F' feltételt is.) A második feltétel, hogy a név 'Ms.'-szel kezdődjön, aminek a tagadására a NOT LIKE összehasonlítást használtuk. Ez az összehasonlítás magát a feltételt tagadja, ami eredetileg az SQL-ben név LIKE 'Ms.%' lett volna.

```

1) CREATE TABLE FilmSzínész (
2)   név CHAR(30) PRIMARY KEY,
3)   cím VARCHAR(255),
4)   nem CHAR(1),
5)   születésiDátum DATE,
6)   CHECK (nem = 'N' OR név NOT LIKE 'Ms.%')
);

```

7.3. ábra. Egy megszorítás a FilmSzínész táblára

7.2.4. A sor- illetve attribútum-alapú megszorítások összehasonlítása

Ha egy sorra vonatkozó megszorítás a sor több attribútumát is tartalmazza, akkor soralapú megszorításként kell megfogalmazni. Amennyiben csak a sor egyetlen attribútumára vonatkozik, akkor megírható akár sor-, akár attribútumalapú formában is. Egyik esetben sem számoljuk az alkérdezben szereplő attribútumokat, azaz még az attribútumalapú megszorítás esetén is szerepelhet ugyanazon relációnak több eltérő attribútuma az alkérdezben.

Ha csak a sor egy attribútumát érinti (nem számolva az alkérdésekkel) az elvárás, akkor a feltétel ellenőrzése ugyanúgy zajlik, függetlenül attól, hogy sor- vagy attribútumalapú megszorítást írtunk. Noha a soralapú megszorítás gyakrabban kerül ellenőrzésre, mint az attribútumalapú. Ugyanis a soralapú megszorítás esetén a feltétel a sor bármely megfelelő attribútumértékének a megváltozásakor ellenőrzésre kerül, míg az attribútumalapúnál csak az adott attribútum értékének megváltozásakor.

7.2.5. Feladatok

7.2.1. feladat. Adjuk meg a következő megszorításokat a

Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)

reláció attribútumaira vonatkozóan:

- a) Az év nem lehet 1915-nél korábbi.
- b) A hossz nem lehet 60-nál kisebb és 250-nél nagyobb.
- c) A stúdió neve csak Disney, Fox, MGM vagy Paramount lehet.

7.2.2. feladat. Adjuk meg a következő megszorításokat az 2.4.1. példában szereplő séma attribútumaira vonatkozóan:

Termék(gyártó, modell, típus)

PC(modell, sebesség, memória, merevlemez, cd, ár)

Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)

Nyomtató(modell, szín, típus, ár)

- a) Egy laptop sebessége nem lehet kisebb 2.0-nál.
- b) A nyomtatók típusa csak lézer, tintasugaras vagy buborék lehet.
- c) A termékek típusa csak PC, laptop vagy nyomtató lehet.
- d) Egy termék modellje egy PC, egy laptop vagy egy nyomtató modellje kell hogy legyen.

7.2.3. feladat. Írjuk meg a következő megszorításokat sorra vonatkozó CHECK feltételek formájában az alábbi relációkra vonatkozóan:

Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)

SzerepelBenne(filmCím, filmÉv, színészNév)

FilmSzínész(név, cím, nem, születésiDátum)

GyártásIrányító(név, cím, azonosító, nettóBevétel)

Stúdió(név, cím, elnökAzon)

Ha a megszorítás két relációt is érint, akkor írunk minden relációhoz megfelelő megszorítást, hogy bármelyik relációra vonatkozó beszúrás vagy módosítás esetén a rendszer ellenőrizze a feltételt. Tegyük fel, hogy törlések nem fordulhatnak elő. A sorra vonatkozó CHECK feltételekkel megadott megszorítások teljesülését törlések esetén nem lehet biztosítani.

- a) Egy filmszínész nem szerepelhet olyan filmben, amelyet a születése előtt készítettek.
- b) Két stúdiónak nem lehet azonos a címe.

- ! c) Egy név, amelyik benne van a FilmSzínész táblában, nem szerepelhet a GyártásIrányító táblában is.
- ! d) A Stúdió táblában szereplő stúdióneveknek a Filmek tábla legalább egy sorában szerepelniük kell.
- !! e) Ha egy film producere egyben egy stúdió elnöke is, akkor neki kell lennie a filmet készítő stúdió elnökének is.

7.2.4. feladat. Adjuk meg a következő megszorításokat sorra vonatkozó CHECK feltételek formájában PC-k sémájára vonatkozóan:

- a) Egy 2.0 GHz-nél kisebb sebességű PC ára nem lehet több 600 dollárnál.
- b) Egy olyan laptopnak, amelynek képernyője kisebb, mint 15 hüvelyk, vagy legalább 40 gigabájtos merevlemezzel kell rendelkeznie, vagy 1000 dollár-nál kevesebbe kell kerülnie.

7.2.5. feladat. Adjuk meg a következő megszorításokat sorra vonatkozó CHECK feltételek formájában a csatahajó sémájára:

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

- a) Egyik hajóosztálynak sem lehetnek 16 hüvelyknél nagyobb kaliberű ágyúi.
- b) Ha egy hajóosztályban az ágyúk száma több mint 9, akkor azok kalibere nem lehet nagyobb 14 hüvelyknél.
- c) Egyik hajó sem csatázhat a felavatása előtt.

! 7.2.6. feladat. A 7.6., illetve 7.8. példákban a FilmSzínész nem attribútumára tettünk megszorításokat. Milyen megkötéssel érhetnénk el, hogy minden megszorítást ki lehessen kényszeríteni, ha a nem értéke NULL?

7.3. Megszorítások módosítása

A megszorításokat bármikor módosíthatjuk, törlhetjük vagy újakat hozhatunk létre. Hogy ezeket a módosításokat pontosan hogyan kell megadnunk, az attól függ, hogy a megszorítás attribútumhoz, táblához vagy adatbázissémához van-e hozzárendelve. Ez utóbbi esetet majd a 7.4. alfejezetben tárgyaljuk.

7.3.1. Megszorítások elnevezése

Ahhoz, hogy módosítani vagy törölni tudjunk egy létező megszorítást, szükséges, hogy a megszorításnak neve legyen. Ezt úgy adhatjuk meg, hogy a megszorítás elő beírjuk a CONSTRAINT kulcsszót és a megszorítás nevét.

7.9. példa. A 2.9. ábra 2. sora helyett, ami azt mondja ki, hogy a név attribútum elsődleges kulcs, a következő írhatjuk:

2) név CHAR(30) CONSTRAINT NévKulcs PRIMARY KEY,

Hasonlóképpen nevezhetjük el a 7.6. példában szereplő attribútumra vonatkozó CHECK feltételt:

4) nem CHAR(1) CONSTRAINT FérfiVagyNő
CHECK (nem IN ('F', 'N')),

Végül pedig a következő módon nevezhetjük el a 7.3. ábra 6. sorában szereplő, sorra vonatkozó CHECK feltételt:

6) CONSTRAINT Titulus
CHECK (nem = 'N' OR név NOT LIKE 'Ms.%');

□

7.3.2. Táblákra vonatkozó megszorítások megváltoztatása

Említettük a 7.1.3. alfejezetben, hogy egy megszorítás ellenőrzését azonnaliról késleltetettre vagy késleltetetről azonnalira állíthatjuk a SET CONSTRAINT utasítással. További módosításokat is tehetünk a megszorításokkal kapcsolatban az ALTER TABLE utasítással. Korábban a 2.3.4. alfejezetben már volt szó az ALTER TABLE utasításról, ott attribútumok hozzáadására, illetve törlésére használtuk ezt az utasítást.

Az ALTER TABLE utasítások többféleképpen is megváltoztathatják a megszorításokat. Egy megszorítás törlése úgy történik, hogy az utasításon belül megadjuk a DROP kulcsszót és a megszorítás nevét. Egy újabb megszorítás megadásához az ADD kulcsszót kell megadnunk, majd magát a megszorítást. Megjegyezzük azonban, hogy a hozzáadott megszorítás olyan típusú kell legyen, amely sorokra vonatkozik, ilyen a soralapú, a kulcs- vagy az idegenkulcs-megszorítás. Sőt megjegyezzük azt is, hogy egy újabb megszorítást csak akkor adhatunk hozzá a táblához, ha a tábla aktuális előfordulása kielégíti a megszorítás feltételét.

7.10. példa. Nézzük meg, hogyan törölhetnénk, majd vihetnénk fel újra a 7.9. példában szereplő megszorításokat a FilmSzínész relációra vonatkozóan. A következő három utasítás töri azokat:

```
ALTER TABLE FilmSzínész DROP CONSTRAINT NévKulcs;
ALTER TABLE FilmSzínész DROP CONSTRAINT FérfiVagyNő;
ALTER TABLE FilmSzínész DROP CONSTRAINT Titulus;
```

Nevezzük el a megszorításokat

Jegyezzük meg, hogy célszerű a megszorításoknak nevet adni még akkor is, ha úgy gondoljuk, hogy soha nem fogunk azokra hivatkozni. Ha egyszer létrehoztuk a megszorítást név nélkül, később már nem fogunk tudni nevet adni neki akkor sem, ha változtatni szeretnénk rajta. Ha azonban egyszer mégis szükségi lesz rá, hogy egy név nélküli megszorítást módosítsunk, azt fogjuk látni, hogy az adatbázis-kezelő lehetővé teszi, hogy megkeressük ezt a megszorítást az összes megszorítások listájából. Mindezt úgy teszi, hogy valójában ezeknek is ad egy belső, saját maga által kiadott nevet, amely nevet használhatunk, ha a megszorításra szeretnénk hivatkozni.

Ha szeretnénk ismét érvényre juttatni a fenti megszorításokat, ezt a FilmSzínész reláció sémájának megváltoztatásával a következőképpen tehetjük meg:

```
ALTER TABLE FilmSzínész ADD CONSTRAINT NévKulcs
    PRIMARY KEY (név);
ALTER TABLE FilmSzínész ADD CONSTRAINT FérfiVagyNő
    CHECK (nem IN ('F', 'N'));
ALTER TABLE FilmSzínész ADD CONSTRAINT Titulus
    CHECK (nem = 'N' OR név NOT LIKE 'Ms.%');
```

Ezek a most újból létrehozott megszorítások mind sorra vonatkozó CHECK feltételek. Attribútumra vonatkozó CHECK feltételek formájában nem is tudnánk ezeket újból létrehozni.

Az újból létrehozott megszorításokra a megszorítás nevének megadása nem kötelező. Az SQL azonban nem jegyzi meg, hogy korábban melyik megszorítás melyik névhez tartozott. Ezért amikor ismét létrehozunk egy korábbi megszorítást, akkor ismét meg kell adnunk a megszorítás teljes leírását, és nem hivatkozhatunk rá csupán csak a korábbi nevével. □

7.3.3. Feladatok

7.3.1. feladat. Mutassuk meg, hogyan kell a filmadatbázis relációsémáit módosítani a következő célok eléréséhez:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevételel)
Stúdió(név, cím, elnökAzon)
```

- a) Adjuk meg a `filmcím` és `év` attribútumokat a `Filmek` tábla kulcsaként.
- b) Követeljük meg azt a hivatkozásiépség-megszorítást, hogy a `Filmek` producereinek szerepelniük kell a `GyártásIrányító` táblában.
- c) Írjuk elő, hogy egy film ne lehessen rövidebb 60 percnél és ne lehessen hosszabb 250 percnél.
- ! d) Zárjuk ki, hogy egy név egyidejűleg előfordulhasson a `filmszínészek` és a `gyártásirányítók` között.
- ! e) Ne engedjük meg, hogy két stúdiónak azonos legyen a címe.

7.3.2. feladat. Mutassuk meg, hogyan kell a csatahajók adatbázissémáit megváltoztatni, hogy abban szerepeljenek a következő sorra vonatkozó CHECK feltételek:

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

- a) A `hajóosztály` és `ország` legyen kulcsa a `Hajóosztályok` relációjának.
- b) Követeljük meg azt a hivatkozásiépség-megszorítást, hogy a `Csaták` táblában szereplő hajók szerepeljenek a `Hajók` táblában is.
- c) Írjuk elő azt a hivatkozásiépség-megszorítást, hogy a `Kimenetelek` táblában szereplő hajók szerepeljenek a `Hajók` táblában is.
- d) Írjuk elő, hogy egy hajónak se lehessen 14-nél több ágyúja.
- ! e) Zárjuk ki, hogy egy hajó a felavatása előtt csatában vehessen részt.

7.4. Önálló megszorítások

Az SQL-beli aktív elemek közül a leghatékonyabbak nincsenek hozzárendelve sem sorokhoz, sem azok komponenseihez. Ezek a triggerek és az önálló megszorítások. Ezek részei az adatbázissémának és a táblákhoz kötődnek.

- Az önálló megszorítás egy logikai értékkal rendelkező SQL-kifejezés, amelynek minden igaznak kell lennie.
- A trigger egy műveletsorozat, amely hozzá van rendelve bizonyos eseményekhez, mint amilyen például egy táblába való beszúrás, és ha az esemény bekövetkezik, akkor a műveletsorozat végrehajtásra kerül.

Az önálló megszorítások használata könnyebb a programozó számára, hiszen itt csak azt kell megmondania, hogy mi az a feltétel, amelynek igaznak kell lennie. Az adatbázis-kezelők mégis inkább a triggert kínálják általános célú aktív elemként. Ennek az az oka, hogy az önálló megszorításokat meglehetősen nehéz hatékonyan megvalósítani. Ez esetben ugyanis az adatbázis-kezelőnek kell kikövetkeztetnie, hogy melyek azok a módosítások, amelyek érinthetik az állítás igazságértékét. A triggerek ezzel szemben pontosan megmondják, hogy mikor kell a rendszernek foglalkoznia velük.

7.4.1. Önálló megszorítások létrehozása

Az SQL-szabvány egy egyszerű formáját javasolja az önálló megszorításoknak, amely bármilyen feltétel ellenőrzését – ami a WHERE után megengedett – lehetővé teszi számunkra. Hasonlóan a többi sémaelemhez, ezeket is egy CREATE utasítással hozzuk létre. Az utasítás formája a következő:

```
CREATE ASSERTION <az önálló megszorítás neve> CHECK (<feltétel>)
```

Egy önálló megszorításban megadott feltételnek minden igaznak kell lennie, és bármilyen adatbázis-módosítást, ami a feltétel megsértését eredményezné, a rendszer visszautasít.¹ Emlékeztetünk rá, hogy az eddig tárgyalt CHECK feltételek bizonyos körülmények között megsérülhetnek, ha azok alkérdéseket is tartalmaznak.

7.4.2. Önálló megszorítások használata

Máshogyan kell megadnunk a sorra vonatkozó CHECK feltételeket és az önálló megszorításokat. A soralapú ellenőrzések direkt módon hivatkozhatnak annak a relációnak az attribútumaira, amelynek deklarációjában szerepelnek. Az önálló megszorítások esetén viszont nincs ilyen jogosultságunk. Ez utóbbiaknál minden, a feltételben szereplő attribútumra magában a megszorításban kell megadnunk, hogy melyik táblában van. Ez általában a táblának egy select-from-where kifejezésben való szerepeltetésével történik.

Mivel a feltételnek egy logikai értéket kell szolgáltatnia, ezért szokásos a feltétel eredményeit valamelyen módon kombinálni, hogy egyetlen igaz vagy hamis értéket kapunk. Például megadhatjuk a feltételt úgy, hogy az egy relációit adó kifejezés legyen, és erre alkalmazzuk a NOT EXISTS műveletet. Vagyis a megszorítás azt jelenti, hogy ez a reláció minden üres. Egy másik lehetőség, hogy valamilyen összesítő műveletet – például a SUM műveletet – alkalmazzuk egy reláció egy oszlopára, és az eredményt összehasonlítjuk egy konstanssal. Ilyen módon előírhatjuk például, hogy az összeg minden kisebb legyen valamelyen korlátnál.

¹ Emlékezzünk azonban vissza a 7.1.3. alfejezetre, amelyben a megszorítás ellenőrzését elhalaszthattuk a hozzá tartozó tranzakció jóváhagyásáig. Ha egy önálló megszorításra is megtehetnénk ugyanezt, akkor röviden mondva a tranzakció végére a feltétele hamissá válhatna.

7.11. példa. Tegyük fel, hogy azt szeretnénk kikötni, hogy senki ne lehessen egy stúdió elnöke, ha a nettó bevétele nem éri el a 10 000 000 dollárt. Létrehozunk egy önálló megszorítást, amelyik azt mondja ki, hogy azon filmstúdiók halmaza, amelyekre az elnök nettó bevétele kisebb, mint 10 000 000 dollár, üres. Ez az önálló megszorítás az alábbi két relációt foglalja magába:

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

Az önálló megszorítást a 7.4. ábrán láthatjuk. □

```
CREATE ASSERTION GazdagElnök CHECK
(NOT EXISTS
  (SELECT *
   FROM Stúdió, GyártásIrányító
   WHERE elnökAzon = azonosító AND
         nettóBevétel < 10000000
  )
);
```

7.4. ábra. Önálló megszorítás, amelyik a stúdióelnökök gazdagságát írja elő

7.12. példa. Vegyük a következő önálló megszorítást, amelyik csak egy relációt, a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

relációt érint. A megszorítás azt mondja ki, hogy egy stúdió által készített filmek összhosszúsága nem haladhatja meg a 10 000 percent.

```
CREATE ASSERTION ÖsszHossz CHECK (10000 >= ALL
  (SELECT SUM(hossz) FROM Filmek GROUP BY stúdióNév)
);
```

A fenti megszorítás csak a **Filmek** relációra mond ki feltételt. Az önálló megszorítás használata helyett ezt kifejezhettük volna egy sorra vonatkozó CHECK feltétellel is. Ehhez a következő sorokat kellett volna a **Filmek** reláció sémadeklarációjához hozzáadnunk:

