ADATBÁZISOK

A tervezéstől az alkalmazásfejlesztésig

Dr. Balázs Péter, egyetemi docens
Dr. Németh Gábor, adjunktus

Szegedi Tudományegyetem
Természettudományi és Informatikai Kar
Informatikai Intézet
2019

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

Előszó

Jelen jegyzetet a Szegedi Tudományegyetem Informatikai Intézete által gondozott szakok hallgatóinak készítettük az alapozó Adatbázisok című tárgy segédleteként. Összeállítása során a nemzetközileg széleskörűen használt [9,26] könyveket tekintettük alapnak, valamint jelentősen támaszkodtunk az egytemünkön korábban Dr. Katona Endre által készített jegyzetre is [12].

Az első részében, mely 7 fejezetet ölel át, az adatbázistervezés alapjait mutatjuk be. A második rész 10 fejezetből áll és az SQL nyelv rejtelmeibe vezet be, míg a harmadik részben 7 fejezeten keresztül ismertetjük az adatbázisokon alapuló alkalmazások fejlesztésének lépéseit. A jegyzet végén egy külön fejezet a *Tanulási Eredmény Alapú Módszertan* szempontrendszere alapján mutatja be a kurzust.

A jegyzet megírása során törekedtünk arra, hogy az Olvasótól minél kevesebb matematikai, informatikai tudást követeljünk meg, de alapvető ismeretek feltételezésétől nem tekinthettünk el. Főként a diszkrét matematika és a JAVA programozási nyelv alapjaira támaszkodunk.

Itt szeretnénk köszönetet mondani Dr. Bodnár Péter és Dr. Kardos Péter adjunktusoknak a kézirat lektorlásáért, valamint Dr. Griechisch Erika tanársegédnek további értékes észrevételeiért. Segítségükkel – reményeink szerint – sikerült egy olyan jegyzetet készítenünk, mely minden az adatbázisok tervezése és megvalósítása iránt érdeklődő olvasó számára hasznos lehet.

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával. Projekt azonosító: EFOP-3.4.3-16-2016-00014.

Dr. Balázs Péter és Dr. Németh Gábor
Szeged, 2019. október

Tartalomjegyzék

I. rész

Adatmodellezés és adatbázisok tervezése

1. fejezet

Adatok, adatbázisok, adatmodellek

Jelen fejezetben az Olvasó megismeri az adatbázisokkal kapcsolatos alapfogalmakat, az adatok különböző strukturáltsági szintjeit és a fontosabb adatmodelleket.

1.1. Az adatok típusa, strukturáltsági szintjei

A számítógépes adatok strukturáltsági szintje eltérő lehet.

Strukturálatlan: Az ilyen adatok egyszerű bájtsorozatban tároltak, bennük a keresés (értelmezhető adatra) nem lehetséges. Ilyenek a digitális médiafájlok (kép, hang, videó).

Egyszerű szöveg: Ezekben a fájlokban (például `txt`) már keresni lehet egy-egy konkrét adatra (szóra, számra, dátumra, stb.).

Formázott szöveg: Megjelennek az adatok közti hierarchiaszintek is (fejezet, alfejezet, bekezdés, stb., mint például a `doc` fájlokban).

Hipertext: A formázott szövegen túlmenően belső és külső hivatkozások is jelzik az adatok közti kapcsolatokat. Ilyenek például a `html` és `xml` fájlok.

Táblázat: A táblázatos szervezettségű adatoknál már komplex rendezéseket, lekérdezésket is megvalósíthatunk. Példaként az `xls` fájlokat említhetjük.

Adatbázis: A legmagasabb szervezettségi szinten állnak az adatbázisok, ahol az adatok már bonyolult kapcsolatrendszert alkothatnak.

1.1. ábra. Adatbázis-alkalmazások szintjei.

1.2. Adatbáziskezelő-rendszerek

Az *adatbázis* adott formátum és rendszer szerint tárolt adatok együttesét jelenti. Ennek alapvető adategységét *sornak* vagy *rekordnak* nevezzük. Az adatbázisokat kezelő szoftvereket *adatbáziskezelő-rendszereknek* hívjuk, melyeket gyakran az angol elnevezésük alapján DBMS-nek (Database Management System) rövidítünk. Egy adatbáziskezelő-rendszer fő feladatai az alábbiak:

- az adatstruktúrák definiálása,
- az adatok aktualizálása (adatfelvitel, törlés, módosítás) és lekérdezése,
- nagy mennyiségű adat hosszú idejű, biztonságos tárolása,
- több felhasználó egyidejű kiszolgálása, jogosultságok szabályozása,
- több feladat egyidejű végrehajtása, tranzakciók kezelése.

Az adatbázis-alkalmazásoknál maga a DBMS a logikai és a fizikai adatstruktúra szintje között helyezkedik el (lásd 1.1. ábra). Példaként három DBMS-t említünk. A Microsoft Access egy könnyen kezelhető, grafikus felületű adatbáziskezelő, mely kisebb alkalmazások elkészítéséhez lehet alkalmas. A MySQL egy nyílt-forráskódú adatbázis szerver, inkább közepes méretű és főként webes alkalmazások létrehozásához javasolt. Végül az Oracle egy nagy teljesítményű, sokfelhasználós rendszer, ami nagyméretű adatbázisok létrehozására és menedzselésére is alkalmas.

1.3. Adatmodellek

Az adatok rendszerezését megkönnyítendő az évek során számos adatmodell alakult ki, melyek közül itt most csak néhányat emelünk ki.

Hierarchikus modell: Az 1960-as évek elején kialakított legelső adatmodell. Az adatok fastruktúrában rendezettek. A fában egy pontnak (szülőnek) több gyereke is lehet, de egy gyerekhez csak egy szülő tartozhat. Az ilyen modellben az adatkeresés viszonylag egyszerű, fabejáró algoritmusokkal történik. Az adatok törlése, módosítása vagy új adat beszúrása azonban általában nem végezhető el gyorsan, hatékonyan. Bár a modell a később kifejlesztett más megközelítések miatt egy időre háttérbe szorult, az Extended Markup Language (XML) megjelenésével az 1990-es évek végén újból előkerült.

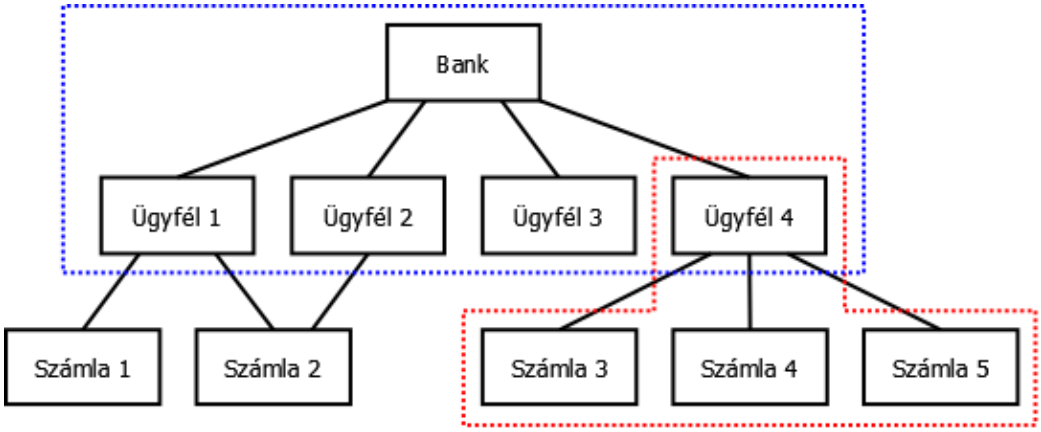
Hálós modell: Az 1970-es évek elején megjelent modell a hierarchikus modell továbbfejlesztésének tekinthető. A rekordok pointerekkel kapcsolódnak egymáshoz. Egy rekord akár több másik rekordhoz is kapcsolódhat. Az adatok módosítása, törlése, beszúrása azonban továbbra is körülményes lehet. A modell alapvető egysége a *set*, ami egy szülő és annak összes gyermeke által alkotott csoportot jelent, melyen a pointerek körbefutnak (lásd 1.2. ábra). A modell ma már nem használatos.

Relációs modell: Szintén az 1970-es évek elején jelent meg. Mind az adatokat mind a köztük lévő kapcsolatokat kétdimenziós táblákban tárolja (lásd 1.3. ábra). A táblákban az azonos sorban álló egyedek alkotnak egy relációt. Az erre a modellre épülő adatbáziskezelőket RDBMS-nek (Relational DBMS) nevezzük. Lekérdező nyelvük az SQL (Structured Query Language). Napjainkban is széles körben használt modell.

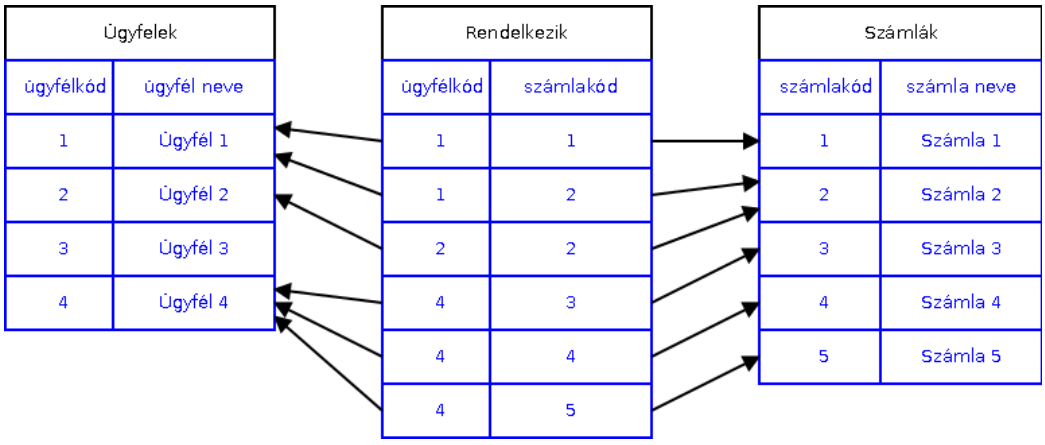
Objektumorientált modell: Az 1980-as évek végén, a 90-es évek elején megjelent modell az objektumorientált programozás eszköztárával reprezentálja az adatokat és kapcsolataikat. Az adatok definiálásához az ODL-t (Object Defintion Language), a lekérdezéshez az OQL-t (Object Query Language) használja, adatbáziskezelő rendszerei az OODBMS-ek (Object Oriented DBMS). A modell megjelenése óta folyamatosan fejlődik, de jelentősége a relációs modellétől elmarad.

Objektum-relációs modell: A relációs modellt bővíti objektumorientált lehetőségekkel, ily módon egyesítve a két modell előnyeit. A tisztán objektumorientált rendszerek helyett a gyakorlatban inkább ezt a kevert modell használó ORDBMS-ek (Object Relational DBMS) terjedtek el.

Jelen jegyzetben a relációs adatkezeléssel fogunk megismerkedni.



1.2. ábra. Egy banki nyilvántartás hálós modellje. A 2-es számla fölött az 1-es és a 2-es ügyfél is rendelkezik. A bank és az ügyfelek setjét kék, a 4-es ügyfél és annak számláit alkotó setet piros szaggatott vonal jelzi.



1.3. ábra. Az 1.2. ábra banki nyilvántartása relációs modellben.

Kérdések és feladatok

1. Adja meg hierarchikus modellben egy olyan cég adatrekordjait, ahol egy főosztály van, három alosztály és az egyik osztályon két dolgozó dolgozik, a másik kettőn pedig három! (Minden dolgozó csak egy osztályon dolgozik.)

2. Írja le hálós modellben a 2. feladat adatviszonyait!

3. Adjon meg a hálós modellben egy olyan adatbázist, mely egy nyelviskola 5 hallgatóját tartalmazza, akik közül 3 angolra és 4 németre jár! Mi okozza a nehézséget, ha ezt az adatbázist hierarchikus modellben szeretnénk megadni?

4. Az 1.3. ábra alapján adja meg a 3. feladat adatait relációs modellben!

2. fejezet

Az egyed-kapcsolat modell

Az egyed-kapcsolat modell (röviden E-K modell) konkrét adatmodelltől függetlenül, szemléletesen adja meg az adatbázis szerkezetét. Ebben a fejezetben ismertetjük az E-K modellezés fogalmait és jelölésrendszerét egy konkrét példán keresztül.

2.1. Egy konkrét probléma

Szeretnénk létrehozni egy internetes **Fórum** adatbázist az alábbiak szerint.

- A fórumba csak bejeletkezett felhasználók írhatnak üzeneteket és olvashatják azokat. A felhasználókat felhasználónévvel azonosítjuk, amelyhez egy jelszó is tartozik. Ezen kívül tárolni szeretnénk a felhasználó email címét, vezeték- és keresztnevét, valamint az utolsó bejelentkezésének időpontját is.

- Az üzenetek hírfolyamokba vannak szervezve, minden hírfolyamhoz tartoznak kulcsszavak is. A hírfolyamokat egy egyértelmű azonosítóval szeretnénk ellátni, ezen kívül még a hírfolyam neve tárolandó.

- Egy üzenet esetén tudnunk kell, hogy annak mi a tartalma, mikor és ki hozta létre, valamint hogy melyik hírfolyamba tartozik.

- Végezetül tudnunk kell, hogy mely felhasználók mely hírfolyamokat követik.

Látható, hogy sokféle, különböző jellegű adatot kell majd tárolnunk. Az E-K modellezést fogjuk segítségül hívni, hogy ezeket az adatokat logikusan rendszerezni tudjuk és megtaláljuk a közöttük lévő kapcsolatokat.

2.2. Alapfogalmak és jelölések

Egyednek vagy *entitásnak* hívjuk a valós világ egy objektumát, melyről az adatbázisban információt szeretnénk tárolni. Megkülönböztetjük a *egyedtípust* és az *egyedpéldányt*. Előbbi általánosságban jelent egyfajta valós objektumot, míg utóbbi, annak egy konkrét megvalósulását jelenti. A létrehozandó **Fórum** adatbázis esetén például a felhasználó egy egyedtípus, míg egy meghatározott felhasználó (például: a szerző, Balázs Péter) egy konkrét egyedpéldányt jelent.

Tulajdonságnak vagy *attribútumnak* hívjuk az egyed egy jellemzőjét. Itt is megkülönböztetjük a *tulajdonságtípust* (például általánosságban a felhasználó jelszava) és a *tulajdonságpéldányt* (például egy konkrét jelszó, mint „X23gF4hU"). Az egyed attribútumainak azt a legszűkebb részhalmazát, mely az egyedet egyértelműen meghatározza, *kulcsnak* nevezzük. Egy felhasználót egyértelműen azonosít például a felhasználóneve, így ez jelen esetben tekinthető az adott egyed(típus) kulcsának.

Az egyedek között *kapcsolatok* alakulhatnak ki, melyeket szintén tárolni szeretnénk az adatbázisban. A fentiekhez hasonlóan megkülönböztetjük a *kapcsolattípust* és a *kapcsolatpéldányt*. Például az, hogy általánosságban egy felhasználó létrehoz egy üzenetet valamely hírfolyamra, kapcsolattípusként értendő (mely a Felhasználó és az Üzenet egyedtípusokat hozza kapcsolatba), míg az, hogy Balázs Péter a 2331. sorszámú üzenetet hozza létre, egy konkrét kapcsolatpéldányt jelent. A kapcsolatoknak ugyanúgy lehetnek tulajdonságaik, mint az egyedeknek.

Azt a modellt, amely az adatbázisban tárolandó adatokat egyedekkel, tulajdonságokkal és kapcsolatokkal írja le, *egyed-kapcsolat modellnek* (röviden *E-K modellnek*), az ezt ábrázoló diagramot pedig *egyed-kapcsolat diagramnak* (röviden *E-K diagramnak*) nevezzük. Az E-K diagram az alábbi jelöléseket használja:

- az egyedeket téglalappal,
- a tulajdonságokat ellipszissel,
- a kulcsot aláhúzással,
- a kapcsolatokat rombuszokkal ábrázolja.

Vizsgáljuk meg a **Fórum** példánkat és próbáljuk meghatározni először az egyedeket. Egyrészt vannak felhasználók, akik üzeneteket hoznak létre (Felhasználó és Üzenet egyed), továbbá az üzenetek hírfolyamok részét képezik,

így célszerű egy Hírfolyam egyedet is létrehozni. Következő lépésben vizsgáljuk meg, hogy ezen egyedekről milyen tulajdonságokat kell eltárolnunk. A következő attribútumokat határozhatjuk meg.

Felhasználó: név, felhasználónév, jelszó, email cím, utolsó belépés időpontja. Ezek közül a felhasználónév egyértelműen azonosít, tehát kulcs. Általában a fórum alkalmazások nem engedik meg, hogy ugyanazzal az email címmel hozzunk létre több felhasználót, így választhatnánk az email attribútumot is kulcsnak. Ennek a lehetőségnek majd a későbbiekben lesz jelentősége. Egyelőre a felhasználónevet fogjuk az azonosításra használni.

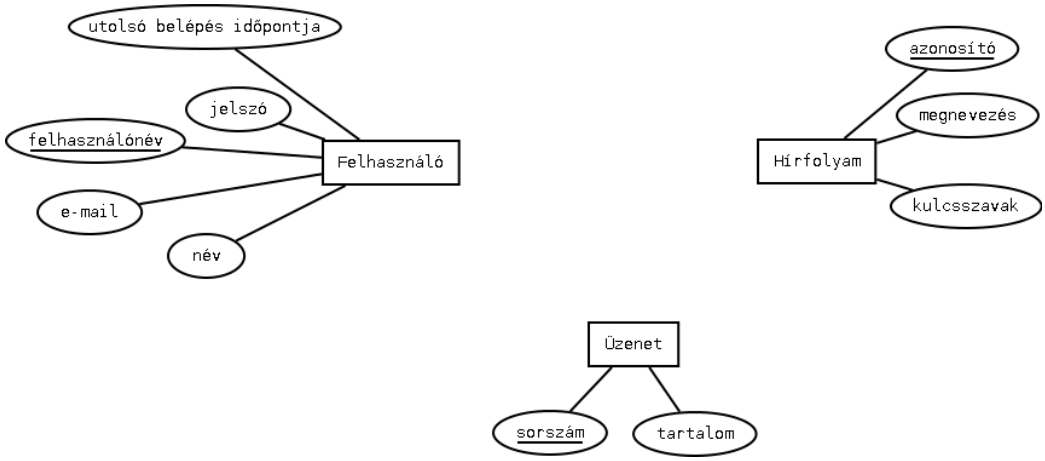
Üzenet: tartalom. Mivel a tartalom nem azonosítja egyértelműen az üzenetet (két üzenet lehet ugyanolyan tartalmú), ezért felveszünk minden üzenethez egy mesterséges egyedi azonosítót is, ami így már kulcs lesz.

Hírfolyam: megnevezés, kulcsszavak. Itt is előfordulhat, hogy két ugyanolyan elnevezésű és ugyanolyan kulcsszavakkal megadott hírfolyam is van, így ehhez az egyedhez is mesterséges egyedi azonosítot rendelünk hozzá.

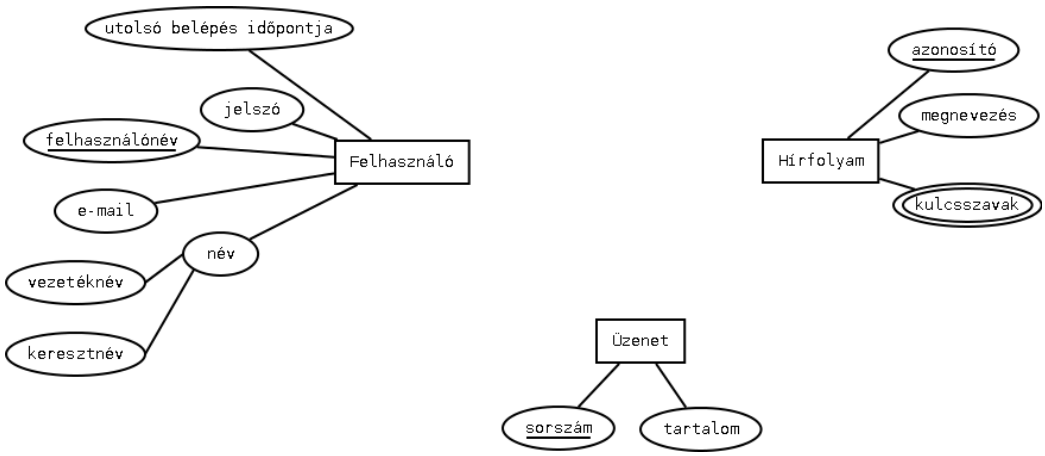
Az eddig összegyűjtött információinkból megrajzolt E-K diagram jelenlegi állását a 2.1. ábra mutatja. A felhasználó nevét tárolhatjuk egy sztringben is, de ha a későbbiek szempontjából célszerű, akkor modellezhető, hogy a vezetéknév és a keresztnév külön-külön sztringben (két attribútumként) kerüljön majd tárolásra. Az ilyen attribútumokat, amelyek maguk is attribútumokkal rendelkeznek, *összetett attribútumoknak* hívjuk. Az összetett attribútum általában egy struktúra, aminek adattagjai külön-külön elemi típusú értékekre képződnek le. Az E-K diagramon ezt úgy jelöljük, hogy a struktúrát alkotó adattagokat újabb ellipszissel kötjük az összetett attribútumhoz. Hasonlóan, ha jelezni kívánjuk, hogy egy attribútum halmaz vagy lista adattípusra képződne le (előbbinél nem számít a sorrend, az utóbbinál viszont igen), akkor ezt a diagramon kettős ellipszissel jelezhetjük. Az ilyen attribútumokat *többértékű attribútumoknak* hívjuk. Ilyen attribútum példánkban a kulcsszavakat tartalmazó, azokat ugyanis nem egy sztringben vesszővel elválasztva, hanem külön-külön szeretnénk tárolni az adatbázisban. A 2.2. ábra az elmondottakat szemlélteti.

Nézzük most meg, hogy az egyedek hogyan kapcsolódnak egymáshoz.

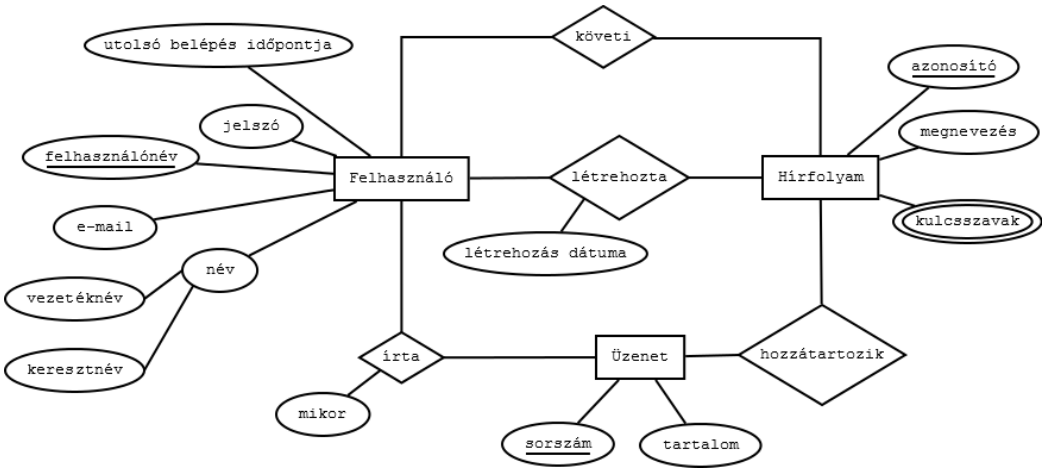
Hozzátartozik: Mivel az üzenetek hírfolyamokba vannak szervezve, így minden esetben tudnunk kell, hogy melyik üzenet melyik hírfolyamhoz tartozik. Ez a kapcsolat ezt valósítja meg.



2.1. ábra. A **Fórum** E-K modellje az egyedek és tulajdonságaik felírása után.



2.2. ábra. A **Fórum** E-K modellje az összetett és többértékű attribútumokat is jelezve.



2.3. ábra. A **Fórum** E-K modellje a kapcsolatok és tulajdonságaik felírása után.

Írta: Tudnunk kell, hogy melyik üzenetet melyik felhasználó írta, ezért ezt a két egyedet is kapcsolatba hozzuk egymással. Ennek a kapcsolatnak tulajdonsága is van, mégpedig az, hogy mikor keletkezett az adott üzenet.

Létrehozta: A hírfolyamokat felhasználók hozzák létre, így ezen egyedek között is kapcsolat alalkul ki. A kapcsolat tulajdonsága emellett a hírfolyam létrehozásának dátuma.

Követi: A felhasználók hírfolyamokat követnek, ezért ezen két egyed is kapcsolódik egymással.

A továbbgondolt E-K diagramot a 2.3. ábra szemlélteti.

A kapcsolatok további vizsgálatra szorulnak. Megkülönböztetünk *két egyed közötti (bináris)* és kettőnél több egyed közötti kapcsolatokat. Ez utóbbi típus ritkábban jelenik meg (példánkban sincs ilyen), és visszavezethető bináris kapcsolatokra. Ezért a továbbiakban csak a bináris kapcsolatokat vizsgáljuk részletesebben, melyek három típusba sorolhatók (E_1-gyel és E_2-vel jelölve a kapcsolódó egyedeket):

Egy-az-egyhez (1:1) kapcsolat esetén egy E_1 egyedpéldányhoz legfeljebb egy E_2 egyedpéldány tartozhat, és viszont, egy E_2 egyedpéldányhoz is legfeljebb egy E_1 egyedpéldány tartozhat. Az E-K diagramon ilyenkor nyilat teszünk a kapcsolatot ábrázoló vonal E_1 és E_2 felöli végére is (vagy egy 1-est írunk a vonal mindkét vége fölé).

Egy-a-többhöz (1:N) kapcsolat esetén egy E_1 egyedpéldányhoz több E_2 egyedpéldány, de egy E_2 egyedpéldányhoz csak egy E_1 egyedpéldány tartozhat. Az E-K diagramon ilyenkor a kapcsolatot ábrázoló vonal E_1 felöli végére teszünk csak nyilat (vagy 1-est írunk fölé, míg a vonal másik vége fölé egy N betűt írunk).

Több-a-többhöz (N:M) kapcsolat esetén egy E_1 egyedpéldányhoz több E_2 egyedpéldány és egy E_2 egyedpéldányhoz több E_1 egyedpéldány tartozhat. Az E-K diagramon ilyenkor a kapcsolatot ábrázoló vonalra nem teszünk nyilat (vagy az egyik vége fölé egy N betűt, a másik vége fölé pedig egy M betűt írunk).

Vizsgáljuk meg most a példánk kapcsolatait a fentiek alapján.

Hozzátartozik: Egy hírfolyamhoz több üzenet is tartozhat, azonban egy konkrét üzenet mindig egyértelműen csak egy hírfolyamhoz tartozik. Ez tehát egy 1:N kapcsolat, melyben a Hírfolyam az 1-oldali egyed.

Írta: Egy felhasználó több üzenetet is írhat, de egy konkrét üzenetet egy meghatározott felhasználó ír, ezért ez is 1:N kapcsolat, ahol a Felhasználó az 1-oldali egyed.

Létrehozta: Egy felhasználó több hírfolyamot is létrehozhat, de egy konkrét hírfolyamot mindig egy meghatározott felhasználó hozhat létre. Ez is 1:N kapcsolat tehát, és a Felhasználó az 1-oldali egyed.

Követi: Egy felhasználó több hírfolyamot is követhet és egy hírfolyamot több felhasználó is követhet, tehát ez N:M kapcsolat.

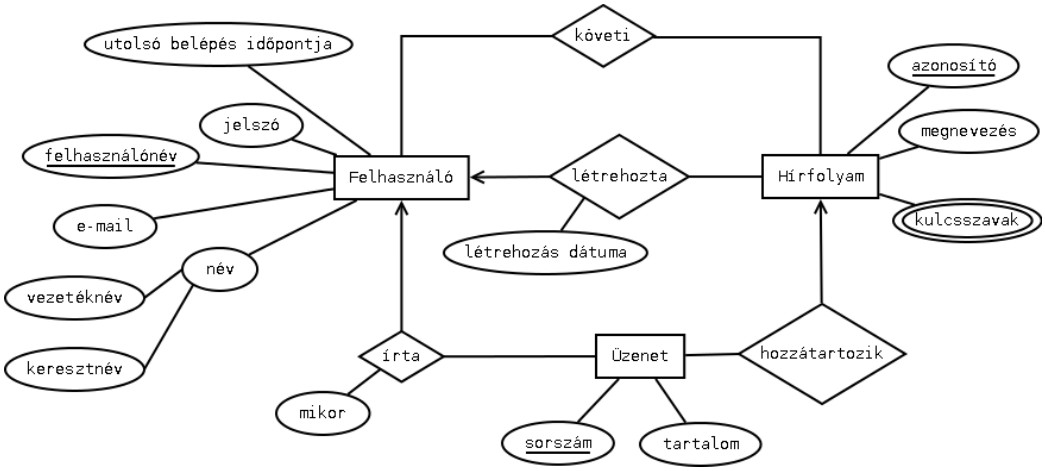
Ennek ismeretében módosítjuk az E-K-diagramot (lásd 2.4. ábra).

A jelölés tovább finomítható a következők szerint. Azt mondjuk, hogy egy egyedtípus *teljesen részt vesz* egy kapcsolatban, ha minden egyedpéldány kapcsolatban áll valamely másik egyeddel. Ebben az esetben kettős vonalat húzunk az egyed és a kapcsolat közé.

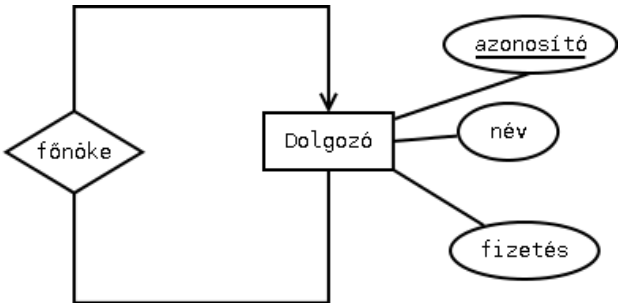
Előfordulhat továbbá, hogy egy egyedtípus önmagával áll kapcsolatban.

2.2.1. példa

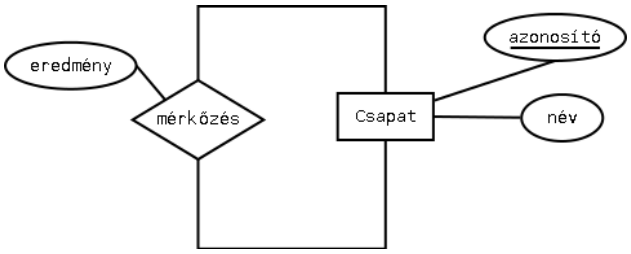
Egy vállalat dolgozóit és főnökeiket szeretnénk nyilvántartani. Ha feltesszük, hogy minden dolgozónak csak egy közvetlen felettese van, akkor a 2.5. ábrán látható 1:N típusú kapcsolathoz jutunk. A főnök maga is dolgozó, így a kapcsolat mindkét oldalán ugyanaz az egyedtípus áll. A legfőbb vezetőnek már nincs felettese. Az ő esetében az adatbázisban NULL értékkel jelezhetjük, hogy ő áll a hierarchia csúcsán.



2.4. ábra. A **Fórum** E-K modellje a kapcsolatok típusainak feltüntetése után.



2.5. ábra. Példa 1:N típusú kapcsolatra, ahol mindkét oldalon ugyanaz az egyedtípus áll.



2.6. ábra. Példa N:M típusú kapcsolatra, ahol mindkét oldalon ugyanaz az egyedtípus áll.

2.2.2. példa

Tekintsünk egy olyan adatbázist, mely egy sportverseny lejátszott mérkőzéseit tartalmazza. Ebben az esetben a Csapat egyed önmagával kerül kapcsolatba, hiszen egy mérkőzést két csapat játszik. Ezt az N:M kapcsolatot a 2.6. ábra szemlélteti.

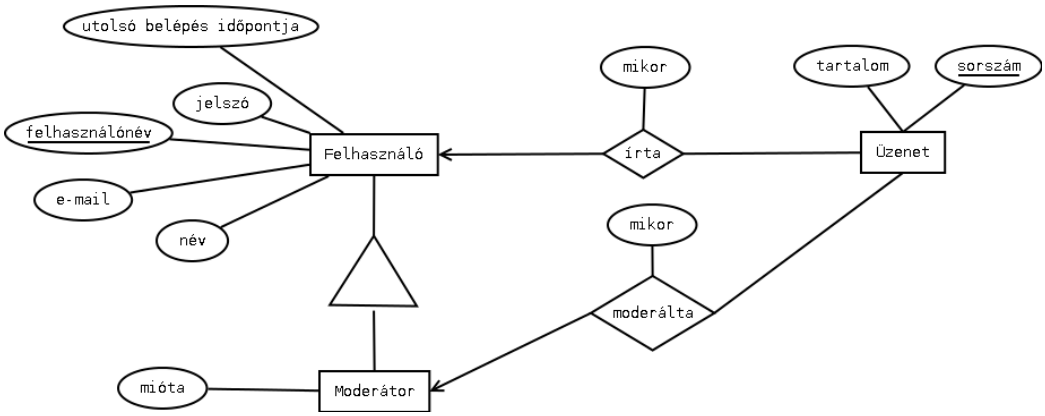
Hangsúlyozzuk, hogy mindkét példában csak az egyedtípus az, ami megegyezik a kapcsolat két oldalán, az egyedpéldányok értelemszerűen nem egyeznek.

Adódhat olyan eset is, hogy egy egyednek bizonyos altípusait külön szeretnénk feltüntetni a modellben. Az E-K diagram erre is nyújt lehetőséget. A főtípus és az altípus viszonyát *specializáló kapcsolattal* adhatjuk meg, melyet a diagramon csúcsával a főtípus felé mutató háromszöggel jelölünk. Az altípus örökli a főtípus minden attribútumát és kapcsolatát, emellett további attribútumokkal és kapcsolatokal is rendelkezhet.

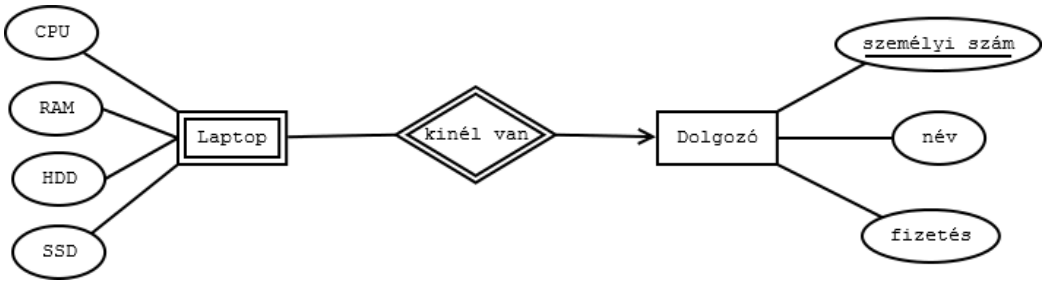
2.2.3. példa

A 2.7. ábra a **Fórum** alkalmazás egy olyan továbbfejlesztett változatát modellezi (annak csak egy részletét kiemelve), ahol megjelennek speciális felhasználók, a moderátorok, akiknek az egyszerű felhasználón kívül még ahhoz is joguk van, hogy egy bejegyzést moderáljanak. Ezt egy újabb kapcsolat formájában tüntetjük fel. A moderátorokról az általános felhasználót jellemző öt tulajdonság mellett egy továbbit is nyilvántartunk, mégpedig azt, hogy mióta rendelkezik a speciális, moderáláshoz való joggal.

Bár igyekszünk az egyedekről olyan tulajdonságokat tárolni, amelyek egyértelműen meghatározzák az egyedpéldányokat, előfordulhat hogy ez mégsem áll fent. Azokat az egyedeket, melyeket attribútumai nem határoznak meg egyértelműen, *gyenge entitásoknak* (gyenge egyedeknek) hívjuk és az E-K di-



2.7. ábra. Példa specializáló kapcsolatra.



2.8. ábra. Példa gyenge egyedre és meghatározó kapcsolatra.

agramon kettős téglalappal jelöljük. Az ilyen egyedeket is egyértelműen meg kell tudnunk határozni, mely az egyed valamely kapcsolata segítségével valósítható meg. Az ilyen kapcsolatokat *meghatározó kapcsolatnak* nevezzük és kettős rombusszal jelöljük.

2.2.4. példa

Tekintsük a 2.8. ábrát, mely azt tünteti fel, hogy egy adott munkahelyen melyik dolgozó milyen konfigurációjú laptopot használ. A laptopokhoz nem rendelünk azonosítót, így az azonos hardverösszetételű gépek nem különböztethetők meg egymástól. Mégis tudnunk kell, hogy mikor melyik számítógép példányról beszélünk, ami a gép tulajdonosának megnevezésével egyértelműsíthető. Ezért a laptop tulajdonosa felé mutató kapcsolat meghatározó kapcsolattá válik.

Kérdések és feladatok

1. Állapítsa meg, hogy az alábbi bináris kapcsolatok milyen típusúak! Indokolja, hogy miért!

   - Gépjárművek és tulajdonosaik.
   - Bankszámlák és tulajdonosaik.
   - Filmek és szereplőik.
   - Magyar állampolgárok és személyi igazolványaik.

2. Egy egyetem karokra, azon belül intézetekre tagolódik. Szeretnénk nyilvántartani, hogy melyik egyetem milyen karokból áll és azon belül milyen intézetekből. Az intézetek különböző kurzusokat hirdetnek meg, melyeknél rögzíteni szeretnénk, hogy hány kredit jár értük és hogy a jelenlegi szemeszterben hetente milyen időpontban tartják az adott kurzust. Szeretnénk továbbá azt is tárolni, hogy melyik hallgató éppen milyen kurzusokra jár. Milyen attribútumokat lát célszerűnek összegyűjteni a különböző egyedtípusokról? Rajzolja fel a problémához tartozó E-K diagramot úgy, hogy az ne tartalmazzon gyenge egyedet!

3. Mondjon példát olyan többértékű attribútumra, melyet halmaz, valamint olyat, melyet lista adatszerkezetre érdemes leképezni!

4. Adjon példát olyan estre, amikor egy egyed teljesen részt vesz egy kapcsolatban! Rajzolja fel hozzá az E-K diagramot!

3. fejezet

A relációs adatmodell

A relációs adatmodell mind az adatokat, mind a köztük lévő kapcsolatokat kétdimenziós (sorokból és oszlopokból álló) táblákban tárolja. Ebben a fejezetben ennek a modellnek az elméleti alapjait ismertetjük.

3.1. Attribútumok, relációsémák

A relációs adatmodellben *attribútumnak* egy névvel és értéktartománnyal megadott tulajdonságot nevezünk. A Z attribútum értéktartományát $dom(Z)$ jelöli, az angol „domain" (tartomány) szóból rövidítve. A relációs modellben az értéktartomány csak elemi típusú értékekből állhat (mint például numerikus értékek, karakterek vagy sztringek), az összetett típusok (például struktúra, lista, halmaz, stb.) nem megengedettek. A típus mellett gyakran megadjuk az ábrázolás hosszát is. Például a fórum üzenetek sorszám tulajdonságának értéktartománya a legfeljebb 10 jegyű egész számok halmaza lehet, míg a tartalom tulajdonság értéktartománya lehet például a legfeljebb 2000 karakter hosszú sztringek halmaza.

A *relációséma* (röviden séma) egy névvel ellátott attribútumhalmazt jelent. Ha $A = \{A_1, \ldots, A_n\}$ jelöli az attribútumhalmazt és a séma neve R, akkor a relációsémát $R(A_1, \ldots, A_n)$ vagy tömörebben $R(A)$ jelöli. Ha két séma (legyenek ezek R és S) azonos nevű attribútumot is tartalmaz (például az A_i attribútumot), akkor ezek megkülönböztethetők egymástól az $R.A_i$ és $S.A_i$ jelölés segítségével (vagyis a séma nevét is kiírjuk az attribútum neve elé).

A relációséma nem tárol adatot, csak egy tábla szerkezetének leírását adja meg. Az adatok relációkkal adhatók meg. Az $R(A_1, \ldots, A_n)$ séma feletti T *reláció* egy $T \subseteq dom(A_1) \times \cdots \times dom(A_n)$ halmazt jelent. Azaz egy reláció nem más, mint az attribútumok értéktartományainak direkt szorzatából

képzett halmaz egy részhalmaza. Ezért értelemszerűen T minden eleme egy olyan $(a_1, \ldots, a_n)$ érték n-es, ahol $a_i \in dom(A_i)$ $(i = 1, \ldots, n)$.

Egy ilyen reláció már valóban megjeleníthető adattábla formájában, ahol a táblázat oszlopai az $A_1, \ldots, A_n$ attribútumoknak, a táblázat sorai pedig a T halmaz egyes elemeinek felelnek meg. A tábla egy sorát *rekordnak* is nevezzük. Hangsúlyozzuk, hogy a reláció egy halmaz, így sorai szükségszerűen különböznek és nem definiált rajtuk semmilyen sorrendiség. A számítógépes megvalósítás ettől a modelltől azonban eltér, hiszen a sorok szükségszerűen egy adott fizikai sorrendben tárolódnak, továbbá az adatbáziskezelők általában megengednek ismétlődő sorokat is. Megjegyezzük továbbá, hogy a relációs modellnek megadható egy olyan általánosabb leírása is, mely az oszlopok sorrendjére sem tesz kikötést. Ennek tárgyalásától azonban itt eltekintünk.

A *relációs adatbázis* több, egymással kapcsolatban lévő adattáblát jelent, mely egy adott jelenség leírására alkalmas (lásd újra a 1.3. ábrát). Látható, hogy a különböző relációsémák azonos attribútumokat tartalmazhatnak, mely által a séma fölötti adattáblák sorai kapcsolatba kerülnek egymással.

3.2. Kulcsok

Ahhoz, hogy egy adattábla soraira egyértelműen hivatkozni tudjunk, szükségünk van attrbibútumok egy olyan halmazára, melyen az adattábla minden egyes sora más-más értéket vesz fel. Az adatábla olyan attribútumhalmazát, amely egyértelműen azonosítja a tábla sorait, *szuperkulcsnak* nevezzük. Formálisan, az $R(A)$ sémában a $K \subseteq A$ halmaz szuperkulcs, ha bármely R feletti T tábla bármely két sora különbözik K-n. Azaz, ha K szuperkulcs, akkor bármely R feletti T tábla és annak tetszőleges két $t_i, t_j \in T$ sora esetén, ha $t_i \neq t_j$, akkor $t_i(K) \neq t_j(K)$. Mivel az adattáblákban ismétlődő sorokat általában nem engedünk meg, így értelemszerűen $K = A$ mindig szuperkulcs.

Felmerül a kérdés, hogy ha $K = A$ szuperkulcs, akkor miért nem az A attribútumhalmazt választjuk mindig a sorok egyértelmű azonosítására. Ez azt jelentené, hogy minden esetben a teljes sort meg kell vizsgálnunk ahhoz, hogy eldöntsük, hogy az adott sorról beszélünk-e. Gyakorlati szempontból célszerűbb, ha minél kisebb olyan attribútumhalmazt keresünk, amely egyértelmű azonosításra alkalmas. Az A attribútumhalmaz K részhalmazát *kulcsnak* nevezzük, ha olyan szuperkulcs, ami halmaztartalmazásra nézve minimális, azaz egyetlen valódi részhalmaza sem szuperkulcs. Ha K egyelemű, akkor *egyszerű kulcsnak*, ha többelemű, akkor *összetett kulcsnak* hívjuk. Előfordulhat, hogy egy relációsémának több kulcsa is van. Ilyenkor praktikus megfontolások alapján kiválasztunk ezek közül egyet. Ezt hívjuk *elsődleges kulcsnak*. Míg kulcsból több is lehet, egy relációséma esetén, elsődleges kulcsból mindig

csak egy van, az, amelyiket kiválasztottuk. A **Fórum** adatbázisunk esetében például a Felhasználó egyednél megállapítottuk, hogy a felhasználónév és az email is alkalmas külön-külön az egyértelmű azonosításra, tehát mindkét attribútum kulcs. Ezek közül a felhasználónevet választottuk elsődleges kulcsnak. A relációsémákban az elsődleges kulcs attribútumait aláhúzással jelöljük.

Most, hogy már egyértelműen tudunk hivatkozni egy tábla egy adott sorára, ki tudjuk alakítani a kapcsolatokat a különböző táblák között is. Ehhez egy új foglamat vezetünk be. Egy $R(A)$ relációséma $K \subseteq A$ részhalmaza *külső kulcs* (más néven idegen kulcs), ha egy másik (vagy ugyanazon) séma elsődleges kulcsára hivatkozik. A külső kulcsot dőlt betűvel vagy a hivatkozott séma kulcsára mutató nyillal jelöljük.

Kiemeljük, hogy mind a kulcs, mind a külső kulcs egy sémára vonatkozó feltétel előírása, azaz az aktuális adattáblák tartalmától függetlenek.

A *relációs adatbázisséma* egy adatbázis összes relációs sémájának megadását jelenti, beleértve az elsődleges kulcsok és külső kulcsok leírását is.

3.2.1. példa

Az 1.3. ábrán látható adatbázis adatbázissémája az alábbi.

`ÜGYFELEK`(ügyfélkód, ügyfél neve)
`RENDELKEZIK`(*ügyfélkód*, *számlaszám*)
`SZÁMLÁK`(számlaszám, számla neve)

Az `ÜGYFELEK` sémában az {ügyfélkód, ügyfél neve} szuperkulcs, alkalmas egy ügyfél egyértelmű azonosítására, de nem minimális, mert az {ügyfélkód} önmagában is egyértelműen kijelöl egy ügyfelet. Az {ügyfélkód} tehát kulcs és mivel egyelemű, így egyszerű kulcs is. Nincs más kulcsa a sémának, így ez az elsődleges kulcs is. A `SZÁMLÁK` sémában a {számlaszám, számla neve} szuperkulcs, alkalmas egy számla egyértelmű azonosítására, de nem minimális, mert a {számlaszám} önmagában is egyértelműen kijelöl egy számlát. A {számlaszám} tehát kulcs és mivel egyelemű, így egyszerű kulcs is. Nincs más kulcsa a sémának, így ez az elsődleges kulcs is. A `RENDELKEZIK` sémában összetett kulcs van, az {ügyfélkód, számlaszám}. Az {ügyfélkód} külső kulcs, az `ÜGYFELEK` séma elsődleges kulcsára mutat. A {számlaszám} is külső kulcs, a `SZÁMLÁK` séma elsődleges kulcsára mutat.

Kérdések és feladatok

1. Adjon meg egy olyan relációs sémát, melyben több kulcs is van!

2. Miért nem elegendő a 3.2.1. példa `RENDELKEZIK` sémájában önállóan az ügyfélkód vagy a számlaszám egyértelmű azonosításra, azaz miért nincs a sémának egyszerű kulcsa?

3. Adjon példát egy olyan valós jelenséget leíró adatbázissémára, mely két relációs sémát tartalmaz úgy, hogy az egyiknek nincs egyszerű, csak összetett kulcsa, a másik pedig ezt külső kulcsként tartalmazza!

4. Tekintsük a `DOLGOZÓK`(személyi szám, adószám, dolgozó neve, dolgozó címe, fizetés) relációs sémát, melyet egy vállalat dolgozóinak tárolására hoztunk létre. Határozza meg a séma szuperkulcsait! Mely(ek) kulcs(ok) ezek közül?

4. fejezet

Relációs adatbázisséma felírása E-K diagramból

Az előzőekben megismertük az E-K modellt mint az adatmodelltől független, az adatok közti összefüggéseket leíró diagram alapú technikát, valamint a konkrét relációs adatmodellt. A következő lépés az, hogy megvizsgáljuk, hogyan írható fel a relációs adatbázisséma az E-K modell ismeretében. Először az egyedeket képezzük le, beleértve a gyenge egyedeket is, majd az összetett és többértékű attribútumokkal foglalkozunk. Ezután az általános kapcsolatok átírását vizsgáljuk meg, legvégül pedig a specializáló kapcsolatokat írjuk át.

4.1. Egyedek, gyenge egyedek leképezése

Az egyedek leképezése egyszerűen adódik: Minden az E-K diagramon szereplő egyedhez egy-egy relációsémát írunk fel, melynek neve az egyed neve, attribútumai az egyed attribútumai, kulcsa pedig az egyed kulcsa.

A **Fórum** példánk 2.4. ábrán látható E-K diagramja alapján a három egyedből három relációséma keletkezik:

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, név, utolsó belépés időpontja)
`ÜZENET`(sorszám, tartalom)
`HÍRFOLYAM`(azonosító, megnevezés, kulcsszavak)

Vegyük észre, hogy itt a többértékű és az összetett attribútumokkal nem foglalkoztunk, azokat egyszerű attribútumként tüntettük fel. Ezeket hamarosan újból megvizsgáljuk, előbb azonban kitérünk arra, mi a teendő gyenge egyedek esetén. A szabály itt ugyanaz, mint az előző esetben, azzal kiegé-

szítve, hogy a gyenge egyed relációsémáját bővíteni kell a meghatározó kapcsolat(ok)on keresztül kapcsolódó egyedek kulcsattribútumaival, ami aztán külső kulcsként jelenik meg a sémában.

4.1.1. példa

A 2.2.4 példához tartozó 2.8. ábrán látható eszköznyilvántartás esetében a Laptop egyed sémája a következő lesz:

`LAPTOP`(*személyi szám*, CPU, RAM, HDD, SSD)

A kulcsattribútum(ok) kiválasztása azonban itt körültekintést igényel. Ha egy dolgozónak több ugyanolyan paraméterű gépe is lehet, akkor a személyi szám önmagában nem alkalmas egy gép egyértelmű azonosítására. A probléma ilyenkor áthidalható, ha felveszünk egy további sorszám attribútumot, amellyel az ugyanahhoz a dolgozóhoz tartozó megegyező gépeket megkülönböztetjük. Ezt az E-K diagramon a meghatározó kapcsolat attribútumaként jelöljük, amely a leképezés során szintén bekerül a gyenge egyed sémájába:

`LAPTOP`(*személyi szám*, sorszám, CPU, RAM, HDD, SSD)

4.2. Összetett és többértékű attribútumok leképezése

Térjünk most vissza az összetett és többértékű attribútumok átírására. Mivel a relációs modell halmazok, listák és struktúrák tárolását nem teszi lehetővé, ezért már a sémák felírása során úgy kell eljárnunk, hogy az E-K diagramból átírt attribútumok elemiek legyenek. Az összetett attribútumok esetén ennek módja az, hogy az eredeti összetett attribútum helyett az azt alkotó elemi attribútumokat szerepeltetjük a sémában. Azaz a **Fórum** példa esetében a felhasználó sémája így alakul:

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév,
utolsó belépés időpontja)

A sémát itt csak a szemléltetés végett alakítottuk ki két lépésben. A gyakorlatban ha egy összetett attribútumot látunk az E-K diagramon, akkor azt az egyed átírása során azonnal helyettesíthetjük az őt alkotó elemi attribútumokkal.

A többértékű attribútumok estén különböző lehetőségeink vannak. A legegyszerűbb megoldás az, hogy eltekintünk attól, hogy az attribútum többértékű, és egyszerű attribútumként reprezentáljuk. Ez a Hírfolyam egyed esetén azt jelentené, hogy a kulcsszavak vesszővel elválasztva egy egyszerű sztringre képződnének le, mint ahogy az a következő példában látható.

HÍRFOLYAM

azonosító	megnevezés	kulcsszavak
1	Adatbázis kérdések	adatbázis, SQL, oktatás
2	PHP hírek	PHP, programozás
3	Ki a legjobb tanár	vélemény, oktatás
4	Milyen gépet vegyek	hardver

Ennek a megközelítésnek azonban hátránya, hogy a kulcsszavak nem kezelhetők külön-külön. Ennek következtében például nehézkes lesz azon hírfolyamok kiválogatása, melyekhez egy konkrét kulcsszót rendeltek hozzá, ez ugyanis különböző sztringműveleteket fog igényelni.

Egy másik megoldás az lehet, hogy minden hírfolyamhoz annyi sort veszünk fel, ahány kulcsszóval rendelkezik.

HÍRFOLYAM

azonosító	megnevezés	kulcsszavak
1	Adatbázis kérdések	adatbázis
1	Adatbázis kérdések	SQL
1	Adatbázis kérdések	oktatás
2	PHP hírek	PHP
2	PHP hírek	programozás
3	Ki a legjobb tanár	vélemény
3	Ki a legjobb tanár	oktatás
4	Milyen gépet vegyek	hardver

Ebben az esetben azonban jól látható módon a Hírfolyam egyed azonosítója már nem elegendő önmagában az egyértelmű azonosításra, megszűnik kulcsnak lenni. Ráadásul a sorok többszörözése redundáns adattároláshoz vezet, ami egyrészt tárpazarló megoldás, másrészt az adatbázis adatainak épségét is nehezebb így garantálni, ahogy azt majd a normalizálásról szóló fejezetben taglaljuk. Ez a megoldás épp ezért kerülendő. Ehelyett célszerű új táblát létrehoznunk, amelybe kigyűjtjük, hogy melyik hírfolyamhoz mely kulcsszavak tartoznak.

HÍRFOLYAM

azonosító	megnevezés
1	Adatbázis kérdések
2	PHP hírek
3	Ki a legjobb tanár
4	Milyen gépet vegyek

HÍRFOLYAM_KULCSSZAVAK

azonosító	kulcsszavak
1	adatbázis
1	SQL
1	oktatás
2	PHP
2	programozás
3	vélemény
3	oktatás
4	hardver

Amennyiben a kulcsszavak sorrendje nem fontos (tehát halmazként képződnek le ezek az adatok), akkor ez a megoldás már kielégítő. Ha azonban

a sorrendnek is szerepe van (azaz lista adatszerkezetet szeretnénk használni), akkor minden rekordot bővítünk még egy sorszám attribútummal is. Amennyiben a kulcsszavak ismétlődését is el kívánjuk kerülni, akkor azokat is kitehetjük egy külön táblába és egy kapcsoló tábla segítségével kötjük össze a kulcsszavakat a hírfolyamokkal. Ha így járunk el, akkor célszerű ezt az azonosítót is jelölni az E-K diagramon.

HÍRFOLYAM

hírfolyam azonosító	**megnevezés**
1	Adatbázis kérdések
2	PHP hírek
3	Ki a legjobb tanár
4	Milyen gépet vegyek

KULCSSZAVAK

kulcsszó azonosító	**kulcsszó**
1	adatbázis
2	SQL
3	oktatás
4	PHP
5	programozás
6	vélemény
7	hardver

HÍRFOLYAM_KULCSSZAVAK

hírfolyam azonosító	**kulcsszó azonosító**
1	1
1	2
1	3
2	4
2	5
3	6
3	3
4	7

4.3. Kapcsolatok, specializáló kapcsolatok leképezése

Kapcsolatok leképezése során a következő általános szabály szerint járunk el: Minden kapcsolathoz felveszünk egy új sémát, melynek neve a kapcsolat neve, attribútumai pedig a kapcsolódó egyedek kulcsattribútumai, továbbá a kapcsolat saját attribútumai. Formálisan, ha a kapcsolat az $E_1, E_2, \ldots, E_n$ egyedeket köti össze, melyek kulcsattribútumai rendre a $K_1, K_2, \ldots, K_n$ halmazokkal adottak, továbbá a kapcsolat saját attribútumai $A_1, A_2, \ldots, A_m$, akkor egy Kapcsolat$(K_1, K_2, \ldots, K_n, A_1, A_2, \ldots, A_m)$ séma keletkezik. A séma kulcsának kiválasztása további megfontolásokat igényel. Ha azonban úgy találjuk, hogy az új séma kulcsa megegyezik valamely kapcsolódó egyed kulcsával, akkor a kapcsolat sémája az egyed sémájába olvasztható. Ezt a lépést hívjuk *konszolidációnak*. Némi gyakorlattal a két lépés már egyben is elvégezhető, azaz nem kell új sémát létrehozni, majd sémákat összeolvasztani, hanem elegendő a már meglévő sémákat bővítenünk. Vizsgáljuk meg ezt a **Fórum** példa kapcsolatain keresztül.

FELHASZNÁLÓ(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
ÜZENET(sorszám, tartalom)
HÍRFOLYAM(azonosító, megnevezés, kulcsszavak)
ÍRTA(felhasználónév, sorszám, mikor)
KÖVETI(felhasználónév, azonosító)
LÉTREHOZTA(felhasználónév, azonosító)
HOZZÁTARTOZIK(sorszám, azonosító)

Az első három sémát korábban már láttuk, ezek az egyedek átírása során keletkeztek. Az ÍRTA és HOZZÁTARTOZIK sémák esetében maga a sorszám meghatározza egyértelműen, hogy melyik fórumbejegyzésről van szó. Mivel a kulcsaik megegyeznek a kapcsolódó Üzenet egyed sémájának kulcsával, így ezek a sémák beolvaszthatók az ÜZENET sémába. Hasonló megfontolások alapján a LÉTREHOZTA séma is beolvasztható a HÍRFOLYAM sémába. A KÖVETI séma esetében a felhasználónév nem elegendő ahhoz, hogy tudjuk, hogy a felhasználó melyik sémát követi. Ugyanígy az azonosító önmagában nem elég ahhoz, hogy megmondjuk, hogy melyik felhasználó követi az adott azonosítójú hírfolyamot. Ezért a két attribútumra együttesen van szükség a kulcs kialakításához. Mivel a csatlakozó egyedek kulcsával ez nem egyezik meg, így összevonás ebben az esetben nem végezhető. A végső sémák az alábbiak lesznek.

FELHASZNÁLÓ(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
ÜZENET(sorszám, tartalom, *felhasználónév*, mikor, *azonosító*)
HÍRFOLYAM(azonosító, megnevezés, kulcsszavak, *felhasználónév*)
KÖVETI(*felhasználónév*, *azonosító*)

Azt láthatjuk tehát, hogy összevonást az 1:N típusú kapcsolatok esetén végeztünk. Ugyanez lenne érvényes az 1:1 típusú kapcsolatok esetében is. Ezek alapján az alábbi szabályokat fogalmazhatjuk meg az E-K diagramok sémákba való átírásával kapcsolatban.

- 1:1 kapcsolat esetén az egyik tetszőlegesen választott kapcsolódó egyed sémáját bővítjük a másik egyed kulcsattribútumaival, valamint a kapcsolat saját attribútumaival.
- 1:N kapcsolat esetén az N-oldali egyed sémáját bővítjük a másik egyed kulcsattribútumaival, valamint a kapcsolat saját attribútumaival.
- N:M típusú vagy többágú kapcsolat esetén új sémát veszünk fel, mely-

nek attribútumai a kapcsolódó egyedek kulcsattbibútumai, valamint a kapcsolat saját attribútumai.

Figyeljük meg, hogy amikor egy egyed kulcsattribútumai bekerülnek egy csatlakozó egyed sémájába vagy egy újonan létrejövő sémába, akkor ott külső kulcs szerepet fognak betölteni. Valóban, ezek a külső kulcsok teremtik meg a kapcsolódást a különböző egyedek között.

Ha most visszaemlékezünk a gyenge egyedek leképezésével kapcsolatban megismertekre, akkor láthatjuk, hogy lényegében ott is egy kapcsolat leképezésére kerül sor: a meghatározó (1:N típusú) kapcsolaton keresztül bővítjük a gyenge egyed sémáját. Ez történt a 2.8. ábrán adott eszköznyilvántartás esetében a Laptop egyed sémájának kialakításakor is. Természetesen itt is igaz, hogy a kapcsolat saját attribútumai is bekerülnek az N-oldali egyed sémájába, ilyen azonban ebben a konkrét példában nem volt.

Végül térjünk ki arra a speciális esetre, amikor a kapcsolat mindkét oldalán ugyanaz az egyedtípus szerepel. Ezek is bináris kapcsolatoknak tekinthetők, így ugyanazok az átírási szabályok érvényesek, mint a korábbiakban. Azonban, hogy mindig egyértelmű legyen, hogy melyik oldali egyedpéldányra hivatkozunk, így ezek kulcsait átnevezéssel különböztetjük meg a sémában.

4.3.1. példa

A 2.2.1 és 2.2.2 példákhoz tartozó 2.5. és 2.6. ábrán látható E-K diagramok az alábbi sémákba íródnak át (a sémákban az attribútumok sorrendje közömbös).

