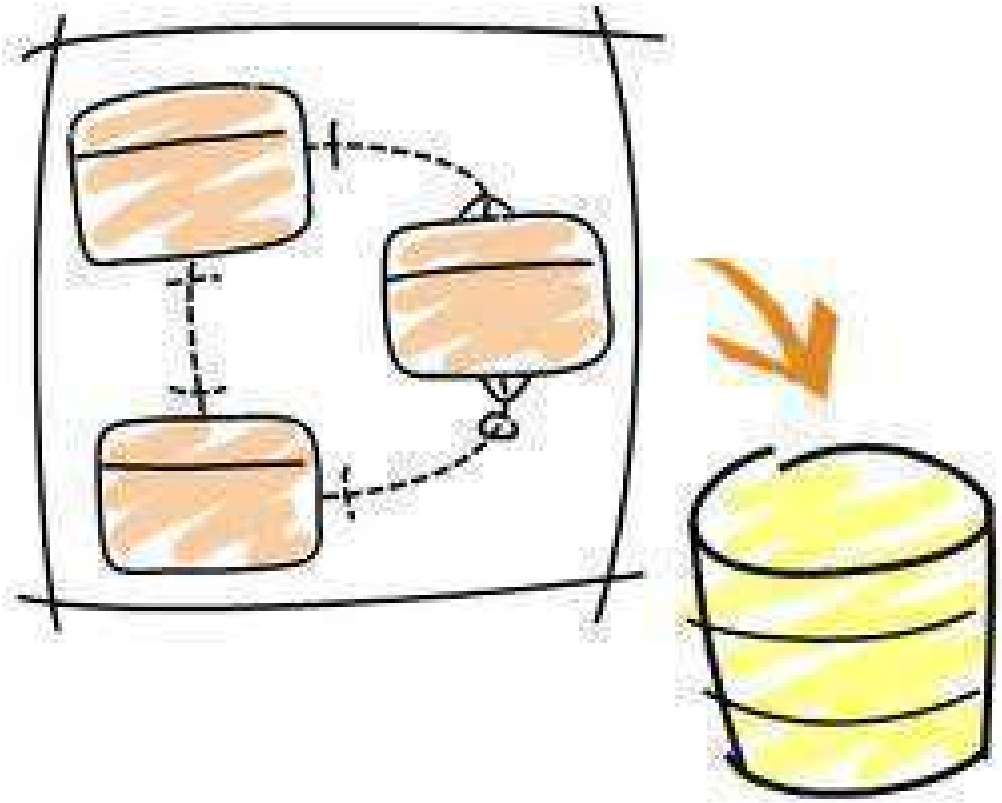
PETKOVICS IMRE

ADATBÁZISOK

TANKÖNYV



SZABADKA, 2013.

TARTALOMJEGYZÉK

1. ELŐSZÓ

Manapság már általánosan elfogadott az a nézet, hogy a fizikai valóság három alapvető "alapanyaga" az anyag, az energia és az információ. Ezt a felfogást igazolják az élőlények is, és az ember által létrehozott szervezetek (szervezett rendszerek) is. A szervezetek nem tudnak fennmaradni információáramlás nélkül: a fejlődésükhöz, szervezettségük folyamatos javításához és továbbfejlesztéséhez folyamatos adat- és információáramlásra van szükség. Ezt a célkitűzést az ember teremtette rendszerek csak a külső környezetükben bonyolódó folyamatok és jelenségek megfigyelésével és az azokat leíró adatok/információk gyűjtésével, rendszerezésével és az azokhoz való alkalmazkodással valósíthatják meg. A rendszer stabilitása a benne lejátszódó folyamatok dinamikájának elemzésével becsülhető meg[1].

Az információt ma ugyanolyan erőforrásnak tekintik, mint a nyersanyagot, a mukaeszközöket vagy a munkaerőt: kulcsfontosságú szerepet tulajdonítanak neki a szervezetek életében illetve működésében. Fontosságát, szükségességét, egyszóval stratégiai szerepét minden tudományos és szakmai fórumon hangsúlyozzák. Ebből ered az igény az ismeretek (adatok, információk) felkutatására, elrendezésére, tárolására, kezelésére, feldolgozására és megjelentetésére. Ezzel a területkörrel foglalkozik az informatika, és az információs rendszerek is ezeket a célkitűzéseket igyekeznek megvalósítani. Az információs rendszerek egyik vetületét adatvetületnek hívják. Az adatvetület az információs rendszerek/alkalmazások adat- illetve adatszolgáltatás-igényeit hivatott biztosítani. Az adatvetületbe manapság az adatbázistervezéstől az információs rendszerek adattal való kiszolgálásáig minden »beleérthető« és bele is tartozik. A fogalmi szintű adatmodellek és adatmodellezések csakúgy, mint a logikai szintű adattervek kidolgozása vagy az adatbázis fizikai megvalósítása. Nem beszélve az adatkezelés sokszínű és szerteágazó megvalósítási lehetőségeiről. A gyakorlat szempontjából mindenképpen fontos a hatékony adattárolási és adatvisszanyerési algoritmusok ismerete és alkalmazási készségének elsajátítása. Az adatbázisok (kialakításának) elmélete és (használatának) gyakorlata már jó régen önálló tudományterületté vált: függetlenítette magát az információs rendszerek témakörétől (bár továbbra is azt szolgálja ki). Ebben a jegyzetben tehát az információs rendszerek adatvetülete kerül bonckés alá, Adatbázisok címen.

Ez a jegyzet, bármilyen szempontból is történik a megítélés, hiánypótló. Az adatbázisok témakörébe vágó ismereteket is tartalmazó magyar nyelvű jegyzet utoljára 2002-ben jelent meg, Információs rendszerek és adatbázisok címen. Szerb nyelven Baze podataka címen ugyan 2004-

[1] A.J.Lerner gondolatai szabadon fogalmazva.

ben megjelent a tantárgyi segédlet, de a fordítását már nem érhette meg: helyette ez a »vadonat új« (ismereteket is tartalmazó), más ismeretátadási-ütemezést támogató tankönyv/jegyzet kerül most a hallgatók kezébe.

A tankönyv/jegyzet formába gyűjtött ismeretanyag elsősorban a Szabadkai Műszaki Szakfőiskola másodéves Informatika szakos hallgatóinak készült, akik a harmadik félévben, 2+2-es heti óraszámmal hallgatják az »Adatbázisok« tárgyat. A rendelkezésre álló idő/óraszám nagymértékben korlátozza a tárgy témakörében fellelhető óriási mennyiségű ismeretanyag átadását. Egyrész ezért, másrészt viszont mivel bevezető/alapozó tárgyról van szó, a jegyzetben csak az adatbázisokra vonatkozó alapismeretek kaptak helyet a gyakorlatban elfogadott fejlesztési sorrendhez igazodóan a témakörben használatos fogalmak tisztázása után:

1. A fogalmi szintű tervezés E/K modellel,
2. a relációs adatmodell és az SQL,
3. más, a gyakolatban használatos adatmodellek (objektumos, UML, XML),
4. a modell-leképzések,
5. a normálformák és normalizálás és
6. a triggerek és tárolt eljárások.

A jegyzet célja az, hogy elsősorban elméleti jellegű[2] ismereteket közvetítsen a hallgatók felé, de a nagyszámú feladat a gyakorlati feladatkörökbe is betkintést nyújt. A laborgyakorlatokon a hallgatók konkrét, életből vett, egyszerűsített problémákat oldanak meg, konkrét eszközök (szoftver) segítségével, az előadásokon megismert eljárások alapján.

A leendő informatikus mérnököknek marad a rég megfogalmazott megjegyzés/tanács: csak akkor választottak helyesen szakirányt, ha az ágazat vezérfonala (fő tárgya, témája: esetükben az informatika) egyben a hobbijuk is. És végül kívánom, hogy minden informatikus hallgatónak ne csak szakmája, de hivatása legyen az informatika!

A jegyzetben kétséget kizáróan akadnak kisebb-nagyobb hibák. Az olvasatok számával a hibák száma is, természetesen, fogyatkozik, de a rendelkezésre álló rövid idő alatt nem lehetett őket teljesen kiiktatni. A jegyzetben leírtak jobbá, szebbé és hasznosabbá tételének érdekében mindennemű és -szándékú megjegyzés[3] segíthet.

Dr. Petkovics Imre, okl. villamosmérnök,
az informatikai tudományok doktora
peti@vts.su.ac.rs

Szabadka, 2013. november 13.

[2] "Semmi nem olyan gyakorlati, mint egy jó elmélet." (Boltzmann)
[3] Vitĭa vituperāre facillĭmum est, emendāre difficillimum. A hibát korholni a legkönnyebb, kijavítani a legnehezebb.

2. BEVEZETŐ

2.1. AZ ISMERET, AZ ADAT, AZ INFORMÁCIÓ ÉS A TUDÁS

2.1.1. AZ ISMERET

Az emberi faj folyamatosan gyűjti az ismereteket (adatokat) az őt körülvevő világról. A gyűjtés alapját az ember érzékszervei által biztosított ismeretek (adatok) jelentik. Nem cáfolható az a tény, hogy akár egy mosoly vagy kellemes illat, esetleg ízletes, ínycsiklandó étel is jelentős ismeretet, információt hodozhat, de az emberek közötti kommunikáció (ismeretátadás) legtöbbször a beszéd vagy az írott beszéd, a nyelv segítségével történik. A természetes beszédben a kijelentés (állítás, tényközlés) legegyszerűbb formája az alanyt és állítmányt tartalmazó tőmondat. A hiányos mondatok is kijelentésnek értelmezhetők, ha a hiányzó rész (alany vagy állítmány) az előző kijelentésekből (mondatokból) kiérthető. Az alany a kijelentés (közlés) tárgyát (miről akarunk valamit kijelenteni) jelöli, az állítmány pedig magát a közlést (mit akarunk kijelenteni az alannyal megjelölt tárgyról) tartalmazza.

Az ismerettől út vezet az adatig, melynek során az „ismeret mesterséges képet ölt, és elveszti természetes képét”(Halassy-E-I-R, 175.).

2.1.2. AZ ADAT

Az adat legegyszerűbb megfogalmazásában a hangsúly a lejegyzésen van (itt veszti el az ismeret a természetes képét, a lejegyzésnél), vagyis ez a szemlélet azt vallja, hogy az adat tetszőleges adathordozón (a barlang falán, a csárda gerendáján, papíron, mágneses médiumom, stb.) lejegyzett valamiféle ismeret a valós világ bizonyos jelenségéről, a lejegyzett adat későbbi sorsától teljesen függetlenül. Mellékes tehát, hogy ezt a lejegyzett ismeretet használják-e, és ha igen, mire a későbbiekben. Ha ezeket a rögzített adatokat a továbbiakban a döntések meghozatalában, vállalatvezetésben és –irányításban használják, úgy (a továbbiakban majd látni fogjuk) az adatok információkká alakulnak át. A lejegyzett ismeretek adat jellegüket egészen addig megőrzik, amíg esély van arra, hogy információvá alakuljanak. Ezután az ismeretek adat jellegüket is elvesztik (Halassy-E-I-R).

Ez az adatokkal és információkkal kapcsolatos nézet ma már nem állja meg a helyét. Ma már azt valljuk, hogy az adat az információ »megtestesülése«, konkrét megjelenési formája, olyan jelhalmaz amelyet számítógépen is feldolgozhatunk, de »szegényebb«, kevésbé jelentős, mint az információ. Az információ tartalmilag gazdagabb az adatnál. Az adat értelmezhető, de

nem értelmezett ismeret. Az információ ezek szerint értelmezett adat, értelmezett ismeret. Az adatok tehát információhordozók, de az adatok »információgazdagsága« igencsak viszonylagos fogalom, hiszen nagymértékben függ az adat vagy információ befogadójától. Ugyanaz az adat egyesek számára továbbra is csak adat marad (a befogadó nem tudta értelmezni), másoknak esetleg értékes információt hordoz (ők értelmezni is tudták a kapott adatot).

Az adat tehát olyan értelmezhető, de nem értelmezett jelsor, amely objektumok állapotát és/vagy tulajdonságait, vagy a valós világ egyéb jellemzőit fogalmazza meg (rögzítésre, szállításra és feldolgozásra megfelelő alakban) a későbbi felhasználás és/vagy feldolgozás céljából.

A modern vállalkozások boldogulása a mindinkább teret hódító piacgazdaság útvesztőin nagyon nehéz. A mai, dinamikus világ sok buktatót és kockázatot rejteget a túlélést kereső cégeknek mint valaha. A világpiac globalizációs törekvései, a felgyorsult technológiai fejlődés és az Internet adta lehetőségek új utak, új ötletek, új üzleti logika felkutatására ösztönzik a vállalkozások, cégek irányítóit a fennmaradás illetve fejlődés érdekében. A vállalatvezetők feladata, hogy olyan körülményeket biztosítsonak, amelyek lehetővé teszik termékeik illetve szolgáltatásaik minőségének folyamatos javítását a költségek csökkentése, de legalább szintentartása mellett.

»Meghatározó tényezővé válni és annak (vagy egyáltalán) megmaradni a piacon«, ez a mai vállalatok legfontosabb célkitűzése. Ez azt jelenti, hogy a termékek és szolgáltatások állandó minőségellenőrzése folyik az igényes cégeknél, figyelve az ágazati konkurrenciára, és egyáltalán a környező világra, hiszen csak így becsülhető meg a vállalat mindenkori helyzete és kilátásai. A környezet legfontosabb szubjektumai az alvállalkozók, a szállítók, a megrendelők, az ide vonatkozó törvények, előírások és szabványok, valamint (utolsóként, de nem utolsó sorban) a fogyasztók. A legjobbnak tűnő ötletek, legzseniálisabbnak vélt újítások is halva születnek, ha a piac (a fogyasztók) nem fogadja el őket. A piac működéséről (kereslet-kínálat alakulásáról), az ágazati szervezetek eredményeiről és a partnerekről tanúskodó **adatoknak egyre nagyobb a becsülete** (értéke, fontossága) a vállalatvezetők körében. Sőt, ezen adatok szerepe együtt növekszik magával a vállalattal, illteve a termékválaszték bővülésével. Az **egyre bővülő adattömeg-igény** a vállalkozás és környezete közötti kommunikáció élénkítésének és bővítésének, illetve a belső, szervezeti egységek közötti kommunikáció fokozásának tudható be, és a fejlődés zálogának tekinthető.

A modern cégek ügyvitelét elsősorban a külső és belső környezetük változása alakítja. A szervezetek válasz-reakciója (felelete) az egyes változásokra, illetve a válaszidő rövidsége határozza meg a cég eredményességét és üzleti sikerét. A mobilitás és a változásokhoz való

alkalmazkodás dinamikája a siker szükséges feltételei közé tartoznak. Ez pedig gyors és jó minőségű kommunikációt (adatáramlást), valamint gyors és az igényeknek megfelelő adatfeldolgozást követel meg. Az ügyvitel fejlesztése céljából elemzik a termelést: vajon minden jól működik a termelési folyamatban, lehet-e javtani az eredményességen vagy a minőségen? Mivel lehetne gazdagítani a terméket illetve szolgáltatást? Hogyan lehetne új piacokat vagy vásárlókat szerezni? Hol vásárolni a nyersanyagot, hol termelni és hol értékesteni az árut figyelembe véve a földrészek, országok nyújtotta össz lehetőségeket (fizikai föltételek, jogi körülmények, munkaerő képzettsége, munkaszokása, bére, igényei, stb.) a termelés és szálltás költségeinek visszaszorítása (minimalizációja) és az eladási ár (olvasd haszon vagy profit) legkedvezőbb szintre hozása (maximalizása) mellett, nem megfeledkezve egy elfogadott minőségszint biztosításáról. Az előzőekben felsorolt dolgokhoz, az elemzésekhez, a döntéshozatalokhoz minden cégben adatokra van szükség (**az adatok használata általános**), és ezek az adatok, a vállalatok külső és belső tényezőit tükröző adatok folyamatosan és egyre gyorsabban változnak.

2.1.3. AZ INFORMÁCIÓ

Az információelmélettel foglalkozó kutatók a mai napig nem tudtak az információ fogalmának pontosabb meghatározásában megegyezni. Ezt igazolandó, lássuk az információ fogalmának különféle definícióit, melyeket nagyon nehéz közös nevezőre hozni.

B. Langefors véleménye a következő:

“Információnak tekintünk minden tudást, üzenetet, amelyet bizonyításokhoz használhatunk, vagy amely lehetővé tesz döntéseket.”[1].

A Webster’s Dictionary-ban az információ két definíciójával is találkozhatunk:

Az információ tudásanyag, intelligencia, hír.

Az információ a tudás vagy intelligencia átadása illetve fogadása.

H. Rittel norvég kutató az információt a tudásszint változásához köti:

“Az információ egy olyan tevékenység, amely valaki tudásának a megváltoztatásához vezet.”[2].

A következő meghatározásban az információ a rendezett ismereteket jelöli:

“Az információ nem egyéb, mint rendezetlen adatsorok vagy tények valamely folyamat által használható formára alakított változata.”[3].

[1] Forrás: [Raffai M-IR-fejl-1999 22.], 13. oldal.
[2] Forrás: [22.], 13. oldal.
[3] J Frates szavai, [22.], 13. oldal.

Hasonló elveket vall NoszkayErzsébet is az információ fogalmáról:

"... az információ nem általában valamiféle új ismeret, hanem olyan új, illetve feltárt ismeret, amely a már meglévő ismeretek (adatok, tények) rendszerezéséből, összevetéséből, elemzéséből, értékeléséből, modellszerű felhasználásából stb. származik."[4].

Az információ értelmezett adat, állítás, ismeret, közlés, hír, amely csökkentheti a bizonytalanságot, döntés és irányítás (vezérlés) alapjául szolgálhat, és mindig gazdagabb, több, mint az adat és az összes fenti fogalom, mert gondolkodásra, döntésre, tevékenységre készteti a címzettet (az információ befogadóját).

Az adatok értelmezhetősége és értelmezése nemesíti őket információvá. Az értelmezésig (az információig) azonban el is kell jutni, méghozzá a következő lépéseken keresztül:

1. érzékelés (detection, perception),
2. észlelés (observation)
3. felfogás (comprehension) és
4. megértés (understanding).

Az informálódás vagy ismeretszerzés (információ szerzés) valamennyi fenti mozzanata (mind a négy lépése) kritikus jelentőségű. Bármelyikben történik is valami rendellenesség, vagy maradjon ki akár egy lépés is, nem jöhet létre információ (ismeretszerzés).

Az érzékelés, mint a neve is mondja, érzékszerveink működését jelenti az adatok fizikai "felfogására", "befogadására". Az érzékszervek (főleg a látás és hallás szervének) csökkent szintű működése hátrányos helyeztbe hozhatják "gazdájukat", az ismeretek címzettjét.