```
CHECK (10000 >= ALL
  (SELECT SUM(hossz) FROM Filmek GROUP BY stúdióNév));
```

Figyeljük meg, hogy ez utóbbi feltétel tulajdonképpen a **Filmek** tábla minden egyes sorára vonatkozik. A fenti megfogalmazásban egyetlen attribútum neve sem szerepel konkrétan, hanem magát a feltételt az alkérdezés biztosítja.

Megszorítások összehasonlítása

Az alábbi táblázatban összefoglaltuk a legfontosabb különbségeket az attribútumra vonatkozó CHECK feltételek, a sorra vonatkozó CHECK feltételek és az önálló megszorítások között.

<i>Megszorítás típusa</i>	<i>Hol deklaráljuk?</i>	<i>Mikor ellenőrzi a rendszer?</i>	<i>Minden esetben igaz marad?</i>
Attribútumra vonatkozó CHECK feltétel	Az attribútum megadásakor	A relációba való beszúráskor vagy az attribútum módosításakor	Alkérdez esetén nem
Sorra vonatkozó CHECK feltétel	A relációséma megadásakor	A relációba való beszúráskor vagy egy sor módosításakor	Alkérdez esetén nem
Önálló megszorítás	Az adatbázisséma megadásakor	Bármelyik, a feltételben szereplő reláció megváltozása esetén	Igen

Azt is fontos megjegyezni, hogy ha a fenti ellenőrzést sorra vonatkozó CHECK feltétel alkalmazásával valósítanánk meg, akkor a feltételt a rendszer nem ellenőrizné a Filmek táblából való törlés esetén. Ebben a konkrét esetben ez nem is okoz problémát, hiszen ha a megszorítás feltétele a törlés előtt teljesült, akkor biztosan teljesül fog a törlés után is. Ha azonban a megszorítás a filmek összhosszúságára vonatkozóan nem felső korlátot adna meg, ahogyan az most a példában szerepelt, hanem alsó korlátot, akkor a feltétel megsérülhetne, ha azt sorra vonatkozó CHECK feltétel segítségével valósítanánk meg. Önálló megszorítás használata esetén ez nem fordulhat elő. □

Végül megjegyezzük, hogy az önálló megszorításokat el is dobhatjuk. Az e célra szolgáló utasítás az egyéb adatbázissémabeli elemeket eldobó utasítások mintájára adható meg a következőképpen:

DROP ASSERTION <önálló megszorítás neve>

7.4.3. Feladatok

7.4.1. feladat. Adjuk meg a következő önálló megszorításokat. Használjuk a 2.4.1. feladat PC-termékek adatbázissémáját:

Termék(gyártó, modell, típus)

PC(modell, sebesség, memória, merevlemez, cd, ár)

Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)

Nyomtató(modell, szín, típus, ár)

- a) A PC-k gyártói laptopokat is készíthetnek.
- b) A PC-k gyártói legalább akkora sebességű processzorral rendelkező laptopokat készíthetnek, mint a PC-ké.
- c) Ha egy laptopnak több memóriája van, mint egy PC-nek, akkor az ára is legyen magasabb, mint a szóban forgó PC-nek.
- d) Ha a Termék relációban szerepel egy modell és a modell típusa, akkor ez a modell elő kell forduljon a megfelelő típushoz tartozó relációban is.

7.4.2. feladat. Fogalmazzuk meg az alábbiakat önálló megszorításként. Használjuk a 2.4.3. feladat csatahajó-adatbázis sémáját:

Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma, kaliber, vízkiszorítás)

Hajók(név, hajóosztály, felavatva)

Csaták(név, dátum)

Kimenetelek(hajó, csata, eredmény)

- a) Egyik osztályban sem lehet 2-nél több hajó.
- ! b) Egyetlen országnak sem lehet csatahajója és csatacirkálója is egyszerre.
- ! c) Egyetlen 9-nél több ágyúval rendelkező hajó sem süllyedhet el 9-nél kevesebb ágyúval rendelkező hajóval folytatott csatában.
- ! d) Egyetlen hajót sem indíthatnak el az osztályának nevet adó, első hajó előtt.
- ! e) minden osztályra van olyan hajó, amely az osztály nevét viseli.

! 7.4.3. feladat. A 7.11. példában szereplő önálló megszorítás megfogalmazható két sorra vonatkozó megszorításként is. Mutassa meg, hogyan.

7.5. Triggerek

A *triggerek*, amelyeket szokás *esemény-feltétel-művelet szabályoknak* is nevezni, a korábban tárgyalt megszorításoktól három dologban térnek el.

1. A triggereket a rendszer csak akkor ellenőrzi, ha bizonyos, az adatbázis-programozó által megadott *események* bekövetkeznek. A megengedett események általában egy adott relációra vonatkozó beszúrás, törlés és módosítás. Egy másikfajta esemény, amit sok SQL-rendszer megenged, a tranzakció befejeződése.
2. Egy kiváltó esemény bekövetkezésekor a trigger egy *feltételt* vizsgál meg. Ha a feltétel nem teljesül, akkor a kiváltó eseményre válaszul a triggerrel összefüggésben semmi nem fog történni.
3. Ha a trigger feltétele teljesül, akkor a rendszer végrehajtja a triggerhez tartozó *műveletet*. Ez egy lehetséges művelet az esemény hatásának valamilyen módon történő megváltoztatásra. Ez akár az eseményhez tartozó tranzakció elvétése is lehet. A megadott művelet azonban adatbázis-műveleteknek egy tetszőleges sorozata lehet, beleértve az olyan műveleteket is, amelyeknek semmi köze sincs a kiváltó eseményhez.

7.5.1. Az SQL triggerei

Az SQL nyelv triggerutasítása számos különböző lehetőséget kínál a felhasználónak az eseményre, a feltételre és a műveletre vonatkozó részében. Az alábbiakban felsoroljuk a legfontosabbakat.

1. A *triggerfeltétel* ellenőrzésének és a *trigger* műveletének végrehajtása történhet akár a kiváltó esemény végrehajtása előtt létező *adatbázis-állapotban*, azaz a relációk aktuális előfordulására nézve, vagy abban az állapotban, amely a kiváltó esemény után áll fenn.
2. A művelet és a feltétel hivatkozhat a műveletet kiváltó esemény által módosított sorok régi és/vagy új értékeire is.
3. Olyan módosítási esemény is megadható, amely csak egy adott attribútumra vagy attribútumhalmazra vonatkozik.
4. A programozó megadhatja, hogy a művelet végrehajtása a következő két lehetőségből melyik módon történjen meg:
 - a) minden módosított sorra egyszer (*sor szintű trigger*) vagy
 - b) egy SQL-módosító utasítás által módosított összes sorra vonatkozóan egyszer (*utasítás szintű trigger*, emlékeztetve arra, hogy egy SQL módosító utasítás több sorra is hatással lehet).

Mielőtt a triggerek szintaktikai részleteire rátérnénk, nézzünk meg egy példát, amely rávilágít a legfontosabb szintaktikai és szemantikai szempontokra. Figyeljük meg a 7.5. ábrán szereplő példatrigger kulcsfontosságú elemeit, illetve ezeknek az előfordulási sorrendjét:

- a) A CREATE TRIGGER utasítás (1. sor).
- b) Egy záradék, amely megfogalmazza a kiváltó eseményt, illetve azt is, hogy a trigger a kiváltó esemény végrehajtása előtti vagy utáni adatbázis-állapotot fogja használni (2. sor).
- c) Egy REFERENCING záradék, hogy lehetővé tegyük a trigger feltételének és műveletének a módosított sorokra történő hivatkozást (3–5. sorok). Egy ehhez hasonló módosítás esetén ez a záradék lehetővé teszi, hogy a módosítás előtti, illetve utáni soroknak nevet adhassunk.
- d) Egy záradék, amely azt fejezi ki, hogy egy SQL-utasítás esetén a trigger minden módosított sorra egyszer vagy az összes módosított sorra egyszer hajtódjón-e végre (6. sor).
- e) Egy feltétel a WHEN kulcsszó használatával, amely egy logikai kifejezést tartalmaz (7. sor).
- f) Egy vagy több SQL-utasítást tartalmazó művelet (8–10. sorok).

Mindegyik fenti elemnek vannak opciói, amelyeket a példát követően ismertetni fogunk.

7.13. példa. A 7.13. ábrán megadunk egy SQL-beli triggert, amelyik a

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

táblára vonatkozik. A kiváltó esemény a nettóBevétel attribútum módosítása. A trigger hatása az lesz, hogy ha valaki megpróbálja csökkenteni a gyártásirányítók nettó bevételét, akkor a művelet ezt meghiúsítja, és a módosítás előtti állapotot hozza ismét létre.

Az 1. sor a CREATE TRIGGER kulcsszót és a trigger nevét tartalmazza. A 2. sor adja meg a kiváltó eseményt, amely most a GyártásIrányító tábla nettóBevétel attribútumának módosítása. A 3. sortól az 5. sorig terjedő rész lehetővé teszi a triggerbeli feltétel és művelet rész számára, hogy hivatkozni tudjanak a módosítás előtti régi, és a módosítás utáni új értékekre is. A módosítás előtti sorokra RégiSor néven, a módosítás utániakra ÚjSor néven lehet majd hivatkozni, ahogy azt a 4. és 5. sorban láthatjuk. A feltételben és a műveletben ezekre ugyanúgy hivatkozhatunk, mintha azok egy szokásos SQL-lekérdezés FROM záradékában megadott sorváltozók lennének.

A 6. sor FOR EACH ROW fejezi ki azt a követelményt, hogy a műveletek végrehajtása minden egyes sorra külön-külön történjen meg. A 7. sor a triggerfeltétel része. Azt mondja ki, hogy a műveletet csak akkor fogjuk végrehajtani, ha a

```

1) CREATE TRIGGER NetBevétTrigger
2) AFTER UPDATE OF nettóBevétel ON GyártásIrányító
3) REFERENCING
4)     OLD ROW AS RégiSor,
5)     NEW ROW AS ÚjSor
6) FOR EACH ROW
7) WHEN (RégiSor.nettóBevétel > ÚjSor.nettóBevétel)
8)     UPDATE GyártásIrányító
9)     SET nettóBevétel = RégiSor.nettóBevétel
10)    WHERE azonosító = ÚjSor.azonosító ;

```

7.5. ábra. Egy SQL-beli trigger

régi nettó bevétel nagyobb, mint az új, vagyis az illető személy nettó bevétele csökkent.

A 8–10. sorok alkotják a triggerművelet részét. Ezek a sorok egy szabályos SQL-beli UPDATE utasítást alkotnak, amely utasításnak az a hatása, hogy az adott személy nettó bevételét visszaállítja a módosítás előtti értékre. Figyeljük meg, hogy az utasítás a GyártásIrányító tábla összes sorát érintené, de a 10. sorban szereplő WHERE záradék biztosítja, hogy csak az a korábbi módosításban szereplő sor legyen érintve, amelyik a megfelelő azonosító-val rendelkezik. □

7.5.2. A trigger szerkesztésének lehetőségei

Természetesen a 7.13. példa csak néhány sajátosságát mutatja be az SQL triggereinek. A következőkben azokat a lehetőségeket fogjuk felvázolni, amelyeket a triggers kínálnak a számunkra, és megmutatjuk azt is, hogyan lehet ezeket a lehetőségeket kifejezni.

- A 7.5. ábra 2. sora azt mondja ki, hogy a szabályban szereplő feltétel-ellenőrzést, illetve műveletet a kiváltó esemény után kell végrehajtani. Ezt az AFTER kulcsszó jelzi. Az BEFORE helyett a BEFORE kulcsszót is használhatnánk, ez esetben a WHEN feltétel a kiváltó esemény előtt, vagyis a triggert kiváltó módosítás előtt kerülne ellenőrzésre. Ha a feltétel igaz, a trigger művelete(i) végrehajtódnak. Végül ezután hajtódna végre a kiváltó esemény függetlenül attól, hogy a feltétel igaz-e. Van egy harmadik lehetőség is, az INSTEAD OF használata, amelyet a 8.2.3. alfejezetben tárgyalunk a nézetek módosításával kapcsolatban.
- A módosítás (UPDATE) mellett további lehetséges kiváltó események még a beszúrás (INSERT) és a törlés (DELETE). A 7.5. ábra 2. sorában szereplő OF nettóBevétel záradék opcionálisan megadható, és a megadása azt jelenti, hogy csak az OF kulcsszó után felsorolt attribútumok módosítása (UPDATE) számít kiváltó eseménynek. Az OF záradék beszúrás (INSERT) és törlés

(**DELETE**) események esetén nem adható meg, hiszen ezek az események minden teljes sorokra vonatkoznak.

- A **WHEN** megadása opcionális. Ha nem adjuk meg, a művelet minden esetben végrehajtódik, amikor a trigger aktivizálódik. Ha megadtuk, a művelet pontosan akkor hajtódik végre, amikor a **WHEN** feltétele teljesül.
- A példában csak egyetlen SQL-utasítás szerepelt a művelet részben, de megadható ott több ilyen utasítás is pontosvesszőkkel elválasztva, **BEGIN** és **END** közé téve.
- Ha egy sor szintű trigger kiváltó eseménye módosítás, akkor a módosítás előtti sort régi sornak, a módosítás után pedig új sornak tekinthetjük. Ezeknek a soroknak nevet adhatunk az **OLD ROW AS**, illetve a **NEW ROW AS** záradékok segítségével, ahogyan az a 7.5. ábra 4. és 5. soraiban látható. Ha a kiváltó esemény beszúrás, akkor a beszúrt sornak a **NEW ROW AS** záradék segítségével adhatunk nevet, az **OLD ROW AS** záradék viszont ilyenkor nem használható. Törlés esetén az **OLD ROW AS** záradék segítségével nevezhetjük el a törölt sort, ilyenkor viszont a **NEW ROW AS** záradék nem használható.
- Ha elhagyjuk vagy kicseréljük az alapértelmezett **FOR EACH STATEMENT** kulcsszóra a 6. sorban szereplő **FOR EACH ROW** megkötést, akkor egy sor szintű triggerből (mint amilyen például a 7.5. ábrán is szerepel) utasítás szintű trigger lesz. Egy utasítás szintű trigger egyszer hajtódik végre kiváltó utasításonként, függetlenül attól, hogy a kiváltó utasítás hány sort érint (nullát, egyet vagy többet). Ha például egy egész táblát módosítunk egy SQL-beli **UPDATE** utasítással, akkor egy utasítás szintű trigger egyszer fog végrehajtódni, míg egy sor szintű trigger minden módosított sorra végrehajtásra kerül.
- Egy utasítás szintű triggerben nem hivatkozhatunk közvetlenül a régi és az új sorokra, ahogy azt a 4. és 5. sorban láttuk. Bármely triggerben (utasítás és sor szintűben is) hivatkozhatunk azonban a *régi sorok* relációjára és az *új sorok* relációjára. Az előbbi a törölt sorokat, illetve a módosított sorok régi verziót jelenti, az utóbbi pedig a beszúrt sorokat, illetve a módosított sorok új verziót jelenti. A deklaráció formája ez esetben **OLD TABLE AS RégiAdat** és **NEW TABLE AS ÚjAdat**.

7.14. példa. Tegyük fel, hogy azt szeretnénk megakadályozni, hogy a gyártás-irányítók nettó bevételének átlaga 500 000 dollár alá csökkenjen. Ezt a

GyártásIrányító(név, cím, azonosító, nettóBevétel)

táblára vonatkozó megszorítást egy beszúrás, egy törlés vagy a **nettóBevétel** oszlop módosítása sértheti meg. Vigyáznunk kell, hiszen egy beszúrás vagy módosítás sok sort érinthet. A módosítások végrehajtása közben az átlag átmene-

tileg 500 000 dollár alá csökken, majd az összes módosítás elvégzése után ismét

afölé megy. Csak azokat a módosításokat szeretnénk visszautasítani, amelyeknek a teljes lefutása után az átlag a megadott határ alá kerül.

A GyártásIrányító táblának minden a három típusú eseményére (beszúrás, módosítás, törlés) egy trigger kell írnunk. A 7.6. ábrán a módosításra vonatkozó trigger láthatjuk. A beszúrásra és a törlésre vonatkozó triggers hasonlóak.

```

1) CREATE TRIGGER ÁttagNetBevétTrigger
2) AFTER UPDATE OF nettóBevétel ON GyártásIrányító
3) REFERENCING
4)      OLD TABLE AS RégiAdat,
5)      NEW TABLE AS ÚjAdat
6) FOR EACH STATEMENT
7) WHEN (500000 > (SELECT AVG(nettóBevétel) FROM
                           GyártásIrányító))
8) BEGIN
9)     DELETE FROM GyártásIrányító
10)    WHERE (név, cím, azonosító, nettóBevétel) IN ÚjAdat;
11)    INSERT INTO GyártásIrányító
12)        (SELECT * FROM OldStuff);
13) END;
```

7.6. ábra. Az átlagos nettó bevétel megszorítása

A 3–5. sorok azt adják meg, hogy annak a relációt a neve, amelyik a régi sorokat tartalmazza: **RégiAdat**, annak a neve pedig, amelyik az új sorokat tartalmazza: **ÚjAdat**. Itt most az, hogy régi, illetve új, azzal az adatbázis-művelettel kapcsolatban értendő, amelyik kiváltotta a triggeret. Egy adatbázis-művelet egy relációt több sorát is módosíthatja, ezért egy ilyen művelet végrehajtása után a **RégiAdat** és az **ÚjAdat** tábláknak nagyon sok sora is lehet.

Ha a kiváltó művelet módosítás, akkor a **RégiAdat** tábla a sorok módosítás előtti régi verzióját, az **ÚjAdat** tábla pedig a sorok módosítás utáni új verzióját tartalmazza. Ha törlés műveletre írnánk egy hasonló triggeret, abban az esetben a törölt sorok lennének a **RégiAdat** táblában, és az **ÚjAdat** táblára vonatkozó deklaráció nem szerepelne a trigger leírásában. Beszúrás műveletre megírt triggerben pedig a beszúrt sorok lennének az **ÚjAdat** táblában, és a **RégiAdat** táblára vonatkozó deklaráció maradna el.

A 6. sor azt mondja ki, hogy a trigger utasításoknál egyszer hajtódjon végre, függetlenül a módosított sorok számától. A 7. sor a feltétel. Ez akkor válik igazzá, ha a módosítás után az átlagos nettó bevétel kisebb lesz 500 000 dollárnál.

A 8–13. sorok két utasítást tartalmaznak, amelyek a régi GyártásIrányító táblát állítják vissza, ha a WHEN feltétel igaz lesz, vagyis amikor az átlagos nettó bevétel túl alacsonnyá válik. A 9–10. sor kitörli az új sorokat, vagyis a módosított sorok új verzióit, a 11–12. sorok pedig ismét beszúrják a módosítás előtti sorokat. □

7.15. példa. A BEFORE triggerek egy fontos alkalmazása, amikor egy beszúrandó sort a beszúrás előtt valamilyen módon a megfelelő formára hoznak. Tegyük fel, hogy filmsorokat kívánunk beszúrni a

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
```

táblába, de nem minden ismerjük a film kiadásának évét. Mivel az év a kulcsnak egy attribútuma, ezért értéke nem lehet NULL. Egy trigger segítségével viszont biztosíthatjuk, hogy az év értéke ne legyen NULL. A trigger a NULL helyett valamilyen megfelelő értéket ír be, természetesen ez akár egy bonyolult módon számított érték is lehet. A 7.7. ábrán szereplő trigger a NULL értéket egyszerűen az 1915-tel helyettesíti (ez lehetne akár egy alapértelmezett érték is, de most példának is megteszí).

A 2. sorban szerepel a beszúrás előtt lefutó feltétel és művelet. A 3–5. sorokban, a hivatkozási záradékban definiáljuk a beszúrandó új sor, illetve a – csak ezt a sort tartalmazó – tábla nevét is. Még akkor is, ha a trigger minden beszúrandó sorra csak egyszer fut le (mivel a 6. sor sor szintű triggerként határozza meg a triggert), a 7. sorban lévő feltételnek tudnia kell hivatkozni a beszúrandó sor egy attribútumára, emellett a 8. sorban lévő műveletnek pedig egy táblára kell tudnia hivatkozni a módosítás leírásához. □

```

1) CREATE TRIGGER ÉvJavítóTrigger
2) BEFORE INSERT ON Filmek
3) REFERENCING
4)      NEW ROW AS újSor
5)      NEW TABLE AS újÉrték
6) FOR EACH ROW
7) WHEN újSor.év IS NULL
8) UPDATE újÉrték SET év = 1915;
```

7.7. ábra. A beszúrt sorok NULL értékeinek helyettesítése

7.5.3. Feladatok

7.5.1. feladat. Írjuk meg a 7.6. ábra triggeréhez hasonló SQL-beli triggereket a GyártásIrányító tábla beszúrás és törlés műveletére.

7.5.2. feladat. Írjuk meg a következőket, mint triggereket. Egyik esetben se engedjük meg a módosítás végrehajtását, ha az nem elégíti ki a megadott megszorítást. Az adatbázisséma a 2.4.1. feladat PC adatbázisának megfelelő.

```
Termék(gyártó, modell, típus)
```

```
PC(modell, sebesség, memória, merevlemez, cd, ár)
```

```
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
```

```
Nyomtató(modell, szín, típus, ár)
```

- a) Amikor egy PC árát módosítjuk, ellenőrizzük, hogy nincs ugyanolyan sebességű, de kisebb árú PC.
- b) Amikor egy új nyomtatót viszünk fel, ellenőrizzük, hogy az adott modellszám benne van a **Termék** táblában.
- ! c) A **Laptop** reláció bármilyen módosítása esetén ellenőrizzük, hogy a laptopok átlagos ára gyártónként legalább 1500 dollár legyen.
- ! d) Amikor egy PC memóriáját vagy merevlemezét módosítjuk, ellenőrizzük, hogy a módosított PC-nek legalább 100-szor akkora a merevlemeze, mint a memóriája.
- ! e) Amikor egy új PC-t, laptopot vagy nyomtatót viszünk fel, ellenőrizzük, hogy az adott modellszám még nem szerepel a **PC**, **Laptop** vagy **Nyomtató** táblában.

7.5.3. feladat. Írjuk meg a következő feladatokat triggerek formájában. Egyik esetben se engedjük meg a módosítás végrehajtását, ha az nem elégíti ki a megadott megszorítást. Az adatbázisséma a 2.4.3. feladat csatahajó adatbázisának megfelelő.

Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma, kaliber, vízkiszorítás)

Hajók(név, hajóosztály, felavatva)

Csaták(név, dátum)

Kimenetelek(hajó, csata, eredmény)

- a) Ha egy új hajóosztályt viszünk fel a **Hajóosztályok** táblába, vigyük fel egy hajót is, amelyiknek ugyanaz a neve, mint a hajóosztálynak, és a felavatási dátuma legyen NULL érték.
- b) Ha egy új hajóosztályt viszünk fel, amelynek a vízkiszorítása nagyobb 35 000 tonnánál, akkor engedjük meg a beszűrást, de a vízkiszorítást változtassuk 35 000 tonnára.
- ! c) Ha egy új sort viszünk fel a **Kimenetelek** táblába, ellenőrizzük, hogy a hajó és a csata benne van-e a **Hajók**, illetve a **Csaták** táblában, és ha nincs, akkor szűrjük be a megfelelő sort egyik vagy minden hajóosztályba, a hiányzó oszlopokba NULL értéket téve.
- ! d) Ha új sorokat viszünk fel a **Hajók** táblába, vagy a tábla **hajóosztály** attribútumát módosítjuk, ellenőrizzük, hogy egy országnak se lehessen több mint 20 hajója.
- !! e) Ellenőrizzük minden lehetséges esetben, ami a feltételt megsértheti, hogy egy hajó nem vehet részt egy csatában, ha egy korábbi dátumú csatában elsüllyesztették.

! 7.5.4. feladat. Írjuk meg a következő feladatokat triggerek vagy önálló megszorítások formájában. Egyik esetben se engedjük meg a módosítás végrehajtását, ha az nem elégíti ki a megadott megszorítást. A feladatok a filmadatbázis tábláira vonatkoznak.

Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)

SzerepelBenne(filmCím, filmÉv, színészNév)

FilmSzínész(név, cím, nem, születésiDátum)

GyártásIrányító(név, cím, azonosító, nettóBevétel)

Stúdió(név, cím, elnökAzon)

Feltételezhetjük, hogy a kívánt feltétel minden esetben teljesül, mielőtt megpróbáljuk módosítani az adatbázist. Ha lehetséges, akkor az utasítás visszautasítása helyett inkább módosítsuk az adatbázist, még akkor is, ha ez NULL értékek vagy alapértelmezés szerinti értékek felvitelével jár.

- a) Biztosítsuk azt a feltételt, hogy a SzerepelBenne relációban lévő színészek benne vannak a FilmSzínész relációban is.
- b) Biztosítsuk azt a feltételt, hogy minden gyártásirányító egyben vagy egy stúdió elnöke, vagy egy film producere, vagy mindkettő egyidejűleg.
- c) Biztosítsuk azt a feltételt, hogy minden filmnek legalább egy férfi és egy női szereplője is van.
- d) Biztosítsuk azt a feltételt, hogy egy stúdió egy évben nem készíthet száznál több filmet.
- e) Biztosítsuk azt a feltételt, hogy az ugyanabban az évben készült filmek hosszának átlaga nem lehet több mint 120 perc.

7.6. Összefoglalás

- ◆ *Hivatkozásiépség-megszorítások:* Előírhatjuk, hogy egy attribútum vagy egy attribútumhalmaz értékeinek elő kell fordulnia a reláció vagy egy másik reláció valamelyik sorának megfelelő attribútumában vagy attribútumaiban. Ezt a relációséma megadásakor a REFERENCES vagy a FOREIGN KEY kulcsszóval adhatjuk meg.
- ◆ *Attribútumra vonatkozó CHECK feltételek:* Egy attribútum értékeire vonatkozó megszorítást előírhatunk úgy, hogy a CHECK kulcsszót és az ellenőrizendő feltételt megadjuk a relációsémaban az attribútum deklarációja után.
- ◆ *Sorra vonatkozó CHECK feltételek:* Egy reláció soraira megadhatunk ellenőrizendő feltételeket úgy, hogy a reláció deklarációja után megadjuk a CHECK kulcsszót és a feltételt.