`DOLGOZÓ`(azonosító, név, fizetés, *főnök azonosító*)
`MÉRKŐZÉS`(hazai csapat azonosító, *vendég csapat azonosító*, eredmény)

A specializáló kapcsolatok leképezésére nincs általánosan előnyös módszer a relációs adatmodellek esetén. Többféle módon is eljárhatunk, de mindegyik megközelítésnek lehetnek hátrányai, ezért a megvalósítás során mérlegelnünk kell, hogy melyik utat választjuk. Eljárhatunk úgy, hogy a főtípust és minden altípust is külön új sémában jelenítünk meg úgy, hogy az altípusok sémájába felvesszük a főtípus sémájának attribútumait is. Ilyenkor minden egyedpéldány csak egy táblában fog szerepelni.

4.3.2. példa

A 2.2.3 példát és annak 2.7. ábrán adott E-K diagramját követve, ha a főtípust és minden altípust külön sémában jelenítünk meg úgy, hogy az altípusok sémájába felvesszük a főtípus sémájának attribútumait is,

akkor az alábbi sémákhoz jutunk.

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, név, utolsó belépés időpontja)
`MODERÁTOR`(felhasználónév, jelszó, email, név, utolsó belépés időpontja, mióta)

A megközelítés hátránya, hogy előfordulhat, hogy kereséskor esetleg több táblát is vizsgálni kell. A 4.3.2 példában ha adott nevű felhasználót keresünk, akkor ahhoz végig kell néznünk mind a `FELHASZNÁLÓ`, mind a `MODERÁTOR` táblát. Másik hátulütője ennek a módszernek az, hogy a kombinált altípusokat, csak új tábla felvételével képes kezelni.

4.3.3. példa

A 4.3.2 példát folytatva tegyük fel, hogy van még egy lektori szerepkör is, ahol megadjuk, hogy ki milyen nyelven lektorál (az egyszerűség kedvéért egy lektor csak egy nyelven lektorál). Ebben az esetben, ha vannak olyan moderátorok, akik egyben lektorok is, akkor azoknak egy további sémát kellene létrehoznunk.

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, név, utolsó belépés időpontja)
`MODERÁTOR`(felhasználónév, jelszó, email, név, utolsó belépés időpontja, mióta)
`LEKTOR`(felhasználónév, jelszó, email, név, utolsó belépés időpontja, nyelv)

Egy másik módszer az, hogy továbbra is minden altípushoz felveszünk egy új sémát, de úgy, hogy abban csak a főtípus kulcsattribútumai jelennek meg. Ilyenkor minden egyedpéldány szerepel a saját altípusának (vagy altípusainak) táblájában és a főtípus táblájában is.

4.3.4. példa

A 2.2.3 példát és annak 2.7. ábrán adott E-K diagramját követve, ha a főtípust és minden altípust külön sémában jelenítünk meg úgy, hogy az altípusok sémájában csak a főtípus kulcsattribútumai jelenjenek meg, akkor az alábbi sémákhoz jutunk.

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, név, utolsó belépés időpontja)
`MODERÁTOR`(felhasználónév, mióta)

Sajnos ebben az esetben is előfordulhat, hogy a keresés során több táblát is igénybe kell vennünk. Ha a 4.3.4 példában az idén moderátori státuszt nyert felhasználók nevét szeretnénk lekérdezni, akkor először a `MODERÁTOR` táblában kell keresnünk, majd a moderátorok nevét a `FELHASZNÁLÓ` táblából kapjuk meg.

Eljárhatunk úgy is, hogy egyetlen közös táblát alkotunk, melyben szerepel a főtípus és az altípusok összes attrbibútuma és a táblában NULL értékkel töltjük fel a nem releváns cellákat.

4.3.5. példa

A 2.2.3 példát és annak 2.7. ábrán adott E-K diagramját követve, ha a főtípust és minden altípust egy közös sémában jelenítünk meg, akkor az alábbi sémához jutunk.

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, név, utolsó belépés időpontja, mióta)

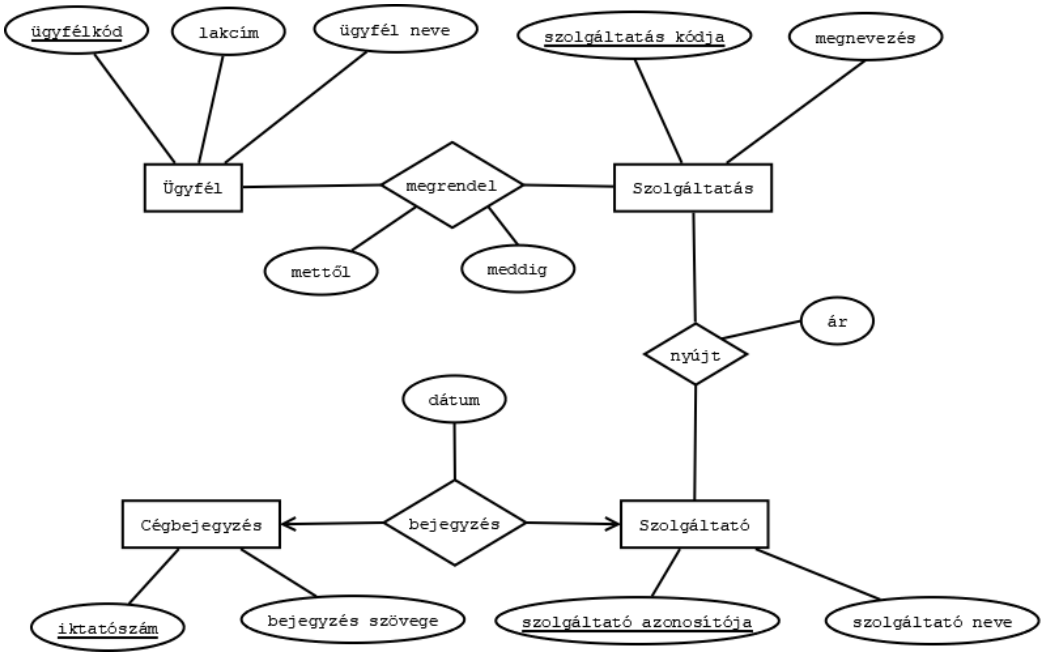
Ilyenkor értelemszerűen csak egy táblában kell keresni, azonban a sok NULL érték (a 4.3.5 példában minden olyan felhasználó esetén, aki nem moderátor, ezt az értéket veszi fel a „mióta” mező) miatt pazarlóan bánunk a memóriával (háttértárral). További problémát okozhat, ha egy mező értéke nem azért lesz NULL, mert ténylegesen nem ismert. A 4.3.5 példában, ha egy lektor esetében nem ismerjük a kinevezésének dátumát, akkor a megfelelő mezőt NULL értékkel töltjük fel. Innentől azonban nem utal semmi arra, hogy az illető ténylegesen lektor vagy csak egy közönséges felhasználó.

Kérdések és feladatok

1. A többértékű attribútumok leképezésénél látott harmadik módszer azt javasolja, hogy vegyünk fel új táblát, melyben felsoroljuk az attribútum által felvett elemi értékeket minden egyedpéldány esetén. Milyen változtatást jelent ez a **Fórum** példa 2.4. ábrán látható E-K diagramján? Ha a többértékű attribútum értékeinek sorrendje is fontos, akkor egy

sorszám attribútummal is bővítjük a sémát. Hol jelenik meg az E-K diagramon ez az attribútum?

2. Értelmezze az alábbi E-K diagramot, majd írja fel a diagram alapján a megfelelő relációs adatbázissémát!



5. fejezet

Relációs algebra

Most, hogy már ismerjük az elméleti megoldást arra, hogy miként tároljuk relációs adatbázisban az adatainkat, megvizsgáljuk, hogy a tárolt adatokon milyen műveleteket végezhetünk. Ebben a fejezetben a lekérdezésekhez szükséges matematikai alapokat tekintjük át. Először megismerkedünk a halmazelméleti műveletekkel, majd a redukciós és kombinációs műveletek bemutatása következik. Bár a relációs modellek halmazokkal dolgoznak, azaz minden rekord különböző kell, hogy legyen, a relációs adatbáziskezelők sokszor megengedik azonos sorok ismétlődését is, azaz nem halmazokkal, hanem úgy nevezett multihalmazokkal végzik a műveleteket. Mi itt a modell alapján a műveleteket halmazokra ismertetjük, de hasonló elven megvalósíthatók a multihalmazokra vett relációs algebrai műveletek is.

5.1. Halmazműveletek

Egy relációs adattáblát értelmezhetünk úgy, mint soroknak (rekordoknak) a halmaza. Ebből természesetesen adódik, hogy akkor adattáblák között legyen lehetőség halmazelméleti műveletek elvégzésére. Ez azonban csak akkor megengedett, ha a műveletben résztvevő két relációséma kompatibilis. Az $R_1(A_1, \ldots, A_n)$ és $R_2(B_1, \ldots, B_m)$ relációsémák *kompatibilisek*, ha $n = m$ és $dom(A_i) = dom(B_i)$ $(i = 1, \ldots, n)$. Két táblát pedig akkor nevezünk kompatibilisinek, ha a sémáik kompatibilisek. Legyen most T_1 és T_2 két tetszőleges kompatibilis tábla. A halmazelméleti műveletek az alábbiak.

Unió: A $T_1 \cup T_2$ tábla azokat a rekordot tartalmazza, melyek T_1 és T_2 közül legalább az egyikben szerepelnek. Ez technikailag úgy érhető el, hogy a két táblát egymás után írjuk és az ismétlődő sorokat töröljük. Természetesen $T_1 \cup T_2 = T_2 \cup T_1$ minden esetben fenáll, azaz az unióképzés kommutatív.

Metszet: A $T_1 \cap T_2$ tábla azokat a rekordokat tartalmazza, melyek T_1-ben és T_2-ben is szerepelnek. A metszetképzés is kommutatív, azaz $T_1 \cap T_2 = T_2 \cap T_1$ mindig teljesül.

Különbség: A $T_1 \setminus T_2$ tábla azokat a rekordokat tartalmazza, melyek T_1-ben szerepelnek, de T_2-ben nem. A különbségképzés nem kommutatív, azaz általában $T_1 \setminus T_2 \neq T_2 \setminus T_1$.

5.1.1. példa

Tekintsünk két táblát, melyek egy intézmény két különböző chipkártyás beléptetőrendszerének adatait tartalmazzák. Az első tábla azt mutatja, hogy mely dolgozók léphetnek be a szerverterembe, a második azt, hogy kik mehetnek be a nyomtatószobába. A két tábla sémája ennek megfelelően:

`SZERVER`(kártyaszám, név)
`NYOMTATÓ`(kártya ID, dolgozó neve)

Vegyük észre, hogy a két tábla kompatibilitásához nem szükséges, hogy az attrbibútumok elnevezései rendre megegyezzenek, elég, ha csak az értéktartományaik egyeznek. Tegyük fel, hogy a táblák tartalama az alábbi:

`SZERVER`

009	Németh Gábor
002	Bodnár Péter
001	Balázs Péter

`NYOMTATÓ`

001	Balázs Péter
103	Kardos Péter

Ha most arra vagyunk kíváncsiak, hogy kik azok, akik a két helyiség közül valamelyikbe (legalább az egyikbe) beléphetnek, akkor az unióképzést kell segítségül hívnunk.

`SZERVER` $\cup$ `NYOMTATÓ`

009	Németh Gábor
002	Bodnár Péter
001	Balázs Péter
103	Kardos Péter

Ha azt szeretnénk megtudni, hogy ki az, aki mindkét helyiségbe bemehet, akkor a két tábla metszetét kell képeznünk.

`SZERVER` $\cap$ `NYOMTATÓ`

001	Balázs Péter

A különbségképzéssel pedig arra kaphatunk választ, hogy kik azok akik

az egyik helyiségbe beléphetnek, de a másikba nem.

SZERVER \ NYOMTATÓ

009	Németh Gábor
002	Bodnár Péter

NYOMTATÓ \ SZERVER

103	Kardos Péter

5.2. Redukciós műveletek

A redukciós műveletek sorok vagy oszlopok elhagyásával képeznek egy táblából egy másik kisebb táblát. Két redukciós művelettel ismerkedünk meg.

Projekció (vetítés): Oszlopok kiválasztása egy tetszőleges T táblából. Jelölése: $\pi_{attribútumlista}(T)$. Az eredménytáblában T csak azon oszlopai jelennek meg (és abban a sorrendben), melyek attribútumai a listában szerepelnek.

Szelekció (kiválasztás): Adott logikai feltételnek megfelelő sorok kiválasztása egy tetszőleges T táblából. Jelölése: $\sigma(T)$, ahol σ egy logikai feltétel.

5.2.1. példa

Tekintsük a **Fórum** adatbázisunkat, annak is a FELHASZNÁLÓ tábláját, és tegyük fel, hogy az az alábbi rekordokat tartalmazza (egyes attribútumok elnevezését a szebb elrendezhetőség érdekében rövidítettük).

FELHASZNÁLÓ

felh. név	jelszó	email	vezetéknév	keresztnév	utolsó belépés időp.
pbalazs	e(RpL9IU2	pbalazs@inf.u-szeged.hu	Balázs	Péter	2018-10-03 11:10:00
pkardos	87fiHh9O	pkardos@inf.u-szeged.hu	Kardos	Péter	2018-10-06 9:45:00
gnemeth	2XgfSStw	gnemeth@inf.u-szeged.hu	Németh	Gábor	2018-10-15 17:00.00
bodnaar	JkFrrS7s	bodnaar@inf.u-szeged.hu	Bodnár	Péter	NULL

Utóbbi esetben a NULL bejegyzés azt jelenti, hogy az illető regisztrációja óta még nem lépett be a rendszerbe. Ha most csak azt szeretnénk kilistázni, hogy mely felhasználók mikor léptek be utoljára a rendszerbe (vezetéknév, keresztnév, felhasználónév, utolsó belépés időpontja) alakban, akkor a projekció műveletét kell alkalmaznunk.

$\pi_{(vezetéknév, keresztnév, felhasználónév, utolsó\ belépés\ időpontja)}$(FELHASZNÁLÓ)

vezetéknév	keresztnév	felhasználónév	utolsó belépés időpontja
Balázs	Péter	pbalazs	2018-10-03 11:10:00
Kardos	Péter	pkardos	2018-10-06 9:45:00
Németh	Gábor	gnemeth	2018-10-15 17:00.00
Bodnár	Péter	bodnaar	NULL

Amennyiben pedig arra lennénk kíváncsiak, hogy kik azok, akik utoljára 2018.10.10 előtt léptek be, akkor szelekciót kell alkalmazzunk:

$\sigma_{utolsó\ belépés\ időpontja\ <\ '2018.10.10\ 0:00:00'}(\texttt{FELHASZNÁLÓ})$

felh. név	jelszó	email	vezetéknév	keresztnév	utolsó belépés időp.
pbalazs	eRpL9IU2	pbalazs@inf.u-szeged.hu	Balázs	Péter	2018-10-03 11:10:00
pkardos	87fiHh9O	pkardos@inf.u-szeged.hu	Kardos	Péter	2018-10-06 9:45:00

Nézzük meg, mi történik, ha a 2018.10.10-ei vagy az utáni bejelentkezéseket listázzuk ki.

$\sigma_{utolsó\ belépés\ időpontja\ \geq\ '2018.10.10\ 0:00:00'}(\texttt{FELHASZNÁLÓ})$

felh. név	jelszó	email	vezetéknév	keresztnév	utolsó belépés időp.
gnemeth	2XgfSStw	gnemeth@inf.u-szeged.hu	Németh	Gábor	2018-10-15 17:00.00

Ha a két feltételt összekötjük a diszjunkció (OR) művelettel és ezzel a teljes időintervallumot lefedjük, ez adódik.

$\sigma_{(utolsó\ belépés\ időp.\ <\ '2018.10.10\ 0:00:00'\ \text{OR}\ utolsó\ belépés\ időp.\ \geq\ '2018.10.10\ 0:00:00')}(\texttt{FELHASZNÁLÓ})$

felh. név	jelszó	email	vezetéknév	keresztnév	utolsó belépés időp.
pbalazs	eRpL9IU2	pbalazs@inf.u-szeged.hu	Balázs	Péter	2018-10-03 11:10:00
pkardos	87fiHh9O	pkardos@inf.u-szeged.hu	Kardos	Péter	2018-10-06 9:45:00
gnemeth	2XgfSStw	gnemeth@inf.u-szeged.hu	Németh	Gábor	2018-10-15 17:00.00

Hova tűnt az 5.2.1 példa utolsó műveleténél az eredeti tábla utolsó sora? A válasz az adatbáziskezelő rendszerek egy sajátosságában rejlik, nevezetesen, hogy háromértékű logikával dolgoznak, azaz az IGAZ (TRUE) és HAMIS (FALSE) logikai értékek mellett megkülönböztetnek még egy ISMERETLEN (UNKNOWN) logikai értéket is. Ez a logika a kétértékű logika kiterjesztéseként adódik, tehát a kétértékű logikában megszokott diszjunkció, konjunkció és negáció műveletek értéktáblázata továbbra is érvényes. A logikai feltételekben azonban a NULL értékkel vett összehasonlítások vezethetnek UNKNOWN eredményre, amit már minden rendszer a sajátosságainak megfelelően kezel. A NULL értékek vizsgálatáról a megvalósításról szóló Lekérdezések című fejezetben még részletesebben lesz szó.

Végezetül megjegyezzük, hogy a szelekció művelete kommutatív, azaz tetszőleges T tábla esetén $\sigma_{feltétel_1}(\sigma_{feltétel_2}(T)) = \sigma_{feltétel_2}(\sigma_{feltétel_1}(T)) = \sigma_{feltétel_1\ \text{AND}\ feltétel_2}(T)$.

5.3. Kombinációs műveletek

A kombinációs műveletek két táblát kapcsolnak össze és egy olyan táblát eredményeznek, melyben a két tábla mindegyikének bizonyos oszlopai megjelennek. A továbbiakban feltesszük, hogy adott egy $R_1(A_1, \ldots, A_n)$ séma

feletti tetszőleges T_1 és egy $R_2(B_1, \ldots, B_m)$ séma feletti tetszőleges T_2 tábla.

Descartes-szorzat

A T_1 és T_2 táblák $T = T_1 \times T_2$ *Descrates-szorzatának* sémája

$$R(A_1, \ldots, A_n, B_1, \ldots, B_m)$$

alakú és a tábla sorait úgy kapjuk, hogy a T_1 tábla minden sorát párosítjuk a T_2 tábla minden sorával. Ha a tábláknak vannak azonos nevű attribútumai, akkor ezeket a táblanévvel mint előtaggal különböztetjük meg egymástól. Ha a T_1 táblának r_1 sora és c_1 oszlopa van, a T_2 táblának pedig r_2 sora és c_2 oszlopa, akkor a $T_1 \times T_2$ táblának $r_1 r_2$ sora és $c_1 + c_2$ oszlopa lesz. Fontos továbbá, hogy a Descartes-szorzatból az eredeti táblák visszanyerhetők a projekció segítségével: $T_1 = \pi_{A_1,\ldots,A_n}(T)$ és $T_2 = \pi_{B_1,\ldots,B_m}(T)$.

5.3.1. példa

Legyen $T_1 = \pi_{(felhasználónév, utolsó\ belépés\ időpontja)}(\texttt{FELHASZNÁLÓ})$ a `FELHASZNÁLÓ` tábla egy redukált változata, melyben csak a felhasználói azonosítók és a belépések időpontjai szerepelnek és $T_2 = \texttt{ÜZENET}$.

T_1

felhasználónév	utolsó belépés időpontja
pbalazs	2018-10-03 11:10:00
pkardos	2018-10-06 9:45:00
gnemeth	2018-10-15 17:00.00
bodnaar	NULL

T_2

sorszám	tartalom	felhasználónév
1	Minden rendben.	pbalazs
2	Én is hozzászólok.	pbalazs
3	Mi újság?	pkardos

Ekkor a $T_1 \times T_2$ tábla tartalma

$T_1 \times T_2$

T_1.felhasználónév	utolsó belépés időpontja	sorszám	tartalom	T_2.felhasználónév
pbalazs	2018-10-03 11:10:00	1	Minden rendben.	pbalazs
pbalazs	2018-10-03 11:10:00	2	Én is hozzászólok.	pbalazs
pbalazs	2018-10-03 11:10:00	3	Mi újság?	pkardos
pkardos	2018-10-06 9:45:00	1	Minden rendben.	pbalazs
pkardos	2018-10-06 9:45:00	2	Én is hozzászólok.	pbalazs
pkardos	2018-10-06 9:45:00	3	Mi újság?	pkardos
gnemeth	2018-10-15 17:00:00	1	Minden rendben.	pbalazs
gnemeth	2018-10-15 17:00:00	2	Én is hozzászólok.	pbalazs
gnemeth	2018-10-15 17:00:00	3	Mi újság?	pkardos
bodnaar	NULL	1	Minden rendben.	pbalazs
bodnaar	NULL	2	Én is hozzászólok.	pbalazs
bodnaar	NULL	3	Mi újság?	pkardos

A Descartes-szorzat az összes lehetséges párosítást tartalmazza, melyek közül vannak „értelmetlenek” is. Az 5.3.1 példában látszólag semmi haszna nincs a ’pbalazs’ felhasználó bejelentkezési adatához párosítani egy másik felhasználó fórum bejegyzéseit. Valóban, önmagában a Descartes-szorzat

nem elegendő gyakorlati szempontból hasznos listák összeállításához, viszont alapját képezi a további kombinációs műveleteknek.

Természetes összekapcsolás

A *természetes összekapcsolás* (Natural Join) a Descartes-szorzatnak csak azon sorait tartja meg, amelyekben a párosított adatok logikailag is valóban összetartoznak. Ez a gyakorlatban legtöbbször külső kulcs mentén történő összekapcsolást jelent. Tudjuk, hogy egy séma egy külső kulcsa egy másik séma elsődleges kulcsára hivatkozik. Adódik tehát, hogy az összekapcsolás során csak azokat a sorokat tartsuk meg, melyekben a hivatkozó és a hivatkozott értékek megegyeznek. Emellett az ismétlődő oszlopkat is csak egyszer jelenítjük meg. Értelemszerűen ezt projekció és szelekció segítségével lehet megtenni. A rövidebb jelölés érdekében vezessük be az $A = \{A_1, \ldots, A_n\}$ és $B = \{B_1, \ldots, B_m\}$ attribútumhalmazokat. A természetes összekapcsoláshoz kellenek a két sémából közös attribútumok, feltesszük tehát, hogy $X = A \cap B \neq \emptyset$. Ekkor a két tábla természetes összekapcsoltja a $T_1 \bowtie T_2 = \pi_{A \cup B}(\sigma_{R_1.X=R_2.X}(T_1 \times T_2))$ tábla. Azaz a Descartes-szorzatból előbb kiválasztjuk a közös attribútumon megegyező sorokat, majd megszüntetjük az ismétlődéseket.

5.3.2. példa

A 5.3.1 példában a felhasználónév a két tábla közös közös attrbibútuma, így az ezen való egyezéssel válogatjuk le a sorokat a természetes összekapcsolás során.

$T_1 \bowtie T_2$

felhasználónév	utolsó belépés időpontja	sorszám	tartalom
pbalazs	2018-10-03 11:10:00	1	Minden rendben.
pbalazs	2018-10-03 11:10:00	2	Én is hozzászólok.
pkardos	2018-10-06 9:45:00	3	Mi újság?

Míg a Descartes-szorzatból projekcióval előállíthatók az eredeti táblák, addig a természetes összekapcsolás esetén ez már nem igaz. Az eredeti táblák azon sorai, amelyek nem találnak párt maguknak, elvesznek az összekapcsolás során. Ezeket *lógó soroknak* hívjuk. A 5.3.2 példában elvesztettük azokat a felhasználókat, akinek még nem voltak hozzászólásaik.

Külső összekapcsolás

A lógó sorok megtartása érdekében bevezetjük a *külső összekapcsolás* (outer join) műveletét is. A külső összekapcsolás lehet baloldali (left outer join), jobboldali (right outer join) vagy kétoldali (full outer join). A $T_1 \bowtie T_2$ baloldali összekapcsolás esetén a természetes összekapcsoláson túl a T_1 tábla

azon sorai is megmaradnak, melyek nem találnak párt maguknak, és esetükben a hiányzó attribútumok NULL értéket vesznek fel. Hasonlóan, a T_1 ⟖ T_2 jobboldali összekapcsolás esetén a természetes összekapcsoláson túl a T_2 tábla azon sorai is megmaradnak, melyek nem találnak párt maguknak, és esetükben a hiányzó attribútumok NULL értéket vesznek fel. A kétoldali T_1 ⟗ T_2 összekapcsolás esetén a lógó sorok mindkét táblából megmaradnak. Ennek következtében baloldali külső összekapcsolás után projekcióval visszanyerhető a baloldali tábla, jobboldali külső összekapcsolás után a jobboldoli tábla, míg teljes külső összekapcsolás után mindkét tábla.

5.3.3. példa

Az 5.3.1 példát követve, a baloldali összekapcsolás megtartja azokat az felhasználókat is, akik még nem szóltak hozzá egy hírfolyamhoz se, ezáltal az eredménytáblából projekcióval visszanyerhetővé válik az összes felhasználó.

T_1 ⟕ T_2

felhasználónév	**utolsó belépés időpontja**	**sorszám**	**tartalom**
pbalazs	2018-10-03 11:10:00	1	Minden rendben.
pbalazs	2018-10-03 11:10:00	2	Én is hozzászólok.
pkardos	2018-10-06 9:45:00	3	Mi újság?
gnemeth	2018-10-15 17:00:00	NULL	NULL
bodnaar	NULL	NULL	NULL

Theta összekapcsolás

A *theta* összekapcsolás (theta join) egy általános feltétel szerinti összekapcsolást jelent. A Descartes-szorzat azon rekordjait tartja meg, melyek egy adott logikai feltételnek megfelelnek. Definíció szerint tehát $T_1 \bowtie_{feltétel} T_2 = \sigma_{feltétel}(T_1 \times T_2)$.

5.3.4. példa

Tegyük fel, hogy egy cégnél a dolgozók években vett munkatapasztalatát tárolják, továbbá azt, hogy legalább hány év munkatapasztalat szükséges egy adott projektben való részvételhez, az alábbi sémájú táblákban:

`DOLGOZÓ`(dolgozókód, név, mukatapasztalat)
`PROJEKT`(projektkód, projektnév, min munkatapasztalat)

Ekkor a `DOLGOZÓ` $\bowtie_{munkatapasztalat \geq min\ munkatapasztalat}$ `PROJEKT` művelettel megadható, hogy mely dolgozó mely projekten dolgozhat.

Kérdések és feladatok

1. Adjon példát olyan T_1 és T_2 táblákra, amelyekre $T_1 \setminus T_2 = T_2 \setminus T_1$!

2. Legyen adott az alábbi két tábla:

DOLGOZÓ

adószám	név	osztálykód	fizetés
101	Kis Béla	1	100000
102	Nagy Katalin	2	200000
103	Kovács Endre	1	175000

OSZTÁLY

osztálykód	osztály neve
1	Pénzügy
2	Munkaügy
3	Műszaki

   Adja meg a két tábla Descartes-szorzatát, természetes összekapcsolását, jobboldali-, baloldali- és kétoldali külső összekapcsolását! Meg tudja-e adni a két tábla metszetét, illetve unióját?

3. Adja meg azt a relációs algebrai kifejezést, mely a fenti két táblából előállít egy olyan táblát, mely a dolgozók nevét, fizetését és osztályának nevét tartalmazza, de csak azokat a dolgozókat tárolja, akiknek a fizetése 150000 forintnál nagyobb.

6. fejezet

Normalizálás

Az előző fejezetben láthattuk, hogyan kapcsolhatók össze a relációs algebra műveleteivel a táblák annak érdekében, hogy a különböző táblákban szereplő, de logikailag összetartozó adatokat együttesen tudjuk kezelni. Felmerülhet a kérdés, hogy mi szükség arra, hogy kisebb táblákban tároljuk az adatainkat, miért nem dolgozunk egy nagy táblával, melyben az összes adat szerepel. A válasz az, hogy ebben az esetben bizonyos adatelmeket többszörözve (redundánsan) kellene tárolnunk, amely az adatbázisműveletek során problémákat okozhat. Ebben a fejezetben először ezeket a felmerülő problémákat ismertetjük, majd bevezetjük a funkcionális függés fogalmát, mellyel az adatelemek között összefüggések vizsgálhatók. Ezután megmutatjuk, hogy a táblákban milyen elvek mentén szüntethető meg fokozatosan a redundancia a dekompozíció és a normálformák segítségével.

6.1. A redundáns adattárolás veszélyei

Nézzük meg, mi történne, ha az alábbi táblában együttesen tárolnánk a felhasználók alapadatait és azt, hogy ki melyik hírfolyamot követi (a korábbiakban ismertetettekhez képest az átláthatóbb ábrázolás végett egyes attribútumokat most elhagyunk).

FÓRUM_KÖVETÉSE

felhasználónév	email	név	hírfolyam azonosító	megnevezés
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	1	Adatbázis kérdések
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	2	PHP hírek
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	4	Milyen gépet vegyek
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter	2	PHP hírek
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter	3	Ki a legjobb tanár
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	1	Adatbázis kérdések
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	2	PHP hírek
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	3	Ki a legjobb tanár
bodnaar	bodnaar@inf.u-szeged.hu	Bodnár Péter	4	Milyen gépet vegyek

Vegyük észre, hogy ebben a sémában a felhasználónév és a hírfolyam azonosító külön-külön nem elegendő egyértelmű azonosításra, a két attribútum

csak együttesen alkot kulcsot.

Ha most szeretnénk tudni, hogy ki milyen hírfolyamot követ, akkor elegendő ehhez az egy táblához fordulnunk, nem szükséges a relációs algebra (esetenként időigényes) kombinációs műveleteit használni. Látható azonban, hogy számos ismétlődő (redundáns) adatot tartalmaz a tábla, ami egyrészt pazarló az adatbázis tárolásának szempontjából, de ennél komolyabb problémák forrása is lehet.

Módosítás esetén: Ha például egy felhasználó email címe megváltozik, akkor ezt minden sorban módosítani kell. Ez időigényes lehet, és ha egy sorban elmarad, akkor egymásnak ellentmondó adatok keletkeznek (ugyanaz a felhasználó különböző email címekkel jelenik meg).

Beszúrás esetén: Ha egy felhasználó elkezd egy újabb hírfolyamot követni, akkor figyelni kell arra, hogy a hírfolyam elnevezése ugyanaz legyen, mint a korábbi esetekben. Ha ez nem így történik, akkor egymásnak ellentmondó adatok keletkeznek (ugyanahhoz a hírfolyam azonosítóhoz több különböző megnevezés fog tartozni). Másik probléma forrása lehet, ha új felhasználót szeretnénk felvenni, aki még nem követ egy fórumot sem. Ekkor a hírfolyam azonosító és a megnevezés mezőkbe NULL értéket kellene írnunk, de a hírfolyam azonosító kulcsban szereplő attribútum, így nem vehet fel NULL értéket.

Törlés esetén: Ha egy hírfolyam összes követőjét töröljük, akkor a hírfolyamhoz tartozó információkat is elveszítjük.

A megoldás értelemszerűen az, hogy a felhasználó és a hírfolyam adatait külön táblákban tároljuk és köztük (mivel N:M típusú kapcsolatban állnak) egy kapcsolótáblát vegyünk fel.

FELHASZNÁLÓ

felhasználónév	**email**	**név**
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor
bodnaar	bodnaar@inf.u-szeged.hu	Bodnár Péter

HÍRFOLYAM

hírfolyam azonosító	**megnevezés**
1	Adatbázis kérdések
2	PHP hírek
3	Ki a legjobb tanár
4	Milyen gépet vegyek

KÖVETI

felhasználónév	**hírfolyam azonosító**
pbalazs	1
pbalazs	2
pbalazs	4
pkardos	2
pkardos	3
gnemeth	1
gnemeth	2
gnemeth	3
bodnaar	4

Látható, hogy ugyanazokat a sémákat kapjuk (eltekintve a kevesebb feltüntetett attribútumtól), mintha az E-K diagram megfelelő részéből indultunk volna ki, és követtük volna a sémába való átírási szabályokat. A helyesen felírt E-K diagram tehát már önmagában segít a redundancia megszüntetésében. De mi van, ha az E-K diagram felírása nem optimális? Vagy ha a táblákat készen kapjuk egy másik alkalmazásból (akár az internetről gyűjtött adatokkal), és nincs is E-K diagramunk? A következőkben formális módszereket vezetünk be arra vonatkozólag, hogy hogyan deríthető fel a redundancia a táblákban és hogyan kell azt a táblák szétbontásával megszüntetni.

6.2. Funkcionális függőség

Az előző fejezetben megadott `FÓRUM_KÖVETÉSE` táblában a redundancia úgy tűnik ki, hogy látjuk, hogy valahányszor két sorban megegyezik a felhasználónév, mindannyiszor ott az email cím és a név is megegyeznek. Továbbá, ha a hírfolyam azonosítója megegyezik két sorban, akkor a megnevezés is. Ezt fogjuk most formálisan megfogalmazni.

Legyen $R(A_1, \ldots, A_n)$ egy relációséma és $P, Q \subseteq \{A_1, \ldots, A_n\}$. Azt mondjuk, hogy P-től *funkcionálisan függ* Q ($P \to Q$), ha bármely R feletti T tábla esetén valahányszor két sor megegyezik P-n, mindannyiszor megegyezik Q-n is, azaz $\forall t_i, t_j \in T \; t_i(P) = t_j(P) \implies t_i(Q) = t_j(Q)$. A $P \to Q$ függést *triviálisnak* nevezzük, ha $Q \subseteq P$, ellenkező esetben *nemtriviálisnak*. A $P \to Q$ függést *teljesen nemtriválisnak* nevezzük, ha $P \cap Q = \emptyset$. A definícióból látszik, hogy a funkcionális függés a táblától független, a sémát jellemző tulajdonság.

6.2.1. példa

A `FÓRUM_KÖVETÉSE` sémában néhány jellemző teljesen nemtriviális funkcionális függőség:

- {felhasználónév} → {email}
- {felhasználónév} → {név}
- {felhasználónév} → {email, név}
- {email} → {felhasználónév}
- {email} → {név}
- {hírfolyam azonosító} → {megnevezés}.

További teljesen nemtriviális funkcionális függőségek például:

- {felhasználónév, email} → {név}
- {felhasználónév, név} → {email}
- {felhasználónév, email} → {név}
- {felhasználónév, hírfolyam azonosító} → {név}.

Nemtriviális funkcionális függésre példa lehet a {felhasználónév, email} → {email, név}. Ez a függés viszont nem teljesen nemtriviális, mert az email attribútum mindkét oldalán megjelenik. Triviális funkcionális függés pedig például a {felhasználónév, email} → {email}.

De vajon hogyan vezethetők le adott függőségekből újabbak? Ehhez az úgynevezett *Armstrong axiómákat* hívjuk segítségül. Belátható, hogy ezek véges sokszori alkalmazásával egy adott függőségi halmazból következő bármely függőség levezethető. Az Armstrong-axiómák az alábbiak:

Reflexivitás: Ha $X \supseteq Y$, akkor $X \to Y$. Valóban, tetszőleges $t_i, t_j \in T$ sorok esetén, ha ezek a sorok X-en megyegyeznek, akkor szükségszerűen annak Y részhalmazán is, azaz $t_i(X) = t_j(X) \implies t_i(Y) = t_j(Y)$.

Augmentivitás (bővítés): Ha $X \to Y$, akkor tetszőleges Z-re $X \cup Z \to Y \cup Z$. Valóban, tegyük fel, hogy a tetszőleges $t_i, t_j \in T$ sorok $X \cup Z$-n megyegyeznek, azaz $t_i(X \cup Z) = t_j(X \cup Z)$. Ekkor értelemszerűen megyegyeznek külön X-en és külön Z-n is, tehát $t_i(X) = t_j(X)$ és $t_i(Z) = t_j(Z)$. Ha $X \to Y$ fennáll, akkor $t_i(X) = t_j(X)$-ből következik, hogy $t_i(Y) = t_j(Y)$. Ez pedig a $t_i(Z) = t_j(Z)$-vel együtt azt jelenti, hogy $t_i(Y \cup Z) = t_j(Y \cup Z)$, ezért $X \cup Z \to Y \cup Z$.

Tranzitivitás: Ha $X \to Y$ és $Y \to Z$, akkor $X \to Z$. Valóban, tetszőleges $t_i, t_j \in T$ sorok esetén, ha ezek a sorok X-en megegyeznek, akkor $X \to Y$ miatt Y-on is, azaz $t_i(X) = t_j(X) \implies t_i(Y) = t_j(Y)$. De akkor $Y \to Z$ miatt $t_i(Z) = t_j(Z)$ is fennáll.

Az Armstrong axiómák segítségével a funkcionális függés további két hasznos tulajdonsága is bizonyítható.

Dekompozíció (szétvágás): Ha $X \to Y \cup Z$, akkor $X \to Y$ és $X \to Z$. Valóban, mivel $Y \cup Z \supseteq Y, Z$, így a reflexivitás miatt $Y \cup Z \to Y$ és $Y \cup Z \to Z$. Innen a tranzitivitás miatt adódik $X \to Y$ és $X \to Z$.

Additivitás (egyesítés): Ha $X \to Y$ és $X \to Z$, akkor $X \to Y \cup Z$. Valóban, az augmentivitás miatt $X \to Y$-ból következik $X \cup X \to Y \cup X$, valamint $X \to Z$-ből következik $X \cup Y \to Z \cup Y$. Ebből pedig a tranzitivitás miatt adódik $X \to Y \cup Z$.

A relációséma és az adattábla fogalma a függőségek figyelembevételével pontosítható: Relációsémának nevezünk egy $R = (A, F)$ párt, ahol $A = \{A_1, \ldots, A_n\}$ attribútumhalmaz, $F = \{f_1, \ldots, f_m\}$ pedig A-n definiált $f_i : P_i \to Q_i$ $(i = 1, \ldots, m)$ alakú függőségek halmaza. Az adattábla az R reláció felett pedig egy olyan $T \subseteq dom(A_1) \times \cdots \times dom(A_n)$ halmaz, amely eleget tesz az F-beli feltételeknek. A továbbiakban maradunk a korábbi $R(A)$ jelölésnél, ha a függőségeket nem kívánjuk hangsúlyozni.

Egy X *attrbiútumhalmaz lezártja* az F függőségi halmaz szerint az $X^+ = \{A_i | X \to A_i\}$ halmaz, ami tehát azon A_i attribútumokból áll, melyekre az $X \to A_i$ függőség F-ből levezethető. Ez a halmaz a következő algoritmus segítségével határozható meg:

1. Legyen $X^{(0)} = \{X\}$. Legyen $i = 0$.
2. Keressünk egy $(P \to Q) \in F$ függőséget úgy, hogy $P \subseteq X^{(i)}$ és $Q \not\subseteq X^{(i)}$. Ha nem találunk ilyet, akkor $X^+ = X^{(i)}$ és VÉGE.
3. Legyen $i = i + 1$ és $X_i = X_i \cup Q$, majd ugorjunk a 2. lépésre.

Mivel az eljárás minden lépésben legalább egy új attribútumot fűz a lezárthoz és A véges, így az algoritmus végés lépés után leáll. Az algoritmus helyességének bizonyításától itt eltekintünk.

6.2.2. példa

Legyen $R(A, F)$ az $A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$ attribútumhalmazzal és az $F = \{\{A_1\} \to \{A_3, A_4\}, \{A_2\} \to \{A_6\}, \{A_3\} \to \{A_5\}, \{A_4, A_5\} \to \{A_7\}\}$ függéshalmazzal. Határozzuk meg az $\{A_1\}^+$ halmazt.

- $X^{(0)} = \{A_1\}$, mely az $\{A_1\} \to \{A_3, A_4\}$ függőség mentén bővíthető.
- $X^{(1)} = \{A_1, A_3, A_4\}$, mely az $\{A_3\} \to \{A_5\}$ függőség mentén bővíthető.
- $X^{(2)} = \{A_1, A_3, A_4, A_5\}$, mely az $\{A_4, A_5\} \to \{A_7\}$ függőség mentén bővíthető.

- $X^{(3)} = \{A_1, A_3, A_4, A_5, A_7\}$ és a halmaz nem bővíthető, azaz $\{A_1\}^+ = \{A_1, A_3, A_4, A_5, A_7\}$.

A szuperkulcs és a funkcionális függés definíciója alapján adódik, hogy egy $K \subseteq A$ attribútumhalmaz akkor és csak akkor szuperkulcs, ha $K \to A$, vagy másként, ha $K^+ = A$. Ez alapján és a fenti algoritmus segítségével már megadhatunk egy eljárást, amellyel meg tudjuk határozni egy séma kulcsát. Legyen kezdetben $K = A$, ami mindig szuperkulcs, majd hagyjunk el K-ból sorra attribútumokat és ellenőrizzük, hogy $K^+ = A$ még teljesül-e.

6.2.3. példa

Legyen adott az $R(A, F)$ séma az $A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$ attribútumhalmazzal és az $F = \{\{A_1\} \to \{A_3, A_4\}, \{A_2\} \to \{A_6\}, \{A_3\} \to \{A_5\}, \{A_4, A_5\} \to \{A_7\}\}$ függéshalmazzal. Legyen kezdetben $K = A$. A 6.2.2 példában láttuk, hogy $\{A_1\}^+ = \{A_1, A_3, A_4, A_5, A_7\}$, így K-ból az A_3, A_4, A_5, A_7 attribútumokat elhagyva a $K = \{A_1, A_2, A_6\}$ még mindig szuperkulcs lesz. Az $\{A_2\} \to \{A_6\}$ függés miatt A_6 is elhagyható ($\{A_2\}^+ = \{A_2, A_6\}$), tehát $K = \{A_1, A_2\}$ is még szuperkulcs és könnyen látható, hogy ez a halmaz már nem szűkíthető úgy, hogy még szuperkulcs maradna, azaz $K = \{A_1, A_2\}$ kulcs.

Az attribútumhalmaz lezártjához hasonlóan egy *függéshalmaz lezártját* is meghatározhatjuk. Egy F függéshalmaz F^+ lezártján az F-ből levezethető összes függést tartalmazó halmazt értjük. Az F^+ egy bázisának nevezzük egy olyan részhalmazát, amelyből F valamennyi függése levezethető. Belátható, hogy $F^+ = \{X \to Y | Y \subseteq X^+\}$. Ez alapján az F^+ halmaz a következő algoritmussal határozható meg.

1. Vegyük az összes lehetséges $X \subseteq A$ részhalmazt és határozzuk meg hozzá annak X^+ lezártját.
2. Minden $Y \subseteq X^+$-ra vegyük fel az $X \to Y$ függést F^+-ba.

6.3. Relációsémák felbontása

Ebben a fejezetben azt mutatjuk be, hogy hogyan bontható fel egy relációséma kisebb sémákra úgy, hogy a redundancia csökkenjen.

Legyen $R(A)$ egy relációséma, és $X, Y \subset A$ úgy, hogy $X \cup Y = A$ és $X \cap Y \neq \emptyset$. Az $R(A)$ séma *felbontása (dekompozíciója)* X és Y szerint egy

$R_1(X)$ és egy $R_2(X)$ sémát jelent. Az R séma feletti T táblát pedig az R_1 feletti $T_1 = \pi_X(T)$ és az $R_2 =$ feletti $T_2 = \pi_Y(T)$ táblákkal helyettesítjük.

Belátható, hogy tetszőleges felbontás esetén $T \subseteq T_1 \bowtie T_2$. Ehhez azt kell megmutatnunk, hogy tetszőleges $t \in T$ sor esetén léteznek olyan $t_1 \in T_1$ és $t_2 \in T_2$ sorok, hogy t_1 és t_2 összekapcsolásával éppen a t sort kapjuk. Ennek azonban éppen megfelel az a $t_1 \in T_1$ sor, amit a $t \in T$ sor π_X projekciójával kapunk, valamint az a $t_2 \in T_2$ sor, amit a $t \in T$ sor π_Y projekciójával kapunk. Ekkor ugyanis $t(X) = t_1(X)$ és $t(Y) = t_2(Y)$, továbbá $X \cap Y \neq \emptyset$ miatt a t_1 és t_2 sorok összekapcsolhatók, éppen a $t \in T$ sort eredményezve.

Egy felbontást *hűségesnek* nevezünk, ha $T \supseteq T_1 \bowtie T_2$ is teljesül, azaz $T = T_1 \bowtie T_2$.

6.3.1. példa

A `FÓRUM_KÖVETÉSE` tábla esetében az

$$X = \{\text{felhasználónév}, \text{email}, \text{név}, \text{hírfolyam azonosító}\},$$

$$Y = \{\text{hírfolyam azonosító}, \text{megnevezés}\}$$

mentén vett felbontás hűséges. Az alábbi táblákat eredményezi, melyek természetes összekapcsolásával valóban éppen a `FÓRUM_KÖVETÉSE` táblát kapjuk:

T_1

felhasználónév	**email**	**név**	**hírfolyam azonosító**
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	1
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	2
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	4
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter	2
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter	3
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	1
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	2
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	3
bodnaar	bodnaar@inf.u-szeged.hu	Bodnár Péter	4

T_2

hírfolyam azonosító	**megnevezés**
1	Adatbázis kérdések
2	PHP hírek
3	Ki a legjobb tanár
4	Milyen gépet vegyek

Ugyanakkor az

$$X = \{\text{felhasználónév}, \text{megnevezés}\}$$

$$Y = \{\text{email}, \text{név}, \text{hírfolyam azonosító}, \text{megnevezés}\}$$

mentén vett felbontás nem hűséges, hiszen az alábbi táblákat eredményezi:

T_1

felhasználónév	megnevezés
pbalazs	Adatbázis kérdések
pbalazs	PHP hírek
pbalazs	Milyen gépet vegyek
pkardos	PHP hírek
pkardos	Ki a legjobb tanár
gnemeth	Adatbázis kérdések
gnemeth	PHP hírek
gnemeth	Ki a legjobb tanár
bodnaar	Milyen gépet vegyek

T_2

email	név	hírfolyam azonosító	megnevezés
pbalazs@inf.u-szeged.hu	Balázs Péter	1	Adatbázis kérdések
pbalazs@inf.u-szeged.hu	Balázs Péter	2	PHP hírek
pbalazs@inf.u-szeged.hu	Balázs Péter	4	Milyen gépet vegyek
pkardos@inf.u-szeged.hu	Kardos Péter	2	PHP hírek
pkardos@inf.u-szeged.hu	Kardos Péter	3	Ki a legjobb tanár
gnemeth@inf.u-szeged.hu	Németh Gábor	1	Adatbázis kérdések
gnemeth@inf.u-szeged.hu	Németh Gábor	2	PHP hírek
gnemeth@inf.u-szeged.hu	Németh Gábor	3	Ki a legjobb tanár
bodnaar@inf.u-szeged.hu	Bodnár Péter	4	Milyen gépet vegyek

Ezek összekapcsolása pedig egy, az eredetinél bővebb táblát eredményez.

$T_1 \bowtie T_2$

felh. név	email	név	hírfolyam azonosító	megnevezés
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	1	Adatbázis kérdések
gnemeth	pbalazs@inf.u-szeged.hu	Balázs Péter	1	Adatbázis kérdések
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	2	PHP hírek
pkardos	pbalazs@inf.u-szeged.hu	Balázs Péter	2	PHP hírek
gnemeth	pbalazs@inf.u-szeged.hu	Balázs Péter	2	PHP hírek
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	4	Milyen gépet vegyek
bodnaar	pbalazs@inf.u-szeged.hu	Balázs Péter	4	Milyen gépet vegyek
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter	2	PHP hírek
pbalazs	pkardos@inf.u-szeged.hu	Kardos Péter	2	PHP hírek
gnemeth	pkardos@inf.u-szeged.hu	Kardos Péter	2	PHP hírek
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter	3	Ki a legjobb tanár
gnemeth	pkardos@inf.u-szeged.hu	Kardos Péter	3	Ki a legjobb tanár
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	1	Adatbázis kérdések
pbalazs	gnemeth@inf.u-szeged.hu	Németh Gábor	1	Adatbázis kérdések
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	2	PHP hírek
pbalazs	gnemeth@inf.u-szeged.hu	Németh Gábor	2	PHP hírek
pkardos	gnemeth@inf.u-szeged.hu	Németh Gábor	2	PHP hírek
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	3	Ki a legjobb tanár
pkardos	gnemeth@inf.u-szeged.hu	Németh Gábor	3	Ki a legjobb tanár
bodnaar	bodnaar@inf.u-szeged.hu	Bodnár Péter	4	Milyen gépet vegyek
pbalazs	bodnaar@inf.u-szeged.hu	Bodnár Péter	4	Milyen gépet vegyek

Ha a felbontás nem hűséges, akkor a kisebb táblák természetes összekapcsolásával általában nem állítható vissza az eredeti tábla. Ilyenkor új, az eredeti táblában nem szereplő (általában értelmetlen) sorok is keletkeznek, ami azt jelenti, hogy információt veszítünk a dekompozíció során. A célunk ezért az, hogy minden esetben hűséges felbontást alkalmazzunk. A következő tétel arra szolgáltat elegendő feltételt, hogy egy felbontás hűséges legyen.

6.3.1. tétel : Heath tétele

Legyen $R(A)$ egy relációséma és $A = B \cup C \cup D$ az A attribútumhalmaz egy diszjunkt felbontása (azaz $B \cap C = \emptyset$, $C \cap D = \emptyset$ és $B \cap D = \emptyset$). Ha $C \to D$, akkor az $R_1(B \cup C)$, $R_2(C \cup D)$ felbontás hűséges.

Bizonyítás. Legyen T egy tetszőleges R feletti tábla és T_1 valamint T_2 a szétbontás során kapott R_1 valamint R_2 feletti táblák. A $T \subseteq T_1 \bowtie T_2$ a korábbiak alapján nyilvánvaló. Azt kell tehát bizonyítanunk, hogy $T_1 \bowtie T_2 \subseteq T$. Legyen $t \in T_1 \bowtie T_2$ egy tetszőleges sor. Ez egy T_1-beli és egy T_2-beli sor összekapcsolásával állt elő, így kell, hogy legyen olyan $t_1 \in T_1$ és olyan $t_2 \in T_2$ sor, hogy $t_1(C) = t_2(C)$. A T_1 és T_2 táblák viszont a T tábla projekciójaként adódtak ($T_1 = \pi_{B \cup C}(T)$ és $T_2 = \pi_{C \cup D}(T)$), tehát létezik egy olyan $u_1 \in T$ sor, melyre $u_1(B \cup C) = t_1(B \cup C)$, továbbá létezik egy olyan $u_2 \in T$ sor, melyre $u_2(C \cup D) = t_2(B \cup C)$. Mivel $t_1(C) = t_2(C)$, így $u_1(C) = u_2(C)$ is fennáll, de akkor a $C \to D$ miatt $u_1(D) = u_2(D)$ is teljesül. Tehát $u_1 = t$, azaz $t \in T$.

6.3.2. példa

A `FÓRUM_KÖVETÉSE` tábla esetében a korábban már vizsgált

$$X = \{\text{felhasználónév, email, név, hírfolyam azonosító}\}$$

$$Y = \{\text{hírfolyam azonosító, megnevezés}\}$$

mentén vett felbontás Heath-tétele szerint hűséges a $B = \{\text{felhasználónév, email, név}\}$, $C = \{\text{hírfolyam azonosító}\}$, $D = \{\text{hírfolyam azonosító, megnevezés}\}$ választással.

Ha a függőségeket is figyelembe vesszük, akkor egy $R(A, F)$ relációséma felbontása X és Y szerint olyan $R_1(X, F_1)$ és $R_2(Y, F_2)$ sémákat jelent, ahol F_1 úgy választandó, hogy F_1^+ az F^+ azon részhalmazával egyezzen meg, amely csak X-beli attribútumokat tartalmaz, F_2 pedig úgy választandó, hogy F_2^+ az F^+ azon részhalmazával egyezzen meg, amely csak Y-beli attribútumokat tartalmaz. Egy felbontást *függőségőrzőnek* nevezünk, ha $F_1 \cup F_2$ az F egy bázisát adják. Belátható, hogy egy hűséges felbontás nem feltétlenül függőségőrző.

6.4. Normálformák

A normálformák segítségével elérhetjük, hogy az adatbázisban a redundanciát lépésről-lépésre szüntessük meg a dekompozíció segítségével úgy, hogy a sémában lévő függőségekre egyre szigorúbb előírásokat teszünk. Először bevezetünk néhány új fogalmat.

Legyen $X, Y \subseteq A$ úgy, hogy $X \to Y$. Azt mondjuk, hogy X-től *teljesen*

függ Y, ha bármely $X' \subset X$ esetén $X' \to Y$ már nem teljesül, azaz X-ből bármely attribútumot elhagyva a függés már nem áll fent. Értelemszerűen, ha K kulcs, akkor a kulcs minimalitása miatt A teljesen függ K-tól.

Egy attribútumot *elsődleges attribútumnak* nevezünk, ha szerepel a relációséma valamely kulcsában, ellenkező esetben *másodlagos attrbibútumnak* nevezzük.

Legyen $X, Z \subseteq A$ úgy, hogy $X \to Z$. Azt mondjuk, hogy X-től *tranzitívan függ* Z, ha létezik olyan $Y \subseteq A$, amelyre $X \to Y$ és $Y \to Z$ úgy, hogy az $Y \to Z$ függés teljesen nemtriviális és X nem függ Y-tól. Ellenkező esetben azt mondjuk, hogy Z *közvetlenül függ* X-től.

Az első három normálforma az alábbi.

1NF: Egy relációsma első normálformában van, ha az attribútumok értéktartománya csak egyszerű adatokból áll.

2NF: Egy relációséma második normálformában van, ha minden másodlagos attribútum teljesen függ bármely kulcstól.

3NF: Egy relációséma harmadik normálformában van, ha minden másodlagos attribútum közvetlenül függ bármely kulcstól, azaz nincs kulcstól vett tranzitív függés.

Mivel már a relációs modell definíciója során kikötöttük, hogy csak elemi attribútumokat szerepeltethetünk (azaz nincsenek halmaz-, lista-, struktúra attribútumok), így a továbbiakban mindig feltételezhetjük, hogy a sémáink 1NF-ben vannak.

A 2NF definíciójából adódik, hogy ha a sémában nincs másodlagos attribútum vagy ha a sémában minden kulcs egyszerű (azaz egy attribútumból áll), akkor a séma 2NF-ben is van.

Ha a séma nincs 2NF-ben, akkor a táblában redundancia léphet fel. Tegyük fel ugyanis, hogy valamely K kulcs $L \subset K$ valódi részhalmazától függ a másodlagos attribútumok egy B részhalmaza (azaz $L \to B$). Ekkor mivel L nem kulcs (mert egy kulcs valódi részhalmaza), így nem is szuperkulcs, azaz lehetnek olyan sorok a táblában, melyek L-en megegyeznek, így az $L \to B$ függés miatt B-n is. Ez pedig a B értékek redundáns tárolását eredményezi.

Ha egy séma nincs 2NF-ben, akkor a 2NF-et sértő függés mentén végzett dekompozícióval elérhető a 2NF. Tegyük fel, hogy van egy olyan K kulcs, melynek egy valódi $L \subset K$ részhalmazától függenek a $B \subset A$ halmaz attribútumai, azaz $L \to B$ (B-t válasszuk úgy, hogy az az összes L-től függő attribútumot tartalmazza). Legyen $C = A \setminus (L \cup B)$. Ekkor a $C \cup L$ és $L \cup B$ mentén vett felbontás Heath tétele szerint hűséges és nyilvánvalóan

megszünteti az $L \to B$ függőséget, hiszen L és B attribútumai külön sémákba kerülnek.

6.4.1. példa

A FÓRUM_KÖVETÉSE sémának két kulcsa van: a {felhasználónév, hírfolyam azonosító} és az {email, hírfolyam azonosító}. Mindkettőnek valódi részhalmaza a {hírfolyam azonosító} és fennáll a {hírfolyam azonosító} → {megnevezés} függés, a séma tehát nincs 2NF-ben. Heath tételét követve dekompozícióval az alábbiakhoz jutunk.

FÓRUM_KÖVETÉSE

felhasználónév	email	név	hírfolyam azonosító
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	1
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	2
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter	4
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter	2
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter	3
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	1
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	2
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor	3
bodnaar	bodnaar@inf.u-szeged.hu	Bodnár Péter	4

HÍRFOLYAM

hírfolyam azonosító	megnevezés
1	Adatbázis kérdések
2	PHP hírek
3	Ki a legjobb tanár
4	Milyen gépet vegyek

Az első séma még mindig nincs 2NF-ben, hiszen a {felhasználónév} is egy kulcs valódi részhalmaza és a {felhasználónév} → {email, név} függés is fennáll, ahol a név másodlagos attribútum. Ismét Heath tétele alapján eljárva adódik:

FELHASZNÁLÓ

felhasználónév	email	név
pbalazs	pbalazs@inf.u-szeged.hu	Balázs Péter
pkardos	pkardos@inf.u-szeged.hu	Kardos Péter
gnemeth	gnemeth@inf.u-szeged.hu	Németh Gábor
bodnaar	bodnaar@inf.u-szeged.hu	Bodnár Péter

KÖVETI

felhasználónév	hírfolyam azonosító
pbalazs	1
pbalazs	2
pbalazs	4
pkardos	2
pkardos	3
gnemeth	1
gnemeth	2
gnemeth	3
bodnaar	4

Így a sémák már 2NF-ben vannak.

A 3NF definíciójából következik, hogy ha egy sémában nincs másodlagos attribútum, akkor a séma 3NF-ben van. Ha azonban egy séma nincs 3NF-ben (de 2NF-ben van), akkor a táblában redundancia léphet fel. Ha ugyanis egy K kulcstól tranzitívan függ a másodlagos attribútumok egy B halmaza,

azaz van olyan $Y \subseteq A$, amelyre $K \to Y$ és $Y \to B$, de K nem függ Y-tól és B üres, akkor Y nem lehet szuperkulcs (mivel K nem függ tőle), ezért a táblában több sor is lehet, amelyek Y-on megegyeznek. Ez az $Y \to B$ függés miatt azt eredményezi, hogy ezek a sorok B-n is megegyeznek, azaz a B értékeket redundánsan tároljuk.

Ha egy séma nincs 3NF-ben, akkor a tranzitív függés mentén végzett dekompozícióval elérhető a 3NF. Tegyük fel, hogy van egy olyan K kulcs, melyre fennáll $K \to Y \to B$. Válasszuk B-t úgy, hogy az az összes Y-tól függő attribútumot tartalmazza és legyen $C = A \setminus (Y \cup B)$. Ekkor a $C \cup Y$ és $Y \cup B$ mentén vett felbontás Heath tétele szerint hűséges és nyilvánvalóan megszünteti az $Y \to B$ függőséget, hiszen Y és B külön sémákba kerülnek.

6.4.2. példa

A 6.4.1 példát folytatva megállapítható, hogy a `HÍRFOLYAM` és a `KÖVETI` sémák már 3NF-ben vannak, ugyanis csak két-két attribútumot tartalmaznak (ráadásul ez utóbbiban nincs is másodlagos attribútum). A `FELHASZNÁLÓ` séma esetében viszont megfigyelhető, hogy fennáll a {felhasználónév} → {email} → {név} függés. Valóban egy email címhez egyértelműen tartozik egy név, aki annak az email címnek a tulajdonosa. Az {email} → {név} függés teljesen nemtriviális. Ennek ellenére nem tranzitív függésről van szó, ugyanis egy email címmel csak egy felhasználó regisztrálhat (általában a fórum alkalmazások nem engedik meg, hogy ugyanazzal az email címmel hozzunk létre több felhasználót), tehát fennáll az {email} → {felhasználónév} függés, ami a tranzitív függés definíciójának ellentmond. Ha most még azt is meggondoljuk, hogy a névtől funkcionálisan nem függ az email cím, hiszen lehetnek azonos nevű, de különböző email című felhasználók, akkor arra jutunk, hogy a sémában nincs tranzitív függés, azaz 3NF-ben van.

6.4.3. példa

Tekintsük az alábbi sémát, amelynek segítségével nyilvántartható, hogy egy cég dolgozói mely projekten dolgoznak. Feltételezzük, hogy egy dolgozó csak egy projekten dolgozhat.

`DOLGOZÓ`(adószám, TAJ szám, dolgozó neve, projektkód, projekt neve)

Mivel egy dolgozó csak egy projekten dolgozhat, ezért a sémában az {adószám} vagy a {TAJ szám} önmagában lehet kulcs. Mi az előbbit választottuk elsődleges kulcsnak. Továbbá mindkét kulcs egyszerű, így a séma 2NF-ben van. Nincs viszont 3NF-ben az {adószám} →

{projektkód} → {projekt neve} tranzitív függés miatt. Ennek következtében, ha ugyanazon a projekten mondjuk 3 dolgozó is dolgozik, akkor a projekt kódját és nevét is háromszor kell tárolnunk. Ezzel szemben, ha felbontjuk a sémát, az alábbi módon, akkor az adott projekt nevét csak egyszer kell tárolnunk.