Az észlelés mozzanata (azért kedvelem ezt a kifejezést, mert jól érzékelteti az ismeretszerzés folyamatának gyors bonyolódását) érthető meg talán a legnehezebben az ismeretszerzés folyamatában. Az észlelés azonban jól megfogható a következő hétköznapi (valószínüleg már mindegyikünkkel megtörtént) eset kapcsán. Reggeli közben az emberek nagy része a napi sajtót lapozza. Ha ilyen helyzetben valaki szól hozzájuk, gyakran a következő kérdéssel felelnek vissza: Mit is mondtál? Az érzékelés tehát megtörtént, de az észlelés nem, hiszen a megszólított ember a figyelmét másfelé irányította! Ilyen esetekben, mihelyt észbekap az ember, hogy valaki hozzá szólhatott, megindul az érzékelés "visszajátszása", és ha ez sikertelen volt, következik a visszakérdezés. A második példa az észlelésre legyen a szolgáltató vállalatok számlája. A számla-jelentés annyira túlzsúfolt adatokkal, hogy általában érthetetlen és a fogyasztó számára felfoghatatlan, hogy hányadán is áll a számla kiállítójával. Egy időben például (pedig ekkor az informatikai részleg főnöke voltam a szabadkai áramelosztóban) nem tudtam, hogy pontosan mennyi is lesz a villanyszámlámon a tartozás (fogyasztás) összege, bár a

[4] Izvor: [20.Noszkay E-GDF-1994], strana: 9.

fogyasztást, ugye le tudtam olvasni a fogyasztásmérőről. A számításhoz szükséges elemek is a számlán helyezkedtek el valahol, de annyira eldugva, hogy ember legyen a talpán, aki megtalálja valamennyit! Nota bene[5]: az észleléshez az ismeretek fizikai szervezése is hozzájárul, ezért a fontos ismereteket mindig feltűnő, szembetűnő helyen kell megjelentetni, hogy azok azonnyomban észlelhetők legyenek.

A felfogás mozzanatának is megvan a saját jellegzetessége. Az ismereteket legtöbbször kijelentés (állítás) vagyis hír formájában fogalmazzuk meg. A kijelentést csak az a címzett tudja felfogni, aki érti annak minden szavát, másszóval, ha a hír megfogalmazója és megcélzottja (címzettje) »ugyanazt a nyelvet beszélik«.

Az ismeret értelmezése nem más, mint az új állítás (hír, adat) összevetése, összekapcsolása a már előbb megszerzett információkkal (ismeretekkel, tényekkel). Ha nem léteznek előző információk, amelyekkel az új kijelentés (ismeret) összeköthető, az ismeret értelmezése nem lehetséges, vagyis új információ nem keletkezik. Új ismeret (információ) akkor sem keletkezik, ha a kapott adat kötése az előzőekben már megtörtént. Információ akkor születik, amikor a kapott adat (hír, állítás) tartalmáról gondolkodunk, amikor valamilyen véleményt fogalmazunk meg vele kapcsolatosan, amikor elhelyezzük a már meglévő ismereteink közé (között).

Az előző megállapításokra alapozva bátran kijelenthető, hogy a számítógépeken csupán adatokat (és nem információkat) tárolunk azzal a céllal, hogy később ezeket visszakeressük, átalakítsuk, feldolgozzuk. Információk csak az emberek fejében születnek, és az információs rendszerek, vagy feldolgozások csak akkor jók, ha megkönnyítik és meggyorsítják az értelmezés folyamatát.

2.1.4. A TUDÁS

Ismereteink a valós világról a jelenségeinek és a folyamatainak megismeréséből és tanulás útján keletkeznek, és következtetés (vélemény) megfogalamzását, ok-okozati összefüggés-vizsgálatot, elvonatkoztatást és új ismeretek megszerzését teszik lehetővé egy magasabb elvonatkoztatási szinten, amit tudásnak nevezünk.

»A tudás a valós világnak, az abban létező dolgoknak, tényeknek, eseményeknek, jelenségeknek, az azok között fennálló kapcsolatoknak, okozati összefüggéseknek az emberi tudatban történő visszatükröződése.« (Raffai M.: Információrendszer-fejlesztés, Novodat Kiadó, Győr, 1999., 15. old.**)**

[5] Jegyezd meg jól

2.2. ADATKEZELÉSI MÓDOK

Ha csupán a klasszikus számítógépes ismeretkezelésre gondolunk (figyelmen kívül hagyva a multimédia bámulatos, és egyre bővülő lehetőségeit: a többfajta grafikai ábrázolást, a hanganyagot, az álló- és mozgóképeket hordozó adatokat), akkor annak két fajtáját ismerjük és műveljük. Ezek pedig a:

1. tényszerű (faktografikus) vagy adatszerű és
2. szövegszerű ismeretkezelés.

A tényszerű ismeretkezeléssel (adatkezeléssel) kapcsolatban meg kell említeni, hogy már a görögök a feljegyzésre szánt tényeket (versenyek győzteseit, azok eredményeit, felismert és bizonyított természeti és társadalmi törvényszerűségeket, dalokat, mondákat, hiedelmeket, stb.) rekordoknak nevezték. Ez a fogalom, ha értelmében módosulva is, de fennmaradt mind a mai napig. Ma a világraszóló eredményeket hívjuk rekordoknak, de ugyanakkor a megfigyelt jelenséget leíró adatsornak is ez a neve a számítógépes adatfeldolgozás területén (és nemcsak magyar nyelven). A valamilyen szempontból hasonló jelenségeket leíró adatsorokat általában egymáshoz közel raktározták (regisztrátorok, dossziék) régen is, a dosszié angol megfelelője pedig talán a legelterjedtebb fogalom a számítástechnikában: a fájl (file). A hasonló jelenségeket leíró adatokat tartalmazza a fájl, a tételei pedig a rekordok (amelyek a konkrét jelenségeket írják le). A hasonló jelenségeket elíró adatsor (a tétel vagy rekord) adataira azt modjuk, hogy azok a mezők. A hierarchikus felépítés tehát: fájl – rekord – mező – mező tartamla. Ezeknek a következő konceptuális szintű fogalmak (fogalmi szintű lényegek) felelnek meg: egyedtípus – egyed-előfordulás – tulajdonságtípus – tulajdonságérték.

A tényszerű ismeret- vagy adatatkezelést ma adatfeldolgozásnak hívják, és két alapvető módját ismerjük:

1. fájl- vagy állománykezelés és
2. adatbáziskezelés.

A klasszikus és túlhaladott állománykezelés extenzionális jellegű kezelést jelentett, az egyedtípusok (fájlok) közötti kapcsolatokkal senki sem törődött. Az adatoknak a háttértárolókon való előnyös fizikai elhelyezése volt a legfontosabb feladat a fájlkezelésnél, az adatok értelmével, jelentőségével kapcsolataival senik sem törődött.

Az adatbázis az egyed-előfordulások, tulajdonságértékek és kapcsolattípus-értékek fizikailag természetesen állományokban elhelyezett, de mindenképpen adatmodell szerint szervezett együttese. Az adatbáziskezelés nem zárja ki a klasszikus, állományszerű kezelést (az állományban-táblában való vertikális mozgást, vagyis az extenzionális kezelést), azonban rá

elsősorban a transzverziális feldolgozás (a táblák-állományok köztötti vízszintes mozgás) a jellemző.

A szövegszerű ismeretkezelést nem szabad összetéveszteni a szövegszerkesztéssel! A szövegszerkesztők szövegírásra, a begépelt szöveg megjelenítésére, és külalaki csinosítására alkalmasak, de adatkezelésre semmiképpen sem. A szövegállományoknak nincsenek rekordjaik, mezőik, szóval az alapvető egyed-tulajdonság-kapcsolat fogalomhármasnak (és előfordulásaiknak) a megfelelője ismeretlen a szövegszerkesztés gyakorlatában, így ismeretkezelésről nem beszélhetünk.

A szövegszerű ismeretkezelés két, ma is alkalmazott formáját a

1. szövegkezelő rendszerek és
2. táblázatkezelők

biztosítják.

A szövegkezelő rendszerek közös megegyezésen alapuló, általános érdeklődésre vonatkozó adatokkal gazadagított (szerző, cím, megjelenés helye, megjelenés időpontja, a témát megjelölő kulcsszavak, stb.) bármilyen tartalmú leírásokat, megfogalmazásokat tartalmazhatnak. Emellett adatszótárral (tezaurusz) is kiegészítik őket, melynek keretében a deszkriptorok a szótárban felvett kulcsszavak (előfordulási értékeinek) megjelenési helyét rögzítik, a tulajdonságtípus-tulajdonságérték pároshoz hasonlóan. A szövegkezelő rendszerekben valamilyen ismérv, kulcsszó alapján együtt, „fájlként” kezelhetők a dokumentumok, megvalósítva ezzel a másik adatdimenzió-párost: az egyedtípus-egyed-előfordulás párost. A harmadik páros, a kapcsolattípus-kapcsolatérték azonban hiányzik a szövegkezelő rendszerek megvalósításából, ezért nem tekinthetők adatbáziskezelő rendszereknek is egyúttal. Az ilyen rendszerek becsületes neve adatbank.

Az adatbank olyan igényesen és célszerűen, valamilyen szempont szerint “állományokba” összeválogatott és “fájlok” szintjén is elrendezett dokumentumok együttese, melynek keretein belül a kezelés a “fájlok” tematikájától függetlenül egységes, és bár az ismeretek az “állományok” szintjén átfedőek is lehetnek, tartalmilag “fájl” szinten nem kapcsolhatók össze.

A táblázatkezelők álmukban adatkezelésről álmodnak, sokan tévesen így is hívják őket, de persze attól messze állnak. A fogalomzavar onnan ered, hogy az egyedtípusnak, mint fogalmi szintű lényegnek a logikai szinten a leggyakrabban táblázattal ábrázolt reláció felel meg. A klasszikus táblázat nem ismeri a tulajdonságtípust, így a táblázat bármelyik oszlopának bármelyik elemébe (sorába), tetszőleges érték beírható mindenfajta ellenőrzés nélkül (ami természetesen lehetetlen egy adatbázis esetében). A táblázatban egyedtípusnak megfelelő fogalmak sem léteznek, tehát kapcsolattípusokról sem beszélhetük.

A multimédiás korszak már rég bekopogtatott az életünkbe, hát magától értetődő a kérdés: vajon hogyan illeszkednek be a képek, az animációk, a grafikonok, a film- és hanganyagok, valamint az esszészerű leírások, stb. a klasszikus adatbázis korlátaiba? Az adatmodell és az adatbázis fogalmának megfogalmazásában csak egyedtípusokról és egyed-előfordulásokról, tulajdonságtípusokról és tulajdonságértékekről, kapcsolattípusokról és kapcsolattípus-értékekről (kapcsolatértékekről) van szó, az elraktározás és a megjelenítés formájáról említés nem esik. Ez azt jelenti, hogy az adatmodell és az adatbázis szempontjából a rögzítés és a megjelenítés formája invariáns, teljesen lényegtelen. A definíciók nem érintik a fogalmi szintű lényegek (egyedtípusok, -előfordulások, tulajdonságtípusok, -értékek, kapcsolattípusok és kapcsolattípus-értékek) ábrázolásmódját. A multimediális adatok más, "közönséges" egyedtípusokhoz kapcsolódó egyedtípusokat jelentenek, pl. a hallgató arcképe a hallgató egyedtípushoz kapcsolódó HALLGATÓ_ARCKÉPE egyedtípusként modellezhető. A hallgatónak több arcképe is lehet elraktározva az adatbázisban, ezekre az azonosítóik alapján tudunk egyértelműen hivatkozani, vagyis a HALLGATÓ_ARCKÉPE egyedtípusban a kapcsoló tulajdonságtípus (külső- vagy idegen kulcs) mellett azonosítónak is lennie kell.

A multimédia-elemek klasszikus adtbázissal való házasításának technikai megoldása már nem egyértelmű. Bármilyen fizikai szinten rögzített (becementezett) megoldás hátrányos, hiszen a bővítés és a változáskezelés ilyen esetekben lehetetlen, mert az adatkezelés fizikai és logikai (program) szinten is merev. Sem fizikai sem logikai adatfüggetlenséggel nem lehet dicsekedni.

A modern adatbáziskezelő rendszerek a (fönt felsorolt) fogalmi szintű lényegeket rögzített fizikai tárolási módszerekkel (még ha mindegyik más-más fizikai szerkezetben is, de általában mindegyik többféle tárolási módszert használ), de fogalmi (tartalmi) és logikai szerkezettől függetlenül valósítják meg. Ezért tekintjük őket általánosított adatkezelő rendszereknek. Itt a változáskezelés (tartalmi vagy foglami változtatás – új egyed-, tulajdonság- és kapcsolattípusok hozzáadása) minden hátrányos következmény nélkül, „élőben" is folyamatosan bonyolítható. Ezt a kényelmet a rendszer bonyolultságával kell megfizetni (egyesek szerint a mai adatbáziskezelő rendszerek bonyolultabbak az operációs rendszereknél is). A multimédia-adatok kezelése egészen más jellegű, mint a klasszikus adatkezelés, tehát az egyébként is bonyolult adatbáziskezelő rendszert ki kellene bővíteni a multimédia-adatok kezelésével ahhoz, hogy a szó igazi értelmében általánosított (mindenféle, szélesebb körben elterjedt adatok kezelésére alkalmas) valódi adatbáziskezelő legyen. Mivel ez korántsem egyszerű feladat, teljességgel érthető, hogy a mutimédia szinten általánosnak tekinthető adatkezelők még nem jelentek meg a piacon.

2.3. ADATFELDOLGOZÁS

A számítógépes adatkezelést, adatfeldolgozást és jelentéskészítést a kezdetektől fogva programok, szoftverek, alkalmazások, információs rendszerek végezték. Az információs rendszerek fejlesztésének célja a különböző működési területen gazdálkodó cégek, esetleg személyek informció iránti szükségleteinek kielégítése, adatfeldolgozás segítségével. Egy adott információs rendszeren belül maguk az ismeretek (adatok) is rendszert (adaterendszert) alkotnak.

A múlt század hatvanas éveiben intenzív érdeklődés jelentkezett a cégek ügyvitelének teljes, de inkább részleges lefedésére, támogatására (állomány- és raktárnyilvántartás, könyvelés, közműszolgáltatások elszámolása és számlázása, stb.). Az akkori fejlesztői környezetek (programnyelvek) még nem voltak függetlenek az adatoktól (logikai adatfüggetlenség), így mindegyik programnak tartalmaznia kellett az általa használt adatok adattári leírását (rekord). Ennek következtében mindeféle adatleírás-módosítás, az összes, módosított adattárat használó programok javítását vonta maga után még akkor is, ha az adott program a módosított adatot nem is használta (a program és az adattár rekordleírásának szükségszerűen egybevágónak kellett lenni). Az alkalmazások élete akkor még évekig, évtizedekig tartott, gyakori módosításokkal. Ennek az állapotnak a kiküszöbölésére fogalmazódott meg az az igény, hogy az adatleírásokat (rekordképek) és az adatkezelési eljárásokat függetlenítsék (eltávolítsák a programokból). Az adatkezelési eljárások általánosítása és optimalizálása a függetlenítés mellett nagyban megkönnyítette a programozók munkáját. A függetlenített, a korábbinál intelligensebb adatkezelő rendszerek (amelyeket előbb adatbankrendszernek, majd adatbázisrendszernek neveztek) mutatták a kiutat a szoftverkrízisből. Ezeket a törekvéseket támogatta (hardver oldalról) a diszkek megjelenése, és az adatismétlések visszaszorítását célző fejlesztői elvárások. Ez az út vezetett az adatbázisok létrejöttéhez.

Az adatok fizikai tárolása továbbra is adattárakban történt, de az adatok fizikai tárolását, visszakeresését, módosítását és törlését az ezekre az eljárásokra optimalizált adatbáziskezelő rendszer vette át. A felhasználói programok ezután már függetlenné váltak az adatoktól, és nagy mértékben megszűntek az adatismétlések is (az adatismétlések oka arra vezethető vissza, hogy az adattárak eredetileg a feldolgozásokhoz/programokhoz lettek kialakítva). Az adatátfedések a relációs adatbázisok esetében, a szervezésükből, kialakításukból adódóan teljes mértékben nem szüntethetők meg.

Másrészről egyre szaporodott azoknak a szakembereknek a száma, akik azt vallották, hogy egy információs rendszer (alkalmazás, szoftver) legmaradandóbb, legkevésbé változó része (tényezője) az adathalmaz, amely a kapcsolódó valós rendszer lényegét leírja. Egy jól

megtervezett adabázis (bizonyos keretek között) a felhasználók új igényeit is képes kielégíteni. Éppen ezért ajánlatos az információs rendszerek fejlesztésénél az adatbázistervezéssel kezdeni a munkát.

Az adatbázis-elv lehetővé teszi az adatok között fenálló természetes kapcsolatok ábrázolását, amely kapcsolataokat az adatbázisban is rögzíteni kell valamilyen módon (adatokkal, természetesen), és ez bizonyosan adatisméltésekhez (relációs adatbázisok), vagy egyéb metaadatok rögzítéséből származó redundanciához vezet.

Az optimális adatátfedések kialakítására adatmodellezési eljárást dolgoztak ki (a múlt sazázad hetvenes éveiben), amelyben (többek között) Bachmann, Codd és Halassy munkái/erőfeszítései a legmarkánsabbak.

Az adatrendszer egymagában is igen bonyolult ahhoz, hogy valakinek csak úgy, kedvtelésből, késztetése legyen a kialakítására, viszont az információs rendszer helyes és gyors működéséhez. Az információs rendszer kialakításához, működéséhez azonban nem csak adatrendszerre/adatbázisra van szükség, no de vagyük sorjába a dolgokat.

2.4. SZÁMÍTÁSTECHNIKA, INFORMATIKA ÉS INFORMÁCIÓS RENDSZER

A számítógépek alkalmazása az élet minden területén észlelhető. Egyes területeken a számítógépek elengedhetetlen "kellékké" váltak, ami természetes is. Másrészről azonban divatossá vált már egy ideje számítógéptulajdonosnak lenni (mint ahogyan manapság okostelefont birtokolni, még akkor is, ha nálunk a lehetőségeit kiaknázó - mondjuk a 4G - szolgáltatások még kilátásban sincsenek): akinek nincs számítógépe az maradi. Ez eddig rendjén is volna, a baj csak az, hogy a gépvásárlástól kezdve minden tulajdonos informatikusnak képzeli magát. Érteni a gép kezeléséhez és informatikusnak lenni két nagyon különböző dolog. A szövegszerkesztő, levelező- vagy rajzprogram használatának ismerete még senkit sem informatikussá. Az információs rendszerek esetében keveredik a cél, az eszköz és a módszer fogalma: ha egy alkalmazás (információs rendszer) lassan vagy rosszul működik, általában gépparkot újítanak, szoftvert frissítenek ahelyett, hogy az alkalmazásba "néznének bele", vagyis ellenőriznék a feladat megoldását (a hiba ugyanis legtöbbször ott lapul).

Fontos rámutatni, hogy a számítógépet és a rajta futó szoftvereket ügyesen kezelő személy nem tekinthető magától értettődően egyúttal és egyidőben informatikusnak is. Ennek csupán egyetlen, de nyomós oka van: a gondok az informatikában általában (pontosabban főleg) nem technikai jellegűek (nem hardverrel vagy szoftverrel – adatbáziskezelőkkel illetve

alkalmazásfejlesztőkkel kapcsolatosak). A számítógép használata nem mindig kapcsolatos az informatikával. Az informatikában a problémát az információs rendszerek bonyolultsága és tényezőinek nagy száma jelenti.

Az informatika fogalmát többféleképpen is megfogalmazták. A sokféle definícióból kettőt ismerünk csak meg: egy lakonikust (rövidet) és egy hosszabbat, átfogóbbat. Az utóbbi szerint:

"Az informatika az információ áramlásának különböző módozataival, feldolgozásának és hasznosításának módszereivel, a termelékenységre és hatékonyságra gyakorolt hatásaival, megfigyelési és ellenőrzési célokra való felhasználásával és végezetül a társadalmi-gazdasági fejlődést és a társadalmat alakító szerepével foglalkozik"[6]. Az informatika fogalma a fenti definíció szerint nagyon általános, hiszen felöleli az adatok rendszerbe fogását, a feldolgozási eszközöket, az adatfeldolgozás módozatait az ismeret születésétől a végfelhasználóhoz való eljuttatásáig, illetve a jelentésekbe foglalt adatok, információk hatásvizsgálatát a gazdasági szubjektumokra. Az előbbiek szerint az informatika egy olyan átfedő (interdiszciplináris) tudomány, amely felöleli a hírközlés (kommunikáció) valamennyi formáját, a számítástechnikát és annak alkalmazását, a mikroelektronika és robotika eredményeinek gyakorlati alkalmazás-lehetőségeit, az irányítás és vezérlés gyakorlatának bizonyos területeit, illetve mindezek érvényesítését (és érvényesítésének lehetőségeit) a gazdaságban, a természet- és társadalomtudományokban. Az informatika ebben az értelmezésben számunkra túlságosan sokrétű fogalmat takar.