8. fejezet

Nézetek és indexek

Ezt a fejezetet a nézettáblák ismertetésével kezdjük. A nézettábla olyan reláció, melyet más relációkra vonatkozó lekérdezésekkel definiálunk. A nézettáblák az adatbázisban nincsenek tárolva, de ugyanúgy lekérdezhetők, mintha létező táblák lennének. A nézettáblára vonatkozó lekérdezés végrehajtásakor a lekérdezésfeldolgozó a nézettábla definícióját felhasználva látszólagosan előállítja a nézettáblát.

A nézettáblák tárolhatók is abban az értelemben, hogy az adatbázisból periodikusan előállíthatók és tárolhatók is. A tárolt nézettáblákkal a lekérdezések végrehajtási sebessége növelhető. A tárolt nézettáblák nagyon fontos speciális típusa az index, ami olyan tárolt adatstruktúra, amelynek egyedüli célja a tárolt relációk meghatározott sorai elérésének felgyorsítása. E fejezetben be fogjuk mutatni az indexeket, és áttekintjük a tárolt táblákhoz a megfelelő index kiválasztásának legfontosabb kérdéseit.

8.1. Nézettáblák

A CREATE TABLE utasítással definiált táblák fizikailag léteznek az adatbázisban, azaz az adatbázisrendszer valamilyen fizikai struktúrában tárolja őket. Megtartják az állapotukat, azaz nem változnak addig, amíg valamilyen táblamódosító SQL-utasítás meg nem változtatja őket.

Létezik az SQL-relációknak egy másik típusa, amit *nézettáblának* nevezünk, amelyek nem léteznek fizikailag az adatbázisban. A nézettáblákat a lekérdezéshez hasonló kifejezés segítségével definiáljuk. A nézettáblákat ugyanúgy lekérdezhetjük, mint a fizikailag létező táblákat, és egyes esetekben még módosíthatjuk is őket.

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, amelyek valamely lekérdezés eredményeképpen jöttek létre. Ezekről is „relációként” fogunk említeni.

8.1.1. Nézettáblák létrehozása

A legegyszerűbb mód egy nézettábla létrehozására a következő:

```
CREATE VIEW <név> AS <definíció>;
```

A nézettábla definíciója egy SQL-lekérdezés.

8.1. példa. Tételezzük fel, hogy egy olyan nézettáblát szeretnénk, amely a

```
Filmek(filmcím, év, hossz, műfaj, stúdiónév, producerAzon)
```

reláció egy részét specifikálja, pontosabban a Paramount stúdió által gyártott filmek címét és gyártási évét. Ezt a nézettáblát a következőképpen definíálhatjuk:

- 1) CREATE VIEW ParamountFilmek AS
- 2) SELECT filmcím, év
- 3) FROM Filmek
- 4) WHERE stúdiónév = 'Paramount';

Az 1. sorban látható, hogy a nézettábla neve `ParamountFilmek`. A nézettábla attribútumait a 2. sor tartalmazza: `filmcím` és `év`. A nézettábla definíciója a 2-4. sorok között található meg. □

8.2. példa. Nézzünk egy példát bonyolultabb lekérdezéssel definiált nézettáblára. A célunk a `FilmProducer` nézettábla létrehozása, amely a filmek címét és a producereik nevét tartalmazza. Ezt a nézettáblát a következő két reláció felhasználásával tudjuk definiálni:

```
Filmek(filmcím, év, hossz, műfaj, stúdiónév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A nézettábla definíciója:

```
CREATE VIEW FilmProducer AS
    SELECT Filmek.filmcím, név
    FROM Filmek, GyártásIrányító
    WHERE producerAzon = azonosító;
```

Fenti definíció két relációt kapcsol össze és az azonosítók egyezését követeli meg. A film címét és a producer nevét a megegyező azonosítójú sorpárokból állítja elő. □

8.1.2. Nézettáblák lekérdezése

A nézettáblák pontosan ugyanúgy kérdezhetők le, mint a ténylegesen tárolt alaptáblák. A **FROM** záradékban megadjuk a nézettábla nevét, és a nézettábla definícióját használva az adatbázisrendszerre bízzuk, hogy előállítsa a lekérdezéshez szükséges sorokat.

8.3. példa. Úgy kérdezhetjük le a **ParamountFilmek** relációt, mintha egy tárolt reláció lenne:

```
SELECT filmcím
FROM ParamountFilmek
WHERE év = 1979;
```

Ezzel a lekérdezéssel a Paramount által 1979-ben gyártott filmek címeit kapjuk meg. □

8.4. példa. Írhatunk olyan lekérdezéseket is, amelyek nézettáblákat és alaptáblákat is tartalmaznak. Ilyen például a következő:

```
SELECT DISTINCT színészNév
FROM ParamountFilmek, SzerepelBenne
WHERE filmcím = filmCím AND
      ParamountFilmek.év = SzerepelBenne.filmÉv;
```

Ez a lekérdezés a Paramount által gyártott filmekben szereplő összes színész nevét megadja. □

Azt, hogy mit is jelent a nézettáblák használata, a legegyszerűbben úgy tudjuk megmutatni, hogyha a **FROM** záradékban a nézettáblákat kicséréljük a nézettáblákat definiáló alkérdéssel. Az alkérést követő sor változóval tudunk az alkérdéssel létrehozott tábla soraira hivatkozni. Például a 8.4. példa lekérdezésének eredménye megegyezik a 8.1. ábrán látható lekérdezés eredményével.

```

SELECT DISTINCT színészNév
FROM (SELECT filmcím, év
      FROM Filmek
      WHERE stúdióNév = 'Paramount',
    ) Pm, SzerepelBenne
WHERE Pm.filmcím = filmCím AND Pm.év = SzerepelBenne.filmÉv;

```

8.1. ábra. A nézettábla használatának alkérdezéssel való megvalósítása

8.1.3. Attribútumok átnevezése

Néha szeretnénk más attribútumneveket használni a nézettábla definíójában, mint amelyek a nézettáblát definiáló lekérdezésből adódnak. A nézettábla attribútumait megadhatjuk egy zárójelek közé írt lista formájában, amelyet a CREATE VIEW utasításban a nézettábla neve után kell írni. Például a 8.2. példa nézettáblájának definícióját a következőképpen lehetne átírni:

```

CREATE VIEW FilmProducer(filmCím, producerNév) AS
  SELECT Filmek.filmcím, név
  FROM Filmek, GyártásIrányító
  WHERE producerAzon = azonosító;

```

A nézettábla ugyanaz lesz, attól eltekintve, hogy az oszlopnevek nem filmcím és név, hanem filmCím és producerNév lesznek.

8.1.4. Feladatok

8.1.1. feladat. A következő alaptáblákból kiindulva:

```

FilmSzínész(név, cím, nem, születésiDátum)
GyártásIrányító(név, cím, azonosító, nettóBevételek)
Stúdió(név, cím, elnökAzon)

```

Építsük fel a következő nézettáblákat:

- Egy **GazdagProducer** nézettáblát, amely a legalább 10 000 000 \$ nettó bevételű gyártásirányítók nevét, címét, azonosító számát és bevételét adj meg.
- Egy **StúdióElnök** nézettáblát, amely azon gyártásirányítók nevét, címét és azonosító számát tartalmazza, akik stúdióelnökök is.
- Egy **ProdSzínész** nézettáblát, amely azon személyek nevét, címét, nemét, születési dátumát, azonosító számát és nettó bevételét tartalmazza, akik egyben gyártásirányítók és színészek is.

8.1.2. feladat. A 8.1.1. feladat nézettábláit használva (alaptáblák használata nélkül) adjuk meg a következő lekérdezéseket:

- 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 \$.

8.2. Adatok módosítása nézettáblákon keresztül

Korlátosztott módon lehetséges beszűrni, törlni vagy változtatni az adatokat egy nézettáblán. 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 beszűrást. Eléggé egyszerű nézettáblák esetén azonban a nézettábla módosítását átalakíthatjuk az alaptábla módosításává, amely ugyanolyan hatású lesz, mintha a nézettáblát módosítanánk. Az ilyen nézettáblákat *módosítható nézettáblák*nak nevezzük. Az „instead of” (a kiváltó esemény helyett működő) trigger is használható arra, hogy a nézettábla módosítását az alaptábla módosításával végezze el. Ezzel a programozó meg tudja határozni, hogy a nézettáblán kereszttüli módosítás hogyan történjen.

8.2.1. Nézettábla megszüntetése

A nézettábla különleges módosításának tekinthető a nézettábla megszüntetése. Ezt a „változtatást” akkor is végre lehet hajtani, ha a nézettábla nem módosítható. A megszüntetési utasítás:

```
DROP VIEW ParamountFilmek;
```

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 Filmek
```

nemcsak a *Filmek* táblát szünteti meg, hanem a *ParamountFilmek* nézettáblát is használhatatlanná teszi, mivel az egy már nem létező *Filmek* relációra hivatkozik.

8.2.2. Módosítható nézettáblák

Az SQL-szabvány formálisan leírja, mikor lehet egy nézettáblát módosítani és mikor nem. Az SQL-szabályok bonyolultak, de nagyjából azt engedik meg, hogy egy R reláción (amely szintén lehet módosítható nézettábla) definiált nézettáblát módosíthassunk, ha definíciójában a SELECT után DISTINCT nem szerepel. Két fontos kikötés van:

- A WHERE záradékban R nem szerepelhet egy alkérdezésben sem.
- A FROM záradékban csak R szerepelhet, csak egyszer fordulhat itt elő és semmilyen más reláció nem szerepelhet (a FROM záradékban).
- A SELECT záradék listája elég attribútumot kell tartalmazzon ahhoz, hogy egy beszúrás esetén a többi attribútumot nullértékkel, vagy az alapértelmezett értékkel tölthessük fel az alaptáblában. Például kötelező az olyan attribútumérték megadása, mely NOT NULL-nak van deklarálva és nincs alapértelmezett értéke.

A nézettáblába való beszúrás közvetlenül a nézettábla R alaptáblájába történik. Az egyetlen finom különbség az, hogy csak azon attribútumoknak tudunk így értéket adni, amelyek a nézettáblát definiáló SELECT záradékban előfordulnak.

8.5. példa. Tételezzük fel, hogy 8.1. példában definiált ParamountFilmek nézettáblába szeretnénk egy sort beszúrni a következő módon:

```
INSERT INTO ParamountFilmek
VALUES('Csillagok háborúja', 1979);
```

A ParamountFilmek nézettábla megfelel az SQL módosíthatósági feltételeinek, mivel a következő alaptábla néhány attribútumára vonatkozik:

```
Filmek(filmcím, év, műfaj, hossz, stúdióNév, producerAzon)
```

A ParamountFilmek nézettáblába való beszúrás a Filmek táblába való következő beszúrásként kerül végrehajtásra:

```
INSERT INTO Filmek(filmcím, év)
VALUES('Csillagok háborúja', 1979);
```

Megjegyezzük, hogy a nézettábla definíciója miatt ebben a beszúrásban csak a filmcím és év attribútumok kaphattak értéket, a Filmek tábla többi attribútumainak nem tudtunk értéket adni.

A Filmek táblába beszúrandó sor attribútumértékei tehát: a filmcím attribútumértéke 'Csillagok háborúja', az év attribútumértéke 1979, az összes többi attribútumértéke NULL. Minthogy a beszúrt sor stúdióNév attribútumának értéke NULL, és ez a ParamountFilmek nézettábla válogatási feltételének nem felel meg, így a beszúrt sornak a nézettáblára nincs hatása. Így például

a 8.3. példa lekérdezése a most beszúrt sort ('Csillagok háborúja', 1979) nem fogja figyelembe venni.

Ezen anomália elkerülése végett a nézettábla definiálásakor a SELECT záradékban stúdióNév attribútumot is szerepettelnünk kell:

```
CREATE VIEW ParamountFilmek AS
    SELECT stúdióNév, filmcím, év
    FROM Filmek
    WHERE stúdióNév = 'Paramount';
```

Ezután a ParamountFilmek nézettáblába beszúrjuk a kívánt sort:

```
INSERT INTO ParamountFilmek
VALUES('Paramount', 'Csillagok háborúja', 1979);
```

Ez a beszúrás a Filmek táblára ugyanúgy hat, mint a következő:

```
INSERT INTO Filmek(stúdióNév, filmcím, év)
VALUES('Paramount', 'Csillagok háborúja', 1979);
```

Megjegyezzük, hogy bár a létrehozott sor a nem említett attribútumokon NULL értéket kap, de a ParamountFilmek nézettáblában éppen a beszúrás által megkívánt sort fogja eredményezni. □

A módosítható nézettáblából törölhetünk is. A törlés, a beszúráshoz hasonlóan, az alaptáblában történik. Azért, hogy biztosítsuk azt, hogy csak azok a sorok töröljenek, amelyek a nézettáblában láthatóak, a törlő utasítás WHERE záradékához „és” (AND) logikai művelettel hozzá kell kapcsolnunk a nézettáblát definiáló utasítás WHERE záradékában szereplő feltételt.

8.6. példa. Tételezzük fel, hogy a módosítható ParamountFilmek nézettáblából szeretnénk kitörölni az összes olyan filmet, amelyek címe tartalmazza a „háború” szót. Ezt a következő utasítás segítségével érhetjük el:

```
DELETE FROM ParamountFilmek
WHERE filmcím LIKE '%háború%';
```

A törlés átalakul egy, a Filmek táblára vonatkozó törlésre azzal a különbséggel, hogy a ParamountFilmek nézettáblát definiáló lekérdezés WHERE feltétele hozzáadódik a törlés WHERE feltételéhez:

```
DELETE FROM Filmek
WHERE filmcím LIKE '%háború%' AND stúdióNév = 'Paramount';
```

a művelet eredménye. □

Hasonlóképpen, a módosítások is az alaptáblán keresztül történnek. A nézettábla módosításainak tehát az a hatása, hogy az alaptábla azon sorait módosítja, amelyek a nézettáblában sort eredményeznek.

Miért nem módosíthatók egyes nézettáblák?

Tekintsük a 8.2. példában előforduló `FilmProducer` nézettáblát, amely a filmcímet párosítja a gyártásirányítókkal. Ez a nézettábla az SQL definíciója szerint nem módosítható, mivel két relációra vonatkozik: a `Filmek` és a `GyártásIrányító` relációkra. Tételezzük fel, hogy szeretnénk beszúrni a következő sort:

(`'Indiana Jones és a haláltemplom'`, `'George Lucas'`)

A `Filmek` és a `GyártásIrányító` relációkba is be kellene szúrni egy-egy sort. Az olyan attribútumokra, mint például a `hossz` vagy a `filmcím`, használhatjuk az alapértelmezett értéket, de mit csinálunk 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 két `NULL` értéket nem tekint egyenlőnek (lásd 6.1.6. alfejezetet). Tehát az `'Indiana Jones és a haláltemplom'` és a `'George Lucas'` nem kapcsolódna össze a `FilmProducer` nézettáblában, tehát a beszúrás nem lenne helyesen véghajtva.

8.7. példa. A következő, nézettáblában történő módosítás:

```
UPDATE ParamountFilmek
SET év = 1979
WHERE filmcím = 'Csillagok háborúja film';
```

az alaptáblában ilyen alakban megy végre:

```
UPDATE Filmek
SET év = 1979
WHERE filmcím = 'Csillagok háborúja film' AND
      stúdióNév = 'Paramount';
```

□

8.2.3. Nézettáblákra vonatkozó „helyette” (instead-of) típusú trigerek

Ha nézettáblára vonatkozó triggert definiálunk, akkor a `BEFORE` és `AFTER` helyett az `INSTEAD OF`-ot kell használnunk. Ha így járunk el, akkor ha egy esemény kiváltja a trigger működését, akkor a triggerben megadott műveletek futnak le a trigger működését kiváltó művelet helyett. Így az instead-of típusú trigger elfogadja a nézettábla módosítási kísérletet, és helyette az adatbázis tervezője által helyesnek gondolt műveletet hajtja végre. A következő egy tipikus példa erre.

8.8. példa. Idézzük fel a Paramount által készített filmek nézettáblájának 8.1. példában szereplő definícióját:

```
CREATE VIEW ParamountFilmek AS
    SELECT filmcím, év
    FROM Filmek
    WHERE stúdióNév = 'Paramount';
```

Amint a 8.5. példában megmutattuk, ez a nézettábla módosítható, de nem ki-vánt következménnyel jár, ha ugyanis a ParamountFilmek nézettáblába beszú-runk egy sort, akkor a rendszer nem tudja kikövetkeztetni, hogy az alaptáblába történő beszúráskor a stúdióNév attribútum helyes értéke Paramount lesz-e, így a stúdióNév NULL értéket kap.

Jobb megoldást érhetünk el, ha egy erre a táblára vonatkozó, a 8.2. ábrán látható, „instead-of” típusú triggeret hozunk létre. A triggerben nincs semmi meglepő. A 2. sorban látjuk az INSTEAD OF kulcsszót, amellyel azt érjük el, hogy a ParamountFilmek nézettáblába való beszúrás nem jön létre, helyette ez a trigger fog működni.

- 1) CREATE TRIGGER ParamountInsert
- 2) INSTEAD OF INSERT ON ParamountFilmek
- 3) REFERENCING NEW ROW AS ÚjSor
- 4) FOR EACH ROW
- 5) INSERT INTO Filmek(filmcím, év, stúdióNév)
- 6) VALUES(ÚjSor.filmcím, ÚjSor.év, 'Paramount');

8.2. ábra. A nézettáblába való beszúrást az alaptáblán történő beszúrásra cserélő trigger

A végrehajtani kívánt beszúrás helyett az 5. és 6. sorokban megadott akció következik be. Ez a Filmek táblába történő beszúrás, melyben a három ismert attribútumot adjuk meg. A filmcím és az év attribútumok a nézettáblába beszúrni kívánt sorból származnak; ezen értékekre az ÚjSor sorváltozó névvel hivatkoztunk, melyet a 3. sorban deklaráltunk azért, hogy meg tudjuk nevezni azt a sort, amelyet a nézettáblába kíséreltünk meg beszúrni. A stúdióNév attribútum értéke a 'Paramount' konstans. Ez nem a nézettáblába beszúrni kísérelt sor része, hanem abból a feltételezésünkben következik, hogy a beszúrás a ParamountFilmek nézeten keresztül érkezett. □

8.2.4. Feladatok

8.2.1. feladat. Melyek módosíthatók a 8.1.1. feladat nézettáblái közül?

8.2.2. feladat. Tegyük fel, hogy megkonstruáltuk a következő nézettáblát:

```
CREATE VIEW DisneyVígjátékok AS
    SELECT filmcím, év, hossz FROM Filmek
    WHERE stúdióNév = 'Disney' AND műfaj = 'vígjáték';
```

- a) Módosítható-e ez a nézettábla?
- b) Készítse el az ezen nézettáblába való beszúrást kezelő instead-of típusú triggert.
- c) Készítsen instead-of típusú triggert, amely ezen nézettáblán keresztül a (címmel és évvel megadott) film hosszának módosítását oldja meg.

8.2.3. feladat. A következő alaptáblákra vonatkozóan:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, ár)
```

megkonstruáltuk a következő nézettáblát:

```
CREATE VIEW ÚjPC AS
    SELECT gyártó, modell, sebesség, memória, merevlemez, ár
    FROM Termék, PC
    WHERE Termék.modell = PC.modell AND típus = 'pc';
```

Megjegyezzük, hogy konzisztencia-ellenőrzést is használunk: a modellszám nemcsak a PC-relációban fordul elő, hanem a Termék reláció típus attribútuma jelzi, hogy a szóban forgó termék PC.

- a) Módosítható-e ez a nézettábla?
- b) Készítse el az ezen nézettáblába való beszúrást kezelő instead-of típusú triggert.
- c) Készítsünk instead-of típusú triggert, amely ezen nézettáblán keresztül az ár módosítását oldja meg.
- d) Készítsen instead-of típusú triggert, amely ezen nézettáblán keresztül adott sor törlését oldja meg.

8.3. Indexek az SQL-ben

A reláció A attribútumára épített index egy olyan adatstruktúra, amely a reláció olyan sorainak megkeresését teszi hatékonyabbá, melyekben az A attribútum értéke valamilyen meghatározott érték. Az indexre úgy is gondolhatunk, mint (kulcs, érték) párokra vonatkozó bináris keresőfára, ahol az a kulcs (az A attribútum egy lehetséges értéke) van társítva az „értékkel”, amely az olyan sorok helyének halmaza, mely sorokban az A attribútum értéke a . Az ilyen index az

- vi) p_2 a valószínűsége a `SELECT * FROM Hajók WHERE osztály = o` típusú lekérdezés használatának.
- vii) p_3 a valószínűsége a `SELECT * FROM Hajók WHERE felavatva = é` típusú lekérdezés használatának.
- viii) $1 - p_1 - p_2 - p_3$ a valószínűsége annak, hogy a `Hajók` relációba új sort szűrünk be.

Az indexek elérésére és a beszúráshoz szükséges szabad hely megtalálására vonatkozóan legyenek azok a feltételezések, amiket a 8.14. példában is alkalmaztunk.

Tegyük fel, hogy a `név`, `osztály` és `felavatva` attribútumokra építünk indexeket. Az indexek minden kombinációjára becsüljük meg a műveletek költségét. A p_1 , p_2 és p_3 függvényében hogyan válasszuk meg legjobban az indexeket?

8.5. Tárolt nézettáblák

A nézettáblát létrehozó utasítás leírja, hogy az adatbázis alaptábláiból, ezen táblákon végrehajtott lekérdezéssel, hogyan hozhatunk létre új táblákat. Eddig úgy tekintettük, hogy a nézettáblák csak a relációk logikai leírásai. Ugyanakkor, ha egy nézettáblát elég gyakran használnak, akkor hatékonyabb lehet, ha a nézettáblát ténylegesen *tároljuk*; azaz a tartalmát állandóan megtartjuk. Ahogy az indexek fenntartása is, úgy a tárolt nézettáblák fenntartása is költséggel jár, hiszen az alaptáblák minden módosítását a tárolt nézettáblákban is követni kell.

8.5.1. A tárolt nézettáblák karbantartása

Az alapelve az, hogy az adatbázisrendszer a tárolt nézettáblákat minden újra számítja, ha az alaptáblákban bármilyen változás történt. Egyszerű nézettáblákra vonatkozóan korlátozhatjuk, milyen gyakran vegyük figyelembe a tárolt nézettábla megváltozását, ezzel korlátozhatjuk a nézettábla karbantartására fordítandó munka mennyiségett. Vizsgálunk meg egy példát, amelyben a nézettábla összekapcsolással (join) keletkezik, és láthatjuk, hogy több lehetőségünk is van a karbantartáshoz szükséges munka mennyiségenek a csökkentésére.

8.15. példa. Tegyük fel, hogy gyakran kell megkeresnünk egy adott film producerének a nevét. Előnyösnek találhatjuk a következő tárolt nézettábla alkalmazását:

```
CREATE MATERIALIZED VIEW FilmProducer AS
  SELECT filmcím, év, név
  FROM Filmek, GyártásIrányító
  WHERE producerAzon = azonosító
```

Kezdjük azzal, hogy az adatbázisrendszer nem kell hogy figyelembe vegye a Filmek vagy a GyártásIrányító tábla bármely olyan attribútuma módosításának hatását a FilmProducer nézettáblára, mely attribútum a tárolt nézettáblát létrehozó definícióban nincs megemlítve. Biztos az is, hogy minden olyan módosítás, amely sem a Filmek, sem a GyártásIrányító táblát nem érinti, figyelmen kívül hagyható. Számos további egyszerűsítésre van lehetőségünk, amellyel a Filmek vagy a GyártásIrányító táblák módosításainak hatásait a tárolt nézettáblában is érvényesíthetjük anélkül, hogy a tárolt nézettáblát létrehozó lekérdezést újra végrehajtanánk.

1. Tegyük fel, hogy új filmet veszünk fel a Filmek táblába, legyenek az új film attribútumai: filmcím = 'Kill Bill', év = 2003 és producerAzon = 23456. Ekkor a GyártásIrányító táblában csak a 23456 azonosítójú sort kell megkeresnünk. Mivel az azonosító a GyártásIrányító táblában kulcs, így a következő kérdés legfeljebb egy nevet ad válaszul.