`DOLGOZÓ`(adószám, TAJ szám, dolgozó neve, projektkód)
`PROJEKT`(projektkód, projekt neve).

A 3NF és a 2NF közötti összefüggést a következő állítás mondja ki: Ha egy relációséma 3NF-ben van, akkor 2NF-ben is van. Ennek igazolásához indirekt módon tegyük fel, hogy egy adott séma 3NF-ben van, de nincs 2NF-ben, azaz van a sémában olyan A_i másodlagos attribútum, amely nem teljesen függ valamely K kulcstól. Ez azt jelenti, hogy van olyan $L \subset K$, amelyre $L \to A_i$. Ekkor azonban $L \subset K$ miatt $K \to L$ is fennál, továbbá mivel K kulcs, tehát minimális, így L-től nem függ K, valamint $A_i \notin L$, mert A_i másodlagos attribútum. Így a $K \to L \to A_i$ függés tranzitív.

A gyakorlatban sok esetben elegendő 3NF-ig elmenni a normalizálás során. Néha azonban előfordulhat, hogy a redundancia tovább csökkenthető. Ezzel kapcsolatban két további normálformát ismerünk meg. Ezek közül az első a Boyce-Codd normálforma.

BCNF: Egy relációséma Boyce-Codd normálformában van, ha bármely nemtriviális $L \to B$ függés esetén L szuperkulcs.

Ha a séma nincs BCNF-ben, akkor a táblában redundancia léphet fel. Ugyanis, ha $L \to B$ és L nem szuperkulcs, akkor a táblában több olyan sor lehet, amelyek L-en megegyeznek, és így a függőség miatt B-n is. Ez azonban a B-értékek redundáns tárolását eredményezi. Ekkor a BCNF-et sértő $L \to B$ függés (ahol B az összes L-től függő attribútumot tartalmazza) mentén bontjuk fel a sémát a normalizálás során. Legyen $C = A \setminus (L \cup B)$. Ekkor a $C \cup L$ és $L \cup B$ mentén vett felbontás Heath tétele szerint hűséges és nyilvánvalóan megszünteti az $L \to B$ függőséget, hiszen L és B attribútumai külön sémákba kerülnek.

6.4.4. példa

Tekintsük újból 6.4.3 példa sémáját, amellyel azt tartjuk nyilván, hogy egy cég dolgozói mely projekten dolgoznak, ezúttal egy tömörebb formában: nem tároljuk se a dolgozók se a projektek nevét. Feltesszük ellenben, hogy most egy dolgozó több projekten is dolgozhat.

`DOLGOZÓ`(adószám, TAJ szám, projektkód)

A sémában az {adószám, projektkód} vagy a {TAJ szám, projektkód} lehet kulcs, mi itt az elsőt választottuk elsődleges kulcsnak. Nincs tehát másodlagos attribútum, ezért a séma 3NF-ben van. Fennáll, viszont az {adószám} → {TAJ szám} függés, pedig az {adószám} nem szuperkulcs (mellékesen a {TAJ szám} → {adószám} függés is teljesül), így a séma nincs BCNF-ben. Valóban, ha egy dolgozó mondjuk 3 projekten is dolgozik, akkor az adószámát és a TAJ számát is 3 sorban tároljuk. Ezzel szemben, ha felbontjuk a sémát, az alábbi módon, akkor az adott dolgozó TAJ számát csak egyszer kell tárolnunk.

`DOLGOZÓ`(adószám, TAJ szám)
`DOLGOZIK`(adószám, projektkód).

A BCNF és a 3NF közötti összefüggést a következő állítás mondja ki: Ha egy relációséma BCNF-ben van, akkor 3NF-ben is van. Ennek bizonyításához indirekt módon tegyük fel hogy egy séma BCNF-ben van, de ennek ellenére nincs 3NF-ben, azaz van olyan $K \to L \to B$ tranzitív függés, ahol K kulcs. A tranzitív függés definiciója alapján ekkor a $L \to B$ függés nemtriviális, valamint L-től nem függ K. Ez utóbbi viszont azt jelenti, hogy L nem szuperkulcs, márpedig a BCNF definíciója szerint nemtriviális függés bal oldalán csak szuperkulcs állhatna, így ellentmondásra jutottunk.

Míg az előző normálformák a funkcionális függés definícióján alapultak, a negyedik normálforma ennek a fogalomnak az általánosításával jut el a redundancia további csökkentéséhez. Legyenek $K, L \subseteq A$, valamint $M = A \setminus \{K \cup L\}$. Azt mondjuk, hogy K-tól *többértékűen függ* L ($K \twoheadrightarrow L$), ha bármely R feletti T táblában valahányszor két sor megegyezik K-n, mindannyiszor a két sor kombinációja is megjelenik T-ben. Formálisan, ha léteznek olyan $t_i, t_j \in T$ sorok, melyekre $t_i(K) = t_j(K)$, akkor van olyan $t \in T$ sor, amelyre $t(K) = t_i(K) = t_j(K)$ továbbá $t(L) = t_i(L)$ és $t(M) = t_j(M)$. A definíció szimmetriájából adódik továbbá, hogy ha $K \twoheadrightarrow L$, akkor $K \twoheadrightarrow M$ is teljesül.

A többértékű függés valóban általánosítása a funkcionális függésnek. Ehhez csak azt kell belátni, hogy ha $K \to L$, akkor $K \twoheadrightarrow L$. Ha ha léteznek olyan $t_i, t_j \in T$ sorok, melyekre $t_i(K) = t_j(K)$, akkor $K \to L$ miatt $t_i(L) = t_j(L)$. Ekkor viszont a $t = t_j$ sorra $t(K) = t_i(K) = t_j(K)$ továbbá $t(L) = t_i(L)$ és $t(M) = t_j(M)$, azaz valóban $K \twoheadrightarrow L$.

6.4.5. példa

Tekintsük az alábbi táblát, amely azt tartalmazza, hogy melyik pizzéria melyik városrészbe milyen pizzát tud kiszállítani.

PIZZASZÁLLÍTÁS

pizzéria kódja	városrész kódja	pizza típusa
P1	V1	sajtos
P1	V1	sonkás
P1	V1	szalámis
P1	V2	sajtos
P1	V2	sonkás
P1	V2	szalámis
P2	V1	sajtos
P2	V1	gombás
P2	V3	sajtos
P2	V3	gombás

Mivel mindhárom attribútum része a kulcsnak, így ezúttal nincs nemtriviális függésünk, így a séma már BCNF-ben van. Világos viszont, hogy ha egy pizzéria tud egy városrészbe szállítani, akkor oda bármilyen típusú pizzát tud szállítani. Azaz, ha a táblázat tartalmazza például a (P1, V2) és a (P1, sonkás) párokat, akkor tartalmazza a (P1, V2, sonkás) hármast is. Ez pedig pontosan azt jelenti, hogy a pizzéria kódjától többértékűen függ a pizza típusa, valamint szimmetria okokból a városrész kódja is.

A $K \twoheadrightarrow L$ többértékű függést *nemtriviálisnak* nevezzük, ha $K \cap L = \emptyset$ és $K \cup L \neq A$. Ezek után a negyedik normálforma definíciója a következőképpen adható meg.

4NF: Egy relációséma negyedik normálformában van, ha minden nemtriviális $K \twoheadrightarrow L$ függés esetén K szuperkulcs.

Ha egy séma nincs 4NF-ben, akkor a tábla redundanciát tartalmazhat. Amennyiben ugyanis $K \twoheadrightarrow L$ és K nem szuperkulcs, akkor a táblában több sor is megegyezhet K-n és ezekben a sorokban az L és M értékek redundánsan szerepelnek, ahogy ezt a fenti példában is láthattuk.

Kérdés, hogy hogyan tudunk ilyenkor dekompozíciót végezni. Eddig mindig Heath tétele szerint jártunk el, amely a funkcionális függések ismeretében arra szolgáltat elegendő feltételt, hogy egy felbontás hűséges legyen. Most azonban egy ennél általánosabb fogalommal, a többértékű függéssel dolgozunk. Szerencsére Heath tételéhez hasonlóan ilyenkor is adható feltétel arra, hogy mikor hűséges egy felbontás. Sőt a következő tétel nemcsak szükséges, de egyben elegendő feltételt is ad.

6.4.1. tétel : Fagin tétele

Legyen adott egy $R(A)$ relációséma és legyen $A = B \cup C \cup D$ az A attribútumhalmaz egy diszjunkt felbontása (azaz $B \cap C = \emptyset$, $C \cap D = \emptyset$ és $B \cap D = \emptyset$). Az $R_1(B \cup C)$, $R_2(C \cup D)$ felbontás akkor és csak akkor hűséges, ha $C \twoheadrightarrow D$.

Bizonyítás. Tegyük fel először is, hogy a felbontás hűséges, azaz $T = T1 \bowtie T2$. Ekkor a többértékű függés és a természetes összekapcsolás definíciói alapján $t_1(B \cup C) \in T_1$ és $t_2(C \cup D) \in T_2$, ezért szükségképpen $t \in T$. A másik irány bizonyításához tegyük fel, hogy $C \twoheadrightarrow D$ és legyen $t_1 \in T_1$ és $t_2 \in T_2$ úgy, hogy $t_1(C) = t_2(C)$. Ekkor a t_1 és t_2 egyesítésével előálló rekord a függőség miatt szerepel T-ben, vagyis $T_1 \bowtie T_2 \subseteq T$. Ugyanakkor $T \subseteq T_1 \bowtie T_2$ nyilvánvaló, így $T = T_1 \bowtie T_2$, tehát a felbontás valóban hűséges.

6.4.6. példa

Fagin tételét alkalmazva, most már fel tudjuk bontani a 6.4.5 példa PIZZASZÁLLÍTÁS tábláját a {pizzéria kódja} $\twoheadrightarrow$ {városrész kódja} függés mentén. Eredményül két táblát kapunk, amelyek már 4NF-ben vannak.

KISZÁLLÍTÁS

pizzéria kódja	városrész kódja
P1	V1
P1	V2
P2	V1
P2	V3

PIZZÁK

pizzéria kódja	pizza típusa
P1	sajtos
P1	sonkás
P1	szalámis
P2	sajtos
P2	gombás

Végezetül belátjuk, hogy ha egy séma 4NF-ben van, akkor már BCNF-ben is. Tegyük fel ugyanis, hogy a sémában van egy $K \to L$ funkcionális függés. Ha $K \cup L = A$, akkor K az összes maradék attribútumot (az L-belieket) meghatározza, így K szuperkulcs. Ha pedig $K \cup L \subset A$, akkor az $L' = L \setminus K$ választásával $K \to L'$ (hiszen $L' \subseteq L$ és $K \to L$). Akkor viszont $K \twoheadrightarrow L'$ is fennáll és ez a függés nemtriviális. Ezért K újból csak szuperkulcs lehet.

Kérdések és feladatok

1. Hány teljesen nemtriviális, hány nemtriviális, és hány triviális funkcionális függés van a **Fórum** példában? Melyek ezek?

2. Hogyan alakulna a **Fórum** adatbázisséma normalizálásának menete a 2NF-ről 3NF-re lépés során, ha ugyanazzal az email címmel több felhasználót is létre lehetne hozni?

3. Mutassa meg indirekt bizonyítással, hogy egy relációséma akkor és csak akkor van 3NF-ben, ha bármely nemtriviális $L \to A_i$ függés esetén L szuperkulcs vagy A_i elsődleges attribútum. Hogyan adódik ebből egyszerűen, hogy minden BCNF-ben lévő séma egyúttal 3NF-ben is van?

4. Legyen adott az $R(A, F)$ séma az $A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$ attribútumhalmazzal és az $F = \{\{A_1\} \to \{A_3, A_4\}, \{A_2\} \to \{A_6\}, \{A_3\} \to \{A_5\}, \{A_4, A_5\} \to \{A_7\}, \{A_7\} \to \{A_1\}\}$ függéshalmazzal. Határozza meg az $\{A_3\}^+$, az $\{A_5\}^+$ és az $\{A_3, A_4\}^+$ halmazt.

5. Legyen adott az $R(\underline{A_1, A_2}, A_3, A_4, A_5, A_6)$ séma úgy, hogy $\{A_1\} \to \{A_3, A_4\}$, $\{A_2\} \to \{A_6\}$ és $\{A_3\} \to \{A_5\}$. Hozza a sémát Heath tétele alapján 2NF-re, majd győződjön meg arról, hogy az így kapott séma már teljesíti a 3NF feltételeit! Az eredeti sémában van egy tranzitív függés. Melyik ez? Mi történne, ha a normalizálást ennek a tranzitív függésnek a megszüntetésével kezdenénk (ugyancsak Heath tétele alapján)?

7. fejezet

Összefoglalás

Ezzel elérkeztünk az adatbázis tervezési folyamatának végéhez. Először E-K diagram segítségével modelleztük az adatbázist, majd abból relációs sémákat írtunk fel. Ezt követte a sémák normalizálása. A relációs modell definíciójából adódóan feltételeztük, hogy a sémák mindig teljesítik az 1NF követelményeit. A továbbiakban a redundancia csökkentése érdekében megvizsgáltuk, hogy milyen funkcionális függések vannak a sémákban. Ezeket aztán lépésről-lépésre szüntettük meg a sémák felbontásával a 2NF, 3NF, BCNF és 4NF feltételeinek megfelelően. Láttuk továbbá, hogy a 4NF teljesüléséből, következik a BCNF, annak teljesüléséből a 3NF, ez utóbbi teljesüléséből pedig a 2NF, tehát a normálformák rendre valóban egyre szigorúbb feltételeket szabnak a sémákra. Végeredményül olyan sémákat kaptunk, melyek már a legtöbb esetben nem tartalmaznak zavaró redundanciát. Így elkezdhetjük az adatbázis számítógépes létrehozását. A jegyzet következő részében erről lesz szó.

II. rész

Az adatbáziskezelő rendszerek és az SQL nyelv

8. fejezet

Az SQL nyelv

Az SQL nyelvet az IBM-nél fejlesztette ki Donald D. Chamberlin és Raymond F. Boyce, akkor még SEQUEL (*Structured English Query Languge*) rövidítéssel az 1970-es években. Később a nyelv nevét SQL-re rövidítették, mivel a SEQUEL már egy másik cég bejegyzet márkaneveként szerepelt [6], ettől függetlenül a „sequel" kifejezés és kiejtés még mindig használatos.

1986 óta szerepel az ANSI és ISO szabványokban az SQL nyelv, és azóta több új verzióját tették közzé (1989, 1992, 1996, 1999, 2003, 2006, 2008, 2011, 2016). A szabvány megújulásának oka az, hogy a technológia fejlődésével az adatbázis-kezelő rendszereket fejlesztő cégek, újabb és újabb kulcsszavakat és funkciókat alkotnak meg. Ezt a szabvány csak némi késéssel tudja követni.

Ebben részben áttekintjük az SQL nyelv legfontosabb utasításait. Az Olvasó megtanulja a legfontosabb adatbázisműveletek (relációs adatbázissémák és adatok létrehozása, módosítása, törlése) SQL utasításait. Megismeri az adatok lekérdezésére vonatkozó utasításokat és összesítő függvényeket különböző bonyolultsági szinteken. Javasoljuk, hogy az Olvasó a példákat saját számítógépen vagy a gyakorlati tantermekben próbálja ki, valamint hajtsa végre a feladatokat, hogy elsajátítsa az itt említett ismereteket és jártasságot szerezzen az SQL utasítások írásában. A fejezetekben nem térünk ki az SQL nyelv minden részletére és utasítására, de az itt leírtak elegendőek az önálló munkavégzéshez, valamint a további ismeretek önálló megkereséséhez és feldolgozásához.

9. fejezet

Az SQL nyelv alapjai

A SQL (*Structured Query Language*) egy lekérdező nyelv. Ennélfogva arra szolgál, hogy adatokat kezeljünk vele, azokat be tudjuk szúrni adatbázisokba, tudjuk törölni és módosítani, és természetesen ki tudjuk nyerni azokat. Az SQL nem algoritmikus nyelv, ezért nem találhatóak meg benne például a ciklusszervező és feltételes vezérlési szerkezetek. Az SQL nyelv nem használ változókat sem. A lekérdezések kimenetei eredménytáblaként jönnek létre, ezek mezőire tudunk hivatkozni, de nem tudjuk azokat változókban tárolni. Minderre kiegészítő nyelvek (mint például Oracle esetében a PL/SQL), programozási nyelvek kiegészítő függvénykönyvtárai, vagy a beágyazott SQL nyújtanak lehetőségeket. Az SQL nyelvet leggyakrabban a relációs adatbázisokhoz használják, de újabb változatai tartalmaznak objektum-relációs utasításokat is. A tananyagunk a relációs adatbáziskezeléssel foglalkozik, ezért csupán az itt használt legfontosabb utasításokra és nyelvi elemekre szorítkozunk.

9.1. Az SQL nyelv részeinek felosztása

Az SQL nyelv elemeit az alábbi két fő részre oszthatjuk:

Adat-definíciós nyelv (*Data Definition Language - DDL*)
Ide tartoznak az adatbázisok, a sémák, a típusok definíciós utasításai, mint például:

- `CREATE DATABASE`
- `CREATE TABLE`
- `ALTER TABLE`
- `DROP TABLE`

- `CREATE TRIGGER`

Adat-manipulációs nyelv (*Data Manipulation Language - DML*)
Ide tartoznak az adat beszúró, módosító és törlő utasítások és még a lekérdező utasítások is:

- `INSERT INTO`
- `UPDATE`
- `DELETE FROM`
- `SELECT`

Egyes irodalmak különválasztják a lekérdező utasításokat az adatmódosító utasításoktól, így azok hármas felosztást javasolnak:

Adat-definíciós nyelv (*Data Definition Language - DDL*)
Ide tartoznak az adatbázisok, a sémák, a típusok definíciós utasításai, mint az előző csoportosításban.

Adat-manipulációs nyelv (*Data Manipulation Language - DML*)
Ide tartoznak az adat beszúró, módosító és törlő utasítások, a lekérdezések (`SELECT`) kivételével.

Adat-lekérdező nyelv (*Data Query Language - DQL*)
Ide tartozik a lekérdező (`SELECT`) utasítás.

9.2. Szintaktikai elemek

Kis- és nagybetűk: A kis- és nagybetűk az SQL nyelv szintaktikája szerint egyenértékűek. Megjegyezzük azonban, hogy a táblák vagy mezők nevében kis- és nagybetűket egyes az egyes adatbáziskezelő rendszerek megkülönböztethetik.

Utasítások: Az SQL utasításokat mindig a pontosvessző karakter zárja. Az „új sor” karakter nem befolyásolja az utasítás végét. Fontos azt is megjegyeznünk, hogy az egyes adatbáziskezelő rendszereknél az utasítások eltérhetnek az SQL szabványtól, valamint, hogy az adatbázis-kezelő rendszerek utasításkészlete bővebb. Hogyan lehet mégis eligazodnunk, és mi az amit érdemes megtanulnunk? Nos, a szabványos SQL utasításokat a népszerű, elterjedtebb rendszerek jórészt követik, legfeljebb saját utasításkészlettel is kiegészítik, így a gyakori szabványos SQL utasítások - amelyeket ebben a tanyagban áttekintünk - jó kiindulópontként szolgálnak. Egy-egy hibajelzés esetén azonban érdemes megnézni

az adott rendszer referencia kézikönyvét, hogy van-e valamilyen szintaktikai vagy kulcsszóbeli eltérés a szabványtól.

Alias nevek, másodnevek: A táblákat és oszlopokat az SQL utasításokban az `AS` kulcsszóval átnevezhetjük. Ez az átnevezés csupán az adott utasításban fog élni.

Szövegkonstansok: A szövegkonstansokat aposztróf (' ') jelek között adjuk meg.

Relációjelek: SQL-ben az alábbi relációjelek használatosak

=	egyenlőség
!=, <>	nem egyenlő
<= vagy >=	kisebb egyenlő, nagyobb egyenlő

Speciális relációs kifejezések:

x `IS NULL`**:** Igaz, ha az x oszlop értéke `NULL`.

x `BETWEEN` a `AND` b**:** Igaz, ha $a \leq x \leq b$.

x `IN` **halmaz:** Igaz, ha x eleme a halmaznak.
Például: `varos IN('Szeged', 'Pécs', 'Budapest')`

x **relációjel** `ALL` **halmaz:** Igaz, ha a relációjel a halmaz minden elemére teljesül. Például: `ev < ALL (1988, 1992, 1996, 2000)`

x **relációjel** `ANY` **halmaz:** Igaz, ha a relációjel a halmaz legalább egy elemére teljesül. Például `ev = ANY(1988, 1992, 1996, 2000)`

`EXISTS` **halmaz:** Igaz, ha a halmaz nem üres. Például `EXISTS (SELECT * FROM felhasznalok)`

x `LIKE` **minta:** Igaz, ha x illeszkedik a mintára. Például `lakcim LIKE '%Szeged,%'`, ahol a százalékjel tetszőleges sztring helyettesítésre szolgál, vagyis azokra a lakcímekre lesz igaz, amelyekben szerepel a "Szeged," karaktersorozat.

A fentiek esetében a reláció tagadására a `NOT` kulcsszó is használható.

Dátum és idő: A dátumot a legtöbb rendszer `'YYYY-MM-DD'` formában kezeli, ahol `YYYY` az év négy számjegye, `MM` a hónap két számjegye és `DD` a nap két számjegye. Az időpontot `'hh:mm:ss'`, formában szokás megadni, ahol `hh` az órát, `mm` a percet, `ss` a másodpercet jelöli. Egyes rendszerek ennél kisebb egységeket is tudnak kezelni. Erre most nem térünk ki.

Háromértékű logika: `TRUE`, `FALSE`, `NULL`

Sztringek konkatenációja: ||, +

9.3. Adattípusok

Ahogy a legtöbb programozási nyelvnél, az SQL-nél is különböző kategóriákba soroljuk az adattípusokat.

9.3.1. Numerikus adattípusok:

`TINYINT(`*számjegyek száma*`):` -128 és 127 közötti egész számokat, vagy az `UNSIGNED` kulcsszóval 0 és 255 közötti számokat jelöl.

`SMALLINT(`*számjegyek száma*`):` -32768 és 32767 közötti egész számokat, vagy az `UNSIGNED` kulcsszóval 0 és 65535 közötti egész számokat tudunk megadni.

`MEDIUMINT(`*számjegyek száma)*`:` -8388608 és 8388607 közötti egész számokat vagy az `UNSIGNED` kulcsszóval 16777215-ig tudunk pozitív egész számokat megadni.

`INT(`*számjegyek száma*`):` -214748648 és 214748647 közötti egész számokat vagy az `UNSIGNED` kulcsszóval 4294967295-ig pozitív egész számokat tudunk megadni.

`BIGINT(`*számjegyek száma*`):` -9223372036854775808 és 9223372036854775807 közötti egész számokat, valamint az `UNSIGNED` kulcsszó használatával 0 és 18446744073709551615 közötti pozitív egész számokat tudunk megadni.

`FLOAT(`*számjegyek száma, tizedesjegyek száma*`):` lebegőpontos számokat jelöl ez az adattípus.

`DOUBLE(`*számjegyek száma, tizedesjegyek száma*`):` a `FLOAT`-hoz képest több biten tárolt lebegőpontos számokat jelöl ez az adattípus.

`DECIMAL(`*számjegyek száma, tizedesjegyek száma*`):` a `DOUBLE` típusú lebegőpontos számot sztringként tárolja.

9.3.2. Szöveges, karakteres adattípusok:

`CHAR(n)`: fix hosszúságú, n karakterből álló szöveg. Adatbázis-kezelő rendszertől függően általában legfeljebb 255 karaktert tárol, de például az Oracle esetében ez a maximális korlát 2000 bájt.

`VARCHAR(n)`: legfeljebb n hosszúságú karaktersorozat. A maximális hossz MySQL esetében 65533 karakter.

`TEXT`: hosszabb szöveget tároló adattípus. A tárolható szöveg maximális karakterszáma az adatbázis-kezelő rendszerektől függ (pl. a MySQL esetében TEXT típus legfeljebb 65 535 karaktert képes tárolni, a Transact-SQL esetében 2 147 483 647 karakter).

9.3.3. Dátumot és időpontot tároló adattípusok:

`DATE`: dátumot tároló adattípus.

`DATETIME`: dátumot és időpontot (órát, percet, másodpercet) tároló adattípus.

`TIMESTAMP`: egy időbélyeget (év, hónap, nap, óra, perc, másodperc) tárol egy adott kezdeti időponttól számított másodpercek számával. A kezdeti időpont értéke rendszerfüggő.

`TIME`: órát, percet és másodpercet tároló adattípus. Egyes rendszereknél a másodperc töredékét is képes számolni, milliszekundum vagy nanoszekundum pontossággal.

9.3.4. Egyéb adattípusok:

`BLOB` **(Binary Large Object):** nagy méretű bináris objektum. A tárolható objektum mérete rendszerfüggő, de jellemzően 2GB.

`BOOLEAN`: igaz/hamis értéket tároló adattípus. Gyakran egy számjegyű egész számként van megvalósítva.

Kérdések és feladatok

1. Soroljon fel 5 adat-definíciós nyelvbe tartozó utasítást!

2. Soroljon fel 3 adat-manipulációs nyelvbe tartozó utasítást!

3. Milyen SQL kulcsszavakkal lehet vizsgálni a halmazba tartozási relációkat?

4. Mire szolgálnak az alias nevek? Hol használhatjuk őket?

5. Soroljon fel 3 adattípust, amellyel dátum és/vagy időadatot lehet reprezentálni SQL-ben!

10. fejezet

Relációs adatbázis sémák létrehozása, módosítása és törlése

A következőkben bemutatjuk a relációs adatbázissémák létrehozására, módosítására és törlésére szolgáló SQL utasításokat. Ismertetjük a leggyakoribb oszlop- és táblafeltételek, valamint külső kulcs kapcsolatok megadásának módját, valamint a további feltételek és megszorítások lehetőségeit is.

10.1. Relációs adatbázissémák létrehozása

A relációs adatbázisséma létrehozására `CREATE TABLE` utasítás használható. A neve arra utal, hogy táblát hozunk létre, de meg kell különböztetnünk a tábla és a séma definícióját. A *séma* a tábla szerkezetét írja le, magában foglalja a mezők típusát, azok oszlopfeltételeit és megszorításait, valamint a táblafeltételeket is. A *tábla* pedig az adatrekordok halmazát jelöli. A `CREATE TABLE` utasítás csak a sémát hozza létre.

A `CREATE TABLE` utasítás

```
CREATE TABLE táblanév (
  mező1 típus [oszlopfeltételek],
  mező2 típus [oszlopfeltételek],
  ...,
  mezőN típus [oszlopfeltételek],
  [táblafeltételek]);
```

10.1.1. Oszlopfeltételek

Az *oszlopfeltételek* olyan feltételek, amelyek csak az adott mezőre vonatkoznak. Az egyes oszlopokra más-más feltétel írható elő. Ilyen feltételek a következők:

`PRIMARY KEY`: Az elsődleges kulcs.

`UNIQUE`: Kulcs, minden érték egyszer fordulhat elő az oszlopban.

`NOT NULL`: Az értéke nem lehet NULL, kötelező megadni.

`REFERENCES` *T(oszlop)*: A *T* tábla *oszlop*ára hivatkozó külső kulcs.

`DEFAULT` *tartalom*: Az oszlop alapértelmezett értéke *tartalom* lesz.

10.1.2. Táblafeltételek

A *táblafeltételek* olyan feltételek, amelyek táblaszinten érvényesek. Itt meg tudunk mondani olyan feltételeket is, amelyek egy-egy oszlopra vonatkoznak, de azoknak a feltételeknek, amelyek egy-egy oszlop tekintetében nem adhatók meg, kötelező jelleggel táblafeltételnek kell lenniük. Például, ha az elsődleges kulcs egy attribútumból áll, akkor ezt oszlopfeltételként és táblafeltételként is megadhatjuk, azonban, ha az elsődleges kulcs több attribútumból áll, akkor kötelező jelleggel táblafeltételként kell feltüntetni.

`PRIMARY KEY(`*oszloplista*`)`: Az elsődleges kulcs.

`UNIQUE(`*oszloplista*`)`: Kulcs, minden érték egyszer fordulhat elő az oszlopban.

`FOREIGN KEY (`*oszloplista*`) REFERENCES` *T(oszloplista)*: A *T* tábla *oszloplistájára* hivatkozó külső kulcs.

10.1.1. példa

Nézzük meg, hogyan lehet megadni elsődleges kulcsot oszlopfeltételként és táblafeltételként. Tekintsük a `szemely`(szemelyiSzam, nev, szuletesiDatum) sémát!

Elsődleges kulcs megadása oszlopfeltételként

```
CREATE TABLE szemely(szemelyiSzam INTEGER
   PRIMARY KEY, nev VARCHAR(255),
   szuletesiDatum DATE);
```

Elsődleges kulcs megadása táblafeltételként

```
CREATE TABLE szemely(szemelyiSzam INTEGER, nev
    VARCHAR(255), szuletesiDatum DATE, PRIMARY
    KEY(szemelyiSzam);
```

Abban az esetben, ha a kulcs több attribútumból áll, csak táblafeltételként adhatunk mindkettőre vonatkozó feltételeket. Ha például a vonat indulásokat tároljuk egy adatbázisban, akkor a `vonat`(indul, honnan, hova, erkezik) sémát a következőképpen hozhatjuk létre:

Több attribútum alkotta elsődleges kulcs megadása

```
CREATE TABLE vonat(indul DATETIME, honnan
    VARCHAR(70), hova VARCHAR(70), erkezeik
    DATETIME, PRIMARY KEY(indul, honnan, hova) );
```

10.1.3. Külső kulcs feltételek és szabályok

A külső kulcsok egy másik séma elsődleges kulcsára hivatkoznak. Ezek képeznek kapcsolatot a táblák között. Egy relációsémához meghatározhatjuk, hogy mely attribútumok lesznek külső kulcsok és azok melyik tábla mely oszlopára vagy oszlopaira mutassanak. Az integritás megőrzése szempontjából a külső kulcsokhoz meghatározhatjuk azt is, hogy „hogyan” viselkedjenek a hivatkozott kulcs törlése vagy módosítása esetén. Erre szolgálnak az `ON DELETE` és `ON UPDATE` szabályok.

Az `ON DELETE` utasítás azt szabályozza, hogy mi történjen a külső kulccsal törlés esetén:

`ON DELETE RESTRICT`: Amennyiben a törlendő rekord kulcsára van hivatkozó külső kulcs, úgy a megtiltjuk a rekord törlését.

`ON DELETE SET NULL`: A törlendő rekord kulcsára hivatkozó külső kulcs értékét `NULL`-ra állítjuk.

`ON DELETE NO ACTION`: A törlendő rekord kulcsára hivatkozó külső kulcs értéke nem változik.

`ON DELETE CASCADE`: A törlendő rekord kulcsára hivatkozó külső kulcsú rekordok is törlődnek.

Vizsgáljuk meg az eseteket! Melyik lesz igazán jó választás? Az `ON DELETE RESTRICT` jó választás lehet, ha nem szeretnénk, hogy olyan rekordot töröljünk, amire van külső kulcs hivatkozás. Az `ON DELETE SET NULL` akkor lehet jó választás, ha azt szeretnénk, hogy a hivatkozott rekord törlésekor hivatkozó rekordnál a külső kulcs NULL értéket kapjon, vagyis jelöljük azt, hogy az a kapcsolat megszűnt, nem hivatkozik semmire. A **Fórum** példa (lásd 10.1.2. példa) esetében ezt választhatjuk az üzenetek és a felhasználók közötti külső kulcs esetében, hiszen, ha egy felhasználót törlünk, még nem kell az ő összes üzenetét törölnünk, lehet, hogy hasznos információt tartalmaz. Az `ON DELETE CASCADE` jó választás, ha azt szeretnénk, hogy a hivatkozott rekordok mind törlődjenek. A **Fórum** példára vonatkozóan a hírfolyam törlése kapcsán jogosan várjuk el a rendszertől, hogy ha törlünk egy hírfolyamot, akkor az ahhoz tartozó üzenetet törölje. Az `ON DELETE NO ACTION` feltétel a gyakorlatban nem igazán használt megoldás, ugyanis maradnának nem létező külső kulcs értékek.

Az `ON UPDATE` utasítás azt szabályozza, hogy mi történjen a külső kulccsal adatmódosítás esetén:

`ON UPDATE RESTRICT:` Amennyiben a módosítandó rekord kulcsára van hivatkozó külső kulcs, úgy a megtiltjuk a rekord kulcsának módosítását.

`ON UPDATE SET NULL:` A módosítandó rekord kulcsára hivatkozó külső kulcs értékét `NULL`-ra állítjuk.

`ON UPDATE NO ACTION:` A módosítandó rekord kulcsára hivatkozó külső kulcs értéke nem változik.

`ON UPDATE CASCADE:` A módosítandó rekord kulcsára hivatkozó külső kulcsú rekordok is az új értékre változnak.

Vizsgáljuk meg most is, hogy mi lehet jó választás! A módosítás esetén az lenne jó, ha a külső kulcs értéke is módosulna, vagyis itt az `ON UPDATE CASCADE` tűnik jó választásnak. A többi változat csak speciális esetekben lenne jól használható.

10.1.2. példa

Készítsünk egy **Fórum** adatbázist! A fórumba csak bejelentkezett felhasználók írhatnak üzenetet. Az üzenetek hírfolyamokhoz kötődnek. A hírfolyamok néhány kulcsszóval is rendelkezdnek. Az egyes felhasználók követhetik a hírfolyamokat. Tekintsük az alábbi relációsémákat! Hozzuk őket létre SQL-ben! A típusokat nem írtuk elő, ezeket magunk választjuk meg a létrehozáskor.

FELHASZNÁLÓ(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
ÜZENET(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
HÍRFOLYAM(azonosító, megnevezés)
KULCSSZAVAK(*hírfolyam azonosító*, kulcsszó)
KÖVETI(*hírfolyam azonosító*, *felhasználónév*)

Felhasználó tábla

```
CREATE TABLE felhasznalo (
felhasznalonev VARCHAR(20) PRIMARY KEY NOT NULL,
jelszo CHAR(40) NOT NULL,
email VARCHAR(100) NOT NULL,
vezeteknev VARCHAR(40) DEFAULT NULL,
keresztnev VARCHAR(40) DEFAULT NULL,
utolsoBelepesIdopontja TIMESTAMP DEFAULT NULL);
```

Hírfolyam tábla

```
CREATE TABLE hirfolyam (
azonosito BIGINT PRIMARY KEY NOT NULL,
megnevezes VARCHAR(60) UNIQUE NOT NULL);
```

Üzenet tábla

```
CREATE TABLE uzenet (
sorszam BIGINT UNSIGNED PRIMARY KEY,
tartalom TEXT NOT NULL,
mikor TIMESTAMP NOT NULL,
felhasznalonev VARCHAR(20)
REFERENCES felhasznalo(felhasznalonev)
ON DELETE SET NULL
ON UPDATE CASCADE,
hirfolyamAzonosito BIGINT
REFERENCES hirfolyam(azonosito) NOT NULL
ON DELETE CASCADE
ON UPDATE CASCADE);
```

Kulcsszavak tábla

```
CREATE TABLE kulcsszavak (
hirfolyamAzonosito BIGINT NOT NULL
REFERENCES hirfolyam(azonosito)
ON DELETE CASCADE
ON UPDATE CASCADE,
kulcsszo VARCHAR(40) NOT NULL,
PRIMARY KEY (hirfolyamAzonosito, kulcsszo));
```

Követi tábla

```
CREATE TABLE koveti (
hirfolyamAzonosito BIGINT NOT NULL
REFERENCES hirfolyam(azonosito)
ON DELETE CASCADE
ON UPDATE CASCADE,
felasznalonev VARCHAR(20) NOT NULL
REFERENCES felhasznalo(felhasznalonev)
ON DELETE CASCADE
ON UPDATE CASCADE,
PRIMARY KEY (hirfolyamAzonosito,
   felasznalonev));
```

Tekintsünk most egy másik példát is, amelyben nem üzeneteket, hanem eseményeket tartunk nyilván!

10.1.3. példa

Készítsünk egy **Programkalauz** adatbázist! Az adatbázisban a különböző kultúrális és egyéb programokat és eseményeket tartjuk nyilván. Eltároljuk a helyszíneket is, egy program csak egy helyszínen lehet. Feljegyezzük továbbá a műfajokat is, egy programnak több műfaja lehet, ezeket szabad szövegként tároljuk, tehát nem kötünk ki külön műfaji kategóriákat!
Tekintsük az alábbi relációsémákat! Hozzuk őket létre SQL-ben! A típusokat nem írtuk elő, ezeket magunk választjuk meg a létrehozáskor.

`PROGRAMOK`(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)

HELYEK(helyazonosító, város, cím, hely neve)
MŰFAJ(*programazonosító*, műfajmegnevezés)

Programok tábla

```
CREATE TABLE programok (
  programazonosito INTEGER PRIMARY KEY,
  cim VARCHAR(60) NOT NULL,
  leiras TEXT,
  mikortol DATE,
  meddig DATE,
  kapcsolat VARCHAR(60),
  helyazonosito INTEGER DEFAULT NULL REFERENCES
     helyek(helyazonosito),
  artol INTEGER,
  arig INTEGER );
```

Helyek tábla

```
CREATE TABLE helyek (
  helyazonosito INTEGER PRIMARY KEY,
  varos VARCHAR(50) NOT NULL,
  cim VARCHAR(100) NOT NULL,
  helyNeve VARCHAR(50) );
```

Műfaj tábla

```
CREATE TABLE mufaj(
  programazonosito INTEGER NOT NULL,
  mufajmegnevezes VARCHAR(40) NOT NULL,
  PRIMARY KEY (programazonosito,
     mufajmegnevezes),
  FOREIGN KEY programazonosito REFERENCES
     program(programazonosito) );
```

10.2. Relációs adatbázissémák módosítása

A relációs adatbázissémák módosítása SQL-ben az `ALTER TABLE` utasítással történik. Fontos felhívni az Olvasó figyelmét, hogy ez a parancs nem az

adat, hanem a relációs adatbázisséma szerkezetének módosítását végzi. A paranccsal hozzá tudunk adni a sémához új oszlopot, meg tudjuk változtatni valamely oszlop típusát, hosszát, vagy éppen a kulcsokat, és külső kulcs kapcsolatokat, valamint törölni is tudunk oszlopot.

Új oszlop hozzáadása

```
ALTER TABLE táblanév ADD (uj_oszlop TIPUS
   [oszlopfeltételek] );
```

Oszlop módosítása

```
ALTER TABLE táblanév MODIFY (meglevo_oszlop TIPUS
   [oszlopfeltételek] );
```

A módosítás során nem kell tudnunk, hogy mi volt az oszlop típusa, vagy milyen feltételek voltak rá kiadva.

Oszlop törlése

```
ALTER TABLE táblanév DROP (oszlop);
```

10.2.1. példa

Tekintsük a **Fórum** adatbázist!

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
`ÜZENET`(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
`HÍRFOLYAM`(azonosító, megnevezés)
`KULCSSZAVAK`(*hírfolyam azonosító*, kulcsszó)
`KÖVETI`(*hírfolyam azonosító*, *felhasználónév*)

Módosítsuk a **Fórum** (10.1.2. példa) adatbázis tábláinak szerkezetét! A jelenlegi `felhasznalok` táblában az e-mail címre nem kötöttük ki, hogy egyedi legyen.

Oszlop módosítása

```
ALTER TABLE felhasznalok MODIFY (email
    VARCHAR(100) UNIQUE NOT NULL);
```

Az `uzenet` táblából töröljük, majd írjuk vissza a `mikor` attribútumot!

Oszlop törlése

```
ALTER TABLE uzenet DROP (mikor);
```

Oszlop hozzáadása

```
ALTER TABLE uzenet ADD (mikor TIMESTAMP NOT
    NULL);
```

10.3. Relációs adatbázissémák törlése

A relációs adatbázissémák törlése a `DROP TABLE` utasítással történik.

A `DROP TABLE` utasítás szintaxisa

```
DROP TABLE táblanév;
```

10.4. Táblákra és attribútumokra vonatkozó megszorítások

A megszorítások elsődleges feladata az, hogy megelőzzük az adatbeviteli hibákat, illetve elkerüljük a hiányzó adatot a kötelezően kitöltendő mezőkből. Fontos szem előtt tartanunk, hogy amennyiben már a tábla létrehozásakor meg tudjuk fogalmazni ezeket a feltételeket, építsük be őket az utasításba, ne kelljen később a tábla szerkezetén módosítani (ugyanis körülményes lehet, ha már fel van töltve adattal)!

`NOT NULL`: A mező kötelezően kitöltendő és nem lehet az értéke NULL.

Példa a `NOT NULL` használatára

```
CREATE TABLE felhasznalok (
  felhasznalonev VARCHAR(20) PRIMARY KEY,
  email VARCHAR(30) UNIQUE NOT NULL,
  nev VARCHAR(50) NOT NULL,
  jelszo CHAR(40) NOT NULL );
```

`CHECK` **(feltétel):** Egy ellenőrző feltétel arra vonatkozóan, hogy milyen értéket vehet fel az oszlop. Használható oszlopra és táblára vonatkozóan is.

Példa a `CHECK` használatára

```
CREATE TABLE fogadoorak (
  oktatokod CHAR(8) PRIMARY KEY,
  nap CHAR(10) NOT NULL,
  kezdodik INT NOT NULL CHECK(kezdodik >= 8),
  vege INT NOT NULL CHECK(vege) <= 20),
  CHECK(kezdodik < vege));
```

`DOMAIN`**:** Értéktartomány egy oszlop értékeire vonatkozóan.

Értéktartomány létrehozása

```
CREATE DOMAIN név típus [DEFAULT érték][CHECK
   (feltétel)];
```

Értéktartomány használata

```
CREATE DOMAIN napTipus CHAR
  CHECK (VALUE IN ('hetfő', 'kedd', 'szerda',
     'csütörtök', 'péntek'));
CREATE TABLE fogadoorak (
  oktatoKod CHAR(8) PRIMARY KEY,
  nap napTipus NOT NULL,
  kezdodik INT NOT NULL CHECK(kezdodik >= 8),
  vege INT NOT NULL CHECK(vege) <= 20),
  CHECK(kezdodik < vege));
```

10.4.1. példa

A `CHECK` feltétellel külső kulcs kapcsolat részlegesen ellenőrizhető.

Külső kapcsolat ellenőrzése a sémában

```
CREATE TABLE terem (
  teremszam CHAR(20) PRIMARY KEY,
  teremnev VARCHAR(20));
CREATE TABLE fogadoorak (
  oktatokod CHAR(8) PRIMARY KEY,
  nap CHAR(10) NOT NULL,
  teremszam CHAR(20) CHECK(teremszam IN (SELECT
     teremszam FROM terem)),
  kezdodik INT NOT NULL CHECK(kezdodik >= 8),
  vege INT NOT NULL CHECK(vege) <= 20),
  CHECK(kezdodik < vege));
```

Ebben az esetben a `CHECK` feltétel biztosítja, hogy csak olyan teremszámra tudunk hivatkozni, amely szerepel a `terem` táblában, de ha a teremszámot a `terem` táblában megváltoztatjuk, akkor már előfordulhat olyan anomália, hogy olyan teremszámra hivatkozunk, amely nem létezik.

10.5. Általános megszorítások

Az általános megszorítások több sémára, általában a teljes adatbázisra vonatkozhatnak. A feltételben szereplő táblák bármelyikének módosításakor a feltétel ellenőrzésre kerül.

Általános megszorítások definiálása

```
CREATE ASSERTION név CHECK(feltétel);
```

10.5.1. példa

Tekintsük a `Fogadoorak(oktatokod, nap, teremszam, kezdodik, vege)` relációs adatbázissémát a fogadóórák nyilvántartására. Készítsünk olyan megszorítást, hogy ugyanabban a teremben egyszerre csak egyvalaki tarthasson fogadóórát! A megszorításban egy lekérdezést használunk, amelyben megszámoljuk azokat a fogadóórákat, amelyek

ugyanabban a teremben vannak és időben részben vagy teljesen átfedőek. Megjegyezzük, hogy itt az `AS` kulcsszóval két külön másodnévvel jelöljük el ugyanazt a táblát, így olyan, mintha ugyanaz a táblánk két külön példányban szerepelne a lekérdezésben.

A `teremüközésNincs` feltétel

```
CREATE ASSERTION teremutkozesNincs
  CHECK (  (SELECT COUNT(*)
    FROM fogadoora AS f1, fogadoora AS f2
    WHERE f1.teremszam = f2.teremszam AND (
      f1.kezdodik BETWEEN f2.kezdodik AND f2.vege
         OR
      f2.kezdodik BETWEEN f1.kezdodik AND
         f1.vege) <= 1)
  );
```

Kérdések és feladatok

1. Hozza létre a `Fogadoorak (oktatokod, nap, teremszam, kezdodik, vege)` táblát az alábbi megszorításokkal:

   - `oktatokod`: 10 hosszú sztring, elsődleges kulcs
   - `nap`: 10 hosszú sztring, megadása kötelező
   - `teremszam`: négyjegyű egész szám, megadása kötelező
   - `kezdodik`: TIME típusú érték, megadása kötelező
   - `vege`: TIME típusú érték, megadása kötelező, értéke nagyobb, mint a `kezdodik` értéke

2. A 10.1.3 példában látott **Programkalauz** adatbázis `programok` táblájában alakítsa át úgy a mezőket, hogy értéküket kötelező legyen megadni!

   `PROGRAMOK`(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)
   `HELYEK`(helyazonosító, város, cím, hely neve)
   `MŰFAJ`(*programazonosító*, műfajmegnevezés)

3. A 10.1.3 példában látott **Programkalauz** adatbázis `mufaj` táblájában a `programazonosito` külső kulcs. Írjon olyan SQL utasítást, amellyel beállítja, hogy a hivatkozott kulcsérték (`program.programazonosito`) módosítása esetén a külső kulcs hivatkozások automatikusan módosuljanak, törlés esetén pedig automatikusan törlődjenek!

4. Hozzon létre olyan általános megszorítást, amely garantálja, hogy a 10.1.3 példában látott **Programkalauz** adatbázisra vonatkozóan egy program kezdési időpontja (`mikortol`) kisebb legyen vége (`meddig`) időpontnál!

5. Állítsa be, hogy a 10.1.3 példa `programok` táblájában az `artol` és `arig` attribútumok alapértelmezett értéke NULL legyen!

11. fejezet

Adatok beszúrása, módosítása és törlése

Miután tudjuk, hogyan hozzunk létre táblákat, ebben a fejezetben bemutatjuk az adatok aktualizálásának (beszúrásának, módosításának törlésének) módját. Az Olvasó megtanulja ebből a fejezetből az aktualizáló műveletek szintaxisát. Hangsúlyt fektetünk arra is, hogy a módosító és törlő műveletek kapcsán egy átgondolatlan feltétel vagy a feltételek elhagyása milyen problémákat okozhat az adatbázisban. Célunk ezzel az, hogy az Olvasó e tananyag útmutatásával is szem előtt tartsa az átgondolt adatmódosításokat és törléseket.

11.1. Rekordok beszúrása táblákba

A rekordok táblába történő beszúrása az `INSERT INTO` utasítással történik, melynek két változata használható.

Az `INSERT INTO` utasítás

```
INSERT INTO táblanév (oszloplista) VALUES
   (értéklista);

INSERT INTO táblanév VALUES (értéklista);
```

A két változat használata attól függ, hogy a rekord beszúrásakor minden adatot meg kívánunk-e adni. Az első változat esetén a táblanév után zárójelben, vesszővel elválasztva soroljuk fel azokat az oszlopokat, amelyekhez értéket akarunk megadni. Itt a sémától független sorrendben sorolhatjuk fel az oszlopok neveit és nem kötelező minden oszlopot megadni (például ahol

van alapértelmezett érték, és nem akarunk vagy nem tudunk azon változtani, azt nem kell felsorolnunk). Az értéklista esetében viszont az értékeket ezen oszloplistának megfelelően, vesszővel elválasztva kell megadnunk. Miért ne tudnánk megadni minden értéket egy rekord felvitelénél?

11.1.1. példa

Tekintsük a **Fórum** adatbázist!

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
`ÜZENET`(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
`HÍRFOLYAM`(azonosító, megnevezés)
`KULCSSZAVAK`(*hírfolyam azonosító*, kulcsszó)
`KÖVETI`(*hírfolyam azonosító*, *felhasználónév*)

Tegyük fel, hogy **Fórum** weboldalára új felhasználóként szeretnénk regisztrálni. A regisztrációs felületeken általában nem kell megadnunk minden adatot, csupán a felhasználónevet, a jelszót, és az e-mail címet. A többi (személyes) adatot csak a bejelentkezés után kell kitölteni, vagyis az már egy rekord mezőinek módosításával és nem egy új rekord beszúrásával jár. Az az SQL utasítás, amely létrehozza a felhasználót a **Fórum** adatbázisba a következő:

Új felhasználó létrehozása a `FELHASZNALOK` táblába

```
INSERT INTO felhasznalok (felhasznalonev,
   jelszo, email) VALUES ('gnemeth', 'jelszo',
   'gnemeth@inf.u-szeged.hu');
```

Az `INSERT INTO` utasítás második változatánál az értéklistát kötelező jelleggel, a sémának megfelelően kell megadnunk. Bizonyos rendszerek megengedik azt, hogy azoknál az oszlopoknál, amelyeknél van alapértelmezett érték, ne adjunk meg értéket, de a „helyét” meg kell az adnunk értékek vesszővel elválasztott felsorolásánál.

11.1.2. példa

Tekintsük ismét a **Fórum** adatbázist!

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)

ÜZENET(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
HÍRFOLYAM(azonosító, megnevezés)
KULCSSZAVAK(*hírfolyam azonosító*, kulcsszó)
KÖVETI(*hírfolyam azonosító*, *felhasználónév*)

Szúrjunk be egy új rekordot a KOVETI táblába!

Új rekord felvétele az összes mező megadásával

```
INSERT INTO koveti VALUES (23456, 'gnemeth');
```

11.2. Rekordok módosítása

A rekordok módosítása az UPDATE utasítással történik. Az UPDATE kulcsszó után megadjuk a táblanevet, majd a SET kulcsszó után felsoroljuk azokat az oszlopokat, amelyekhez új értéket rendelünk. Az új érték nem csak konstans, hanem egy kifejezés is lehet. A módosítás csak azokat a rekordokat fogja érinteni, amelyekre a WHERE záradékban megadott feletétel teljesül. Nagyon fontos – és ezzel kapcsolatban nagyon körültekintően kell bánni – hogy ha nem adunk meg feltételt, akkor a módosítás a tábla minden sorára lefut, vagyis nem csak a kívánt adatokat írjuk felül! A szögletes zárójelben lévő részek opcionálisak, azokat nem kötelező megadni az utasításban.

Az UPDATE utasítás

```
UPDATE táblanév
  SET oszlop=kifejezés [, oszlop2=kifejezés2, ...]
  [WHERE feltétel];
```

11.2.1. példa

Az előző alfejezetben szó volt arról (lásd 11.1.1. példa), hogy egy regisztrációs oldal esetében a regisztrációnál csak a felhasználónevet, a jelszót és az e-mail címet kérjük be és létrehozzuk a felhasználót. Hogyan töltjük ki a többi adatát? Az első sikeres bejelentkezésnél megjelenik egy űrlap a felhasználó előtt, hogy töltse ki a személyes adatait. Ekkor viszont már a felhasználói rekord megvan az adatbázisban, nem

kell még egyet létrehozni, csupán módosítani a következő utasítással.

Felhasználó (személyes) adatainak megadása

```
UPDATE felhasznalok
  SET vezeteknev='Németh', keresztnev='Gábor'
  WHERE felhasznalonev = 'gnemeth';
```

Megjegyezzük, hogy a felhasználónév adatokat a 11.1.1 példa alapján írtuk be. Elegendő a `WHERE` záradékban a felhasználónév értékét megadni feltételként, hiszen az elsődleges kulcs.
A módosítás előtt a táblázat tartalma:

FELHASZNALOK

felh.nev	jelszo	email	vezeteknev	keresztnev	u._belepes_datuma
...	...	...	...	...	...
gnemeth	jelszo	gnemeth@inf.u-szeged.hu	NULL	NULL	NULL
...	...	...	...	...	...

A módosítás utáni értékek pedig a következők:

FELHASZNALOK

felh.nev	jelszo	email	vezeteknev	keresztnev	u._belepes_datuma
...	...	...	...	...	...
gnemeth	jelszo	gnemeth@inf.u-szeged.hu	Németh	Gábor	2018-11-18 10:00:00
...	...	...	...	...	...

11.2.2. példa

Most nézzük meg, hogy a **Fórum** adatbázis `FELHASZNALOK` táblájában hogyan jegyezzük be az utolsó belépés időpontját! Ezt az utasítást minden sikeres bejelentkezéskor le kell futtatni (természetesen a bejelentkezési felhasználónévvel, amely most *gnemeth*)!

Utolsó bejelentkezés időpontjának frissítése

```
UPDATE felhasznalok SET
   utolso_belepes_idopontja=CURRENT_TIMESTAMP
   WHERE felhasznalonev='gnemeth';
```

Az aktuális rendszeridő lekérdezésére az egyes adatbáziskezelő rendszerekben más-más kulcsszó illetve függvény szolgál. Ebben a példában a `CURRENT_TIMESTAMP` kulcsszót használtuk, mert ez több adatbáziskezelő rendszerben megtalálható.
A módosítás előtti érték a következő:

FELHASZNALOK

felh.nev	jelszo	email	vezeteknev	keresztnev	u._belepes_datuma
...	...	...	...	...	...
gnemeth	jelszo	gnemeth@inf.u-szeged.hu	Németh	Gábor	2018-11-18 10:00:00
...	...	...	...	...	...

A módosítás utáni érték pedig a következő:

FELHASZNALOK

felh.nev	jelszo	email	vezeteknev	keresztnev	u._belepes_datuma
...	...	...	...	...	...
gnemeth	jelszo	gnemeth@inf.u-szeged.hu	Németh	Gábor	2018-11-18 16:25:00
...	...	...	...	...	...

11.3. Rekordok törlése

A rekordokat a `DELETE FROM` utasítással tudjuk törölni. Az utasítás után csak a táblanevet kell megadni, illetve opcionálisan egy `WHERE` záradékban meg lehet adni a feltételt. Fontos, hogy amennyiben nem adunk meg `WHERE` záradékot, a törlés a teljes tábla tartalomra vonatkozik, vagyis minden rekord ki lesz törölve a táblából (maga a séma megmarad). Amennyiben meg van adva a `WHERE` záradékban egy feltétel, csak azok a sorok törlődnek, amelyek eleget tesznek a feltételnek.

A `DELETE FROM` utasítás

```
DELETE FROM táblanév [WHERE feltétel];
```

11.3.1. példa

Tekintsük most is a **Fórum** példát! Tegyük fel, hogy az aktuális felhasználó (legyen most is *gnemeth*) leiratkozik az 5382 számú hírfolyamról. Töröljük a bejegyzést a táblából!

gnemeth **leiratkozása az 5382-es hírfolyamról**

```
DELETE FROM koveti WHERE hirfolyam_azonosito =
   5382 AND felhasznalonev = 'gnemeth';
```

Ha ezután kilistázzuk a `koveti` tábla rekordjait, akkor a törölt rekordot már nem találjuk a listában.

11.3.2. példa

Töröljük a **Fórum** adatbázisból az összes olyan felhasználót, aki több, mint egy éve nem lépett be.

Régi felhasználók törlése

```
DELETE FROM felhasznalok WHERE
   YEAR(CURRENT_DATE) -
   YEAR(utolso_belepes_idopontja) >= 1;
```

A `CURRENT_TIMESTAMP`-hez hasonlóan a `CURRENT_DATE` kulcsszó a több nagyobb adatbáziskezelő rendszerben létezik és az aktuális rendszer-dátumot adja vissza. A `YEAR()` függvény a paraméterként megadott dátum év értékével tér vissza.

Kérdések és feladatok

1. Írjon SQL utasítást, amely létrehozza a 362-es számú hírfolyamot „SQL utasítások” megnevezéssel!
2. Írjon SQL utasítást, amely a **Programkalauz** adatbázisba beszúrja a 2018. december 31-én 20:00-24:00 tartó „Városi szilveszteri party” rekordot a programok közé! A rendezvény honlapja: „www.pecs.hu” a kapcsolat mezőhöz pedig a „szilveszter@pecs.hu” értéket kell rendelni. A rendezvény ingyenes.
3. Írjon SQL utasítást, amely a **Programkalauz** adatbázisból törli az összes elmúlt eseményt!
4. Írjon SQL utasítást, amely **Fórum** adatbázis `FELHASZNÁLÓK` táblájában az összes felhasználó nevét nagybetűssé alakítja. Használja az `UPPER()` függvényt a nagybetűssé alakításhoz, amelynek egyetlen paramétere a nagybetűssé alakítandó karaktersorozat!
5. Írjon SQL utasítást, amely a **Fórum** adatbázisban hozzárendeli a „Szilveszter” szót az 542-es sorszámú hírfolyamhoz!

12. fejezet

Lekérdezések SQL-ben

Már láttuk a korábbi fejezetben, hogy hogyan tudunk egy adatbázisba adatokat felvinni, módosítani, valamint azt is, hogy hogyan tudjuk azokat törölni. Ebben a fejezetben arra látunk példákat, hogy hogyan lehet az adatbázisból adatokat kiolvasni. Habár ezek az utasítások nem módosítják az adatokat, mégis gondosan kell összeállítani azokat, mivel egy hibásan leírt lekérdező utasítás nem a megfelelő információt fogja szolgáltatni. Megjegyezzük azt is, hogy a lekérdezések bizonyos tekintetben a relációs algebrában megismert műveletek implementációi.

12.1. A SELECT utasítás

Az adatok lekérdezése a `SELECT` utasítással történik. Ennek legegyszerűbb formája a következő:

A SELECT utasítás egyszerűbb formája

```
SELECT oszloplista FROM tábla;
```

Megjegyezzük, hogy az `oszloplista` helyére *-ot is írhatunk. Ha az oszloplista szerepel a `SELECT` utasítás után, akkor ezen oszloplistának megfelelő adatokat fogjuk kilistázni a táblából. Ha pedig * szerepel, akkor a `tábla` minden oszlopa ki lesz listázva a séma szerkezetének megfelelően.

12.1.1. példa

Listázzuk ki a **Fórum** adatbázisból az `UZENET` tábla tartalmát!

Az UZENET tábla tartalmának listázása

```
SELECT * FROM uzenet;
```

Az eredmény ehhez hasonló lesz:

UZENET

sorszám	tartalom	mikor	felhasznalonev	hirfolyam_azonosito
1	Hello mindenkinek!	2018-10-03 11:10:00	gnemeth	1
2	Sziasztok!	2018-10-06 09:45:00	pbalazs	1
3	Sziasztok!	2018-10-07 10:01:00	pkardos	1
4	Tudja-e valaki, hogy ...	2018-10-15 17:00.00	pkardos	1
5	Szerintem itt nézd meg ...	2018-10-16 14:32:00	bodnaar	1

A SELECT utasítás teljes szintaxisa a következő, ahol az opcionális kulcsszavakat szögletes zárójellel jelöltük:

A SELECT utasítás

```
SELECT [DISTINCT] oszloplista
FROM táblalista
[WHERE feltétel]
[GROUP BY oszloplista]
[HAVING csoportfeltétel]
[ORDER BY oszloplista [DESC]];
```

A SELECT utasításban az egyes kulcsszavak és záradékok a következő műveleteket jelentik:

`SELECT` **oszloplista:** itt a projekció műveletét (vagyis oszlopok kiválasztását) hajtjuk végre az `oszloplistában` megadott oszlopok szerint.

`DISTINCT`**:** ez a kulcsszó az eredménytáblában ismétlődő sorok közül csak egyet-egyet ír ki.

`FROM` **táblalista:** Descartes-szorzat készítése a `táblalistában` felsorolt táblákból.

`WHERE` **feltétel:** szelekció a `feltétel` szerint.

`GROUP BY` **oszloplista:** csoportosítás az `oszloplistában` szereplő oszlopok értékei szerint. Több oszlop esetében érték $n-$esket kell figyelembe venni, nem csak különálló oszlopértékeket.

`HAVING` **csoportfeltétel:** a csoportosítás után a csoportok rekordjaira vonatkozó szelekció.