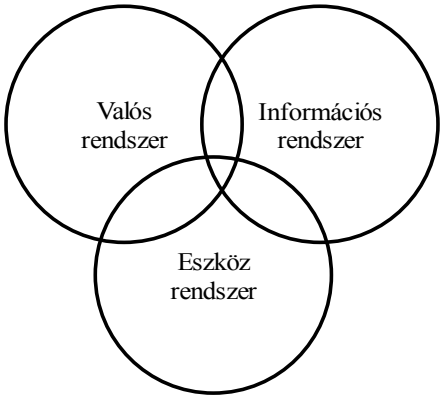
A rövidebb meghatározás szerint, "**Az informatika az ismeretek megismerésének, azok célszerű elrendezésének és kezelésének a tudománya. Az informatikus az a szakember, aki ebben a tudományban jártas**"[7].

A valós világ megismerése, ábrázolása és az azt támogató információs rendszer megvalósítása közben az informatikus mindig a következő három rendszert és azok összefüggéseit kell, hogy kövesse:

1. az objektív valóságot (valós rendszer – VR),
2. a róla kialakított ismereteket (információs rendszer – IR) és

[6] A budapesti Központi Statisztikai hivatal meghatározása, forrás: [20.Noszkay-GDF-1994], 8. Oldal.
[7] Forrás: [11.Halassy-E-I-R], 35. oldal

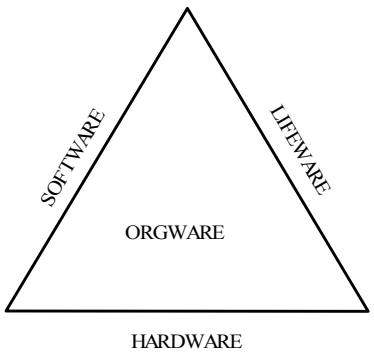
3. az ismeretek kezelését biztosító rendszert (amely a hardvert, a szoftvert, az élőerőt – lifewaret és a szervezést – az orgvert, egyszóval az eszközrendszert – ER jelöli).



4.1. ábra

Itt három, részben átfedő rendszerről beszélhetünk (4.1. ábra[8]), amelyek az átfedések mellett aránylag nagy függetlenséggel is rendelkeznek. A három rendszer harmonikus egységének a megteremtése a végleges cél, amit manapság egyre inkább figyelmen kívül hagynak. A mai átlagember az eszközök bűvöletében él: a gyors számítógépek és a legújabb szoftverek jelentik számára a végcélt.

Az eszközrendszer grafikusan is ábrázolható[9], ahogy ezt a 4.2. ábra is mutatja. Az eszközrendszer négy részrendszerének szoros összetartozására utal (vagy legalábbis szeretne utalni) a 4.2. ábra eredeti angol kifejezésekkel és az összetartozást szimbolizáló zárt vonal alkalmazásával.



4.2. ábra

A **hardvert** (hardware) a rendszer megfogható, anyagi eszközei alkotják: a számítógép(ek) az összes tartozékaikkal, bemeneti, bemeneti-kimeneti és kimeneti egységeikkel,

[8] Forrás: [11.Halassy_E-I-R], 36. oldal
[9] Forrás [6.Djordjevic-Vodoprivredni]

a vezetékes és vezeték nélküli hálózat megvalósításának berendezéseivel, az elsődleges adatgyűjtést, -rögzítést, előfeldolgozást és –elosztást végző egységekkel. A hardver rohamos léptekkel fejlődik, alkalmazásaink nem tudnak lépést tartani az ilyen mérvű fejlődéssel, és a (hardver) lehetőségeknek csupán néhány százalékát használják, bár a szoftverek újabbnál újabb változatai egyre igényesebben a hardverrel szemben (sokak szerint a hardver csupán a szoftver-követelmények miatt fejlődik, mások szerint a helyzet pontosan fordított). Általános becslések szerint a hardver fejlettségi foka tíz-tizenöt évvel megelőzi a szoftverét.

A **szoftver** a (rendszer szintű) program-megoldásokat jelöli: az operációs rendszereket, a különböző kiszolgáló-karbantartó programokat, a programnyelveket, adatbáziskezelő rendszereket, szövegszerkesztőket, táblázatkezelőket, rajzprogramokat, kép- hang- és filmfeldolgozó valamint más általános célú (nagyobb embercsoport számára érdekes) problémát érintő alkalmazásokat. A szoftverfejlődés talán látványosabb volt az utóbbi években a hardverénél, de egyesek még a fent említett lemaradást is kevésnek taksálják, és húsz évet emlegetnek. A mai adatbáziskezelő rendszerek, a szoftverfejlődést igazolandó, olyan bonyolultak, hogy a laikusok talán csak elindítani tudják őket, de érdemlegesen használni nem, a problémakörhöz jól értő szakemberek (rendszerelemzők, rendszertervezők) pedig lehetőségeiknek csak néhány százalékat tudják alkalmazni.

Az információs rendszerek megvalósítását végző szakemberek alkotják az élő erőt, a **lifewaret**. A mai informatikai káderek a számítástechnikában talán jobban képzettek mint kellene (ami természetesen nem baj), viszont az információs rendszerek természetét, tényezőit, tényezőinek számát, jellegét és összefüggéseit nem ismerik eléggé (ami azonban hiba).

Az eszközrendszer negyedik részrendszere, az **orgver** az előző három eszköz-rendszerrész struktúrába szervezését valósítja meg. Az eszközrendszer struktúrájának kialakítása nem önös érdekből történik, hanem azzal a céllal, hogy az információs rendszer a megrendelő valamennyi elvárását maradéktalanul teljesíteni tudja. Az orgver az információs rendszer fejlesztéséért és működtetéséért felelős a szervezés oldaláról.

Az információs rendszer kifogástalan működésének szempontjából az eszközrendszer négy rendszerrészének működési és egyéb szempontokból is harmonizálniuk kell. Bármely rendszerrész elhanyagolása vagy nem megfelelő működése (együttműködése) a maradék három rész bármelyikével az információs rendszer működésében előbb-utóbb komoly zavart idéz elő.

Az információs rendszer, ahogy a neve is elárulja, maga is rendszer, tehát rá is érvényesek a rendszerelméletben megállapított és bizonyított törvényszerűségek. Itt a megfelelő pillanat, hogy rámutassunk az információs rendszer és az információk rendszere (információ-rendszer) fogalmak közötti különbségre. Az információk rendszere csak az adatok (információk)

elrendezésére, szervezésére szorítkozik, az információs rendszer ennél sokkal többet takar. Az információs rendszerek és fejlesztésük megismerésének első lépéseként az információs rendszer fogalmával ismerkedünk meg.

»Az információs rendszer

1. adatoknak (információknak);
2. a velük kapcsolatos információs eseményeknek;
3. a rajtuk végrehajtott információs tevékenységeknek;
4. az előzőekkel kapcsolatos erőforrásoknak;
5. az információk felhasználóinak;
6. ill. A fentieket szabályozó szabványoknak és eljárásoknak

szervezett együttese.«[10]

A fenti definíció egyes tényezőinek alaposabb jellemzése persze elkerülhetetlen, de lássuk előbb magát a meghatározást, úgy egészében véve. Az első megállapítás a fenti definíció kapcsán az, hogy az információs rendszerek az inhomogén rendszerek családjába tartoznak. Ezt igazolja a tényezők és kapcsolataik fajtáinak, jellegeinek különbözősége. A felhasználók, adatok, standardok, események, tevékenységek, hardver és szoftver különbözőségét, gondolom nem szükséges feltétlenül bizonygatni.

Az információs rendszer kísérőjelzői a mindennapi beszédben éppúgy, mint a szakirodalomban gyakran tévesek, túlzók vagy ismételgetéseket (tautológiát) rejtenek. Gyakran használják a komplex és integrált jelzőket, sőt kapcsoltan is mindkét jelzőt, valahogy így: »komplex, integrált információs rendszer«. Az inhomogén rendszerek (amelyek közé az információs rendszerek is tartoznak) már eleve (»ab ovo« vagy a priori) komplexek, tehát nem kell ezt még a jelzővel is hangsúlyozni (tautológia). Az integrált jelző ugyancsak tautológiát rejt, hiszen minden rendszer a tényezők szervezett – integrált együttese (már a fenti definíció szerint is). A »vezetői« jelző használata járatlanságra, butaságra vall, bár divatos volt és ma is az még a vezetői információs rendszer (MIS – Management Information System) kifejezés. Vezetői információs rendszer, mint olyan ami a nevéből kikövetkeztethető lenne, magában nem is létezik, vagy ha igen, hát nem sok értelme van. A vezetők (menedzserek) számára szükséges információk mindenképpen az alapinformációkból származnak, és egyfajta kivonatként (információ-kivonatként) foghatjuk fel őket. A döntésekhez elengedhetetlen információk (adatok) ugyanis mindenképpen megtalálhatók egy „mindent átfogó, komplex, integrált információs rendszer” rendszerrészeiben. Nevezzük azonban ezt a „mindent átfogó, komplex, integrált információs rendszer”-t egyszerűen csak információs rendszernek.

[10] Forrás: [11.Halassy-E-I-R], 43. oldal.

3. RELÁCIÓS ADATMODELL

A relációs adatmodell a hetvenes években alakult ki, Edward Codd 1970-ben megjelent iránymutató, és az elméleti alapokat lefektető mukája alapján. A relációs adatmodell fejlődési útvonalát olyan ismert kutatók és szakemberek nevei fémjelezik mint Fagin, Ullmann, Rissanen, Beeri és Bernstein (sok más, ismert és elismert szakember mellett). Az addig a gyakorlatban is használt két legismertebb adatmodell (a hierarchikus és hálós) gyengeségei képezték az új (relációs) adatmodell fejlesztési célkitűzéseit (Mogin):

1. a logikai és fizikai adatstruktúra szétválasztása
2. egszerű felépítés és
3. leíró (deklaratív, nemprocedurális) nyelv kidolgozása az adatkezelés biztosítására.

A logikai és fizikai adatstruktúra szétválasztásával kapcsolatos kutatások az adatok logikai és fizikai függetlenségének elveihez vezettek. Ezek az elvek beépültek a relációs adatmodellbe, és a minőségi újítás mellett lehetővé tették a programoktól és hardver-környezettől független relációs adatbázisok megvalósítását.

A felépítés egyszerűségét a relációs adatmodell alappillére, a táblázattal is szemléltethető reláció biztosítja. Az egyes relációkban (táblázatokban) elhelyezkedő adatok közötti kapcsolatokat kulcs-külső kulcs mechanizmussal (adatismétléssel-az egyik reláció kulcsa a másik relációban külső kulcsként jelenik meg) vagy új relációval (lényegében ez is adatismétlés, csak itt az új relációba a két, kapcsolódó adatokat tartalmazó reláció kulcsai jelennek meg) valósítják meg.

A deklaratív lekérdező nyelv fejlesztéséhez a relációs algebra szolgált alapul, további új, a relációs adatmodell igényeit kielégítő halmazműveletek megfogalmazásával, bevezetésével. A relációs algebra műveletei leírhatók egy relációs operátorokat tartalmazó logikai kifejezéssel. A relációs algebrára épülő deklarativ SQL (Structure Querry Language) nyelvet elsődlegesen az adatleírási és adatkarbantartási utasítások, valamint az adatok lekérdezésére alakították ki. Az adatok feldolgozása továbbra is a procedurális nyelvek segítségével történik, amelyekbe az SQL már régóta be van építve. Mint minden teljes adatmodellnek, a relációsnak is három jól elkülöníthető része, komponense van, amelyekről a továbbiakban lesz szó.

3.1. SZERKEZETI/STRUKTURÁLIS KOMPONENS

A relációs model szerkezeti része rámutat a relációs model alapelemeire (primitivni koncept), és a belőlük épülő egyéb (mind összetettebb) szerkezetekre, strutúrákra. Tartalmi/logikai szinten (intenzió) a következő elemeket különböztetjük meg: tulajdonság (obeležje), relációséma (šema relacije) vagy egyszerűen csak reláció és adatbázisséma (šema baze podataka). Konkrét/előfordulási/fizikai szinten pedig a következőket: tulajdonságérték/értékhalmaz (domen obeležja), értéksor-előfordulások/táblázat (torka/tabela) adatbázis-előfordulás (pojava baze podataka). A relációs modellnek mindössze két alapeleme van: a tulajdonság (obeležje) és a tulajdonságérték/értékhalmaz (domen obeležja). A többi elemet matematikai szabályok szerint alakították ki.

A **relációt** a matematikában halmazok szigorúan meghatározott sorrendben értelmezett keresztszorzatának (Descartes-szorzatának, Cartesian product) részhalmazaként definiálják. Matematiai megfogalmazással: legyenek B_i, i=1, 2, …, n a halmazok, akkor a halmazokon értelmezett reláció $R \subseteq B_1 \times B_2 \times \ldots \times B_n$ kifejezéssel adott. Mivel esetünkben a halmazok tulajdonságértékek halmazait jelentik, az A_i, i=1, 2, …, n tulajdonságok/attribútumok (az A az angol atribut/attribute szóból-tulajdonságok ered) értékhalmazait, a reláció leírható a következő formában is: $R \subseteq dom(A_1) \times dom(A_2) \times \ldots \times dom(A_k)$, ahol R a reláció neve. A reláció előfordulásai/konkretizációi az az életben is előforduló, érvényes n-esek vagy értéksor-előfordulások, amelyek egymás alá írva táblázatot alkotnak. Az így kapott táblázatokat a reláció nevével illetik (jelölik), táblázatok fejléce a konkrét tulajdonságokat tartalmazza, a sorok pedig a relációelőfordulásokat (n-eseket vagy értéksor-előfordulásokat).

Példa:

Név={Tesla, Chopin, Newton},

Hivatás={tudós, művész}

Név× Hivatás={[Tesla, tudós], [Tesla, művész], [Chopin, tudós], [Chopin, művész], [Newton, tudós], [Newton, művész]}

Híres emberek$\subseteq$ Név× Hivatás

Híres emberek={[Tesla, tudós], [Chopin, művész], [Newton, tudós]}

A **relációséma** az n-esek (relációelőfordulások vagy értéksor-előfordulások) intenzió szintű leírása/képe, amely a következő alakot ölti: R(A, M), ahol R a relációséma (reláció) neve, A a tulajdonságok/attribútumok halmaza, az M pedig a megszorítások/korlátok halmaza. A relációséma egy adatokkal leíni kívánt lényeg (egyedtípus, entitás) szerkezeti (strukturális) felépítését ábrázolja. A relációséma egy-egy előfordulása (relációelőfordulás, a tábla egy-egy

sora) így egy konkrét egyedtípust képvisel (a tulajdonságtípusainak/tulajdonságainak konkrét értékeinek felsorolásával. Minden relációelőfordulás (táblasor) értékeire az adatbáziskezelő rendszer érvényesíti az M megszorításhalmazt (ellenőrzi a korlátokat). A megszorításokról átfogóbban majd a későbbiekben lesz szó, most csak a relációséma kulcsa kerül bemutatásra mint megszorítás.

A relációséma kulcsa olyan tulajdonság/attribútum vagy tulajdonsághalmaz/attribútumhalmaz, amelynek két elvárást kell teljesítenie:

1. egyrészről egyedinek kell lennie (értéke nem ismétlődhet a tábla soraiban, így egyértelműen különbözni fognak a tábla sorai/relációelőfordulások),
2. másrészt a kulcsnak minimálisnak kell lennie (semmilyen részhalmaza nem töltheti be a kulcs szerepét) a relációban.

Amennyiben egy attribútumhalmaz csak az első feltételt teljseíti, a másodikat viszont nem, akkor az szuperkulcsnak (a kulcsnál bővebb halmazok megnevezésére szogáló kifejezés – kulcsok szuperhalmaza) tekintendő.

A relációsémának elvben több, akár egyformán jó kulcsa (kulcsjelöltje) is lehet, ilyenkor tetszőlegesen választható az egyik elsődleges (primáris) kulcsnak (a többi alternatív kulccsá válik). A kulcs lehet egyszerű (egy tulajdonság/attribútum a kulcs), összetett (több saját attribútum vagy több külső/idegen kulcs alkotja) vagy hierarchikus (legalább egy-egy vagy több- idegen/kulső kulcs és legalább egy-egy vagy több- saját tulajdonság képezi).

Az **adatbázisséma** a relációsémához hasonló, de tőle általánosabb fogalom hiszen relációsémákból és megszorításokból (többek között relációsémák közötti korlátozásokból is, ami a relációsémára nem volt elmondható) áll: AS(RS, AM), ahol AS az adatbázisséma neve, az RS a relációsémák halmaza, az AM pedig az adatbázisséma szintjén megfogalmazott korlátok halmaza. Az adatbáziskezelő rendszer a relációelőfordulásoknál az M megszorításhalmaz érvényesítése mellett ellenőrzi az AM korláthalmazban megfogalmazott megszorításokat is.

3.2. MEGSZORÍTÁSI/INTEGRITÁSI KOMPONENS

A relációs adatmodellben sokfajta megszorítást fogalmaztak meg, és azt többféleképpen is csoportosították. A megszorítások csak megadásuk pillanatától fejtik ki hatásukat (ellenőrzik az utasításokat, amelyek megsérthetik őket), visszamenőleges hatásuk nincs. Egyes utasítássorozatok bizonyos utasításai átmenetileg a megszorítások átmeneti sérülését válthatják ki, ez az állapot késleltetett ellenőrzés igénylésével oldható fel. A következő korltátcsoportosítás a (Lazarevic et al) forrásból való, és a következőképpen ötvözi két csoportba a megszorításokat:

1. A relációs modellre vonatkozó megszorítások
    1. kulcsmegszorítás (integritet ključa – az előző pontban ismertetve)
    2. hivatkozásiépség megszorítás (referencijalni integritet)
2. Üzletszabályzati megszorítások (poslovna pravila integriteta)
    1. doméjnértékekre vonatkozó megszorítások (pravila integriteta za domene)
    2. attribútumértékekre vonatkozó megszorítások (pravila integriteta za atribute)
    3. relációra vonatkozó megszorítások (pravila integriteta za relacije)
    4. adatbázisra vonatkozó megszorítások (pravila integriteta za bazu podataka)
    5. átmenetekre/értékváltozásokra vonatkozó megszorítások (pravila integriteta prelaza)

1.2. Hivatkozásiépség megszorítás. Ez a megszorítás azt írja elő, hogy adott reláció attribútumaiban (ezeket küslő vagy idegen kulcsnak nevezik) csak olyan értékek jelenhetnek meg, amelyek egy megjelölt másik relációban elsődleges kulcsértékek (esetleg NULL értékek lehetnek még az idegen kulcs értékei). Az idegen kulcsot tartalmazó reláció táblájába (hivatkozó tábla) történő beírás (beszúrás) vagy adatmódosítás vezethet ennek a megszorításnak a megsértéséhez (ha a vele kapcsolatos elsődleges kulcsot tartalmazó – hivatkozott tábla – nem tartalmazza a hivatkozó táblába beírni/módosítani kívánt értéket – ekkor a megszorítás megsértését kiváltó művelet MINDEN ESTEBEN visszautasításra, megtagadásra kerül, és nem hajtódik végre). A megszorítás megsértését kiválthatja a hivatkozott táblában történő módosítás vagy törlés is olyan sorokon, amelyekre voltak hivatkozások a hivatkozó táblában. Ilyen esetekben a hivatkozásiépség megszorítást megelőzendő akciók (operációk), amelyeket a hivatkozótábla létrehozásakor lehet a hivatkozott táblában való törlés és módosítás eseteire megfogalmazni/előírni, a következők lehetnek (az első három az általánosan elfogadott):

1. RESTRICT – a hivatkozásiépséget sértő művelet visszautasítása/megtagadása
2. CASCADE – tovagyűrűző eljárás, amely a hivatkozott táblában kért módosítást automatikusan elvégzi a hivatkozó tábla soraiban is
3. SET NULL – a hivatkozott táblában történő módosítás/törlés esetén a hivatkozó táblában az idegen kulcs értékét NULL-ra váltja
4. SET DEFAULT – a hivatkozott táblában történő módosítás/törlés esetén a hivatkozó táblában az idegen kulcs értékét egy alapértelmezett értékre váltja
5. NO ACTION – a kért művelet végrehajtódik, az adatbázis SÉRÜL!