```
SELECT név FROM GyártásIrányító
WHERE azonosító = 23456;
```

Minthogy e kérdés válasza név = 'Quentin Tarantino', így az adatbázisrendszer egy ennek megfelelő sort kell hogy beszűrjon a FilmProducer tárolt nézettáblába. A megfelelő INSERT utasítás:

```
INSERT INTO FilmProducer
VALUES('Kill Bill', 2003, 'Quentin Tarantino');
```

Megjegyezzük, hogy mivel a FilmProducer egy tárolt nézettábla, bármely másik táblához hasonlóan tárolódik, és ez a művelet is ugyanúgy értelmezett, ezért ezt nem kell instead-of triggerrel vagy más módszerrel külön értelmezni.

2. Tegyük fel, hogy a Filmek táblából mondjuk az 1994-ben gyártott, 'Dumb & Dumber' című filmet töröljük. Az adatbázisrendszer ekkor a FilmProducer táblából is csak ezt a filmet kell hogy törölje a következő utasítással:

```
DELETE FROM FilmProducer
WHERE filmcím = 'Dumb & Dumber' AND év = 1994;
```

3. Tegyük fel, hogy a GyártásIrányító táblába új sort szúrunk be. Az új sor tartalma azonosító = 34567 és név = 'Max Bialystock'. Ekkor az adatbázisrendszer a FilmProducer táblába azon filmek sorait kell hogy beszűrja, melyek korábban a producer ismeretlensége miatt nem voltak még ott. A megfelelő utasítás:

```
INSERT INTO FilmProducer
    SELECT filmcím, év, 'Max Bialystock',
    FROM Filmek
    WHERE producerAzon = 34567;
```

4. Tegyük fel, hogy a GyártásIrányító tábla 45678 azonosítójú sorát töröljük. Ekkor az adatbázisrendszernek a FilmProducer táblából is törölnie kell az olyan sorokat, amelyekben szereplő producer azonosítója 45678, hiszen az ilyen filmekhez már nem tartozik sor a GyártásIrányító táblában. Így az adatbázisrendszer végrehajtja a következő utasítást:

```
DELETE FROM FilmProducer
WHERE (filmcím, év) IN
    (SELECT filmcím, év FROM Filmek
    WHERE producerAzon = 45678);
```

Megjegyzés: Nem jó megoldás csak a GyártásIrányító tábla szérint a 45678 azonosítóhoz tartozó nevet felhasználni, és kitörölni a FilmProducer táblából az összes ilyen nevű producert tartalmazó sorokat. Ennek az az oka, hogy mivel a név nem kules a GyártásIrányító táblában, így előfordulhat két azonos nevű producer.

A Filmek tábla olyan módosításait, amelyek a filmcím és év attribútumokat is érintik, és a GyártásIrányító tábla azonosító attribútumát érintő módosítások vizsgálatát gyakorlatként az olvasóra hagyjuk. □

A 8.15. példából levonható nagyon fontos következtetés az, hogy a tárolt nézettábla módosításai *növekményesek* (inkrementálisak). Azaz soha nem előlről kezdve konstruáltuk újra a teljes nézettáblát, hanem az alaptáblákban történt beszűrást, törlést és módosítást a tárolt nézettáblában is követtük. Ezt az alaptáblákon való néhány lekérdezéssel és a tárolt nézettábla módosításával értük el. Ezen felül ezek a módosítások nem érintették a nézettábla összes sorát, csak azokat, amelyeknek legalább egy attribútumában az adott konstans fordul elő.

Nem lehet minden tárolt nézettáblához a 8.15. példában bemutatott szabályokhoz hasonlókat konstruálni, esetleg csak túl bonyolultakat. De a tárolt nézettáblák sok hasonló típusa lehetővé teszi a nézet növekményes karbantartását. A fejezethez tartozó feladatokban a tárolt nézettáblák egy másik típusát – az összesítő nézettáblákat – feladatokon keresztül vizsgáljuk meg.

8.5.2. A tárolt nézettáblák rendszeres karbantartása

Van a tárolt nézettábláknak olyan felhasználása, amely nem igényli azok állandó karbantartását, amikor az azt felépítő alaptáblákban változás történik, és így nem kell aggódunk a naprakészsgég fenntartásának költségén vagy bonyolultságán. A 10.6. alfejezetben fogjuk tanulmányozni az OLAP-ot, de már itt

megjegyezzük, hogy az adatbázisok általában kétféle célt szolgálnak. Például egy áruházi részleg arra használja az adatbázisát, hogy a pillanatnyi raktárkészletet nyilvántartsa, amely minden eladással megváltozik. Ugyanezt az adatbázist az elemzők arra használhatják, hogy a vásárlási szokásokat értékeljék, és megjósolják, hogy egy adott árucikkből mikor kell ismét feltölteni a raktárt.

Előfordulhat, hogy az elemző kérdéseire a tárolt nézettáblából sokkal hatékonyabban lehet válaszolni, különösen, ha a nézettábla összesített adatokat tartalmaz (például a fazón szerinti csoportosítás után a különböző méretű ingek leltári összesenjét). Mivel az adatbázist minden eladás módosítja, így a módosítás sokkal gyakoribb, mint a lekérdezés. Ha a módosítás a jellemző, akkor a tárolt nézettáblák és az indexek fenntartása is költséges.

Hasznos lehet azonban, ha tárolt nézettáblát hozunk létre, de azt nem tartjuk állandóan naprakészen, ha az azt felépítő alaptáblákban változás történik. Az ilyen tárolt nézettáblákat inkább rendszeresen újraépítjük, amikor az adatbázist nem használják intenzíven (tipikusan éjszakánként). Az ilyen tárolt nézettáblákat csak az elemzők használják, az adatai nem naprakészek, hanem 24 óránként frissülnek. Átlagos helyzetben egy árucikk eladási mértéke lassan változik, így az ilyen (naponta frissülő) adat „elég jó” az elemzéshez arra, hogy megjósolhassuk, hogy az adott árucikk jól, vagy gyengén fog fogyni. Természetesen, ha Brad Pitt hawaii ingben látható, akkor az elemző nem tudja megjósolni, hogy másnap kifogy-e a hawaii ing, de ilyen rizikó nem gyakran fordul elő.

8.5.3. Lekérdezések átírása a tárolt nézettáblák használatához

A tárolt nézettáblákra a lekérdezések FROM záradékában ugyanúgy hivatkozhatunk, ahogy a nézettáblákra is (lásd 8.1.2. alfejezetben). Mivel a tárolt nézettábla az adatbázisban ténylegesen tárolódik, így a lekérdezéseket átírhatjuk olyanná, amely a tárolt nézettáblát használja még akkor is, ha az eredeti lekérdezésben a nézettábla nem volt megemlítve. Az ilyen átírás a lekérdezés végrehajtását sokkal gyorsabbá teheti azáltal, hogy a válasz kidolgozási idejének jelentős részét kitevő műveleteket – például a relációk összekapcsolását – már a tárolt nézettábla létrehozásakor elvégeztük.

A lekérdezés tárolt nézettáblák használatára való átírásakor nagyon körültekintően kell eljárnunk. A tárolt nézettáblák használatára vonatkozó összes szabály ismertetése túlmutat e könyv keretein. Mindazonáltal a 8.15. példában bemutatott – és ahoz hasonló – tárolt nézettáblákra vonatkozó viszonylag egyszerű szabályt tudunk javasolni.

Tegyük fel, hogy V tárolt nézettáblát a következő formájú lekérdezéssel definiáltuk:

```
SELECT  $L_V$ 
  FROM  $R_V$ 
 WHERE  $C_V$ 
```

ahol L_V attribútumok listája, R_V relációk listája, C_V pedig egy feltétel. Ha-sonlóan tegyük fel, hogy a Q lekérdezés a következő alakú:

```
SELECT  $L_Q$ 
FROM  $R_Q$ 
WHERE  $C_Q$ 
```

Azok a szabályok, amelyek figyelembevételével a Q kérdés a V tárolt nézettáblát használó lekérdezéssé alakítható, a következők:

1. Az R_V listában szereplő relációk mindegyike előfordul az R_Q listában is.
2. A C_Q feltétel ekvivalens a $C_V \text{ AND } C$ feltétellel, ahol C valamilyen feltétel. Abban a speciális esetben, amikor C_Q feltétel C_V -vel azonos, akkor az „ $\text{AND } C$ ” szükségtelen.
3. Ha C feltétel szükséges, akkor az R_V listában szereplő relációk C feltételben is hivatkozott attribútumainak az L_V listában is elő kell fordulniuk.
4. Az L_Q listában szereplő attribútumoknak az L_V listában (melyben az R_V relációk attribútumai állhatnak) is elő kell fordulniuk.

Ha a fenti feltételek mindegyike teljesül, akkor a Q lekérdezést a V tárolt nézettáblára vonatkozó lekérdezéssé írhatjuk át, mégpedig a következő módon:

- a) Az R_Q listát cseréljük ki a V relációból és az R_Q -ban szereplő, de R_V -ben elő nem forduló relációkból álló listára.
- b) A C_Q feltételt cseréljük ki a C feltételre. Ha C nem szükséges (azaz $C_V = C_Q$), akkor a WHERE záradékot hagyjuk el.

8.16. példa. Tegyük fel, hogy rendelkezésünkre áll a 8.15. példában szereplő `FilmProducer` tárolt nézettábla. Ez a tárolt nézettábla a következő V lekérdezéssel volt definiálva:

```
SELECT filmcím, év, név
FROM Filmek, GyártásIrányító
WHERE producerAzon = azonosító
```

Tegyük fel továbbá, hogy arra a Q kérdésre kell válaszolnunk, hogy milyen nevű színészek szerepelnek a Max Bialystock producer által gyártott filmekben. A kérdés megválaszolásához a következő relációkra van szükségünk:

```
Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)
SzerepelBenne(filmCím, filmÉv, színészNév)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A megfelelő Q lekérdezés:

```
SELECT színészNév
FROM SzerepelBenne, Filmek, GyártásIrányító
WHERE filmCím = filmcím AND Filmek.év = SzerepelBenne.filmÉv
      AND producerAzon = azonosító AND név = 'Max Bialystock';
```

Hasonlítsuk össze a tárolt nézettábla V definícióját és a Q lekérdezést, látjuk, hogy a fentebb megadott feltételeket kielégítik.

1. A V definíció FROM záradékában szereplő relációk minden előfordulnak a Q lekérdezés FROM záradékában is.
2. A Q lekérdezésben szereplő feltétel a (V -ből származó feltétel) AND C alakban is írható, ahol $C =$

```
filmCím = filmcím AND Filmek.év = SzerepelBenne.filmÉv
      AND név = 'Max Bialystock'
```

3. A C feltétel attribútumai (filmcím, év és név) a V -beli relációk (Filmek és GyártásIrányító) attribútumai.
4. A Q kérdés SELECT listája egyetlen attribútuma sem származik a V definíció FROM listájában felsorolt relációkból.

Így a Q lekérdezésben a V tárolt nézettáblát is használhatjuk a lekérdezés következő átírásával:

```
SELECT színészNév
FROM SzerepelBenne, FilmProducer
WHERE filmCím = filmcím
      AND FilmProducer.év = SzerepelBenne.filmÉv
      AND név = 'Max Bialystock';
```

Azaz a FROM záradékban a Filmek és GyártásIrányító relációneveket kicserélük a FilmProducer tárolt nézettábla nevére. A tárolt nézettábla definiálásakor szerepelt feltételt is eltávolítottuk a lekérdezésből, és így csak a C feltételt hagyjuk meg. Az átalakított lekérdezés három helyett már csak két reláció összekapcsolásával dolgozik, így számíthatunk arra, hogy az átalakított lekérdezés végrehajtása gyorsabb az eredetinél. \square

8.5.4. Tárolt nézettáblák automatikus előállítása

A 8.4.4. alfejezetben az indexek automatikus előállítására vonatkozó meggondolások a tárolt nézettáblákra is alkalmazhatók. Először is fel kell mérnünk, vagy meg kell becsülnünk, hogy az adott alkalmazásban milyen lekérdezések fordulnak elő. Az automatikus „tárolt-nézettábla-előállító tanácsadó” nézettáblajelölteket állít elő. Ez a feladat jóval bonyolultabb, mint az indexjelöltek előállítása.

Az indexek esetében minden reláció minden attribútumára egy index építése lehetséges. Indexeket építhetünk a reláció attribútumainak (kisebb) halmazaira is, de ha így járunk el, az indexjelöltek előállítása ekkor is egyszerű. Ugyanakkor tárolt nézettáblákat elméletileg bármilyen lekérdezéssel definiálhatunk, így a figyelembe veendő nézettáblák mennyisége határtalan.

A feladatot korlátozhatjuk azzal, ha arra gondolunk, hogy nincs értelme olyan tárolt nézettáblát létrehozni, amely egyetlen – a felmérésünkben előforduló – szóba jöhető lekérdezés végrehajtását sem segíti. Tegyük fel például, hogy a felmérésünkben szereplő összes lekérdezés a 8.5.3. alfejezetben bemutatott formájú. Ekkor a 8.5.3. alfejezetben adott elemzést felhasználhatjuk azon nézettáblák megkereséséhez, amelyek az adott lekérdezésekben hasznosak lehetnek. A tárolt nézettáblajelöltek keresését a következőkre korlátozhatjuk:

1. A **FROM** záradék relációlistájában csak olyan relációk forduljanak elő, amelyek a felmérésben szereplő lekérdezések valamelyikének **FROM** záradékában is szerepeltek.
2. A **WHERE** záradékban **AND** művelettel összekapcsolva csak olyan feltételek szerepeljenek, amelyek mindegyike a felmérés legalább egy lekérdezésében is előfordult.
3. A **SELECT** záradék attribútumlistája olyan legyen, amely elegendő a felmérésben szereplő lekérdezések legalább egyikéhez.

A tárolt nézettáblák hasznosságának megítéléséhez a lekérdezés optimalizáló megbecsüli a lekérdezések végrehajtási idejét tárolt nézettábla használatával és nélküle. Ehhez természetesen az optimalizálónak képesnek kell lennie a tárolt nézettáblák vizsgálatára; az indexek vizsgálatára a modern optimalizálók minden egyike képes, de a tárolt nézettáblák vizsgálatára nincs mindegyik felkészítve. A 8.5.3. alfejezet olyan okfejtésre volt példa, amilyent az optimalizálónak is végre kell hajtani, hogy megállapíthassa, hogy az adott nézettábla hoz-e valami előnyt.

Van másik szempont is, amely felvetődik akkor, amikor a tárolt nézettáblák automatikus kiválasztását vizsgáljuk, és amely az indexekkel kapcsolatban nem jelent meg. A relációhoz épített index általában kisebb, mint maga a reláció; egy relációhoz építhető indexek helyigénye nagyjából ugyanakkora. A tárolt nézettáblák nagyon változóak a helyigény tekintetében: azok, amelyek összekapcsolást (join) igényelnek, sokkal nagyobbak lehetnek, mint az vagy azok a relációk, amelyekből felépülnek. Így át kell gondolnunk a tárolt nézettáblák „hasznosságá” definícióját. Definiálhatjuk például a hasznosságot úgy is, hogy a felmérésben szereplő lekérdezések átlagos végrehajtási idejének csökkenési mértékét elosztjuk a tárolt nézettábla által okozott tárolási igény növekedésének mértékével.

8.5.5. Feladatok

8.5.1. feladat. Egészítsük ki a 8.15. példát az alaptáblák tartalma módosításának figyelembenbevételelvel.

! 8.5.2. feladat. Tegyük fel, hogy a 8.2.3. feladat ÚjPC nézettáblája tárolt nézettábla. A Termék és a PC alaptáblák mely módosításai igénylik a tárolt nézettábla módosítását is? Hogyan tudjuk ezeket a módosításokat növekményesen megvalósítani?

! 8.5.3. feladat. Ez a feladat az összesítésekre alapozott tárolt nézettáblákat vizsgálja. Tegyük fel, hogy a hajókkal kapcsolatos példaadatbázisunk a következő alaptábláit használva:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
    kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
```

tárolt nézettáblát építünk:

```
CREATE MATERIALIZED VIEW HajóStatisztika AS
    SELECT ország, AVG(vízkiszorítás), COUNT(*)
    FROM Hajóosztályok, Hajók
    WHERE Hajóosztályok.osztály = Hajók.osztály
    GROUP BY ország;
```

A Hajóosztályok és a Hajók alaptáblák mely módosításai igénylik a tárolt nézettábla módosítását is? Hogyan tudja ezeket a módosításokat növekményesen megvalósítani?

! 8.5.4. feladat. A 8.5.3. alfejezetben megadtunk feltételeket, amelyek teljesülése esetén egyszerű tárolt nézettáblák hasonló lekérdezések végrehajtásában felhasználhatók. A 8.15. példa tárolt nézettáblájára nézve határozzuk meg az összes lekérdezéstípusokat, melyekhez ez a nézettábla használható.

8.6. Összefoglalás

- ◆ *Virtuális nézettáblák:* A virtuális nézettábla egy definíció; annak definíciója, hogy a nézettáblát logikailag hogyan lehet megkonstruálni az adatbázis tárolt tábláiból vagy más nézettáblákból. A nézettáblák ugyanúgy lekérdezhetők, mint a tárolt táblák, de az SQL átalakítja a lekérdezést úgy, hogy az a nézettábla definíálásánál használt tárolt táblákon fog lefutni.
- ◆ *Módosítható nézettáblák:* Az egy relációból felépített virtuális nézettáblák módosíthatók abban az értelemben, hogy a nézettáblába sorokat szúrhatunk be, törölhetünk és módosíthatunk ugyanúgy, mintha tárolt tábla lenne. Az ilyen műveleteket az adatbázisrendszer a nézettábla definíciójában szereplő alaptáblára vonatkozó ekvivalens műveletekké alakítja át.

Az SQL-szabvány nyelvei

Egy – az SQL-szabványnak megfelelő – megvalósításnak támogatnia kell a következő hét befogadó nyelv legalább egyikét: ADA, C, Cobol, Fortran, M (hivatalosan Mumpsnak nevezik, és általában orvosi körökben használják), Pascal, illetve PL/I. Mi a példáinkban C-t használunk.

annyit követel meg, hogy az SQL-implementációknak ezek közül legalább egyet biztosítaniuk kell a felhasználók számára.

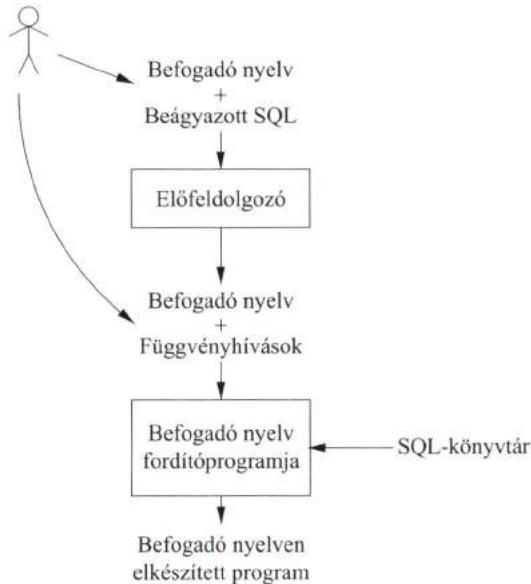
1. *Általános SQL-felület.* A felhasználók SQL-utasításokat írhatnak be, és az SQL-szerverrel végre is hajthatják azokat. Ezen megoldásnál minden lekérdezés, illetve minden más utasítás is modulnak minősül. Bár ebben a könyvben példáinkban leggyakrabban ezt a módszert használjuk, a gyakorlatban történő alkalmazása viszont ritka.
2. *Beágyazott SQL.* Ezzel foglalkozunk a 9.3. alfejezetben. A beágyazott SQL-utasításokat általában egy előfeldolgozó program alakítja befogadó nyelvi utasításokhoz illeszkedő SQL-rendszerbeli függvény- vagy eljáráshívásokká. A lefordított befogadó nyelvi program – az előzőleg említett függvényhívásokat is beleértve – alkot egy modult.
3. *Valódi modulok.* Az SQL által támogatott legáltalánosabb modulok, amelyek különféle tárolt eljárásokra és függvényekre épülnek (ezek akár valamelyen befogadó nyelven, akár SQL nyelven készülhetnek). Ezek az eljárások és függvények paraméterek átadásával és osztott változókkal kommunikálnak egymással. A PSM-modulok (lásd 9.4. alfejezet) alkalmas példák ezekre a modulokra.

Egy modul egy végrehajtását nevezük *SQL-ágensnek*. A 9.4. ábrán egy SQL-ágens és egy modult egyetlen alkotóelemként tüntettük fel (az ábrán ez az alkotóelem egy SQL-kliens segítségével kommunikál egy SQL-szerverrel). Ennek ellenére ügyeljünk az ágens és a modul között meglevő különbségekre, amelyek leginkább az operációs rendszerek program- és folyamatabsztrakciója közti analógiára hasonlítanak: a modul egy végrehajtható kód, az ágens pedig ezen kód végrehajtása.

9.3. Az SQL és a befogadó nyelv közötti felület

Az eddigiekben csak az ún. *általános SQL-felüleettel* foglalkoztunk, amikor azt feltételeztük, hogy az SQL-parancsainkat egy azok bekérésére és végrehajtására felkészített SQL-interpreterrel hajtjuk végre. Bár majdnem minden ABKR

nyújt erre lehetőséget, mégis ez a használati mód valójában igen ritka. A 9.1. ábrán bemutatottakhoz hasonló valós rendszerekben ezeket az alkalmazásokat általában valamilyen ún. befogadó nyelven (például C nyelven) készítik, de az alkalmazás egyes részei valójában SQL-utasítások.



9.5. ábra. Beágyazott SQL-utasításokat is tartalmazó program feldolgozása

A 9.5. ábra szemlélteti egy SQL-utasításokat tartalmazó tipikus programozási rendszer vázlatát. Láthatjuk, hogy a programozó feladata a befogadó nyelven megfogalmazott program megírása, amely „beágyazott” SQL-utasításokat is tartalmaz. A beágyazás az elhelyezkedésétől függően kétféleképpen történhet.

1. *Hívásszintű felület.* Tartalmaz egy könyvtárat és ezen könyvtárcsomag függvényeinek és eljárásainak hívásait végző befogadó nyelvben írt beágyazott SQL-t is. Az SQL-utasítások általában a könyvtárbeli eljárásoknak a karaktersor argumentumai. Ezt a megközelítést, amelyet gyakran hívásszintű felületnek vagy CLI-nek neveznek, a 9.5. alfejezetben tárgyaljuk. Ezt az elképzélést szemlélteti a 9.5. ábrán szereplő, a felhasználó és a befogadó nyelv között lévő görbített nyíl.
2. *Közvetlenül beágyazott SQL.* Itt a befogadó nyelven megfogalmazott teljes program (a beágyazott SQL-utasításokkal együtt) átkerül egy előfeldolgozóhoz, amely átalakítja a beágyazott SQL-utasításokat a befogadó nyelven is értelmezhető alakra. Az SQL-utasításokat általában könyvtári függvény-, illetve eljáráshívásokkal helyettesíti. Azaz a CLI és a közvetlenül beágyazott SQL közötti különbség inkább csak „kinézet és felfogás”

kérdése, és nem lényegi eltérés. A változtatások után az előfeldolgozott befogadónyelvi program a megszokott módon kerül lefordításra, és az adatbázison végzendő műveletek végrehajtását könyvtári hívásokon keresztül végzi el.

Ebben az alfejezetben az SQL-szabvány a befogadó nyelv (főként a C) közvetlen beágyazásával foglalkozó részeit tekintjük át. Sőt, néhány olyan fogalmat is bevezetünk, mint a kurzor, amely mindegyik vagy majdnem mindegyik beágyazott SQL-t támogató rendszerben előfordul.

9.3.1. A típuseltérés problémája

Az SQL-utasítások és a hagyományos programozási nyelvek összekapcsolásának alapvető problémája az úgynevezett típuseltérés, azaz az a tény, hogy az SQL adatmodellje lényegesen eltér a többi programozási nyelv modelljétől. Mint tudjuk, az SQL magját a relációs adatmodell képezi. Azonban a C és a hozzá hasonló programozási nyelvek egy olyan modellt használnak, amely tartalmazhat egész számokat, valós számokat, aritmetikai értékeket, karaktereket, mutatókat, rekordstruktúrákat, tömböket és egyéb hasonló struktúrákat. Például a C programozási nyelv sem támogatja közvetlenül nyelvi eszközökkel a halmazfogalom alkalmazását, míg az SQL nem támogatja a tömb, mutató és több más programnyelvi elem alkalmazását. Ezért az SQL és más nyelvek közötti adatátvitel nem közvetlen, így szükségesek az elkövetkezőkben ismertetett mechanizmusok, amelyek támogatják az SQL-t és a befogadó nyelven írt kód részeket egyaránt tartalmazó alkalmazások fejlesztését.

Az olvasó első ránézésre azt is gondolhatná, hogy a legjobb megoldás az, hogy egyáltalán ne keverjük a programozási nyelveket: vagy végezzünk minden számítást SQL-ben, vagy egyáltalán ne is használjuk az SQL-t. Könnyen belátható, hogy ez az út nem járható, mivel ha szükségünk van adatbázisok elérésére, akkor erre az SQL egy jól használható, hatékony és magas szintű eszköz. Az SQL-t használva a programozónak nem kell törödni azzal, hogyan szervezi meg az adatok hatékony tárolását a rendelkezésre álló háttéráron, illetve azzal sem kell törödni, hogy az adatbázison végzett műveleteknél hogyan használják ki hatékonyan a tárolt struktúrákat.

Másrészről vannak dolgok, amelyeket egyáltalán nem lehet SQL nyelven ki fejezni. Például nem írhatunk egy olyan SQL-lekérdezést, amely kiszámítja egy tetszőleges szám faktoriálisát. Ilyen számítások például C vagy hozzá hasonló nyelvek¹ bármelyikén könnyen elvégezhetők. Hasonlóan nem lehetséges SQL nyelven például a lekérdezések eredményének grafikus megjelenítését specifikálni. Látható, hogy a gyakorlati életben használt adatfeldolgozó programok elkö-

¹ Bánjunk óvatosan a választással, ugyanis vannak olyan kiterjesztései az alap SQL nyelvnek (mint például a 10.2. alfejezetben tárgyalt rekurzív SQL vagy a 9.4. alfejezetben tárgyalt SQL/PSM), amelyek „Turing-teljességet” garantálnak, azaz a képességet ahhoz, hogy minden kiszámíthatunk, ami bármely más nyelven is kiszámítható lenne. Ezek a kiterjesztések viszont sohasem általános célú számításokra lettek bevezetve, és ezeket nem tekinthetjük általános célú nyelveknek sem.

szítéséhez szükség van mind az SQL, mind pedig a hagyományos, ún. befogadó programozási nyelvre.

9.3.2. Az SQL és a befogadó nyelv közötti interfész

Ha egy befogadó nyelven írt programban SQL-utasítást kívánunk használni, akkor ezt jelezünk kell az előfeldolgozónak az utasítás elé írt EXEC SQL kulcsszó segítségével. Az osztott változók használatával információt cserélhetünk a befogadó nyelvi program és a csak SQL-utasításokkal elérhető adatbázis között. Ezen változók már mind a befogadó nyelvi, mind az SQL-utasításokban szerepelhetnek. Az SQL-utasításokon belül az osztott változókat egy kettőspont előzi meg, a befogadó nyelvi utasításokban viszont kettőspont nélkül szerepelnek.

Az SQL-szabvány definiál egy SQLSTATE nevű változót az SQL és a befogadó nyelvi környezetek összekapcsolására. Ez a változó öt karaktert tartalmaz (általában egy ötelemű karakteres tömb). Az adatbázis elérését támogató könyvtárak úgy vannak elkészítve, hogy visszatéréskor ebben a változóban helyezik el a végrehajtott SQL-művelet során fellépett esetleges problémákat leíró kódokat. Az SQL-szabvány jó néhány 5 karakter hosszú kódot és a hozzá tartozó jelentést is meghatározza.

Például a '00000' (öt darab nulla számjegy) azt jelzi, hogy a függvény végrehajtása során nem léptek fel problémák, míg a '02000' kód egy lekérdezési művelet végrehajtása után azt jelezheti, hogy nincs a lekérdezésben megadott kritériumoknak eleget tevő sor az adatbázisban. Az utóbbi kód nagyon fontos, hiszen lehetővé teszi számunkra, hogy egy olyan ciklust fogalmazhassunk meg a befogadó nyelvben, amely egyenként megvizsgálja az adott reláció sorait, illetve amely az utolsó sor megvizsgálása után be is fejeződik.

9.3.3. A deklarációs rész

Az osztott változók deklarációját két beágyazott SQL-utasítás közé kell tenni az alábbi minta szerint:

```
EXEC SQL BEGIN DECLARE SECTION;  
...  
EXEC SQL END DECLARE SECTION;
```

A fenti két SQL-utasítás közti részt (amelyet a példában kipontoltunk) *deklarációs résznek* nevezzük. A deklarációs részben a változó deklaráció szintaxisa megegyezik a befogadó nyelven megszokott deklarációs szintaxissal, és mivel a változókat a befogadó nyelven is és az SQL-ből is el akarjuk érni, ezért csak olyan adattípussal dekláralhatjuk őket, amelyek mindenben egyaránt elérhetők (például egészek, valósak, karakterláncok, tömbök).

9.3. példa. Az alábbi utasítások előfordulhatnak például a Stúdió relációt módosító C függvényekben.

```
EXEC SQL BEGIN DECLARE SECTION;
    char stúdióNév[50], stúdióCím[256];
    char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

Az első és az utolsó (negyedik) utasítás a deklarációs rész kezdetét és végét jelzik. A középső két sor két változót definiál: a `stúdióNév` és a `stúdióCím` változókat. Ezek mindenketen karakterekből álló tömb típusú változók: a későbbiekben a Stúdió relációba beszűrni kívánt sorok stúdiónév- és stúdiócímértékeit fogjuk bennük tárolni. A harmadik sorban pedig az `SQLSTATE` változót egy hat elem hosszú karaktertömbnek² definiáljuk. □

9.3.4. Osztott változók használata

Az osztott változókat az SQL-utasítások azon részein használhatjuk, ahol vagy konstans értéket várunk, vagy a konstans érték megengedett. Mint már azt korábban említettük, a nevét kettősponttal megelőzve használjuk. A következő példában a 9.3. példában definiált változók értékét fogjuk beszűrni a Stúdió relációba.

9.4. példa. A 9.6. ábrán egy olyan C nyelvű – `beolvassStúdió` nevű – függvény vázlatát láthatjuk, amely a felhasználótól egy stúdiónevét és egy stúdiótímet vár bemenetként, és beszűrja a beadott adatokat a Stúdió táblába. Az 1–4. sor megegyezik a 9.3. példában bemutatott változódeklarációval. A példából kihagytuk a képernyőre író és a billentyűzetről olvasó sorokat (azok helyét egy megjegyzésben megadtuk).

Az 5. és a 6. sorban egy `INSERT` utasítást láthatunk, amit az említett `EXEC SQL` kulcsszavak vezetnek be azt jelölve, hogy ez nem egy szokásos C nyelvű utasítás, hanem egy beágyazott SQL-utasítás. Az említett előfordító program majd elvégzi a szükséges átalakításokat az `EXEC SQL` kulcsszavakkal bevezetett utasításokon. Az 5. és 6. sorokban megadott beszúrandó értékek nem szövegkonstansok, mint azt az SQL-ben azon a helyen írhatnánk (lásd például a 6.34. példát), hanem (megosztott elérésű) változók, amelyeknek az aktuális értéke lesz behelyettesítve az SQL-utasítás kiértékelésekor. A program a 6. sorban megadott változók aktuális értékéből állít össze egy sort, amelyet beszűr a táblába. □

² Az 5 karakteres `SQLSTATE` változó számára azért foglaltunk le 6 karaktert, mert a továbbiakban olyan C-függvényekkel fogjuk azt kezelni, mint amilyen pl. az `strcmp`, amelyik elvárja, hogy a karakterláncokat a '`\0`' karakterrel zárnak le. Ezt a lezáró karaktert kell a 6. pozícióban tárolnunk. Ennek a 6. karakternek a kezdeti '`\0`'-ra való beállítását a programjainkban nem fogjuk feltüntetni.

```

void beolvasStúdió() {

1)      EXEC SQL BEGIN DECLARE SECTION;
2)          char stúdióNév[50], stúdióCím[256];
3)          char SQLSTATE[6];
4)      EXEC SQL END DECLARE SECTION;

    /* Kiírja a képernyőre azt a felhasználónak
szóló utasítást, hogy adjon be egy nevet és
egy címet, majd a beadott válaszokat eltárolja
a stúdióNév és stúdióCím változókba */

5)      EXEC SQL INSERT INTO Stúdió(név, lakcím)
6)              VALUES (:stúdióNév, :stúdióCím);
}

```

9.6. ábra. Osztott változók alkalmazása egy új stúdió beszúrására

Az EXEC SQL kulcsszó segítségével az összes olyan SQL-utasítás beágyazható, amelynek nincs visszatérési értéke, azaz amely nem lekérdezés: beágyazhatók például a beszúrást, törlést, módosítást leíró utasítások, valamint az adatbázissémát manipuláló utasítások, amelyek például táblákat vagy nézettáblákat törölnek vagy hoznak létre.

A SELECT SQL-lekérdezések általában nem ágyazhatók be közvetlenül a befogadó nyelvbe az ún. „típuseltérés” miatt. A lekérdezések egy multihalmazt adnak vissza eredményül, és a legtöbb programozási nyelv viszont nem támogatja közvetlenül a halma vagy a multihalmaz adatszerkezetet. Azaz a beágyazott SQL-nek az alábbi két módszer valamelyikét kell használni ahhoz, hogy össze tudja kapcsolni a lekérdezések eredményeit és a befogadó nyelvi programjait:

1. Az *egyetlen sort eredményező lekérdezések* a lekérdezés eredményeként létrejött eredménysort eltárolhatják akár osztott változókba oly módon, hogy az eredménysor egyes komponensei külön-külön osztott változókba lesznek elhelyezve.
2. *Kurzorok*. Egy olyan lekérdezés, amely egynél több sort (azaz egy sorhalmazt) is visszaadhat eredményül, csak úgy hajtható végre, ha egy *sormutatót* definiálunk hozzá. A sormutató majd befutja az eredményreláció összes sorát, és egy-egy eredménysor egyes komponenseit külön-külön osztott változókba elhelyezve juttathatjuk el az adatokat a befogadó programnak.

A következő alfejezetekben minden fenti lehetőséget megvizsgáljuk.

9.3.5. Egyetlen sort eredményező lekérdezések

Egy alkalmazásokba beágyazható egyetlen sort eredményező lekérdezés formájaannyiban különbözik az SQL nyelv elemeinek bemutatásakor ismertetett közönséges SELECT utasítástól, hogy a SELECT záradék után egy INTO kulcsszót kell írni, e kulcsszó mögé pedig fel kell sorolni azokat a változókat, amelyekbe a lekérdezés eredményeként visszakapott adatokat el akarjuk tárolni. A már megismert szabályok alapján az itt felsorolt változók neve elé egy-egy kettőspont karaktert kell írni. Amennyiben a lekérdezés egyetlen sort eredményez, úgy a megadott változók rendre felveszik az eredménysor komponenseiben képződött értékeket. Amennyiben a lekérdezés egyetlen sort sem eredményez, vagy éppen egnél több sort eredményez, akkor a felsorolt változók semmilyen értéket sem kapnak, és az adatbázisrendszer az SQLSTATE változóba beírja a megfelelő hibakódot.

9.5. példa. Most elkészítünk egy C nyelvű függvényt, amely bekéri egy stúdió nevét a felhasználótól, és kiírja a stúdió igazgatójának nettó jövedelmét. A program vázlatos forráskódját a 9.7. ábra tartalmazza. Az 1–5. sorok tartalmazzák a szükséges változók deklarációját. Ezután – a megjegyzéssel jelölt helyen – következne a stúdió nevét bekérő programrész (az ezeket megvalósító C nyelvű sorokat a példából elhagytuk).

```

void nettóFizetésKiírása() {
    1)      EXEC SQL BEGIN DECLARE SECTION;
    2)          char stúdióNév[50];
    3)          int igazgatóNettóFizetése;
    4)          char SQLSTATE[6];
    5)      EXEC SQL END DECLARE SECTION;

    /* Kérjünk be a felhasználótól egy stúdiónevet,
    és a választ tegyük a stúdióNév változóba */

    6)      EXEC SQL SELECT nettóBevétel
    7)              INTO :igazgatóNettóFizetése
    8)              FROM Stúdió, GyártásIrányító
    9)              WHERE elnökAzon = azonosító AND
                  Stúdió.név = :stúdióNév;

    /* Itt kiírhatjuk az eredményt,
    miután megbizonyosodtunk arról, hogy az SQLSTATE
    változó öt darab nulla karaktert tartalmaz */
}

```

9.7. ábra. Egy egyetlen sort eredményező lekérdezés beágyazása

Majd a 6–9. sorokban egy egyetlen sort eredményező lekérdezést láthatunk, amelynek felépítése hasonlít a korábbi fejezetekben már megismert lekérdezések szerkezetéhez. A korábban bemutatott lekérdezésekhez képest itt két különbözőt is megfigyelhetünk: egyszerűt a 7. sorban használjuk azt az INTO kulcsszóval bevezetett záradékot, amelyben megadjuk, hogy melyik változókban akarjuk visszakapni a lekérdezés eredményét; másrészt pedig a stúdióNév változó értékét helyezzük az SQL-utasítás feltételei közé a 9. sorba (ahol régebben ehetett például szöveges konstansértéket írtunk). E lekérdezés végrehajtása után egyetlen soros válaszra számítunk, és a válaszsor láthatóan egyetlen oszlopot tartalmaz, amely a visszakapott nettóBevétel attribútum értékét tárolja. Ezen egyetlen sor egyetlen oszlopának tartalma az igazgatóNettóFizetése nevű közös változóban lesz eltárolva. □

9.3.6. Sormutatók

Az SQL-lekérdezések és a befogadó nyelv összekapsolásának legsokoldalúbb módszere egy ún. sormutató használata, amely végigmegy egy reláció sorain. A reláció lehet egy tárolt tábla vagy egy lekérdezés által előállított reláció is. Egy sormutató létrehozása és használata a következőképpen történhet:

1. Deklarálni kell a sormutatót. A sormutató deklarációjának legegyszerűbb módja a következő:

```
EXEC SQL DECLARE <sormutatónév> CURSOR FOR <lekérdezés>
```

A lekérdezés lehet akár egy egyszerű select-from-where lekérdezés vagy egy reláció neve. A sormutató a lekérdezés eredményeként meghatározott reláció sorain fut véig.

2. Ezután egy EXEC SQL OPEN utasítás következik, amelyet a sormutató neve követ. Ez az utasítás inicializálja a sormutatót a hozzá tartozó reláció első sorának pozíójával.
3. Ezt követheti egy vagy több ún. *fetch utasítás*. Ennek az utasításnak a feladata a kurzorhoz tartozó reláció következő sorának meghatározása. Egy fetch utasítás általános alakja a következő:

```
EXEC SQL FETCH FROM <sormutatónév> INTO <változók listája>
```

A változók listáján a reláció sorának minden egyes attribútumához tartozik egy változó. Ha van elérhető sor, akkor ezeknek a változóknak az értéke a sor megfelelő komponensének az értéke lesz. Ha egy sormutatóval már nincs több elérhető sor, akkor a fetch utasítás nem tesz semmit a benne megadott változókba, a SQLSTATE változóba pedig a '02000' konstans értéket teszi, ami azt jelenti, hogy nincs több beolvasható sor.

4. Végül az **EXEC SQL CLOSE** utasítás következik, amelyet a sormutató neve követ. Ez az utasítás lezárja a sormutatót, amely így már nem fut tovább a reláció sorain. Egy újabb **OPEN** utasítással azonban újrainicializálható a sormutató, és ekkor a szóban forgó reláció sorainak új értékein fut majd végig.

9.6. példa. Meg szeretnénk tudni azoknak a filmgyártásvezetőknek a nevét, akiknek a nettó jövedelmük egy előre megadott, exponenciálisan növekedően meghatározott sávok valamelyikébe esik; az egyes sávokat a jövedelem számjegyeinek a száma szerint definiáltuk. Egy olyan lekrédítést készítünk, amely beolvassa a **GyártásIrányító** tábla összes sorának a **nettóBevétel** attribútumát egy **jövedelem** nevű osztott változóba. A tábla sorain egy **irányítókSormutató** nevű sormutatóval haladunk végig, kiszámoljuk az aktuálisan beolvastott jövedelemérték számjegyeinek a számát, ez alapján határozzuk meg, hogy a beolvastott jövedelem mely sávba tartozik, majd az erre a célra létrehozott **számláló** tömb megfelelő elemét eggyel megnöveljük.

```

1) void jövedelemSávok() {
2)     int i, számjegyek, számláló[15];
3)     EXEC SQL BEGIN DECLARE SECTION;
4)         int jövedelem;
5)         char SQLSTATE[6];
6)     EXEC SQL END DECLARE SECTION;
7)     EXEC SQL DECLARE irányítókSormutató CURSOR FOR
8)             SELECT nettóBevétel FROM GyártásIrányító;

9)     EXEC SQL OPEN irányítókSormutató;
10)    for(i=1; i<15; i++) számláló[i] = 0;
11)    while(1) {
12)        EXEC SQL FETCH FROM irányítókSormutató
13)            INTO :jövedelem;
14)        if(NINCS_TÖBB_SOR) break;
15)        számjegyek = 1;
16)        while((jövedelem /= 10) > 0) számjegyek++;
17)        if(számjegyek <= 14) számláló[számjegyek]++;
18)    }
19)    EXEC SQL CLOSE irányítókSormutató;
20)    for(i=0; i<15; i++)
21)        printf("számjegyek = %d: előfordulások
22)                           száma = %d\n", i, számláló[i]);
}

```

9.8. ábra. A jövedelmek exponenciálisan növekvő sávok szerinti osztályozása

A **jövedelemSávok** nevű C-függvény a 9.8. ábra 1. sorában kezdődik. A 2. sorban deklaráljuk azokat a változókat, amelyekre csak C-utasításokban kívánunk hivatkozni (beágyazott SQL-ben nem). Az itt deklarált számláló tömb minden egyes jövedelemsávhoz tartalmaz egy-egy számlálót. A **számjegyek** változóban számoljuk meg a **jövedelem** változóban levő érték leírásához szükséges számjegyek számát. Az i változót pedig tömbindexként használjuk a számláló tömb elemein történő végighaladáshoz.