`ORDER BY` **oszloplista:** az `oszloplistában` szereplő oszlopok adatainak rendezése növekvő (`ASC`) vagy csökkenő (`DESC`) sorrendben. Az alapértelmezett rendezés a növekvő sorrend.

A SELECT utasítás műveleteinek sorrendje eltér a szintaxisban megadottaktól. A műveleteket az alábbi sorrendben kell figyelembe venni:

1. `FROM` - Descartes-szorzat készítése a felsorolt táblákból.
2. `WHERE` - A Descartes-szorzatból a megadott feltétel szerint sorokat választunk ki.
3. `GROUP BY` - Csoportosítás.
4. `HAVING` - Csoport szelekció.
5. `SELECT [DISTINCT]` - Projekció.
6. `ORDER BY` - Rendezés.

Amennyiben a lekérdezés csoportosítást tartalmaz, a `SELECT` után csak olyan oszlopnév lehet, amely szerepel a `GROUP BY` oszloplistájában, a csoportosítás alapján értéke egyértelmű, vagy összesítő függvényben szerepel.

12.1.2. példa

Tekintsük a **Fórum** adatbázist!

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
`ÜZENET`(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
`HÍRFOLYAM`(azonosító, megnevezés)
`KULCSSZAVAK`(*hírfolyam azonosító*, kulcsszó)
`KÖVETI`(*hírfolyam azonosító*, *felhasználónév*)

Listázzuk ki az üzenetek tartalmát, írjuk ki minden üzenethez a beküldés dátumát illetve a felhasználó nevét is! Rendezzük az üzeneteket a beküldési dátumuk (`uzenet.mikor` attribútum) szerint csökkenő sorrendbe!

Üzenetek listázása

```
SELECT felhasznalo.fehasznalonev,
   felhasznalo.vezeteknev,
   felhasznalo.keresztnev, uzenet.tartalom,
   uzenet.mikor
FROM felhasznalo, uzenet
WHERE felhasznalo.felhasznalonev =
   uzenet.felhasznalonev
ORDER BY uzenet.mikor DESC;
```

Az eredmény ehhez hasonló lesz:

UZENET

sorszám	tartalom	mikor	felhasznalonev	hirfolyam_azonosito
1	Hello mindenkinek!	2018-10-03 11:10:00	gnemeth	1
2	Sziasztok!	2018-10-06 09:45:00	pbalazs	1
3	Sziasztok!	2018-10-07 10:01:00	pkardos	1
4	Tudja-e valaki, hogy ...	2018-10-15 17:00.00	pkardos	1
5	Szerintem itt nézd meg ...	2018-10-16 14:32:00	bodnaar	1

FELHASZNALO

felhasznalonev	jelszo	vezeteknev	keresztnev	utolso_belepes_idopontja
pbalazs	Bp1234	Balázs	Péter	2019-07-03 12:10:00
gnemeth	Ng5678	Németh	Gábor	2018-11-23 15:25:00
pkardos	Kp4567	Kardos	Péter	2019-02-24 09:52:00
bodnaar	Bp9876	Bodnár	Péter	2019-03-16 18:30:00
vargalg	Vl6543	Varga	László Gábor	2019-02-10 15:00:00

EREDMÉNY (A *felhasznalo* táblanevet itt f-fel rövidítjük, mert hosszú lenne.)

f.felhasznalonev	f.vezeteknev	f.keresztnev	uzenet.tartalom	uzenet.mikor
bodnaar	Bodnár	Péter	Szerintem itt nézd meg ...	2018-10-16 14:32:00
pkardos	Kardos	Péter	Tudja-e valaki, hogy ...	2018-10-15 17:00.00
pkardos	Kardos	Péter	Sziasztok!	2018-10-07 10:01:00
pbalazs	Balázs	Péter	Sziasztok!	2018-10-06 09:45:00
gnemeth	Németh	Gábor	Hello mindenkinek!	2018-10-03 11:10:00

12.2. Összesítő függvények

Az összesítő függvényeket általában a csoportosítással (`GROUP BY`) együtt szoktuk használni, de enélkül is használhatóak. Leginkább a `SELECT` utáni oszloplistában szokás megjeleníteni, de a `WHERE` és `HAVING` záradék feltételeiben is használható. Az összesítő függvényeket tartalmazó eredményoszlopokat az `AS` kulcsszóval jelölhetjük egy másodnéven, azonban fontos látni azt is, hogy ezek a másodnevek, – mivel a `SELECT` utasítás után szerepelnek, – csak a projekció végrehajtásakor lépnek érvénybe, vagyis nem feltételezhetjük, hogy az adatbáziskezelő rendszer figyelembe veszi őket már a `WHERE` és `HAVING` záradékokban.

`MIN(OSZLOP)`: A paraméterként megadott oszlopban lévő értékek minimumát adja vissza.

`MAX(oszlop)`: A paraméterként megadott oszlopban lévő értékek maximumát adja vissza.

`AVG(oszlop)`: A paraméterként megadott oszlopban lévő értékek átlagát adja vissza.

`SUM(oszlop)`: A paraméterként megadott oszlopban lévő értékek összegét adja vissza.

`COUNT([DISTINCT] oszlop)`: A lekérdezés eredményében szereplő rekordok darabszámát adja vissza. Ha a `DISTINCT` kulcsszó meg van adva, akkor az ismétlődő értékeket figyelmen kívül hagyja, vagyis csak a különböző értékeket fogja összeszámolni.

12.2.1. példa

Tekintsük ismét a **Fórum** adatbázist! Számoljuk meg, hány felhasználó van a táblában!

Felhasználók számának lekérdezése

```
SELECT COUNT(*) AS felhasznalokSzama FROM
   felhasznalok;
```

Nézzünk egy olyan lekérdezést, amelyben csoportosítással használjuk a `COUNT()` összesítő függvényt!

12.2.2. példa

Határozuk meg, hogy a **Fórum** adatbázisban hány üzenet szerepel hírfolyamonként!

`FELHASZNÁLÓ`(<u>felhasználónév</u>, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
`ÜZENET`(<u>sorszám</u>, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
`HÍRFOLYAM`(<u>azonosító</u>, megnevezés)
`KULCSSZAVAK`(<u>*hírfolyam azonosító*, kulcsszó</u>)
`KÖVETI`(<u>*hírfolyam azonosító*, *felhasználónév*</u>)

Üzenetek száma hírfolyamonként

```
SELECT hirfolyam.megnevezes, COUNT(*) AS
    uzenetekSzama
FROM uzenet, hirfolyam
WHERE uzenet.hirfolyam_azonosito =
    hirfolyam.azonosito
GROUP BY hirfolyam.azonosito,
    hirfolyam.megnevezes;
```

Az eredmény ehhez hasonló lesz:

UZENET

sorszám	tartalom	mikor	felhasznalonev	hirfolyam_azonosito
1	Hello mindenkinek!	2018-10-03 11:10:00	gnemeth	1
2	Sziasztok!	2018-10-06 09:45:00	pbalazs	1
3	Sziasztok!	2018-10-07 10:01:00	pkardos	1
4	Tudja-e valaki, hogy ...	2018-10-15 17:00.00	pkardos	1
5	Szerintem itt nézd meg ...	2018-10-16 14:32:00	bodnaar	1
6	A múlt órán ...	2018-10-17 12:30:00	bodnaar	2
7	A jövő héten ...	2018-10-18 14:50:00	gnemeth	2
8	Kijavítottam a ZH-kat ...	2018-10-19 08:32:00	pkardos	2

HIRFOLYAM

azonosito	megnevezes
1	Információ
2	Gyakorlat

EREDMÉNY

hirfolyam.megnevezes	uzenetekSzama
Információ	5
Gyakorlat	3

Megjegyezzük, hogy a csoportosításnál fel kell tüntetni a hírfolyam azonosítót is és a hírfolyam megnevezést is, mert a hírfolyam megnevezést szeretnénk feltüntetni az eredménytáblában, de a hírfolyam azonosító lesz egyedi, mivel kulcs. A `hirfolyam` táblában a megnevezés nem kulcs, vagyis lehet két különböző, azonos nevű hírfolyam. A projekciónál nem kell minden oszlopot felsorolni, ami a `GROUP BY` után szerepel, mert a projekció a csoportosítás (és a csoport-szelekció) után hajtódik végre. Viszont csak olyan mező szerepelhet a `SELECT` után, amely szerepel a `GROUP BY`-ban vagy összesítő függvényben. A `COUNT()` függvényben írt * bármilyen oszlopot helyettesít.

Azt is megjegyezzük továbbá, hogy ez az összekapcsolás csak azokat a hírfolyamokat fogja kilistázni, amelyhez tartozik üzenet, valamint ha egy üzenet hírfolyam azonosítója nem szerepel az `hirfolyam` táblában, akkor az a rekord nem lesz figyelembe véve a számoláskor.

12.3. Természetes összekapcsolás

Az $R_1(A)$ és $R_2(B)$ relációsémák feletti T_1 és T_2 táblák természetes összekapcsolása az alábbi formulával írható le a relációs algebrában, ahol A és B a relációsémák attribútumhalmazait, $\prod$ a projekció, σ a szelekció műveletét jelöli, továbbá legyen $X = A \cap B$ és $A \cap B \neq \emptyset$:

$$T_1(A) \bowtie T_2(B) = \prod_{A \cup B} (\sigma_{T_1.X=T_2.X}(A \times B))$$

A természetes összekapcsolás a `SELECT` utasítás eddig bemutatott kulcsszavaival is elvégezhető, használjuk most sematikusan a T_1 és T_2 táblákat, valamint legyen az X most csak egy oszlop, kulcs – külső kulcs kapcsolat. Ha a kulcs – külső kulcs kapcsolat több oszlopból áll, akkor minden oszlopra el kell végezni az összehasonlítást.

Természetes összekapcsolás

```
SELECT * FROM T1, T2 WHERE T1.X = T2.X;
```

A természetes összekapcsolásra az SQL nyelvben használható még az `INNER JOIN` kulcsszó is. Ennek használata a következő:

Természetes összekapcsolás az `INNER JOIN` kulcsszóval

```
SELECT * FROM T1 INNER JOIN T2 ON T1.X = T2.X;
```

12.3.1. példa

Tekintsük a **Fórum** adatbázist és kérdezzük le természetes összekapcsolással a hírfolyamok üzeneteit!

Hírfolyamok üzenetei

```
SELECT * FROM hirfolyam, uzenet WHERE
   hirfolyam.azonosito =
   uzenet.hirfolyam_azonosito;
```

Hírfolyamok üzenetei (`INNER JOIN`-nal)

```
SELECT * FROM hirfolyam INNER JOIN uzenet ON
    hirfolyam.azonosito =
    uzenet.hirfolyam_azonosito;
```

A természetes összekapcsoláshoz használható még a `NATURAL JOIN` kulcsszó is, ez azonban kicsit másképpen működik. A `NATURAL JOIN` esetében a két összekapcsolt táblának a közös attribútumhalmaza ugyanazokat az oszlopneveket tartalmazza mindkét táblában és az párosított oszlopok típusa is megegyezik. Ebből kifolyólag nem kell külön megadnunk a kapcsolódó (kulcs – külső kulcs) oszlopokat. Az eredményben a két összekapcsolt tábla oszlopainak uniója jelenik meg úgy, hogy az közös oszlop csak egy példányban van jelen.

Természetes összekapcsolás `NATURAL JOIN` kulcsszóval

```
SELECT * FROM T1 NATURAL JOIN T2;
```

12.3.2. példa

Tekintsük a **Fórum** adatbázist és kérdezzük le, hogy mely üzenetet mely felhasználó írta!

Felhasználók és üzeneteik

```
SELECT * FROM felhasznalo NATURAL JOIN uzenet;
```

A két tábla:

FELHASZNALO

felhasznalonev	jelszo	vezeteknev	keresztnev	utolso_belepes_idopontja
pbalazs	Bp1234	Balázs	Péter	2019-07-03 12:10:00
gnemeth	Ng5678	Németh	Gábor	2018-11-23 15:25:00
pkardos	Kp4567	Kardos	Péter	2019-02-24 09:52:00
bodnaar	Bp9876	Bodnár	Péter	2019-03-16 18:30:00
vargalg	Vl6543	Varga	László Gábor	2019-02-10 15:00:00

UZENET

sorszám	tartalom	mikor	felhasznalonev	hirfolyam_azonosito
1	Hello mindenkinek!	2018-10-03 11:10:00	gnemeth	1
2	Sziasztok!	2018-10-06 09:45:00	pbalazs	1
3	Sziasztok!	2018-10-07 10:01:00	pkardos	1
4	Tudja-e valaki, hogy ...	2018-10-15 17:00.00	pkardos	1
5	Szerintem itt nézd meg ...	2018-10-16 14:32:00	bodnaar	1

Az eredmény ehhez hasonló lesz:

EREDMÉNY

felhasz...	jelszo	veze...	kere..	utolso...	sorszám	tartalom	mikor	hir...
gnemeth	Ng...	Németh	Gábor	2018-11-...	1	Hello ...	2018-10-...	1
pbalazs	Bp...	Balázs	Péter	2019-07-...	2	Sziasztok!	2018-10-...	1
pkardos	Kp...	Kardos	Péter	2019-02-...	3	Sziasztok!	2018-10-...	1
pkardos	Kp...	Kardos	Péter	2019-02-...	4	Tudja-e ...	2018-10-...	1
bodnaar	Bp...	Bodnár	Péter	2019-03-...	5	Szerintem ...	2018-10-...	1

12.4. Jobboldali, baloldali és teljes külső összekapcsolás

Két tábla külső összekapcsolása esetében valamelyik vagy mindkét tábla rekordjai szerepelnek az eredménytáblában (ami a természetes összekapcsolásra nem igaz).

A baloldali külső összekapcsolásra SQL-ben a `SELECT` utasításon belül a `LEFT JOIN` kulcsszó szolgál. Ebben az esetben a baloldali tábla minden rekordja megmarad, és a rekordokhoz párosítjuk a jobboldali tábla rekordjait (ahol lehet). A jobboldali külső összekapcsolás esetén fordítva van, a jobboldali tábla minden rekordja megmarad, és hozzájuk párosítjuk a baloldali tábla rekordjait. A jobb oldali külső összekapcsoláshoz a `SELECT` utasításban a `RIGHT JOIN` kulcsszót használjuk. A teljes külső összekapcsolásnál mindkét tábla minden rekordja szerepel az eredménytáblában, de ahol a rekordokat a kapcsoló attribútumhalmazon (kulcs – külső kulcs kapcsolaton) össze tudjuk kapcsolni, ott összekapcsoljuk, ahol pedig nem találtunk párt, ott a mezőket üresen hagyjuk. A teljes külső összekapcsoláshoz a `SELECT` utasításban a `FULL OUTER JOIN` kulcsszót használjuk. Jelöljük most sematikusan a baloldali táblát $T1$-gyel, a jobboldalit pedig $T2$-vel! Legyen most is a közös attribútum az X.

Baloldali külső összekapcsolás

```
SELECT * FROM T1 LEFT JOIN T2 ON T1.X = T2.X;
```

Jobboldali külső összekapcsolás

```
SELECT * FROM T1 RIGHT JOIN T2 ON T1.X = T2.X;
```

Teljes külső összekapcsolás

```
SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.X = T2.X;
```

12.4.1. példa

Tekintsük a **Fórum** adatbázist!

FELHASZNÁLÓ(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
ÜZENET(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
HÍRFOLYAM(azonosító, megnevezés)
KULCSSZAVAK(*hírfolyam azonosító*, kulcsszó)
KÖVETI(*hírfolyam azonosító*, *felhasználónév*)

Határozzuk meg, hogy az egyes felhasználók hány üzenetet írtak eddig fórumba. A megoldáshoz baloldali külső összekapcsolást kell alkalmazni, mivel előfordulhat, hogy van olyan felhasználó, aki még nem írt üzenetet.

Felhasználók üzeneteinek száma

```
SELECT felhasznalo.felhasznalonev, COUNT(*) AS
   uzenetekSzama
FROM felhasznalo LEFT JOIN uzenet ON
   felhasznalo.felhasznalonev =
   uzenet.felhasznalonev;
GROUP BY felhasznalo.felhasznalonev;
```

Az eredmény ehhez hasonló lesz:

UZENET

sorszám	tartalom	mikor	felhasznalonev	hirfolyam_azonosito
1	Hello mindenkinek!	2018-10-03 11:10:00	gnemeth	1
2	Sziasztok!	2018-10-06 09:45:00	pbalazs	1
3	Sziasztok!	2018-10-07 10:01:00	pkardos	1
4	Tudja-e valaki, hogy ...	2018-10-15 17:00.00	pkardos	1
5	Szerintem itt nézd meg ...	2018-10-16 14:32:00	bodnaar	1

FELHASZNALO

felhasznalonev	jelszo	vezeteknev	keresztnev	utolso_belepes_idopontja
pbalazs	Bp1234	Balázs	Péter	2019-07-03 12:10:00
gnemeth	Ng5678	Németh	Gábor	2018-11-23 15:25:00
pkardos	Kp4567	Kardos	Péter	2019-02-24 09:52:00
bodnaar	Bp9876	Bodnár	Péter	2019-03-16 18:30:00
vargalg	Vl6543	Varga	László Gábor	2019-02-10 15:00:00

EREDMÉNY

felhasznalo.felhasznalonev	uzenetekSzama
gnemeth	1
pbalazs	1
pkardos	2
bodnaar	1
vargalg	0

Tekintsünk most egy másik példát, ahol a csoport-feltételben használjuk az összesítő függvény eredményét!

12.4.2. példa

Tekintsük a **Fórum** adatbázist! Listázzuk ki azon felhasználók felhasználóneveit, akik még nem írtak bejegyzést! Ehhez meg kell számolni, hogy a felhasználók egyenként hány bejegyzést írtak, és kik azok, akiknél az üzenetek száma 0. Ennek ellenőrzésére csoport-szelekciót használunk.

Nem írtak még bejegyzést

```
SELECT felhasznalo.felhasznalonev
FROM uzenet RIGHT JOIN felhasznalo
ON felhasznalo.felhasznalonev =
   uzenet.felhasznalonev
GROUP BY felhasznalo.felhasznalonev
HAVING COUNT(*) = 0;
```

Az eredmény ehhez hasonló lesz:

UZENET

sorszám	tartalom	mikor	felhasznalonev	hirfolyam_azonosito
1	Hello mindenkinek!	2018-10-03 11:10:00	gnemeth	1
2	Sziasztok!	2018-10-06 09:45:00	pbalazs	1
3	Sziasztok!	2018-10-07 10:01:00	pkardos	1
4	Tudja-e valaki, hogy ...	2018-10-15 17:00.00	pkardos	1
5	Szerintem itt nézd meg ...	2018-10-16 14:32:00	bodnaar	1

FELHASZNALO

felhasznalonev	jelszo	vezeteknev	keresztnev	utolso_belepes_idopontja
pbalazs	Bp1234	Balázs	Péter	2019-07-03 12:10:00
gnemeth	Ng5678	Németh	Gábor	2018-11-23 15:25:00
pkardos	Kp4567	Kardos	Péter	2019-02-24 09:52:00
bodnaar	Bp9876	Bodnár	Péter	2019-03-16 18:30:00
vargalg	Vl6543	Varga	László Gábor	2019-02-10 15:00:00

EREDMÉNY

felhasznalo.felhasznalonev
vargalg

12.5. Theta-összekapcsolás

Két tábla Theta-összekapcsolásákor képezzük a két tábla Descartes-szorzatát, majd egy adott feltétel alapján kiszűrünk ebből sorokat. Megjegyezzük, hogy

a Theta-összekapcsolás során nem feltételezzük, hogy lenne a két táblának közös kapcsolómezeje (pl. kulcs – külső kulcs kapcsolat). A T_1 és T_2 tábla Theta-összekapcsolása SQL-ben az alábbi utasítással adható meg:

Theta-összekapcsolás

```
SELECT * FROM T1, T2 WHERE feltétel;
```

12.6. Unió, metszet, különbség

A halmazműveletekre is létezik kulcsszó az SQL-ben. Az unió, metszet és különbség halmazműveletek csak kompatibilis táblákra értelmezhetők. A halmazműveletek két oldalán egy-egy lekérdezés áll. Az unió művelethez a `UNION`, a metszethez a `INTERSECT`, a különbséghez a `EXCEPT` kulcsszavakat használjuk. Megjegyezzük, hogy az egyes adatbáziskezelő rendszerek esetében ezekre a műveletekre a kulcsszavak eltérhetnek a szabványtól (például a különbség esetén a `MINUS` is használható).

Az `UNION` művelet használata

```
(SELECT * FROM tabla1) UNION (SELECT * FROM tabla2)
```

A `INTERSECT` utasítás használata

```
(SELECT * FROM tabla1) INTERSECT (SELECT * FROM
   tabla2)
```

A `EXCEPT` utasítás használata

```
(SELECT * FROM tabla1) EXCEPT (SELECT * FROM tabla2)
```

12.6.1. példa

Tekintsük a **Fórum** adatbázist! Listázzuk ki azon felhasználók felhasználóneveit, akik még nem írtak bejegyzést! Ehhez az egyik lekérdezésben ki kell listáznunk a felhasználók felhasználóneveit, majd ebből a halmazból ki kell vonni azokat a felhasználóneveket, amelyek szerepelnek az `UZENET` táblában.

Nem írtak még bejegyzést

```
(SELECT felhasznalonev FROM felhasznalo)
EXCEPT
(SELECT felhasznalonev FROM uzenet);
```

Kérdések és feladatok

Az alábbi feladatok a **Programkalauz** adatbázisra vonatkoznak.

`PROGRAMOK`(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)
`HELYEK`(helyazonosító, város, cím, hely neve)
`MŰFAJ`(*programazonosító*, műfajmegnevezés)

1. SQL lekérdezéssel listázza ki a mai programokat kezdési időpontjuk (`mikortól` attribútum) szerint! Az eredményben tüntesse fel, hogy melyik városban, melyik helyen van az adott program, a program nevét és kezdési időpontját. Ügyeljen arra, hogy a többnapos programok nem biztos, hogy a mai napon kezdődnek!

2. Írjon egy SQL lekérdezést, amely megadja, hogy melyik a legolcsóbb program, amelyik ebben a hónapban kezdődik. A hónap meghatározásához használja a `YEAR()` és `MONTH()` függvényeket, amelyeknek egyetlen paramétere egy dátum vagy dátum-idő típusú érték.

3. Listázza ki egy SQL lekérdezéssel a szegedi programokat!

4. Írjon egy SQL lekérdezést, amely városonként megadja, hogy hány program van a városban!

5. Írjon SQL lekérdezést, amely megadja, hogy melyek azok a helyek, amelyek idén nem rendeztek programot!

13. fejezet

Alkérdések

Most, hogy megismertük a `SELECT` utasítást, és tudjuk, hogy hogyan írjunk lekérdezéseket, egy újabb ismerettel bővítjük eszköztárunkat. Az alkérdések olyan lekérdezések, amelyeket más SQL utasításban használunk. Ebben a fejezetben azt tanulhatja meg az Olvasó, hogy miként adhat ki adatmanipulációs (beszúró, módosító vagy törlő) utasításokat egy szűkebb halmazra, vagy éppen hogyan lehet egy lekérdezés feltételében megadni egy másik lekérdezést.

Az alkérdés is tulajdonképpen egy `SELECT` utasítás. Alkérdéseket leginkább más lekérdezések `WHERE` és `HAVING` feltételeiben szoktunk megadni, illetve – ha az adott adatbázis-kezelő rendszer megengedi – akkor a `FROM` kulcsszó után is megadható, hiszen egy lekérdezés eredménye egy tábla lesz. Alkérdéseket megadhatunk továbbá `UPDATE` és `DELETE` utasítás `WHERE` záradékában is. Az `INSERT` utasítás egy speciális változatában is megadható alkérdés, amikor egy táblába egy lekérdezés eredmény-sorait szeretnénk beszúrni. Az alábbiakban ezekre a lehetőségekre látunk példákat.

13.1. Alkérdés használata beszúró, módosító és törlő utasításokban

Az `INSERT INTO` utasításnak létezik olyan változata, amely egy alkérdés alapján szúr be rekordokat egy táblába.

Az `INSERT INTO` utasítás alkérdéssel

```
INSERT INTO táblanév [(oszloplista)] AS (alkérdés);
```

Megjegyezzük, hogy a szögletes zárójelben jelölt oszloplista opcionális,

viszont megadása kötelező, ha az alkérdés oszlopainak sorrendje nem azonos a `táblanév` tábla oszlopainak sorrendjével. Természetesen a tábla oszlopnevei eltérhetnek a lekérdezett oszlopnevektől.

Tegyük fel, hogy van egy olyan táblánk, amelyekben egy bolt kiadásait és bevételeit tartjuk nyilván az alábbi séma szerint:

KASSZA(dátumidő, összeg)

Az összeg értéke negatív, ha kiadásról van szó, és pozitív, ha bevételről van szó. Tisztában vagyunk azzal, hogy hosszú távon nagyon sok rekord gyűlik össze ebben a táblában. Szeretnénk valahogy évekre bontani (archiválni) ezeket az adatokat, mivel egy korábbi évben már nem változnak ezek az összegek. Létrehozunk hasonló táblákat az egyes évekre, amelyek ugyanezeket az oszlopokat fogják tartalmazni, viszont csak azok a rekordok kerülnek az egyes táblákba, amelyek arra az évre vonatkoznak:

KASSZA2016(dátumidő, összeg)
KASSZA2017(dátumidő, összeg)

Milyen SQL utasítással tudjuk feltölteni ezeket táblákat?

13.1.1. példa

Töltsük fel a KASSZA2016 és KASSZA2017 táblákat!

Adatok beszúrása alkérdéssel

```
INSERT INTO kassza2016 (datumido, osszeg) AS
   (SELECT datumido, osszeg FROM kassza WHERE
   YEAR(datumido) = 2016);

INSERT INTO kassza2017 (datumido, osszeg) AS
   (SELECT datumido, osszeg FROM kassza WHERE
   YEAR(datumido) = 2017);
```

Módosító utasításban az alkérdés az `UPDATE` utasítás `WHERE` záradékában lévő feltételben szerepel:

Alkérdés használata `UPDATE` utasításban

```
UPDATE táblanév SET érték = kifejezés WHERE oszlop
   relációjel (alkérdés);
```

A fenti sematikus utasításban a `relációjel` lehet pl. =, <, >, <>, !=, <=, >=, `IN`.

13.1.2. példa

Tekintsük a **Programkalauz** adatbázist!

PROGRAMOK(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)
HELYEK(helyazonosító, város, cím, hely neve)
MŰFAJ(*programazonosító*, műfajmegnevezés)

Csökkentsük a 2018. decemberi szegedi programok árát 10%-kal! Ehhez az UPDATE utasítás feltételében egy alkérdést adunk meg.

Decemberi szegedi programok árának csökkentése

```
UPDATE programok SET artol=0.9*artol,
   arig=0.9*arig
WHERE artol IS NOT NULL AND arig IS NOT NULL
   AND programazonosito IN (SELECT
   programazonosito FROM programok, helyek
   WHERE programok.helyazonosito =
   helyek.helyazonosito AND varos = 'Szeged'
   AND YEAR(mettol)=2018 AND YEAR(meddig)=2018
   AND MONTH(mettol)=12 AND MONTH(meddig)=12);
```

A törlő utasításokban hasonlóan járunk el, mint a módosítókban. Itt is a WHERE záradék feltételében használjuk az alkérdést.

Alkérdés használata DELETE FROM utasításban

```
DELETE FROM táblanév WHERE oszlop <reláció>
   (alkérdés);
```

13.1.3. példa

Tekintsük most is a **Programkalauz** adatbázist!

PROGRAMOK(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)
HELYEK(helyazonosító, város, cím, hely neve)
MŰFAJ(*programazonosító*, műfajmegnevezés)

Töröljük a *Sarki Kávézó* programjait, mert megszűnt!

A *Sarki Kávézó* programjainak törlése

```
DELETE FROM programok WHERE helyazonosito =
   (SELECT helyazonosito FROM helyek WHERE
   helyNeve='Sarki Kávézó');
```

13.2. Alkérdés használata lekérdezésben

Lekérdezésben leginkább a `WHERE` és `HAVING` záradékban használhatunk alkérdést, azonban egyes adatbáziskezelő rendszerek a `FROM` után is megengedik ezt. Ennek oka érthető, hiszen egy lekérdezés eredménye egy tábla, amelynek képezhető Descartes-szorzata más táblákkal. Bizonyos szempontból ez a lehetőség hatékonyságot is jelenthet, mivel egy lekérdezés eredménye általában kevesebb sort tartalmaz, mint a teljes tábla, és ekkor a Descartes-szorzatnál is kevesebb lesz az eredménytábla sorainak száma.

Alkérdés használata lekérdezésben

```
SELECT oszloplista
FROM táblalista
WHERE feltétel <reláció> (alkérdés)
[GROUP BY oszloplista]
[HAVING csoportfeltétel <reláció> (alkérdés)]
[ORDER BY oszloplista];
```

Nézzünk példát olyan lekérdezésre, amely a `WHERE` záradékában tartalmaz alkérdést!

13.2.1. példa

Tekintsük a **Programkalauz** adatbázist!

`PROGRAMOK`(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)
`HELYEK`(helyazonosító, város, cím, hely neve)
`MŰFAJ`(*programazonosító*, műfajmegnevezés)

Listázzuk ki a szegedi programokat! A megoldáshoz használjunk alkérdést!

Szegedi programok listázása

```
SELECT * FROM programok WHERE helyazonosito IN
   (SELECT helyazonosito FROM helyek WHERE
   varos = 'Szeged');
```

Gondoljuk végig, hogy a megoldást összekapcsolással is elvégezhettük volna, azonban akkor a PROGRAMOK és a HELYEK tábla Descartes-szorzatát kellett volna számolni. Ha a PROGRAMOK tábla m, a HELYEK tábla pedig n sort tartmaz, akkor Descartes-szorzatban $m * n$ sort kellett volna vizsgálni. Ebben a megoldásban az alkérdés előre kiértékelődik, amely n feltételvizsgálatot jelent, majd a főkérdésben lévő feltétel m vizsgálatot jelent, vagyis ennek a lekérdezésnek a műveletigénye $\mathcal{O}(m + n)$.

Most nézzünk arra példát, hogy a HAVING záradék feltételében hogyan használhatunk alkérdést!

13.2.2. példa

Tekintsük a fent említett KASSZA táblát!
KASSZA(datumido, osszeg)

Melyek voltak azok az évek, amelyek bevétele jobb volt az átlagosnál? Emlékezzünk, hogy a negatív értékek kiadást, a pozitív értékek pedig bevételt jelentenek.

Nyereséges évek

```
SELECT YEAR(datumido) AS év
FROM kassza
WHERE osszeg > 0
GROUP BY YEAR(datumido)
HAVING AVG(osszeg) > (SELECT AVG(osszeg) FROM
   kassza);
```

Az alkérdés ebben a példában is előre értékelődik ki.

Nézzünk most egy olyan példát, ahol az alkérdést nem tudjuk előre kiértékelni! Ez akkor fordul elő, ha az alkérdés feltételében a főkérdés valamelyik oszlopa is szerepel.

13.2.3. példa

Tekintsük ismét a KASSZA táblát!
KASSZA(datumido, osszeg)

Melyik évben mennyi volt a legnagyobb bevétel?

Az évek legnagyobb bevételei

```
SELECT YEAR(datumido) AS ev, osszeg
FROM kassza AS k1
WHERE k1.osszeg > 0 AND k1.osszeg = (SELECT
   MAX(osszeg) FROM kassza AS k2 WHERE
   YEAR(k1.datumido) = YEAR(k2.datumido) );
```

Kérdések és feladatok

A következő feladatokhoz tekintsük a **Programkalauz** adatbázist!

PROGRAMOK(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)
HELYEK(helyazonosító, város, cím, hely neve)
MŰFAJ(*programazonosító*, műfajmegnevezés)

1. Írjon olyan SQL utasítást, amely 10%-kal növeli a budapesti programok árát! Használjon alkérdést!
2. Írjon olyan SQL utasítást, amely lekérdezi a Szegedi Nemzeti Színház programjait! Használjon alkérdést!
3. Írjon olyan SQL utasítást, amely törli a „talkshow” műfajú programokat, mivel a rendszer már nem foglalkozik ezekkel! Használjon alkérdést!
4. Írjon olyan SQL lekérdezést, amely kilistázza, hogy hol volt a legkorábbi program! Használjon alkérdést! A listában tüntesse fel a várost és a hely nevét!
5. Írjon SQL lekérdezést, amely kiírja, hogy melyik hely adott helyszínt eddig a legtöbb programnak!

14. fejezet

Virtuális táblák, nézettáblák

Ebben a fejezetben a nézettáblák létrehozását és használatát mutatjuk be. A fejezet végére az Olvasó tisztában lesz a nézettábla fogalmával, létre tudja hozni azokat SQL-ben. Továbbá megtanulja azt, hogy mikor használhatunk egy nézettáblán aktualizáló műveletet és mikor nem.

14.1. A nézettáblák létrehozása

A nézettáblák, vagy más néven virtuális táblák egy lekérdezés transzformációs formulái, amelyekre SQL-ben táblaként hivatkozhatunk. A nézettáblák azonban nem fizikai táblák, mint amivel eddig ismerkedtünk meg. A nézettáblák rendelkeznek sémával, de tartalmuk, rekordjaik az aktuális táblatartalomnak megfelelően változnak. A nézettáblákat a `CREATE VIEW` utasítással hozzuk létre.

A `CREATE VIEW` utasítás

```
CREATE VIEW tablanev [(oszloplista)] AS (alkerdes);
```

A `CREATE VIEW` utasításban meg kell adnunk a nézettábla nevét és nem kötelező, de meg lehet adni az oszlopneveket. Az oszlopok száma meg kell egyezzen az alkérdés eredménytáblájában lévő oszlopszámmal, ugyanis ezek az oszlopok fognak szerepelni a nézettáblában is, csak az oszlistában feltüntetett neveken.

14.1.1. példa

Tekintsük a **Fórum** adatbázist!

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév,

utolsó belépés időpontja)
ÜZENET(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
HÍRFOLYAM(azonosító, megnevezés)
KULCSSZAVAK(*hírfolyam azonosító*, kulcsszó)
KÖVETI(*hírfolyam azonosító*, *felhasználónév*)

Készítsünk nézettáblát azokról a felhasználókról, akik nem léptek be idén fórumba! A nézettábla tartalmazza a felhasználó nevét és utolsó belépésének időpontját!

Régi felhasználók nézettáblája

```
CREATE VIEW regiFelhasznalok(felhasznalonev,
   utolso_belepes_idopontja) AS
SELECT felhasznalonev, utolso_belepes_idopontja
   FROM felhasznalok WHERE
   YEAR(utolso_belepes_idopontja) <
   YEAR(CURRENT_DATE);
```

14.2. Milyen műveleteket használhatunk nézettáblákra?

A nézettáblák ugyan transzformációs formulák, de ha létrehozunk egy nézettáblát, akkor az ugyanúgy látszik SQL-ben, mintha egy fizikai tábla lenne. A használatában azonban van különbség.

A lekérdezéseknél minden további nélkül használhatjuk a nézettáblákat, mintha fizikai táblák lennének.

14.2.1. példa

Kérdezzük le a 14.1.1. példában létrehozott nézettábla rekordjait!

REGIFELHASZNALOK nézettábla lekérdezése

```
SELECT * FROM regiFelhasznalok;
```

Beszúrás, módosítás és törlés azonban nem lehetséges akkor, ha az alábbi esetek valamelyike fennáll:

`DISTINCT` szerepel a lekérdezésben: A `DISTINCT` kulcsszó az ismétlődő sorokból csak egy példányt tart meg. Ezáltal nem lesz egyértelmű, hogy mely rekordra vonatkozna a módosítás vagy törlés.

`FROM` után legalább két tábla áll: Ha két tábla Descartes-szorzatát képezzük, nem lesz egyértelmű, hogy mely rekordra vonatkozik módosítás és törlés.

`GROUP BY` szerepel a lekérdezésben: A csoportosítás során sorokat vonunk össze, így bizonyos sorok eltűnnek az eredményhalmazból, ezáltal nem lesz egyértelmű a rekord módosítása, törlése.

ha az alkérdés nem tartalmazza a kulcsot: Ebben az esetben nem tudjuk egyértelműen meghatározni a rekordot.

ha a nézettábla egyik oszlopa származtatott adatot tartalmaz: Ebben az esetben, például, ha összesítő függvényt, vagy valamilyen kifejezést tartalmaz a nézettábla alkérdésének eredménytáblája, akkor nem lehet módosítani, mert nem tudjuk, hogy mit módosítsunk.

Nézzünk olyan esetekre példát, amikor lehet használni beszúrást, módosítást, törlést!

14.2.2. példa

Tekintsük a **Fórum** adatbázist!

`FELHASZNÁLÓ`(<u>felhasználónév</u>, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
`ÜZENET`(<u>sorszám</u>, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
`HÍRFOLYAM`(<u>azonosító</u>, megnevezés)
`KULCSSZAVAK`(<u>*hírfolyam azonosító*</u>, <u>kulcsszó</u>)
`KÖVETI`(<u>*hírfolyam azonosító*</u>, <u>*felhasználónév*</u>)

Hozzunk létre egy nézettáblát `GNEMETHUZENETEI` néven, amely a `gnemeth` felhasználó üzeneteit tartalmazza csupán!

`GNEMETHUZENETEI` **nézettábla**

```
CREATE VIEW gnemethUzenetei AS SELECT sorszam,
   tartalom, mikor FROM uzenet WHERE
   felhasznalonev='gnemeth';
```

Szúrjunk be új rekordot!

Rekord beszúrása a GNEMETHUZENETEI táblába

```
INSERT INTO gnemethUzenetei VALUES (16,
   'Hello', NOW());
```

Ez az utasítás beszúrja a rekordot a táblába, ha az UZENET tábla külső kulcsainak alapértelmezett értéke adott. Ha nem, és nekünk kell megadni, akkor viszont nem tudja beszúrni a rekordot, mivel a beszúrás nem a GNEMETHUZENETEI táblába történik (ahol nincs ilyen oszlop), hanem az UZENET táblába.
Lássuk, hogy tudunk-e módosítani?

Módosítás a GNEMETHUZENETEI táblában

```
UPDATE gnemethUzenetei SET mikor = NOW() WHERE
   sorszam = 16;
```

Ez működik, hiszen a sorszam oszlop kulcs az UZENET táblában, és ez a módosítás átvezethető. Nézzünk egy másik módosítást!

Módosítás a GNEMETHUZENETEI táblában

```
UPDATE gnemethUzenetei SET tartalom = 'Szia'
   WHERE tartalom = 'Hello';
```

Ez a módosítás az üzenet szövegének tartalmát módosítja. Az utasítás végrehajtódik, de nem csupán gnemeth üzeneteire, hanem az összes üzenetre, amelyek tartalma „Hello”.
Lássuk a törlés lehetőségét!

Törlés a GNEMETHUZENETEI táblából

```
DELETE FROM gnemethUzenetei WHERE sorszam = 16;
```

Ez az utasítás egy rekordot töröl, mert a sorszám alapján be tudtuk azonosítani a rekordot az UZENET táblában is.

Törlés a GNEMETHUZENETEI táblából

```
DELETE FROM gnemethUzenetei WHERE
   YEAR(mikor)<YEAR(CURRENT_DATE);
```

Ez az utasítás is lefut, azonban nem csupán `gnemeth` üzeneteit fogja törölni, hanem az összeset, amelyet nem idén írtak.

Kihangsúlyozzuk, hogy a nézettáblákon keresztüli módosítás és törlés kerülendő, ha a nézettáblában nem szerepel a rekord kulcsa, ugyanis anomáliákhoz és nemkívánatos adatvesztéshez vezethet.

14.3. Adatok elrejtése és kiválasztása nézettáblákkal

A nézettáblák egyik alkalmazási területe az, hogy általuk elrejthetünk bizonyos információkat a felhasználók elől. A nézettáblák jogosultságkezelése ugyanis külön kezelhető. Emiatt megoldható az, hogy ha egy cég dolgozói adatait kezeljük, az összes dolgozói adat (személyes, orvosi és beosztás adatok) mind egy táblában van, de külön-külön nézettáblaként jelenítjük meg a személyes, az orvosi és a beosztás adatokat az arra illetékes felhasználóknak, például az üzemorvos és a munkaügyis kolléga számára. Ez lényegében egy projekciót jelent az adatokra vonatkozóan.

14.3.1. példa

Tekintsük a DOLGOZÓ táblát az alábbi séma szerint!

DOLGOZO(azonosito, nev, lakcim, fizetes, munkakor, beosztas, betegseg, betegseg_mikortol)

Személyes adatok

```
CREATE VIEW szemelyesAdatok AS SELECT
   azonosito, nev, lakcim FROM dolgozo;
```

Orvosi adatok

```
CREATE VIEW orvosiAdatok AS SELECT azonosito,
   nev, lakcim, betegseg, betegseg_mikortol
   FROM dolgozo;
```

Munkaügyi adatatok

```
CREATE VIEW munkaugyiAdatok AS SELECT
   azonosito, nev, lakcim, munkakor, fizetes,
   beosztas FROM dolgozo;
```

A másik lehetőség, hogy ha a cég bevételeit és kiadásait évente akarjuk figyelembe venni, akkor minden évre vonatkozóan létrehozhatunk egy nézettáblát, ahol az alkérdés `WHERE` záradékának felételében megadjuk a kívánt évet. Ekkor a nézettáblában csak az az adott évi rekordok jelennek meg. Milyen előnnyel jár ez? Emlékezzünk vissza az alkérdések témakörére, amikor megemlítettük, hogy a lekérdezés `FROM` záradékában is használhatunk alkérdést. Ha ott nézettáblát használunk, akkor tulajdonképpen alkérdést használunk, amelyekkel várhatóan csökkentjük a Descartes-szorzaban lévő sorok számát.

14.3.2. példa

Tekintsük ismét a `DOLGOZO` táblát az alábbi séma szerint!

`DOLGOZO(azonosito, nev, lakcim, fizetes, munkakor, beosztas, ev, honap)`

Készítsünk kimutatást kifizetéséről beosztások szerint a 2017-es évre vonatkozóan!

Éves kimutatás a 2017-es évre beosztasok szerint

```
CREATE VIEW fizetesek2017 (beosztas,
   evi_osszeg, ev) AS SELECT beosztas,
   SUM(fizetes) FROM dolgozo GROUP BY beosztas,
   ev HAVING ev=2017;
```

Kérdések és feladatok

1. Mikor nem használhatunk beszúrást, módosítást, törlést nézettáblán?
2. Hogyan gyorsíthatják a lekérdezést a nézettáblák?
3. Hozzon létre nézettáblát a **Programkalauz** adatbázisba, amely a 2018-as programokat listázza ki!

   PROGRAMOK(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)
   HELYEK(helyazonosító, város, cím, hely neve)
   MŰFAJ(*programazonosító*, műfajmegnevezés)

4. Hozzon létre nézettáblát a **Programkalauz** adatbázisba, amely a szegedi programokat tartalmazza!
5. Hozzon létre nézettáblát, amely azokat a dolgozókat gyűjti ki, akik az átlagfizetés alatt keresnek!
   DOLGOZO(azonosito, nev, lakcim, fizetes, munkakor, beosztas, betegseg, betegseg_mikortol)

15. fejezet

Indexek

Az indexek rendezett adatstrukturák, amelyek hatékonyabbá teszik a keresést és a rendezést az adatokban. Ebben a fejezetben bemutatjuk, hogy hogyan lehet indexeket létrehozni. Az Olvasó megismeri az indexek célját, betekintést nyer fizikai megvalósításukba, és ezen ismeretek birtokában saját munkájában körültekintően tudja majd használni azokat.

15.1. Az indexek fizikai szerkezete

Ahhoz, hogy megértsük az indexek hasznát és használhatóságát, kicsit bele kell látnunk az adatbázisok fizikai megvalósításába is. Az adatbázisok a fizikai adattáron tárolódnak. Az egyes rekordokat egymást követően írja a rendszer a fizikai adattárra, ilyenformán nincs rendezés a rekordok között. Ez lehetővé teszi az adatok gyors mentését - hiszen mindig az adatbázis-fájl végére írjuk az új rekordot, - viszont problémát jelent az adatok rendezésénél és a keresésnél. Gondoljuk végig, ha egy rendezetlen adatfájlban keresünk rekordokat, akkor az összes rekordot végig kell olvasnunk, és ki kell gyűjtenünk az általunk keresetteket. Ez a keresés $\mathcal{O}(n)$ műveletigénnyel [7] jár n darab rekord esetén. Ennél hatékonyabb kereséshez és rendezéshez az adatstruktúrán kellene változtatni, viszont az a beszúrást tenné lassabbá. A rekordok más adatstruktúra szerinti fizikai tárolása még nem oldja meg azt a problémát, hogy akár több attribútum szerinti keresés hatékony legyen. A megvalósítás tekintetében megoldást jelent az *indexek* létrehozása, amelyek oszlopokra vagy oszloplistákra vonatkozóan az abban tárolt értékekre hatékony adatstruktúrát épít, mely gyors keresést és rendezést tesz lehetővé. Az indexek az adatfájloktól függetlenül külön *indexfájlban* rendezett struktúrában (például szekvenciális fájlban vagy B+-fában) tárolódnak.

15.1.1. Index tárolása szekvenciális fájlban

A szekvenciális fájlban az *indexkulcsok* rendezett módon tárolódnak. Az indexkulcsok mellett szerepel a hozzá tartozó adatrekord címe (lásd 15.1. ábra). Így amikor egy indexszel rendelkező oszlop elemeit rendezzük, vagy abban keresünk értéket, az indexben lévő indexkulcsok alapján végezzük a rendezést és keresést. Ebben az esetben a műveletigény csupán $\mathcal{O}(\log_2 n)$ bináris keresés esetén.

15.1.2. Index tárolása B+-fában

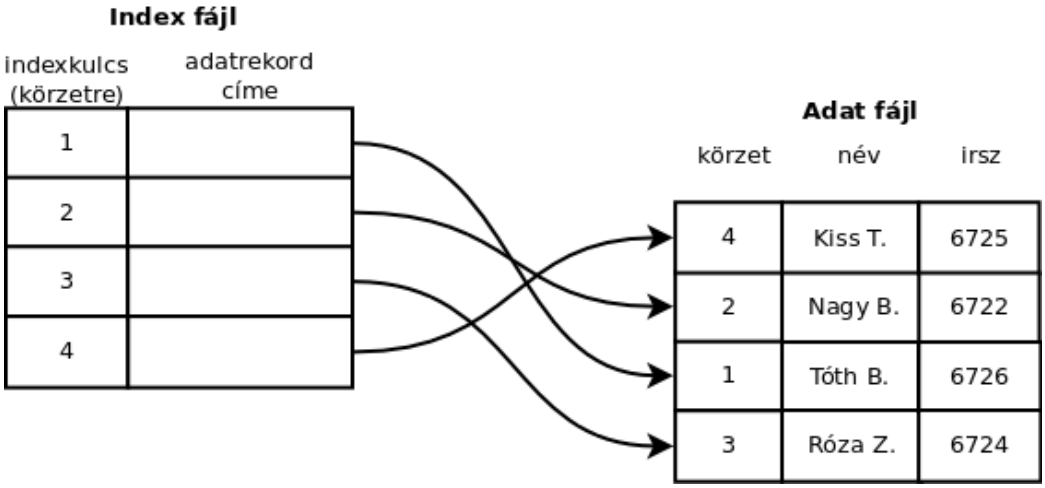
A B+-fa egy kiegyensúlyozott fa szerkezet, vagyis a gyökértől a levélig vezető utak egyforma hosszúak. A B+-fa csomópontjai blokkokból állnak. A B+-fa *rendje* r, ha egy blokkban r keresési kulcsérték tárolható. A B+-fa blokkjaiban ezen kívül $n + 1$ mutató található. A B+-fa csomópontjait három kategóriába soroljuk: gyökér, belső csúcsok, levelek. A B+-fa esetében az alábbi szabályokat kell megtartani:

- A gyökér csúcsban legalább egy keresési kulcs szerepel, és legalább két mutató használatban van. Minden mutató a B+-fa következő szintjére mutat.
- A belső csúcsokban legalább $\lceil \frac{n+1}{2} \rceil$ mutató használatban van. Minden mutató a következő szint egyik blokkjára mutat.
- Egy belső csúcsban vagy gyökérben a j-edik mutató olyan blokkra mutat, ahol a keresési kulcsok értéke kisebb, mint a j-edik keresési kulcs. A $(j + 1)$-edik mutató olyan blokkra mutat, ahol a keresési kulcsok értéke nagyobb vagy egyenlő, mint a j-edik keresési kulcs.
- A levél csúcsokban $\lfloor \frac{n+1}{2} \rfloor$ mutató használatban van és az adatrekordokra mutat, az $n + 1$-edik mutató pedig a következő levélblokkra mutat.
- Bármely szinten a keresési kulcsok rendezetettek.

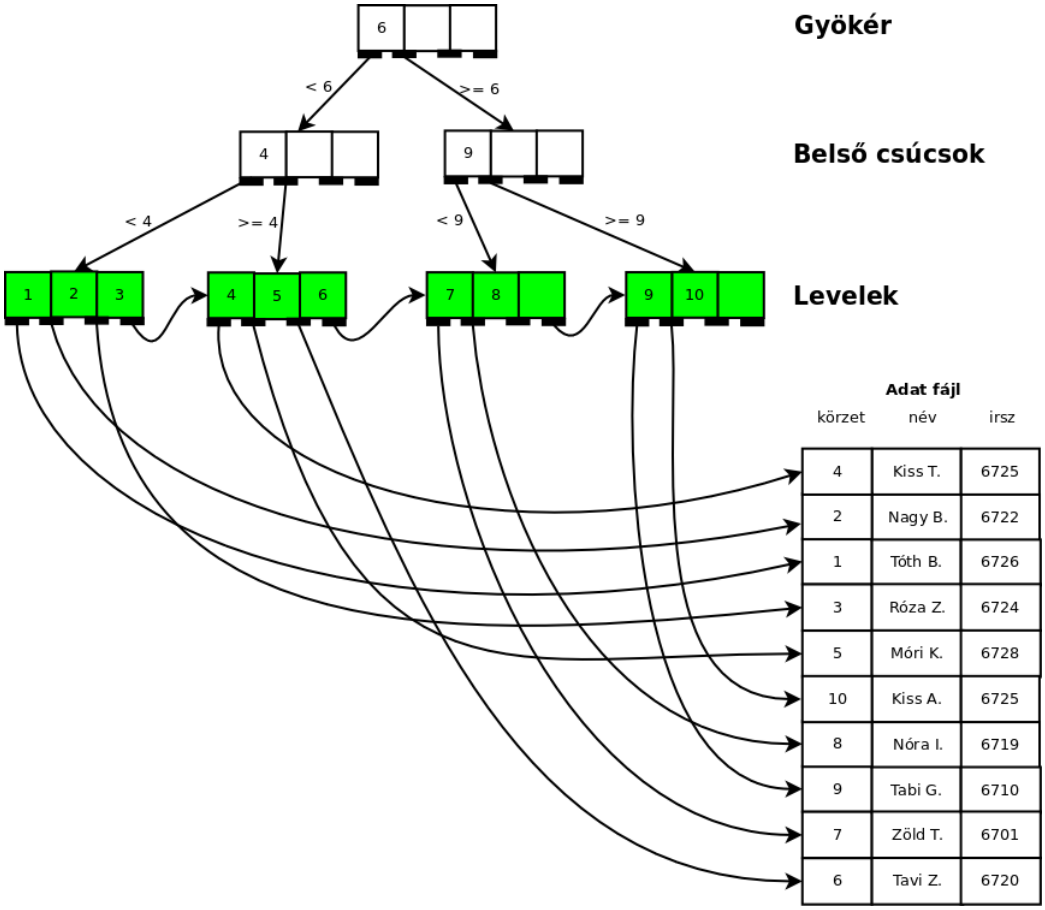
A 15.2. ábra egy 3-rendű B+fa indexet szemléltet, amelyben a leveleket zöld színnel jelöltük.

15.2. Indexek létrehozása SQL-ben

Az indexeket táblák egy-egy oszlopára vagy oszloplistájára hozzuk létre. Az elsődleges kulcs és a kulcsfeltétellel ellátott mezőkre az indextábla a legtöbb rendszerben automatikusan létrejön, ezeket *elsődleges indexeknek* nevezzük.



15.1. ábra. Index szekvenciális fájlban.



15.2. ábra. Index B+-fában.

A másodlagos indexek azok, amelyeket azokra az adatokra hozunk létre, amelyekben nagyon gyakran adunk ki keresést vagy rendezést. Az indexeket a `CREATE INDEX` utasítással hozzuk létre.

A `CREATE INDEX` utasítás

```
CREATE INDEX indexnév ON tábla(oszloplista);
```

15.2.1. példa

Tekintsük **Fórum** adatbázisban (10.1.2. példa) szereplő `UZENET` táblát!

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja)
`ÜZENET`(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
`HÍRFOLYAM`(azonosító, megnevezés)
`KULCSSZAVAK`(*hírfolyam azonosító*, kulcsszó)
`KÖVETI`(*hírfolyam azonosító*, *felhasználónév*)

Hozzunk létre a `mikor` attribútumra indexet, mivel szeretnénk az üzeneteket időpont szerint rendezni!

UzenetIdopontIndex

```
CREATE INDEX UzenetIdopontIndex ON
   uzenet(mikor);
```

Célszerű az `UZENET` tábla `hirfolyam_azonosito` oszlopára is létrehozni egy indexet, mert külső kulcsként gyakran kell használni az összekapcsoláshoz.

UzenetHirfolyamAzonositoIndex

```
CREATE INDEX UzenetHirfolyamAzonositoIndex ON
   uzenet(hirfolyam_azonosito);
```

15.2.2. példa

Tekintsük a 10.1.3 példában szereplő **Programkalauz** adatbázist!

`PROGRAMOK`(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)

HELYEK(helyazonosító, város, cím, hely neve)
MŰFAJ(*programazonosító*, műfajmegnevezés)

Hozzunk létre indexet a HELYEK táblában lévő varos oszlopra, mivel gyakran fogjuk egy-egy városra szűrni a programokat!

HelyekVarosIndex

```
CREATE INDEX HelyekVarosIndex ON helyek(varos);
```

Hozzunk létre indexet a PROGRAMOK táblában szereplő mikortol és meddig attribútumra, mivel gyakran kell az aktuális vagy a közeljövőben lévő programokat listázni!

ProgramDatumIndex

```
CREATE INDEX ProgramDatumIndex ON
   programok(mikortol, meddig);
```

Kérdések és feladatok

1. Mi a különbség a szekvenciális fájl és rendezetlen adathalmaz között? Milyen időbonyolultságú a keresés az egyikben és a másikban?
2. Milyen időbonyolultságú a beszúrás egy szekvenciális fájlba?
3. Milyen időbonyolultságú a keresés egy r-rendű B+-fában?
4. Hozzon létre indexet **Programkalauz** adatbázisban (10.1.3 példa) szereplő PROGRAMOK tábla helyazonosito mezejére!
5. Hozzon létre indexet **Programkalauz** adatbázisban (10.1.3 példa) szereplő PROGRAMOK tábla cim mezejére!

16. fejezet

Triggerek

A triggerek olyan kis programok, aktív elemek az adatbázisokban, amelyek valamilyen adataktualizáló művelet vagy rendszerszintű művelet esetén hajtódnak végre. Ebben a fejezetben az Olvasó azt tanulja meg, hogy hogyan lehet ilyen triggereket létrehozni, illetve mutatunk néhány példát a használatukra is.

A triggereket két csoportba soroljuk:

adatszintű triggerek, amelyek valamilyen adatmanipulációs művelet esetén hajtódnak végre,

rendszerszintű triggerek, amelyek valamilyen rendszerművelet (például felhasználó bejelentkezése) alkalmával futnak le.

A triggerek létrehozása a `CREATE TRIGGER`, törlése a `DROP TRIGGER` utasítással történik. A triggerek működését lehet engedélyezni és tiltani is:

Triggerek létrehozása

```
CREATE TRIGGER triggernév ...
```

Triggerek engedélyezése

```
ALTER TRIGGER triggernév ENABLE;
```

Triggerek tiltása

```
ALTER TRIGGER triggernév DISABLE;
```

Triggerek törlése

```
DROP TRIGGER triggernév;
```

16.1. Adatszintű triggerek

Az adatszintű triggerek adat beszúrására, módosítására és/vagy törlésére aktiválódnak. Aktiváció alapján a triggerek lefuthatnak a művelet előtt vagy után. A tekintetben, hogy minden érintett sor esetén lefutnak-e, megkülönböztetünk *sorszintű* és *utasításszintű* triggereket. A sorszintű triggerek minden érintett sor esetén aktiválódnak, míg az utasításszintű triggerek csak az utasítás hatására (egyszer) futnak le, függetlenül attől, hogy az utasítás hány sort érint. Vagyis ha például egy módosító utasításunk öt sort érint, akkor ezen módosításra aktiválódó sorszintű triggerek ötször fognak lefutni, az utasításszintűek, pedig csak egyszer.

Miután megismertük a triggerek funkcióit, nézzük meg, hogyan lehet triggereket létrehozni! Megjegyezzük, hogy az egyes adatbáziskezelő rendszerek triggerekre vonatkozó szintaxisa eltér, ezért mindig érdemes megnézni a használt adatbáziskezelő rendszer referencia kézikönyvét!

A `CREATE TRIGGER` utasítás

```
CREATE TRIGGER triggernév
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF oszlopok]}
ON táblanév
[REFERENCING [OLD ROW AS réginév][NEW ROW AS újnév]]
[FOR EACH ROW]
[WHEN (feltétel)]
Programblokk;
```

Tekintsük át a fenti utasítás részleteit! A kapcsos zárójelben lévő elemek olyan kulcsszavak, amelyek közül használnunk kell egyet, a szögletes zárójelben lévőeket pedig használhatóak, de nem feltétlen kell használnunk őket. Az egyes kulcsszavak jelentése a következő:

`BEFORE | AFTER | INSTEAD OF:` a `BEFORE` kulcsszó hatására a trigger az aktualizáló művelet előtt fut le, az `AFTER` kulcsszó hatására pedig az aktualizáló művelet után. Magától értetődő, hogy a két kulcsszó közül csak az egyik szerepelhet az utasításban. A `BEFORE` és `AFTER` kulcsszavak

csak fizikai táblákra használhatók, az `INSTEAD OF` kulcsszó pedig csak nézettáblára használható.

`INSERT | DELETE | UPDATE [OF oszlop]`: Ezek a kulcsszavak az akutalizáló műveletre utalnak. Ha szerepel az `INSERT` kulcsszó az utasításban, akkor a trigger beszúrás hatására aktiválódik. Ha szerepel a `DELETE` kulcsszó az utasításban, akkor a trigger törlés hatására aktiválódik. Ha szerepel az `UPDATE` kulcsszó az utasításban, akkor a trigger módosítás hatására aktiválódik. Ha az `UPDATE` mellett az `OF oszlop` is meg van adva, akkor a trigger csak a megjelölt oszlop módosításának hatására aktiválódik.

`ON` **táblanév:** melyik táblára vonatkozik az adatmódosító utasítás.

`REFERENCING...`: A régi és az új értékek oszlophivatkozásait lehet megjelölni másodlagos névvel. Ha nem használunk másodlagos nevet, ez a kulcsszó kihagyható, a régi és új sor értékekre rendre a `OLD` és `NEW` kulcsszavakkal hivatkozhatunk.

- Törlésnél csak `OLD` használható, nincs új név, csak régi.
- Beszúrásnál csak `NEW` használható, nincs régi név, hiszen új sort szúrunk be.
- Módosításnál használható a régi és az új név is.

`FOR EACH ROW`: Ez a kulcsszó jelöli, hogy a triggerünk sorszintű, vagy utasításszintű. Ha a `FOR EACH ROW` szerepel a triggerben, akkor a trigger sorszintű, minden egyes érintett sor esetén lefut. Ha nem szerepel, akkor a trigger utasításszintű.

`WHEN (feltétel)`: A trigger csak akkor fut le, ha az itt megadott feltétel teljesül.

Programblokk: Ez a rész jelöli azokat az utasításokat, amelyek a trigger aktiválódásakor végrehajtódnak.

Most, hogy már ismerjük a szintaktikai elemeket, nézzünk néhány példát a triggerekre!