2.1. Doméjnértékekre vonatkozó megszorítások. A doméjnek olyan értékhalmazok amelyekből az attribútumok az értékeiket meríthetik. Kétfajta doméjntípus létezik:

1. beépített doméjnek – adattípusok, amelyek rendelkezésre állnak az adatbáziskezelő rendszerben

2. szemantius doméjnek – a felhasználó által, a már létező doméjnek alapján létrehozott új doméjnek.

2.2. Attribútumértékekre vonatkozó megszorítások. Az attribútumértékek a típusukkal és elfogadható értékhatáraikkal vagy értékhalmazaikkal korlátozhatók, és a következő módon fogalmazhatók meg:

<attribútumnév, doméjnnév, megszorítás, akció**>**

A beépített doméjnek nevei adattípusokat jelölnek, de a leírásban szerepelhetnek szemantikus doméjnek is. A megszorítás egy logikai kifejezés, amelyben temészetesen szerepel az attribútumnév is. Amikor ez a megszosítás sérül, az utasítás végrehajtását az adatbáziskezelő rendszer megtagadja, ezért az akció feltüntetése általában kimarad ennek a megszorításának a leírásából.

A NULL érték egy sajátságos attribútumérték, amelynek semmi köze sincs a 0 számhoz (ez az érték az adatbáziskezelő rendszerek legtöbbjénél az attribútummezőben elhelyezhető legnagyobb számértéket jelenti), és a következő három esetre értelmezik:

1. visszatartott érték – jogosulatlanok számára a titkosított értékekre való rákérdezéskor NULL értéket ad vissza az adatbáziskezelő rendszer
2. alkalmazhatatlan érték – az attribútumnak a beírás pillanatában bizonyosan nincs még értéke (signifikantna nula) – pl. az »Első szülés dátuma« attribútum értéke gyermektelen nőknél,
3. ismeretlen érték – az attribútumnak a beírás pillanatában bizonyosan van értéke, csak nem ismert (nesignifikantna nula) – pl. a hallgató az első íratkozáskor nem hozta magával a személyi igazolványát, így a »Személyi igazolvány száma« attribútum értéke NULL lesz, és
4. értelmezhetetlen érték (neprimenjeno svojstvo) – pl. a »Leánykori név« attribútum értéke férfiak esetében.

2.3. Relációra vonatkozó megszorítások. Itt az attribútumértékek egymáshoz fűződő (érték)viszonyát korlátozhatjuk le egy relácón belül.

2.4. Adatbázisra vonatkozó megszorítások. Ezek a megszorítások formailag (szintaktikailag) megegyeznek a relációkra vonatkozó megszorításokkal (a példák a megszorításokra az SQL részben következnek) azzal a különbséggel, hogy itt tetszőleges összetettségű attribútumérték-összefüggések adhatók meg az adatbázis valamennyi relációjának attribútumértékei között.

2.5. Átmenetekre/értékváltozásokra vonatkozó megszorítások. Az eddig leírt megszorítások logikai kifejezései az egyes relációból származó attribútumokat tratalmazták. Megfelelő közös megegyezéssel (jelölje például vesszős érték az operáció előtti attribútumértéket, vesszőtlen pedig az operáció utánikat) felírhatók megszorítások az értékváltozásokra is egy adott reláció keretein belül. Példa – az Informatika szakra íratkozott hallgató nem válthat Villamosságtan szakra:

CREATE INTEGRITY RULE Hallgatók szakváltása

∀ Hallgató, Hallgató' (IF Hallgató'.Szak=”Informatika” THEN Hallgató.Szak ≠ “Villamosságtan”);

3.3.MŰVELETI/OPERÁCIÓS KOMPONENS

A szerkezeti és megszorítási részek a relációs modell statikus állapotát írják le. A műveleti rész a modell dinamikájáról szól, és az adatkezelési utasítások alapjaként szolgál. Ezek az utasítások képviselik a relációs modell adatkezelési nyelvét, amelyet kialakításakor elsősorban az adatok lekérdezésére, illetve adatkezelésre szántak. A nyelv alapját a relációs algebra képviseli, amely a relációkon végzett műveletek eredményeként szintén relációt ad. A műveletek így tetszőlegesen egymásba ágyazhatók, aminek köszönve bonyolult elvárások is megfogalmazhatók.

3.3.1. A relációs algebra

A relációs algebra egy és kétváltozós műveletekkel rendelkezik, az operandusok pedig relációk. A relációs algebra műveleteit is (a megszorításokhoz hasonlóan) többféleképpen csoportosítják. A **hagyományos relációs algebrai műveletek** négy csoportja a következő (Ullman-Alapvetés):

1. relációkra értelmezett halmazműveletek,
2. reláció részeit eltávolító műveletek,

3. két relációt összekapcsoló műveletek
4. relációséma elemeinek átnevezése

1. Relációkra értelmezett halmazműveletek Mivel a relációk is halmazok, értelmezhetők rájuk az alapvető (kétváltozós) halmazműveletek: az unió (egyesítés), metszet és a különbség (csak azonos attribútumokkal rendelkező relációkra értelmezett).

Únió/egyesítés esetén (jelölése $R_1(A) \cup R_2(A)$) eredményként olyan relációt kapunk, amelyben ismétlődésmentesen minden n-es (minden t sor) megtalálható akár csak az egyik, akár csak a másik vagy akár mindkét relációban is előfordul. Formálisan:

$$R_1 \cup R_2 = \{t \mid t \in R_1 \vee t \in R_2\}$$

Metszet esetén (jelölése $R_1(A) \cap R_2(A)$) az eredményreláció csak azokat az n-eseket (azon t sorokat) tartalmazza amelyek mindkét operandus-relációban megtalálhatók. Az ezt leíró kifejezés a következő alakú:

$$R_1 \cap R_2 = \{t \mid t \in R_1 \wedge t \in R_2\}$$

Különbség esetén (jelölésben $R_1(A) - R_2(A)$) az eredményrelációban $R_1(A)$ azon n-esei (t sorai) találhatók, amelyek nincsenek benne $R_2(A)$-ban. Relációs algebrai kifejezéssel:

$$R_1 - R_2 = \{t \mid t \in R_1 \wedge t \notin R_2\}$$

2. Reláció részeit eltávolító műveletek. Az ide tartozó két unáris/egyváltozós művelet a projekció/vetítés és a szelekció/kiválasztás.

A **projekció/vetítés** adott reláció oszlopainak kiválasztását teszi lehetővé az eredményrelációba. Az R reláció X attribútumhalmazra való vetítettje a következő kifejezéssel írható le (t itt most az oszlopneveket/attribútumokat jelöli):

$\pi_X(R) = \{t[X] \mid t \in R\}$.

A **szelekció/kiválasztás** folyamán olyan n-esek választhatók ki egy adott relációból az eredményrelációba, amelyek megfelelnek egy megfogalmazott megszorításnak. A következő kifejezés azokat az n-eseket (t sorokat választja ki az R relációból, amelyek megfelelnek az F megszorításnak:

$\sigma_F(R) = \{t \mid t \in R \wedge t$ kielégíti F-et$\}$.

3. Két relációt összekapcsoló műveletek. Ezek a teljes összekapcsolás (keresztszorzat, Descartes szorzat), feltételes összekapcsolás (théta összekapcsolás) és természetes összekapcsolás (natural join)

Általánosságban az összekapcsolásoknál a következő algoritmus szerint kell eljárni:

1. a keresztszorzat (Descartes szorzat) kiszámítása
2. azon n-esek (sorok) kiválasztása a keresztszorzatból, amelyek megfelelnek a feltüntetett megszorításnak, és
3. a párban egybevágó attribútumok (oszlopok) közül az egyiknek a megtartása (csak a természetes összekapcsolásra érvényes)

A **teljes összekapcsolás (Descartes szorzat)** esetén az előző algoritmus első lépése kerül végrehajtásra: az egyik reláció minden sorát összepárosítjuk (folytatjuk) a másik reláció minden sorával, és ez kerül az eredménytáblába. Jelölése: $R_1 x R_2$..

A **théta összekapcsolás** feltételhez kötött összekapcsolás, amely a fenti algoritmus első két lépésének elvégzésekor keletkezik eredményrelációként. Az első lépésben előállított keresztszorzatból az eredménytáblába csak az adott Θ feltételnek (a Θ feltételben, amely összetett is lehet az attribútumok mellett a következő logikai operátorok szerepelhetnek: {=, NOT=, >, ≥, < , ≤}) megfelelő sorok kerülnek. A jelölése: $R_1 \bowtie_\Theta R_2$ .Amennyiben az egyszerű feltételben a logikai operátor „=“ az összekapcsolást equijoin-nak nevezik.

A **természetes összekapcsolás** az equijointól csak abban különbözik, hogy az eredménytábla a páronként azonos értékeket tartalmazó oszlopokból csak egyet-egyet tartalmaz (az előző algoritmus harmadik lépése is elvégzésre kerül). A jelölése: $R_1 \bowtie R_2$.

4. Relációséma elemeinek átnevezése. Sükség adódhat a relációséma elemeinek megváltoztatására, legyen szó akár az attribútumnevekről, akár a relációnévről. Az R reláció átnevezése attribútumneveivel együtt jelölésben a következő: $\rho_{S(A1,\ A2,\ ...An)}(R)$. Csak relációnév változtatására a ρ_S (R) kifejezés szolgál.

3.3.2. A relációs algebra kiterjesztése

A relációs algebra újabb műveleteit az adatbáziskezelési alkalmazásban megfogalmazott igények váltották ki, és a gyakorlatban sűrűn használatosak.

Ismétlődések kiiktatása. A művelet az egybevágó n-esek (duplikált sorok) megkeresését és kiiktatását végzi, a duplikált sorokból egy sor megtartásával. Jelölése: δ(R).

Összesítő (aggregációs) műveletek. Az alábbiakban felsorolt összesítő műveletek attríbútumokra értelmezettek tábla szinten, de csoportképzésekkel is párosulhatnak, és olyankor az összesítő műveletek nem a teljes táblára, hanem a csoportokra hajtódnak végre.

Összesítő művelet	Értelmezés
COUNT(A)	Az A oszlopban lévő adattal rendelkező (NOT NULL) sorok számát adja
COUNT(*)	A tábla sorainak számát biztosítja
SUM(A)	Az A numerikus oszlopban lévő értékek összegét adja
AVG(A)	Az A numerikus oszlopban található értékek átlagát számítja ki
MIN(A)	Az A sor legkisebb értékét (A numerikus), illetve a használt abc szerinti első értéket (A szöveges) keresi ki
MAX(A)	Az A sor legnagyobb értékét (A numerikus), illetve a használt abc szerinti utolsó értéket (A szöveges) keresi ki

Csoportképzés. Ez a művelet a tábla sorait csoportosítja egy vagy több oszlopban található értékek alapján valamelyik összesítő művelet szerint. A csoportképző művelet operátorjele γ, operandusai (indexe) egy L listát alkotnak. Az L listában csoportosító attribútumok és összesített attribútumok vannak. Jelölésben: $\gamma_L(R)$. A jelölés szemantikája (értelmezése) a következő:

1. Lérejönnek a csoportok R soraiból úgy, hogy az egyes csoportokban a csoportosító attribútumok azonos értéket tartalmaznak (ha nincs csoportosító attribútum feltüntetve, akkor az egész tábla egy csoportnak tekinthető).
2. minden csoporthoz egy sor tartozik az eredménytáblában (eredményrelációban), amelnek a felépítése a következő:
    a. a csoportosító attribútumok adott csoporthoz illeszkedő értékei,
    b. a csoporthoz tartozó, az L listában felsorolt összesített attribútumokhoz tartozó összesítő műveletek eredményei.

Rendezés. A tábla (reláció) sorait egy vagy több attribútum értékei szerint sorrendbe rakja, listaként. A tábla (listaként) rendezett alakja sokkal hatékonyabb lekérdezés-végrehajtást eredményez, mint a rendezetlen (halmazként, multihalmazként használt) reláció (tábla). Jelölése: $\tau_L(R)$, ahol L az attribútumok listája, amely szerint a rendezés végbemegy. A lista elsődlegesen az első attribútum szerint lesz rendezve, az első attribútum azonos értékeit tartalmazó sorok az L-ben második attribútumként megjelölt attribútumérték szerint kerül sorbaállításra, és így tovább a következő L-ben szereplő attribútumra nézve.

A kibővített vetítés/projekció. A kibővített vetítés az egyszerű oszlopkiválasztás mellett új, egy vagy több attribútumot tartalmazó aritmetikai kifejezés eredményeként létrehozott oszlopot is biztosítani tud. Jelölésben: $\pi_L(R)$, ahol most az L a vetítési lista, amelynek elemei a következők lehetnek:

1. A_i - R attribútumai közül bármelyik (akár több is) – ez az egyszerű vetítés,
2. Egy $A_1 \rightarrow B_1$ kifejezés, amely argumentumként attribútumokat tartalmaz, és egy adott oszlop (A_1) egyszerű átnevezését jelöli (B_1 az új név)
3. Egy $E \rightarrow B_1$ kifejezés, ahol E egy attribútumokat, aritmetikai, karakterlánc-műveleteket és konstansokat tartalmazó kifejezés, a B_1 pedig az új, számított attribútum neve.
4. Konstans érték. Ha karakteres konstans, akkor aposztrofok között kell feltüntetni, ha számérték, akkor anélkül.

Külső összekapcsolások. A külső összekapcsolások a lógó, nem kapcsolható sorokat is beveszik az eredménytáblába, amit a szokványos összekapcsoló műveletek nem tesznek meg. Az ilyen nem kapcsolható sosroknál ugyanis nincs értékegyezés a közös oszlopokon (attribútumokon). A **külső természetes összekapcsolások** három féle változata használatos a gyakorlatban: a baloldali (jelölése: $R_1 |\bowtie R_2$), a jobboldali (jelölése: $R_1 \bowtie| R2$) és a teljes (jelölése: $R_1 |\bowtie| R_2$) külső összekapcsolás. Mindhárom fajta külső összekapcsolás első lépése a hagyományos természetes összekapcsolás. A baloldali külső összekapcsolásnál második lépésként a baloldali argumentum, az R_1 nem kapcsolható sorai kerülnek az eredményrelációba (NULL értékekkel kitöltve az eredménytábla többi oszlopát). A jobboldali külső összekapcsolásnál ugyanaz történik meg a második lépésben mint az előző esetben, csak itt most a jobboldali táblából, az R_2-ből kerülnek az eredménytáblába a nem kapcsolható sorok NULL értékekkel az eredménytábla sorainak többi attribútumaira nézve. A természetes külső összekapcsolásnál a második lépésben a bal- és jobboldali külső összekapcsolás második lépése is végrehajtódik: a bal- és jobboldali logo sorok is az eredménytáblába kerülnek NULL értékekkel kiegészítve azokra az attribútumokra nézve, amellyel az eredménytáblába emelt sor nem rendelkezett. A **Θ feltétel szerinti külső összekapcsolás** ugyancsak három változatban van jelen a gyakorlatban, a balodali (jelölése $R_1 \ _\Theta |\bowtie R_2$), a jobboldali (jelölése $R_1 \bowtie|_\Theta R_2$) és a teljes (jelölése $R_1 \ _\Theta |\bowtie| R_2$) összekapcsolás-változatban. A megvalósítási/végrehajtási lépéseket illetően ez a fajta külső összekapcsolás megegyezik az előző, külső természetes összekapcsolás lépéseivel, értelemszerűen az első lépés nem a természetes (egyenlőségen a lapuló), hanem a Θ feltétel szerinti összekapcsolás less.

4. SQL – A RELÁCIÓS ADATBÁZISOK ADATKEZELÉSÉNEK DEKLARATÍV NYELVE

A relációs adatmodel egyszerű szerkezeti felépítésű, relációkon/kétdimenziós táblázatokon alapuló modell. Az adatbáziskezeléshez éppen ezért egy relációkat jól kezelő, deklaratív (leíró, nemprocedurális) nyelv kialakítása volt az elsődleges cél. Elsőként az 1974-ben megjelent SEQUEL (Structured English Query Language) nyelvet használták erre a célra, amely nyelv érthető angol szavakból épült ki, a névben is hangsúlyozva, hogy adatlekérdezésre szánták. A nyelv alapvetően a relációs algebrára épül, de túllépve azon sok egyéb más lehetőséget is kínál a használójának. A fejlődés/fejlesztés folyamán a nyelv elnevezése SQL-re módosult, és az adatlekérdezések mellett, adatmódosítások véghezvitelére, nézettáblák, indexek, eljárások kezelésének elvégzésére is alkalmassá vált.

Az SQL szabványosítása terén az első lépést az ANSI (American National Standards Institute) tette meg 1986-ban, egy évvel később az ISO (International Standards Organisation) is megjelentette az ISO SQL szabványt. Több szükségszerű szabványmódosítás is követte az 1986-os első SQL-szabványt: 1989-ben, 1992-ben, majd 1999-ben is. Az első szabványt már az akkori adatbáziskezelő rendszerek SQL nyelvei is túlszárnyalták, ezért hamarosan módosításra szorult. Az SQL92 és az SQL99 már jövőbe mutató szabványoknak számítanak.

Az SQL nyelv segítségével történik az adatbázisoknak, a tábláiknak és az adatfeldolgozáshoz és –lekérdezéshez szükséges egyéb segédmechanizmusok leírása, az adatok lekérdezése és módosítása, valamint az adatokhoz való hozzáférési jogosultságok kezelése. Az SQL nyelv utasításait többféleképpen csoportosítják, itt most egy nemszokványos csoportosítás következik:

1. adatdefiníciós résznyelv/funkciók (DDL – Data Definition Language),
2. adatmanipulációs résznyelv/funkciók (DML – Data Manipulation Language),
3. SQL-lekérdezések (Query) és
4. adatvezérlési/adatfelügyeleti résznyelv/funkciók (DCL – Data Control Language).

Az SQL-ben három fajta tábla/reláció ismert és használatos:

1. Tárolt relációk – klasszikus (az adatbázisban elhelyezkedő) táblák
2. Számítással kapott relációk – nézettáblák/nézetek, amelyeket nem tárolunk az adatbázisban (csak a leírásukat).
3. Ideiglenes táblák, amelyeket az adatbáziskezelő rendszer/az SQL feldolgozója használ a feladatok elvégzésénél (ilyen például egy lekérdezés eredményét

bemutató eredménytábla). A soron következő feladat elvégzése előtt eldobja/törli az előző feladat eredménytábláját.