A 3–6. sorokban deklaráljuk a **jövedelem** és a szokásos SQLSTATE változókat, amelyeket az SQL-utasításokból is elérhetünk. Az **irányítókSorMutató** nevű sormutatót a 7. és 8. sorokban deklaráljuk, amely a 8. sorban megadott lekérdezés eredményeként létrejött tábla sorain megy végig. Ez a lekérdezés a **GyártásIrányító** tábla **nettóBevétel** oszlopának tartalmát adja vissza eredményül. A sormutatót a 9. sorban megnyitjuk, majd a 10. sorban inicializáljuk nulla értékkel az előbb említett számláló nevű tömb elemeit.

A függvény lényegi része a 11–16. sorokban látható. A 12. sorban olvassuk be a következő rekordot, a beolvasott érték a **jövedelem** változóba kerül. Mivel a 8. sorban deklarált lekérdezés egyoszlopos, ezért itt elég csak egy változót megadni (általában az oszlopok számával megegyező számú változót kell megadni). A 13. sorban ellenőrzük azt, hogy volt-e még beolvasható adat. Itt egy **NINCS_TÖBB_SOR** nevű makrót adtunk meg, amelyet az alábbi módon definiálhatunk:

```
#define NINCS_TÖBB_SOR !(strcmp(SQLSTATE, "02000"))
```

Emlékezzünk rá, hogy a "02000" érték jelöli azt, ha nincs több adat a sormutató olvasása során. Ha nem sikerült adatot olvasni, akkor kiugrunk a ciklusból a 17. sorra.

Ha sikerült adatot olvasni, akkor a 14. sorban inicializáljuk a számjegyeket számláló változót 1-re. A 15. sorban megszámoljuk a beolvasott érték számjegyeinek a számát (minden egyes 10-zel való osztás után növeljük eggyel a számjegyeket számláló változót, amíg a 10-zel való osztások során nullát nem kapunk). Végül a 16. sorban megnöveljük eggyel a számláló tömb megfelelő elemét. Láthatjuk, hogy a programban a 15 vagy annál több jegyű számokkal nem foglalkozunk – a statisztikát nem befolyásoljuk az ilyen óriási számokkal (ilyenkor nem növelünk egyetlen tömbelemet sem).

A 17. sorban kezdődik a függvény kiíratási része. Lezártuk a sormutatót, majd a 18–19. sorban kiírjuk a statisztikát tároló számláló tömb tartalmát. □

9.3.7. Sormutatóval történő módosítások

Amikor egy sormutatóval végigmegyünk egy adatbázistábla (vagyis egy adatbázisban tárolt reláció) sorain, az egyes sorokat nemcsak olvashatjuk, hanem módosíthatjuk is azok tartalmát, vagy akár törölhetjük is az illető sort. Az ilyen módosító és törlő UPDATE és DELETE utasítások szintaxisa megegyezik a 6.5. alfejezetben megismertekkel attól eltekintve, hogy itt a WHERE záradékban

csak a CURRENT OF kulcsszavakat adhatjuk meg, azután pedig annak a sormutatónak a nevét kell megadni, amelyikből a módosítandó vagy törlendő sort olvastuk. Természetesen egy ilyen módosító vagy törlő beágyazott SQL-utasítás végrehajtása bekerülhet a befogadó nyelven írt elágazások egy-egy ágába attól függően, hogy az alkalmazásnak milyen feladatot kell megoldania.

```

1) void bevételVáltozás() {
2)     EXEC SQL BEGIN DECLARE SECTION;
3)         int azonosító, jövedelem;
4)         char gyártóNév[31], gyártóCím[256], SQLSTATE[6];
5)     EXEC SQL END DECLARE SECTION;
6)     EXEC SQL DECLARE irányítókSormutató CURSOR FOR
                    GyártásIrányító;
7)     EXEC SQL OPEN irányítókSormutató;
8)     while(1) {
9)         EXEC SQL FETCH FROM irányítókSormutató INTO
                    :gyártóNév, :gyártóCím,
                    :azonosító, :jövedelem;
10)        if(NINCS_TÖBB_SOR) break;
11)        if (jövedelem < 1000)
12)            EXEC SQL DELETE FROM GyártásIrányító
                    WHERE CURRENT OF irányítókSormutató;
13)        else
14)            EXEC SQL UPDATE GyártásIrányító
                    SET nettóBevétel = 2 * nettóBevétel
                    WHERE CURRENT OF irányítókSormutató;
15)    }
        EXEC SQL CLOSE irányítókSormutató;
    }
}

```

9.9. ábra. A gyártó nettó bevételének módosítása

9.7. példa. A 9.9. ábrán egy olyan C-függvényt láthatunk, amely végigmegy a GyártásIrányító tábla sorain, és az illető sorra vonatkozóan eldönti, hogy törölje-e azt vagy kétszeresse meg a nettó bevételének értékét. A 3. és 4. sorokban a GyártásIrányító négy attribútumának, illetve a szükséges SQLSTATE értékének a tárolására alkalmas változókat deklarálunk. Majd a 6. sorban az irányítókSormutató-t is deklaráljuk, hogy a GyártásIrányító reláció sorait végigolvashassuk a segítségével.

A 8–14. sorban található a ciklus, amelyben az irányítókSormutató nevű sormutató minden lépésben a GyártásIrányító reláció egyes soraira mutat. A 9. sorban kérjük be az aktuális sor értékeit – az erre a célról fenntartott –

négy változóba. Vegyük észre, hogy most csak a *jövedelem* változót használjuk. A 10. sor ellenőrzi, hogy van-e egyáltalán elérhető sor a *GyártásIrányító* relációban. Itt is a korábban már használt *NINCS_TÖBB_SOR* makrót használjuk feltételként (az *SQLSTATE* jelentése „nincs több sor”, kódja "02000").

A 11. sorban vizsgáljuk, hogy az aktuális nettó bevétel kevesebb-e 1000 \$-nál. Amennyiben a válasz igen, akkor a 12. sorban lévő *DELETE* utasítás segítségével töröljük ezt a sort. Vegyük észre, hogy a *WHERE* záradékban a sormutatóra hivatkozunk, és így a *GyártásIrányító* aktuális sorára. Éppen arra a sorra, amelyet az előbb kértünk be. Ezt a sort töröljük a *GyártásIrányító* relációból. Ha a nettó bevétel legalább 1000 \$, akkor a 14. sor pedig – az előzőekkel szemben – megduplázza az aktuális sor nettó bevételének értékét. □

9.3.8. Egyidejű módosítások elleni védelem

Tegyük fel, hogy a gyártásirányítóknak a nettó bevételeit vizsgáljuk a 9.8. ábra *jövedelemSávok* függvényét felhasználva, és hogy eközben valamilyen más alkalmazás módosítja a vizsgálatunk alapjául szolgáló *GyártásIrányító* relációt. Mit tehetnék ekkor? A válasz: „Talán semmit.” Megelégedhetünk a hozzávetőleges adatokkal, és figyelmen kívül hagyjuk azt, hány olyan végrehajtás van futtatás alatt, amely például sorokat fog törölni a relációból. Ekkor egyszerűen csak elfogadjuk a sormutató által visszaadott sorokat.

Az is elképzelhető viszont, hogy nem szeretnénk megengedni az olyan konkurens módosításokat, amelyek hatással lehetnek a sormutatónk által mutatott sorokra. Ehelyett inkább úgy tekintenénk a relációkra, mintha azok egy adott pillanatban levő állapotot tükröznének. A 6.6. alfejezetben szereplő tranzakciók jegyében a reláción futó sormutató kódját a reláción végezzet egyéb műveletekkel együtt sorolható módon kívánjuk végrehajtani. Ezt a hatást érhetjük el a módosításokra *érzéketlen* sormutató deklarálásával.

9.8. példa. Most módosítsuk a 9.8. ábrában megadott beágyazott SQL-utasítás 7. és 8. sorát a következőkre:

```
7) EXEC SQL DECLARE irányítókSormutató INSENSITIVE CURSOR FOR
8)      SELECT nettóBevétel FROM GyártásIrányító;
```

Ekkor az adatbázisrendszer az *irányítókSormutató* nevű sormutató megnyitása és lezárása közben a *GyártásIrányító* táblán elvégzett módosításokat az alkalmazás elől teljesen eltakarja, vagyis ez nem fogja befolyásolni a visszakapott sorokat. □

Egy-egy sormutatóról esetleg biztosan tudhatjuk, hogy végigolvasása során nem módosít egy *R* nevű relációt, amelyből adatokat olvas. Az ilyen sormutatók feldolgozása az ugyanezen táblán végzett módosításokra érzéketlen sormutatókkal egyidejűleg is történhet, hiszen ekkor nem fordulhat elő, hogy azt az *R* relációt módosítják, amelyet az említett erre érzéketlen sormutató is felhasznál. Ha egy sormutató definícióját kiegészítjük a *FOR READ ONLY* kulcsszavakkal, úgy

az országnak a nevét, amelynek a legtöbb hajója veszett oda az illető csatában, és azt az országot, amelynek a legtöbb hajója megsérült az illető csatában, és írassuk ki e két ország nevét.

- c) Kérjük be a felhasználótól a Hajóosztályok tábla egy új sorának az adatait, majd kérjünk be további adatokat: az újonnan beadott hajóosztályba tartozó hajók neveit és felavatásuk időpontját. Vegyük fel ezeket az adatokat a fenti adatbázisba. Ne kelljen minden egyes hajónál beadni, hogy melyik osztályba tartozik, mivel mindegyik a feladat első lépéseként beadtott osztályba tartozik.
- ! d) Vizsgáljuk meg a Csaták, Kimenetelek, Hajók táblákat. Keressünk olyan hajókat, amelyek részt vettek egy csatában, még mielőtt felavatták volna. Ha ilyen hajót találunk, akkor írjuk ki ezt a minden bizonnal hibás adatot a felhasználónak, és kérdezzük meg, hogyan javítsuk az adatokat. Két javítási lehetőséget ajánljunk fel: vagy a csata időpontjának a módosítását, vagy pedig a hajó felavatásának az időpontját (ha szükséges, akkor a felhasználó végezhesse el minden módosítást).

9.4. Sémában tárolt eljárások

Ebben az alfejezetben betekintést kívánunk adni az olvasónak az SQL-szabvány ún. *Tartós, Tárolt Modulok* (Persistent, Stored Modules) részébe, amelyekre SQL/PSM vagy egyszerűen PSM néven szoktak hivatkozni. A PSM az SQL-szabvány legutóbbi módosításának (az SQL:2003-nak) a része. A PSM segítségével eljárásokat fogalmazhatunk meg egy egyszerű, általános célú nyelvben, illetve sémaelemként letárolhatjuk ezeket az adatbázisba. A tárolás után ezeket az eljárásokat használhatjuk SQL-lekérdezésekben, illetve egyéb olyan utasításokban, amelyekkel kizárálag az SQL használatával nem kiszámítható számításokat is elvégezhetjük. minden forgalomban lévő rendszernek van saját PSM-kiterjesztése. Jelen könyünkben az SQL/PSM-szabványt kívánjuk bemutatni, amely tartalmazza a PSM lehetőségeinek fő alapelveit. Emellett segít megérteni a konkrét rendszerhez tartozó nyelvet is. Néhány főbb kereskedelmi rendszer PSM-kiterjesztésére történő hivatkozás is szerepel az irodalomjegyzékben.

9.4.1. PSM-függvények és eljárások létrehozása

A PSM-ben létrehozhatunk ún. *modulokat*, amelyek tartalmazhatják függvények és eljárások definícióinak gyűjteményét, ideiglenes relációk deklarációját és számos választható deklarációs lehetőséget. Az eljárásdeklaráció lényeges elemei az alábbiak:

```
CREATE PROCEDURE <név> (<paraméterek>)
    <lokális deklarációk>
    <eljárás törzse>;
```

A fenti alak ismerősnek tűnhet jó néhány programozási nyelvből, tehát az eljárásdefiníció tartalmaz egy eljárásnevet, egy zárójelezett paraméterlistát, néhány opcionális lokális változódeklarációt és a törzs futtatható kódját. A függvények definíciója is hasonló módon történik azzal a két kivételel, hogy FUNCTION kulcsszót használunk, illetve, hogy egy visszatérési típust is meg kell adnunk. Azaz egy függvénydefiníció elemei a következők:

```
CREATE FUNCTION <név> (<paraméterek>) RETURNS <típus>
    <lokális deklarációk>
    <függvény törzse>;
```

A PSM-eljárások paraméterei mód-név-típus hármasok lesznek, vagyis a paraméter nevét nemcsak a deklarált típusa követi, mint ahogy az a programozási nyelvekben megszokott, hanem a nevet megelőzi egy „mód”, ami lehet IN, OUT vagy INOUT is. Ez a három kulcsszó – sorrendben – azt fejezi ki, hogy az adott paraméter csak bemenet, csak kimenet vagy egyben bemenet és kimenet is. Mivel az IN az alapértelmezett, ezért az IN el is hagyható.

Másrészről a függvényparaméterek csak IN módúak lehetnek, vagyis a PSM tiltja azt, hogy a függvénynek mellékhatásai legyenek, tehát csak a függvény visszatérési értékét használhatjuk a függvényre vonatkozó információként. Ha-bár az eljárások paramétereinek megadtuk az IN módot, a függvények paramétereinek ezt nem szükséges megtennünk.

9.11. példa. Mivel egyenlőre még nem néztük meg azon utasítástípusokat, amelyek szerepelhetnek eljárások, illetve függvények törzsében, ezért az SQL-típusú utasítás használata remélhetőleg nem fog meglepni minket. Ezen utasítások korlátai ugyanazok, mint a beágyazott SQL esetében, amit a 9.3.4. alfejezetben láthattunk, azaz csak egysoros kiválasztás és a sormutató alapú hozzáférés engedélyezett lekérdezésként. A 9.11. ábrán szereplő PSM-eljárás két cím (egy új és egy régi cím) alapján az összes olyan színészhez tartozó cím attribútumot kicseréli az újra, amelynél a színész címe megegyezik régicím értékével.

```
1) CREATE PROCEDURE Kölözés(
2)     IN régicím VARCHAR(255),
3)     IN újcím VARCHAR(255)
4) )
5) UPDATE FilmSzínész
6) SET cím = újcím
7) WHERE cím = régicím;
```

9.11. ábra. Címet megváltoztató eljárás

Az 1. sorban vezetjük be az eljárást és a nevét (Kölözés). A 2. és 3. sorok két input paramétert tartalmaznak, amelyeknek a típusa maximum 255 hosszú, változó méretű karakterlánc (VARCHAR(255)) lesz. Figyeljük meg, hogy ez a típus megegyezik a FilmSzínész reláció cím attribútumának típusával, amelyet a

2.8. ábrán láthatunk. A 4. és 6. sorok között egy hagyományos UPDATE utasítást találhatunk. Azzal a megjegyzéssel, hogy a paraméternevek csak konstansként használhatóak. A befogadó nyelv változói általában ellentétben, amelyek előtt egy kettőspont használata volt kötelező SQL-beli alkalmazásuk esetén (lásd 9.3.2. alfejezet), a paraméterek és a PSM-eljárások, illetve -függvények egyéb, lokális változóihoz nem kell kettőspontot használnunk. □

9.4.2. Néhány egyszerű utasítás alakja PSM-ben

Kezdjük egy olyan, utasításalakok egyvelegével, amelyek megértése egyszerű:

1. *A Call utasítás:* Az eljáráshívás alakja az alábbi:

```
CALL <eljárás neve> (<argumentumlista>);
```

Azaz a nyelvek túlnyomó többségéhez hasonlóan egy CALL kulcsszó, amelyet az eljárás neve és argumentumainak zárójelezett listája követ. Viszont ezen hívás több helyről is történhet. Pontosabban:

- i) Egy befogadó nyelvi programból, amelyben az alakja például lehet a következő:

```
EXEC SQL CALL Foo(:x, 3);
```

- ii) Egy másik PSM-függvény vagy -eljárás utasításaként.
- iii) Egy általános SQL-felületen kiadott SQL-parancsként. Egy ilyen felületen kiadhatunk például egy olyan utasítást, mint a következő:

```
CALL Foo(1, 3);
```

Ennek hatására a Foo tárolt eljárás lefut az 1 és 3 értékű két paraméterével.

Megjegyezzük azt, hogy a fentiek függvényhívásra nem megengedettek. PSM-ben a C-hez hasonló függvényhívás létezik, vagyis a függvényt és a megfelelő argumentumokat egy kifejezés részeként használhatjuk.

2. *Visszatérési utasítás:* Alakja a következő:

```
RETURN <kifejezés>;
```

Ez az utasítás csak függvényekben szerepelhet. Először kiértékelni a kifejezés értékét, majd ennek eredményét értékül adja a függvény visszatérési értékének. A hagyományos programozási nyelvekkel ellentétben viszont a PSM visszatérési értéke *nem* fejezi be a függvényt, hanem folytatja a vezérlést a következő utasítással, és így az is előfordulhat, hogy a visszatérési érték a függvény lefutása előtt megváltozik.

3. *Lokális változók deklarációja:* Az alábbi utasításforma egy adott nevű, adott típusú változót deklarál:

```
DECLARE <név> <típus>;
```

Ez a változó lokális, és így az értékét az ABKR a függvény vagy eljárás lefutását követően nem őrzi meg. A deklaráció meg kell előzze a függvény vagy az eljárás törzsében szereplő, futtatható utasításokat.

4. *Értékadó utasítások:* Az értékadás alakja a következő:

```
SET <változó> = <kifejezés>;
```

A bevezetőként szereplő SET kulcsszótól eltekintve egy PSM-beli értékadás nagyon hasonlít más nyelvek értékadásához. Az egyenlőség jobb oldalán lévő kifejezés kiértékelésre kerül és a bal oldalon szereplő változó értéke a kapott érték lesz. Hiányzó kifejezés esetén NULL értéket kap. A kifejezés akár egy SQL-lekérdezés is lehet, amennyiben egy értékkel tér vissza.

5. *Utasításcsoportok:* Pontosvesszővel végződő utasítások egy listáját fogalmazhatjuk meg egy BEGIN és egy END kulcsszavak között. Ez a szerkezet egy utasításként kezelendő, és így bárhol előfordulhat, ahol egy egyszerű utasítás is előfordulhat. Azaz, mivel egy eljárás, illetve függvény törzse egy egyszerű utasítás kell legyen, így ezzel utasítások sorozatát tehetjük be a törzsbe a BEGIN és az END kulcsszavak közé.
6. *Utasításcímek:* Egy utasítás címkézéséhez az utasítás elő írunk egy nevet (a címkét), amit egy kettőspont követ.

9.4.3. Elágazásutasítások

Az első összetettebb PSM-utasítástípusunkhoz tekintsük az if utasítást, amelynek alakja kicsit furcsának tűnhet, mivel eltér a C-ben vagy más nyelvben megszokottaktól a következő értelemben:

1. Az utasítás az END IF kulcsszóval végződik.
2. Az else záradékba ágyazott if utasításokat az ELSEIF kulcsszóval vezethetjük be.

Azaz a 9.12. ábra egy if utasítás általános alakját mutatja be. A feltétel bármilyen olyan logikai értékű kifejezés lehet, amely SQL-utasítások WHERE záradékában is szerepelhet. minden utasításlista pontosvesszővel végződő utasításokat tartalmazhat, és nem kell hozzá BEGIN...END környezet sem. Végezetül az ELSE utasítás és ennek utasítása(i) opcionálisak, ugyanis az IF...THEN...END IF önmagában és ELSEIF-ekkel együtt is helyes.

```

IF <feltétel> THEN
    <utasításlista>
ELSEIF <feltétel> THEN
    <utasításlista>
ELSEIF
    ...
ELSE
    <utasításlista>
END IF;

```

9.12. ábra. Az if utasítás egy alakja

9.12. példa. Írunk függvényt, amely az é év és az s stúdió bementre vár, és pontosan akkor és csak akkor tér vissza igaz értékkel, ha az s stúdió legalább egy vígjátékot produkált vagy egyetlen filmet sem adott ki az é évben. A 9.13. ábrán szerepel az ehhez tartozó kód.

- 1) CREATE FUNCTION Vígjáték(é INT, s CHAR(15))
 RETURNS BOOLEAN
- 2) IF NOT EXISTS(
 3) SELECT * FROM Filmek WHERE év = é AND
 stúdióNév = s)
 4) THEN RETURN TRUE;
 5) ELSEIF 1 <=
 6) (SELECT COUNT(*) FROM Filmek WHERE év = é AND
 stúdióNév = s AND műfaj = 'vígjáték')
 7) THEN RETURN TRUE;
 8) ELSE RETURN FALSE;
 9) END IF;

9.13. ábra. Ha van legalább egy film, akkor legalább az egyiknek
vígjátéknak kell lennie

Az 1. sor vezeti be a függvényt és annak argumentumait. Nem kell megadjuk az argumentumok módját, mivel egy függvénynek csak IN módú argumentumai lehetnek. A 2. és 3. sorok azt az esetet vizsgálják, hogy az s stúdiónak az é évben volt-e egyáltalán filmje. Amennyiben nem, a 4. sorban a visszatérési értéket TRUE-ra állítjuk. Megjegyezve azt, hogy a 4. sor nem eredményezi a függvény visszatérését. Technikailag az if utasítás vezérlésével folytatódik a végrehajtás, ami azt eredményezi, hogy a 4. sorból a 9. sorra ugrik a vezérlés, ahol a függvény befejeződik és visszatér.