16.1.1. példa

Tekintsük az alábbi relációsémákat!

```
Dolgozo(azonosito, nev, fizetes)
Fizetesnaplo(azonosító, datum, regifizetes, ujfizetes)
```

Fizetésnapló trigger

```
CREATE TRIGGER fizetesnaplo
AFTER UPDATE OF fizetes
ON Dolgozó
FOR EACH ROW
  INSERT INTO Fizetesnaplo VALUES (NEW.azonosito,
     CURRENT_DATE, OLD.fizetes, NEW.fizetes);
```

Ez a trigger olyan utasítások alkalmával hajtódik végre, amelyek a DOLGOZO tábla `fizetes` oszlopát érintik, például:

Egy dolgozó fizetésének 10%-os emelése

```
UPDATE Dolgozo SET fizetes=fizetes * 1.1 WHERE
   azonosito=1234567890;
```

A fizetésnaplózást az aktualizáló művelet után kell végrehajtani, mivel meg kell bizonyosodni róla, hogy a művelet lefutott, szintakilag helyes és van olyan sor, amit módosított. A rendszer képes megvizsgálni és figyelembe venni a DOLGOZO tábla érintett sorának régi és új fizetési értékét, és ezeket eltárolni a FIZETESNAPLO táblába.

Most lássunk egy másik példát!

16.1.2. példa

Tekintsük az alábbi bolti adatbázist!

KESZLET(cikkszam, megnevezes, darabszam, ar)
ELADAS(*cikkszam*, idopont, darabszam)

Készítsünk triggert, amely az ELADAS táblába beszúrt új rekord esetén csökkenti a KÉSZLET táblából az áru mennyiségét. Az egyszerűség kedvéért feltételezzük, hogy nem tudunk több darabot eladni egy áruból, mint amennyi készleten van. Máskülönben ezt ellenőriznünk kell, amely ellenőrzés meghaladja az általános SQL-re vonatkozó leírásokat, mivel a különböző rendszerekben ezt különbözőképpen lehet megoldani.

Árucsökkentő trigger

```
CREATE TRIGGER arucsokkentoTrigger
BEFORE INSERT
ON Eladas
FOR EACH ROW
  UPDATE Keszlet SET Keszlet.darabszam =
     Keszlet.darabszam - NEW.darabszam WHERE
     Keszlet.cikkszam = NEW.cikkszam;
```

Ez a trigger azelőtt kell, hogy lefusson, mielőtt az eladást rögzítjük az adatbázisba. Ez különösen igaz akkor, ha a készletmennyiségre vonatkozó feltételt is ellenőrizzük (amit itt nem tettünk meg).
A készletellenőrző változata a fenti triggernek az alábbi módon néz ki, de hangsúlyozzuk, hogy ezt a szintaxist nem minden rendszer engedi meg.

Árucsökkentő trigger készletellenőrzéssel

```
CREATE TRIGGER
   arucsokkentoTriggerKeszletellenorzessel
BEFORE INSERT
ON Eladas
FOR EACH ROW
WHEN (NEW.darabszam <= (SELECT darabszam FROM
   Keszlet WHERE Keszlet.cikkszam =
   NEW.cikkszam))
  UPDATE Keszlet SET Keszlet.darabszam =
     Keszlet.darabszam - NEW.darabszam WHERE
     Keszlet.cikkszam = NEW.cikkszam;
```

16.2. Rendszerszintű triggerek

A rendszerszintű triggerek nem adatmanipulációs műveletek, hanem rendszerszintű műveletek esetén aktiválódnak, mint adatbáziselem létrehozása (`CREATE`), módosítása (`ALTER`) és törlése (`DROP`) vagy például felhasználó belépése (`LOGON`), illetve kilépése (`LOGOFF`). Lássunk ezekre is egy példát, amikor a belépési időpontokat naplózzuk!

16.2.1. példa

Tekintsük az alábbi relációsémát!

BelepesNaplo(felhasznalo, idopont)

Hozzunk létre egy olyan triggert, amely naplózza a bejelentkezéseket. A bejelentkezést, mint eseményt az LOGON kulcsszóval tudjuk kezelni.

Bejelentkezések naplózása

```
CREATE TRIGGER bejelentkezes
AFTER LOGON ON CURRENT_USER.schema
  INSERT INTO BelepesNaplo VALUES
     (CURRENT_USER, CURRENT_TIMESTAMP);
```

Fontos megjegyezni, hogy a belépést naplózó triggerek mindig a művelet után (AFTER) kell, hogy lefussanak (mert akkor sikerült a belépés és akkor kap a CURRENT_USER értéket), valamint a kijelentkezést naplózó triggerek mindig a kijelentkezés előtt (BEFORE) kell, hogy lefussanak, mert akkor van még értéke a CURRENT_USER változónak.

Kérdések és feladatok

1. Tekintsük az alábbi adatbázist!
   Keszlet(cikkszam, megnevezes, darabszam, ar)
   Eladas(*cikkszam*,idopont,darabszam, osszertek)
   Készítsen triggert, amely kiszámolja az Eladas táblába az eladott áru összértékét egy-egy vásárlás során!

2. Kiválthatja-e egy trigger egy másik trigger aktiválását? Ha igen, hogyan?

3. Írjon SQL utasítást, amellyel letiltja a bejelentkezes trigger működését!

4. Írjon SQL utasítást, amellyel engedélyezi a bejelentkezes trigger működését!

5. Igaz-e, hogy egy trigger egyszerre több táblára vonatkozó utasításra is aktiválódhat?

17. fejezet

Jogosultságkezelés

A jogosultságkezelés egy nagyon fontos témakör az adatbázisok tekintetében, ugyanis ennek segítségével tudjuk megszabni, hogy egy-egy adatbáziselemet (adatbázist, táblát) mely felhasználók láthatnak, kik azok, akik létrehozhatnak adatbázist, táblát, vagy nézettáblát, kik azok, akik egy tábla tartalmát módosíthatják, törölhetnek rekordot, vagy éppen új rekordot szúrhatnak be. Az Olvasó a fejezet végére megtanulja hogyan tudja kezelni a jogosultságokat SQL utasításokkal, és felelősségteljesen tudja majd alkalmazni ezeket az ismereteket munkája során.

17.1. Jogosultságok adományozása és elvétele SQL-ben

Számos műveletre adhatunk jogot egy felhasználónak adatbáziselemenként (például adatbázisonként, táblánként, soronként, nézettáblákra vonatkozóan). Ilyen műveletek az adatbázist vagy táblát létrehozó műveletek. Ezek látszólag nem „veszélyes” műveletek, bár léteznek olyan adatbázis-támadási módszerek, amelyek saját külön táblát hoznak létre, hogy az adatbázisbeli adatokat ellopják. Egy másik kategória az adatkezelő műveletek köre (beszúrás, módosítás, törlés), amelyet szintén meg kell fontolni, hogy mely felhasználóknak adunk. Az adatok lekérdezésének lehetősége is szabályozható jogosultságokkal.

Ennyi bevezető után, már képet kaptunk ennek a műveletnek a fontosságáról. Lássuk, hogy hogyan tudunk jogokat adni bizonyos műveletekhez.

A `GRANT` utasítás

```
GRANT műveletnév ON adatbáziselem TO {felhasználónév,
   PUBLIC, szerepkör} [WITH GRANT OPTION];
```

A fenti utasításban:

műveletnév: A művelet neve, például `INSERT INTO`, `SELECT`, `UPDATE`, `DELETE FROM`, ...

{felhasználónév, `PUBLIC`, szerepkör}:

- fehasználónév esetén csak az adott felhasználó kap jogot,
- `PUBLIC` esetén mindenki jogot kap az adott műveletre az adatbáziselemen
- szerepkör esetében pedig az adott szerepkörhöz, csoportokhoz tartozó felhasználók kapnak jogot.

`WITH GRANT OPTION`: Amennyiben valakinek jogot adunk az adatbázisműveletre, ő tovább tudja osztani ezt a jogot más felhasználóknak.

A jogosultságok megvonása a `REVOKE` utasítással történik.

A `REVOKE` utasítás

```
REVOKE műveletnév ON adatbáziselem FROM
   {felhasználónév, PUBLIC, szerepkör} [CASCADE];
```

Amennyiben a `CASCADE` kulcsszó szerepel az utasításban, akkor a jogot nem csupán a megjelölt felhasználótól vonjuk meg, hanem azoktól is akiknek, azt továbbadományozták. A jogosultságkezelés kapcsán az adatbáziskezelő rendszerek egy jogosultsági gráfot tartanak nyilván, amelynek csúcspontjai olyan hármasok, amelyek tartalmazzák, hogy ki, milyen joggal rendelkezik egy adatbáziselemen. A gráf élei a jogok továbbadományozását jelentik. Például az F_1 felhasználó az A_1 elemre vonatkozó J_1 jogot továbbadja F_2-nek, akkor a gráfban megjelenik egy $(F_1, A_1, J_1) \longrightarrow (F_2, A_1, J_1)$ él.

17.1.1. példa

Tekintsük a `SZEMÉLY (adószám, név, lakcím, fizetés, egészségi_állapot)` táblát! Adjunk lekérdezési jogot `gnemeth` felhasználónak!

Lekérdezési jog adományozása

```
GRANT SELECT ON szemely TO gnemeth;
```

Látjuk, hogy ő nem tudja továbbadni a lekérdezési jogosultságát, mivel nincs az utasításban a `WITH GRANT OPTION`. Most vonjuk meg tőle ezt a jogot.

Lekérdezési jog megvonása

```
REVOKE SELECT ON szemely FROM gnemeth;
```

17.2. Adatok elkülönítése nézettáblákkal és jogosultságokkal

A 14.3.1. példában láttuk, hogy egy fizikai táblából el tudunk rejteni oszlopokat nézettáblák segítségével, vagy akár meg is fordíthatjuk a nézőpontot, ki tudunk választani oszlopokat a táblákból nézettáblák segítségével. Azt is tárgyaltuk, hogy ezáltal el tudunk rejteni bizonyos adatokat, és az egyes munkatársak számára csak olyan adatokat mutatunk, amelyekkel dolgozniuk kell.

17.2.1. példa

Tekintsük a `DOLGOZO` táblát az alábbi séma szerint!

`DOLGOZO(azonosito, nev, lakcim, fizetes, munkakor, beosztas, betegseg, betegseg_mikortol)`

Hozzuk létre a nézettáblákat, mint a 14.3.1. példában!

Személyes adatok

```
CREATE VIEW szemelyes_adatok AS SELECT
   azonosito, nev, lakcim FROM dolgozo;
```

Orvosi adatok

```
CREATE VIEW orvosi_adatok AS SELECT azonosito,
   nev, lakcim, betegseg, betegseg_mikortol
   FROM dolgozo;
```

Munkaügyi adatatok

```
CREATE VIEW munkaugyi_adatok AS SELECT
   azonosito, nev, lakcim, munkakor, fizetes,
   beosztas FROM dolgozo;
```

Tegyük fel, hogy van három felhasználói csoportunk: `orvos`, `munkaügy`, `szemelyes`.
Vonjuk meg a jogokat a DOLGOZO táblától a három csoport számára és adjunk lekérdezési jogokat a megfelelő csoportoknak a megfelelő táblához!

Minden jog megvonása

```
REVOKE ALL PRIVILEGES ON dolgozo FROM orvos;
REVOKE ALL PRIVILEGES ON dolgozo FROM munkaugy;
REVOKE ALL PRIVILEGES ON dolgozo FROM szemelyes;

GRANT SELECT ON orvosAdatok TO orvos;
GRANT SELECT ON munkaugyiAdatok TO munkaugy;
GRANT SELECT ON szemelyesAdatok TO szemelyes;
```

Megjegyzés: *Az* `ALL PRIVILEGES` *kulcsszó minden jogra vonatkozik.*

Most nézzünk egy példát a sorok kiválasztására és a horizontális jogosultságkezelésre! Elevenítsük fel ehhez a 14.3.2. példát!

17.2.2. példa

Tekintsük ismét a DOLGOZÓ táblát az alábbi séma szerint!

DOLGOZO(azonosito, nev, lakcim, fizetes, munkakor, beosztas, betegseg, betegseg_mikortol)

Készítsünk kimutatást kifizetésekről beosztások szerint!

Éves kimutatás a 2017-es évre beosztasok szerint

```
CREATE VIEW fizetesek2017 (beosztas,
   evi_osszeg) AS SELECT beosztas, SUM(fizetes)
   FROM dolgozo GROUP BY beosztas;
```

Most vonjunk meg módosításra és törlésre vonatkozó jogokat a dolgozó tábláról!

Törlés és módosítás jogosultságok megvonása

```
REVOKE UPDATE ON dolgozo FROM orvos;
REVOKE UPDATE ON dolgozo FROM munkaugy;
REVOKE UPDATE ON dolgozo FROM szemelyes;
REVOKE DELETE ON dolgozo FROM orvos;
REVOKE DELETE ON dolgozo FROM munkaugy;
REVOKE DELETE ON dolgozo FROM szemelyes;
```

Adjunk lekérdezési jogot a `munkaugy` csoportnak!

Lekérdezési jog adományozása

```
GRANT SELECT ON fizetesek2017 TO munkaugy;
```

Kérdések és feladatok

1. Mit takar az `ALL PRIVILEGES` kifejezés a `GRANT` utasításban?
2. Miért fontos a jogosultságok beállítása?
3. Hogyan vonjuk meg a továbbadományozott jogokat, ha nem tudjuk kinek adta tovább egy felhasználó?
4. Gondolja végig, hogy egy tanulmányi rendszerben (például ETR, Neptun, CooSpace) hol tudná hasznosítani a jogosultságkezelést!
5. Lehet-e jogokat adni olyan adatbáziselemre, amit nem én hoztam létre?

Összefoglalás

Ebben a fejezetben áttekintettük az SQL lekérdező nyelv fontosabb parancsait és kulcsszavait, függvényeit, valamint a példák kapcsán begyakoroltuk használatukat. Ismereteinket az adatbázisok létrehozásától építettük fel a táblák létrehozásán, és a táblaszerkezet módosításán keresztül az adatkezelésig, ideértve a lekérdezéseket is. A nézettáblák kapcsán megismertük a származtatott adatok könnyű és gyors kiszámítását, valamint szót ejtettünk a lekérdezések gyorsításáról is. A lekérdezések gyorsítását illetően megtanultuk az indexek megoldási módjait, megértettük hogyan segítik a keresést és a rendezést. A triggerek kapcsán olyan aktív elemekkel ismerkedtünk meg, amelyek segítségünkre lehetnek bizonyos folyamatok és műveletek automatizálásában. Ezen ismeretek és példák segítsével az Olvasó kellő magabiztossággal fogja tudni ellátni a gyakori adatkezelési feladatokat munkája során. Nagyon fontos szem előtt tartani az adatokra vonatkozó jogosultságok kezelését is, amelyeket nézettáblákkal könnyedén megtehetünk.

III. rész

Alkalmazások fejlesztése relációs adatbázisokhoz

18. fejezet

Adatbázis-kezelő rendszerek

Az előzőekben már megismertük az SQL nyelvet. Ebben a részben két adatbázis-kezelő rendszerrel ismerkedünk meg. A következőkben az Olvasó megismeri az adatbázis-kezelő rendszerek főbb tulajdonságait és szolgáltatásait. Meg tudja különböztetni ezek alapján az adatbázis-kezelő rendszereket más rendszerektől, amelyek ugyan képesek egyenként relációs adatbázisokat kezelni, de szolgáltatásaik elmaradnak a valódi adatbázis-kezelő rendszerekétől. A következő fejezetekben az Olvasó megtanulhatja és a példák elvégézése után készségszinten alkalmazhatja az itt bemutatott adatbázis-kezelő rendszereket.

18.1. Az adatbázis-kezelő rendszerek tulajdonságai és szolgáltatásai

Egy adatbázis-kezelő rendszernek az alábbi szolgáltatásokat kell nyújtania:

Maradandó tárolás: Nagy mennyiségű adatot, hosszú ideig képes tárolni. A tárolás mellett fontos kritérium az is, hogy a felhasználók a keresett adatokhoz gyorsan hozzáférjenek. Ezt az adatbázis-kezelő rendszerek speciális adatszerkezetekkel biztosítják.

Programozási felület: Az adatbázis-kezelő rendszerek lehetőséget biztosítanak az adatok külső lekérdező nyelvvel történő elérésére, írására, olvasására.

Tranzakciókezelés: Az adatbázis-kezelő rendszerek támogatják az adatok konkurens elérését különböző folyamatok számára a nemkívánatos következmények elkerülésével. Az adatbázis-kezelő rendszerek támogatják az elkülönítést, vagyis azt a látszatot, hogy a rendszer egyszerre

csak egy tranzakciót hajt végre, valójában bizonyos tranzakciók párhuzamosan is végrehajthatók. Az atomiság azt jelenti, hogy egy adott tranzakciót vagy teljes egészében végrehajtunk vagy nem.

18.2. Az adatbázis-kezelő rendszerek típusai

Tananyagunk a relációs adatbázisokkal foglalkozik, azonban itt megragadjuk a lehetőséget, hogy a felsorolás szintjén bemutassuk a leggyakoribb adatbázis-típusokat.

Relációs adatbázisok: A relációs adatbázisok táblákban tárolják az adatokat. A táblák sorai az egyes egyedpéldányokat, amelyeket rekordoknak hívunk, oszlopai pedig azok tulajdonságait jelölik. A különböző táblákban lévő sorok között valamilyen relációt, kapcsolatot hozunk létre. Ennek talán legelterjedtebb módja a külső kulcsos kapcsolat, de sorazonosítókkal is hivatkozhatunk egyes rekordokra.

Kulcs-érték tárolók: Ezek az adatbázisok az adatokat kulcs-érték párokban tárolják. Adatszerkezetük az asszociatív tömbökre, szótárakra vagy tördelőtáblákra hasonlít. A tárolt érték lehet összetett adattípusú, amely külön önálló mezőket tartalmaz. Ebbe a típusba tartozik a Redis[1], illetve az Oracle NoSQL Database is[2].

Dokumentumtárolók: Ezek az adatbázisok dokumentumszerű adatokat tárolnak. Minden rekord egy-egy dokumentum. Gyakran JSON, YAML, XML, BSON formátumban tárolják az adatot, ilyenformán önleíró adatstruktúrát definiálnak. Ebbe a típusba tartozik pl. a MongoDB[3] adatbázis-kezelő rendszer is.

Gráfadatbázisok: A gráfadatbázisok az adatokat és a közöttük lévő kapcsolatot gráfszerűen tárolják. A gráfok csúcsai az egyedek, amelyek gyakran önleíró adatstruktúrával rendelkeznek, az élek pedig a közöttük lévő kapcsolatokat mutatják. Gyakran használják ezeket az adatbázisokat térinformatikai rendszerekben. Ebbe a típusba tartozik a Neo4J[4], az Oracle Spatial and Graph Database[5].

[1] `https://redis.io/`
[2] `https://www.oracle.com/database/technologies/related/nosql.html`
[3] `https://www.mongodb.com/`
[4] `https://neo4j.com/`
[5] `https://www.oracle.com/database/technologies/spatialandgraph.html`

19. fejezet

A MySQL adatbáziskezelő rendszer

A MySQL kettős licencszű és jelenleg is elérhető egy ingyenes és egy kereskedelmi változat[1]. Az ingyenes változatot jelenleg MySQL Community Edition néven érhető el, amely nyílt forráskódú, GPL licensszel rendelkezik, és egy hatalmas fejlesztői közösség áll mögötte. Számos olyan funkcióval rendelkezik, amely ki tud szolgálni egy kis- illetve közepes adatbázissal rendelkező rendszert. A kereskedelmi változatok, (*MySQL Standard Edition* és *MySQL Enterprise Edition*) kiegészített szolgáltatásokkal is rendelkeznek.

A MySQL ingyenes változatát foglalkozunk. Először áttekintjük a főbb alkalmazásokat, azután látni fogunk egy-egy példát használatukra.

A MySQL-ben történő elmélyüléshez a [4,16,23,25,27] könyveket ajánljuk.

`mysqld:` A MySQL szerver a `mysqld` utasítással indítható. Az adatbázis szerver, mint szolgáltatás, háttéralkalmazásként futtatható.

`mysql:` A MySQL parancssoros kliens programja.

`mysqladmin:` Kliens program adatbázis adminisztrátoroknak.

`mysqldump:` A MySQL adatbázisok kimentésére szolgál. Például biztonsági mentés esetén használható.

`mysqlimport:` Adatbázisok importálására szolgáló kliens program.

`mysqlshow:` Adatbázisok megmutatására (listázására) és szerkezeti felépítésének kiíratására szolgáló program.

[1] Lásd: https://www.mysql.com/products/

19.1. A szerver futtatása

A szervert a `mysqld` programmal indítjuk el. Az egyes kiadásokban a szervernek lehetnek különböző változatai is, amelyek önálló programként is megjelennek (pl. `mysqld-debug` vagy `mysqlds_safe`). A `-help` kapcsolóval le tudjuk kérdezni az egyes opciókat, amelyeket be is állíthatunk.

```
> mysqld --verbose --help
```

19.2. A parancssoros kliens futtatása

A parancsoros klienst a `mysql` programmal tudjuk futtatni. A paramétereknél az alábbi módon lehet megadni a hosztot, az adatbázist és a felhasználói nevet[1]:

```
> mysql -h <hoszt> <adatbázis> -u <felhasználónév> -p
```

Ezt követően a rendszer bekéri a jelszót, és ha jól adtuk meg, akkor megjelenik a MySQL prompt, ahová MySQL parancsokat írhatunk:

```
MYSQL>
```

Ha nem adtunk meg adatbázisnevet, akkor nem nyit meg adatbázist. A MySQL promptban listázzuk ki az elérhető adatbázisokat a `SHOW DATABASES` paranccsal, majd válasszuk ki az adatbázist! Attól függően, hogy milyen felhasználóval jelentkeztünk be, az elérhető adatbázisok listája különbözhet (hiszen lehet, hogy a felhasználónk nem fér hozzá minden adatbázishoz).

```
MYSQL> SHOW DATABASES;
...
MYSQL> USE <adatbázisnév>;
```

A kliens programba írhatunk SQL utasításokat és MySQL utasításokat egyaránt.

[1] A paramétereket <paraméter> formában jelöljük.

19.3. Az adminisztrátori kliens

Az adminisztrátori kliens program a `mysqladmin` paranccsal futtatható. Nem szükséges a kliens felületre belépnünk, parancsokat paraméterként is megadhatunk. Az alábbi parancsnál a lehetséges elemeket szögletes zárójellel jelöljük.

```
> mysqladmin [opciók] <parancs> [argumentumok]
   ↪ [<parancs> [argumentumok]]
```

A részletes leírásért javasoljuk, hogy az aktuálisan használt változat hivatalos dokumentációjának megtekintését, de néhány parancsot it is felsorolunk:

`create <adatbázisnév>:` Új adatbázis létrehozása.

`debug:` Utasítás a szervernek, hogy írjon információkat az error log fájlba.

`drop <adatbázisnév>:` Adatbázis törlése.

`extended-status:` Kiírja a szerver állapotváltozóit és azok értékeit.

`flush-hosts:` Kiüríti az információkat a hoszt cache-ből.

`flush-logs [<log-típus>]:` Kiürít minden naplót.

`flush-privileges:` Újratölti a jogosultsági táblákat.

`flush-status:` Kiüríti az állapotváltozókat.

19.4. Adatbázisok kimentése

Az adatbázisok kimentésére a `mysqldump` parancs használható. Az adatbázisokat leggyakrabban egy SQL utasításokat tartalmazó `.sql` kiterjesztésű szöveges fájlba mentjük ki, mivel a célunk az, hogy az adatbázis szerkezetét, illetve a táblák tartalmát vissza tudjuk állítani a későbbiekben (például egy nem kívánt adatvesztés esetén). Természetesen nem ez az egyetlen formátum, amiben menthetünk. Az `.sql` kiterjesztésű fájl nem csupán biztonsági mentésre szolgál. Ebben a fájlformátumban ugyanis az adatbázist – bizonyos megszorítások mellett – hordozhatóvá tesszük, így például másik (MySQL) rendszerre is át tudjuk vinni.

A `mysqldump` parancs használata a következő:

```
> mysqldump [<kapcsolók>] > dump.sql
> mysqldump [<kapcsolók>] --result-file=dump.sql
```

Számos kapcsolója közül most az általunk legfontosabbnak vélteket soroljuk fel:

`-databases <adatbázis>`: Kimenti a paraméterként megadott adatbázist.

`-all-databases`: Kimenti az összes adatbázis összes tábláját.

`-no-create-db`: Nem hozza létre a `CREATE DATABASE` utasítást.

`-xml`: XML formátumban menti ki az adatbázist. Praktikus funkció, ha a későbbiekben az adatot XML formátumban dolgozzuk fel.

19.5. Adatbázisok importálása

A kimentett adatbázisok importálása a `mysqlimport` parancsot használhatjuk. Az importálásnál a `.sql` fájlban lévő SQL utasítások sorozatát fogja értelmezni és végrehajtani a program. Lássuk a `mysqlimport` parancs használatát!

```
> mysqlimport [<kapcsolók>] <adatbázisnév> <szöveges
   ↪ állomány> [<szöveges  állomány> ... ]}
```

A paraméterekben az `<adatbázisnév>` az adatbázis nevét jelenti. Ez azért fontos, hogy a beolvasáskor a rendszer tudja, hogy az adatokat melyik adatbázisba mentse (nem biztos, hogy a `.sql` fájl tartalmaz `CREATE DATABASE` sort).

Számos kapcsolója közül most csak néhányat tárgyalunk:

`-delete`: Kiüríti a táblákat, mielőtt importálja a sorokat a szöveges fájlból. Azért fontos, hogy ha már létezik az importálandó tábla, és tartalmaz sorokat is, akkor ne szúrjunk be ismét meglévő sorokat.

`-ignore` **és** `-replace`: Ezek a kapcsolók azokat a input sorokat kezelik, amelyek már meglévő kulcs-tartalmú (`unique`) adatmezőket is tartalmaznak.

19.6. Adatbázisok tartalmának megtekintése

Az adatbázisok tartalmát gyorsan meg tudjuk nézni a `mysqlshow` programmal. Ezzel nem csupán egy konkrét adatbázis tartalmát lehet megnézni, hanem lehet keresni is, illetve használhatóak a helyettesítő karakterek is az adatbázisok vagy a táblák neveiben. A program a következő módon használható:

```
> mysqlshow [<kapcsolók>] [<adatbázisnév [<táblanév>
   ↪ [<oszlopnév>]]]
```

A napi munkahelyi rutin során ritkán van erre a parancsra szükség, mivel ha adatbázisokkal dolgozunk, akkor többnyire belépünk a szerverre és ottani MySQL utasításokkal listázzuk ki az adatbázisokat, oszlopokat.

19.7. Csatlakozás MySQL-hez PHP-vel

Ebben az alfejezetben megnézzük, hogyan csatlakozhatunk MySQL szerverhez PHP-ból. Itt csupán a csatlakozás módját nézzük meg, rendszerfejlesztéssel későbbi fejezetben foglalkozunk. Tananyagunkban a PHP 7-es verzióját használjuk. Tekintettel arra, hogy a PHP nyelv is folyamatosan fejlődik, lehetséges, hogy az itt leírt kapcsolók, paraméterek és függvénynevek idővel megváltoznak a PHP újabb verzióiban, ezért javasoljuk, hogy hiba esetén először mindig a hibaüzenetet és a leírt függvényt, valamint a PHP referencia-kézikönyvet nézzék meg! Megjegyezzük továbbá, hogy itt a procedurális változatot mutatjuk be, de a PHP-ben objektum-orientált módon is használhatjuk a függvényeket.

A csatlakozáshoz a `mysqli_connect(hoszt, felhasználónév, jelszó)` függvény használható. A függvény egy csatlakozási azonosítót ad vissza amennyiben sikerült csatlakozni, ellenkező esetben pedig NULL értékkel tér vissza. A csatlakozást követően kiválasztjuk a használni kívánt adatbázist a `mysqli_select_db(csatlakozás azonosító,adatbázisnév)` függvénnyel. A függvény első paraméterének a `mysqli_connect(...)` által visszaadott csatlakozási azonosítót kell megadni, majd az adatbázis nevét. Amennyiben sikerült kiválasztani az adatbázist, a függvény `TRUE` értéket, ellenkező esetben `FALSE` értéket ad vissza.

Az adatbázis használata után a kapcsolatot a `mysqli_close(csatlakozás azonosító)` függvénnyel tudjuk lezárni. Fontosnak tartjuk megjegyezni, hogy az alkalmazások kapcsán mindig csak addig tartsuk nyitva az adatbázis-kapcsolatot, amíg valóban szükség van rá, a lehető legrövidebb ideig. Ez azért

fontos, mert a MySQL szerver csak korlátos számú felhasználót tud kiszolgálni egy időben. Ha hosszan tartjuk nyitva az adatbázis-kapcsolatot (amikor nincs is szükségünk rá), akkor azzal gyakorlatilag foglaljuk a rendszert és ezzel lassítjuk működését.

Az adatbázis-kapcsolathoz ennyi ismeret elég, részletesebben a PHP-val történő alkalmazásfejlesztésnél fogunk részletesebben megismerkedni a további függvényekkel és lehetőségekkel.

Az alábbiakban egy példán keresztül bemutatjuk a csatlakozás menetét. A példa megértéséhez annyit még megjegyzünk, hogy a PHP nyelvben a változóneveket $ karakterrel kezdjük és a szövegkonstansokhoz az egyszeres és a dupla idézőjeket is használhatjuk. A példában használt `die(...)` függvény hibaüzenetet ír ki a képernyőre. A `mysqli_prepare()` függvénnyel utasításokat tudunk előkészíteni, amelyet majd a `mysqli_stmt_execute()` függvénnyel futtatunk. Erről később részletesebben is lesz szó.

Csatlakozás adatbázishoz

```
1
2  $db = mysqli_connect('localhost','root','')
3    or die('Nem sikerült csatlakozni az szerverhez.');
4
5  $dbOK = mysqli_select_db($db, 'adatbazisom')
6    or die('Nem sikerült csatlakozni az adatbázishoz.');
7
8  // itt használjuk az adatbázist: kiíratjuk az adatbá
     ↪ zis tábláit
9  $utasitas = mysqli_prepare($db, 'SHOW TABLES');
10 mysqli_stmt_execute($utasitas); // végrehajtjuk az
     ↪ utasítást
11 // ... kilistázzuk a táblák neveit ...
12
13 // Lezárjuk az adatbázis-kapcsolatot
14 mysql_close($db);
```

19.8. Csatlakozás MySQL adatbázishoz Javaból

Ebben az alfejezetben azt tekintjük át, hogyan csatlakozunk MySQL adatbázishoz egy Java alkalmazásból. Ehhez a Java Database Connector osztálykönyvtárat, röviden JDBC-t használjuk, amelyet a MySQL oldaláról is letölthetünk. Fontos lesz megjegyeznünk, hogy a Java program fordításánál

ezt a függvénykönyvtárat is meg kell jelölnünk a `CLASSPATH`-ban. A Java nyelv szintaxisát itt sem részletezzük, ezeket az ismereteket számos magyar és angol nyelvű szakkönyvből [2, 17, 22] és más kurzusról[1] megszerezhetjük.

A Java nyelvben a JDBC-vel csatlakozunk a MySQL adatbázishoz. Első lépésben regisztráljuk a driver-t a programban a `Class.forName("com.myslq.jdbc.Driver");` statikus metódushívással. Ezt követően a `DriverManager.getConnection(url,felhasználónév, jelszó)` metódussal létrehozunk egy `Connection` objektumpéldányt. A metódus paraméterlistájában az `url` az adatbázis-szerver elérési útvonalát jelöli, amely tartalmazza csatlakozási portot és az adatbázis nevét is, a `felhasználónév` és `jelszó` pedig a bejelentkezéshez szükséges azonosítók. Abban az esetben, ha nem sikerült a csatlakozás, a `Connection` objektumpéldány helyett `null` érték lesz a visszatérési érték. Az adatbázis kapcsolatot a `Connection` objektum `close()` metódusával zárjuk le. Itt is megemlítjük, hogy törekedjünk az adatbázis-kapcsolatot csak addig nyitva tartani, ameddig tényleg szükségünk van rá! Ebben az alfejezetben nem részletezzük külön, de a `Connection.prepareStatement()` metódussal utasítást tudunk előkészíteni, amelyet majd a `PrepareStatement.executeQuery()` metódussal hajtunk végre.

Az alábbi kódrészlet mutatja be, hogyan csatlakozunk egy Java nyelvű programból JDBC-vel MySQL-hez. A kódrészlet után bemutatjuk a program fordítását és futtatását is.

Csatlakozás MySQL adatbázishoz JDBC-vel

```
1  // ... Osztály deklaráció, metódus deklaráció, stb.
2
3  // A driver regisztrálása
4  Class.forName("com.mysql.jdbc.Driver");
5
6  // Csatlakozás a localhost-on lévő pelda adatbázishoz a 3306
       ↪ porton
7  String url = "jdbc:mysql://localhost:3306/pelda"
8  Connection con = DriverManager.getConnection(url, "root", "");
9
10 // Előkészítjük az utasítást, pl. kiíratjuk a táblákat
11 PreparedStatement utasitas = con.prepareStatement("SHOW TABLES");
12 ResultSet eredmeny = utasitas.executeQuery();
13 // ... kilistázzuk a táblák neveit ...
14
```

[1] A Szegedi Tudományegyetem Informatikai Intézeténél folyó informatikus képzésében az *Adatbázisok* kurzust megelőzi a Programozás I. kurzus, amelyen a hallgatók megismerkednek a Java nyelv szintaxisával és az objektum-orientált programozással.

```
15 // Lezárjuk az adatbáziskapcsolatot
16 con.close();
```

Fordítás és futtatás:

```
> javac -cp .;mysql-mysql-connector-java-5.0.8-bin.jar
   ↪  Pelda.java
> java -cp .;mysql-mysql-connector-java-5.0.8-bin.jar
   ↪ Pelda
```

19.9. Csatlakozás ODBC-vel MySQL adatbázishoz

Az Open DataBase Connectivity, röviden ODBC egy szabványos, nyílt, C nyelvű csatlakozási programozói függvénykönyvtár. Az ide vonatkozó adattípusok és függvények az `sql.h`-ban találhatóak. Részben igaz, hogy ezzel a függvénykönyvtárral egy adott program különböző adatbáziskezelő rendszerekkel használható, viszont ezen rendszerek SQL szintaxisa eltérhet. Az ODBC-ben a környezeteket és kezelőket hierarchikusan építjük egymásra. Ez hasonlít a JDBC-hez, de itt a strukturáltság jobban megfigyelhető. A hierarchiát az alábbi módon építjük fel:

1. Környezet (*Environment*), ez a legmagasabb szint, a kliens hozza létre az adatbáziskapcsolat előkészítéséhez.

2. Adatbázis kapcsolat (*Connection*), ez a második szint, amely egy környezethez kapcsolódik. Egy környezethez több adatbázis-kapcsolat is létrehozható.

3. Utasítás (*Statement*), ez a harmadik szint, amely egy adatbázis-kapcsolathoz kötődik. Egy kapcsolathoz több utasítás létrehozható. Az utasításokat elő lehet készíteni, illetve lehet közvetlenül is futtatni.

A JDBC-ben a `PreparedStatement` is tulajdonképpen az utasításokat reprezentálja.

E szintek kezelése külön kezelők (*handle*-k, adatstruktúrákra mutató pointerek) segítségével történik:

- Környezet: `SQLHENV`

- Kapcsolat: `SQLHDBC`
- Utasítás: `SQLHSTMT`

Az egyes kezelők létrehozására az `SQLAllocHandle(hType, hIn, hOut)` függvény szolgál, amelyben az első paraméter (`hType`) a kezelő típusa, a második paraméter a kezelő bemenetét, illetve feljebb lévő szintjét jelző pointer, az utolsó paraméter pedig az a cím, ahová az új kezelőt létrehozzuk. A környezetnek nincs feljebb lévő szintje, ezért az ő bemenetére az `SQL_NULL_HANDLE`-t írjuk. Ebben az alfejezetben is csak az adatbázis-kapcsolat létrehozására szorítkozunk, az adatkezelést nem tárgyaljuk.

Az alábbi kódrészlet bemutatja ezek használatát:

Csatlakozás MySQL-hez ODBC-vel

```
/* letrehozzuk a szukseges valtozokat */
SQLHENV env;
SQLHDBC dbc;
SQLHSTMT stmt;
SQLRETURN ret;

/*
  beallitjuk a kornyezeti valtozokat,
  az alapveto mukodeshez szukseges
*/
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION,
    (void *) SQL_OV_ODBC3, 0);
SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);

/* megnyitjuk a kapcsolatot az SQLDriverConnect utasitassal*/
ret = SQLDriverConnect(dbc, NULL,
  (unsigned char *)"DRIVER={MySQL ODBC 3.51 Driver};
  SERVER=localhost;DATABASE=pelda;",
  SQL_NTS, NULL, 0, NULL, SQL_DRIVER_COMPLETE);

/* ellenorizzuk, hogy a kapcsolat megnyitasa sikeres volt-e */
if (SQL_SUCCEEDED(ret)) {
  printf("Connected\n");
} else {
  printf("Failed to connect\n");
  exit(-1);
}

SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

/* letrehozzuk az utasitast tartalmazo stringet*/
```

```
33 SQLCHAR *text = (SQLCHAR *)"SELECT szam,szoveg from varosok";
34
35 /* elokeszitjuk, majd vegrehajtjuk az utasitast*/
36 SQLPrepare(stmt, text, SQL_NTS);
37 ret = SQLExecute(stmt);
38
```

Megfigyelhetjük, hogy a fenti kódban az `SQLPrepare(...)` függvénnyel készítjük elő az utasítást a `stmt` utasításkezelőnek. Ez az utasításkezelő az `SQLPrepare(...)` függvényhívás után a neki megadott SQL utasítást (jelen esetben az `SELECT szam, szoveg FROM varosok;` utasítást) reprezentálja. Az `SQLPrepare(...)` függvényhívás még nem futtatja le az utasítást, csupán előkészíti. Az utasítást az `SQLExecute(...)` függvény fogja lefuttatni.

Kérdések és feladatok

1. Melyek a fontosabb segédprogramok a MySQL-hez?
2. Mire és hogyan használható a `mysqldump` segédprogram?
3. Hogyan lehet MySQL-be adatot importálni külső adatfájlból?
4. Hogyan lehet kilistázni MySQL-ben az adatbázisokat?
5. Mire lehet használni a `.sql` fájlokat?

20. fejezet

Az SQLite adatbázis-kezelő rendszer

Az SQLite egy viszonylag új, nyílt forráskódú adatbázis-kezelő rendszer és függvénykönyvtár. 2000. május 9-én jelent meg az első verziója. Kis mérete miatt nagyon gyorsan elterjedt, olyannyira, hogy az Android-os telefonokban is ez a rendszer található meg. Ebben a fejezetben röviden bemutatjuk az SQLite adatbáziskezelő rendszert és annak főbb funkcióit, tulajdonságait. Nem részletezzük azokat a nyelvi elemeket, amelyek a szabványos SQL nyelvben megvannak, inkább a sajátosságokat hangsúlyozzuk ki.

Az adatbázis-kezelő rendszerek esetében általában egy szerver írja és olvassa az adatbázis-állományokat. A szerverhez egy kliens programmal csatlakozunk a szerverhez, és az adatokat hálózati kommunikációval valósítjuk meg.

Az SQLite abban is különbözik például a MySQL-től, hogy nincs külön szerver-kommunikáció, az eljárások ezen függvénykönyvtár segítségével közvetlenül írják és olvassák az adatbázis-fájlokat. Az SQLite az úgynevezett klasszikus szervernélküli adatbázis-kezelő rendszerek közé tartozik, amely azt jelenti, hogy az adatbázis motor ugyanazon processzuson, szálon és címen fut, mint az alkalmazás[1].

Az SQLite-ban minden adatbázishoz egyetlen adatbázisfájl[2] tartozik, amely platform-függetlenné teszi az adatbázist. Ez a fájl – és az adatbázis is – hordozható 32-bites, 64-bites, nagy- és kis-endiánú (*little endian* és *big endian*) architektúrájú gépek között is.

[1] Lásd az SQLite weboldalát: `https://www.sqlite.org/`

[2] A fájlformátum leírása a következő honlapon található: `https://www.sqlite.org/fileformat2.html`

Mikor érdemes SQLite-ot használni?

- beágyazott rendszerek esetében
- alkalmazás fájl formátumoknál (pl. CAD programoknál)
- kis- és közepes forgalmú weboldalaknál
- adatelemzésnél (pl. bioinformatikai kutatásoknál)
- fájl archivumoknál, adattárolóknál
- oktatásban

Mikor jobb egy kliens-szerver adatbázist választani?

- kimondottan kliens-szerver alkalmazásoknál
- nagyméretű adatbázisok esetén
- nagy forgalmú adatbázisoknál (ahol sok az olvasó folyamat, de kevés az író folyamat).
- nagyméretű és nagy forgalmú weboldalaknál, amelyeket több szerver szolgál ki.

20.1. Nyelvi sajátosságok az SQLite-ban

Az alábbiakban áttekintjük az SQLite néhány fontos nyelvi sajátosságát. Mindent felsorolni terjedelmessé tenné a jegyzetet, javasoljuk emiatt, hogy az egyes függvényekkel kapcsolatosan az Olvasó nézze meg az SQLite honlapját[1]!

20.1.1. Adattípusok

Az adattípusokat az SQLite-ban *adattípus-osztályokba* sorolják, amelyek a következők:

`NULL`: A `NULL` érték.

`INTEGER` Előjeles egész érték, amelyet 1,2,3,4,6 vagy 8 bájton tárolnak az értéktől függően.

[1] `https://www.sqlite.org/draft/datatype3.html`

`REAL`: Lebegőpontos szám, amelyet 8-bájtos IEEE lebegőpontos számként tárolnak.

`TEXT`: Szöveg.

`BLOB`: Bináris nagy objektum.

Logikai értékek: Az SQLite nem használ külön logikai értékeket, helyette a 0-ként tárolja a `FALSE` és 1-ként a `TRUE` értéket.

Dátum és idő adattípusok: Az SQLite a dátum- és idő típusokat sem kezeli külön, a megfelelő dátumkezelő függvényekkel `TEXT`, `REAL` vagy `INTEGER` értékekből konvertálja át.

`TEXT`: ISO8601 sztring formában tárolja (ahol a formátumsztring "YYYY-MM-DD HH:MM:SS.SSS").

`REAL`: Kr.e. 4714. november 24. óta eltelt napok száma.

`INTEGER`: Unix-időként, 1970-01-01 00:00:00 óta eltelt másodpercek száma.

Típus affinitás: Az SQLite rugalmasan kezeli a típus konverziókat. Ha a séma definíciójában meg van határozva egy típus, például `INTEGER`, akkor egy sztring konstanst is, ha az számmá alakítható, átalakít számmá. Ez fordítva is igaz, ha számot adunk meg egy sztring helyére, akkor azt sztringgé alakítja. Az SQLite3-ban minden oszlop megfelel az alábbi öt affinitási típus egyikének:

- `TEXT`
- `NUMERIC`
- `INTEGER`
- `REAL`
- `BLOB`

A típus affinitás eldöntésének öt szabálya:

1. Ha a deklarált típus tartalmazza az "INT" karaktersorozatot, akkor `INTEGER` affinitású lesz.
2. Ha a deklarált oszlop típus a tartalmazza a "CHAR", "CLOB" vagy a "TEXT" karaktersorozatok egyikét, akkor `TEXT` affinitású lesz.

3. Ha a deklarált oszlop típusa tartalmazza a "BLOB" karaktersorozatot vagy nincs megjelölve típus, akkor `BLOB` affinitású lesz.

4. Ha a deklarált oszlop típusa tartalmazza a "REAL", "FLOA" vagy "DOUB" karaktersorozatot, akkor `REAL` affinitású lesz.

5. Különben `NUMERIC` affinitású lesz.

Ha több szabály is illeszkedne egy típusra, akkor az előrébb lévő szabály érvényesül. A 20.1. táblázat foglalja össze a típus affinitásokat[1].

20.1.2. Sémák módosítása

A sémák módosítására az `ALTER TABLE` utasítás szolgál, azonban a MySQL-hez képest vannak különbségek.

Táblához könnyedén hozzá tudunk adni új oszlopot.

20.1.1. példa

Adjunk hozzá a `szemelyek` táblához egy új `szulhely` oszlopot!

Új oszlop hozzáadása

```
ALTER TABLE szemelyek ADD szulhely TEXT;
```

A tábla átnezvezésére is használhatjuk az `ALTER TABLE` utasítást.

20.1.2. példa

Nevezzük át a `szemelyek` táblát `szemely`-re.

Tábla átnevezése

```
ALTER TABLE szemelyek RENAME TO szemely;
```

Oszlop(ok) törlése vagy módosítása már bonyolultabb műveletet igényel. Ezt az alábbi listában leírt módon kell végrehajtani.

1. Ha a külső kulcsok meg vannak adva, akkor „le kell kapcsolni” őket: `PRAGMA FOREIGN_KEYS=OFF;`

[1] Forrás: `https://www.sqlite.org/datatype3.html`

Típusnevek	Affinitás	Affinitási szabály
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	1
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	2
BLOB nem definiált adattípus	BLOB	3
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	4
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	5

20.1. táblázat. Affinitási táblázat.

2. Indítsunk új tranzakciót!
   `BEGIN TRANSACTION;`

3. Nevezzük át a jelenlegi táblánkat!
   `ALTER TABLE tabla RENAME TO ideiglenes_tabla;`

4. Hozzunk létre egy új táblát a kívánt sémának megfelelően!
   `CREATE TABLE tabla (oszlop_definiciok);`

5. Másoljunk át mindent az ideiglenes (régi) táblából az újba!

```
INSERT INTO tabla(oszloplista) SELECT oszloplista FROM
ideiglenes_tabla;
```

6. Töröljük a régi táblát!
   `DROP TABLE ideiglenes_tabla;`

7. Zárjuk a tranzakciót!
   `COMMIT;`

8. Kapcsoljuk vissza külső kulcsokat!
   `PRAGMA FOREIGN_KEYS=ON;`

Ne felejtsük el, hogy a régi táblánkra vonatkozó indexeket, a nézettáblákat és a triggereket is át kell állítani, mivel új táblát hoztunk létre!

20.1.3. AUTOINCREMENT vagy ROWID

Az SQLite3-ban létezik az egyedi pozitív egész szám, a sorazonosító vagy ROWID, amely minden sorra egyedi lesz. Minden tábla sorát be kell azonosítanunk valahogy. Az SQLite erről úgy gondoskodik, hogy minden táblához automatikusan hozzárendel még egy oszlopot, a ROWID-t. Amennyiben egy sémában egyetlen elem a kulcs, és azt sorszámnak tekintjük, akkor a következő típust és megszorítást adjuk neki: `INTEGER PRIMARY KEY`, ez az oszlop lesz ROWID másodnévvel (*alias*-szal) elnevezve. Az ilyen attribútumok esetében, ha nem töltjük ki a mező értékét, akkor a rendszer automatikusan a következő sorszámot állítja be rá. Ha ezt el szeretnénk kerülni, akkor a séma definíciójában adjuk meg a `WITHOUT ROWID` megszorítást.

Az `AUTOINCREMENT` kulcszót is használhatjuk, ez azonban többlet memóriát és tárhelyet, valamint olvasási műveletet igényel.

Egy `INSERT` utasításnál érdekes lehet számunkra, hogy mi volt az utoljára létrehozott azonosító. Ezt a következőképpen tudjuk lekérdezni:

Utolsó automatikusan létrehozott azonosító lekérése

```
SELECT last_insert_rowid();
```

20.1.4. Dátum- és időfüggvények

Láttuk korábban, hogy a dátum- és időpont viszonylag rugalmasan adható meg a típus affinitás miatt. A SQLite dátum- és időkezelő függvényei azonban további kényelmi funkciókat is nyújtanak.

A következő függvényekkel lehet megadni a dátum- és idő értékeket:

`date(...)`	dátumot ad meg
`time(...)`	időpontot ad meg
`datetime(...)`	dátumot és időpontot ad meg
`julianday(...)`	a napok számát adja meg a Kr.e. 4714. november 24-hez képest
`strftime(...)`	a C-ben használt függvényhez hasonló[1]

A dátumot formátum-sztringgel adjuk meg, ehhez a 20.2. táblázatban felsorolt betűket használhatjuk.

sztring-elem	jelentés	érték
%d	a hónap napja	00
%f	tört másodperc	SS.SSS
%H	óra	00-24
%j	az év napja	001-366
%J	A Julianus napok száma	
%m	hónap	01-12
%M	perc	00-59
%s	másodperc 1970. január 1. óta	
%S	másodperc	00-59
%w	a hét napja	0-6, ahol 0=vasárnap
%W	a hét száma az évben	00-53
%Y	év	0000-9999
%%	%	

20.2. táblázat. Formátum-sztring elemek dátumokhoz és időkhöz

A dátum- és időfüggvényekben úgynevezett *módosítókat* is megadhatunk, amelyek további lehetőségeket kínálnak számunkra.

±NNN days	NNN nappal előbb, később
±NNN hours	NNN órával előbb, később
±NNN minutes	NNN perccel előbb, később
±NNN.NNNN seconds	NNN.NNNN másodperccel előbb, később
±NNN months	NNN hónappal előbb, később
±NNN years	NNN évvel előbb, később
start of the month	a hónap kezdőnapja
start of the year	az év kezdőnapja
start of the day	a nap kezdete
weekday N	az N. nap (0=vasárnap, 1=hétfő, ...)
unixepoch DDDDDDDDDD	1970 óta eltelt másodpercek száma
localtime	helyi idő
utc	Universal Coordinated Time (UTC)

20.1.3. példa

Tekintsünk néhány példát a dátum- és időkezelő függvényekre!

Aktuális dátum

```
SELECT DATE('now');
```

Az aktuális hónap utolsó napja

```
SELECT DATE('now', 'start of month','+1
   month','-1 day');
```

Az aktuális unix-idő

```
SELECT strftime('%s','now');
```

Milyen napra esik idén október első keddje?

```
SELECT DATE('now','start of year','+9
   months','weekday 2');
```

20.2. A parancssori program

A parancssori program neve `sqlite3` vagy Windows-on `sqlite3.exe`. Ez a program egy SQLite-parancsort indít, amelybe SQL és SQLite utasításokat írhatunk. Hasonlít a MySQL kliens programjához, azonban ne feledjük, hogy a MySQL egy kliens-szerver rendszer, az SQLite viszont nem szerver alapú adatbáziskezelő.

20.2.1. példa

Készítsünk egy **pelda.db** adatbázist! Hozzunk létre egy `szemelyek(nev TEXT, szuldatum DATE)` táblát! Szúrjunk be két sort, majd kérdezzük le a tábla tartalmát!

```
$ sqlite3 pelda.db
sqlite> create table szemelyek(nev TEXT, szulev
   ↪ DATE);
sqlite> insert into szemelyek values   ('Teszt
   ↪ Elek', '1980-12-03');
sqlite> insert into szemelyek values   ('Buda
   ↪ István', '1968-06-12');
sqlite> select * from szemelyek;
Teszt Elek|1980-12-03
Buda István|1968-06-12
sqlite>.quit
```

A fenti példában a `.quit` egy SQLite utasítás. Az SQLite utasításokat a parancssori programban a `.help` utasítás listázza ki. A terjedelem miatt ezeket az utasításokat nem soroljuk fel itt[1].

Nem csak parancssori argumentumként nyithatunk meg adatbázist. Ezt megtehetjük az `.open` SQLite utasítással is.

```
$ sqlite3
SQLite version 3.20.1 2017-08-24 16:21:36
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent
   ↪ database.
sqlite> .open pelda.db
sqlite> select * from szemelyek;
Teszt Elek|1980-12-03
Buda István|1968-06-12
sqlite> update szemelyek SET szulev='1968-07-12' WHERE
   ↪  nev='Buda István';
sqlite> select * from szemelyek;
Teszt Elek|1980-12-03
Buda István|1968-07-12
sqlite> .quit
```

Ha nem nyitottunk meg még adatbázist, de létrehoztunk táblákat és végeztünk rajtuk bizonyos műveleteket, akkor a változásokat a `.save` utasítással tudjuk elmenteni.

A parancsori SQLite utasítások tekintetében fontos még a `.schema` uta-

[1] Az utasítások listája megtalálható a `https://www.sqlite.org/cli.html` honlapon.

sítás, amely az aktuális adatbázis sémáit listázza ki. Nézzük meg, hogyan működik a **Programkalauz** adatbázis esetében, amely három táblát tartalmaz!

```
$ sqlite3 programkalauz.db
SQLite version 3.20.1 2017-08-24 16:21:36
Enter ".help" for usage hints.
sqlite> .schema
CREATE TABLE Helyek (helyazonosito INTEGER PRIMARY KEY,
   ↪ varos TEXT, cim TEXT, hely_neve TEXT);
CREATE TABLE Programok (programazonosito INTEGER PRIMARY
   ↪  KEY, cim TEXT, leiras TEXT, mikortol DATETIME,
   ↪ meddig DATETIME, web TEXT, kapcsolat TEXT,
   ↪ helyazonosito INTEGER REFERENCES Helyek(
   ↪ helyazonosito), artol INTEGER, arig INTEGER);
CREATE TABLE Mufaj(programazonosito INTEGER REFERENCES
   ↪ Programok(programazonosito), mufajmegnevezes TEXT,
   ↪  PRIMARY KEY( programazonosito, mufajmegnevezes));
   ↪
```

20.3. Az SQLite adatbázis-elemző

Az SQLite-hoz tartozik egy `sqlite3_analyzer` parancssoros segédprogram, amely megmutatja, hogy mennyire hatékonyan használja az SQLite a tábláknak és és az indexeknek a tárhelyét. Parancssori argumentumként a vizsgált adatbázis fájl nevét kell megadni.

A fent említett `pelda.db` fájl esetében az alábbi eredményt írta ki a program:

```
$ sqlite3_analyzer pelda.db
/** Disk-Space Utilization Report For pelda.db

Page size in bytes.................. 4096
Pages in the whole file (measured)... 2
Pages in the whole file (calculated). 2
Pages that store data.............. 2
   ↪ 100.0%
Pages on the freelist (per header)... 0
   ↪ 0.0%
Pages on the freelist (calculated)... 0
   ↪ 0.0%
Pages of auto-vacuum overhead........ 0
```

```
   ↪ 0.0%
Number of tables in the database..... 2
Number of indices..................... 0
Number of defined indices............. 0
Number of implied indices............. 0
Size of the file in bytes............. 8192
Bytes of user payload stored......... 48   0.59%

*** Page counts for all tables with their indices
   ↪ *************

SQLITE_MASTER......................... 1     50.0%
SZEMELYEK............................. 1     50.0%

*** Page counts for all tables and indices separately
   ↪ ********

SQLITE_MASTER......................... 1      50.0%
SZEMELYEK............................. 1      50.0%

*** All tables **********************

Percentage of total database......... 100.0%
Number of entries..................... 3
Bytes of storage consumed............. 8192
Bytes of payload...................... 123    1.5%
Bytes of metadata..................... 128    1.6%
Average payload per entry............. 41.00
Average unused bytes per entry........ 2647.00
Average metadata per entry............ 42.67
Maximum payload per entry............. 75
Entries that use overflow............. 0      0.0%
Primary pages used.................... 2
Overflow pages used................... 0
Total pages used...................... 2
Unused bytes on primary pages........ 7941   96.9%
Unused bytes on overflow pages........ 0
Unused bytes on all pages............. 7941   96.9%

*** Table SQLITE_MASTER ****************

Percentage of total database........  50.0%
Number of entries..................... 1
Bytes of storage consumed............. 4096
Bytes of payload...................... 75     1.8%
Bytes of metadata..................... 112    2.7%
```

```
B-tree depth..................... 1
Average payload per entry........ 75.00
Average unused bytes per entry... 3909.00
Average metadata per entry....... 112.00
Maximum payload per entry........ 75
Entries that use overflow........ 0       0.0%
Primary pages used............... 1
Overflow pages used.............. 0
Total pages used................. 1
Unused bytes on primary pages.... 3909    95.4%
Unused bytes on overflow pages... 0
Unused bytes on all pages........ 3909    95.4%
*** Table SZEMELYEK ***************

Percentage of total database.....  50.0%
Number of entries................ 2
Bytes of storage consumed........ 4096
Bytes of payload................. 48      1.2%
Bytes of metadata................ 16      0.39%
B-tree depth..................... 1
Average payload per entry........ 24.00
Average unused bytes per entry... 2016.00
Average metadata per entry....... 8.00
Maximum payload per entry........ 25
Entries that use overflow........ 0       0.0%
Primary pages used............... 1
Overflow pages used.............. 0
Total pages used................. 1
Unused bytes on primary pages.... 4032    98.4%
Unused bytes on overflow pages... 0
Unused bytes on all pages........ 4032    98.4%
```

20.4. SQLite C/C++ interfész

Egy C/C++ programban az alábbi függvényekkel lehet kezelni az SQLite adatbázisokat:

`sqlite3`	Adatbázis csatlakozási adatbázis-objektum, amelyet egy `sqlite3_open()` hoz létre, és egy `sqlite3_close()` szabadít fel.
`sqlite3_open()`	Új kapcsolatot hoz létre, eredménye egy `sqlite3` objektum.

`sqlite3_close()`	Bezár egy megnyitott adatbázis kapcsolatot és megszünteti az `sqlite3` adatbázis-objektumot.
`sqlite3_stmt`	Utasítás-objektum (stuktúra). Az objektumot az `sqlite3_prepare()` hozza létre.
`sqlite3_prepare()`	A paraméterül kapott SQL utasítást lefordítja és előállít egy `sqlite3_stmt` objektumot.
`sqlite3_bind()`	Az `sqlite3_stmt` objektumokhoz köt külső paramétereket. A függvénynek több alakja van, attól függően, hogy milyen paramétert kötünk az utasításba.
`sqlite3_step()`	Ez a függény az `sqlite3_stmt` utasítást értékeli ki (futtatja). Lekérdezés esetén a következő eredménysorra lép.
`sqlite3_column()`	Lekérdezés esetén visszaadja az aktuális sor i-edik oszlopának értékét. Az oszlop számát paraméterként kell megadni. A függvénynek több alakja van. Attól függően kell használni, hogy milyen típusú az oszlopban tárolt adat.
`sqlite3_finalize()`	Az `sqlite3_stmt` objektum destruktora.
`sqlite3_exec()`	Önmagába foglalja az `sqlite3_prepare()`, `sqlite3_step()`, `sqlite3_column()` és `sqlite3_finalize()` utasításokat.

Az alábbi példa mutatja be, hogyan használhatjuk az SQLite függvénykönyvtárat C/C++ programhoz[1].

Csatlakozás az SQLite C/C++ függvénykönyvtárral

```
1  #include <stdio.h>
2  #include <sqlite3.h>
3
4  static int callback(void *NotUsed, int argc, char **argv, char **
     azColName){
5    int i;
6    for(i=0; i<argc; i++){
7      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
8    }
9    printf("\n");
10   return 0;
11 }
12
```

[1] A példa a `https://www.sqlite.org/quickstart.html` honlapról lett átdolgozva.

```
13 int main(int argc, char **argv){
14   sqlite3 *db;
15   char *zErrMsg = 0;
16   int rc;
17
18   if( argc!=2 ){
19     fprintf(stderr, "Hasznalat: %s <adatbazis>\n", argv[0]);
20     return(1);
21   }
22   rc = sqlite3_open(argv[1], &db);
23   if( rc ){
24     fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db
     ));
25     sqlite3_close(db);
26     return(1);
27   }
28   rc = sqlite3_exec(db, "SELECT * FROM szemelyek", callback, 0, &
     zErrMsg);
29   if( rc!=SQLITE_OK ){
30     fprintf(stderr, "SQL error: %s\n", zErrMsg);
31     sqlite3_free(zErrMsg);
32   }
33   sqlite3_close(db);
34   return 0;
35 }
36
```

A program fordítása és futtatása a következőképpen történik:

```
$ gcc -o sqlitepelda -lsqlite3 sqlitepelda.c

$ $ ./sqlitepelda pelda.db
nev = Teszt Elek
szulev = 1980-12-03

nev = Buda István
szulev = 1968-07-12
```

20.5. Csatlakozás SQLite adatbázishoz JDBC-vel

A JDBC-vel történő csatlakozáshoz telepíteni kell az `sqlite-jdbc` programot.