Az adatfeldolgozási programok/alkalmazások természetesen procedurális utasításokat is igényelnek, ami kétféle módon valósíthat meg:

1. az SQL nyelv procedurális nyelvbe való beiktatásával és
2. az SQL nyelv procedurális utasításokkal való kibővítésével.

A továbbiakban az SQL legfontosabb, az egyes szoftverházak relációs adatbázisaiban eltérő SQL megvalósításaiban általában megegyező utasításai találhatók (a teljesség igénye nélkül).

4.1.ADATDEFINÍCIÓS RÉSZNYELV (DDL)

Egy adatbázis tényleges használatba vétele előtt a jól megtervezett adatbázist fizikailag létre kell hozni. Ehhez szükségszerű az adatmodellből kialakított relációs (adatbázis)modell fizikai szerkezetének meghatározása, illetve annak létrehozása. Ez az üres adatbázis definiálása, amely az adatbázis megnevezéséből (az adatbáziskezelő rendszer nyilvántartásába való beviteléből), és a táblaleírásainak iktatásából áll. Táblák csak adatbázison belül létezhetnek mégpedig egyedi nevekkel. A táblák oszlopokból és sorokból állnak, a létrehozásnál az oszlopokat névvel kell illetni (az oszlopneveket attribútumoknak vagy mezőknek is szokás nevezni), a sorokat a tábla adatokkal való feltöltésekor alakítják ki. Az ugyancsak egyedi neveket viselő mezők különböző, a létrehozáskor rögzített adattípusokat tartalmazhatnak (további megszorításokkal fűszerezve az adatok értéktartományára vonatkozóan, esetleg a sorban/más mezőben-mezőkban beírni kívánt adatokhoz viszonyítva). Következzenek most az adatdefiníciós résznyelv utasításai egy lehetséges, a létrehozáshoz szükséges logikai sorrendben. A táblák létrehozásánál a hivatkozási épség megszorítás is megfogalmazható, de szükség esetén minden módosítható és törölhető is. Minden SQL utasítás (ezzel értelem szerűen DDL utasítás is) egy »mondatot« képvisel, amelynek a végére a klasszikus mondatvég (pont) helyett pntvesszőt kell tenni!

4.1.1. Adatbázis szintű utasítások

Az adatbázis létrehozását a következő utasítás teszi lehetővé:

CREATE DATABASE *adatbázisnév*;

Az utasítás végrehajtásával az adatbáziskezelő rendszer nyilvántartásba veszi az adott nevő adatbázist, és azt fizikailag is létrehozza az alapértelmezett (vagy kérésre más) könyvtárban.

Egy adatbáziskezelő rendszer több adatbázist is kezelhet. Az iktatott adatbázisokról különböző adatbáziskezelő rendszerek különböző adatokat vesznek nyilvántartásba (adatbázisnév, befogadó könyvtár neve, létrehozás kelte, létrehozó azonosítója, stb.). Ezeket az adatokat a következő, argumentum nélküli utasításokkal lehet lekérni:

SHOW DATABASE;

Az adatbázison belüli ténykedéshez (táblalétrehozás, adatbevitel, -lekérdezés) az adatbázist meg kell nyitni (aktualizálni kell), amit a következő utasítás eredményez:

START DATABASE *adatbázisnév*;

Az adatbázis lezárása (deaktualizálása) szükségszerű művelet abban az esetben, ha esetleg az adatbázist törölni kellene, vagy másik adatbázist kell aktualizálni. Az utasítás formája::

STOP DATABASE;

A fent említett adatbázistörlés utasításának igénye és végrehajtása is olyan ritka, mint a fehér holló (emellett még azt is tudni kell, hogy a bázis törlésekor nemcsak az adatok törlődnek, hanem a teljes szerkezet: a táblák, a nézetek, az indexek, és az adatbázis neve is). Az utasítás a következő::

DROP DATABASE *adatbázisnév*;

4.1.2. Adattípusok

Az SQL-ben használt fontosabb adattípusok rövid leírásával a táblalétrehozási utasítás felé nyílik meg az út, ugyanis minden attribútumhoz (táblamezőhöz vagy –oszlophoz) kötelezően tartozik egy adattípus:

1. Karakter-adattípusok. Itt általában a rögzített, a változó hosszúságú és a különösen nagy hosszúságú karaktersorok jelölésére van lehetőség:
    a. CHAR(n) – rögzített n hosszúságú karaktersor,

b. VARCHAR(n) – legtöbb n hosszúságú karaktersor
c. CLOB (Character Large Object) – legfeljebb 128 terrabyte hooszúságú karaktersor

2. Számértéket tartalmazó adattípusok. Az egész számokat és lebegőpontos értékeket tartalmazó adattípusok legfőbb képviselői:
   a. INTEGER (INT) – egész számokat jelölő adattípus
   b. SMALLINT (SHORTINT) – kisebb egész számú értékeket jelölő adattípus
   c. FLOAT (REAL) – lebegőpontos számértékeket jelölő adattípus
   d. DOUBLE PRECISION – nagyobb pontosságú lebegőpontos számértékeket jelölő adattípus
   e. DECIMAL(n,m) vagy DECIMAL(n,m) fixpontos valós (nem egész) számértékeket tartalmazó adattípus (n-a számjegyek száma, ebből m a tizedesjegyek száma).
3. Bitsorokat leíró adattípusok. Kötött vagy változó számú bitsorokat tratalmazó adattípusok:
   a. BIT(n) – (fix) n hosszúságú bitsort jelölő adattípus
   b. BIT VARYING(n) – legtöbb n bitet befogadni képes adattípus
4. Dátum- és időértékeket jelölő adattípusok. A dátumértéket, az időértéket és az időbélyeget sorolják általában ebbe az adattípus-csoportba:
   a. DATE – a teljes dátumformát 'ÉÉÉÉ-HH-NN' formában tartalmazó adattípus.
   b. TIME – az idő(ponto)t 'HH:MM:SS,SS 'tartalmazó adattípus.
5. Logikai értékeket leíró adattípus. Ez az adattípus a BOOLEAN, és az igaz (TRUE), hamis (FALSE), és furamód az ismeretlen (UNKNOWN) értékeket tartalmazhatja (meg persze a NULL értéket is, hiszen ez minden adattípusnál érvényes értéknek számít, ha nincs rá külön tiltás (megszorítás) a konkrét attribútumnál).

4.1.3. Doméjnkezelési utasítások

Az angol domain az attribútumok értékhalmazát vagy értéktartományát jelöli: adott adattípuson tetszőleges értékkorlátok feltüntetésével. A doméjnek alkalmazása egyszerűsíti az attribútumok típus- illetve értékmeghatározását, különösen akkor ha a szemantikus (felhasználó által definiált, sajátságos, új) adattípusok alkamazása gyakori. A doméjnek érvényessége (»láthatósága«) az adatbázisra (adatbázissémára) korlátozódik. A több különböző táblában

elhelyezkedő attribútumok esetében egyszerűbb a doméjn alkalmazása, mint az összes attribútumot tartalmazó tábladefinicióknál az értékkorlátok leírásának ismételgetése.

A doméjn létrehozását a következő utasítás váltja ki:

CREATE DOMAIN *doméjnnév adattípus*
[DEFAULT érték]
[[CONSTRAINT megszorításnév] CHECK megszorítás];

- *doméjnnév* – a doméjn egyedi neve (az aktuális sémában)
- *adattípus* – beépített adattípus, vagy már létező doméjn neve
- *megszorításnév* – az értékhalmazra (doméjnre) megfogalmazott megszorítás egyedi elnevezése
- *megszorítás* – az értékmegszorítás kifejtése

A doméjn módosítása a következő utasítással végezhető el:

ALTER DOMAIN *doméjnnév*
[SET DEFAULT érték | DROP DEFAULT]
[ADD [CONSTRAINT megszorításnév] CHECK megszorítás |
DROP CONSTRAINT megszorításnév];

A doméjn törlése a következő egyszerű utasítással végezhető:

DROP DOMAIN *doméjnnév*;

4.1.4. Táblakezelési utasítások

A táblakezelési utasítások a tábla szerkezetének létrehozását és iktatását, a szerkezeti módosításokat, névváltoztatásokat, az attribútumok érték- és megszorításváltozásait és törléseit ölelik fel. A bonyolult utasítások esetében (itt a CREATE TABLE) mindenkor és mindenütt, itt is, csak egyszerűsített alakot lehet tárgyalni (mellette nem lehet elégszer hangsúlyozni, hogy ezek az utasítások némileg eltérők lehetnek az egyes konkrét adatbáziskezelő rendszerek esetében).

A táblalétrehozási utasítás alakja::

CREATE TABLE *táblanév* (*mezőnév*$_1$ *adattípus*$_1$*(mezőhossz*$_1$*) [mezőértékmegszorítás*$_1$*]*
[,...,mezőnév$_n$ *adattípus*$_n$*(mezőhossz*$_n$*) [mezőértékmegszorítás*$_n$*]]*
[, relációra_vonatkozó_megszorítás]);

- a *táblanév* adatbázissémán belül, a mezőnevek/attribútumnevek pedig relációsémáz (táblán) belül egyediek

- a mezőnév után az adattípus helyett doméjnnév is szerepelhet
- **egy mezőre vonatkoztatható megszorítások**:
  a. kulcsmegszorítás: **PRIMARY KEY,**
  b. hivatkozásiépség megszorítás (idegen/külső kulcs definiálása):

REFERENCES táblanév(kulcsnév) **[[ON DELETE CASCADES], [ON UPDATE CASCADES]]**

  c. attribútumértékre vonatkozó megszorítások:
    i. **UNIQUE** – egyedi értékeket vehet fel
    ii. **NOT NULL** – nem lehet null értékű
    iii. **DEFAULT** *alapértelmezett érték* – alapértelmezett érték megadása
    iv. **CHECK** *feltétel* – ellenőrzi a feltételt
- **relációra (az egy relácón belüli attribútumértékek egymáshoz fűződő (érték)viszonyára) megfogalmazható megszorítások**:
  a. kulcsmegszorítás: **PRIMARY KEY (***mezőnév*$_1$ [,...,*mezőnév* $_n$]),
  b. hivatkozásiépség megszorítás (idegen/külső kulcs definiálása):

FOREIGN KEY (*mezőnév*$_1$ [,...,*mezőnév* $_n$]) **REFERENCES** táblanév (*kulcsmezőnév*$_1$ [,...,*kulcsmezőnév*$_n$]**) [[ON DELETE CASCADES], [ON UPDATE CASCADES]]**

  c. attribútumértékre vonatkozó megszorítások:
    i. **UNIQUE (***mezőnév*$_1$ [,...,*mezőnév* $_n$]) – egyedi értékeket vehet fel a mezőcsoport
    ii. **CHECK** *feltétel* – ellenőrzi a feltételt relációséma szinten.

Példa (táblalétrehozás relációséma alapján):

A relációséma:

Hallgató (leckekönyvszám, hallgatónév, születésiév, lakcím-város, lakcím, telefon)

A tábla létrehozása:

```
CREATE TABLE hallgató (leckekönyvszám CHAR(8) PRIMARY KEY,
                       hallgatónév CHAR(20) NOT NULL,
                       neme CHAR(1)  DEFAULT »f« CHECK (neme IN (»f«, »n«))
                       születésiév NUMBER(4),
                       lakcím-város CHAR(20),
                       lakcím CHAR(20),
                       telefon CHAR(12));
```

A táblamódosítási utasítások száma igen jelentős, ami nem meglepő, hiszen a táblalétrehozási utasításnak nagyszámú paramétere van, és ezek mindegyikét módosítani is lehet. A módosító utasítások három csoportosításban kerülnek felsorolásra: új paraméterek bevitele a meglévő táblába, létező paraméterek módosítása és azok szükség szerinti törlése.

1. A tábla szerkezeti változtatása/módosítása (a tábladefiníció/táblaleírás) új szerkezeti elemek létrehozásával

    a. Új attribútum (oszlop, mező) hozzáadása

ALTER TABLE *táblanév*
ADD [**COLUMN**] *mezőnév adattípus(mezőhossz) [mezőértékmegszorítás] [, relációra_vonatkozó_megszorítás]***;**

    b. Elsődleges kulcs hozzáadása

ALTER TABLE *táblanév*
ADD CONSTRAINT PRIMARY KEY (*mezőnév*$_1$ [,...,*mezőnév*$_n$])**;**

    c. Idegen (külső) kulcs hozzáadása

ALTER TABLE *táblanév*
ADD CONSTRAINT FOREIGN KEY (*mezőnév*$_1$ [,...,*mezőnév*$_n$]) **REFERENCES** *táblanév*(*kulcsmezőnév*$_1$ [,...,*kulcsmezőnév*$_n$**]**)
[[**ON DELETE CASCADES**], [**ON UPDATE CASCADES**]]**;**

    d. Index (keresést gyorsító tábla) hozzáadása

ALTER TABLE *táblanév*
ADD INDEX|**KEY** [*indexnév]* (*mezőnév*$_1$ [,...,*mezőnév*$_n$]]) **;**

    e. Új megszorítás hozzáadása

ALTER TABLE *táblanév*
ADD CONSTRAINT [*megszorításnév]* **CHECK** *feltétel* **;**

2. A tábla szerkezeti módosítása az attribútumok (oszlop, mező) leírásának megváltoztatásával

    a. Az attribútum alapértelmezett értékének hozzárendelése/törlése

ALTER TABLE *táblanév*
ALTER [**COLUMN**] *mezőnév*
SET DEFAULT *érték* |**DROP DEFAULT ;**

    b. Az attribútum nevének és leírásának módosítása

ALTER TABLE *táblanév*

CHANGE [**COLUMN**] *régi_mezőnév*
*új_ mezőnév új_adattípus(új_mezőhossz) [új_mezőértékmegszorítás]***;**

    c. Az attribútum leírásának módosítása (nevének meghagyásával)

ALTER TABLE *táblanév*
MODIFY [**COLUMN**] *régi_mezőnév*
*új_adattípus(új_mezőhossz) [új_mezőértékmegszorítás]***;**

3. Táblamódosítás meglévő szerkezeti elemek törlésével

a. Attribútumok (oszlop, mező) törlése

ALTER TABLE *táblanév*
DROP [**COLUMN**] *mezőnév***;**

b. Elsődleges kulcs törlése

ALTER TABLE *táblanév*
DROP PRIMARY KEY;

c. Idegen kulcs törlése

ALTER TABLE *táblanév*
DROP FOREIGN KEY (*mezőnév*$_1$ [,...,*mezőnév*$_n$])**;**

d. Index törlése

ALTER TABLE *táblanév*
DROP INDEX|**KEY** [*indexnév]* **;**

e. Megszorítás törlése

ALTER TABLE *táblanév*
DROP CONSTRAINT *megszorításnév***;**

A tábla törlése egyszerű és veszélyes utasítás, mert végrehajtásával minden törlődik a táblával kapcsolatban: a benne lévő adatok és a szerkezet (a szerkezeti leírás) egyaránt:

DROP TABLE *táblanév***;**

4.1.5. Indexek gondozása (kezelése)

A kulcsok (elsődleges kulcsok) létrehozásának pillanatában automatikusan létrejön egy keresést meggyorsító mechanizmus/adatszerkezet is – az index. Az index az adatbázisok megvalósításánál általában bináris keresőfaként jelentkezik. A sok adattal (nagyszámú relációval és relációelőfordulással, vagy táblával és táblasorral) rendelkező adatbázisok esetében az adatfeldolgozás jelentősen gyorsítható ha hatékony az adatok kinyerése az adatbázisból. A hatékonyság fokozásához nagymértékben hozzájárulnak az indexek is.

Az index létrehozását a következő utasítás végzi:

CREATE [**UNIQUE**] **INDEX** *indexnév*
ON *táblanév* (*mezőnév*$_1$ [,...,*mezőnév*$_n$])*;*

A tábla törlése már ismerős lehet az eddigi törlésutasítások megismerése után:

DROP INDEX *indexnév*;

4.1.6. Nézettábla (nézet) kezelése

A nézettábla, aklasszikus táblákkal ellentétben nem létezik fizikailag az adatbázisban, az adatbáziskezelő rendszer csupán a leírásukat rögzíti. A nézettáblák fizikailag a rájuk való hivatkozáskor jönnek létre, és a lekérdezésük semmiben sem különbözik a CREATE TABLE paranccsal létrehozott klasszikus tábláktól. Bizonyos feltételek mellett (egyetlen R relációra épül – R csak egyszer és egyedül fordulhat elő a FROM-ban, R nem lehet a WHERE-ben és egyetlen alkérdésben sem, a SELECT-ben minden olyan mezőnek szerepelnie kell, amely NOT NULL leírású és nincs alapértelmezett értéke az R-ben - Ullman) még adatmódosítás is eszközölhető rajtuk.

A nézettábla leírásának megfogalmazása (a nézettábla létrehozása):

CREATE VIEW *nézettáblanév* [(*mezőnév*$_1$ [,...,*mezőnév*$_n$])]
AS *műveletsor*;

- a *műveletsor* általában egy SELECT lekérdezés

A nézettábla törlése:

DROP VIEW *nézettáblanév*;

4.2. ADATMANIPULÁCIÓS RÉSZNYELV/FUNKCIÓK (DML)

Az adatbázis állapotának (a felhasználói adatok értékeinek) megváltoztatását az adatmanipulációs műveletek (más néven a módosítások) végzik. A továbbiakban a következő három adatmanipulációs művelet ismertetése következik:

1. adatbevitel (új sorok beszúrása a táblába),
2. létező adatsorok egyes adatainak értékváltoztatása,
3. létező adatok bizonyos részhalmazának törlése és
4. a művelethalmaz atomiságának biztosítása – a tranzakciók.

4.2.1. Új sor/sorok beszúrása a táblába

Az adatbevitelt soronkénti beszúrással, de akár több sor bevitelét kiváltó utasítással is meg lehet valósítani. Indulásként az egy sor beszúrását vező utasítás leírása következik:

INSERT INTO *táblanév* [(*mezőnév*$_1$ [,...,*mezőnév*$_n$])]
VALUES (*érték*$_1$, ..., *érték*$_n$)

- ha az összes attribútumérték beszúrásra/beírásra kerül (ugyanannyi érték helyezkedik el a beszúrási értéklistában ahány attribútum van, és a sorrendjük is megfelelő), akkor az attribútumlista ([(*mezőnév*$_1$ [,...,*mezőnév*$_n$])]) el is hagyható,
- ha a két lista (attribútum- és értéklista) tagszámban nem egyezik vagy a tagszámbeli egyezés mellett sorrendbeli egyezés nincs, akkor az attribútumlista nem mellőzhető.

A több sor bszúrását végző művelet szintaktikája a következő:

INSERT INTO *táblanév* [(*mezőnév*$_1$ [,...,*mezőnév*$_n$])]
AS *műveletsor*;

- az attribútumlista itt is elmaradhat, de csak abban az esetben, ha a *műveletsor* -t képviselő SELECT mondatban a kiválasztott attribútumnevek megegyeznek a *táblanév* tábla attribútumneveivel (és létrehozási sorrendjével).

4.2.2. Létező adatsorok egyes attribútumainak módosítása (értékváltoztatása)

A beszúráshoz hasonlóan itt is lehetőség van »gyalogos« vagy »kézi« adatmódosításra, amikoris egyes kiválasztott attribútumok értékét az UPDATE utasításba kézzel begépelt értékekre változtatja az adatbáziskezelő rendszer. Az új értékeket (amelyekre az UPDATE utasításban felsorolt attribútumok értékeit csrélni kell) persze ki lehet kerestetni az adatbáziskezelő rendszerrel magából az adatbázisból is (ha egyáltalán vannak olyan értékek az adatbázisban).