Ha s stúdió készített filmet az é évben, akkor az 5. és 6. sor megyizsgálja, hogy ezek közül volt-e legalább egy vígjáték. Ha volt, akkor most a 7. sor ha-

tására itt is igaz értékre állítjuk a visszatérési értéket. A megmaradó esetben pedig, amikor az *s* stúdió csak olyan filmet adott ki, amely nem vígjáték műfajú volt, akkor a 8. sorban FALSE-ra állítjuk a visszatérési értéket. □

9.4.4. Lekérdezések PSM-ben

PSM-ben több módja is van a select-from-where típusú lekérdezések használatának:

1. Feltételekben vagy – általánosabban – minden olyan helyen, ahol SQL-ben is megengedett lenne, szerepelhetnek alkérdezések. Például a 9.13. ábra 3., illetve 6. sorában már láthattunk erre két példát.
2. Az olyan lekérdezések, amelyek egyetlen értéket adnak vissza, használhatók értékkedő utasítások jobb oldalán.
3. PSM-ben egy egysoros select utasítás is megengedett. Emlékezzünk vissza arra, hogy az ilyen utasításnak volt egy INTO záradéka, amely azt a változót adta meg, amelybe az egyetlen visszatérő sor került be. Ez a változó lehet egy lokális változó vagy egy PSM-eljárás paramétere is. Ennek az általános alakját a beágyazott SQL összefüggésében már a 9.3.5. alfejezetben megtárgyalunk.
4. Deklarálhatunk és használhatunk sormutatókat is. Lényegében ezt ugyanúgy tehetjük meg, mint ahogyan azt a 9.3.6. alfejezetben megbeszéltek szerint a beágyazott SQL-re tettük. A sormutató deklarációja, OPEN, FETCH és CLOSE utasítások ugyanazok, mint korábban, a következő eltérésekkel:
 - a) EXEC SQL nem szerepelhet az utasításokban, illetve
 - b) a változóknál, lévén lokálisak, nem kell kettőspontot használni.

9.13. példa. A 9.14. ábrán szereplő egysoros kiválasztásnak PSM-be átírt változata a 9.7. ábrán látható egy képzelt eljárás definíójában. Figyeljük meg, hogy ugyanezt a hatást érhetnék el egy értékadás segítségével is, mivel az egysoros kiválasztásunknak csak egyetlen komponense van, azaz írhatnánk a következőt:

```
SET elnökNettóFizetése = (SELECT nettóBevétel  
    FROM Stúdió, GyártásIrányító  
    WHERE elnökAzon = azonosító AND Stúdió.név = stúdióNév);
```

A sormutatós példát a következő alfejezetben tárgyaljuk, amelyben megtanuljuk a PSM-ciklus használatát. □

nemcsak deklarál egy sormutatót, hanem jó néhány „kényelmetlen részletet” is kezel helyettünk: megnyitja és lezárja a sormutatót, bekéri a sorokat és ellenőrzi, hogy van-e egyáltalán bekérődő sor. Viszont mivel nem mi kérjük be a sorokat, így olyan változó(ka)t sem adhatunk meg, amely(ek)be egy sor komponense(i)t elhelyezhetnénk. Vagyis a PSM a lekérdezés eredményeinek attribútumainál meghatározott neveket használja megfelelő típusú lokális változókként.

```

FOR <ciklus neve> AS <sormutató neve> CURSOR FOR
    <lekérdezés>
DO
    <utasításlista>
END FOR;

```

9.16. ábra. A PSM for ciklusa

9.15. példa. Vegyük a 9.15. ábrán szereplő eljárás átírását for ciklussal. A kapott kód a 9.17. ábrán látható, ami sok tekintetben hasonlít az elődjéhez. Az eljárás 1–4. sorok közötti deklarációja megegyezik a 9.17. ábrán lévőkkel, mint ahogy a **filmSzámláló** nevű lokális változó 5. sorbeli deklarációja is ugyanolyan, mint korábban.

Itt viszont nincsen szükség sem az eljárás deklarációs területén szereplő sormutató deklarációjára, sem a **Nincs_Több** feltétel definíciójára. A 6–8. sorokban a korábbiakkal azonos kezdeti értékadás történik. Ezt követően a 9. sorban láthatjuk a for ciklust, amely az előzőhöz hasonlóan definiál egy **filmSormutató** nevű sormutatót. A 11–13. sorok közötti részben látható a ciklus törzse. Vegyük észre azt, hogy a 12–13. sorban a hosszt a sormutatón keresztül a **hossz** nevű attribútumból kapjuk meg, nem pedig egy új **Hossz** nevű lokális változóból, amely a jelen eljárásban nem is szerepel. A 15–16. sorok – a korábbi eljárás verzióval szinkronban – kiszámítják a kimeneti változók helyes értékét. □

9.4.7. PSM-kivételek

Az SQL-rendszer a hibákat egy ötjegyű, nem csak nulla számjegyekből álló **SQLSTATE** nevű karakterlánc beállításával jelzi. Az ilyen kódokra már láthatunk példát: '02000' jelzi, hogy nem talált sort. Egy másik példaként említhető a '21000', ami azt fejezi ki, hogy egy egysoros lekérdezés több mint egy sort eredményezett.

A PSM egy ún. *kivitelkezelő* kódrészlet deklarálását engedi meg számunkra, amely minden esetben kiváltódik, ha ezen hibákódok egy listájának valamely tagja előfordul az **SQLSTATE**-ben az utasítás vagy utasítássorozat végrehajtása során. minden egyes kivitelkezelőhöz egy kódblokk tartozik, amelyet egy **BEGIN...END** foglal magába. A kezelő ezen blokkban szerepel, és csak a blokkban lévő utasításokat használja fel.

```

1) CREATE PROCEDURE ÁtlagEltér(
2)     IN s CHAR(15),
3)     OUT átlag REAL,
4)     OUT szórás REAL
    )
5) DECLARE filmSzámláló INTEGER;

    BEGIN
6)     SET átlag = 0.0;
7)     SET szórás = 0.0;
8)     SET filmSzámláló = 0;
9)     FOR filmCiklus AS filmSormutató CURSOR FOR
        SELECT hossz FROM Filmek WHERE stúdióNév = s;
10)    DO
11)        SET filmSzámláló = filmSzámláló + 1;
12)        SET átlag = átlag + length;
13)        SET szórás = szórás + hossz * hossz;
14)    END FOR;
15)    SET átlag = átlag/filmSzámláló;
16)    SET szórás = szórás/filmSzámláló - átlag * átlag;
END;

```

9.17. ábra. A filmhosszak átlagának és szórásának kiszámítása
for ciklus segítségével

A kezelő komponensei a következők:

1. Azon kivétfeltételek listája, amelyek kiváltódásuk esetén meghívják a kivételkezelőt.
2. Az a kód, amelyet lefuttat, amikor egy hozzá tartozó kivétel végrehajtó-dik.
3. Egy utalás arra nézve, hogy a kezelő lefutása után hol folytatódjon a végrehajtás.

A kezelő deklarációja az alábbi formát ölti:

```
DECLARE <hova menjen> HANDLER FOR <feltétellista>
    <utasítás>
```

A „hova menjen” lehetőségek a következők lehetnek:

- a) **CONTINUE**, amely azt fejezi ki, hogy a kezelő deklarációjában szereplő utasítás lefutását követően folytatjuk a hibát kiváltó utasítást követő utasítás végrehajtával.

Miért szükséges neveket használni a for ciklusokban?

Vegyük észre, hogy a `filmCiklus`-t és a `filmSormutató`-t ugyan deklaráltuk a 9.17. ábra 9. sorában, ám az eljárás során sehol sem használtuk őket. Ennek ellenére viszont minden for ciklushoz, minden iterálható sormutatóhoz be kell vezetnünk őket. Ennek oka, hogy a PSM-értelmező minden for ciklust egy olyan hagyományos ciklusra fordít le, amely nagyon hasonlít a 9.15. ábrán szereplőhöz, és az ilyen kódoknál pedig minden kettőnek nevet kell adni.

- b) EXIT, amely azt fejezi ki, hogy a kezelő utasításának lefutását követően, a vezérlés a kezelőhöz deklarált BEGIN...END blokknál marad. Majd az ezt a blokkot követő utasítás futtatásával folytatja a futást.
- c) UNDO, amely ugyanolyan, mint az EXIT, azzal a különbséggel, hogy az adatbázisban vagy a lokális változókban a blokk végrehajtásának eredményeként történt változásokat „visszavonja”. Vagyis, ezen utasításoknak nem lesz hatásuk, így tulajdonképpen ugyanolyanok lesznek, mintha végre sem hajtottuk volna őket.

A „feltételeknek” a feltételeknek egy olyan vesszőkkel elválasztott listája, amelyben akár deklarált feltételek is szerepelhetnek, mint például a 9.15. ábra 5. sorában található `Nincs_Több`, vagy az `SQLSTATE`-ből és egy 5 karakteres karakterláncból álló kifejezések.

9.16. példa. Írunk egy olyan PSM-függvényt, amely egy filmargumentumra visszaadja annak gyártási évét. Amennyiben nincs, vagy több azonos nevű film is van, akkor NULL-lal térjen vissza. Ennek kódja a 9.18. ábrán található.

```
1) CREATE FUNCTION KiadásiÉv(fc VARCHAR(255))
   RETURNS INTEGER

2) DECLARE Nincs_Találat CONDITION FOR SQLSTATE '02000';
3) DECLARE Túl_Sok CONDITION FOR SQLSTATE '21000';

BEGIN
4)   DECLARE EXIT HANDLER FOR Nincs_találat, Túl_Sok
5)     RETURN NULL;
6)   RETURN (SELECT év FROM Filmek WHERE filmcím = fc);
END;
```

9.18. ábra. Kivételkezelés, ahol egy egysoros kiválasztás nem egy sorral tér vissza

A 2., illetve 3. sorok szimbolikus feltételeket deklarálnak. Nem feltétlen kelene használnunk ezeket a definíciókat, mivel helyettük az SQL-állapotokat is ugyanígy használhatnánk a 4. sorban. A 4., 5. és 6. sorok egy olyan blokkot alkotnak, amelyben először deklaráljuk a hibakezelőket arra a két esetre vonatkozóan, amikor üres sort, illetve egynél több sort kapunk vissza. Az 5. sorban lévő kezelő egyszerűen csak beállítja NULL-ra a visszatérési értéket.

A 6. sorban lévő utasítás pedig elvégzi a KiadásiÉv függvény feladatát. Ezt tulajdonképpen egy egész értékkal visszatérő SELECT utasításnak feltételezzük, mivel a KiadásiÉv függvény visszatérési értéke is ez. Ha pontosan egy *fc* című film van, ahol *fc* a függvény bemeneti paramétere, akkor a függvény ennek a kiadási évével tér vissza. Habár, ha a 6. sor végrehojtása kivételt vált ki – attól függően, hogy egy sor sincs vagy több sor is van –, akkor a kezelő meghívódik, és a visszatérési érték NULL-ra változik. Sőt, mivel a kezelő EXIT kezelő, ezért a vezérlés az END kulcsszó utánra adódik át, és mivel ez a függvény lefutását jelenti, ezért a KiadásiÉv befejeződik, és a visszatérési érték pedig NULL lesz. □

9.4.8. PSM-függvények és -eljárások használata

A 9.4.2. alfejezetben említettek alapján egy PSM-eljárást vagy -függvényt meg-hívhatunk egy beágyazott SQL-beli programból, magából a PSM-kóból vagy az általános felületen kiadott hagyományos SQL-utasításokból. Egy eljárást egy CALL előtaggal hívhatunk meg, a függvényeket pedig valamilyen kifejezésnek a részeként használhatjuk, például a WHERE záradékban. A most következőkben egy példát mutatunk arra, hogyan hívható meg egy függvény egy kifejezésen belül.

9.17. példa. Tegyük fel, hogy a 9.18. ábrán szereplő KiadásiÉv függvényt modulként tartalmazza az adatbázissémánk. Képzeljük azt, hogy egy általános felület előtt állunk, és azt a tényadatot kívánjuk rögzíteni, hogy Denzel Washington az *Emlékezz a titánokra!* című film szereplője volt, ám a film elkészítésének évét elfelejtettük. Mivel csak egy ilyen című film van, ami a Filmek relációban meg is található, egy előzetesen kiadott lekérdezés megtekintése nélkül visszakaphatjuk ezt az értéket. Ezt elérhetjük az általános felületen kiadott alábbi beszúrás segítségével:

```
INSERT INTO SzerepelBenne(filmCím, filmÉv, színészNév)
VALUES('Emlékezz a titánokra!', 
       KiadásiÉv('Emlékezz a titánokra!'), 
       'Denzel Washington');
```

Mivel a KiadásiÉv függvény NULL értékkel tér vissza, ha az *Emlékezz a titánokra!* című film nem pontosan egyszer szerepel, így előfordulhat, hogy a beszúrás eredményének középső komponense NULL értékű lesz. □

9.4.9. Feladatok

9.4.1. feladat.

Használjuk a korábbi filmadatbázisunkat, azaz:

Filmek(filmcím, év, hossz, műfaj, stúdióNév, producerAzon)

SzerepelBenne(filmCím, filmÉv, színészNév)

FilmSzínész(név, cím, nem, születésiDátum)

GyártásIrányító(név, cím, azonosító, nettóBevétel)

Stúdió (név, cím, elnökAzon)

Írunk PSM-függvényt vagy -eljárást a következő feladatokhoz:

- a) Adott nevű filmstúdióhoz állítsuk elő a stúdió elnökének nettó bevételét.
- b) Adott név és cím esetén adjuk vissza az 1 értéket, ha a kapott személy filmszínész és nem gyártásirányító, a 2 értéket, ha a kapott személy gyártásirányító és nem filmszínész, a 3 értéket, ha a kapott személy gyártásirányító és filmszínész is egyben, illetve a 4 értéket, ha egyik sem.
- c) Adott stúdiónév mellett jelölje a stúdió két leghosszabb filmje címének két kimeneti paraméterét. Rendeljünk NULL értéket az egyik vagy minden paraméterhez azokban az esetekben, amikor vagy az egyikre, vagy egyikre sincs megfelelő film (például, amikor csak egyetlen film tartozik az adott stúdióhoz, akkor nem lehet „második leghosszabb” film).
- d) Adott a filmszínész neve, keressük meg a legkorábbi (év szerint) olyan filmet, amelyben szerepelt, és amely 120 percnél hosszabb volt. Ha nincs a feltételnek megfelelő film, akkor a visszatérési év értéke legyen 0.
- e) Adott lakkímhez keressük meg azt az egyértelműen meghatározható színész nevét, akinek ez a lakkíme. Ha nincs, vagy egynél több ilyen név van, akkor NULL-t adjunk vissza.
- f) Adott a színész neve, töröljük ki a FilmSzínész táblából a hozzá tartozó bejegyzés(ek)eit, és a Filmek, illetve SzerepelBenne táblákból is töröljük az összes ezzel kapcsolatos bejegyzést is.

9.4.2. feladat.

Írjuk meg az alábbi PSM-függvényeket vagy -eljárásokat a 2.4.1. feladat adatbázissémája alapján:

Termék(gyártó, modell, típus)

PC(modell, sebesség, memória, merevlemez,
cserélhető lemez, ár)

Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)

Nyomtató(modell, színes, típus, ár)

- a) Egy árargumentum alapján adjuk vissza azon PC-modellszámokat, amelyeknek az ára nagyságrendileg akkora, mint az adott ár.
- b) Egy gyártót, egy modellt és egy árat argumentumként használva adjuk meg a modellnek megfelelő összes terméktípusnak az árát.
- c) Vegyük úgy, hogy az argumentum tartalmazza a modell, a sebesség, a cserélhető lemez, illetve az ár információkat, és szűrjuk be ezeket az információkat a PC relációba. Ám, ha már volt egy PC, amelynek modellszáma ugyanaz (egy kulcs ütközési megszorítás feltételezése esetén a beszúrás ekkor kiváltja a '23000' értékű SQLSTATE hibát), akkor addig növeljük egyesével a modellszámot, amíg találunk egy olyan modellszámot, amely már nem PC-modellszám.
- d) Adott ár alapján állítsuk elő azon PC-knek, laptopoknak és nyomtatóknak a számát, amelyek az adott összegnél drágábbak.

9.4.3. feladat. Írjuk meg az alábbi PSM-függvényeket vagy -eljárásokat a 2.4.3. feladat adatbázissémája alapján:

Hajóosztályok(osztály, típus, ország, ágyúSzáma,
kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)

- a) Egy hajó tűzereje megközelítőleg arányos a rajta levő fegyverek számának és a fegyverkaliberek köbének a szorzatával. Keressük meg a legmagasabb tűzerejű hajóosztályt.
- b) Egy adott csata neve alapján határozzuk meg azt a két országot, amelyeknek a hajói részt vettek a csatában. Amennyiben több mint két ország, vagy egy sem szerepelt a csatában, akkor minden kettőnek legyen az értéke NULL.
- c) Legyenek az argumentumok a következők: egy új hajóosztálynév, egy típus, egy ország, egy ágyúszám, egy kaliberérték, egy vízkiszorítási érték. Adjuk hozzá ezeket az információkat a Hajóosztályok relációhoz, illetve adjunk hozzá egy ilyen hajóosztálynévvel rendelkező hajót a Hajók relációhoz is.
- d) Adott hajónévhez határozzuk meg, hogy részt vett-e a hajó olyan csatában, amelynek dátuma korábbi, mint a hajó felavatási dátuma. Ha igen, akkor mind a felavatás, mind a csata dátumát állítsuk nullára.

! 9.4.4. feladat. A 9.15. ábrán egy trükkös formulát használtunk egy x_1, x_2, \dots, x_n számsorozat szórásának a kiszámításához. Megjegyezzük, hogy a számoknak az átlaguktól való eltérésük négyzetének átlaga adja a szórást. A

szórás eredeti képlete: $(\sum_{i=1}^n (x_i - \bar{x})^2)/n$, ahol \bar{x} a $(\sum_{i=1}^n x_i)/n$ átlagot jelöli. Bizonyítsuk be, hogy a 9.15. ábráról származó

$$\left(\sum_{i=1}^n (x_i)^2 \right) / n - \left(\left(\sum_{i=1}^n x_i \right) / n \right)^2$$

képlet is ugyanazt az értéket adja, mint a szórás képlete.

9.5. Hívásszintű interfészek használata

Hívásszintű interfész (Call Level Interface, CLI) használatakor szokványos befogadó nyelvű kódot írunk, és olyan függvénykönyvtárakat használunk, amelyek lehetővé teszik az adatbázishoz való kapcsolódást, annak elérését és SQL-utasítások átadását. A mostani megközelítés és a beágyazott SQL használata közti különbség – bizonyos értelemben – csupán kozmetikai, mivel az előfeldolgozó a beágyazott SQL-hívásokat az SQL/CLI-szabványban szereplő függvényekhez hasonló könyvtári függvényekkel helyettesíti.

Három példát adunk hívásszintű interfészre. Ebben a fejezetben a szabványos SQL/CLI-interfészt tekintjük át, ami az ODBC (Open Database Connectivity, nyílt adatbázis-összekapcsolhatóság) egy átdolgozása. A következő szakaszban áttekintjük a JDBC-t (Java Database Connectivity, Java adatbázis-összekapcsolhatóság), amely Java-programokat köt össze adatbázisokkal egy osztálygyűjtemény segítségével. Végezetül átnézzük a PHP-t, amely a HTML-ben írt weblapokból adatbázishoz történő hozzáféréshez ad egy keretmódszert.

9.5.1. Bevezetés az SQL/CLI-be

Egy C nyelven írt, SQL/CLI-t (a továbbiakban csak CLI-t) használó program beemeli az `sqlcli.h` fejállományt (header), melyből nagy számú függvényt, típusdefiníciót, struktúrát és szimbolikus konstanst nyer. Ezután a program négyfajta rekord (C-ben ezeket „struct”-nak hívjuk) létrehozására és kezelésére képes:

1. *Környezetek (environments)*. Az ilyen típusú rekordot az alkalmazás (a kliens) hozza létre egy vagy több adatbázis-kapcsolat előkészítéséhez.
2. *Kapcsolatok (connections)*. Ezen létrehozott rekordok egyike kapcsolja az alkalmazási programot az adatbázishoz. minden egyes kapcsolat valamely környezeten belül létezik.
3. *Utasítások (statements)*. Az alkalmazói program létrehozhat egy vagy több utasításrekordot. Mindegyik egy-egy SQL-utasításról hordoz információt, beleértve egy magában foglalt sormutatót, ha az utasítás egy lekérdezés. Különböző időpontokban ugyanaz a CLI-utasítás különböző SQL-utasításokat reprezentálhat. minden egyes utasítás valamely kapcsalon belül van értelmezve.

10.1.2. feladat. Mutassuk meg a 4. és a 6. lépés végrehajtása után kialakult engedélyezési diagramot a 10.6. ábrán leírt műveletek végrehajtásakor. Tegyük fel, hogy a p jogosultság olyan relációra vonatkozik, melynek tulajdonosa A .

Lépés	Végrehajtója	Tevékenység
1.	A	GRANT p TO B WITH GRANT OPTION
2.	A	GRANT p TO C
3.	B	GRANT p TO D WITH GRANT OPTION
4.	D	GRANT p TO B, C, E WITH GRANT OPTION
5.	B	REVOKE p FROM D CASCADE
6.	A	REVOKE p FROM C CASCADE

10.6. ábra. A 10.1.2. feladathoz tartozó műveletsorozat

10.1.3. feladat. Mutassuk meg az 5. és a 6. lépés végrehajtása után kialakult engedélyezési diagramot a 10.7. ábrán leírt műveletek végrehajtásakor. Tegyük fel, hogy a p jogosultság olyan relációra vonatkozik, melynek tulajdonosa A .