Az alkalmazásban importálni kell a `java.sql.Connection`, `java.sql.DriverManager` és `java.sql.SQLException` csomagokat. A csatlakozást egy Java programból az alábbi példa mutatja be:

Csatlakozás SQLite adatbázishoz JDBC-vel

```
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.SQLException;
4  import java.sql.Statement;
5  import java.sql.ResultSet;
6
7  public class SQLitePelda {
8
9    public static void main(String[] args) {
10     try {
11       // csatlakozas a pelda.db adatbazishoz
12       String url = "jdbc:sqlite:pelda.db";
13       Connection conn = DriverManager.getConnection(url);
14       System.out.println("Sikerült csatlakozni az adatbázishoz!");
15
16       // ... itt dolgozunk az adatbazissal ...
17       String query = "SELECT * FROM szemelyek";
18       Statement stmt = conn.createStatement();
19       ResultSet rs = stmt.executeQuery(query);
20
21       while ( rs.next() ) {
22         String name = rs.getString("nev");
23         String date = rs.getString("szulev");
24         System.out.println(name + " " + date);
25       }
26
27       // lezarjuk az adatbazis kapcsolatot
28       conn.close();
29     } catch (SQLException ex) { // hiba tortent
30       System.out.println("Nem sikerült csatlakozni az adatbázishoz.
             ↪ ");
31       ex.printStackTrace();
32     }
33   }
34 }
```

A program fordítása és futtatása következőképpen zajlik:

```
$ javac -cp .:/usr/lib/java/sqlite-jdbc.jar
   ↪ SQLitePelda.java

$ java -cp .:/usr/lib/java/sqlite-jdbc.jar SQLitePelda
Sikerült csatlakozni az adatbázishoz!
Teszt Elek  1980-12-03
Buda István  1968-07-12
```

20.6. Csatlakozás SQLite adatbázishoz PHP-ben

A PHP nyelvben megvannak az SQLite-hoz tartozó függvények.

Első lépésként ellenőrizzük le, hogy a `php.ini` fájlban engedélyezve van az SQLite használata.

```
extension=php_sqlite.dll
extension=php_sqlite3.dll
extension=php_pdo_sqlite.dll
extension="sqlite.so"
```

Az SQLite3 kapcsán a PHP-ben már az objektum-orientált változatot kell követnünk. A függvények nevei itt is az SQLite3 függvénykönyvtáréhoz hasonlóak. Egy egyszerű példát mutatunk be a már jól ismert `pelda.db` adatbázissal:

sqlitepelda.php

```
1  <?php
2    echo '<H1>SQLite példa</H1>';
3
4    $conn = new SQLite3('pelda.db');
5    if ( $conn ) {
6      $result = $conn->query("SELECT * FROM szemelyek");
7      echo '<table border="1">';
8      while ($row = $result->fetchArray(SQLITE3_ASSOC))
   ↪ {
9        echo '<tr><td>' . $row['nev'] . '</td><td>' .
   ↪ $row['szulev'] . '</tr>';
10     }
11     echo '</table>';
```

```
12      $conn->close();
13    }
14  ?>
```

Kérdések és feladatok

1. Miben különbözik az SQLite a MySQL-től?
2. Soroljon fel 3-3 alkalmazási lehetőséget, amikor lehet SQLite-ot használni és amikor nem!
3. Milyen segédprogramjait ismeri az SQLite-nak? Sorolja fel azokat és mondja meg, hogy mire való.
4. Hogyan működik az `ALTER TABLE` utasítás SQLite-ban?
5. Hogyan lehet csatlakozni Java-hoz SQLite-ban?

21. fejezet

Adatbázisok biztonsága

Az adatbázisok biztonsága különösen fontos manapság, amikor már szinte minden adatunk digitálisan is tárolva van. Naponta többszáz weboldal keletkezik, és az interneten történő ügyintézés, vásárlás már szinte mindenkinek megszokottá vált. Emiatt nagy felelősség hárul a programozókra, a tesztelőkre és a rendszerek üzemeltetőire. Habár a programozási nyelvek és programozói függvénykönyvtárak is igyekeznek ezen támadásokat kiküszöbölni, rendkívül fontos, hogy a programozók is felelősen, körültekintően járjanak el az alkalmazások fejlesztésekor.

Ha az adatbázis elleni támadásokat csoportosítani szeretnénk, akkor megkülönböztethetünk SQL befecskendezéses támadásokat, amikor felhasználói felületen keresztül érvényes, ám rossz szándékú SQL utasításokat írnak be a támadók, valamint az adatbázis szerver ellen történő támadásokat, amikor a számítógépes hálózat gyenge pontjait használják ki és maguk a fizikai szerverek válnak célponttá. Ebben a fejezetben az SQL befecskendezésekkel, valamint azok kivédésével foglalkozunk, a hálózat ellen történő támadásokat nem tárgyaljuk az *Adatbázisok* kurzus és ezen jegyzet keretén belül. Az Olvasó megismerheti a leggyakoribb SQL befecskendezési típusokat, amelyek okát és ellene történő védekezési lehetőségeket bemutatjuk, továbbá megismerheti a munkamenetek kezelésének lehetőségeit.

21.1. SQL befecskendezések

Az SQL befecskendezések a felhasználói felületen keresztül megadott SQL utasítások vagy olyan karaktersorozatok, amelyek a beírt adatokat feldolgozó SQL utasításokkal adatmegjelenítést vagy kárt (pl. adatvesztést) okoznak. Ezek a támadások a figyelmetlenül, vagy hanyagul kezelt adatfeldolgozást használják ki. A következőkben a leggyakoribb SQL befecskendezéseket tár-

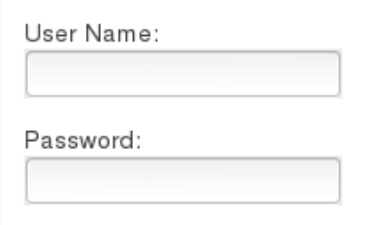
gyaljuk példákon keresztül. Ezek a támadások általában a webes felületű rendszereket érik. A példáinkban is azt fogjuk feltételezni, hogy valamilyen webes felületű alkalmazásunk van, amely rendelkezik egy vagy több beviteli mezővel, amely lehetőséget biztosít a támadásra. A példáinkat kellően általánosan fogalmazzuk meg, nem szögezünk le semmilyen programozási nyelvet.

21.1.1. Felhasználók kiíratására szolgáló támadások

A felhasználók kiíratására szolgáló támadások célja az, hogy elérjék és a képernyőre kiírassák az adatbázisban tárolt felhasználói adatokat. A támadás lényege abban rejlik, hogy a támadó feltételezi, hogy a beírt adat közvetlenül beszúrásra kerül egy `SELECT` utasítás `WHERE` záradékába.

21.1.1. példa

A legegyszerűbb példa, amikor egy bejelentkezési felületünk van.



Az átlag felhasználóknak teljesen természetes, hogy a felhasználói nevüket és a jelszavukat adják meg, de aki a felhasználói adatokat szeretné kilistázni a képernyőre azzal a feltételezéssel él, hogy a beírt adat közvetlenül behelyettesítődik egy SQL utasításba, amely lekérdezi a felhasználó adatait az alábbi módon.

Bejelentkezési adatokat feldolgozó kódrészlet

```
1  kapott_felhasznalonev =
   ↪ felhasznalonev_a_beviteli_mezobol();
2  kapott_jelszo = jelszo_a_beviteli_mezobol();
3
4  eredmeny = sql_execute("SELECT * FROM
   ↪ felhasznalok
5      WHERE felhasznalonev='" +
   ↪ kapott_felhasznalonev + "' AND
6      jelszo='"+ kapott_jelszo + "'");
7
```

Ebben az esetben a támadó nem a felhasználónévvel és jelszóval fog próbálkozni, hanem a felhasználónév és jelszó beviteli mezejőkbe beírja az `' OR ''='` karaktersorozatot. Vizsgáljuk meg miért tenne így, helyettesítsük be az SQL utasításba ezeket (hagyjuk figyelmen kívül az idézőjeleket)!

SQL utasítás a behelyettesítés után

```
SELECT * FROM felhasznalok WHERE
   felhasznalonev='' OR ''='' AND jelszo = ''
   OR ''=''}
```

A támadó kihasználja azt, hogy a beírt karaktersorozatok körül aposztróf jel van, mivel az SQL-ben a sztring konstansokat aposztróf jelek között adjuk meg. Ez az utasítás ki fogja listázni a felhasználók adatait az `eredmeny` változóba, amely utána feldolgozásra kerül (valószínűleg a képernyőre is kiírják). Ha csak egy eredménysort feltételeznek akkor csak az első felhasználó adatait kérték le, aki éppen lehet, hogy adminisztrátor az oldalon, mivel őket szokták elsőként felvenni a rendszerbe. Abban az esetben, ha azt ellenőrzik, hogy van-e ilyen felhasználó, akkor is működik a támadás, hiszen az

SQL utasítás a behelyettesítés után

```
EXISTS(SELECT * FROM felhasznalok WHERE
   felhasznalonev='' OR ''='' AND jelszo = ''
   OR ''='')
```

utasítás `TRUE` értékkel tér vissza.

Hasonló támadások fordulhatnak elő akkor is, ha a beviteli mező számot kér, ekkor azonban nem szabad feltételeznünk azt, hogy aposztróf jel van a behelyettesítés körül. Ekkor a `0 OR 1=1` karaktersorozatot lehet beírni. A 0 helyett bármilyen számot beírhatunk, az 1=1 reláció mindig teljesül.

Hogyan védekezhetünk ilyen támadások ellen? Ami a mi felelősségünk, megnézhetjük, hogy a beírt adat valóban szám-e, ha számot kérünk, illetve sztring-e ha sztringet kérünk. Reguláris kifejezésekkel ellenőrizhetjük, hogy nincs-e benne aposztróf jel. Szerencsénkre a programozási nyelvek is támogatnak bennünket, rendelkeznek például olyan parancsokkal, amely a beírt szövegben lévő szóközöket, speciális karaktereket lecserélik a karakterkódjuk-

ra, így ártalmatlanok lesznek.

21.1.2. Táblák keresése

Ha a támadó az adatbázis tábláira vonatkozóan szeretne információhoz jutni, akkor szintén használhatja a felhasználói felületet. Ezeknek a támadásoknak a célja ténylegesen egy konkrét tábla (például a `felhasználók` táblának) létezésének kiderítése.

21.1.2. példa

Ebben az esetben is abból indulunk ki, hogy egy beviteli mezőbe tudunk beírni valamilyen SQL befecskendezést. Tegyük fel, hogy van egy beviteli mezőnk amely egy terméknevet vár és azt dolgozza fel egy lekérdezésben.

Adatfeldolgozó kódrésztel

```
1    kapott_sztring = sztring_a_beviteli_mezobol();
2
3    eredmeny = sql_execute("SELECT * FROM termek
4      WHERE termeknev='" + kapott_sztring + "'");
5
```

Írjuk a beviteli mezőbe a következő karaktersorozatot:
`1' OR 0 < (SELECT COUNT(*) FROM <keresett táblanév>) OR ''='`
A `<keresett táblanév>` helyére beírjuk azt a táblát, amelynek létezésére kíváncsiak vagyunk. Ha létezik a tábla, akkor érvényes lesz az utasítás, és vagy kapunk vissza eredményrekordot vagy nem, de ha nincs ilyen tábla, akkor hibaüzenet jön létre (amely vagy le van kezelve vagy nincs). Az is előfordulhat, – bár nem valószínű, – hogy a keresett tábla üres.
Az ilyen kereséseket szintén a karaterek átlakításával, karakterkódjukra cseréjével tudjuk kivédeni.

21.1.3. Kötegelt SQL utasítások

A kötegelt SQL utasítások olyan SQL utasítások, amelyeket egymás után adnak meg. Az ilyen támadások szintén a beviteli mezőkön keresztül zajlanak és szintén a behelyettesítést használják ki. A támadó a beviteli mezőbe „lezárja” az előző utasítást és egy kártékony utasítást ír mögé, kötegelve.

21.1.3. példa

Tegyük fel, hogy webes felületen egy számértéket kérünk be, például ruhaméretet. Feltételezzük, hogy az adatfeldolgozó kódrészlet az alábbi módon néz ki:

Adatot feldolgozó kódrészlet

```
kapott_szam = szam_a_beviteli_mezobol();

eredmeny = sql_execute("SELECT * FROM ruhak
    WHERE meret = kapott_szam");

```

Az utasítást könnyedén lezárhatjuk, és írjunk mögé egy kártékony SQL utasítást!
`42; DELETE FROM felhasznalok;`
Ekkor az `sql_execute(...)` függvény által az alábbi utasítás lesz lefuttatva:

A kötegelt SQL utasítás

```
SELECT * FROM ruhak WHERE meret = 42; DELETE
   FROM felhasznalok;
```

A lekérdezés szempontjából a megadott érték lényegtelen, a hangsúly a rákövetkező utasításon van. A támadó nem tudja, hogy létezik-e a `felhasználók` tábla, de addig próbálkozik különböző táblanevekkel, amíg rá nem talál valamire. Neki mindegy milyen adatot töröl ki, a célja az, hogy kárt tegyen.

Hogyan lehet az ilyen támadást kivédeni? Szerencsére már a legtöbb programozási nyelvben és függvénykönyvtárakban tiltva van a kötelgelt SQL utasítások végrehajtása. Amennyiben programozóként van adminisztrátori jogunk az adatbázis felett, jogosultságokkal tilthatjuk azt, hogy a felhasználók (adminisztrátort kivéve) adatot töröljön a táblákból. Ezen kívül készíthetünk rendszeres biztonsági mentést vagy naplót az adatokról. Továbbá ellenőrizhetjük az adatot (valóban szám-e az amit beírtak), illetve átalakíthatjuk a karaktereket a karakterkódjaikra.

21.1.4. Adatbázis feltérképezése

Az adatbázis feltérképezésével kapcsolatban fontos megjegyeznünk, hogy a legtöbb ilyen utasítás rendszergazdai jogot igényel. Habár az egy általános rendszerben nemigen fordul elő, hogy egy látogató felhasználó rendszergazdai jogokkal rendelkezzen, mégis fontos szót ejteni ezekről a fajta támadásokról. Az egyik lehetőség, ami az oldalra látogató felhasználót „rendszergazdai jogokkal ruházza fel", **programozói hibából ered**! Arról van szó ugyanis, hogy a rendszer fejlesztése során a fejlesztő programozó rendszergazdaként (*root*) vagy hasonló jogkörrel fejleszti és teszteli a rendszert, és az adatbázis műveleteket is ezzel a jogkörrel adja ki a programon belül, például az adatbázishoz történő csatlakozásnál megadott felhasználó rendszergazda vagy magas jogkörrel rendelkező felhasználó.

Az adatbázis feltérképezésére a következő lekérdezéseket lehet lefuttatni rendszergazdai jogokkal:

Adatbázisok listázása:

```
SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA;
```

Táblák listázása:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA !='MYSQL' AND
TABLE_SCHEMA!='INFORMATION_SCHEMA';
```

Oszlopok listázása:

```
SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA !='MYSQL' AND
TABLE_SCHEMA!='INFORMATION_SCHEMA';
```

21.1.4. példa

Tegyük fel, hogy eredménylistát az alábbi programkód egy lekérdezés eredményét listázza ki a képernyőre! A lekérdezés eredményéből látszik, hogy milyen adatokat, illetve hány adatot listáz ki. Azt nem tudjuk ugyan, hogy milyen sorrendben, de ha táblázatot látunk, még akár a sorrendre is asszociálhatunk.

Kódrészlet termékek listázásához

```
maximalis_ar = maximalisAr_beviteli_mezo_erteke();

eredmenylista = sql_execute("SELECT termeknev,
 ↪ gyarto, ar
FROM termekek WHERE ar <" + maximalis_ar );

tablazatkent_listaz(eredmenylista);

```

A fenti kódrészletben az egyszerűség kedvéért a listázást a `tablazatkent_listaz(...)` függvény végzi.
Ebben az esetben azt látjuk, hogy három értéket kérdez le a lekérdezés, amelynek típusai: sztring, sztring, szám. Ezt elegendő tudnunk.
Kérdezzük le az adatbázisokat! Adjuk meg a beviteli mezőben a következő karaktersorozatot!
`0 UNION ALL SELECT schema_name, ",0 FROM information_schema.schemata;`
Ekkor a végrehajtott utasítás a következő lesz:

A végrehajtott utasítás

```
SELECT termeknev, gyarto, ar
  FROM termekek WHERE ar < 0 UNION ALL SELECT
     schema_name, '',0 FROM
     information_schema.schemata;
```

A `schema_name` mellé azért kell az üres sztring és a 0 érték, mert az eredeti lekérdezésnél is feltételeztük, hogy három értéket kapunk, amelyik ilyen típusú, és az unió művelet csak kompatibilis táblákon működik. Mi történik, ha csak a `schema_name` attribútumot írjuk oda? Vagy nem ír ki az oldal semmit, ha nem jeleníti meg a hibaüzeneteket, vagy pedig kiírja a hibaüzenetet. Ilyenkor lehet próbálkozni több, vagy kevesebb attribútummal. Termékek így valószínűleg nem jelennek meg eredményként, viszont megjelenik a második lekérdezés eredménye, vagyis az adatbázisnevek listája. A többi adat lekérését is hasonlóképpen próbálhatjuk ki.

Hogyan tudunk védekezni az ilyen támadások ellen? Egyrészről, ha szám-

értéket kérünk be, akkor mindenképpen érdemes meggyőződni arról, hogy a felhasználó valóban számot írt be a mezőbe, tehát ellenőriznünk kell a beírt adatot a feldolgozás és a felhasználás előtt. Abban az esetben, ha nem számot kér be a mező, hanem sztringet, akkor az egyik legbiztonságosabb módszer, hogy a kapott karaktersorozatot lecseréljük a karakterkódjaikra, így az SQL utasítás ártalmatlanná válik.

21.1.5. Rendszerinformáció kiíratására szolgáló támadások

A rendszerinformáció kiíratására szolgáló támadások elsődleges célja nem a károkozás. Ez sokkal inkább, feltérképezés, tapogatózás a rendszer szoftverkonfigurációját illetően. Ennek a támadásnak az a célja, hogy valamilyen hibaüzenetet írassunk ki a képernyőre. A hibaüzenet valószínűleg tartalmazni fogja az operációs rendszert és verzióját, az adatbázis szerver verzióját, amely után már a támadó rákereshet az adott verziójú szoftver sérülékenységeire[1].

21.1.5. példa

A támadás szintén beviteli mezőn keresztül történik. Tekintsük az előző példát, amikor a maximális árat kérte be a termékeket listázó honlap. Tegyük fel, hogy az alábbi kódrészlet dolgozza fel a maximális ár figyelembevételét a termékek listázásánál.

Kódrészlet termékek listázásához

```
1 maximalis_ar = maximalisAr_beviteli_mezo_erteke();
2
3 eredmenylista = sql_execute("SELECT termeknev,
   ↪ gyarto, ar
4 FROM termekek WHERE ar <" + maximalis_ar );
5
6 tablazatkent_listaz(eredmenylista);
7
```

A cél most az, hogy valamilyen hibát írjunk ki a képernyőre. Az egyik legegyszerűbb megoldás erre, hogy ha számot vár az űrlap, akkor nem számot, hanem valamilyen karaktersorozatot adunk meg. Elképzelhető, hogy az oldal ezt valamilyen formában lekezeli, de ha nem akkor

[1] Az egyes operációs rendszerek és szerverek sérülékenységeit és gyenge pontjait gyakran összegyűjtik informatikai biztonsággal foglalkozó weboldalakon. Mivel ezek ismertek, valószínűleg ki is javították azokat.

hibaüzenetet ír ki.

Szintén rendszerinformációra vonatkozóan vannak olyan támadási formák is, ahol SQL utasításokkal kérjük le az információt. Ezen lekérdezések végrehajtásához is adminisztrátori jog kell, de ha a fejlesztők elfelejtik ezt visszaállítani, akkor az oldalra látogató felhasználó is rendelkezik ezekkel. Erre vonatkozóan az alábbi lekérdezéseket használhatjuk:

Hoszt név és IP cím:

```
SELECT @@HOSTNAME;
```

Hosztnév, felhasználónév, jelszó:

```
SELECT HOST, USER, PASSWORD FROM MYSQL.USER;
```

Aktuális felhasználónév és hosztnév:

```
SELECT USER();
SELECT SYSTEM_USER();
```

Verziószám kiíratása:

```
SELECT VERSION();
```

21.1.6. példa

Ebben az esetben is, csakúgy, mint az előző példában a `UNION ALL SELECT` utasítással kísérleljük meg ezeket az adatokat lekérni. Továbbra is feltételezzük, hogy három adatot kapunk eredményül: két sztringet és egy számot. Írjuk a beviteli mezőbe a következőt:
`0 UNION ALL SELECT VERSION(), USER(), 0;`
Ekkor a végrehajtott lekérdezés a következő lesz:

A végrehajtott SQL utasítás

```
SELECT termeknev, gyarto, ar
FROM termekek WHERE ar < 0 UNION ALL SELECT
   VERSION(), @@hostname, 0;
```

Egy másik kísérlet a rendszeradatok listázására a következő lehet:
`0 UNION ALL SELECT HOST, USER, 0 FROM MYSQL.USER`
Ekkor az alábbi lekérdezés hajtódik végre:

A végrehajtott SQL utasítás

```
SELECT termeknev, gyarto, ar
FROM termekek WHERE ar < 0 UNION ALL SELECT
   host, user, 0 FROM mysql.user;
```

A megszerzett rendszerinformációkkal a támadók már hálózaton keresztül tudják támadni a szervert. Ha megszerezték a felhasználói nevet és a jelszót is, akkor már az adatbázishoz is hozzáférhetnek. Megjegyezzük, hogy a jelszót általában kódolt formában írja ki a MySQL a képernyőre, de előfordulhat, hogy néhány rendszer így is elfogadja.

Hogyan védekezhetünk ilyen jellegű támadások ellen? Egyrészről megint az adatellenőrzést tudjuk javasolni. Ha érvénytelen értéket írunk be, akkor már alkalmazás szinten elháríthatjuk a hibát, az beírt karaktersorozatot nem is helyettesítjük be az utasításba. Másrészről ellenőrizzük, hogy az az adatbázis-felhasználó, aki csatlakozik az adatbázishoz, ne rendelkezzen adminisztrátori jogkörrel.

21.1.6. Lokális fájlok elérése

A lokális fájlokra vonatkozóan is léteznek SQL befecskendezési lehetőségek. Ezek a támadások azt használják ki, hogy ha a szerver bizonyos fájljaira a jog nincs megadva a „látogatók” számára (csak tulajdonosnak, esetleg a csoportnak van joga olvasni, írni vagy futtatni a fájlt), a rendszeren – például az adatbázis szerveren – keresztül el lehet érni a fájlt. Az ilyen utasításokhoz is adminisztrátori utasítás kell. A fájlok elérése is a fenti példákhoz hasonlóan `UNION ALL SELECT` utasításokkal történik:

Fájl elérése, letöltése:

```
SELECT LOAD_FILE('/etc/password');
```

Tábla tartalma kimentése fájlba:

```
SELECT * FROM táblanév INTO dumpfile '/tmp/mentettFajl'
```

Hogyan védjük ki ezeket a támadásokat? Egyrészről az adatbázis-felhasználó jogkörét állítsuk be, ne legyen adminisztrátor. Ellenőrizzük az adatot, ha számot várunk és az nem szám, akkor az alkalmazás szinten írjunk hibát és ne futtassuk le a lekérdezést. Ha szöveges adatmezőből kaptuk az adatot, akkor pedig alakítsuk át a karaktereket a karakterkódjaikra.

21.1.7. Védekezési lehetőségek összefoglalása

A fentiekben számos példát láttunk SQL befecskendezésekre, láttuk, hogy a támadók ilyen SQL utasításokkal milyen adatokhoz juthatnak hozzá és milyen károkat tehet. Most foglaljuk össze, hogy programozóként, rendszerfejlesztőként milyen lépéseket tehetünk az ilyen jellegű támadások ellen!

1. Hozzunk létre egy adatbázis-felhasználót a csatlakozáshoz és korlátozzuk a jogkörét! Rendkívül fontos lépés ez, ugyanis a legtöbb SQL injekció adminisztrátori végrehajtással működik csak. Ha mindjárt a fejlesztés kezdetekor egy külön felhasználóval érjük el az adatbázist, amely csak adatmanipulációs jogokkal rendelkezik, akkor nem tud rendszerre vonatkozó adatokat lekérdezni. Gondoljuk végig, hogy ha felhasználói szerepkörök tartoznak az adatbázisunkhoz, akkor ezeknek a szerepköröknek milyen adatbázis műveleteket szabad végrehajtaniuk. Ha lehetőségünk van, akkor minden felhasználói szerepkörhöz rendelhetünk jogkört, vagy megkülönböztethetünk egy „hagyományos felhasználót”, és egy „főnököt”, aki nagyobb jogkörrel rendelkezik. Korlátozhatjuk az adatbázis műveleteket az egyes táblákra vonatkozóan is, például egy webshop esetében a „hagyományos felhasználó” a `Termékek` táblából lekérdezni tud adatokat, a „főnök” viszont hozzáadhat új rekordot, módosíthatja és törölheti is a rekordokat.

2. Ellenőrizzük a kapott adatot! Ez a második fontos védvonalunk az adatbiztonság tekintetében. Amennyiben tudjuk, hogy a beírt adat milyen karaktereket tartalmazhat, milyen hosszú lehet, ismerjük a formátumát (például dátum esetében), akkor írjunk ezekre vonatkozóan reguláris kifejezéseket és ellenőrizzük alkalmazás szinten azt, hogy beírt adat megfelel-e az elvárásoknak! Amennyiben akár formai vagy hosszbeli eltérést észlelünk, alkalmazás szinten tudunk hibajelzést küldeni a felhasználónak és az adatot nem helyettesítjük be az SQL utasításba. Ha a megadott adat minden ilyen elvárásnak megfelel és átmegy a szűrésen, akkor biztos megoldás, hogy a karaktereit lecseréljük a karakterek kódjaira. Ha így írjuk be, akkor az SQL utasítások sem utasításként hajtódnak végre, csupán karakterkódok sorozata lesz. Ily módon ártalmatlaníthatjuk az SQL befecskendezéseket.

3. Készítsük elő az SQL utasításokat! A legtöbb programozási nyelvben és függvénykönyvtárban, amely SQL adatbázisokat kezelni tud már alapvető biztonsági eszköz az, hogy az SQL utasításokat elő lehet készíteni és a kívülről érkező paramétereket hozzá lehet kötni az utasításokhoz.

Így nem sztringként fűzzük be az utasításba, hanem rábízzuk a feldolgozó függvényekre a külső adat bekötését. Ezekre a megoldásokra nem hoztunk itt példát, mert a sérülékeny, érzékeny kódrészleteket szerettük volna itt bemutatni, de az alkalmazásfejlesztésről szóló fejezetben megmutatjuk, hogyan kell ezeket megvalósítani Java-ban és PHP-ben (lásd 23. és 24. fejezet).

21.2. Sütik és munkamenetek

A sütik- és munkamenetek kezelését illetően jelen tananyagon belül csak részlegesen foglalkozunk, ugyanis ez a témakör inkább a rendszerfejlesztés valamint a webes alapú alkalmazásfejlesztés témaköréhez kapcsolódik szorosabban, de szemléletformáló szándékkal fontosnak tartjuk az itt leírtakat. Az Olvasó ebben az alfejezetben azt tanulhatja meg, hogy adatbázisokkal hogyan támogathatja a munkamenetek és sütik kezelését. Azzal, hogy az Olvasó nem csupán az alkalmazás szintjén kezeli a munkameneteket, fontos előrelépést tesz az általa fejlesztett adatbázisok és adatok biztonsága érdekében.

A munkamenetek (*sessions*) kezelése elengedhetetlenül fontos a mai webes alkalmazásfejlesztésben. A legtöbb webes alkalmazásfejlesztésre használt nyelv, – mint például a PHP is – támogatja a munkamenetek kezelését. A munkamenet kezelés lényege, hogy a felhasználó a weboldalon végzett tevékenységeit munkamenetként tartjuk nyilván. A munkameneteket létrehozhatjuk akkor, amikor a felhasználó meglátogatja az oldalt, vagy amikor belép egy webes felületű rendszerbe. A lépéseit, adatait munkamemenet változókban a webszerver tárolja. A munkamenet-változókban tárolt adatokat használhatjuk arra, hogy navigáljuk őt az oldalon, tegyünk elérhetővé számára új űrlapokat, felületeket. A munkamenetek létrehozásakor a rendszer minden munkamenethez egy új munkamenet azonosítót rendel, amelyet a szerveren tárol. A munkamenet során az alkalmazás eltárolhat bizonyos információkat ehhez a munkamenet-azonosítóhoz, például, hogy milyen oldalon állunk éppen, és ezen információ alapján felkínálhat a rendszer olyan lehetőségeket, hogy milyen oldalakra mehetünk tovább.

21.2.1. példa

Mindenki számára ismerős lehet az a példa, amikor az interneten vásárolunk. Belépünk a webshop oldalára, ekkor elindíthatunk egy munkamenetet. A munkamenet-azonosítóba eltárolhatjuk a felhasználói azonosítónkat, így a továbbiakban nem kell minden alkalommal azonosítanunk magunkat. Böngészünk az áruház terméklistájában, kiválasztjuk

a megvásárolni kívánt termékeket és megyünk a „pénztárhoz” fizetni. Azt, hogy mi elindítjuk a fizetés folyamatát (és a „pénztár” oldalára lépünk) eltárolhatjuk a munkamenet-változókban is. Innen a rendszer fel fogja ajánlani, hogy a következő lépésben adjuk meg a számlázási és szállítási címünket. Ha erre az oldalra lépünk, akkor munkamenet változókban megjegyezheti a címeket, a termékeket és megjelenít egy oldalt, hogy a fizetés és a tényleges vásárlás előtt tekintsük át ezeket az adatokat, majd ha jóváhagytuk, akkor továbbléphetünk a fizetési felületre és ekkor ténylegesen adatbázisba is eltárolható a rendelésünk (addig ugyanis bármikor meggondolhatjuk magunkat és tovább vásárolhatunk, vagy kivehetjük a terméket a „kosárból”).

A sütik (*cookies*) is hasonló szerepet játszanak a webes alkalmazásokban, de nem a szerveren, hanem a kliensen tárolják az adatokat. Funkcióikat tekintve hasonlóan használhatóak az adatok tárolására, mint a munkamenetek.

Hogyan jutnak szerephez az adatbázisok a munkamenetek és a sütik kezelésében? Léteznek olyan hálózaton keresztüli támadások, amelyek a munkamenetek vagy sütik megszerzésére irányulnak, így a támadó a megszerzett munkamenet azonosítóval a vásárló vagy felhasználó „nevében” tud bizonyos műveleteket végrehajtani. Az adatbázisokkal úgy támogathatjuk a rendszer biztonságát, hogy a felhasználókhoz eltároljuk a munkamenetek azonosítóit és minden egyes alkalommal, amikor új oldalra lép ellenőrizzük, hogy az érvényben lévő munkamenet-azonosító megegyezik-e az adatbázisban eltárolt munkamenet-azonosítóval. Ezt a módszert sütik esetében is alkalmazhatjuk.

21.2.2. példa

Tegyük fel, hogy a korábban látott **Fórum** példát használjuk. A felhasználói felületen a rendszerbe történő belépéshez a felhasználóknak meg kell adniuk e-mail címüket és jelszavukat. A `FELHASZNÁLÓ` táblában csak az alábbi adatok vannak eltárolva:

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó_belépés_időpontja)

Vegyünk hozzá fel még egy mezőt `sid`(=*session ID*) néven, tehát a `FELHASZNÁLÓ` sémánk az alábbi lesz:

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó_belépés_időpontja, sid)

Amikor a felhasználót hitelesítettük, akkor létrehozunk számára egy munkamenet-azonosítót, amit adatbázis szinten is eltárolunk. Amíg a felhasználó az oldalon tevékenykedik, addig ez a munkamenet-azonosító érvényben marad. Minden esetben, amikor új oldalra lép a felhasználó, a munkamenet-azonosítóit (a webszerveren tároltat és az adatbázisban tároltat) ellenőrizzük. Kilépéskor a munkamenet-azonosítót a szerverről töröljük és az adatbázisban az `sid` mező értékét `NULL`-ra állítjuk.
Tegyük fel, hogy amíg a felhasználónk tevékenykedik a fórumba valaki éppen be akar lépni a rendszerbe az „ellopott" felhasználónevével és jelszavával. A bejelentkezés új munkamenet-azonosítót generál a `FELHASZNÁLÓK` táblába, így amikor a felhasználónk új oldalra lép, nem fog egyezni a webszerveren és az adatbázisban tárolt munkamenet-azonosító. Ebben az esetben célszerű a felhasználót kiléptetni a rendszerből, megszüntetni a munkamenet-azonosítóját, és értesíteni őt a többszörös bejelentkezésről.
A munkamenet-kezelés tovább bonyolítható, ha minden egyes oldal váltás esetén újrageneráljuk a munkamenet-azonosítót, és az adatbázisban is frissítjük azt.

A sütikről és a munkamenetek kezelésének ismertetéséről szóló legfontosabb ismereteket említettük meg itt. A webes alkalmazás fejlesztés kapcsán a későbbi fejezetekben még lesz gyakorlati példa is a munkamenetek kezelésére, illetve a téma iránt érdeklődők a szakkönyvekben [5, 20] is utánanézhetnek a részleteknek.

Kérdések és feladatok

1. Mi az SQL befecskendezés?
2. Mit jelent a kötegelt utasítás?
3. Hogyan lehet lekérdezni egy adatbázis-kezelő rendszer verziószámát?
4. Soroljon fel olyan adatbázis-kezelő rendszer-adatokat, amelyekhez adminisztrátori jogok szükségesek!
5. Miyen utasítással lehet lekérdezni a táblákat MySQL-ben?

6. Hogyan védekezhetünk az SQL befecskendezések ellen?

22. fejezet

A PHP nyelv

Ebben a fejezetben az Olvasó megismerkedhet a PHP nyelv általános szintaktikájával, vezérlési szerkezeteivel. Ezek az ismeretek elengedhetetlenül szükségesek a további programkódok megértéséhez.

A PHP nyelvet általában dinamikus weboldalak fejlesztéséhez használják, amikor a tartalom valamilyen bevitt adat vagy pedig felhasználói interakció hatására dinamikusan jön létre. Használható azonban nem webes tartalmú programokhoz is, ekkor a `php` értelmező program fogja a programkódot értelmezni. Ebben a formában ritkábban használják.

Korábban a PHP nyelvben nem valósítottak meg objektum-orientált programozási lehetőségeket, csak függvényeket lehetett használni, az utóbbi időben, különösen az adatbázis-kezelés kapcsán megjelentek már az objektum-orientált megoldások is.

A PHP nyelvű kódrészletet a `<?php ... ?>` jelek nyitó- és zárószimbólumok közé kell tenni. A szimbólumok között több programsor, akár teljes program szerepelhet, de olyan kis részletek is, amelyek valamilyen dinamikus tartalmat tartalmaznak. Az utasításokat a ; karakter zárja.

A nyelvben a kis- és nagybetűk különbözőek.

22.1. Adattípusok, változók

A PHP gyengén típusolt nyelv, amely azt jelenti, hogy a változó típusát a benne tárolt érték határozza meg, és újbóli értékadás esetén a típus is megváltozhat.

22.1.1. Adattípusok

A PHP nyelv az alábbi típusokat támogatja:

String: Karaktersorozat. A sztringeket egyszeres vagy dupla idézőjelek között adjuk meg.

Integer: Egész érték -2 147 483 648 és 2 147 483 647 között.

Float(/Double): Lebegőpontos számérték.

Boolean: Logikai adattípus, TRUE és FALSE értéket vehet fel.

Array: Egy változóban több értéket is tud tárolni.

Object: Osztály példány adattagokkal és függvényekkel[1].

NULL: Speciális adattípus a NULL értékhez.

Resource: Speciális adattípus (függvény-)referenciák tárolására.

22.1.2. Változók

A változókat PHP-ben mindig a $ karakterrel kezdjük. Egy változónak többször is adhatunk értéket. A típusát a benne tárolt érték határozza meg.

22.1.3. Konstansok

A konstansokat a `define(konstans neve, értéke, kis- és nagybetű különböző-e)` függvénnyel hozhatjuk létre. Eléjük nem kell $ dollárjelet tenni. A `define()` függvény harmadik paraméterének alapértelmezett értéke `false`. Az alábbi kódrészletben létrehozunk egy konstanst, majd az `echo` utasítással kiírjuk[2].

Konstans definiálása

```
1 <?php
2 define("UDVOZLOM", "Üdvözlöm ");
3 echo UDVOZLOM;
4 ?>
```

[1] Létezik PHP-ben egy `stdclass` általános osztálytípus, amely hasonlít a Java nyelvben található `Object` osztályhoz, de ezzel jelen tananyag keretén belül nem foglalkozunk.

[2] Az `echo` utasítás webes felület esetén a weboldalra, parancsoros programok esetében pedig konzolra ír.

Tömbök

Tömböket az `array()` függvény segítségével definiálhatunk. Ha tudjuk a tömbök értékeit, akkor ezeket az `array()` függvényben megadhatjuk felsorolva, ha csak inicializálni szeretnénk a tömböt, akkor pedig nem kell értéket adnunk az `array()` függvényben. A PHP-ben csak egydimenziós tömböket tudunk deklarálni, de minden tömb egy cellájában tárolhat tömb típusú értéket. Ennek kezelése körülményes a többi programozási nyelvben megszokott többdimenziós tömbökhöz képest. A PHP a tömböket *asszociatív tömbként* kezeli, vagyis kulcs-érték párokat tárol, ennélfogva nem csak számmal történő folytonos indexelést használhatunk. Ha nem kulcs-érték párt adunk meg a tömb deklarációjánál, akkor 0-tól kezdődően kezdi számozni az elemeket. A tömböket a `[]` zárójelpárral indexelhetjük. Az alábbi példában bemutatjuk a tömbök kezelését. A kulcs-érték párokat a `"kulcs"=>"érték"` formában kell megadnunk.

Tömbök deklarálása és indexelése

```
1  <?php
2    $tomb = array("Hétfő", "Kedd", "Szerda", "Csütörtök"
   ↪ , "Péntek", "Szombat", "Vasárnap");
3    echo $tomb[1]; // Kedd
4
5    $szerzoTomb = array();
6    $szerzoTomb['adatbazisok'] = array("BP"=>"Dr. Balázs
   ↪  Péter", "NG"=>"Dr. Németh Gábor");
7    echo $szerzoTomb['adatbazisok'][0];   // Dr. Balázs
   ↪ Péter
8  ?>
```

22.1.4. Műveletek

A PHP nyelv műveletei, operátorai nagyon hasonlóak a többi nyelvhez. Kivételt képez talán a sztring konkatenáció.

=	értékadás
+,-,*,/, **, %	összeadás, kivonás, szorzás, osztás, hatványozás, maradékképzés
==	értékre vonatkozó egyenlőség
>, <, <=, >=	nagyobb, kisebb, kisebb vagy egyenlő, nagyobb vagy egyenlő
!=, <>	nem egyenlő
===	azonosan egyenlő (például tömböknél)
!==	nem azonos érték és típus (például tömböknél)
++, −−	incrementáció, dekrementáció
+=, -=, /=, %=, **=	értékadással összekötött aritmetikai műveletek
.	sztringek összefűzése
.=	sztringek konkatenációja értékadással
and, &&	logikai ÉS művelet
or, ||	logikai VAGY művelet
xor	logikai KIZÁRÓ VAGY művelet
!	logikai negáció

22.2. Vezérlési szerkezetek

A PHP nyelvben használt vezérlési szerkezetek szintaxisa hasonlít más nyelvek (például C és Java) szintaxisára.

22.2.1. Feltételes vezérlés

A feltételes ellenőrzésnél a feltétel teljesülése esetén a blokkon belüli programrészlet végrehajtódik. Több feltételes eset esetén az első teljesülő feltétel blokkjában lévő programrészlet hajtódik végre. Ha egyik feltétel sem teljesül és van `else`-ág, akkor annak blokkjában szereplő kódrészlet hajtódik végre.

Feltételes vezérlés

```
1  // egyszerű feltétel ellenőrzés
2  if ( feltétel ) {
3    ...
4  }
5
6  // if - else
7  if ( feltétel ) {
8    ...
```

```
} else {
  ...
}

// if - else if
if ( feltétel ) {
  ...
} else if ( feltétel2 ) {
  ...
}

// if - else if - else
if ( feltétel ) {
  ...
} else if ( feltétel2 ) {
  ...
} else {
  ...
}
```

22.2.2. Esetkiválasztásos vezérlés

Esetkiválasztásos vezérlés esetén egy változó vagy kifejezés értékét vizsgáljuk. A program elkezdi ellenőrizni a megadott eseteket. Ha valamelyiknek megfelel, akkor a hozzá tartozó programrészlet hajtódik végre. Ha egy ilyet sem talált, akkor `default` esetben leírt kódrészlet hajtódik végre. Minden esetnél fontos, hogy `break;` utasítással zárjuk le az eset kezelését, mert különben az összes azután lévő eset programblokkja végrehajtódik.

Esetkiválasztásos vezérlés

```
switch ( változó ) {
  case érték1:
    ...
    break;
  case érték2:
    ...
    break;
  case érték3:
    ...
    break;
  default:
    ...
```

```
13 }
```

22.2.3. Számlálos ismétléses vezérlés

Számlálásos ismétléses vezérlésnél a `for`-ciklus magja annyiszor hajtódik végre, ahányszor a `feltétel` teljesül.

for-ciklus

```
1
2  // minta: for(inicializálás; feltétel; inkrementálás)
3
4  for ($x=1; $x<=10; $x++ ) { // 10-szer hajtódik végre
    ↪ a ciklusmag
5    // ciklusmag
6    ...
7  }
```

22.2.4. Tömb elemeinek feldolgozása

A tömböket általában a `foreach`-ciklussal dolgozzuk fel, amikor a kulcs-érték párokat is tudjuk kezelni.

foreach-ciklus

```
1  $napok = array("H"=>"hétfő", "K"=>"kedd", "SZE"=>"
    ↪ szerda", "CS"=>"csütörtök", "P"=>"péntek","SZO"=>"
    ↪ szombat", "V"=>"vasárnap");
2  foreach ( $napok as $kulcs=>$ertek ) {
3    echo $kulcs . " = " . $ertek;
4  }
```

22.2.5. Elöltesztelős ismétléses vezérlés

Az elöltesztelős ismétléses vezérlésnél minden ciklus elején ellenőrizzük a feltétel teljesülését, és ha teljesül, végrehajtjuk a ciklusmagot.

Elöltesztelős ismétléses vezérlés

```
while ( feltétel ) {
  // ciklusmag
  ...
}
```

22.2.6. Hátultesztelős ismétléses vezérlés

A hátultesztelős ismétléses vezérlés első lépésében a ciklusmag egyszer mindenképpen végrehajtódik, és amennyiben a feltétel utána is teljesül, a ciklusmag ismét végrehajtódik.

Hátultesztelős ismétléses vezérlés

```
do {
  // ciklusmag
  ...
} while ( feltétel );
```

22.3. Függvények, eljárások

A függvények és eljárások deklarálása a `function` kulcsszóval történik. A függvények esetében van visszatérési érték, de ennek típusát a deklarációnál nem kell jelölnünk. Az eljárások abban különböznek a függvényektől, hogy nincs visszatérési értékük.

Az alábbi példában egy duplázó függvényt hozunk létre. Megvizsgáljuk az `isnumeric()` beépített függvénnyel, hogy a paraméter értéke szám-e, ha igen, duplázunk, különben visszatérünk egy `null` értékkel.

Függvény deklaráció

```
function duplaz( $x ) {
  if ( is_numeric($x) ) {
    return 2*$x;
  }
  return null;
}
```

22.4. Osztályok, objektum-orientált programozás

Az osztályok összetett adattípusok, amelyek rendelkeznek adattokkal és függvényekkel is. Az osztályok deklarálására a `class` kulcsszót használjuk. Az osztályon belül a saját adattagokra a `$this->adattag` formában hivatkozunk. Az osztályok adattagjaira és metódusaira beállíthatjuk a `public`, `private` láthatósági szinteket[1].

Az objektumpéldányokat a `new` kulcsszóval hozzuk létre. Az objektumok függvényeit a `->` operátorral érjük el, a statikus függvényekre az `Osztálynév::függvény()` formában is hivatkozhatunk. A PHP 5-től kezdve használhatjuk a `__construct()` függvényt konstruktorként illetve a `__toString()` függvényt az osztályunk String-gé konvertálására[2]

Osztály deklaráció és példányosítás

```
<?php
  class Szemely {
    // adattag
    private $nev;


    // metódusok
        public function __construct($pnev) {
      $this->nev = $pnev;
    }

    public function __toString() {
      return 'Szemely('.$this->nev.')';
    }


    public nevetAd($ujnev) {
      $this->nev = $ujnev;
    }

    public nevetKiir() {
      echo $this->nev;
    }
  }

  $egy_szemely = new Szemely();
```

[1] https://www.php.net/manual/en/language.oop5.visibility.php
[2] https://www.php.net/manual/en/language.oop5.magic.php.

```
27    $egy_szemely->nevetAd("Teszt Elek");
28    $egy_szemely->nevetKiir();
29  ?>
```

22.5. Külső programkódok importálása

Külső programrészletek, forrásfájlok importálására, használatára az `include()` és `include_once()` függvények használhatóak, amelyeket kötelezően a fájl elején kell megadni. Ezekkel olyan forrásfájlokat célszerű importálni, amelyek valamilyen (függvény, osztály, konstans változó) deklarációt tartalmaznak, és másik fájlban is szeretnénk használni őket. Amennyiben előfordul, hogy több forrásban szeretnénk az adott kódrészletet használni, úgy az `include_once()` függvény használatát javasoljuk, mert különben a deklarált elemek többszörös importálás során többször lesznek deklarálva, ami hibához vezet.

Hasonlóképpen használhatók még a `require()` és `require_once()` függvények is[1].

adatbazis_fuggvenyek.php

```
1  <?php
2    function adatbazis_csatlakozas( $hoszt,
     ↪ $felhasznalonev, $jelszo ) {
3      ...
4    }
5  ?>
```

weboldal.php

```
1  <?php
2  include_once(adatbazis_fuggvenyek.php)
3    ...
4    adatbazis_csatlakozas("localhost", "felhasznalo",
5         "jelszo");
6    ...
7  ?>
```

[1] https://www.php.net/manual/en/function.require-once.php

Kérdések és feladatok

1. Miről lehet felismerni egy PHP kódrészletet?
2. Milyen kiíratási lehetőségek vannak a PHP-ben?
3. Hogyan lehet deklarálni egy PHP változó típusát?
4. Hogyan lehet definiálni a konstansokat PHP-ben?
5. Hogyan lehet tömböket létrehozni és indexelni PHP-ben?

23. fejezet

Webes alkalmazás fejlesztése PHP nyelven MySQL adatbázishoz

Ebben a fejezetben az Olvasó lépésről lépésre végigkövetheti egy adatbázissal támogatott webes minta alkalmazás fejlesztését. A leírásban szem előtt tartjuk az adatbiztonsággal foglalkozó fejezetben leírtakat.

Az alkalmazás, amin a fejlesztés menetét bemutatjuk a korábbi fejezetekben látott Fórum lesz. Az adatbázis tervezésén már túlvagyunk, hiszen ezekre láttunk példát a korábbi fejezetekben.

A tervezés során a modell-nézet-vezérlő (angolul *Model-View-Controller, MVC*) tervezési mintát használjuk [10, 19]. Az MVC tervezési minta szerint elkülönítjük az adatokat kezelő egységeket (=*modell*), a (grafikus) felhasználó felülethez kapcsolódó egységeket (=*nézet*), és az őket összekötő, és közöttük adatokat továbbító (=*vezérlő*) egységeket.

23.1. Az adatbázis előkészítése

Az egyszerűség kedvéért a továbbiakban az ingyenesen elérhető XAMPP[1] programcsomagot használjuk, amely magában foglal egy Apache webszervert, a MySQL adatbázis szervert (jelenleg a MariaDB változatot), és egy PHP értelmezőt. Segítségünkre lesz ebben a programcsomagban a PhpMyAdmin webes felületű kliens program az adatbázis előkészítéséhez. A XAMPP telepítését nem tárgyaljuk, mivel az aktuális verzió telepítésének leírása a XAMPP honlapján elérhető Windows, Linux és MacOS rendszerre. A XAMPP Control Panel-t elindítva indítsuk el az *Apache* és *MySQL* szolgáltatásokat! Bön-

[1] `https://www.apachefriends.org/hu/`

gészőnkbe nyissuk meg a `localhost/phpmyadmin/` oldalt! Ez a saját gépünkön fut. A forrásfájlokat a XAMPP `htdocs` könyvtárába létrehozott `forum` könyvtárba helyezzük el!

A felső sorban a fülek közül válasszuk a **Felhasználói fiókok** fület és hozzunk létre egy új felhasználót! Állítsuk be a felhasználónevet, jelszót, valamint az űrlap alján állítsuk be a jogokat[1]! Csak adatokra vonatkozó jogokat engedjünk a felhasználónak! Az űrlap alján kattintsunk az **Indítás** gombra! Hozzunk létre egy új adatbázist *forum* néven, a karakterkódolása legyen `utf8_hungarian_ci`.

A felső fülsoron a **Felhasználói fiókok** gombra kattintva megjelennek a felhasználók. Válasszuk a *forumtag* felhasználó sorában a jogosultságok szerkesztését! A megjelenő űrlapon válasszuk az **Adatbázis** gombot a lap tetején! A gördülő listából válasszuk ki a *forum* adatbázist és kattintsunk az **Indítás** gombra! Jelöljük itt is be az adatok lekérdezésére, módosítására, törlésére vonatkozó jogokat! A jogokat az alábbi SQL utasítással is beállíthatjuk.

Jogokat beállító SQL utasítás

```
GRANT SELECT, INSERT, UPDATE, DELETE ON `forum`.* TO
   'forumtag'@'%';
```

Hozzuk létre a *forum* adatbázisban az alábbi táblákat! A korábbi fejezetben látott példákhoz képest ez abban különbözik, hogy hozzávettük a `sessionid` és `cookie` mezőket a munkamenetek kezelése érdekében.

`FELHASZNÁLÓ`(felhasználónév, jelszó, email, vezetéknév, keresztnév, utolsó belépés időpontja, sessionid, cookie)
`ÜZENET`(sorszám, tartalom, mikor, *felhasználónév*, *hírfolyam azonosító*)
`HÍRFOLYAM`(azonosító, megnevezés)
`KULCSSZAVAK`(*hírfolyam azonosító*, kulcsszó)
`KÖVETI`(*hírfolyam azonosító*, *felhasználónév*)

A `felhasznalok` tábla létrehozása

```
CREATE TABLE felhasznalok (
  felhasznalonev VARCHAR(40) PRIMARY KEY,
  jelszo CHAR(40) NOT NULL,
  email VARCHAR(50) UNIQUE NOT NULL,
```

[1]Ebben a minta alkalmazásban a *forumtag* felhasználónevet és a *tagjelszo123* jelszót választottam, de valódi alkalmazás esetén ennél erősebb felhasználónév és jelszó kell!

```
  vezeteknev VARCHAR(60) NOT NULL,
  keresztnev VARCHAR(60) NOT NULL,
  utolso_belepes_idopontja TIMESTAMP DEFAULT NULL,
  sessionid CHAR(128) DEFAULT NULL,
  cookie CHAR(128) DEFAULT NULL);
```

A hirfolyam tábla létrehozása

```
CREATE TABLE hirfolyam (
azonosito INT PRIMARY KEY AUTO_INCREMENT,
megnevezes VARCHAR(60) NOT NULL);
```

Az uzenet tábla létrehozása

```
CREATE TABLE uzenet (
  sorszam INT PRIMARY KEY AUTO_INCREMENT,
  tartalom TEXT NOT NULL,
  mikor TIMESTAMP NOT NULL,
  felhasznalonev VARCHAR(40) NOT NULL
    REFERENCES hirfolyam(azonosito) ON DELETE CASCADE
       ON UPDATE CASCADE,
  hirfolyam_azonosito INT DEFAULT NULL
    REFERENCES hirfolyam(azonosito) ON DELETE SET
       NULL ON UPDATE CASCADE);
```

A kulcsszavak tábla létrehozása

```
CREATE TABLE kulcsszavak (
  hirfolyam_azonosito INT REFERENCES
     hirfolyam(azonosito) ON DELETE CASCADE ON UPDATE
     CASCADE,
  kulcsszo VARCHAR(20),
  PRIMARY KEY (hirfolyam_azonosito, kulcsszo));
```

A koveti tábla létrehozása

```
CREATE TABLE koveti (
hirfolyam_azonosito INT REFERENCES
   hirfolyam(azonosito) ON DELETE CASCADE ON UPDATE
   CASCADE,
felhasznalonev VARCHAR(40) REFERENCES
```

```
    felhasznalo(felhasznalonev) ON DELETE CASCADE ON
    UPDATE CASCADE,
PRIMARY KEY (hirfolyam_azonosito, felhasznalonev));
```

23.2. Modell függvények elkészítése

Most elkészítjük a modell-függvényeket, amelyek az adatbázist kezelik. Ezeket a *db_fuggvenyek.php* fájlba gyűjtjük össze. Megjegyezzük, hogy a fájl elejére és végére kitesszük a `<?php ...?>` nyitó- és zárószimbólumokat, de itt a kódrészletben nem jelöljük, mert az egyes függvényeket külön ismertetjük.

Először az adatbázis kapcsolatot létrehozó `forum_csatlakozas()` függvényünket fogjuk elkészíteni. Az adatbázis kapcsolatot a `mysqli_connect()` függvénnyel hozzuk létre, amely paraméterül várja a *hosztnevet*, a *felhasználónevet* és a felhasználó *jelszavát*. A visszatérési értéke egy csatlakozási azonosító, amelyet a további adatbázis-kezelő parancsokhoz tudunk felhasználni. Ezt követően a `mysqli_select_db()` függvénnyel kiválasztjuk az adatbázist. A függvény első paramétere a csatlakozási azonosító, második paramétere pedig az adatbázis neve. A `forum_csatlakozas()` függvényben állítjuk be azt is, hogy a bevitt és kapott adatokat milyen karakterkódolással kezeljük (célszerű ezt UTF-8-ra állítani). Ehhez SQL utasításokat kell futtatni a `mysqli_query()` függvénnyel. A `mysqli_query()` függvény első paramétere a `mysqli_connect()` által visszaadott csatlakozási azonosító, a második paramétere pedig az SQL utasítást tartalmazó sztring. A `forum_csatlakozas()` függvényünk a visszatérési értéke a csatlalkozási azonosító lesz sikeres csatlakozás esetén, különben pedig `null` értékkel tér vissza.

A `forum_csatlakozas()` függvény

```
1 function forum_csatlakozas() {
2
3   $conn = mysqli_connect("localhost", "forumtag", "
  ↪ tagjelszo123") or die("Csatlakozási hiba");
4   if ( false == mysqli_select_db($conn, "forum" )  ) {
5     return null;
6   }
7
8   // a karakterek helyes megjelenítése miatt
9   // be kell állítani a karakterkódolást!
```

```
10    mysqli_query($conn, 'SET NAMES UTF-8');
11    mysqli_query($conn, 'SET character_set_results=utf8'
      ↪ );
12    mysqli_set_charset($conn, 'utf8');
13
14    return $conn;
15  }
```

23.2.1. Felhasználó regisztrálása

A következő függvényünk, amit elkészítünk, az új felhasználók regisztrálására szolgál. A függvény öt paraméterrel rendelkezik: e-mail cím, felhasználónév, keresztnév, vezetéknév és jelszó. Fontos, hogy ezeket az értékeket a rendszer „kívülről” kapja, ezért az SQL befecskendezések kivédése érdekében az adatot át kell alakítanunk (ezeket majd a *vezérlő* elemben fogjuk elvégezni), illetve úgynevezett előkészített utasításba fogjuk bekötni. A jelszót SHA1-es kódolással fogjuk lekódolni, amely tetszőlegesen hosszú szöveget 40 hosszú hexadecimális karaktersorozattá alakít át. Az `INSERT` utasítást a `mysqli_prepare()` függvénnyel készítjük elő. A külső paraméterek helyét kérdőjellel jelöljük. A függvény visszatérési értéke egy *statement* referencia, amely az `INSERT` utasításunkat fogja reprezentálni. A paramétereket ehhez fogjuk kötni a `mysqli_stmt_bind_param()` függvénnyel, amely első paramétere a *statement* referenecia, második paramétere egy formátum sztring, amelyben az *s* a sztring típusú paramétert jelöli, *d* pedig az egész típusú paramétert jelöli, a további paraméterek pedig a bekötendő paraméterek. A formátumsztringben annyi karakternek kell szerepelnie, ahány paramétert kötünk be. Az `INSERT` utasítást a `mysqli_stmt_execute()` függvénnyel futtatjuk le. Ne feledkezzünk meg a függvény végén az adatbázis kapcsolatot lezárni!

A `uj_felhasznalo_regisztralasa()` függvény

```
1  function uj_felhasznalo_regisztralasa($email,
   ↪ $felhasznalonev, $keresztnev, $vezeteknev, $jelszo
   ↪ ) {
2    // csatlakozunk az adatbazishoz
3    $conn = forum_csatlakozas();
4
5    if ( $conn ) {
6
```

```
7       // a jelszót bekódoljuk SHA1-es kódolással
8       $kodolt_jelszo = sha1($jelszo);
9
10      // elokeszitjuk az utasitast
11      $stmt = mysqli_prepare( $conn,"INSERT INTO
    ↪ felhasznalok(felhasznalonev, email, keresztnev,
    ↪ vezeteknev, jelszo) VALUES (?, ?, ?, ?, ?)");
12
13      // bekotjuk a parametereket (igy biztonsagosabb az
    ↪  adatkezeles)
14      mysqli_stmt_bind_param($stmt, "sssss",
    ↪ $felhasznalonev, $email, $keresztnev, $vezeteknev,
    ↪  $kodolt_jelszo );
15
16      return mysqli_stmt_execute($stmt);
17      mysqli_close($conn);
18    }
19    return false;
20  }
```

23.2.2. Új bejegyzés felvitele

A következő függvény, amit megvalósítunk, az új bejegyzés felvitele az adatbázisba. Ez a függvény csak három paramétert vár, a felhasználónevet, a beszúrandó szöveget és a hírfolyam azonosítót, amelyhez az üzenet tartozik. A korábban látott módon itt is előkészített `INSERT` utasítással fogjuk az adatokat az adatbázisba beszúrni. Az adatok „tisztítását” a *vezérlő* elem végzi el, így feltételezhetjük, hogy az függvénynek átadott adatok már mentesek az SQL befecskendezésektől. Az aktuális időbélyeget a MySQL `NOW()` függvényével adjuk meg.

Ami ebben a függvényben újdonságként jelenik meg, az a `munkamenetet_frissit()` (leendő) függvényünk lesz. Minden üzenet felvitelekor frissíteni fogjuk a munkamenetünket, hogy megelőzzük azok eltérítését.

Az `uj_bejegyzes()` függvény

```
1   function uj_bejegyzes($felhasznalo, $hirfolyam,
    ↪ $szoveg) {
2     // csatlakozunk az adatbazishoz
3     $conn = forum_csatlakozas();
4
5     if ( $conn ) {
```

```
6       echo $felhasznalo . "<br/>";
7       echo $szoveg . "<br/>";
8
9       // elokeszitjuk az utasitast
10      $stmt = mysqli_prepare( $conn,"INSERT INTO uzenet(
    ↪ felhasznalonev, tartalom, mikor,
    ↪ hirfolyam_azonosito) VALUES (?, ?, NOW(), ?)") or
    ↪ die(mysqli_error($conn));
11
12      // bekotjuk a parametereket (igy biztonsagosabb az
    ↪  adatkezeles)
13      mysqli_stmt_bind_param($stmt, "ssd", $felhasznalo,
    ↪  $szoveg, $hirfolyam );
14
15      $state = mysqli_stmt_execute($stmt) or die(
    ↪ mysqli_stmt_error($stmt));
16      echo $state;
17
18      mysqli_close($conn);
19      $sikeres = munkamenetet_frissit($felhasznalo);
20      if ( $sikeres != false ) {
21        return true;
22      }
23    }
24    return false;
25  }
```

Megjegyezzük, hogy csak a példaalkalmazás kedvéért írattuk ki a hibákat a képernyőre. Ezt érdemes kerülni, hogy a felhasználók ne lássák ezeket a hibákat, mivel azok támadási felületet nyújthatnak.

23.2.3. Munkamenetek frissítése

A munkamementek frissítése már előkerült az előző függvény során. Nézzük most meg, hogy hogyan történik ez! Nagyon fontos, hogy szem előtt tartsuk az adatbiztonság kérdését, ezért adatbázis-szinten is támogatjuk a rendszerünket a munkamenetek kezelésével. A példában mind a szerver oldali munkamenet (*session*) azonosítót, mind pedig a kliens oldali süti (*cookie*) tokent is eltároljuk az adatbázisban. Erre szolgál a `felhasznalok` táblában lévő `sessionid` és `cookie` oszlop. Megjegyezzük, hogy amikor a felhasználó nincs bejelentkezve, akkor ezen mezők értéke `null`.

A munkameneteket a `session_start()` függvénnyel hozzuk létre, amikor a felhasználó megnyitja a bejelentkező vagy a regisztráció oldalát, de bármely

weboldalon, ha kezelni és követni akarjuk a munkamenetet a `session_start()` függvénnyel kell indítanunk. Ekkor a munkamenetünkhöz – ha még nem létezik – létrejön egy egyedi munkamenet azonosító. A `session_start()` függvényt tehát majd a webes űrlapokkal foglalkozó programrészleteknél fogjuk csak meghívni, most a modell-függvényeknél nem jön elő. Azért tértünk ki mégis erre, mert a modell-függvényekben valósítjuk meg a munkamenet frissítését.

A munkamenetek frissítését a `munkamenet_frissitese()` függvénnyel hajtjuk végre. A függvény paramétere a felhasználó azonosítója (jelen esetben a `felhasznalonev`). A munkamenet-azonosító újragenerálást a `session_regenerate_id()` PHP-függvény végzi, amely visszatérési értékként megadja az új munkamenetazonosítót. Az új süti token azonosítót úgy állítjuk elő, hogy az új munkamenet-azonosítóhoz hozzáfűzünk egy ötjegyű véletlen számot, majd ezt a karaktersorozatot bekódoljuk SHA1-es kódolással (így fix 40 hosszú hexadecimális karakterlánc lesz a tokenünk)[1]. A `mysqli_prepare()` függvénnyel előkészítjük az `UPDATE` utasítást, hozzákötjük a külső paramétereket: a munkamenet-azonosítót, a süti tokent és a felhasználónevet, majd a `mysqli_stmt_execute()` utasítással végrehajtjuk a módosítást. Ellenőrizzük, hogy az utasítás egyetlen sort érintett-e, mert különben hiba van a rendszerben! A sütik létrehozásához szükségünk lesz néhány paraméterre: a süti nevére, az értékére, a lejáratára, hogy biztonságos-e. Ha ezeket a paramétereket beállítottuk, akkor a `setcookie()` függvénnyel létrehozzuk az új sütit. Az alábbi kódrészlet ezeket a műveleteket mutatja be.

A `munkamenetet_frissit()` függvény

```
1  function munkamenetet_frissit( $felhasznalo ) {
2    // csatlakozunk az adatbazishoz
3    $conn = forum_csatlakozas();
4
5    if ( $conn ) {
6      session_regenerate_id(true); // törli a régi
   ↪ munkamenetet és újat generál
7      $uj_munkamenet_azonosito = session_id();
8      $uj_tokenID = sha1($uj_munkamenet_azonosito . rand
   ↪ (10000,99999));
9
10     $stmt = mysqli_prepare($conn, "UPDATE felhasznalok
   ↪  SET sessionid = ?, cookie = ? WHERE
```

[1] Az SHA1-es kódolást sokáig nem tudták visszafejteni, nemrégiben azonban elkészítettek egy visszafejtési algoritmust. Bár a példában ezt használjuk, célszerű, olyan kódolást választani, amely még nem visszafejthető.