A »kézi« módosítási művelet alakja a következő:

UPDATE *táblanév*
SET *mezőnév*$_1$ = *érték*$_1$ [,...,*mezőnév*$_n$ = *érték*$_n$]
[**WHERE** *feltétel*];

A módosítási művelet másik alakja pedig így néz ki:

UPDATE *táblanév*
SET [(*mezőnév*$_1$ [,...,*mezőnév*$_n$]) = (SELECT lekérdezés)
[**WHERE** *feltétel*];

A SELECT mondatról vagy lekérdezésről eddig nem esett szó, a későbbiekben viszont annál több fog, minden pótlásra kerül. A SELECT utasítás ismeretlen volta azonban nem zavarja az eddigiekben ismertetett műveletek érthetőségét, viszont a WHERE záradékban feltüntetett *feltétel* (amely lehet egyszerű-összehasonlítás, vagy logikai műveletek segítségével összekötött egyszerű összehasonlítások láncolata) most már magyarázatra szorul. A feltétel kiértékelésekori igazságérték kétféle lehet: igaz (TRUE) vagy hamis (FALSE). A fenti és a későbbi utasításokban megjelenő *feltétel* lehetséges alakja/tartalma a következő lehet:

1. egyszerű kiválasztási műveletet/műveleteket tartalmazó kifejezés, amelyben az operátorok között konstansok és/vagy a FROM utáni tábla/táblák attribútumai szerepelhetnek. Az összehasonlítási operátorok a következők: : = , > , < , >= , <= , <> ,
2. összetett kifejezés, amelyben a fenti összehasonlítási operátorok mellett matematikai alapműveletek (+, -, *, /), karakterlánc-összekapcsolási művelet (‖) és logikai műveletek (AND, OR, NOT) is szerepelhetnek,
3. BETWEEN relációs operátort tartalmazó kifejezés (adott értékhatárokon belül van-e egy adott kifejezés), melynek alakja:

   kifejezés$_1$ **BETWEEN** *kifejezés*$_2$ **AND** *kifejezés*$_3$
4. IN operátort tartalmazó kifejezés (tartalmazza-e az adott értéket az IN után feltüntetett lista) a következő szintakszissal:

   mezőnév **IN** (*listaelem*$_1$ [, *listaelem*$_2$,..., *listaelem*$_n$])
5. LIKE operátort tartalmazó kifejezés – mintával ('abcd') való összehasonlítás, melynek formája

   mezőnév **LIKE** 'abcd' [**ESCAPE** 'x']

   – speciális jelek a mintában:
     - „_” – egy karaktert helyettesít
     - „%” – 0 vagy nagyobb hosszúságú karaktersort helyettesít
   – ha épp a speciális jelekre kell rákeresni, akkor a jelek speciális jellegét az ESCAPE 'x' karaktere semlegesíti. Példa: rákeresés az »50%«-kal kezdődő mezőnév$_1$-értékekre

   mezőnév$_1$ **LIKE** '50#%%' [**ESCAPE** '#']
6. IS NULL relációs operátort tartalmazó kifejezés

   mezőnév **IS NULL**

4.2.2.1.Műveletek NULL értékekkel

Sajátságos eredményeket adnak az aritmetikai és logikai műveletek, amennyiben a kifejezésekben NULL értékű tagok (argumentumok, attribútumok) vannak. A két szabály a NULL értékeket tartalmazó műveletekre a következő (Ullman):

1. ha egy aritmetikai műveletben legalább az egyik argumentun NULL, akkor az eredmény is NULL.
2. Ha NULL érték kerül összehasonlításra bármely értékkel, az eredmény ISMERETLEN (UNKNOWN).

Logikai műveletek a háromértékű logikában – x és y logikai változók (Ullman):

x	**y**	**x AND y**	**x OR y**	**NOT x**
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

4.2.3. Létező adatok bizonyos részhalmazának törlése

Bizonyos helyzetekben szükségszerűvé válhat az adatok fizikai eltávolítása, törlése egy adott táblából. Az adatok törlésének egysége a sor, amit az uasításban szereplő WHERE záradék segítségével lehet kijelölni. Fontos odafigyelni ennek a záradéknak a jelenlétére, nélküle ugyanis kitrörlődik a tábla teljes tartalma. A törlési utasítás egyszerű:

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

Az előző utasításnál már tisztázódott az a tény, amely itt is érvényes: csak azok a sorok kerülnek törlésre, amelyekre a WHERE záradékban található feltétel igaz értéket vesz fel.

4.2.4. A művelethalmaz atomiságának biztosítása – a tranzakciók

A gyakorlatban gyakran előfordulnak olyan helyzetek, amikor egy utasításcsoport egyes utasításainak elvégzése után az adatbázis átmenetileg ellentmondásos (nemkonzisztens) állapotba kerül, de az utasításcsoport utolsó utasításának végrehajtása után már ellentmondásmentes (konzisztens) állapotúvá válik. Ilyen esetekben az utasításcsoport végrehajtásának megszakítása (bármilyen okból is történjen) nem elfogadható. Ezt elkerülendő vezették be az adatbáziskezelő rendszerekben a tranzakció fogalmát, amely az utasításcsoport atomi végrehajtását biztosítja. Ha nem lehet atomian végrehajtani az össz, tranzakción belül megfogalmazott utasítást, úgy egyet sem szabad végrehajtani közülük. A tranzakció kezdetét a következő SQL utasítás jelzi:

START TRANSACTION

A tranzakció befejezésének két fajtája van:

1. **COMMIT** – a tranzakció atomian végrehajtott utasításainak módosításai véglegesítődnek az adatbázisban és
2. **ROLLBACK** – a tranzakció utasításainak végrehajtása megszakad és marad az eredeti, tranzakció-végrehajtás előtti adatbázisállapot.

4.3. SQL-LEKÉRDEZÉSEK

Az SQL nyelv talán leggyakrabban használt utasítása a lekérdezéseket leíró, illetve megvalósító SELECT mondat. A relációs algebrára épülő, több záradékkal és sok paraméterrel rendelkező utasítás kissé bonyolult, de az egyszerű lekérdezési igények megfogalmazásához nem kell ismerni a mondat teljes felépítését. A SELECT ismertetésekor két hozzáállást lehet megkülönbözteni. Az egyik a minimális/kötelezően felírandó záradékoktól (a legegyszerűbb lekérdezések bemutatásától) indulva ismerteti az egyre bonyolultabb lekérdezések igényelte újabb záradékokat, a másik pedig a teljes SELECT mondat felírásával indít, és sorban bemutatja a záradékokat és az általuk nyujtott lehetőségeket. A továbbiakban a második forma ismertetési algoritmusát követve kerül bemutatásra a teljes SELECT mondat. Hangsúlyozni kell a tényt, hogy a lekérdezések eredménye olyan ideiglenes tábla (speciális esetben adatsor vagy akár egyetlen érték) lesz, amely a következő lekérdezés végrehajtásának pillanatáig él (az aktuális lekérdezés mindig átírja az előző lekérdezés eredménytábláját). Az eredménytábla kialakításának saját algoritmusa van, amelyben a záradékok végrehajtási sorrendje eltér a SELECT-ben feltüntetett és kötelező záradéksorrendtől. A SELECT mondatok összekapcsolhatók egymással

és belső lekérdezéseket is tartalmazhatnak. A szabvány szerinti teljes SELECT mondat szerkezete a következő:

SELECT ...	*attribútumok kiválasztása – **projekció/vetítés***
FROM ...	*táblanevek felsorolása – **keresztszorzat***
[WHERE ...]	*sorok kiválasztása - **szelekció***
[[GROUP BY ...	*csoportosítás*
[HAVING ...]]	*csoportok kiválasztása*
[ORDER BY ...]	*az eredménytábla rendezése*
[UNION ...]	*két eredménytábla összeolvasztása*

A legrövidebb érvényes SELECT mondat az első két záradékot (SELECT, FROM) tartalmazza, és ezt a két záradékot tartalmaznia kell minden, tetszőleges bonyolultságú lekérdezésnek. A lekérdezésekben tehát legalább azt fel kell tüntetni, hogy honnan és mi jelenjen meg az eredménytáblában. A többi záradék esetleges. A SELECT mondat végrehajtási algoritmusa rögzíti a záradékok végrehajtási sorrendjét, ismerete segít a záradékokban rögzített kérelmek megértéséhez:

FROM ...	*táblanevek felsorolása – **keresztszorzat***
[WHERE ...]	*sorok kiválasztása - **szelekció***
[[GROUP BY ...	*csoportosítás*
[HAVING ...]]	*csoportok kiválasztása*
SELECT ...	*attribútumok kiválasztása – **projekció/vetítés***
[ORDER BY ...]	*az eredménytábla rendezése*

4.3.1. A SELECT záradék

A SELECT záradék teljes alakja a mondat minimális/kötelező szerkezetében feltüntetve a következő:

SELECT [ALL | DISTINCT] *mezőnév*$_1$ [,...,*mezőnév* $_n$] | *****
 FROM *táblanév*$_1$ [,...,*táblanév* $_n$];

A SELECT záradék a végrehajtásban az utolsó előtti aktivitás az eredménytábla végleges rendezése előtt. Megszorítást is tartalmazhat (DISTINCT – adott attribútumoknak csak a különböző értékeit jelenítse meg) a kivetítésre vonatkozóan. A kivetítés természetesen a FROM záradékban megtalálható táblák attribútumaira vonatkozhat. A kulcsszavak és opciók jelentése:

ALL – az eredménytábla sorai az attribútumlistában szereplő értékeit fogják tartalmazni a FROM-ban feltüntetett tábla/táblák minden sorából, még akkor is, ha az értékek ismétlődnek. Ez az alapértelmezett érték (akkor is ez fog végrehajtódni, ha semmi sincs feltüntetve),

DISTINCT – az eredménytáblában csak azokat a sorokat jeleníti meg, amelyek a DISTINCT-ben foglalt attribútumértékekre különböznek,

***** - az eredménytábla az összes attribútumértéket tartalmazni fogja a CREATE TABLE utasításban feltüntetett sorrendben,

mezőnév $_1$ [,...,*mezőnév* $_n$] – csak a attribútumlistában szereplő mezőértékek jelennek meg az eredménytáblában, a listában feltüntetett sorrendben.

Példa:

```
SELECT *
    FROM hallgató;
```

kilistázza a hallgató tábla minden sorát a létrehozáskor feltüntetett attribútum-sorrendben.

Példa:

```
SELECT leckekönyvszám, hallgatónév, születésiév
    FROM hallgató;
```

az eredménytábla a »leckekönyvszám, hallgatónév, születésiév« attribútumlista értékeit tartalmazza eredményként.

Példa:

```
SELECT DISTINCT godrodjstu
    FROM hallgató;
```

ismétlődésmentesen listázza a hallgatók születési éveit.

A projekcióban vetítésre kerülő lista általában mezőneveket (attribútumneveket) tartalmaz, de rendelkezésre állnak még egyéb más lehetőségek is. A lista elemei a következők lehetnek:

- mezőnév – ha több táblában is található azonos attribútumnév, akkor azokat a listában a *táblanév.mezőnév* jelöléssel fogadja csak el az adatbáziskezelő rendszer,
- aritmetikai kifejezés, amelynek argumentumai konstansok és attribútumnevek lehetnek,
- attribútumokra értelmezett összesítő műveletek (aggregációs funkciók),
- egyéb kifejezés.

Példa – **aritmetikai kifejezés (az SQL DATE és YEAR beépített függvényeivel)**:

SELECT leckekönyvszám, hallgatónév,YEAR(DATE())-születésiév
FROM hallgató;

az eredménytábla a hallgató leckekönyvszámát, nevét és életkorát fogla tartalmazni (a DATE() függvény az aktuális dátumot adja vissza/képviseli, amiből a YEAR függvény kiemeli az évszámot).

Példa – **karakteres kifejezés (LEFT és UPPER karakterfüggvényekkel)**:

SELECT LEFT(brindeksa,1,1) UPPER(imestu) FROM student;

az eredménytábla csupán egy oszlopot fog tartalmazni, amelyben a leckekönyvszám első (baloldali) jele összeolvad a hallható nevével (nagybetűkkel írva).

Összesítő műveletek/aggregációs funkciók)

Az összesítő műveletek kiszámolnak vagy kikeresnek valamilyen értéket a GROUP BY záradékban feltüntetett csoportra (a GROUP BY záradék után következő mező minden értékére). A csoportosítás alapját képező mező minden értékére létrejön egy sor az eredménytáblában, benne a csoportosítás alapját képező mező egy-egy értéke és az összesítő művelet eredménye az adott értékre. Amenniben nics csoportosítási meghagyás (nincs GROUP BY záradék), úgy a rendszer az egész táblát egy csoportnak tekinti, és az eredménytábla egyetlen értéket tartalmaz (minden összesítő műveletre). Az összesítő műveletek már ismertek az előző fejezetből, most az AQL-leírásuk következik:

1. Számlálási művelet:

COUNT([DISTINCT] *mezőnév* **| *)**

A fenti megfogalmazás eredménye a mezőnév NOT NULL értékű sorainak a száma. A DISTINCT csak a mezőnév különböző értékű sorainak számát eredményezi az eredménytáblában, míg a csillag argumentummal rendelkező számlálási művelet a tábla sorainak számát biztosítja.

Példa:

SELECT COUNT (leckekönyvszám,)
FROM hallgató;

az eredménytábla az össz nyilvántartásba vett hallgatók száma kerül, mivel a tábla/reláció kulcsa (leckekönyvszám) nem vehet fel NULL értéket.

2. Összegzési művelet

SUM (**[ALL]** *kifejezés* | **[ALL|DISTINCT]** [*mezőnév*])

Az összegzési művelet használható adott attribútum táblában (a különböző sorokban) elhelyezkedő értékeinek összeadására, az attribútumnak a táblában lévő csak különböző értékeinek összeadására, vagy egy érvényes kifejezés sorokra értelmezhető/számítható értékének az összegzésére.

Példa:

```
SELECT SUM (DISTINCT(születésiév))
    FROM hallgató;
```

Egyedi számértékű attribútum lévén, csak a születési évre lehet felírni összegzési összesítő műveletet. A lekérdezésnek gyakorlati értéke nincs, mert a hallgatók születési éveinek (ismétlődés nélküli értékek) összegét (az összegzésnél minden évszámot csak egyszer vesz figyelembe) nyújtja.

3. Átlagérték-számítási művelet

AVG (**[ALL]** *kifejezés* | **[ALL|DISTINCT]** [*mezőnév*])

A művelet alakja (szintaktikája) teljesen megegyezik ay összegzési műveletével, lényegében a tartalom (szemantika) is ugyanaz, csak itt ay átlagérték kisyámításáról van szó.

Példa:

```
SELECT AVG(YEAR(DATE())-születésiév)
    FROM hallgató;
```

A lekérdezés a nyilvántartásban lévő hallgatók átlagéletkorát adja meg.

4. Legkisebb érték kikeresése

MIN(*mezőnév*)

a *mezőnév* jelölhet karakteres, dátum típusú vagy számértéket tartalmazó attribútumot is.

Példa:

```
SELECT MIN(születésiév)
    FROM hallgató;
```

A lekérdezés eredménytáblája a legidősebb hallgató(k) születési évét jeleníti meg.

5. Legnagyobb érték kikeresése

MAX(*mezőnév*)

a *mezőnév* hasonlóan az előző összesítő műveletnél elmondottakhoz csak rendezhető adattípusú lehet (karakteres, dátum vagy számérték típusú).

Példa:

```
SELECT MAX(születésiév)
    FROM hallgató;
```

A legfiatalabb hallgató(k) születési évét adja vissza eredményként az előző SELECT.

4.3.2. A FROM záradék

Ez a második és egyben utolsó kötelező záradék a SELECT mondatban. Ahogy a SELECT záradék nélkül nem lehetséges megfogalmazni, hogy mit is jelenítsen meg az adatbáziskezelő rendszer, úgy a FROM hiányában nem fogalmazható meg az elvárás, hogy honnan jelenítse meg az adatokat.A FROM záradék alakja a következő:

FROM *táblanév*$_1$ [*táblanév_rövid_neve*$_1$]
[,...,*táblanév*$_n$ [*táblanév_rövid_neve*$_n$]]

Ebben a záradékban kerülnek felsorolásra azok a táblanevek amelyek segítségével a rendszer létrehozza a keresztszorzatot (Descartes szorzat). Ez magában még nem azt jelenti, hogy a felsorolt táblák mindegyikéből adatok is kerülnek az eredménytáblába. A teljes keresztszorzat a gyakorlatban sohasem felel meg egy konkrét lekérdezéshez, így a későbbi záradékokban a keresztszorzat sorai szűrésre kerülnek (elsősorban a WHERE záradékban leírt feltétel alapján). A táblák összekapcsolását a leggyakrabban a théta összekapcsolás egyik változatával, az equijoin összekapcsolással végzik (ez a két táblában elhelyezkedő kulcs-idegen kulcs értékpárok egyenlőségét jelöli). Egyes esetekben a tábla saját magával is összekapcsolható (ugyancsak a kulcs-idegen kulcs értékpárok egyenlősége alapján), amit visszamutató vagy

rekurzív kapcsolatnak neveznek. A FROM záradék leírásából látható, hogy a táblanévhez néhány karakteres rövid név is rendelhető, ha a záradék több (és főleg hosszú, sokkarakteres) táblanevet tartalmaz. A táblanévhez rendelt (a SELECT-en belül egzedi) rövid név kötelező ha visszamutató, rekurzív kapcsolatot ír le a FROM záradék.

4.3.3. A WHERE záradék

A WHERE záradék *feltétel* részében megtalálhatók azok a leírások (feltételek) amelyek a táblák összekapcsolását írják le, mellettük megfogalmazhatók egyéb más megszorítások (feltételek) is. Az eredménytáblába csak azok a sorok kerülnek, amelyekre a feltétel igaznak bizonyul. A záradék alakja igazán egyszerű:

WHERE *feltétel*

A záradék tartalmazta *feltétel* teljes leírása a módosítási műveletnél (UPDATE) található.

Példa:

```
SELECT *
    FROM hallgató
    WHERE telefon IS NOT NULL;
```

A lekérdezés eredménytáblája a rögzített telefonszámmal rendelkező hallgatók össz adatát tartalmazza.

4.3.4. A GROUP BY záradék

A GROUP BY záradék a sorok csoportosítását váltja ki azokon az attribútumokon, amelyek a kulcsszót követik (csoportosító attribútumok). A csoportosító attribútumoknak a SELECT záradékban is meg kell jelenniük. Rajtuk kívül más, összesítési operátor nélküli attribútum nem jelenhet meg a SELECT záradékban! A csoportosító attribútumokon kívül a SELECT záradék olyan összesítő műveleteket tartalmazhat, amelyeknek argumentumai attribútumok vagy attribútumot tartalmazó kifejezések. A csoportosító attribútumok minden egyes értéke egy csoportot képvisel (a csoportba azok a táblasorok tartoznak, amelyek a csoportosító attribútumok azonos értékét tartalmazzák – ezekre a csoportokra hajtódnak végre az összesítő műveletek). A csoportosító attribútumok sorrendje a beágyazási sorrendet határozza meg. A GROUP BY záradék egyszerű alakú:

GROUP BY *mezőnév*$_1$ [,...,*mezőnév*$_n$]

Példa:

```
SELECT születésiév, COUNT(*)
    FROM hallgató
    WHERE telefon IS NOT NULL
    GROUP BY születésiév;
```

A SELECT mondat ilyen formában való írása (minden új záradék új sorban helyezkedik el) nem kikötés (szabály, törvény) de egyszerű áttekinthetőséget biztosít, és nagyban megkönnyíti az értelmezést is az emberek számára. Az eredménytábla két oszlopot tartalmaz: a hallgató táblában fellelhető születésiév-értékeket (ismétlődésmentesen), és az abban az évben született hallgatók számát.