Lépés	Végrehajtója	Tevékenység
1.	A	GRANT p TO B, E WITH GRANT OPTION
2.	B	GRANT p TO C WITH GRANT OPTION
3.	C	GRANT p TO D WITH GRANT OPTION
4.	E	GRANT p TO C
5.	E	GRANT p TO D WITH GRANT OPTION
6.	A	REVOKE GRANT OPTION FOR p FROM B CASCADE

10.7. ábra. A 10.1.3. feladathoz tartozó műveletsorozat

! 10.1.4. feladat. Mutassuk meg az alábbi lépések végrehajtása után kialakult engedélyezési diagramot. Tételezzük fel, hogy a p jogosultság egy olyan táblára vonatkozik, amelynek tulajdonosa az A felhasználó.

Lépés	Végrehajtója	Tevékenység
1.	A	GRANT p TO B WITH GRANT OPTION
2.	B	GRANT p TO B WITH GRANT OPTION
3.	A	REVOKE p FROM B CASCADE

10.2. Rekurzió az SQL-ben

Az SQL-99 szabványa tartalmazza a lekérdezések rekurzív meghatározásának lehetőségét. Létezik legalább egy nagy rendszer – az IBM DB2 –, amely implementálja az SQL-99 előírásait annak ellenére, hogy az SQL-99 szabványnak nem „lényegi” része az, hogy minden ABKR implementálja ezt a lehetőséget. Ebben a részben ezzel foglalkozunk.

10.2.1. Rekurzív relációk definiálása az SQL-ben

A WITH utasítás segítségével SQL-ben megfogalmazhatjuk ideiglenes relációt a rekurzív vagy nem rekurzív definícióját. Rekurzív reláció megadásánál a relációt a WITH utasításon belül használhatjuk. A WITH utasítás egyszerű alakja:

WITH R AS $\langle R \text{ definíciója} \rangle \langle R\text{-et használó lekérdezés} \rangle$

Tehát definiálunk egy R nevű ideiglenes relációt, majd használjuk egy lekérdezésben. Az ideiglenes relációk csak a WITH utasításhoz tartozó lekérdezésekben belül használhatók.

Általában több reláció is definiálható a WITH után, a definíciókat vesszővel elválasztva. A definíciók közül bármelyik lehet rekurzív. A definiált relációk lehetnek kölcsönösen rekurzívak, azaz mindenkit néhány másik függvényében definiáljuk (akár önmagukkal is). minden olyan relációt, amely rekurzív definícióban vesz részt, a RECURSIVE kulcsszó kell hogy megelőzzön. A WITH utasítás általános alakja a 10.8. ábrán található.

```
WITH
  [RECURSIVE]  $R_1$  AS  $\langle R_1 \text{ definíciója} \rangle,$ 
  [RECURSIVE]  $R_2$  AS  $\langle R_2 \text{ definíciója} \rangle,$ 
  ...
  [RECURSIVE]  $R_n$  AS  $\langle R_n \text{ definíciója} \rangle$ 
 $\langle R_1, R_2, \dots, R_n \text{ relációkat tartalmazó lekérdezés} \rangle$ 
```

10.8. ábra. A WITH utasítás több ideiglenes relációra vonatkozó definíciója

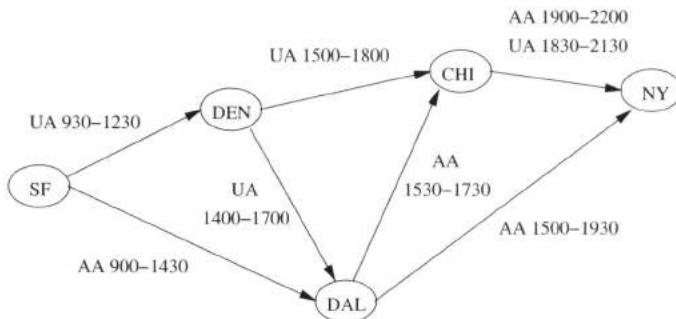
10.8. példa. Sok olyan példa található a gráfok útjainak tanulmányozása esetén, amely rekurziót használ. A 10.9. ábra egy gráfiot reprezentál, amely két fiktív légitársaság (*Untried Airlines* (UA), *Arcane Airlines* (AA)) San Francisco, Denver, Dallas, Chicago és New York városok között közelkedő járatait tartalmazza. A gráfiot az alábbi relációval ábrázolhatjuk:

Járatok(légitársaság, honnan, hova, indulás, érkezés)

Az ábra konkrét sorait a 10.9. ábra szemlélteti.

A legegyszerűbb rekurzív kérdés, amelyet feltehetünk, hogy „Mely (x, y) várospárokra lehetséges egy vagy több átszállással eljutni x városból y városba?”. Mielőtt megrírnánk a rekurzív SQL-lekérdezést, érdemes a rekurziót az 5.3. alfejezetben használt jelöléssel Datalogban is kifejezni, hiszen számos elképzelést, mint a rekurziót is, könnyebb kifejezni Datalogban, mint SQL-ben. Szükségünk lehet a korábbi részben tárgyalt terminológiá áttekintésére. A következő két Datalog-szabály írja le az $\text{Eljut}(x, y)$ relációt, amely már a valódi várospárokat fogja a tartalmazni:

1. $\text{Eljut}(x, y) \leftarrow \text{Járatok}(l, x, y, i, é)$
2. $\text{Eljut}(x, y) \leftarrow \text{Eljut}(x, z) \text{ AND } \text{Eljut}(z, y)$



10.9. ábra. Néhány járat egy térképe

légitársaság	honnán	hova	indulás	érkezés
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

10.10. ábra. A Járatak reláció sorai

Az első szabály azt fejezi ki, hogy az Eljut reláció minden olyan várospárt tartalmaz, amely városok között közvetlen járat van az elsőből a másodikba. Az l légitársaság, az i indulási idő és az \acute{e} érkezési idő tetszőlegesek a szabályban. A második szabály azt fogalmazza meg, hogy ha az x városból már eljutottunk z városba, és z városból eljuthatunk y városba, akkor x -ből is el tudunk jutni y -ba.

Egy rekurzív szabály kiértékeléséhez a Datalog-szabályok többszöri alkalmazására van szükségünk. Kezdetben az Eljut relációt üresnek feltételezzük. Az első szabály alkalmazásával a következő Eljut párokat kapjuk: (SF, DEN) , (SF, DAL) , (DEN, CHI) , (DEN, DAL) , (DAL, CHI) , (DAL, NY) és (CHI, NY) . Ez pontosan az a hét irányított él, amelyet a 10.9. ábra is tartalmaz.

A következő menetben a második szabályt alkalmazzuk rekurzív módon azért, hogy összevonhassuk az olyan épárokat, amelyeknél az egyik végpontja a másik kezdőpontja lesz, ekkor a következőket kapjuk: (SF, CHI) , (DEN, NY) és (SF, NY) . A harmadik menet már az összes egy élből vagy két élből álló párokot rakja össze, és ezzel maximum négy hosszúságú irányított élből álló utakat nyerünk. A mostani diagramnál viszont ekkor már nem kapunk újabb párokat, ezért az Eljut reláció tíz olyan (x, y) párt fog tartalmazni, amelyekben a 10.9. ábra alapján az x -ből elérhető lesz az y .

A fenti Datalog-szabályok alapján megadható az `Eljut` reláció ekvivalens SQL-definíciója. Az SQL-lekérdezés képezi az `Eljut` definícióját egy `WITH` utasítás segítségével, és a `WITH` utasítást kibővítiük a kívánt lekérdezéssel. A példánkban az eredmény az egész `Eljut` reláció volt, megjegyezve, hogy a lekérdezésben feltételeket is kiköthetnénk az `Eljut`-ra, például azt, hogy csak a Denverből elérhető városokat kapjuk meg.

```

1) WITH RECURSIVE Eljut(honnan, hova) AS
2)           (SELECT honnan, hova FROM Járatok)
3)           UNION
4)           (SELECT R1.honnan, R2.hova
5)             FROM Eljut R1, Eljut R2
6)             WHERE R1.hova = R2.honnan)
7) SELECT * FROM Eljut;
```

10.11. ábra. Légi kapcsolatban lévő városokat meghatározó SQL-lekérdezés

A 10.11. ábra lekérdezése bemutatja az `Eljut` relációt meghatározó SQL-lekérdezést. Az 1. sor bevezeti az `Eljut` relációt, és a 2–6. sorok között található a reláció definíciója.

Mint ahogyan az `Eljut` megadásánál két Datalog-szabály volt, úgy a definíció is két lekérdezés egyesítéséből áll. A 2. sorbeli lekérdezés az unió első tagja és az első (vagy alap-) szabálynak felel meg. Azt állítja, hogy minden egyes `Járatok`-beli sor esetén a második és a harmadik komponens (`honnan` és `hova`) által alkotott sor megtalálható legyen az `Eljut` sorai között.

A 4–6. sorok közötti lekérdezés az `Eljut` definíciójának második (vagy rekurzív) szabályának felel meg. A két `Eljut` részcél 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.

Végül a 7. sor írja le az egész reláció által eredményezett relációt, mely az `Eljut` reláció másolata. Egy lehetőség, hogy a 7. sort kicseréljük egy bonyolultabb lekérdezéssel, mint például:

```
7) SELECT hova FROM Eljut WHERE honnan = 'DEN';
```

amely a Denverből elérhető városokat adná meg. □

10.2.2. Problémás kifejezések rekurzív SQL-ben

Az SQL-szabvány nem engedélyezi, hogy a `WITH` záradék után tetszőleges kölcsönösen rekurzív relációk gyűjteménye álljon. Van egy kis probléma az utóbbival, és ezért a szabvány csak a *lineáris* rekurziót támogatja. A Datalog-kifejezéseknel egy lineáris rekurzióban minden szabálynak csak egyetlen olyan részcélja lehet,

Kölcsönös rekurzió

Létezik egy gráfelméleti módszer annak ellenőrzésére, hogy két reláció vagy predikátum kölcsönösen rekurzív-e. Egy *függősségi 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, amelynek a fejében B van. Az SQL esetében A megjelenne B definíciójában, általában a `FROM` záradékban (akár egy részkérdésben is). Ha létezik olyan irányított kör, amelyben megtalálhatók az R és S csúcsok, akkor R és S kölcsönösen rekurzív. A legtöbb esetben az R csúcsba hurokél vezet, azaz R rekurzívan függ önmagától.

amely kölcsönösen rekurzív a fej predikátumával. Figyeljük meg a 10.8. példában, hogy a 2. szabálynak két részcélja van azzal az `Eljut` predikátummal, amely kölcsönösen rekurzív kapcsolatban áll a fej predikátummal (hiszen a predikátum minden kölcsönösen rekurzív viszonyban van önmagával, lásd a Kölcsönös rekurzió keretezett résznél). Azaz technikailag az ABKR visszautasíthatja a 10.11. ábrán lévő utasítás végrehajtását, miközben illeszkedik a szabványhoz.¹

Emellett van egy fontosabb megsorítás is az SQL-rekurziókra, amelynek megsértése ahoz vezet, hogy a lekérdezés feldolgozója nem képes értelmes módon végrehajtani a rekurziót. Egy SQL-rekurzió akkor legális, ha az R rekurzív relációdefiníciója csak olyan kölcsönösen rekurzív S relációt használ (R -et is beleérte), amelynek a használata S -ben „monoton”. Egy S használat *monoton*, ha S -hez adunk egy tetszőleges sort, akkor ez azt eredményezheti, hogy R -be is bekerül egy vagy több sor, vagy hogy R változatlan marad, de soha nem eredményezheti azt, hogy R valamely sora kitörlődjön. A most következő példa azt szemlélteti, mi történhet, amikor figyelmen kívül hagyjuk a monotonitási kritériumot.

10.9. példa. Tegyük fel, hogy R egy unáris (egyattribútumos) reláció, és egyetlen sora van: a (0). Használjuk R -et EDB-relációként az alábbi Datalog-szabályra:

1. $P(x) \leftarrow R(x) \text{ AND NOT } Q(x)$
2. $Q(x) \leftarrow R(x) \text{ AND NOT } P(x)$

Informálisan a két szabály azt fejezi ki, hogy az x akár P -ben, akár Q -ban lehet, de mindenkorban nem. Figyeljük meg, hogy a P és a Q is kölcsönösen rekurzív.

¹ Figyeljük meg, hogy ha az 5. sorban az egyik `Eljut` használata helyett a 10.11. ábrán szereplő `Járatok`-at használnánk, akkor lineáris rekurziót kapnánk. A nem-lineáris rekurzió általában – nem feltétlenül minden esetben – átírható a neki megfelelő lineáris változatra.

Ha abból indulunk ki, hogy P és Q is üres, és alkalmazzuk egyszer a szabályokat, akkor azt kapjuk, hogy $P = \{(0)\}$ és $Q = \{(0)\}$, azaz a (0) mindenkét IDB-relációba bekerül. A következőben, amikor a P és Q értékére alkalmazzuk a szabályokat, akkor azt kapjuk, hogy minden a kettő üres. Ez a ciklus folytatódik, ameddig akarjuk, de sosem fog a megoldáshoz konvergálni.

Valójában két „megoldás” van a Datalog-szabályokhoz:

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

Nincs okunk, hogy bármelyiket is előnyben részesítsük a másikkal szemben, és az előbb bemutatott egyszerű lépéssorozat, mint a rekurzív relációk egyik kiszámítási módszere, nem konvergál egyikhez sem. Azaz arra az egyszerű kérdésre sem tudunk választ adni, hogy: „Teljesül a $P(0)$? ”

```

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;

```

10.12. ábra. A nem monoton viselkedésű lekérdezések nem megengedettek SQL-ben

Ez a probléma nem csak a Datalogra áll fenn. A példánk két Datalog-szabálya kifejezhető rekurzív SQL-ben. A 10.12. ábra ennek egy módját szemlélteti. Ez az SQL-utasítás nem felel meg a szabvány előírásainak, ezért egyetlen ABKR sem fogja végrehajtani. □

A 10.9. példával az a probléma, hogy a 10.12. ábrán szereplő P és Q definíciója nem monoton. Vegyük például a P definícióját a 2. és 5. közötti sorokból. P a Q -tól függ, és ezzel kölcsönös rekurzív lesz ugyan, de a Q -ba történő beszúrás törlést eredményezhet R -re. Figyeljük meg, hogy ha $R = \{(0)\}$, a Q pedig üres, akkor $P = \{(0)\}$ lesz. Viszont ha a (0) -t Q -hoz adjuk, akkor kitöröljük a (0) -t P -ből. Így P definíciója nem monoton Q -ban, és a 10.12. ábrán lévő SQL-kód megsérti a szabványt.

10.10. példa. Az összesítések szintén a monotonitás elvesztéséhez vezethetnek. Tételezzük fel, hogy a következő két unáris (egy attribútummal rendelkező) P és Q relációt definiáljuk:

1. P a Q és egy R EDB-reláció összesítése.
2. Q -nak egy sora van, amely a P elemeinek összesítését tartalmazza.

Ezeket a feltételeket kifejezhetjük egy WITH utasítással, azonban ez az utasítás megszegi az SQL monotonitásra vonatkozó feltételeit. A 10.13. ábra lekérdezése P értékét keresi.

```

1)   WITH
2)       RECURSIVE P(x) AS
3)           (SELECT * FROM R)
4)           UNION
5)               (SELECT * FROM Q),
6)       RECURSIVE Q(x) AS
7)           SELECT SUM(x) FROM P
8)   SELECT * FROM P;

```

10.13. ábra. Nem rétegzett lekérdezés összesítéssel, amely szabálytalan az SQL-ben

Tételezzük fel, hogy eredetileg R a (12) és a (34) sorokból áll, kezdetben P és Q üres. A 10.14. ábra foglalja össze az első hat lépésben kiszámított értékeket. Ne feledjük, hogy a relációk új értékeit az összes reláció előző lépéshoz közelítően értékei alapján számítjuk ki. Így az első lépés után P egyezni fog az R -rel, míg $Q = \{\text{NULL}\}$ lesz, mivel a kiszámításához P üres értékét használtuk a 7. sorból.

A második lépésben a 3–5. sorok közötti rész eredménye az

$$R \cup \{\text{NULL}\} = \{(12), (34), \text{NULL}\}$$

halmaz lesz, tehát ez lesz P új értéke, a régi érték $\{(12), (34)\}$ volt. Így Q új értéke a $\{(46)\}$ halmaz lesz, mivel $12 + 34 = 46$.

A harmadik lépésben $P = \{(12), (34), (46)\}$ -t kapjuk a 2–5. sorok alapján. P régi értékét használjuk ($P = \{(12), (34)\}$) és a 6–7. sorok alapján $Q = \{(46)\}$ lesz újra.

Lépés	P	Q
1.	$\{(12), (34)\}$	$\{\text{NULL}\}$
2.	$\{(12), (34), \text{NULL}\}$	$\{(46)\}$
3.	$\{(12), (34), (46)\}$	$\{(46)\}$
4.	$\{(12), (34), (46)\}$	$\{(92)\}$
5.	$\{(12), (34), (92)\}$	$\{(92)\}$
6.	$\{(12), (34), (92)\}$	$\{(138)\}$

10.14. ábra. Iteratív kiszámítás nem monoton összesítés esetén

A negyedik lépésben P értéke ugyanaz, de Q értéke megváltozik $\{(92)\}$ -re, mert $12 + 34 + 46 = 92$. Figyeljük meg, hogy Q elvesztette a (46) sort, és egy új sort kapott helyette. Tehát a (46) sor beszűrása P -be egy sor kitörlését eredményezte Q -ból (véletlenül ugyanannak a sornak a kitörlését). Ez ellenkezik a monotonitási szabálytalán, amit az SQL előír, tehát az 10.13. ábra lekérdezése szabálytalan. Általában a $2i$ -edik lépésben P tartalmazza a (12), (34), illetve a $(46i - 46)$ sorokat, míg Q csak a $(46i)$ sort tartalmazza. \square

10.2.3. Feladatok

10.2.1. feladat. A 10.8. példa relációja:

Járatok(légitársaság, honnan, hova, indulás, érkezés)

tartalmaz az érkezési és indulási időről is információt, amit eddig nem vettünk figyelembe. Tegyük fel, hogy nemcsak az érdekel minket, hogy el tudunk-e jutni az egyik városból a másikba, hanem az is, hogy utazásunk során az átszállások is ésszerűek legyenek. Azaz, ha egynél több repülőgépet használunk, akkor az összes járatnak legalább egy órával a rá következő járat indulása előtt meg kell érkeznie. Azt is feltehetjük, hogy nincs egy napnál hosszabb utazás, így nem kell aggódunk amiatt, hogy egy éjfél körüli érkezés esetén csak korán reggel lesz tovább csatlakozás.

- a) Írunk erre rekurziót Datalogban.
- b) Írunk erre rekurziót SQL-ben.

! 10.2.2. feladat. A 10.8. példában a honnan (az angol eredetiben `frm`) az egyik attribútum neve. Miért nem használjuk inkább a `from` nevet?

10.2.3. feladat. Adott a következő reláció:

Folytatások(film, folytatás)

mely egy film azonnali folytatásait adja meg. Definiálni akarjuk még a Sorozat rekurzív relációt is, amelynek elemei olyan (x, y) sorok, hogy y vagy x azonnali folytatása, vagy egy folytatás folytatása stb.

- a) Adjuk meg Sorozat definicióját rekurzív Datalog-szabályokkal.
- b) Adjuk meg Sorozat definicióját egy rekurzív SQL-kifejezés segítségével.
- c) Adjunk meg egy olyan rekurzív SQL-lekérdezést, amely azokat az (x, y) filmpárokat eredményezi, melyekre y egy nem direkt folytatása x -nek.
- d) Adjunk meg egy olyan rekurzív SQL-lekérdezést, amely azokat az (x, y) filmpárokat eredményezi, amelyekre y folytatása x -nek, de nem direkt folytatása és nem egy direkt folytatás folytatása.

- ! e) Adjunk meg egy olyan rekurzív SQL-lekérdezést, amely azokat az x filmet eredményezi, amelyeknek legalább két folytatása volt. Mindkét folytatás lehet direkt, vagy az egyik direkt, a másik pedig annak direkt folytatása.
- ! f) Adjunk meg egy rekurzív SQL-lekérdezést, amely azokat az (x, y) filmpárokat eredményezi, melyekre y folytatása x -nek, és y -nak legfeljebb egy folytatása van.

10.2.4. feladat. Adott a következő reláció:

Kapcsolatok(osztály, kosztály, mult)

amely leírja azt, hogy egy ODL-osztály hogyan viszonyul más osztályokhoz. Általában a relácionál van egy (c, d, m) sora, ha létezik egy kapcsolat a c osztályból a d osztályba. Ez a kapcsolat többértékű, ha $m = \text{'multi'}$, és egyértékű, ha $m = \text{'single'}$. A Kapcsolatok relációt tekinthetjük úgy, mint egy gráfot, melynek a csúcsai osztályok, és a c osztálytól a d osztályig akkor és csak akkor létezik egy m címkeljű él, ha (c, d, m) egy él a Kapcsolatok-ban. Adjunk meg egy rekurzív SQL-lekérdezést, amely:

- a) Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c -től d -ig.
- b) azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c -től d -ig, amely végig egyértékű kapcsolatokból áll.
- ! c) Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c -től d -ig, amelyben legalább az egyik él többértékű kapcsolatot jelképez.
- d) Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c -től d -ig, de nem létezik közöttük olyan útvonal, amely végig egyértékű kapcsolatokból áll.
- ! e) Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c -től d -ig, amely váltakozóan egyértékű és többértékű kapcsolatokat jelképező élekből áll.
- f) Azokat a (c, d) párokat adja meg, amelyekre létezik egy útvonal c -től d -ig és d -től c -ig, amely végig egyértékű kapcsolatokból áll.

10.3. Az objektumrelációs modell

A relációs modell és az objektumorientált modell (amelyet az ODL jellemz) két fontos választási lehetőség a sok közül, amelyekre egy ABKR épülhet. Hosszú időn keresztül a relációs modell volt a meghatározó a forgalomban lévő ABKR-ek világában. Az objektumorientált ABKR-ek megróbálkoztak betörni a piacra az 1990-es évek folyamán, de nem nyertek jelentős piaci részesedést a relációs