```
    ↪ felhasznalonev = ?");
11      mysqli_stmt_bind_param($stmt, "sss",
    ↪ $uj_munkamenet_azonosito, $uj_tokenID,
    ↪ $felhasznalo);
12      mysqli_stmt_execute($stmt);
13      $modositott_sorok_szama =
    ↪ mysqli_stmt_affected_rows($stmt);
14      mysqli_close($conn);
15      if ( $modositott_sorok_szama == 1 ) {
16        $sutinev = 'forumsuti';
17        $ertek = $uj_tokenID;
18        $lejar = 0;
19        $biztonsagos = TRUE;
20        $httponly = TRUE;
21        if ( setcookie($sutinev, $ertek, $lejar ) ){
22          return $uj_tokenID;
23
24        } else {
25          return false;
26        }
27      } else {
28        return false; // hiba történt, mert nem talált
    ↪ megfelelő sort
29      }
30    } else {
31      return false;
32    }
33  }
```

23.2.4. Felhasználóhoz tartozó munkamenet azonosító lekérése

A következőkben egy egyszerű de fontos segédfüggvényt készítünk el. Ez a függvény visszaadja egy adott felhasználóhoz tartozó munkamenet-azonosítót és süti tokent. A függvényünk a `felhasznalonev` értékét kéri paraméterként. Maga a lekérés egy `SELECT` utasítással történik, mivel azonban paramétert tartalmaz, úgy biztonságos, ha nem sztringként fűzzük hozzá a lekérdezéshez, hanem paraméterként kötjük be az erre szolgáló `mysqli_stmt_bind_param()` függvénnyel. A függvényünk egy asszociatív tömbben adja vissza a munkamenet-azonosítót és a süti tokent. Feltételezhetően a lekérdezés egyetlen sort ad vissza. A kapott mezőértékeket a `mysqli_stmt_bind_result()` PHP függvénnyel változókhoz rendeljük, és a `mysqli_stmt_fetch()` PHP függ-

vénnyel feldolgozzuk az eredmény halmaz egyetlen sorát, az eredményhez kötött változók ekkor kapnak értéket.

A `felhasznalo_munkamenete()` függvény

```
1  function felhasznalo_munkamenete($felhasznalo) {
2    // csatlakozunk az adatbazishoz
3    $conn = forum_csatlakozas();
4
5    if ( $conn ) {
6      $stmt = mysqli_prepare($conn,"SELECT sessionid,
   ↪ cookie FROM felhasznalok WHERE felhasznalonev = ?"
   ↪ );
7      mysqli_stmt_bind_param($stmt, "s", $felhasznalo);
8      mysqli_stmt_execute($stmt);
9      mysqli_stmt_bind_result($stmt,
   ↪ $munkamenet_azonosito, $tokenID);
10     mysqli_stmt_fetch($stmt);
11     $tomb = array("munkamenet_azonosito" =>
   ↪ $munkamenet_azonosito,
12     "tokenID" => $tokenID );
13
14     mysqli_close($conn);
15
16     return $tomb;
17   } else {
18     return false;
19   }
20 }
```

23.2.5. Bejelentkezés és kijelentkezés

A következőkben megnézzük, hogyan a bejelentkezésnél és kijelentkezésnél használt adatbázis-műveleteket.

A `bejelentkezes()` függvényünk két paramétert vár: az e-mail címet és a beírt jelszót. Fontos megjegyeznünk, hogy ezeket az adatokat a *vezérlő* kódrészleteknél fogjuk tisztítani, tehát feltételhezjük, hogy az itt megadott adatok „tiszták”. A függvényen belül a beírt jelszót le kell kódolnunk SHA1-es kódolással, hiszen az adatbázisban ebben a formában tároljuk az adatot. A bejelentkezési adatoknak megfelelően egy `SELECT` utasítással megnézzük, hogy létezik-e a olyan felhasználó, akinek e-mail címe és jelszava megegyezik a megadottal. Mivel külső paramétert használunk, emiatt a `SELECT` utasítást előkészítjük a `mysqli_prepare()` PHP függvénnyel és hozzákötjük a kívülről érkező adatot. Az eredményhez most is változót rendelünk a

`mysqli_stmt_bind_result()` függvénnyel, amely csak a `mysqli_stmt_fetch()` függvény meghívásakor kap értéket. Fontos, hogy mivel most végeztük el a bejelentkeztetést (egy új oldalra fogunk lépni a felhasználói felületen), hívjuk meg a `munkamentet_frissit()` függvényt a munkamenet frissítéséhez sikeres bejelentkezés esetén.

A `bejelentkezes()` függvény

```
1function bejelentkezes($email, $beirt_jelszo) {
2  // csatlakozunk az adatbazishoz
3  $conn = forum_csatlakozas();
4
5  if ( $conn ) {
6    // jelszo atalakitas
7    $kodolt_jelszo = sha1($beirt_jelszo);
8
9    // elokeszitjuk az utasitast
10   $stmt = mysqli_prepare( $conn,"SELECT felhasznalonev
     ↪  FROM felhasznalok WHERE email = ? AND jelszo = ?"
     ↪ ) or die(mysqli_error($conn));
11   mysqli_stmt_bind_param($stmt, "ss", $email,
     ↪ $kodolt_jelszo); // megadjuk a parametert
12   mysqli_stmt_execute($stmt) or die(mysql_error());
     ↪       // lefuttatjuk az utasitast
13
14   mysqli_stmt_store_result($stmt);
15   mysqli_stmt_bind_result($stmt, $felhasznalo); // az
     ↪ eredmenyt egy valtozohoz kotjuk
16   mysqli_stmt_fetch($stmt); // kiolvassuk az elso sort
     ↪  az eredmenybol
17   echo mysqli_stmt_num_rows($stmt);
18
19   mysqli_close($conn);
20
21   if (mysqli_stmt_num_rows($stmt) == 1) {
22     munkamenetet_frissit($felhasznalo);
23     return $felhasznalo;
24   }
25
26   // nem sikerult az azonositas, mert akkor mar nem
     ↪ kerulne ide a vezerles
27   return false;
28 }
29 return false;
30}
```

A kijelentkezés művelete valamivel egyszerűbb. Adatbázis-szinten csu-

pán az aktuálisan bejelentkezett felhasználó sorában `NULL`-ra kell állítani a `sessionid` és a `cookie` mező értékét. Rendszer-szinten pedig meg kell szüntetni a sütiket. (A munkamenet-azonosítót nem itt szüntetjük meg.) A `kilepes()` függvényünk egyetlen paramétere a felhasználó azonosító.

A `kilepes()` függvény

```
1  function kilepes($felhasznalo) {
2    // csatlakozunk az adatbazishoz
3    $conn = forum_csatlakozas();
4
5    if ( $conn ) {
6      $stmt = mysqli_prepare($conn, "UPDATE felhasznalok
   ↪  SET sessionid = NULL, cookie = NULL WHERE
   ↪ felhasznalonev = ?") ;
7      mysqli_stmt_bind_param($stmt, "s", $felhasznalo);
8      $sikeres1  = mysqli_stmt_execute($stmt) or die(
   ↪ print mysqli_stmt_error($stmt));
9      $sikeres2 = setcookie('forumsuti', "", 0);  // tör
   ↪ öljük a sütiket
10     mysqli_close($conn);
11     return $sikeres1 && $sikeres2;
12   } else {
13     return false;
14   }
15 }
16
```

23.2.6. Bejegyzések listázása

A bejegyzések listázását is a modell-függvények között kezeljük. Erre szolgáló függvényünk meglehetősen egyszerű. A függvény egyetlen paramétere a felhasználó azonosítója, visszatérési értéke pedig a lekérdezés eredménye. Ebben az esetben a lekérdezésünk feltehetően több sort ad vissza, ezért az eredményhalmazt a `mysqli_stmt_get_result()` PHP függvénnyel kapjuk meg. A lekérdezés készítése kapcsán fontos, hogy itt csak azokat a bejegyzéseket listázzuk ki, amely azokhoz a hírfolyamokhoz tartoznak, amelyekre a felhasználó feliratkozott.

A `bejegyzeseket_listaz()` függvény

```
1  function bejegyzeseket_listaz($felhasznalo) {
2    // csatlakozunk az adatbazishoz
3    $conn = forum_csatlakozas();
```

```
4
5     if ( $conn ) {
6       // elokeszitjuk az utasitast
7       $stmt = mysqli_prepare( $conn,"SELECT mikor,
    ↪ uzenet.felhasznalonev AS irta, tartalom FROM
    ↪ uzenet, koveti WHERE uzenet.hirfolyam_azonosito =
    ↪ koveti.hirfolyam_azonosito AND koveti.
    ↪ felhasznalonev = ? ORDER BY mikor DESC");
8       mysqli_stmt_bind_param($stmt, "s", $felhasznalo);
9       mysqli_execute($stmt) or die (mysqli_stmt_error(
    ↪ $stmt));
10      $result = mysqli_stmt_get_result($stmt);
11      return $result; //mysqli_store_result($conn);
12    }
13    return false;
14  }
```

23.2.7. Új hírfolyam-követés felvitele

Ezt a műveletet is a modell-függvények között valósítjuk meg. Az `uj_kovetes()` függvényünk két paramétert kap a felhasználó nevét illetve a hírfolyam azonosítóját. Az `INSERT` utasítást a `mysqli_prepare()` függvénnyel készítjük elő, és a két paramétert a `mysqli_stmt_bind_param()` függvénnyel kötjük hozzá. Végül a `mysqli_stmt_execute()` függvénnyel futtatjuk le.

Az `uj_kovetes()` függvény

```
1  function uj_kovetes($felhasznalo, $hirfolyam) {
2    // csatlakozunk az adatbazishoz
3    $conn = forum_csatlakozas();
4
5    if ( $conn ) {
6      $stmt = mysqli_prepare("INSERT INTO koveti (
    ↪ felhasznalonev, hirfolyam_azonosito) VALUES (?, ?)
    ↪ ");
7      $stmt = mysqli_stmt_bind_param($stmt, 'ss',
    ↪ $felhasznalo, $hirfolyam_azonosito);
8      mysqli_stmt_execute($stmt) or die (mysql_error());
9      mysqli_close($conn);
10     return true;
11   }
12   // nem sikerult az azonositas
```

```
13    return false;
14  }
```

23.2.8. Új hírfolyam indítása

Az új hírfolyam indításánál nem csupán a hírfolyam-rekordot kell felvinnünk, hanem a hozzátartozó kulcsszavakat is. Ez azt jelenti, hogy két táblába kell beszúrnunk rekordokat. Először a hírfolyam rekordját kell létrehoznunk, utána pedig a kulcsszavakat kell beszúrnunk. Ehhez azonban szükségünk lesz a hírfolyam automatikusan létrehozott azonosítójára. A generált azonosítót a `mysqli_insert_id()` függvénnyel kapjuk meg. A kulcsszavakat összefűzött, vesszővel elválasztott sztringként adjuk át a függvénynek és az `explode()` PHP függvénnyel bontjuk szét. Ebben a függvényben oldjuk meg azt is, hogy a felhasználó az általa létrehozott hírfolyamokat követi is.

Az `uj_hirfolyam()` függvény

```
1  function uj_hirfolyam($megnevezes,
   ↪ $osszefuzott_kulcsszavak, $felhasznalo) {
2    // adatbazis csatlakozas
3    // csatlakozunk az adatbazishoz
4    $conn = forum_csatlakozas();
5    if ( $conn ) {
6
7      // elokeszitjuk az utasitast
8      $stmt = mysqli_prepare( $conn,"INSERT INTO
   ↪ hirfolyam (megnevezes) VALUES (?)");
9      mysqli_stmt_bind_param($stmt, "s", $megnevezes);
10     mysqli_execute() or die(mysql_error());
11     // kulcsszavak beszúrása
12     $hirfolyam_azonosito = mysqli_insert_id($conn);
13     $kulcsszavak = expolode(',' $kulcsszo_sztring);
14     foreach ($kulcsszavak as $index => $kulcsszo) {
15       $stmt = mysqli_prepare($conn, 'INSERT INTO
   ↪ kulcsszavak VALUES (?, ?)');
16       mysqli_stmt_bind_param($hirfolyam_azonosito,
   ↪ $kulcsszo);
17       mysqli_stmt_execute() or die(mysql_error());
18     }
19     mysqli_close($conn);
20     $sikeres = uj_kovetes($felhasznalo,
   ↪ $hirfolyam_azonosito);
```

```
21      if ( $sikeres == false ) {
22        return false;
23      }
24      return true;
25    }
26    return false;
27  }
```

23.2.9. A felhasználó követett hírfolyamai

Egy újabb modell-függvényt hozunk létre a felhasználó követett hírfolyamainak listázására. Ez a függvény csak a felhasználó nevét kapja paraméterül. Most is elő kell készítenünk a lekérdezést a `mysqli_prepare()` függvénnyel, majd bekötni a paraméterként kapott felhasználónevet a lekérdezésbe. A függvény visszatérési értéke egy asszociatív tömb lesz, ahol a kulcs a hírfolyamazonosító, a hozzátartozó érték pedig a megnevezés lesz.

A `felhasznalo_kovetett_hirfolyamai()` függvény

```
1function felhasznalo_kovetett_hirfolyamai($felhasznalo)
    ↪ {
2  // csatlakozunk az adatbazishoz
3  $conn = forum_csatlakozas();
4
5  if ( $conn ) {
6    $stmt = mysqli_prepare($conn,"SELECT azonosito,
    ↪ megnevezes FROM koveti, hirfolyam WHERE koveti.
    ↪ hirfolyam_azonosito = hirfolyam.azonosito AND
    ↪ koveti.felhasznalonev = ?");
7    mysqli_stmt_bind_param($stmt, 's', $felhasznalo);
8    mysqli_stmt_execute($stmt) or die (mysql_error());
9    $hirfolyamlista = mysqli_stmt_get_result($stmt);
10   mysqli_close($conn);
11   return $hirfolyamlista;
12 }
13 // nem sikerult az azonositas
14 return false;
15}
```

23.3. Űrlapok létrehozása

A modell-függvények előkészítése után ebben az alfejezetben az űrlapokat állítjuk össze. Tisztában vagyunk vele, hogy számos függvénykönyvtár létezik webes űrlapok elkészítésére, azonban mi példáinkban a tiszta HTML és PHP űrlap-elemekre hagyatkozunk. Ennek oka az, hogy nehéz kiválasztani, hogy hosszútávon mi lesz az a technológia, amely fenn tud maradni. Jegyzetünket több éves használatra szánjuk, emiatt olyan alapokat akarunk adni az Olvasónak, amelyet a későbbiekben is használni tud, és erre a HTML és a PHP alkalmas. A könnyebb áttekinthetőség és követhetőség kedvéért az űrlap adatainak feldolgozását (vagyis az MVC tervezés séma *vezérlő* elemeit) is ebben a fejezetben tárgyaljuk.

Az űrlapok esetében minden űrlapot külön fájlban készítünk el. Ezek kiterjesztése lehet `.php` vagy `.html` is. Ha HTML mellett döntünk, akkor mellőznünk kell a PHP utasításokat, és tudomásul kell vennünk, hogy weboldalunk csak statikus tartalommal bír.

Az űrlap adatait a következőképpen dolgozzuk fel: minden űrlap-elem rendelkezik egy `name` paraméterrel, amelyet egyfajta változóként tudunk majd használni. Az űrlap-elem ebben tárolja az értékeket. Az űrlapokat a HTML `<form method="`*`metódus`*`", action="`*`feldolgozó oldal`*`"> ...</form>` nyitó- és zárócímke között adjuk meg. A `method` attribútum az adattovábbításra utal, amely nagyon gyakran `GET` vagy `POST`, az `action` attribútumnál pedig azt adjuk meg, hogy az itt beírt értékeket melyik oldal fogja feldolgozni. Attól függően, hogy az adattovábbítás `GET` vagy `POST`, a feldolgozó oldalon az űrlap adatait `$_GET[]` vagy `$_POST[]` szuperglobális tömbben kapjuk meg, úgy, hogy a tömb kulcsa az űrlap-elem `name` attribútumában megadott név, a kulcshoz tartozó érték pedig az űrlap-elem tartalma.

Röviden tekintsük át a leggyakrabban használt űrlap-elemeket:

`<input type="text" />`: Egysoros beviteli mező.

`<input type="password" />`: Jelszó mező, beviteli mező a tartalom elrejtésével.

`<textarea ></textarea>`: Többsoros beviteli mező.

`<input type="checkbox" />`: Jelölőnégyzet.

`<input type="radio" />`: Rádiógomb, több választási lehetőség közül egyet választhat a felhasználó.

`<select ></select>`: Lenyíló lista. A nyitó- és zárócímkék között `<option>...</option>` címkékkel adhatjuk meg a lenyíló lista elemeit.

`<input type="submit" />`: Adatküldésre szolgáló gomb. Hatására a vezérlés az `action` paraméterben megadott oldalhoz kerül.

23.3.1. Felhasználó regisztrációja

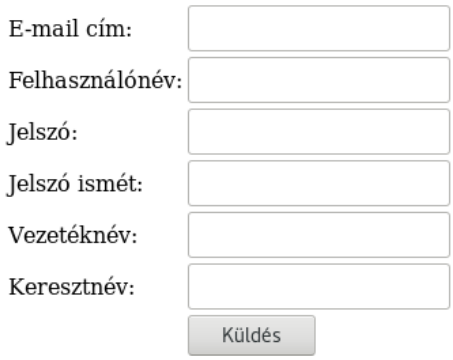
A felhasználó regisztrációjához nem kell dinamikus tartalmat használnunk, de az egységesség kedvéért a fájlkiterjesztést `.php`-ként adjuk meg. Az adattovábbítást `POST`-tal végezzük, így a továbbított adatok nem fognak látszódni a böngésző címsorában. A feldolgozó fájlunk pedig a *regisztral.php* lesz.

Az űrlapon bekérjük a felhasználó nevét, e-mail címét, a vezeték- és keresztnevét, valamint jelszavát. A jelszót kétszer kérjük be, hogy ezzel megerősítse a begépelt karaktersorozatot.

A `regisztracio.php` fájl

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
  ↪ charset=UTF8" >
  <style>
    p {display: table-row;}
    label {display: table-cell;}
    input {display: table-cell;}
  </style>
</head>
<body>
  <form method="POST" action="regisztral.php">
  <p>
  <label for="email">E-mail cím:</label>
  <input type="text" name="email">
  </p>
  <p>
  <label for="felhasznalonev">Felhasználónév:</label>
  <input type="text" name="felhasznalonev">
  </p>
  <p>
  <label for="jelszo">Jelszó:</label>
  <input type="password" name="jelszo">
  </p>
  <p>
  <label for="jelszoismet">Jelszó ismét:</label>
  <input type="password" name="jelszoismet">
  </p>
  <p>
  <label for="vezeteknev">Vezetéknév:</label>
  <input type="text" name="vezeteknev">
```

```
    </p>
    <p>
    <label for="keresztnev">Keresztnév:</label>
    <input type="text" name="keresztnev">
    </p>
    <p>
    <label></label>
    <input type="submit" name="Regisztrál">
    </p>
    </form>
  </body>
  </html>
```



23.1. ábra. A Fórum regisztrációs oldala.

Amikor a felhasználó megadta az adatait, akkor a vezérlés a *regisztral.php* oldalra kerül. Ez az oldal fogja feldolgozni a regisztrációs űrlap adatait. Ennek a fájlnak mindenképpen `.php` kiterjesztéssel kell rendelkeznie, mivel PHP forráskódot tartalmaz.

A fájl elején behívjuk a *db_fuggvenyek.php* fájlt az `require_once()` függvénnyel. Ezt követően lekérjük a szuperglobális `$_POST` tömbből a paramétereket és lokális változókban tároljuk el.

Adatbiztonság szempontjából nagyon fontos, hogy ezekről a beírt adatokról még semmit nem tudunk, vagyis nem szúrhatjuk csak úgy be az adatbázisba. Előtte ellenőriznünk kell, esetleg át kell alakítanunk. Fontos programozói hozzáállás, hogy a „nyers” és a tisztított adatot külön változóban tároljuk és a változó neve tükrözze ezt. Erre jó megoldás az, ha a még nem ellenőrzött

változókat `piszkos_` az ellenőrzött vagy tisztított változókat pedig a `tiszta_` prefixszel lássuk el [5]. A `$_POST` tömbből az értékeket tehát `piszkos_` prefixű változókba mentjük. Emlékezzünk rá, hogy az űrlap-elemek `name` attribútumában megadott értékek lesznek a `$_POST` szuperglobális asszociatív tömb kulcsai.

Az átküldött értékeket ellenőrizni kell. Ennek első lépéseként megvizsgáljuk az `isset()` PHP függvénnyel, hogy van-e a változónak értéke (különben a változó nem létezik). Ha minden változó kapott értéket, akkor következhet a további ellenőrzés. Vizsgáljuk meg a `preg_match()` PHP függvénnyel, hogy a beírt e-mail cím megfelel-e az e-mail cím formátumnak! A jelszónak legalább 8 karakterből kell állnia, ezt is ellenőrizzük. Miután ellenőriztük az e-mail formátumot és a jelszó hosszát, akkor a legegyszerűbb megoldás az adatok tisztítására, hogy a speciális karaktereket lecseréljük a karakterek kódjaira. Ezt a `htmlspecialchars()` PHP-függvény végzi. Ha minden értéket átalakítottunk, akkor az `uj_felhasznalo_regisztralasa()` modellfüggvényünkkel felvesszük az új rekordot.

A `regisztral.php` oldal

```
1  <?php
2  require_once("db_fuggvenyek.php");
3
4  $piszkos_felhasznalonev = $_POST["felhasznalonev"];
5  $piszkos_email = $_POST["email"];
6  $piszkos_jelszo = $_POST["jelszo"];
7  $piszkos_jelszoismet = $_POST["jelszoismet"];
8  $piszkos_vezeteknev = $_POST["vezeteknev"];
9  $piszkos_keresztnev = $_POST["keresztnev"];
10
11 if ( isset($piszkos_email) && isset(
   ↪ $piszkos_felhasznalonev) &&
12 isset($piszkos_jelszoismet) && isset($piszkos_jelszo)
   ↪ &&
13 isset($piszkos_vezeteknev) && isset(
   ↪ $piszkos_keresztnev)  ) {
14
15   $email_regex = '^[\w\.-]{1,}\@(\da-zA-Z-]){1, }\.)
   ↪ {1, }[\da-zA-Z-]+$';
16   if ( preg_match($email_regex, $piszkos_email,
   ↪ $illeszkedes) == 0 ) {
17     echo "Hibás az e-mail cím!";
18   } else if ( preg_match("/^[a-zA-Z0-9@*#_]{8,}$/",
   ↪ $piszkos_jelszo) == 0 ) {
19     echo "A jelszónak legalább 8 karakterből kell á
   ↪ llnia, kis- és nagybetűket, számokat, valamint
```

```
    ↪ speciális karaktereket tartalmazhat!";
20   } else {
21      // egyszerű átalakítást végzünk
22      $tiszta_email = htmlspecialchars($piszkos_email);
23      $tiszta_felhasznalonev = htmlspecialchars(
    ↪ $piszkos_felhasznalonev);
24      $tiszta_jelszo = htmlspecialchars($piszkos_jelszo)
    ↪ ;
25      $tiszta_jelszoismet = htmlspecialchars(
    ↪ $piszkos_jelszoismet);
26      $tiszta_vezeteknev = htmlspecialchars(
    ↪ $piszkos_vezeteknev);
27      $tiszta_keresztnev = htmlspecialchars(
    ↪ $piszkos_keresztnev);
28
29      if ( $tiszta_jelszo == $tiszta_jelszoismet ) {
30        uj_felhasznalo_regisztralasa($tiszta_email,
    ↪ $tiszta_felhasznalonev,
31        $tiszta_vezeteknev, $tiszta_keresztnev,
    ↪ $tiszta_jelszo);
32      }
33    }
34  }
35  header("Location: index.php");
36  ?>
```

23.3.2. Bejelentkezés

A következő űrlap a bejelentkeztetés, amit elő kell készítenünk. A bejelentkeztetés során ellenőrizni kell a belépési adatokat (felhasználónév és jelszó), valamint ha helyesek az adatok, akkor új munkamenetet kell létrehozni, azt rögzíteni kell az adatbázisban, és a munkamenet-változókba el kell tárolni a felhasználó nevét, mivel azt majd külső kulcsként felhasználjuk a további funkciókhoz.

Az oldal annyiban is különleges, hogy a kezdőlapon (*index.php*) van a bejelentkező felület, másrészről, amikor a felhasználó már bejelentkezett, akkor viszont az üzeneteket listázza ki számára, illetve kínál egy űrlapot új üzenet írására. Azt, hogy van-e bejelentkezett felhasználó onnan fogjuk tudni, hogy a sikeres bejelentkezés után a `$_SESSION[]` szuperglobális tömbbe eltároljuk a felhasználónevét.

Nagyon fontos, hogy az oldal elején indítsunk egy új munkamenetet a `session_start()` PHP függvénnyel! Ezután importáljuk a modell-függvé-

nyeinket, hiszen ezeket fogjuk majd használni. Ezt követően a honlap tartalmi részében ellenőriznünk kell, hogy van-e a bejelentkezett felhasználó, vagyis van-e a `$_SESSION["felhasznalonev"]`-nek van-e értéke. Ha nincs, akkor a bejelentkezési felületet kell mutatnuk. Ez két beviteli mezőt (az e-mail címnek és a jelszónak), illetve egy gombot tartalmaz, amely elküldi az adatokat. A beírt adatokat a *bejelentkezes.php* fogja feldolgozni. Az *index.php*-ból most csak a bejelentkezési űrlapot mutatjuk.

A bejelentkezési űrlap a kezdőlapon

```
<?php
  session_start();
  require_once("db_fuggvenyek.php");
?>
<HTML>
<HEAD>
  <meta http-equiv="content-type" content="text/html;
  ↪ charset=UTF8" >
  <style>
    p {display: table-row;}
    label {display: table-cell;}
    input {display: table-cell;}
  </style>
</HEAD>
<BODY>
<?php
if ( isset($_SESSION["felhasznalonev"]) == false ) {
  echo '<HR/>';
  $urlap  ='';
  $urlap .= '<FORM method="POST" action="bejelentkezes
  ↪ .php">';
  $urlap .= '  <p align="right">';
  $urlap .= '     <label ><a href="regisztracio.php">
  ↪ Regisztráció</a></label>|';
  $urlap .= ' <label>E-mail:</label><input type="text"
  ↪  name="email" />';
  $urlap .= ' <label>Jelszó:</label><input type="
  ↪ password" name="jelszo" />';
  $urlap .= ' <label></label><input type="submit"
  ↪ value="Bejelentkezés" />';
  $urlap .= '  </p>';
  $urlap .= '</FORM>';
  echo $urlap;

} else {
  // a bejelentkezés után látható rész
  ...
```

23.2. ábra. A Fórum bejelentkezési űrlapja.

```
32  }
33  ?>
34  <HR/>
35  </BODY>
36  </HTML>
```

Most tekintsük a feldolgozó részt, vagyis a *bejelentkezes.php* tartalmát! Először is nagyon fontos, hogy a munkamenetet fenntartsuk, ezért a fájl elején meg kell hívni a `session_start()` függvényt. Ezt követően importálni kell a modell-függvényeket. A begépelt e-mail címet és jelszót még piszkos adatként kell kezelnünk. Itt az adatellenőrzés a következőképpen zajlik. Először is megnézzük, hogy a beírt e-mail cím, nem túl hosszú-e, vagyis nem hosszabb-e mint 60 karakter (ezzel már bizonyos SQL befecskendezéseket ki lehet szűrni). Ezután azt vizsgáljuk, hogy a jelszó megfelelő karaktereket tartalmaz-e, illetve kellő hosszú-e. Ha igen, akkor az e-mail címet a korábban látott egyszerű módon „tisztítjuk”, a speciális karakterek átalakításával. A jelszót nem kell tisztítani, mivel egyrészről az SHA1-es kódolás után ártalmatlan karaktersorozat lesz, másrészről pedig a reguláris kifejezéssel történő ellenőrzés után kiderült, hogy nem tartalmaz szóköz karaktert. Meghívjuk a `bejelentkezes()` modell-függvényünket, amely visszaadja a felhasználónevet.

A `bejelentkezes.php` fájl tartalma

```
1   <?php
2     session_start();
3     require_once("db_fuggvenyek.php");
4
5     $piszkos_email = $_POST["email"];
6     $piszkos_jelszo = $_POST["jelszo"];
7
8     // ellenőrzöm az e-mail hosszát és tartalmát
9     $email_length = strlen($piszkos_email);
10
11    $tiszta_jelszo = htmlspecialchars($piszkos_jelszo);
```

```
12
13    $felhasznalo = bejelentkezes($tiszta_email,
      ↪ $tiszta_jelszo);
14    $_SESSION["felhasznalonev"] = $felhasznalo;
15    header("Location: index.php");
16
17    // ha nem sikerül a bejelentkezés
18    echo 'Hibás azonosító vagy jelszó!' . '<BR/>';
19    echo '<a href="index.php">Vissza a kezdőlapra ... </
      ↪ a>';
20  ?>
```

23.3.3. Üzenet beírása

Az üzenet beírására szolgáló űrlapot is az *index.php* tartalmazza, azonban ez csak akkor jelenik meg, amikor a `$_SESSION["felhasznalonev"]` értéke nem üres. Ekkor az üzenet szövegét egy többsoros szövegmezőbe lehet beírni. Egy lenyíló listával ki kell választani, hogy az üzenet melyik hírfolyamhoz kapcsolódik. A beviteli mező alatt azok az üzenetek látszódnak, amelyek a felhasználó által követett hírfolyamokhoz tartoznak.

Az üzenet küldésére szolgáló űrlap a kezdőoldalon

```
1   <?php
2     session_start();
3     require_once("db_fuggvenyek.php");
4   ?>
5   <HTML>
6   <HEAD>
7   <meta http-equiv="content-type" content="text/html;
    ↪ charset=UTF8" >
8   <style>
9     p {display: table-row;}
10    label {display: table-cell;}
11    input {display: table-cell;}
12  </style>
13  </HEAD>
14  <BODY>
15
16  <?php
17    if ( isset($_SESSION["felhasznalonev"]) == false ) {
18      // bejelentkezesi urlap
19      ...
```

```
20    } else {
21      $kovetett_hirfolyamok =
    ↪ felhasznalo_kovetett_hirfolyamai($_SESSION["
    ↪ felhasznalonev"]);
22      $lenyilo = '<select name="hirfolyam">';
23      while ( $sor = mysqli_fetch_assoc(
    ↪ $kovetett_hirfolyamok) ) {
24        $lenyilo .= '<option value="'.$sor["azonosito"].
    ↪ '">' . $sor["megnevezes"] . '</option>';
25      }
26      $lenyilo .= '</select>';
27      echo '<p align="right"><a href="kilepes.php">Kilép
    ↪ és</a></p>';
28      echo '<HR/>';
29      $urlap  ='';
30      $urlap .= '<FORM method="POST" action="ujbejegyzes
    ↪ .php">';
31      $urlap .= '  <p>';
32      $urlap .= '  <label for="szoveg">Új bejegyzés:</
    ↪ label><textarea cols="100" rows="5" name="szoveg"
    ↪ ></textarea>';
33      $urlap .= '  </p>';
34      $urlap .= '  <p>';
35      $urlap .= '  <label>Követett hírfolyamok:</label>'
    ↪ .$lenyilo;
36      $urlap .= '  </p>';
37      $urlap .= '  <p>';
38      $urlap .= '  <label></label><input type="submit"
    ↪ value="Felvisz" />';
39      $urlap .= '  </p>';
40      $urlap .= '</FORM>';
41      echo $urlap;
42      echo '<H1>Legutóbbi bejegyzések</H1>';
43
44      $bejegyzesek = bejegyzeseket_listaz($_SESSION["
    ↪ felhasznalonev"]);
45      //print_r($bejegyzesek);
46      if ( mysqli_num_rows($bejegyzesek) == 0 ) {
47        echo 'Még nincs bejegyzés a fórumban';
48      } else {
49        while ( $sor = mysqli_fetch_assoc($bejegyzesek)
    ↪ ) {
50          echo '<table border="1">';
51          echo '<tr><td align="left">'.$sor["irta"].'</
    ↪ td><td align="right">'.$sor["mikor"].'</td></tr>';
```

```
52          echo '<tr><td align="justified" colspan="2">'.
    ↪ $sor["tartalom"].'</td></tr>';
53          echo '</table><br/>';
54        }
55      }
56    }
57  ?>
58  <HR/>
59  </BODY>
60  </HTML>
```

23.3. ábra. A Fórum üzenetírási felülete

Az üzenetek felvitelét az adatbázisba az *ujbejegyzes.php* végzi. Ebben kell megvalósítanunk az adatok tisztítását, hiszen egy többsoros beviteli mezőbe bármit beírhatott a felhasználó.

Nagyon fontos, hogy itt is folytatjuk a munkamenetet, tehát a fájlt a `session_start()` függvény meghívásával kezdjük. A szöveget és a hírfolyamazonosítót a `$_POST[]` szuperglobális asszociatív tömb tartalmazza. A szöveget tisztítani kell, az azonosítót nem, mivel az az adatbázisból és nem felhasználó által megadott (és változtatható) inputból származik. Fontos, hogy ellenőrizzük a munkamenetet. Bár alkalmazásunk nagyon kicsi, ezt az oldalt frissítjük leginkább, ezért itt kell ellenőriznünk, hogy nem térítették-e el a munkamenetünket. Elővigyázatosságból újrageneráljuk a munkamenet-azonosítót és megújítjuk a süti tokent is.

Az `ujbejegyzes.php` fájl tartalma

```php
<?php
session_start();
include_once("db_fuggvenyek.php");
$piszkos_szoveg = $_POST["szoveg"];
$hirfolyam = $_POST["hirfolyam"];
$felhasznalo = $_SESSION["felhasznalonev"];

if ( isset($piszkos_szoveg) && isset($felhasznalo) ) {
  // egy egyszerű megoldással átalakítom a szöveget,
  // hogy biztonságosabb legyen
  $tiszta_szoveg = htmlspecialchars($piszkos_szoveg);
  // ellenorizzuk a munkamenetet
  // ha nem jó, akkor kilépünk
  $munkamenet_azonositok = felhasznalo_munkamenete(
  $felhasznalo);
  if ( session_id() == $munkamenet_azonositok["
  munkamenet_azonosito"] &&
  $_COOKIE['forumsuti'] == $munkamenet_azonositok["
  tokenID"] ) {
    uj_bejegyzes($felhasznalo, $hirfolyam,
  $tiszta_szoveg);
    header("Location: index.php");
  } else { // eltérítették a munkamenetet
    print_r($munkamenet_azonositok);
    kilepes($felhasznalo);
    if ( session_destroy() ) {
      session_unset();
    } else {
      echo "Nem sikerült törölni a munkamenetet.";
    }
  }
} else {
  header("Location: index.php");
}
?>
```

23.3.4. Hírfolyam létrehozása

A hírfolyam létrehozása egy viszonylag egyszerű űrlap, hiszen csak a hírfolyam megnevezését és a kulcsszavakat kell megadnunk a hírfolyamhoz. Ez két beviteli mezőt, illetve egy **Létrehoz** feliratú gombot jelent. A munkameneteket itt is követnünk kell, ezért az űrlap első sora a `session_start()`

lesz.

A hírfolyam létrehozására szolgáló űrlap forráskódja

```
<?php
  session_start();
  require_once("db_fuggvenyek.php");
?>
<HTML>
<HEAD>
<meta http-equiv="content-type" content="text/html;
 ↪ charset=UTF8" >
<style>
  p {display: table-row;}
  label {display: table-cell;}
  input {display: table-cell;}
</style>
</HEAD>
<BODY>
<HR/>
<?php
  if (isset($_SESSION["felhasznalonev"])) {
    $urlap='';
    $urlap.='<form method="POST" action="
 ↪ hirfolyamLetrehozasa.php">';
    $urlap.='<p>';
    $urlap.='<label for="megnevezes">Megnevezés:</
 ↪ label>';
    $urlap.='<input type="text" name="megnevezes"/>';
    $urlap.='</p>';
    $urlap.='<p>';
    $urlap.='<label>Kulcsszavak:</label>';
    $urlap.='<input type="text" name="kulcsszavak"/>';
    $urlap.='</p>';
    $urlap.='<p>';
    $urlap.='<label></label>';
    $urlap.='<input type="submit" name="Létrehoz"/>';
    $urlap.='</label>';
  }
?>
</BODY>
</HTML>
```

Az űrlapadatok feldolgozásánál most is ellenőriznünk kell a felhasználó által beírt adatokat. Egyrészről megnézzük, hogy tartalmaz-e adatot, vagyis létezik-e a `$_POST["megnevezes"]` és a `$_POST["kulcsszavak"]` változóknak értéke. Ha igen, akkor a megnevezést a kulcsszavakat a speciális

karakterek átalakításával „tisztítjuk”. Megjegyezzük, hogy a vesszővel elválasztott kulcsszavakat most elég továbbadnunk az `uj_hirfolyam` modellfüggvénynek, mivel a feldarabolást abban valósítjuk meg. Fontos, hogy itt is ellenőrizzük a munkameneteket. Amennyiben a munkamenet-azonosítók nem egyeznek, a felhasználót kiléptetjük.

A `hirfolyamLetrehozasa.php` forrásfájl

```
1  <?php
2  session_start();
3  require_once('db_fuggvenyek.php');
4
5  $piszkos_megnevezes = $_POST["megnevezes"];
6  $piszkos_kulcsszavak = $_POST["kulcsszavak"];
7  $felhasznalo = $_SESSION["felhasznalonev"];
8
9  if ( isset($piszkos_megnevezes) && isset(
   ↪ $piszkos_kulcsszavak) && $felhasznalo ) {
10   // egy egyszerű megoldással átalakítom a kapott
   ↪ sztringeket,
11   // hogy biztonságosabb legyen
12   $tiszta_megnevezes = htmlspecialchars(
   ↪ $piszkos_megnevezes);
13   $tiszta_kulcsszavak = htmlspecialchars(
   ↪ $piszkos_kulcsszavak);
14
15   // ellenorizzuk a munkamenetet
16   // ha nem jó, akkor kilépünk
17   $munkamenet_azonositok = felhasznalo
18   _munkamenete($felhasznalo);
19   print_r($_COOKIE);
20   print ($_COOKIE['forumsuti']);
21   if ( substr(session_id(), 0,10) ==
   ↪ $munkamenet_azonositok["munkamenet_azonosito"] &&
22   $_COOKIE['forumsuti'] == $munkamenet_azonositok["
   ↪ tokenID"] ) {
23     uj_hirfolyam($tiszta_megnevezes,
   ↪ $tiszta_kulcsszavak, $felhasznalo);
24     header("Location: index.php"); // minden rendben,
   ↪ visszaterunk az index.php-re
25   } else { // eltérítették a munkamenetet
26     kilepes($felhasznalo);
27     if ( session_destroy() ) {
28       session_unset();
29     } else {
30       echo "Nem sikerült törölni a munkamenetet.";
31     }
```

```
32      }
33    } else {
34      header("Location: index.php");
35    }
36    ?>
```

23.4. Fájlok jogosultságainak kezelése

Amikor az űrlapokkal, a vezérlő oldalakkal és a modell-függvényekkel készen vagyunk, a fájlokat töltsük fel a XAMPP **htdocs** könyvtár egy általunk létrehozott könyvtárába. Ezután gondoskodnunk kell arról, hogy akárki ne érje el a modell- és vezérlő-fájlokat. Erre legjobb megoldás a fájlok láthatóságának beállítása. A írási, olvasási és futtatási jogosultságokat a 23.1. táblázat alapján kell beállítani.

	Tulajdonos			Csoport			Bárki		
Fájlok típusa	Olvasás	Írás	Futtatás	Olvasás	Írás	Futtatás	Olvasás	Írás	Futtatás
Modell-fájlok	•	•	•	•	•	•			
Vezérlő-fájlok	•	•	•	•	•	•			
Űrlapok	•	•	•	•	•	•	•	•	•

23.1. táblázat. Jogosultságok beállítása webes alkalmazás fájljaira.

23.5. Összefoglalás

Bár a példa – szándékosan – nem volt nagy, és nem volt bonyolult, az Olvasó megértheti belőle hogyan kell kezelni a munkameneteket, milyen PHP-függvényekkel tud dolgozni MySQL adatbázis esetén, mit jelent az SQL utasítás előkészítése, és hogy hogyan adjon át paramétereket SQL utasításnak úgy, hogy azzal csökkentse az SQL befecskendezések kockázatát. Felhívjuk az Olvasó figyelmét, hogy az itt leírtak csupán egy lehetséges módja a biztonságos webalkalmazások fejlesztésének, számos más technika létezik, amelyeket esetleg speciálisan webes alkalmazás-fejlesztésre kidolgozott függvénykönyvtárakban valósítottak meg. Tankönyvünkben nem foglalkozunk a szerver

elleni támadásokkal azok kivédésével. Ebben a témakörben érdemes elolvasni a [5,20] könyveket. Rendkívül fontosnak tartjuk, hogy az Olvasó egy alkalmazás fejlesztésénél magára vonatkozóan kötelezőnek érezze ezeket az biztonsági elveket!

A szemfüles Olvasó bizonyára észrevette, hogy a példánkból hiányzik a követendő hírfolyamok kiválasztásának lehetősége. Ez szándékos, mivel a **Kérdések és feladatok** fejezetben feladatként adjuk ki.

Kérdések és feladatok

1. Készítsen űrlapot a hírfolyamok követésére és valósítsa meg a hozzá tartozó adatfeldolgozó fájlt!

2. Valósítsa meg az üzenetre történő válasz lehetőségét! Ehhez az adatbázis tervet, és a relációsémákat is változtatni kell! Készítsen új űrlapot a válaszadáshoz!

3. Valósítsa meg az üzenet törlésének lehetőségét az alkalmazásban! Ehhez a funkcióhoz készítenie kell modell-függvényt is, valamint az üzenetek mellé külön törlésre vonatkozó `<form>...</form>` űrlapot.

4. Írassa ki a hírfolyamok mellé, hogy hány darab üzenet található a hírfolyamban!

5. Valósítsa meg az alkalmazásban az üzenetek megjelenítését (szűrését) a kiválasztott hírfolyam szerint!

24. fejezet

Minta alkalmazás fejlesztése SQLite adatbázishoz Java nyelven

Miután az előző fejezetben áttekintettük, hogy hogyan, milyen szempontok és technikai megfontolások figyelembe vételével lehet készíteni PHP-ben webes alkalmazást MySQL adatbázishoz, készítsünk egy minta alkalmazást egy SQLite adatbázishoz Java nyelven.

Habár nagyon sok programozói függvénykönyvtár, és keretrendszer létezik önálló (*stand-alone*) alkalmazások fejlesztéséhez, mi a Java SDK által nyújtott Swing és AWT csomagokat használjuk. Ennek oka, hogy az *Adatbázisok* tantárgyunkat akkor tanulják hallgatóink, amikor már megismerkedtek a Java nyelv alapjaival. Jelen fejezet így is tartogat újdonságokat számukra, hiszen megtanulhatják, hogyan készítsenek grafikus felhasználói felülettel rendelkező alkamazást Java nyelven. Amennyiben az Olvasó még nem ismeri Java nyelv alapjait, szintaxisát, illetve még nem találkozott objektum-orientált programozási nyelvvel, javasoljuk, hogy a fejezet elolvasása előtt ismerkedjen meg ezekkel [13, 15, 17], mivel ennek hiánya jelentősen megnehezíti a megértést.

A fejlesztés során most is a modell-nézet-vezérlő (MVC) tervezési mintát [10, 19] követjük. Mivel ennek az elveit és alapjait az előző fejezetben elsajátítottuk, így egy-egy funkción keresztül fogjuk bemutatni a három szint összekapcsolását. Megjegyezzük továbbá, hogy nem minden funkció megvalósítását mutatjuk be itt, mivel az túlfeszítené a terjedelmi korlátokat. Mutatunk viszont példát beszúrásra, módosításra törlésre, valamint lekérdezésre, amely után már a többi funkció megvalósítása sem fog gondot okozni.

24.1. Az adatbázis létrehozása

A **Programkalauz** példánkban az alábbi sémák szerepelnek:

`PROGRAMOK`(programazonosító, cím, leírás, mikortól, meddig, web, kapcsolat, *helyazonosító*, ártól, árig)
`HELYEK`(helyazonosító, város, cím, hely neve)
`MŰFAJ`(*programazonosító*, műfajmegnevezés)

Hozzuk őket létre SQLite-ban! Az `sqlite3` programnév után megadjuk az adatbázis-fájl nevét (`programkalauz.db`), ezzel létrejön maga a fájl és kapunk egy `sqlite3` parancsértelmezőt. Hozzuk létre a táblákat az alábbi parancsokkal! Érdemes azokat a táblákat létrehozni előbb, amelyekben nincs külső kulcs, mert utólag körülményes beállítani a külső kulcs feltételt az SQLite-ban, mivel jelenleg nincs még `ALTER TABLE` utasítás.

A `Helyek` tábla létrehozása

```
CREATE TABLE Helyek (helyazonosito INTEGER PRIMARY KEY,
    varos TEXT, cim TEXT, hely_neve TEXT);
```

A `Programok` tábla létrehozása

```
CREATE TABLE Programok (programazonosito INTEGER
    PRIMARY KEY, cim TEXT, leiras TEXT, mikortol
    DATETIME, meddig DATETIME, web TEXT, kapcsolat TEXT,
    helyazonosito INTEGER REFERENCES
    Helyek(helyazonosito), artol INTEGER, arig INTEGER);
```

A `Mufaj` tábla létrehozása

```
CREATE TABLE Mufaj(programazonosito INTEGER REFERENCES
    Programok(programazonosito), mufajmegnevezes TEXT,
    PRIMARY KEY ( programazonosito, mufajmegnevezes));
```

A példánkban a programok beszúrásán, módosításán és törlésén keresztül mutatjuk be az egyes fejlesztési lépéseket és módszereket, így szükségünk lesz néhány olyan hely rekordra, amelyek már rendelkezésre fognak állni a programok felvételekor. Ezenkívül, hogy a lekérdezést és programlistázást ki tudjuk próbálni, egy a `Programok` táblába is felveszünk egy rekordot.

Helyek felvétele

```
INSERT INTO Helyek (varos, cim, hely_neve) VALUES
   ("Szeged", "Dóm tér", "Dóm templom");

INSERT INTO Helyek (varos, cim, hely_neve) VALUES
   ("Szeged", "Dóm tér", "Dóm tér");

INSERT INTO Helyek (varos, cim, hely_neve) VALUES
   ("Szeged", "Széchenyi tér 1", "Széchenyi tér");
```

Egy program felvétele

```
INSERT INTO Programok (cim, leiras, mikortol, meddig,
   helyazonosito, artol, arig) VALUES ("A Dóm
   látogatása", "Idegenvezetés a Szegedi Dóm
   templomban.", "2019-04-21 16:00:00", "2019-04-21
   17:00:00",1, 500, 1000);
```

Figyeljük, meg, hogy a `Helyek` táblába beszúrt rekordoknál nem adtunk meg helyazonosítót, az azonban mégis létrejön, mivel a `Helyek` táblában a `helyazonosito` mező `INTEGER` típusú és `PRIMARY KEY`. A beszúrt rekordok helyazonosítói így rendre 1, 2 és 3 lesznek.

Az adatbázisunk készen van, rátérhetünk az alkalmazás fejlesztésére.

24.2. Az alkalmazás forrásfájljainak hierarchiája

A Java alkalmazások fejlesztése során nagyon fontos, hogy a forrásfájlokat milyen hierarchiába szervezzük, ugyanis ezeket az útvonalakat, mint csomagokat kell használnunk a fejlesztés során.

```
src
├── model
│   ├── HelyekModell.java
│   ├── ProgramModell.java
│   └── ProgramKalauzDB.java
├── view
│   ├── ProgramDialog.java
│   └── ProgramkalauzAblak.java
└── control
```

```
    ProgramVezerlo.java
ProgramKalauz.java
```

24.3. Adatbázis műveletek a modell-osztályokkal

Az adatbázis műveleteket ebben a példában is a modell-osztályok végzik, sem a vezérlő-, sem pedig a nézet-osztályok nem érik el közvetlenül az adatbázist. Azt a koncepciót követjük, hogy az osztályok alapértelmezett konstruktorral rendelkeznek csak, mindegyik adattagjukhoz tartozik beállító- és lekérő metódus. Az adatmanipuláció az `adatotFelvisz()`, `adatotFrissit()`, `adatotTorol()` metódusokkal zajlik. Tehát például egy beszúrás úgy zajlik, hogy példányosítunk egy modellt, beállítjuk az adattagjait, és meghívjuk az `adatotFelvisz()` metódust.

24.3.1. Az adatbázis-kapcsolat létrehozása

Az adatbázis-kapcsolatot a `ProgramKalauzDB` osztály valósítja meg. Az osztály felépítése nagyon egyszerű, mindössze egy statikus metódusa van, amely egy `java.sql.Connection` objektumot ad vissza. Felmerülhet az Olvasóban a kérdés, hogy akkor miért van szükség egy különálló osztályra? Így csak egyszer kell megadni a csatlakozási adatokat. Megjegyezzük, hogy ez az osztály átalakítható úgy is, hogy a csatlakozási adatok paraméterként legyenek megadhatók, erre azonban most nincs szükség.

ProgramKalauzDB.java

```
1  package src.model;
2  import java.sql.*;
3  import java.io.*;
4
5  class ProgramKalauzDB {
6
7    public static Connection getConnection() {
8      try {
9        String url = "jdbc:sqlite:data/programkalauz.db";
10       return DriverManager.getConnection(url);
11     } catch (SQLException ex) {
12       System.out.println("Nem sikerult csatlakozni az adatbazishoz!
             ↪ ");
13       ex.printStackTrace();
14     }
```

```
15      return null;
16    }
17  }
```

Itt jegyezzük meg, hogy ugyan a példa SQLite-hoz készült, nem túl sok SQLite-specifikus rész van a forráskódban (az automatikusan generált azonosítók lekérését kivéve), így kis módosítással más relációs adatbázisokhoz (pl. MySQL-hez) is használható.

24.3.2. A `HelyekModell` osztály

Ez az osztály az első olyan osztályunk, amely követi a fenti tervezési mintát, vagyis, hogy alapértelmezett konstruktorral rendelkezik és a paramétereket beállító metódusokon keresztül adjuk meg. Az osztály feladata, hogy kezelje a `Helyek` táblában tárolt adatokat. A tábla attribútumainak megfelelően hozzuk létre az osztály adattagjait, és az alapértelmezett konstruktorban inicializáljuk őket.

A `HelyekModell` osztály adattagja és konstruktora

```
1  package src.model;
2  import java.sql.*;
3  import java.io.*;
4  import java.util.ArrayList;
5
6  public class HelyekModell {
7
8    private int helyazonosito;
9    private String varos;
10   private String cim;
11   private String helyNeve;
12
13   public static int SQL_OK = 1;
14   public static int SQL_HIBA = 0;
15
16   public HelyekModell() {
17     this.helyazonosito = -1;
18     this.varos = null;
19     this.cim = null;
20     this.helyNeve = null;
21   }
```

Hely lekérdezése

Az adatbázis rekordok lekérdezése is ebben az osztályban van megvalósítva. Figyelembe kell vennünk, hogy az osztály egy példánya egy rekordot reprezentál, tehát annak a metódusnak amely lekér a `Helyek` táblából egy helyet, be kell állítania a példány adattagjait. Példánkban ezt az `adatotLekerdez()` metódus végzi az előre megadott helyazonosító alapján. A módszer tehát hasonló, mint a korábban említett adatfelvitelnél: példányosítás, azonosító megadása, rekord lekérdezése. Magát a lekérdezést a `PreparedStatement` objektummal hajtjuk végre, amelynek paraméterül beállítjuk helyazonosítót.

Az `adatotLekerdez()` metódus

```
1  public int adatotLekerdez() {
2    try {
3      Connection conn = ProgramKalauzDB.getConnection();
4      PreparedStatement stmt = conn.prepareStatement("SELECT * FROM
           ↪ Helyek WHERE helyazonosito = ?");
5      stmt.setInt(1, this.helyazonosito);
6      ResultSet eredmeny = stmt.executeQuery();
7      while (eredmeny.next()) {
8        this.cim = eredmeny.getString("cim");
9        this.varos = eredmeny.getString("varos");
10       this.helyNeve = eredmeny.getString("hely_neve");
11     }
12     conn.close();
13     return HelyekModell.SQL_OK;
14   } catch (SQLException ex) {
15     System.out.println("Hiba történt!");
16     ex.printStackTrace();
17     return HelyekModell.SQL_HIBA;
18   }
19 }
```

Az osztály tartalmaz két statikus konstanst: SQL_HIBA és SQL_OK. Ezek a konstansok arra szolgálnak, hogy általuk a függvény sikerességét, vagy hibás működését ellenőrizzük. Habár hibás SQL utasítás esetében létrejön egy kivétel, ezzel önmagában nem tudjuk beazonosítani a hiba helyét. Célszerű úgy tervezni a metódusokat, – ahogy erre több példát látunk nagyobb függvénykönyvtárak esetében is, – hogy a metódus rendelkezzen egy visszatérési értékkel a sikerességet illetően, amely plusz információkat is tartalmazhat.

Hely felvétele

Az adatok felvételére a `adatotFelvisz()` metódust használjuk. Mire meghívásra kerül, az osztály adattagjainak már rendelkezniük kell az adatbázisba rögzítendő értékekkel. Itt a `PreparedStatement`-tel adjuk meg a paraméterezett `INSERT` utasítást. Az egyes paramétereket a `PreparedStatement` `setString()` metódusával kötjük be az utasításba.

Az `adatotFelvisz()` metódus

```
public int adatotFelvisz() {
  try {
    Connection conn = ProgramKalauzDB.getConnection();
    PreparedStatement stmt = conn.prepareStatement("INSERT INTO
        ↪ Helyek (cim, varos, hely_neve) VALUES (?, ?, ?)");
    stmt.setString(1, this.cim);
    stmt.setString(2, this.varos);
    stmt.setString(3, this.helyNeve);
    int sikeres = stmt.executeUpdate();
    if ( sikeres == 0) {
      System.out.println("Nem sikerült beszúrni a rekordot!");
    } else {
      System.out.println(sikeres + " rekordot szúrtunk be.");
      stmt = conn.prepareStatement( "SELECT last_insert_rowid() AS
          ↪ sorszam" );
      ResultSet eredmeny = stmt.executeQuery();
      while ( eredmeny.next() ) {
        this.helyazonosito = eredmeny.getInt("sorszam");
      }
    }
    conn.close();
    return HelyekModell.SQL_OK;
  } catch (SQLException ex) {
    System.out.println("Hiba történt!");
    ex.printStackTrace();
    return HelyekModell.SQL_HIBA;
  }
}
```

Vegyük észre, hogy az `INSERT` utasításnál csak a várost, a címet és a hely nevét adjuk meg, a helyazonosítót viszont nem. Azt a beszúrás után tudjuk lekérdezni az alábbi utasítással.

Legutolsó ROWID lekérdezése

```
SELECT last_insert_rowid() AS sorszam;
```

Az így kapott értéket a helyazonosító adattaghoz rendeljük.

Hely törlése

A helyek törlése is hasonlóképpen történik az `adatotTorol()` metódusban. Az utasítás egyetlen paramétere a `WHERE` záradékban megadott helyazonosító.

Az `adatotTorol()` metódus

```
1  public int adatotTorol() {
2    try {
3      Connection conn = ProgramKalauzDB.getConnection();
4      PreparedStatement stmt = conn.prepareStatement("DELETE FROM
           ↪ Helyek WHERE helyazonosito = ?");
5      stmt.setInt(1, this.helyazonosito);
6      int sikeres = stmt.executeUpdate();
7      if ( sikeres == 0 ) {
8        System.out.println("Nem sikerült törölni a rekordot!");
9      } else {
10       System.out.println(sikeres + " rekordot töröltünk.");
11     }
12     conn.close();
13     return HelyekModell.SQL_OK;
14   } catch (SQLException ex) {
15     System.out.println("Hiba történt!");
16     ex.printStackTrace();
17     return HelyekModell.SQL_HIBA;
18   }
19 }
```

Helyek módosítása

Ez a művelet is hasonlít az adatfelviteli metódushoz. Fontos megjegyezni, hogy a módosításnál minden attribútumot felülírunk és az `adatotFrissit()` metódus meghívása előtt a példány adattagjaiban már az új értékek szerepelnek.

Az `adatotFrissit()` metódus

```
1 public int adatotFrissit() {
2   try {
3     Connection conn = ProgramKalauzDB.getConnection();
4     PreparedStatement stmt = conn.prepareStatement("Update Helyek
          ↪ SET cim= ?, varos = ?, hely_neve = ? WHERE helyazonosito
          ↪ = ?");
```

```
5     stmt.setString(1, this.cim);
6     stmt.setString(2, this.varos);
7     stmt.setString(3, this.helyNeve);
8     stmt.setInt(4, this.helyazonosito);
9     int sikeres = stmt.executeUpdate();
10    if ( sikeres == 0) {
11      System.out.println("Nem sikerült módositani a rekordot!");
12    } else {
13      System.out.println(sikeres + " rekordot módosítottunk.");
14    }
15    conn.close();
16    return HelyekModell.SQL_OK;
17  } catch (SQLException ex) {
18    System.out.println("Hiba történt!");
19    ex.printStackTrace();
20    return HelyekModell.SQL_HIBA;
21  }
22}
```

Minden hely lekérdezése

A programunkban a helyeket egy lenyíló listában szeretnénk felsorolni, emiatt szükséges készítenünk egy olyan metódust, amely a `Helyek` táblában tárolt rekordokat valamiféle listában adja vissza. Kihasználjuk, hogy a Java-ban lévő `ArrayList` osztálynak paraméterként meg lehet adni tárolandó objektumok típusát. Ennélfogva a legegyszerűbb, ha `HelyekModell` típusú objektumokat tárolunk, és egy ilyen listát ad vissza a metódusunk. Az osztályban szereplő `mindenHelyetListaz()` metódus statikus metódus, hiszen ennek nem kell egy objektumhoz kötődnie.