Példa:

```
SELECT születésiév, neme, COUNT(*)
    FROM hallgató
    WHERE telefon IS NOT NULL
    GROUP BY születésiév, neme;
```

Az eredménytábla három oszlopot tartalmaz: a hallgató táblában fellelhető születésiév-értékeket és a hallgató nemét (a kettőt összevonva ismétlődésmentesen), és az abban az évben született férfi és női hallgatók számát. Az eredménytábla alakja a következő:

születésiév	**neme**	**COUNT(*)**
1991	férfi	56
1991	nő	8
1992	férfi	63
1992	nő	11

4.3.5. A HAVING záradék

Ha a GROUP BY záradék által kialakított csoportok közül valamelyik összesített tulajdonság alapján kell válogatni, akkor azt a HAVING záradékban lehet megtenni. A záradék alakja a következő:

HAVING *feltétel*

Az eredménytáblába csak azok a csoportok jutnak, amelyek kielégítik a HAVING kulcsszót követő feltételt. Megkötések/értelmezések a HAVING záradék alkalmazásához (Ullman):

1. A HAVING záradékban szereplő összesítő műveletek (aggregációs funkciók), egy-egy csoport kialakítását/feldolgozását követően azonnal végrehajtódnak, és csak arra a csoportra vonatkoznak.
2. A HAVING záradékban bármely FROM-ban felsorolt tábla attribútumára értelmezett összesítő művelet megjelenhet. Attribútumok önmagukban a HAVING záradékban csak akkor jelenhetnek meg, ha a GRUP BY záradékban is szerepelnek.

Példa:

```
SELECT születésiév, COUNT(*)
    FROM hallgató
    WHERE telefon IS NOT NULL
    GROUP BY születésiév
    HAVING születésiév BETWEEN 1980 AND 1985;
```

Az eredménytábla azon telefonnal rendelkező hallgatók számát adj meg évekre bontva, akik 1980 és 1985 között születtek. A feladat megoldható a HAVING záradék alkalmazása nélkül is:

```
SELECT születésiév, COUNT(*)
    FROM hallgató
    WHERE telefon IS NOT NULL AND születésiév BETWEEN 1980 AND 1985
    GROUP BY születésiév;
```

Példa:

```
SELECT születésiév, COUNT(*)
    FROM hallgató
    WHERE telefon IS NOT NULL AND születésiév BETWEEN 1980 AND 1985
    GROUP BY születésiév
    HAVING COUNT(*)>10;
```

Az eredménytábla azokat az 1980 és 1985 közötti éveket jeleníti meg, amelyekben az akkor született és telefonnal rendelkező hallgatók száma nagyobb 10-nél (a HAVING záradék nem nélkülözhető).

Példa:

```
SELECT lakcím-város, COUNT(*)
    FROM hallgató
```

```
WHERE telefon IS NOT NULL
GROUP BY lakcím-város
HAVING MIN(születésiév)<1970;
```

Az eredménytábla azokat a városokat (csoportokat) jeleníti meg a telefonnal rendelző hallgatók számával egyetemben, amelyekben van legalább egy olyan hallgató aki 1970 előtt született (a HAVING záradék nem nélkülözhető).

4.3.6. Az ORDER BY záradék

Az eredménytábla véglegesítése előtt, utolsó lépésként kerül sor az adatok rendezésére ha a SELECT mondat tartalmazza az ORDER BY záradékot. A záradék alakja a következő:

ORDER BY *mezőnév*$_1$ | *attribútum_sorszáma_a_projekcióban*$_1$ [ASC|DESC]
[,...,*mezőnév* $_n$ | *attribútum_sorszáma_a_projekcióban* $_n$ [ASC|DESC]];

Az eredménytábla adatainak rendezése az ORDER BY kulcsszót követő lista attribútumértékei szerint történik. Az első attribútum értékének egyenlősége esetén a rendezés a második attribútum értékei szerint folytatódik (történik), majd a második attribútum értékeinek egyezése esetén a harmadik attribútum értékei szerint, stb. Megjegyzések:

1. Az ORDER BY záradékban csak azok az attribútumok (*mezőnév*$_i$) jelenhetnek meg, amelyek a projekcióban (SELECT záradék) is jelen vannak,
2. Az *attribútum_sorszáma_a_projekcióban*$_i$ érték az az attribútum sorszámát jelenti a SELECT záradék listájában,
3. A záradék listájában kifejezés is alkalmazható,
4. Az ASC kulcsszó (ASCENDING) növekvő sorrendű rendezést, a DESC (DESCENDING) pedig csökkenő sorrendű rendezést vált ki.

Példa:

```
SELECT lakcím-város, születésiév, hallgatónév, lakcím
        FROM hallgató
        ORDER BY lakcím-város, születésiév, hallgatónév;
```

A lekérdezés eredménytáblájának első attribútuma (lakcím-város) addig nem fog megváltozni, amíg atlálhatók különböző érétkű születési évek az adott városhoz kötötten. Az eredménytábla második attribútuma (születésiév) változatlan marad, amíg a lakcím-város+születésiév értékkettőshöz található hallgatónevek (amelyeket rendezetten illeszt a

rendszer a változatlan város+születésiév értékkettőshöz. A hallgatónevek elfogytával először az eggyel külsőbb attribútum (születésiév) értéke változik meg, majd ha a változások ezen a szinten is elfogytak, akkor a lakcím-város értékváltozására kerül sor.

4.3.7. Többtáblás (több relációra vonatkozó) lekérdezések

Egyszerű egytáblás lekérdezések esetén a FROM záradék csak egyetlen táblanevet, vagy egy táblanevet és annak rövidített nevét tartalmazza:

FROM *táblanév* [*táblanév_rövid_neve*]

A több relációra vonatkozó lekérdezésekesetén két ismérv szerinti csoportosítás különböztethető meg: az összekapcsolás típusa szerint és az összekapcsolási igény megfogalmazása szerint. Az összekapcsolás leggyakrabban használt típusai:

1. keresztszorzat,
2. théta(equi)join,
3. természetes összekapcsolás,
4. külső összekapcsolás.

Az összekapcsolás megfogalmazása kétféle lehet:

1. implicit (az összekapcsolás feltételei a WHERE záradékban helyezkednek el)
2. explicit (az összekapcsolás típusa és feltételei a FROM záradékban vannak).

Az alábbiakban az összekapcsolások és megfogalmazásaik változatai következnek. Az ismertetésben a következő relációk kerülnek összekapcsolásra:

Hallgató (leckekönyvszám, hallgatónév, neme, születésiév, lakcím-város, lakcím, telefon)

Tárgy (tárgykód, tárgynév, heti_óraszám, kredit)

Vizsga (*leckekönyvszám*, *tárgykód*, dátum, jegy)

A hivatkozásiépség megszorítás vizuálisan is látható a relációleírásokból (az idegen kulcs dőlt betűvel szedett), de álljanak itt explicite is:

Vizsga[leckekönyvszám] $\subseteq$ Hallgató[leckekönyvszám] és

Vizsga[tárgykód] $\subseteq$ Tárgy[tárgykód]

Az összekapcsolás-változatok:

1. A keresztszorzat implicit megfogalmazás nélkül (nincs gyakorlati értéke)

```
SELECT *
    FROM hallgató, vizsga;
```

2. Equijoin keresztszorzat segítségével, implicit megfogalmazással a WHERE-ben

```
SELECT *
```

FROM hallgató, vizsga
WHERE hallgató.leckekönyvszám = vizsga. leckekönyvszám;

3. Equijoin [INNER] JOIN utasítás segítségével, explicit megfogalmazással a FROM-ban

Az INNER JOIN utasítás alakja:

SELECT [ALL | DISTINCT] *mezőnév*$_1$ [,...,*mezőnév*$_n$] | *
FROM *táblanév*$_1$
[**INNER**] **JOIN** *táblanév* $_2$ **ON** *összekapcsolási_feltétel*$_1$ |**USING** (*mezőlista*$_1$) **...**
[**INNER**] **JOIN** *táblanév*$_n$ **ON** *összekapcsolási_feltétel*$_{n-1}$ |**USING** (*mezőlista*$_{n-1}$);

Példa-1:

SELECT *
FROM hallgató
JOIN vizsga **ON** hallgató.leckekönyvszám = vizsga. leckekönyvszám;

Példa-2:

SELECT *
FROM hallgató
JOIN vizsga **USING** (leckekönyvszám);

4. Természetes összekapcsolás keresztszorzar segítségével a WHERE-ben

SELECT h.*, v.tárgykód, v.dátum, v.jegy
FROM hallgató **AS** h, vizsga **AS** v
WHERE h.leckekönyvszám = v. leckekönyvszám;

5. Természetes összekapcsolás explicit megfogalmazással A FROM-ban

A természetes összekapcsolás általános alakja:

SELECT [ALL | DISTINCT] *mezőnév*$_1$ [,...,*mezőnév*$_n$] | *
FROM *táblanév*$_1$
NATURAL JOIN *táblanév* $_2$ |**NATURAL JOIN** *táblanév*$_2$ **USING** (*mezőlista*$_1$)**...**
NATURAL JOIN *táblanév*$_n$ | **NATURAL JOIN** *táblanév*$_n$ **USING** (*mezőlista*$_{n-1}$);

Példa-1:

SELECT *
FROM hallgató
NATURAL JOIN vizsga;

Példa-2:

SELECT *
FROM hallgató
NATURAL JOIN vizsga **USING (**leckekönyvszám);

6. Tábla összekapcsolása saját magával (selfjoin) explicit megfogalmazással A FROM-ban

A selfjoin alakja:

SELECT [ALL | DISTINCT] *mezőnév*$_1$ [,...,*mezőnév* $_n$] | *
FROM *táblanév*$_1$ **AS** *1*
[**INNER**] **JOIN** *táblanév*$_2$ **AS** *2* **ON** *összekapcsolási_feltétel*$_1$ |**USING** (*mezőlista*$_1$) **...**
[**INNER**] **JOIN** *táblanév*$_n$ **AS** *n* **ON** *összekapcsolási_feltétel*$_{n-1}$ |**USING** (*mezőlista*$_{n-1}$);

A reláció, amelyre a selfjoin alkalmazásra kerül:

Dolgozó(<u>személyi_szám</u>, vezetéknév, név, *főnök_személyi_száma*)

Példa

```
SELECT  db.személyi_szám, db.vezetéknév, db.név, db.főnök_személyi_száma,
          df. személyi_szám, df.vezetéknév, df.név
        FROM dolgozó db
        JOIN dolgozó df ON db.főnök_személyi_száma=df.személyi_szám;
```

A lekérdezés listázza a dolgozó(beosztott-rövid név db/dolgozó-beosztott) személyi számát, vezetéknevét, nevét, a főnöke személyi számát a beosztott dolgozó sorából, valamint a főnök személyi számát, vezetéknevét és nevét a főnök sorából (a tábla df rövid néven – dolgozó-főnök – kezeli ugyanazt a dolgozó táblát).

7. Külső összekapcsolások explicit megfogalmazással A FROM-ban

A külső összekapcsolások általános alakja:

SELECT [ALL | DISTINCT] *mezőnév*$_1$ [,...,*mezőnév*$_n$] | *
FROM *táblanév*$_1$
LEFT|**RIGHT** | **FULL** [**OUTER**] **JOIN** *táblanév*$_2$ **ON** *összekapcsolási_feltétel*$_1$
|**USING** (*mezőlista*$_1$) **...**
LEFT|**RIGHT** | **FULL** [**OUTER**] **JOIN** *táblanév*$_n$ **ON** *összekapcsolási_feltétel*$_{n-1}$
|**USING** (*mezőlista*$_{n-1}$);

Semantika:

- **LEFT [OUTER] JOIN** – a baloldali külső összekapcsolás a bal oldalon lévő (a FROM kulcsszó után feltüntetett) tábla minden sorát listázza, függetlenül attól, hogy kapcsolható-e sor a jobboldali (a JOIN kulcsszó után felírt) táblából. Kapcsolható sor hiányában NULL értékekkel tölti fel a jobboldali tábla attribútumait.
- **RIGHT [OUTER] JOIN** – a jobboldali külső összekapcsolás a jobb oldalon lévő (a JOIN kulcsszó után felírt) tábla minden sorát listázza, függetlenül attól, hogy

kapcsolható-e sor a baloldali (a FROM kulcsszó után feltüntetett) táblából. Kapcsolható sor hiányában NULL értékekkel tölti fel a baloldali tábla attribútumait.

- **FULL [OUTER] JOIN** – a teljes külső összekapcsolás mindkét tábla minden sorát listázni fogja (függetlenül attól, hogy kapcsolható-e hozzá sor a másik oldalról) és a hiányzó attribútumokat NULL értékkel tölti fel.

Példa-1

```
SELECT h.*, v.*
    FROM hallgató h
    LEFT OUTER JOIN vizsga v ON h.leckekönyvszám=v. leckekönyvszám;
```

A hallgazó és vizsga reláció össz attribútumát/attribútumértékét listázza, azzal, hogy minden hallgató megjelenik az eredménytáblában. Azoknál, akiknek nincs iktatott vizsgájuk a vizsga táblában, NULL értékű lesz a vizsga tábla minden attribútuma.

Példa-2

```
SELECT h.*, v.*
    FROM hallgató h
    LEFT OUTER JOIN vizsga v ON h.leckekönyvszám=v. leckekönyvszám
    WHERE jegy IS NULL;
```

Azokat a hallgatókat listázza az eredménytáblában, akiknek nincs osztályzatuk (jegyük).

Példa-3 – a Példa-1 RIGHT OUTER JOIN-nal

```
SELECT h.*, v.*
    FROM vizsga v
    RIGHT OUTER JOIN hallgató h ON v. leckekönyvszám = h.leckekönyvszám;
```

Példa-4 – a Példa-2 RIGHT OUTER JOIN-nal

```
SELECT h.*, v.*
    FROM vizsga v
    LEFT OUTER JOIN hallgató h ON h.leckekönyvszám=v. leckekönyvszám
    WHERE jegy IS NULL;
```

4.3.8. SELECT-ek (eredménytábláinak) összekapcsolása

A relációs algebra alapvető halmazműveletei, az egyesítés (unió-UNION), a metszet (INTERSECT) és a különbség (EXCEPT) esetenként nagyon hasznosak lehetnek bizonyos lekérdezések esetén, viszont csak a megegyező attribútumhalmazú relációkra érvényesíthetők (a CORRESPONDING ezt a korlátot oldja fel). Az összekapcsolás általános alakja:

SELECT ...
UNION | INTERSECT | EXCEPT [**ALL | DISTINCT**] [**CORRESPONDING |**
CORRESPONDING (*közös_attribútumok_listája*)]
SELECT ...
[**UNION | INTERSECT | EXCEPT** [**ALL | DISTINCT**] [**CORRESPONDING |**
CORRESPONDING (*közös_attribútumok_listája*)]
SELECT ...]

A két SELECT mondat közé írt halmazművelet tulajdonképpen a két lekérdezés eredménytábláját érinti, és a kért halmazművelet után egy eredménytáblát ad. Egyesek az ilyen lekérdezéseket (SELECT-eket), amelyek egy másik lekérdezés részét képezik alkérdéseknek nevezik. Ha nem kettő, hanem több lekérdezés (alkérdés) szerepel az összekapcsolásban, akkor minden újabb lépésben az előző eredménytáblához kapcsolódik a következő konkrét SELECT eredménytáblája. A végeredmény mindig csak egy reláció (eredménytábla) lesz. Az UNION|INTERSECT|EXCEPT leírásában adatbáziskezelő rendszerenként különböző megszorítások vannak. Egyes rendszerek azonos sorrendű, megegyező nevű és típusú attribútumhalmazt írnak elő az összekapcsolás elvégzésének feltételeként, mások csak a sorrendet és a típust jelölik szükségszerűnek, ismét mások pedig a CORRESPONDIG kulcsszóval teszik lehetővé a különböző attribútumhalmazú (nemkompatibilis) eredménytáblák összekapcsolását. Nagyjából egységes a DISTINCT kulcsszó (amely az ismétlődésmentességet bizosítja) alapértelmezett volta. A sorok ismétlődését az eredménytáblában az ALL kulcsszó teszi lehetővé. A CORRESPONDING kulcsszó hatására a két különböző attribútumhalmazú eredménytáblának az összekapcsolható attribútumai (az összes kapcsolható attribútum) kerülnek az eredménytáblába, a „**CORRESPONDING** (*közös_attribútumok_listája*)“ kifejezés esetén pedig csak a közös attribútumok listájában szereplő attribútumok.

Az adatbázisséma a relációsémák és a megszorítások hallmazából áll:

A= { Hallgató (<u>leckekönyvszám</u>, hallgatónév, neme, születésiév, lakcím-város, lakcím, telefon)
Tárgy (<u>tárgykód</u>, tárgynév, heti_óraszám, kredit)
Vizsga (<u>*leckekönyvszám*, *tárgykód*</u>, <u>dátum</u>, jegy) }

M= { Vizsga[leckekönyvszám] $\subseteq$ Hallgató[leckekönyvszám]
Vizsga[tárgykód] $\subseteq$ Hallgató[tárgykód] }

Példa

SELECT *.leckekönyvszám*
FROM hallgató
EXCEPT
SELECT *.leckekönyvszám*
FROM *vizsga*;

Az eredménytábla azokat a hallgatókat (leckekönyvszám) listázza, akiknek nincsenek nyilvántartott adataik egyetlen vizsgáról sem (a példa OUTER JOIN-nal is, de a későbbiekben látható lesz majd hogy beépített SELECT-tel is megoldható).

4.3.9. Alkérdések vagy beépített SELECT-ek

Az alkérdéseket három helyen, a WHERE, a HAVING és a FROM záradékban lehet alkalmazni, és az egyetlen megszorítás az alkérdésekre nézve, hogy nem tartalmazhatnak csoportosítást (GROUP BY záradékot). Az alkérdés ugyancsak tartalmazhat beépített SELECT-et vagy alkérdést (egyes források szerint nyolc szintig, mások szerint a megkívánt mélységig). Az alkérdéseket zárójelbe kell tenni. A lekérdezés végrehajtása a legbelső alkérdéstől indul és kifelé folytatódik a beágyazás fordított sorrendjében. Az alkérdések a következő két ismérv szerint kerülnek csoportosításra:

1. az alkérdés eredménytáblájának felépítése
2. az alkérdés végrehajtásának száma

4.3.9.1.*Az alkérdés eredménytáblájának felépítése*

Az alkérdés eredménytáblájának felépítése alapján az alábbi három alkérdés fajtát eredményezi:

1. az alkérdés egysoros, egyoszlopos eredménytáblát ad (ezek a skalár értéket adó alkérdések),
2. az alkérdés többsoros, egyoszlopos eredménytáblát biztosít (ezek az egy attribútumos, többértéket-értékhalmazt, értéksort-visszaadó alkérdések),
3. az alkérdés többsoros, többoszlopos eredménytáblát biztosít (ezek a klasszikus táblát generáló alkérdések).

A skalár értéket visszaadó alkérdések a WHERE és/vagy a HAVING záradékban, a *feltétel*-ben használhatók az egyszerű összehasonlítást végző logikai operátorok (=, <, >, <> [NOT =], ≤ [NOT >], ≥ [NOT <]) alkalmazásával.

Példa

SELECT *leckekönyvszám*
FROM vizsga
WHERE jegy=(**SELECT MAX**(jegy) **FROM** vizsga);

Az eredménytábla azokat a hallgatókat (leckekönyvszám) listázza, akik a nyilvántartásban (vizsga tábla) a legnagyobb osztályzatok kapták.