A `mindenHelyetListaz()` metódus

```
1  public static ArrayList<HelyekModell> mindenHelyetListaz() {
2    try {
3      ArrayList<HelyekModell> lista = new ArrayList<HelyekModell>();
4      Connection conn = ProgramKalauzDB.getConnection();
5      PreparedStatement stmt = conn.prepareStatement("SELECT * FROM
           ↪ Helyek");
6      ResultSet eredmeny = stmt.executeQuery();
7      while (eredmeny.next()) {
8        int azonosito = eredmeny.getInt("helyazonosito");
9        String cim = eredmeny.getString("cim");
```

```
10        String varos = eredmeny.getString("varos");
11        String helyNeve = eredmeny.getString("hely_neve");
12
13        HelyekModell hm = new HelyekModell(); // új objektum
14        hm.helyazonositotMegad(azonosito);
15        hm.cimetMegad(cim);
16        hm.helyNevetMegad(helyNeve);
17        hm.varostMegad(varos);
18
19        lista.add(hm); // hozzáadjuk az új objektumot a listához
20      }
21      conn.close();
22      return lista;
23    } catch (SQLException ex) {
24      System.out.println("Hiba történt!");
25      ex.printStackTrace();
26      return null;
27    }
28  }
```

A `HelyekModell` objektum String reprezentációja

Már említettük, hogy a programunkban a helyeket listákban szeretnénk megjeleníteni, viszont ezek a listák `HelyekModell` objektumokat tartalmaznak. Fontos lenne emiatt, ha ezeknek az objektumoknak lenne String reprezentációja is. Ehhez felül kell definiálnunk az osztály `toString()` metódusát[1].

A felüldefiniált `toString()` metódus

```
1  public String toString() {
2    return this.helyNeve + ". " + this.varos + ", " + this.cim;
3  }
```

24.3.3. A `ProgramModell` osztály

A `ProgramModell` osztályunkban lesz megvalósítva a `Programok` tábla adatainak kezelése. Továbbra is azt az elvet követjük, hogy adatfelvitelnél és módosításnál egy `ProgramModell` példány adattagjait előre beállítjuk és az

[1] Java-ban minden osztály az `Object` osztályból származik, amely rendelkezik egy `toString()` metódussal.

adat felvitelét itt is az `adatotFelvisz()`, módosítását az `adatotFrissit()`, törlését pedig az `adatotTorol()` metódus végzi majd el, amely tartalma hasonlít az előző alfejezetben megismert módhoz, emiatt itt ezt nem részletezzük. A `ProgramModell` osztálynak is van egy olyan statikus metódusa, amely kilistázza az összes programot. Ez, a `mindenProgramotListazIdoSzerint()` nevű metódus is hasonlít a `HelyekModell` osztályban bemutatott metódushoz.

24.3.4. A `MufajModell` osztály

Adatbázisunkban a `Mufaj` táblának mindössze két attribútuma van: a programazonosító, amely külső kulcs és a `Programok` tábla elsődleges kulcsára vonatkozik, valamint a `mufajmegnevezes` amely egy tetszőlegesen beállítható sztring.

Adat felvitele, módosítása és törlése

Az `adatotFelvisz()` és `adatotTorol()` metódusok hasonlóan működnek az előzőekben látott példákhoz. Az `adatotFrissit()` metódus, amely a bejegyzések módosítását végzi. Egy programhoz több műfajt is megadhatunk. Ebben a példában feltételezzük, hogy legfeljebb három műfaji megnevezés tartozik majd egy programhoz, de lehetne több is. Emiatt az `adatotFrissit()` metódus három `String` paramétert vár. Megjegyezzük, hogy ha általánosabban szeretnénk megvalósítani ezt a metódust, és tetszőleges számú műfaji megnevezést is megengedünk, akkor a három `String` paraméter helyett egy `String[]` tömböt kellene paraméterül fogadnia a metódusnak. A módosítást illetően nem tudjuk pontosan, hogy melyik rekordot hogyan módosítjuk, annyit tudunk mindössze, hogy egy programhoz tartozó műfaji megnevezések megváltoztak. Emiatt célszerű úgy eljárni, hogy töröljük az adott programhoz tartozó összes műfaji megnevezést, és a paramétereknek megfelelően beállítjuk az újakat. A metóduson belül ellenőrizzük, hogy az egyes paramétereknek van-e valójában értéke, és ha igen, akkor beszúrunk egy új rekordot.

Az `adatotFrissit()` metódus

```
1  public int adatotFrissit(String _mufaj1, String _mufaj2, String
       ↪ _mufaj3) {
2    // először törlünk minden erre a programra vonatkozó műfajt,
         ↪ majd új rekordokat szúrunk be
3    int sikeres = this.adatotTorol();
4    if ( _mufaj1.length() > 0 ) {
```

```
5       this.megnevezes = _mufaj1;
6       sikeres += this.adatotFelvisz();
7     }
8     if ( _mufaj2.length() > 0 ) {
9       this.megnevezes = _mufaj2;
10      sikeres += this.adatotFelvisz();
11    }
12    if ( _mufaj3.length() > 0 ) {
13      this.megnevezes = _mufaj3;
14      sikeres += this.adatotFelvisz();
15    }
16    if ( sikeres == 0 ) {
17      return MufajModell.SQL_HIBA;
18    }
19    return MufajModell.SQL_OK;
20  }
```

Lekérdezések

A `MufajModell` osztályban kétféle lekérdezést valósítunk meg. Egyrészről szeretnénk olyan listákat kapni, amely tartalmazza, hogy egy adott műfajban milyen programok szerepelnek. Másrészt pedig olyan listákra is szükségünk lesz, amely azt tartalmazza, hogy egy programnak milyen műfaji megnevezései vannak.

Az első lekérdezést a `mindenProgramAMufajban()` metódus valósítja meg, amely paraméterül vár egy sztringet (ami a műfaji megnevezés) és visszatér egy `ProgramModell`-eket tartalmazó `ArrayList` objektummal. A lekérdezésben össze kell kötnünk a `Programok` és a `Mufaj` táblát. Ez a metódus is statikus, mivel nem egy objektumhoz kötődik.

A `mindenProgramAMufajban` metódus

```
1   public static ArrayList<ProgramModell> mindenProgramAMufajban(
        ↪ String _mufaj) {
2     try {
3       ArrayList<ProgramModell> lista = new ArrayList<ProgramModell
            ↪ >();
4       Connection conn = ProgramKalauzDB.getConnection();
5       PreparedStatement stmt = conn.prepareStatement("SELECT
            ↪ programazonosito, cim,"
6       + "leiras, mikortol, meddig, helyazonosito, artol, arig"
7       + "FROM Programok, Mufaj WHERE Programok.programazonosito =
```

```
        ↪ Mufaj.mufajazonosito AND Mufajok.mufajmegnevezes = ?");
    stmt.setString(1, _mufaj);
    ResultSet eredmeny = stmt.executeQuery();
    while (eredmeny.next()) {
      int azonosito = eredmeny.getInt("programazonosito");
      String cim = eredmeny.getString("cim");
      String leiras = eredmeny.getString("leiras");
      String mikortol = eredmeny.getString("mikortol");
      String meddig = eredmeny.getString("meddig");
      int helyazonosito = eredmeny.getInt("helyazonosito");
      int artol = eredmeny.getInt("artol");
      int arig = eredmeny.getInt("arig");

      ProgramModell pm = new ProgramModell(); // új objektum
      pm.azonositotMegad(azonosito);
      pm.cimetMegad(cim);
      pm.leirastMegad(leiras);
      pm.mikortoltMegad(mikortol);
      pm.meddigetMegad(meddig);
      pm.helyazonositotMegad(helyazonosito);
      pm.artoltMegad(artol);
      pm.arigotMegad(arig);

      lista.add(pm); // hozzáadjuk az új objektumot a listához
    }
    conn.close();
    return lista;
  } catch (SQLException ex) {
    System.out.println("Hiba történt!");
    ex.printStackTrace();
  }
  return null;
}
```

A második lekérdezést az `egyProgamMufajai()` valósítja meg, amely egy programazonosítót vár paraméterül és eredményül egy `String`-eket tartalmazó `ArrayList` objektumot ad vissza.

Az `egyProgramMufajai` metódus

```
public static ArrayList<String> egyProgramMufajai(int
    ↪ _programazonosito) {
  try {
    ArrayList<String> lista = new ArrayList<String>();
    Connection conn = ProgramKalauzDB.getConnection();
```

```
5     PreparedStatement stmt = conn.prepareStatement("SELECT
          ↪ mufajmegnevezes"
6     +" FROM Programok, Mufaj WHERE Programok.programazonosito =
          ↪ Mufaj.programazonosito AND Mufaj.programazonosito = ?
          ↪ AND mufajmegnevezes != ''");
7     stmt.setInt(1, _programazonosito);
8     ResultSet eredmeny = stmt.executeQuery();
9     while (eredmeny.next()) {
10      String megnevezes = eredmeny.getString("mufajmegnevezes");
11      // hozzáadjuk az új sztringet a listához
12      lista.add(megnevezes);
13    }
14    conn.close();
15    return lista;
16  } catch (SQLException ex) {
17    System.out.println("Hiba történt!");
18    ex.printStackTrace();
19  }
20  return null;
21 }
```

24.4. A vezérlő osztályok

A vezérlő-osztályok teremtenek kapcsolatot a grafikus felhasználói felület valamint a modell-osztályok között. Tulajdonképpen közvetítő szerepet játszanak az adattovábbításban, de fontos funkciójuk, hogy ebben a rétegben kell megvalósítani az adatellenőrzést illetve az adat átalakítását (ha szükséges).

A példában most csak a programok felvitelére, módosítására, törlésére és lekérdezésére fókuszálunk, ezért az itt bemutatott osztályunk a `ProgramVezerlo` lesz, de a helyek felviteléhez is hasonló vezérlő osztályt kell létrehozni. A műfajokat illetően nem szükséges külön vezérlőt létrehozni, mivel azok az adatok erőteljesen a programokhoz kötődnek. Tulajdonképpen egy többértékű attribútum leképezése miatt jön külön sémában létre. Ennélfogva amikor a programok adatait kezeljük, hozzávesszük majd a műfajokat is, így a `ProgramVezerlo` osztályban kell a programok adatait szétbontani a `Programok` és `Mufajok` táblák adataira.

A `ProgramVezerlo` osztály statikus metódusokkal rendelkezik, hiszen ahogy korábban említettük, elsődleges célja az adattovábbítás és -ellenőrzés, nem kell belőle objektumot létrehozni.

Az adatok beszúrásához a `programFelvitele()` metódust kell majd meg-

hívnunk, amely paraméterül várja az összes olyan adatot – beleértve a műfaji adatokat is – amely egy programhoz kapcsolódik. Nagyon fontos, hogy ezeket az adatokat a modell-osztálynak (nevezetesen a `ProgramokModell` osztálynak) kell majd átadni és az ő `adatotFelvisz()` metódusa fogja az adatbázisba beszúrni az új rekordot. Az adatok átalakítása és ellenőrzése kapcsán az alábbi feladatokat kell végrehajtani:

- a sztring-adatokat változatlanul hagyjuk;
- a dátum adatokat ellenőrizzük, a `SimpleDateFormat` osztály segítségével;
- a szám adatokat a grafikus felületről sztring formátumban kapjuk meg, viszont az adatbázisban már számként kell kezelni, így azokat átalakítjuk;
- az ár kapcsán van egy alsó és egy felső határa az ártartománynak, amelynél ellenőrizzük, hogy az alsó határa kisebb legyen, mint a felső határa.

A `programFelvitele()` metódus

```
1  public static int programFelvitele(String _cim, String _leiras,
       ↪ String _mikortol, String _meddig, String _web, String
       ↪ _kapcsolat, int _helyazonosito, String str_artol, String
       ↪ str_arig, String _mufaj1, String _mufaj2, String _mufaj3) {
2    String jo_meddig;
3    String jo_mikortol;
4    // dátum ellenőrzése
5    try {
6      SimpleDateFormat datumFormatum = new SimpleDateFormat("yyyy-mm
         ↪ -dd hh:mm:ss");
7      Date mikortolDatum = datumFormatum.parse(_mikortol);
8      Date meddigDatum = datumFormatum.parse(_meddig);
9      jo_mikortol = datumFormatum.format(mikortolDatum);
10     jo_meddig = datumFormatum.format(meddigDatum);
11   } catch (ParseException ex) {
12     ex.printStackTrace();
13     return 0;
14   }
15
16   ProgramModell program = new ProgramModell();
17   program.cimetMegad(_cim);
18   program.leirastMegad(_leiras);
19   program.mikortoltMegad(_mikortol);
20   program.meddigetMegad(_meddig);
```

```
21    program.webetMegad(_web);
22    program.kapcsolatotMegad(_kapcsolat);
23    program.helyazonositotMegad(_helyazonosito);
24    int artol = 0;
25    if ( str_artol.equals("") == false ) {
26      artol = Integer.parseInt(str_artol);
27    }
28    int arig = 0;
29    if ( str_arig.equals("") == false ) {
30      arig = Integer.parseInt(str_arig);
31    }
32    if ( artol > arig ) {
33      int seged = artol;
34      artol = arig;
35      arig = seged;
36    }
37    program.artoltMegad(artol);
38    program.arigotMegad(arig);
39    int sikeres = 1;
40    sikeres &= program.adatotFelvisz();
41    MufajModell mufaj = new MufajModell();
42    mufaj.programAzonositotMegad(program.azonosito());
43
44    if ( _mufaj1.length() > 0) {
45      mufaj.megnevezestMegad(_mufaj1);
46      sikeres &= mufaj.adatotFelvisz();
47    }
48    if ( _mufaj2.length() > 0 ) {
49      mufaj.megnevezestMegad(_mufaj2);
50      sikeres &= mufaj.adatotFelvisz();
51    }
52    if ( _mufaj3.length() > 0) {
53      mufaj.megnevezestMegad(_mufaj3);
54      sikeres &= mufaj.adatotFelvisz();
55    }
56
57    return sikeres;
58  }
```

Láthatjuk, hogy külön kezeljük a `Programok` tábla adatait és külön a `Mufajok` tábla adatait. Ráadásul a sorrend sem mindegy! A `Programok` táblába történő beszúráskor (az `adatotFelvisz()` metódusban) kérdezzük le ugyanis az újonnan beszúrt rekord automatikusan generált programazonosítóját, amit fel kell használnunk a `Mufaj` táblában. Azt, hogy sikerült-e beszúrni a rekordot, vagy kaptunk-e valami hibát, mindkét beszúrás esetében

ellenőriznünk kell. Ha minden rendben ment, akkor minden beszúrás után 1-es értékkel tér vissza az `adatotFelvisz()` metódus. Emiatt a `sikeres` változó értékét 1-re inicializáljuk, és bitenkénti `ÉS` művelettel képezzük az új értékét minden beszúrás után. A műfajok esetében azért kell leellenőriznünk a paraméterben megadott sztring hosszát, mert nem garantált, hogy mindhárom érték meg van adva és nem szeretnénk, hogy az adatbázisba üres sztringek is bekerüljenek.

A programok módosítása és törlése, amelyeket rendre a `programModositasa()` és `programTorlese()` metódusok valósítanak meg, hasonlóképpen épülnek fel.

A `programModositasa()` metódus

```
1  public static int programModositasa(int _programazonosito, String
       ↪ _cim, String _leiras, String _mikortol, String _meddig,
       ↪ String _web, String _kapcsolat, int _helyazonosito, String
       ↪ str_artol, String str_arig, String str_mufaj1, String
       ↪ str_mufaj2, String str_mufaj3) {
2    String jo_meddig;
3    String jo_mikortol;
4    // dátum ellenőrzése
5    try {
6      SimpleDateFormat datumFormatum = new SimpleDateFormat("yyyy-mm
         ↪ -dd hh:mm:ss");
7      Date mikortolDatum = datumFormatum.parse(_mikortol);
8      Date meddigDatum = datumFormatum.parse(_meddig);
9      jo_mikortol = datumFormatum.format(mikortolDatum);
10     jo_meddig = datumFormatum.format(meddigDatum);
11   } catch (ParseException ex) {
12     ex.printStackTrace();
13     return 0;
14   }
15
16   ProgramModell program = new ProgramModell();
17   program.azonositotMegad(_programazonosito);
18   program.cimetMegad(_cim);
19   program.leirastMegad(_leiras);
20   program.mikortoltMegad(_mikortol);
21   program.meddigetMegad(_meddig);
22   program.webetMegad(_web);
23   program.kapcsolatotMegad(_kapcsolat);
24   program.helyazonositotMegad(_helyazonosito);
25   int artol = 0;
26   if ( str_artol.equals("") == false ) {
27     artol = Integer.parseInt(str_artol);
28   }
```

```
29    int arig = 0;
30    if ( str_arig.equals("") == false ) {
31      arig = Integer.parseInt(str_arig);
32    }
33    if ( artol > arig ) {
34      int seged = artol;
35      artol = arig;
36      arig = seged;
37    }
38    program.artoltMegad(artol);
39    program.arigotMegad(arig);
40    MufajModell mufaj = new MufajModell();
41    mufaj.programAzonositotMegad(_programazonosito);
42
43    return program.adatotFrissit() & mufaj.adatotFrissit(str_mufaj1,
          ↪  str_mufaj2, str_mufaj3);
44  }
```

A `programTorlese()` metódus

```
1   public static int programTorlese(int _programazonosito) {
2     ProgramModell program = new ProgramModell();
3     program.azonositotMegad(_programazonosito);
4     MufajModell mufaj = new MufajModell();
5     mufaj.programAzonositotMegad(_programazonosito);
6     return program.adatotTorol() & mufaj.adatotTorol();
7   }
```

24.5. A grafikus felhasználói felület elkészítése

A modell-és vezérlő-osztályok bemutatása után a grafikus felület elkészítésével folytatjuk a fejlesztést. Ebben a fejezetben két grafikus panel elkészítését mutatjuk be, a főablakét és a programok adatainak kezelésére szolgáló párbeszédablakét. A többi hasonlóképpen képpen készíthető el. A párbeszédablakban fogjuk tudni felvenni az újabb rekordokat, módosítani és törölni a már meglévőket. Alkamazásunk kinézetét a 24.1 ábra szemlélteti.

24.5.1. A főablak elkészítése

A főablak a `ProgramkalauzAblak` osztályban kerül megvalósításra. Az osztály a `JFrame` Java osztályból származik, ezáltal rendelkezik kerettel, címsorral, menüsorral.

Ahogy azt a 24.1. ábrán láthatjuk, a főablakban csak a menü és a programok listája látszik. A menüsoron három menü található: **File**, **Programok**, **Helyek**. A **File** menüben csak egy kilépés menüpont (`JMenuItem`) található, a **Programok** menüben egy-egy menüpont van a programok felvitelét, módosítását és törlését lehetővé tevő párbeszédablak megjelenítésére. A **Helyek** menü hasonlóan épül fel.

A programok listáját egy görgethető panelre (`JScrollPane`) tesszük. Minden program egy külön panelen (`JPanel`) van, amelyre három cimkét (`JLabel`) teszünk egymás alá függőleges elrendezésben.

Az osztály konstruktorában állítjuk be az ablak méreteit, megjelenését, és példányosítjuk az egyes elemeket.

A `ProgramkalauzAblak` osztály konstruktora

```
1  public ProgramkalauzAblak(String _ablakcim) {
2    // ablak tulajdonsagainak beallitasa
3    super(_ablakcim);
4    this.setSize(400,400);
5    this.setLocation(40,40);
6    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7
8
9    // ablak elemeinek letrehozasa
10   this.fopanel = new JPanel();
11   this.programpanel = new JPanel();
12   this.gorgetoPanel = new JScrollPane(programpanel);
13   this.gorgetoPanel.setPreferredSize(new Dimension(400,400));
14   this.menusor = new JMenuBar();
15   this.fileMenu = new JMenu("File");
16   this.programokMenu = new JMenu("Programok");
17   this.helyekMenu = new JMenu("Helyek");
18   this.ujProgramMenupont = new JMenuItem("Új program felvitele ...
         ↪ ");
19   this.programModositasaMenupont = new JMenuItem("Program módosítá
         ↪ sa ...");
20   this.programTorleseMenupont = new JMenuItem("Program törlése ...
         ↪ ");
21   this.ujHelyMenupont = new JMenuItem("Új hely felvitele ...");
22   this.helyModositasaMenupont = new JMenuItem("Hely módosítása ...
         ↪ ");
```

```
23    this.helyTorleseMenupont = new JMenuItem("Hely törlése ...");
24    this.kilepesMenupont = new JMenuItem("Kilépés");
25
26    this.programDialog = new ProgramDialog(this);
27
28    // esemenykezeles beallitasa
29    this.esemenykezelestBeallit();
30
31    // ablak felepitese
32    this.ablakFelepitese();
33
34    // kinézet frissitese
35    this.repaint();
36    this.setVisible(true);
37  }
```

Elrendezés

A könnyebb áttekinthetőség kedvéért magát az elrendezést nem a konstruktorban, hanem az `ablakFelepitese()` metódusban valósítjuk meg.

A főablak tartalom konténerére (`contentPane`) tesszük rá a főpanelt, arra helyezzük a görgetőpanelt, amely már a programokat tartalmazó panelt `programpanel` is tartalmazza. Talán túlságosan összetettnek tűnik, de minden egyes panelhez más-más elrendezést tudunk hozzárendelni, amely segít a grafikus felület igényes kialakításában. A programokat tartalmazó panelt gyakran fogjuk újragenerálni, emiatt ezt a `programokatListaz()` metódusban valósítjuk meg. Megjegyezzük, hogy maguk a panelek nem láthatóak, hacsak színben nem térnek el egymástól. Méretük mindig igazodik a tartalmukhoz. A menüsorhoz előbb a menüket, majd a menükhöz a menüelemeket tesszük hozzá.

Az `ablakFelepitese()` metódus

```
1   private void ablakFelepitese() {
2     this.getContentPane().add(fopanel);
3
4     // menu kialakitasa
5     this.fileMenu.add(kilepesMenupont);
6     this.programokMenu.add(this.ujProgramMenupont);
7     this.programokMenu.add(this.programModositasaMenupont);
8     this.programokMenu.add(this.programTorleseMenupont);
9     this.helyekMenu.add(this.ujHelyMenupont);
```

```
10    this.helyekMenu.add(this.helyModositasaMenupont);
11    this.helyekMenu.add(this.helyTorleseMenupont);
12
13    this.menusor.add(fileMenu);
14    this.menusor.add(programokMenu);
15    this.menusor.add(helyekMenu);
16    this.setJMenuBar(this.menusor);
17
18    this.programpanel.setLayout(new BoxLayout(programpanel,
          ↪ BoxLayout.PAGE_AXIS));
19    this.programokatListaz();
20    this.fopanel.add(gorgetoPanel);
21  }
```

Eseménykezelés

A menüpontokra kattintás `ActionEvent` eseményt hoz létre, amelyeket egy `ActionListener` eseményfigyelővel figyelünk. Minden olyan grafikus elemhez, amelynél a kattintást, gombnyomást szeretnénk lekezelni, hozzá kell rendelni egy `ActionListener`-t az `addActionListener()` metódusával, melynek paramétere fogja megmondani, hogy melyik objektumpéldány kezeli majd le az eseményt. Esetünkben maga a főablak lesz, amely lekezeli az eseményt (ezt jelöljük a paraméterként megadott `this` szóval), és ez az osztály fogja megvalósítani az `ActionListener` interfészt, vagyis tartalmazza és definiálja annak `actionPerformed()` metódusát.

Eseményfigyelő hozzárendelése a grafikus elemekhez

```
1   private void esemenykezelestBeallit() {
2     this.ujProgramMenupont.addActionListener(this);
3     this.programModositasaMenupont.addActionListener(this);
4     this.programTorleseMenupont.addActionListener(this);
5     this.ujHelyMenupont.addActionListener(this);
6     this.helyModositasaMenupont.addActionListener(this);
7     this.helyTorleseMenupont.addActionListener(this);
8     this.kilepesMenupont.addActionListener(this);
9   }
```

Az `actionPerformed()` metódusban meg kell vizsgálni, hogy mely objektum váltotta ki az eseményt (az `ActionEvent getSource()` metódusával), ez alapján feltételes vezérlési szerkezetben kell lekezelni azt. Például, a **Kilép** menüponttal be tudjuk zárni a programot, a **Programok** menü három

menüpontja pedig megjelenítni a párbeszédablakot (erről még később lesz szó).

Események kezelése

```
1  public void actionPerformed(ActionEvent esemeny) {
2    if ( esemeny.getSource().equals(this.kilepesMenupont) ) {
3      this.dispose();
4      System.exit(0);
5    }
6    if ( esemeny.getSource().equals(this.ujProgramMenupont) ) {
7      programDialog.modBeallitas(ProgramDialog.BESZURAS_MOD);
8      programDialog.show();
9    }
10   if ( esemeny.getSource().equals(this.programModositasaMenupont) )
       ↪ {
11     programDialog.modBeallitas(ProgramDialog.MODOSITAS_MOD);
12     programDialog.show();
13   }
14   if ( esemeny.getSource().equals(this.programTorleseMenupont) ) {
15     programDialog.modBeallitas(ProgramDialog.TORLES_MOD);
16     programDialog.show();
17   }
18 }
```

24.5.2. Párbeszédablak készítése a programok adatainak kezelésére

A párbeszédablakot a `ProgramDialog` osztályban valósítjuk meg. Ez az ablak teszi majd lehetővé, hogy az adatbázisban lévő programokat kezelni tudjuk, vagyis a beszúrás, módosítás és törlés adatait itt adjuk meg, majd a `ProgramVezerlo` osztaly fogja továbbítani az adatokat a modellosztályoknak. A `ProgramDialog` osztály ezért háromféle móddal rendelkezik, amelyeket statikus konstansként jelölünk az osztályon belül, amelyeket a `modBeallitas()` metóduson keresztül lehet beállítani. A szerepe elsősorban az, hogy az egyes módokhoz csak bizonyos grafikus elemeket (beviteli mezőket, lenyíló listákat és gombokat) engedjen használni, a többit tiltsa le, hogy a felhasználó véletlenül se tudjon rákattintani.

Módok beállítása

```
1  ...
2  public static int BESZURAS_MOD = 1;}
```

```
3  public static int MODOSITAS_MOD = 2;}
4  public static int TORLES_MOD = 3;}
5
6  ...
7  public void modBeallitas(int _mod) {
8    this.inicializal();
9    if ( _mod == ProgramDialog.BESZURAS_MOD ) {
10     this.setTitle("Új program felvétele");
11     this.mezoketEnged(true);
12     this.programLista.setEnabled(false);
13     this.felviszGomb.setEnabled(true);
14     this.torolGomb.setEnabled(false);
15     this.modositGomb.setEnabled(false);
16   } else if ( _mod == ProgramDialog.MODOSITAS_MOD ) {
17     this.setTitle("Program módosítása");
18     this.mezoketEnged(false);
19     this.programLista.setEnabled(true);
20     this.felviszGomb.setEnabled(false);
21     this.torolGomb.setEnabled(false);
22     this.modositGomb.setEnabled(true);
23   } else if ( _mod == ProgramDialog.TORLES_MOD ) {
24     this.setTitle("Program törlése");
25     this.mezoketEnged(false);
26     this.programLista.setEnabled(true);
27     this.felviszGomb.setEnabled(false);
28     this.torolGomb.setEnabled(true);
29     this.modositGomb.setEnabled(false);
30   }
31 }
```

Az adatbázisban szereplő program rekordokat egy lenyíló listába gyűjtjük ki, a módosításhoz és törléshez innen fogjuk kiválasztani a rekordokat, amelyek adataival kitöltjük a beviteli mezőket. A lenyíló listához egy `ActionListener`-t rendelünk, és a lenyíló lista által kiváltott `ActionEvent`-et a `ProgramDialog`-ban lévő `actionPerformed()` metódusban lesz kezelve. A kiválasztott program adatai bekerülnek a beviteli mezőkbe, illetve a helyszín lenyíló listájába, ezt követően lehet módosítani vagy törölni a rekordot. Megfigyelhető, hogy hogyan használjuk a `ProgramVezerlo` statikus metódusait az adatfelvitelhez, módosításhoz és törléshez, valamint az is, hogy hogyan használjuk ki azt, hogy a lenyíló listáink `HelyModell` és `ProgramModell` példányokat tartalmaznak, amelyek magukban foglalják adataikait.

Eseménykezelés

```
public void actionPerformed(ActionEvent esemeny) {
  if ( esemeny.getSource().equals(this.felviszGomb) ) {
    System.out.println("Adat felvitele");
    HelyekModell hm = (HelyekModell) (this.helyekLista.
        ↪ getSelectedItem());

    int sikeres = ProgramVezerlo.programFelvitele(
    this.cimMezo.getText(),
    this.leirasMezo.getText(),
    this.mikortolMezo.getText(),
    this.meddigMezo.getText(),
    this.webMezo.getText(),
    this.kapcsolatMezo.getText(),
    hm.helyazonosito(),
    this.artolMezo.getText(),
    this.arigMezo.getText(),
    this.mufaj1Mezo.getText(),
    this.mufaj2Mezo.getText(),
    this.mufaj3Mezo.getText());
    if (sikeres == HelyekModell.SQL_HIBA) {
      System.out.println("Hiba történt az adatfelvitelnél");
    } else {
      ((ProgramkalauzAblak) (this.getOwner())).programokatListaz();
    }
    this.setVisible(false);
  }
  if ( esemeny.getSource().equals(this.programLista) ) {
    ProgramModell pm = (ProgramModell) (this.programLista.
        ↪ getSelectedItem());
    this.mezoketEnged(true);
    this.inicializal();

    this.cimMezo.setText(pm.cim());
    this.leirasMezo.setText(pm.leiras());
    this.mikortolMezo.setText(pm.mikortol());
    this.meddigMezo.setText(pm.meddig());
    this.webMezo.setText(pm.web());
    this.kapcsolatMezo.setText(pm.kapcsolat());
    // helyazonositot be kell állítani a lenyíló listában
    for ( int i = 0; i < this.helyekLista.getItemCount(); i++ ) {
      HelyekModell hm = this.helyekLista.getItemAt(i);
      if ( pm.helyazonosito() == hm.helyazonosito() ) {
        helyekLista.setSelectedIndex(i);
        break;
      }
```

```
44      }
45      this.artolMezo.setText(pm.artol()+""); // int --> String
46      this.arigMezo.setText(pm.arig()+""); // int --> String
47
48      ArrayList<String> mufajok = MufajModell.egyProgramMufajai(pm.
            ↪ azonosito());
49      if ( mufajok.size() >=1 ) {
50        this.mufaj1Mezo.setText(mufajok.get(0));
51      }
52      if ( mufajok.size() >= 2 ) {
53        this.mufaj2Mezo.setText(mufajok.get(1));
54      }
55      if ( mufajok.size() >= 3 ) {
56        this.mufaj3Mezo.setText(mufajok.get(2));
57      }
58    }
59    if ( esemeny.getSource().equals(this.modositGomb) ) {
60      System.out.println("Adat módosítása");
61      // HelyekModell hm = (HelyekModell) (this.helyekLista.
            ↪ getSelectedItem());
62      ProgramModell pm = (ProgramModell) (this.programLista.
            ↪ getSelectedItem());
63      int sikeres = ProgramVezerlo.programModositasa(
64      pm.azonosito(),
65      this.cimMezo.getText(),
66      this.leirasMezo.getText(),
67      this.mikortolMezo.getText(),
68      this.meddigMezo.getText(),
69      this.webMezo.getText(),
70      this.kapcsolatMezo.getText(),
71      pm.helyazonosito(),
72      this.artolMezo.getText(),
73      this.arigMezo.getText(),
74      this.mufaj1Mezo.getText(),
75      this.mufaj2Mezo.getText(),
76      this.mufaj3Mezo.getText());
77      if (sikeres == ProgramModell.SQL_HIBA) {
78        System.out.println("Hiba történt az adatfelvitelnél");
79      } else {
80        ((ProgramkalauzAblak) (this.getOwner())).programokatListaz();
81      }
82      this.setVisible(false);
83    }
84    if ( esemeny.getSource().equals(this.torolGomb) ) {
85      System.out.println("Adat törlése");
86      ProgramModell pm = (ProgramModell) (this.programLista.
```

```
        ↪ getSelectedItem());
87      int sikeres = ProgramVezerlo.programTorlese(pm.azonosito());
88      if ( sikeres == ProgramModell.SQL_HIBA ) {
89        System.out.println("Hiba történt az adatfelvitelnél");
90      } else {
91        ((ProgramkalauzAblak) (this.getOwner())).programokatListaz();
92      }
93      this.setVisible(false);
94    }
95  }
```

24.6. A főprogram

A főprogram felépítése viszonylag egyszerű, hiszen nincs más dolgunk, minthogy példányosítsuk a `ProgramkalauzAblak` osztályt. A főprogram a csomagokon kívül helyezkedik el.

ProgramKalauz.java

```
1   package src;
2   import src.view.*;
3   import java.io.*;
4
5   public class ProgramKalauz {
6     public static void main(String[] args) {
7       System.out.println("Program indul ...");
8       ProgramkalauzAblak ablak = new ProgramkalauzAblak("
            ↪ Programkalauz");
9     }
10  }
```

Fordítás után már használhatjuk is az alkalmazásunkat!

24.7. Fordítás és futtatás

A fordítás és futtatás során a CLASSPATH környezeti változóban meg kell adnunk az `sqlite-jdbc.jar` fájlt, valamint figyelembe kell venni, hogy a programot különböző csomagokban készítettük el (`model`, `view`, `controller`), ami azért fontos, mert a főprogram a csomagokon kívül helyezkedik el. A példák Linux-on operációs rendszeren készültek, ezért az elérési útvonalak is

ennek megfelelőek. Megjegyezzük, hogy Windows-on, csupán a fájlok elérési útvonalaiban van különbség.

Parancssorból az alábbi paranccsal tudjuk fordítani a programunkat:

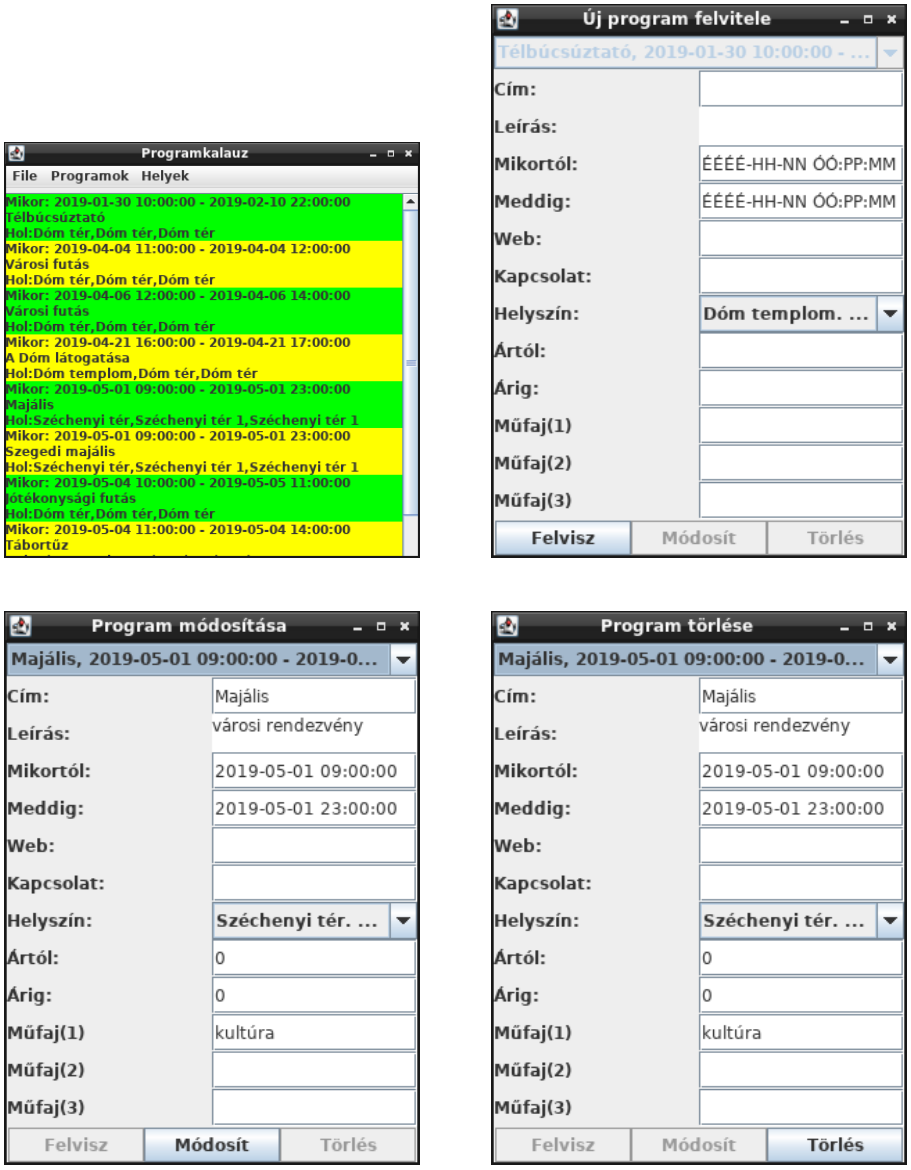
```
javac -cp .:/usr/lib/java/sqlite-jdbc.jar:./src/:./src
   ↪ /view/:./src/controller/ src/ProgramKalauz.java
   ↪ src/view/ProgramkalauzAblak.java src/model/*.
   ↪ java src/view/*.java src/controller/*.java
```

Az alábbi paranccsal tudjuk futtatni:

```
java -cp .:/usr/lib/java/sqlite-jdbc.jar:./src/:./src/
   ↪ view/:./src/controller/ src.ProgramKalauz
```

Kérdések és feladatok

1. Valósítsa meg a programban a beszúrást, módosítást és a törlést minden táblához!
2. Egészítse ki a programot azzal a funkcióval, hogy ne mutassa a már elmúlt programokat!
3. Mi a JDBC?
4. Hogyan lehet csatlakozni a programmal MySQL adatbázishoz?
5. Használható-e az itt bemutatott adatbázist létrehozó utasítássorozat MySQL-ben közvetlenül? Indokolja válaszát?



24.1. ábra. A Programkalauz alkalmazás főablaka és a programokat kezelő párbeszédablak adatfelvitel, módosítás és törlés opcióban.

Tárgyleírás

A következő oldalakon a *Tanulási Eredmény Alapú Módszertan* szempontrendszere alapján adjuk meg azon kurzus adatait, melyhez a jegyzet készült.

Tantárgyleíráshoz alkalmazható sablon

–a MAB hivatalos űrlapja alapján[1]–

ALAPSZAK

(1.) Tantárgy neve: **Adatbázisok**	**Kreditértéke: E.a.:2 + Gyak.:2**
A tantárgy **besorolása**: **kötelező**	
A tantárgy elméleti vagy gyakorlati jellegének mértéke, „képzési karaktere”[12]: 50/50 (kredit%)	
A **tanóra**[2] **típusa**: ea. / szem. / gyak. / konz. és **óraszáma**: heti 2 előadás és 2 óra gyakorlat az adott **félévben**, (*ha nem (csak) magyarul oktatják a tárgyat, akkor a **nyelve**: ……)* Az adott ismeret átadásában alkalmazandó **további** (*sajátos*) **módok, jellemzők**[3] *(ha vannak)*: ……	
A **számonkérés** módja (koll. / gyj. / **egyéb**[4]**)**: Előadás: kollokvium, Gyakorlat: gyakorlati jegy Az ismeretellenőrzésben alkalmazandó **további** (*sajátos*) **módok**[5] *(ha vannak)***:** ……	
A tantárgy **tantervi helye** (hányadik félév): 3	
Előtanulmányi feltételek *(ha vannak)*: *Diszkrét matematika*, *Programozás alapjai*	

Tantárgy-leírás: az elsajátítandó **ismeretanyag tömör, ugyanakkor informáló leírása**

A tantárgy célja:

A tárgy az adatbázisok kezelésével és tervezésével kapcsolatos elméleti és gyakorlati ismereteket foglalja össze. A tárgy keretében a hallgatók megismerkedhetnek a relációs adatbázis-kezelő rendszerek működésével a relációs adatbázisok tervezésével és az ehhez kapcsolódó elméleti eredményekkel. A tárgy célja, hogy hallgatók ismerjék és önállóan alkalmazzák a relációs adatbázisok tervezésének lépéseit, az adatmodellezést, relációs adatbázissémák átalakítását a redundáns adattárolást csökkentő normálformákra. Képesek lesznek adatbázissal támogatott szoftverek önálló elkészítésére és az adatbázisok SQL nyelven történő kezelésére.

Témakörök / tartalom:

1. Adatbázis-kezelő rendszerek feladatai, komponensei.
2. Az egyed-kapcsolat modell lényege, összetett és többértékű attribútumok, gyenge entitások, altípusok kezelése.
3. A relációs adatmodell: tábla, rekord, mező fogalma. Szuperkulcs, kulcs, elsődleges kulcs és külső kulcs (idegen kulcs) fogalma, relációs adatbázis séma.
4. Egyed-kapcsolat modellből relációs modell létrehozása: egyedek, kapcsolatok, többértékű attribútumok és altípusok leképezése.
5. A relációs algebra alapvető műveletei.
6. Funkcionális függőség fogalma, attribútumhalmaz lezártja.
7. Tábla dekompozíciója két táblára, hűségesség. Dekompozíció a függőség alapján (Heath tétele).
8. Normalizálás: 1., 2. és 3. normálforma, Boyce-Codd normálforma, 4. normálforma.
9. Az SQL nyelv alapjai. Relációsémák, kulcsok és indexek kezelése, adattáblák aktualizálása.
10. Lekérdezések SQL-ben: a relációs algebra műveleteinek megvalósítása, összesítő függvények, alkérdések.
11. Virtuális tábla (nézettábla) fogalma, létrehozása, használata.
12. Aktív elemek, megszorítások és triggerek SQL-ben.
13. A beágyazott SQL lényege, utasításai. Speciális beágyazási technikák: ODBC, PHP.
14. Tranzakciós feldolgozás, jogosultság kezelés SQL-ben.
15. Egy konkrét adatbázis-kezelő rendszer megismerése, mintaalkalmazás fejlesztése.

A **2-5** legfontosabb *kötelező*, illetve *ajánlott* **irodalom** (jegyzet, tankönyv) felsorolása bibliográfiai

[1] A Magyar Akkreditációs Bizottság honlapjának 2018. januári állása alapján, az ott szereplő űrlapot - tantárgyleírásra konkretizált résszel – kiegészítve készült.

[2] **Nftv. 108. §** 37. *tanóra*: a tantervben meghatározott tanulmányi követelmények teljesítéséhez az oktató személyes közreműködését igénylő foglalkozás (előadás, szeminárium, gyakorlat, konzultáció), amelynek időtartama legalább negyvenöt, legfeljebb hatvan perc.

[3] pl. esetismertetések, szerepjáték, tematikus prezentációk stb.

[4] pl. folyamatos számonkérés, évközi beszámoló

[5] pl. esettanulmányok, témakidolgozások, dolgozatok, esszék, üzleti, szervezési tervek stb. bekérése

adatokkal (szerző, cím, kiadás adatai, (esetleg oldalak), ISBN)

Ullman J. D., Widom J.: Adatbázis rendszerek - Alapvetés. Második, átdolgozott kiadás, Panem, 2008.
Gruber M.: SQL A-Z. Kiskapu kiadó, 2003.
Pétery Kristóf: Access 2000. LSI Oktatóközpont, Budapest, 2000.
Reese, G., Yarger, R. J., King, T.: A MySQL kezelése és használata. Kossuth Kiadó, 2003.
Katona E.: Adatbázisok. Előadási jegyzet, Szegedi Tudományegyetem, 2008. www.inf.u-szeged.hu/oktatas/jegyzetek
Dr. Balázs Péter, Dr. Németh Gábor: Adatbázisok. Egyetemi jegyzet. Szegedi Tudományegyetem, 2019.

Azoknak az **előírt szakmai kompetenciáknak, kompetencia-elemeknek** (*tudás*, *képesség* stb., *KKK* **7.** *pont*) a felsorolása, **amelyek kialakításához a tantárgy jellemzően, érdemben hozzájárul**

pl.:

Tudás	Képesség	Attiitűd	Autonómia/felelősség
Informatika szakterületéhez kapcsolódó matematikai, számítástudományi elvek, tények, szabályok, összefüggések	Képes az általános és specifikus matematikai, számítástudományi elveket, tényeket, szabályokat, összefüggéseket alkalmazni informatikai szakterületen	Vállalja és hitelesen képviseli informatikai szakterülete szakmai alapelveit.	
Informatika szakterület általános elméletei, összefüggései, tényanyagai, fogalomrendszere	Képes az informatika formális modelljeinek alkalmazására.	Nyitott a képesítésével, szakterületével kapcsolatos szakmai, technológiai fejlődés és innováció megismerésére és befogadására.	
Informatika szakterület tervezési, fejlesztési, működtetési és irányítási folyamatainak alapvető feladatmegoldási elvei, módszerei és eljárásai	Képes az informatikai szakterület tudásanyagát alkalmazni meglévő rendszertervek értelmezése és szoftverfejlesztési módszertanok és technológiák alkalmazása során.	Reflektív módon tekint saját szakmai kompetenciáira és tevékenységére.	Felelősséget vállal szakmai tevékenységéért.
Ismeri az informatikai rendszerek hardver és szoftver elemeinek működését, megvalósításuk technológiáját, működtetéséből származó feladatok megoldásának mikéntjét, valamint informatikai és egyéb műszaki rendszerek összekapcsolásának lehetőségeit.	Képes az informatikai szakterület tudásanyagát alkalmazni WEB-es alkalmazások fejlesztésére. Képes az informatikai szakterület tervezési, fejlesztési, üzemeltetési és irányítási rutinfeladatainak ellátására szoftver rendszerek, adatbázis kezelő rendszerek, vállalati információs rendszerek, döntéstámogató rendszerek, szakértői rendszerek esetében.	Törekszik a folyamatos szakmai képzésre és általános önképzésre. Reflektív módon tekint saját szakmai kompetenciáira és tevékenységére.	Törekszik a hatékony és minőségi munkavégzésre.
Alapvető adatbiztonsági ismeretekkel bír.	Képes az informatikai szakterület tudásanyagát alkalmazni információbiztonsági és kriptográfiai problémák esetében.		Munkáját az információbiztonsági szempontok tiszteletben tartásával végzi.
Ismeri az informatika és a mérnöki szakma szókincsét és kifejezési sajátosságait magyar és angol nyelven, legalább alapszinten.	Képes a szakmai információforrások használatára, a megoldandó problémához szükséges ismeretanyag megkeresésére. Meglévő ismereteire alapozva hatékonyan sajátít el új		

Tudás	Képesség	Attiitűd	Autonómia/felelősség
	technológiákat és paradigmákat.		
Ismeri és érti az analízis, valószínűségszámítás, lineáris algebra, operációkutatás, statisztika, illetve a számítástudomány alapvető fogalmait és összefüggéseit, valamint az alkalmazási területekhez kapcsolódó rutinszerű problémák formális modelljeit.			
Ismeri a számítástechnikai infrastruktúra elvi komponenseit, a hardver komponensek elvi felépítését, a kommunikációt és a rendszerszoftvereket, az adatmenedzsment területeit, beleértve az adatbázisok, adatfeldolgozás, reprezentáció és vizualizáció alapvető fogalmait is.	Képes alkalmazást fejleszteni, kliens-szerver és WEB, mobil rendszereket programozni, multiplatform rendszereket kialakítani. Az elsajátított informatikai eljárások és módszerek segítségével képes valós üzleti, szervezeti körülmények között az alkalmazások működési feltételeinek feltárására, előnyök, veszélyek, kockázatok mérlegelésére és kommunikációjára. Képes adatbázisok menedzselésével kapcsolatos feladatok ellátására, egyszerű adatmigrációs feladatok megoldására.		
Ismeri a programozással összefüggésben az alapvető programozási struktúrákat, a szoftverfejlesztés módszertanát és a fontosabb programozási környezeteket.		Nyitott az új módszerek, programozási nyelvek, eljárások megismerésére és azok készség szintű elsajátítására. Törekszik a hatékony és minőségi munkavégzésre. Reflektív módon tekint saját szakmai kompetenciáira és tevékenységére.	

A tantárggyal kialakítandó <u>konkrét</u> tanulási eredmények:

Tudás	Képesség	Attitűd	Autonómia-felelősség
Ismeri az adatbázis-kezelő rendszerek feladatait, komponenseit.	Felismeri az adatok és az adatbázis-kezelő rendszerek típusait. Megkülönbözteti az adatbázis-kezelő rendszerek komponenseit és azok feladatait.	Törekszik az adatbázis-kezelő rendszerekkel kapcsolatos megfelelő fogalomhasználat elsajátítására.	Alkalmazza az adatbázis-kezelő rendszerekkel kapcsolatos fogalmak és komponensek megfelelő használatát megnyilatkozásaiban.
Ismeri az egyed-kapcsolat modellezés feladatát. Tisztában van az egyed-kapcsolat diagram jelölésrendszerével.	Felismeri az adatbázisban tárolandó adatok közötti összefüggéseket és elkészíti ezt a logikai kapcsolatot grafikusan leíró egyed-kapcsolat diagramot.	Törekszik az adatok csoportosítására és a közöttük lévő kapcsolatok megállapítására.	Önállóan elkészíti az adatbázisban tárolandó adatok közötti logikai összefüggéseket leíró egyed-kapcsolat diagramot.
Ismeri a relációs adatmodell alapfogalmait (tábla, rekord,	Megfelelően használja a relációs adatmodell	Szem előtt tartja a helyes fogalomhasználatot a relációs	Felelősséget vállal az általa készített dokumentációk helyes

mező, szuperkulcs, kulcs, elsődleges kulcs, relációs adatbázis séma).	alapfogalmait dokumentációkban, rendszertervekben, leírásokban.	adatmodellel kapcsolatos leírások készítésénél.	fogalomhasználatáért. A hibás megfogalmazásokat önállóan javítja.
Ismeri az egyed-kapcsolat modellek relációs adatbázissémákká alakításának módját.	Alkalmazza az egyed-kapcsolat modell leképezésének szabályait.	Törekszik az egyed-kapcsolat diagram helyes leképezésére. Figyelembe veszi az egyszerűsítési szabályokat.	Önállóan készít egy egyed-kapcsolat modellből a relációs adatbázis sémákat. Felismeri és javítja a hibásan leképezett sémákat, a hiányosságokat.
Tisztában van a relációs algebra alapvető műveleteivel.	Helyesen alkalmazza a relációs algebra műveleteit.	Szem előtt tartja a precíz munkát a relációs algebra műveleteinek végrehajtásakor és leírásakor.	Felismeri és javítja hibásan megfogalmazott, leírt relációs algebrai kifejezéseket.
Tisztában van a funkcionális függőség fogalmával és az attribútumhalmaz lezártjával.	Felismeri az adatbázis leírásokban a funkcionális függőségeket. Meghatározza egy attribútumhalmaz lezártját.	Alapos az adatbázis leírásokban szereplő attribútumok közötti funkcionális függőségeinek felderítésében.	Önállóan megállapítja az adatbázis leírásokban szereplő funkcionális függőségeket.
Ismeri a táblák dekompozíciójára, hűségességére vonatkozó alapfogalmakat és tételeket (pl. Heath tétele).	Hűséges módon felbontja a táblákat. Heath tétele alapján képes megkülönböztetni a hűséges és nem hűséges felbontásokat.	Szem előtt tartja a táblák felbontásainál a hűséges felbontások szabályait.	Önállóan megállapítja a hűséges és nem hűséges tábla-felbontásokat.
Ismeri a normalizálás alapfogalmait, az 1., 2., 3., 4. és a Boyce-Codd normálformára vonatkozó állításokat. Ismeri a relációs adatbázissémák különböző normálformákra hozásra vonatkozó algoritmikus lépéseket..	Helyesen alkalmazza az egyes normálformákra (1NF, 2NF, 3NF, 4NF, BCNF) vonatkozó átalakító lépéseket.	Figyelembe veszi a relációs adatbázissémák lehetőségét a redundáns adattárolás csökkentése érdekében.	Önállóan normalizálja a relációs adatbázis sémákat 1NF, 2NF, 3NF, 4NF és BCNF normálformáknak megfelelően.
Ismeri az SQL nyelv alapjait.	Létrehoz, töröl, módosít, aktualizál táblákat SQL nyelven relációs adatbázisokban. Kezeli relációsémákat, kulcsokat, indexeket. Ismeri az adattáblák aktualizálásának műveleteit SQL nyelven.	Szem előtt tartja az SQL utasítások szintaktikai szabályait és az adatvesztés lehetőségeit aktualizáló utasítások során. Az adatvesztést elkerülendő, különös megfontoltsággal készít és hajt végre törlésre és módosításra vonatkozó SQL utasításokat.	Önállóan megírja az SQL nyelvű utasításokat relációs adatbázisokban történő szerkezeti és tartalmi változtatásokhoz. Az adatvesztést elkerülendő, különös megfontoltsággal készít és hajt végre törlésre és módosításra vonatkozó SQL utasításokat.
Ismeri az SQL lekérdezésekre vonatkozó utasításokat.	Megfogalmazza az SQL lekérdezéseket a relációs adatbázisokból történő adatkinyeréshez.		Önállóan megírja és végrehajtja az SQL nyelvű lekérdezéseket.
Ismeri a virtuális tábla fogalmát,. Ismeri a virtuális táblák létrehozására és használatára vonatkozó SQL utasításokat, szabályokat.	Létrehoz SQL nyelven nézettáblákat relációs adatbázisokban.		Önállóan létrehoz nézettáblákat SQL nyelven relációs adatbázisokhoz.
Ismeri a relációs adatbázisok aktív elemeinek (megszorítások, feltételek, triggerek) lehetőségét és használatát SQL nyelven.	Alkalmazza az SQL nyelv által nyújtotta aktív elemeket relációs adatbázisok készítésénél és használatánál.	Megfontolja és magára nézve kötelezőnek tartja az aktív elemek használatát relációs adatbázisokban a táblák létrehozásánál és az aktualizáló műveleteknél.	Önállóan felismeri az aktív elemek használatának lehetőségét a relációs adatbázisoknál és alkalmazza azokat.
Ismeri a beágyazott SQL utasításait valamint a speciális beágyazási technikák lehetőségeit az ODBC függvénykönyvtárral C-ben és PHP nyelven.	Képes C és PHP nyelvű programokban beágyazott SQL, ODBC, és PHP függvénykönyvtárak segítségével adatbázis-kezelő funkciót készíteni.		Önállóan készít C és PHP nyelvű programrészleteket adatbázis-kezeléshez.
Ismeri a tranzakciós feldolgozás alapelveit, az ezzel kapcsolatos fogalmakat. Tisztában van a jogosultságkezelés lehetőségeivel SQL nyelvben.	Megállapítja a tranzakciós szintek közötti különbségeket és anomáliákat. Alkalmazza a differenciált jogosultságkezelést a relációs adatbázisoknál.	Megfontoltan határozza meg az adatbázis-elemekre vonatkozó jogosultsági műveleteket, engedélyeket és korlátozásokat. Megfontoltan határozza meg a tranzakció-kezeléssel kapcsolatos beállításokat.	Önállóan megváltoztatja az adatbázis-elemek jogosultságait SQL nyelven. Felelősséget vállal az általa létrehozott adatbázis-elemek jogosultsági beállításaiért (pl. illetéktelen hozzáférés).
Ismeri a MySQL adatbázis-kezelő rendszer használatát, annak összetevőit és adatbázis-kezelésre vonatkozó parancsait.	Elkészít egy MySQL adatbázissal támogatott szoftvert PHP nyelven (webes felületre).	Figyelembe veszi a moduláris szoftverfejlesztés során az adatbiztonsági szempontokat (például a forrásfájlokra vonatkozó láthatósági	Önállóan elkészíti egy MySQL adatbázissal támogatott webes felületű alkalmazást PHP nyelven. Felelősséget vállal a programban lévő sérülékenység (adatelérés,

		szinteket) valamint az adatbázis-használatának szükséges idejét. Törekszik a program sérülékenységének csökkentésére.	hekkelés) lehetőségért.

Tantárgy felelőse (*név*, *beosztás*, *tud. fokozat*)**: Dr. Balázs Péter, egyetemi docens, PhD habil**

Tantárgy oktatásába bevont oktató(k), ha van(nak) (*név*, *beosztás*, *tud. fokozat*)**: Dr. Németh Gábor, adjunktus, PhD**

Irodalomjegyzék

[1] Adonyi Róbert: Adatstruktúrák és algoritmusok. Typotex Kiadó, 2011.

[2] Gazihan Alankus, Rogério Theodoro de Brito, Basheer Ahamed Fazal, Vinicius Isola and Miles Obare: Java Fundamentals. Packt Publishing, 2019.

[3] Grant Allen, Mike Owens: The Definitive Guide to SQLite, Second Edition. Apress, 2010.

[4] Karthik Appigatla: MySQL 8 Cookbook. Packt Publishing, 2018.

[5] Tricia Ballad, William Ballad: Biztonságos webalkalmazások PHP nyelven. Kiskapu Kft., 2010.

[6] Donald D. Chamberlin: Early History of SQL. IEEE Annals of the History of Computing, pp. 78-82, IEEE, (2012).

[7] Thomas H. Cormen, Charles E. Leiserson: Algoritmusok. Műszaki Könyvkiadó Kft. 2003.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Új algoritmusok. Scolar, 2008.

[9] Hector Garcia-Molina, Jeffrey D. Ullmann, Jennifer Widom: Adatbázisrendszerek megvalósítása. Panem Kft, 2008.

[10] Erich Gamma, Richard Helm: Programtervezési minták. Kiskapu Kiadó Kft, 2004.

[11] Sibsankar Haldar: SQLite Database System Design and Implementation. Google Books, O'Reilly, 2015.

[12] Dr. Katona Endre: Adatbázisok. Szegedi Tudományegyetem, 2013.

[13] Katona József, Kővári Attila: Objektumorientált szoftverfejlesztés alapjai. Kiadó nélkül, 2015.

[14] Kishore Bhamidipati: SQL programozói referenciakönyv. Panem Könyvkiadó, 1999.

[15] Kondorosi Károly, Szirmay-Kalos László, László Zoltán: Objektum orientált szoftverfejlesztés. ComputerBooks Kft., 2007. (Tankönyvtár: `https://www.tankonyvtar.hu/hu/tartalom/tkt/objektum-orientalt/index.html`)

[16] Kozmajer Viktor: PHP és MySQL az alapoktól. BBS-INFO Könyvkiadó és Informatikai Kft., 2011.

[17] Kövesdán Gábor: Szoftverfejlesztés Java SE platformon. Magánkiadás, 2018.

[18] Jay A. Kreibich: Using SQLite. O'Reilly Media, 2010.

[19] Kusper Gábor, Radványi Tibor: Programozás techinka. Eszterházy Károly Főiskola, Kempelen Farkas Hallgatói Információs Központ, 2011. (Tankönyvtár: `https://www.tankonyvtar.hu/hu/tartalom/tamop425/0038_informatika_Projektlabor/index.html`)

[20] Dr. Leitold Ferenc: Adatbiztonság, adatvédelem. Dunaújvárosi Főiskola, 2011. (Tankönyvtár:`https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0035_adatbiztonsag_adatvedelem/adatok.html`)

[21] Radványi Tibor: Adatbázisrendszerek. Eszterházy Károly Főiskola, 2011.

[22] Nick Samoylov: Learn Java 12 Programming. Packt Publishing, 2019.

[23] Steven Suehring, Janet Valade: PHP, MySQL, JavaScript&HTML5. Tantusz Könyvek. Panem Kft., 2014.

[24] Szabó Bálint: Adatbázis-kezelés, Eszterházy Károly Főiskola, 2011.

[25] Laura Thomson, Luke Welling: PHP és MySQL webfejlesztőknek - Hogyan építsünk webáruházat? Perfacto-Pro Kft., 2010.

[26] Jeffrey D. Ullmann, Jennifer Widom: Adatbázisrendszerek - Alapvetés. Panem Kft, 2009.

[27] Eric Vanier, Tejaswi Malepati and Tejaswi Malepati: Advanced MySQL 8. Packt Publishing, 2019.