Értéksort/értékhalmazt visszaadó alkérdések esetében az előzőekben felsorolt, összehasonlítást végző logikai operátorok nem, vagy csak az egyes alábbi operátorokkal közösen/párosítva alkalmazhatók. Helyettük a következő operátorok alkalmazása megengedett: **IN**, **ANY**, **ALL** és **EXISTS** (a **NOT** kulcsszóval párosítva is), értelmezésük pedig a következő:

- *érték* **IN** (*alkérdés*) – igaz értékű, ha az *érték* egyenlő valamelyik alkérdés által visszaadott értékkel
- *érték* **NOT IN** (*alkérdés*) – igaz értékű, ha az *érték* nem egyenlő egyetlen alkérdés által visszaadott értékkel sem (megegyezik az „*érték* <> **ALL** (*alkérdés*)” megfogalmazással)
- *érték* = **ANY** (*alkérdés*) – igaz értékű, ha az *érték* egyenlő valamelyik alkérdés által visszaadott értékkel (ugyanaz, mint az „*érték* **IN** (*alkérdés*)”)
- *érték* <> **ANY** (*alkérdés*) – igaz értékű, ha az *érték* különbözik az alkérdés által visszaadott bármelyik értéktől (az alkérdés által visszaadott értékhalmazban van legalább egy *érték*-től eltérő halmazelem/érték)
- *érték* > **ANY** (*alkérdés*) – igaz értékű, ha az *érték* nagyobb az alkérdés által visszaadott bármelyik értéktől
- **NOT** *érték* > **ANY** (*alkérdés*) – igaz értékű, ha az *érték* nem nagyobb az alkérdés által visszaadott egyetlen bármelyik értéktől sem
- *érték* > **ALL** (*alkérdés*) – igaz értékű, ha az *érték* nagyobb az alkérdés által visszaadott értékhalmaz mindegyik értékétől
- **NOT** *érték* > **ALL** (*alkérdés*) – igaz értékű, ha az *érték* nem nagyobb az alkérdés által visszaadott mindegyik értéktől
- **EXISTS** (*alkérdés*) – igaz értékű, ha az alkérdés által visszaadott értékhalmaz nem üreshalmaz
- **NOT EXISTS** (*alkérdés*) – igaz értékű, ha az alkérdés által visszaadott értékhalmaz üreshalmaz

Példa1 – **IN** operátor

```
SELECT hallgatónév
   FROM hallgató
   WHERE leckekönyvszám IN (SELECT leckekönyvszám
                            FROM vizsga
                            WHERE tárgykód=(SELECT tárgykód
                                            FROM predmet
                                            WHERE tárgynév="Matematika1"));
```

A fenti lekérdezés azon hallgatók neveit listázza, akiknek Matematika1 tárgyból van vizsgajegyük.

Megjegyzés: Amennyiben a Tárgy táblában egynél több "Matematika1" nevű tárgy létezik (a tárgynév attribútum legalább két előfordulás-értéke "Matematika1"), eltérő tágykód értékkel, akkor az első szintű alkérdés **WHERE** záradékában nem egyenlőségjelet (=), hanem **IN** záradékot kell alkalmazni:

```
SELECT hallgatónév
   FROM hallgató
   WHERE leckekönyvszám IN (SELECT leckekönyvszám
                            FROM vizsga
                            WHERE tárgykód IN (SELECT tárgykód
                                               FROM predmet
                                               WHERE tárgynév="Matematika1"));
```

Példa2 – **ANY** operátor – a Példa1 megoldása **ANY** operátorral

```
SELECT hallgatónév
   FROM hallgató
   WHERE leckekönyvszám = ANY (SELECT leckekönyvszám
                               FROM vizsga
                               WHERE tárgykód = ANY (SELECT tárgykód
                                                     FROM predmet
                                                     WHERE tárgynév="Matematika1"));
```

Példa3 – **ANY** operátor

```
SELECT1 hallgatónév
   FROM hallgató
   WHERE leckekönyvszám IN (SELECT2 leckekönyvszám
            FROM vizsga
            WHERE tárgykód IN (SELECT3 tárgykód
                               FROM tárgy
                               WHERE tárgynév = "Matematika1") AND
                  jegy>ANY (SELECT4 jegy
                            FROM vizsga
                            WHERE tárgykód IN (SELECT5 tárgykód
                                               FROM tárgy
                                               WHERE tárgynév="Matematika1")
                            AND leckekönyvszám IN (SELECT6 leckekönyvszám
                                                   FROM hallgató
                                                   WHERE születésiév=1980)));
```

A fenti lekérdezés azoknak a hallgatóknak a neveit jeleníti meg, akik Matematika1 tárgyból magasabb osztályzatot kaptak az 1980-ban született hallgatók által Matematika1 tárgyból megszerzett osztályzatok legalább egy (legkisebb) értékénél.

A **SELECT$_1$** listázza a hallgatók neveit a hallgató táblából. Csak azok a hallgatók kerülnek listázásra akiknek a leckekönyvszámát a **SELECT$_2$** kiszűri. A **SELECT$_2$** szűrője (**WHERE**) két ismérv szerint szűr: **1)** a tárgynak „Matematika1"-nek kell lennie (mivel a vizsga tárgyban nincs nyilvántartva a tárgynév, ezt a feladatot a **SELECT$_3$** alkérdés végzi el a **SELECT$_2$** alkérdés számára), és **2)** az osztályzatnak nagyobbnak kell lennie az 1980-ban született hallgatók által Matematika1 tárgyból megszerzett osztályzatok legalább egy (legkisebb) értékénél (az 1980-ban született hallgatók által Matematika1 tárgyból megszerzett osztályzatok halmazát a **SELECT$_4$** biztosítja a vizsga táblából, de hogy a Matematika1 tárgyból legyenek az osztályzatok, azt a **SELECT$_5$** , a **SELECT$_4$** alkérdése biztosítja a tárgy táblából, annak biztosításáért viszont, hogy az osztályzatokat 1980-ban született hallgatók kapták, a **SELECT$_6$**, ugyancsak a **SELECT$_4$** alkérdés a felelős.

Megjegyzés: A fenti lekérdezés megoldható az **ANY** operátor alkamlazása nélkül is, mivel az az elvárás, hogy a keresett osztályzat nagyobb (jobb) legyen „az 1980-ban született hallgatók által Matematika1 tárgyból megszerzett legalább egy (vagyis a megszerzett osztályzatok kötül a legkisebb osztályzat) értékénél. A **MIN** összegző függvényt tartalmazó **SELECT** mondat a következő:

```
SELECT1 hallgatónév
   FROM hallgató
   WHERE leckekönyvszám IN (SELECT2 leckekönyvszám
            FROM vizsga
            WHERE tárgykód IN (SELECT3 tárgykód
                                 FROM tárgy
                                 WHERE tárgynév = "Matematika1") AND
                     jegy > (SELECT4 MIN(jegy)
                                 FROM vizsga
                                 WHERE tárgykód IN (SELECT5 tárgykód
                                                      FROM tárgy
                                                      WHERE tárgynév="Matematika1")
                                 AND leckekönyvszám IN (SELECT6 leckekönyvszám
                                                          FROM hallgató
                                                          WHERE születésiév=1980)));
```

Az adatbázisséma:

A= { Hallgató (<u>leckekönyvszám</u>, hallgatónév, neme, születésiév, lakcím-város, lakcím, telefon)
Tárgy (<u>tárgykód</u>, tárgynév, heti_óraszám, kredit)
Vizsga (<u>*leckekönyvszám*, *tárgykód*, dátum</u>, jegy) }

M= { Vizsga[leckekönyvszám] ⊆ Hallgató[leckekönyvszám]
Vizsga[tárgykód] ⊆ Hallgató[tárgykód] }

Példa4 – **ALL** operátor

```
SELECT1 hallgatónév
   FROM hallgató
   WHERE leckekönyvszám IN (SELECT2 leckekönyvszám
            FROM vizsga
            WHERE tárgykód IN (SELECT3 tárgykód
                                 FROM tárgy
                                 WHERE tárgynév = "Matematika1") AND
                        jegy>ALL (SELECT4 jegy
                                 FROM vizsga
                                 WHERE tárgykód IN (SELECT5 tárgykód
                                                    FROM tárgy
                                                    WHERE tárgynév="Matematika1")
                                 AND leckekönyvszám IN (SELECT6 leckekönyvszám
                                                        FROM hallgató
                                                        WHERE születésiév=1980)));
```

A fenti lekérdezés azoknak a hallgatóknak a neveit jeleníti meg, akik Matematika1 tárgyból jobb osztályzatot kaptak az 1980-ban született hallgatók által Matematika1 tárgyból megszerzett osztályzatok mindegyikénél.

Megjegyzés: Ez a lekérdezés megoldható az **ALL** operátor alkamlazása nélkül is, mivel ha az a kikötés, hogy a keresett osztályzat nagyobb (jobb) legyen „az 1980-ban született hallgatók által Matematika1 tárgyból megszerzett osztályzatok mindegyikénél”, akkor ez úgy is megfogalmazható, hogy az osztályzatnak nagyobbnak kell lenni, mint a legnagyobb osztályzat, amelyet az 1980-ban született hallgatók közül Matematika1 tárgyból bárki is kapott. Az utóbbi értelmezésre írt **SELECT** mondat a következő:

```
SELECT1 hallgatónév
   FROM hallgató
   WHERE leckekönyvszám IN (SELECT2 leckekönyvszám
            FROM vizsga
            WHERE tárgykód IN (SELECT3 tárgykód
                                 FROM tárgy
                                 WHERE tárgynév = "Matematika1") AND
                        jegy > (SELECT4 MAX(jegy)
                                 FROM vizsga
                                 WHERE tárgykód IN (SELECT5 tárgykód
                                                    FROM tárgy
                                                    WHERE tárgynév="Matematika1")
                                 AND leckekönyvszám IN (SELECT6 leckekönyvszám
                                                        FROM hallgató
                                                        WHERE születésiév=1980)));
```

Példa5 – **EXISTS** operátor

```
SELECT hallgatónév
   FROM hallgató
   WHERE EXISTS (SELECT *
                 FROM vizsga
                 WHERE tárgyazon IN (SELECT tárgyazon
                                     FROM tárgy
                                     WHERE tárgynév = "Matematika1") AND
                   leckekönyvszám=hallgató.leckekönyvszám AND
                   jegy>5);
```

A lekérdezés azoknak a hallgatóknak a neveit listázza, akik letették a "Matematika1" tárgyat.

4.3.9.2.*Az alkérdés végrehajtásának száma*

Az alkérdések témakörének bevezetésében arról volt szó, hogy először az alkérdés hajtódik végre a külső, fő lekérdezés, amely az alkérdést tartalmazza. Ennek a megfogalmazásnak csak akkor van igazságértéke, ha a az alkérdés csupán egyszer kerül végrehajtásra. A végrehajtások száma szerint is lehet osztályozni az alkérdéseket:

1. egyszer kiértékelt (egyszerű) alkérdések és
2. korrelált (többször végrehajtásra kerülő) alkérdések.

Az egyszerű alkérdések esetében az először végrehajtásra kerülő (legbelső) alkérdés eredménytábláját a külső (tőle eggyel magasabb szintű, külsőbb) SELECT használja.

Az alkérdés többszöri végrehajtását paraméterátadás okozza a két lekérdezés között. Az alkérdés tartalmazza az őt beágyazó SELECT-ben feltüntetett attribútumot, amelynek értékváltozása kiváltja a belső SELECT (alkérdés) végrehajtását.

Példa

```
SELECT tárgynév, jegy, hallgatónév, leckekönyvszám
      FROM tárgy t
      JOIN vizsga vk ON vk.tárgykód=t.tárgykód
      JOIN hallgató h ON h.leckekönyvszám=vk.leckekönyvszám
      WHERE jegy=(SELECT MAX(jegy)
                  FROM vizsga vb
                  WHERE
vb.taargykooaleckekönyvszám=vk.taargykoodleckekönyvszám);
```

A lekérdezés a tárgynevet, a tárgyból a nyilvántartásban szereplő legmasabb ostályzatot és a hallgató(ka)t (névvel és leckekönyvszámmal) listázza. Valahányszor megváltozik a külső SELECT sorában a leckekönyvszám, az alkérdés kiértékelésre (végrehajtásra kerül).

4.4. ADATVEZÉRLÉSI/ADATFELÜGYELETI RÉSZNYELV/FUNKCIÓK (DCL)

Az adatvezérlési/adatfelügyeleti funkciók a jogosultságok odaítélését és azok meg(vissza)vonását végzik. A jogosultságok egy adatbázistábla különböző adatmanipulációs műveletére vonatkoznak/vonatkozhatnak és egy konkrét, vagy egész felhasználói csoportot érinthetnek. Az adatbázis nagyszámú felhasználója nem használhatja (láthatja, módosíthatja, törölheti) egységesen az össz rögzített adatot, így hát szükségszerű egy olyan mechanizmus, amely kezelni fogja a hozzáférések és az adatkezelési műveletek irányítását. A felhasználók legnagyobb része csak alkalmazáson keresztül férhet a számára hozzáférhető adatokhoz, így azokat nem kell külön kezelni. A hozzáértőbb felhasználók, azonban, esetleg (kéretlenül is) közvetlenül (alkalmazáson kívül) is megpróbálhatnak hozzáférni az adatbázis adataihoz. Ez esetenként szükséges is lehet. Ilyenkor tesznek jó szolgálatot a DCL-funkciók, lehetővé téve bizonyos személyeknek, a számukra megengedett adatmanipulációs műveletek elvégzését, a számukra hozzáférhető táblákon. A hozzáférési és műveletvégzési jogosultságok az adatbázis tetszőleges objektumára megfogalmazhatók, így a táblákra is.

A jogosultságok kezelése a táblák esetében a következő lehetőségekkel (külön műveleti megszorításokkal) rendelkezik:

1. SELECT – adatok lekérdezése a táblából (adat-láthatóság),
2. INSERT – adatok beszúrása a táblába,
3. DELETE – adatok törlése a táblából és
4. UPDATE – a tábla adatainak módosítási lehetősége.

Minden iktatott adatbázis-felhasználónak van felhasználói neve és jelszava. A jogosultsággal kapcsolatos utasítások vagy a konkrét felhasználókat (azok felhasználói nevét) vagy a felhasználók csoportjának nevét tartalmazzák. A tábla létrehozója és az adatbázis adminisztrátora minden jogosultsággal rendelkezik a tábla felett (az adminisztrátor mindegyik tábla felett).

A jogosultságok hozzárendelését a GRANT utasítás biztosítja, melynek formája a következő:

GRANT ALL PRIVILEGES | *jogosultság*
ON *táblanév*
TO PUBLIC | *felhasználó* [WITH GRANT OPTION];

A *jogosultság* paraméter értéke a fenti négy operáció kerül ki (egy, de akár több operációt is fel lehet sorolni, vesszővel elválasztva őket): SELECT, INSERT, DELETE és UPDATE. A *felhasználó* egy vagy vesszőkkel elválasztott több felhasználói nevet tartalmazó opció. A PUBLIC kulcsszó mindenki (az összes adatbázishasználatra iktatott, felhasználói névvel és jelszóval rendelkező) számára biztosítja a *jogosultság* paraméterben felsorolt (az ALL PRIVILEGES jogosultság esetén az össz jogosultságot biztosító) lehetőségeket. Amennyiben az utasítás végén szerepel a WITH GRANT OPTION kifejezés, úgy a feltüntetett felhasználó/k, más felhasználó/k számára, a saját jogosultsági lehetőségeiket egy GRANT utasítással „tovább adhatják).

A jogosultság megvonása a REVOKE utasítással történik, amely törölheti (vagy csak megnyirbálhatja, bizonyos jogosultságot/kat kiiktatva a jogosultsági listából) az össz jogosultságot egy vagy több felhasználóra és egy adott táblára nézve. A REVOKE alakja:

REVOKE ALL PRIVILEGES | *jogosultság*
    ON *táblanév*
    FROM PUBLIC | *felhasználó*;

5. ADATMODELLEK

Az információs rendszerek működéséhez/működtetéséhez már régóta adatbázisokat alkalmaznak. Az adatbázisok megtervezéséhez, kialakításához és megvalósításához többfajta adatmodell használata/alkalmazása szükséges. A relációs adatmodell már ismertetésre került a harmadik fejezetben, mert a tárgy programja/felépítése így kívánja, didaktikailag azonban helytelen a megközelítés, hiszen az általánostól szokás a konkrét lényegek felé haladni a fogalmak magyarázatában a felsőoktatásban. Ezt pótlandó, most az adatmodelleknek a szükségleteknek megfelelő teljes bemutatása következik.

5.1. AZ ADATMODELLEK FORMÁLIS MEGKÖZELÍTÉSE

Az adatmodell az információs rendszerrel támogatni kívánt valós rendszer adatainak matematikai leképzése, az adatok leírására megfogalmazott jelölés, jelrendszer/jelölésrendszer. Az adatmodellek az adatok leírását három jól elkülöníthető részben valósítják meg:

1. strukturális rész (szerkezeti, statikus leírások)
2. megszorítások leírása (integritási rész/komponens) és
3. műveleti rész (operációs/a dinamikus tulajdonságokat leíró komponens)

Az adatmodell strukturális része (strukturális komponense) az adatszerkezet leírását tartalmazza. A két alapfogalomra épülő egyre összetetteb fogalmak képezik a szerkezet-leírás alapját. A valós rendszer adatokkal leírni kívánt lényegeit fogalmi szinten egyedtípusnak, egyednek (entitásnak-entity) nevezik. Egyedtípus elvileg bármi lehet (szubjektum, objektum, fogalom, elképzelés, esemény, vagy ezeknek egy csoportja), amit adatokkal kell és lehet jellemezni/leírni. A róluk szóló (a leírásukra előrelátott) adatokat fogalmi szinten tulajdonságtípusnak, tulajdonságnak (attribútumnak, sz. obeležje) nevezik. A strukturális rész harmadik pillérét az adatok közötti, pontosabban az egyedtípusok közötti kapcsolatok fogalmi szintjét képező kapcsolattípusok képezik.

A strukturális rész két alapfogalma a következő:

- tulajdonságtípus (az egyedtípusok valamilyen fontos jellemzőét leíró ismérv – obeležje) és
- értékhalmaz – doméjn (egy-egy tulajdonságtípus előfordulási értékeit tartalmazó halmaz – domen).

A szerkezeti rész összetett fogalmai az alábbiak:

- egyedtípus
- kapcsolattípus és
- adatbázisséma.

A fenti fogalmak két absztrakciós szintű modellhalmaz kiépítését teszik lehetővé: az egyedtípus modellhalmazét és az egyedelőfordulási modellhalmazét. Az egyedtípus modell az egyedtípus nevéből és az őt leíró tulajdonságtípus halmazból áll. Az egyedelőfordulási modell a tulajdonságtípusok konkrét értékeit tartalmazza egy vagy több konkrét egyed (egyedtípus-előfordulás) számára.

Példa. A személygépkocsi és a személy egy-egy lehetséges egyedtípus modellje (egyedtípusa) látható a következő sorban.

SZEMÉLYGÉPKOCSI(rendszám, alvázszám, gyártó, típus, szín).
SZEMÉLY(személyi_szám, név, vezetéknév)

Példa. Az előzőekben felírt SZEMÉLYGÉPKOCSI egyedtípus modellhez illeszkedő személygépkocsi egyedelőfordulási modellje a következőképpen néz ki (szerb eredetiben, a régi forgalmi engedély alapján):

(SU 372-66,TMBEEA614TO334911,Škoda, Felicija GLS, trula višnja).

A valós rendszerben a leírni kívánt fontos dolgok (lényegek, entitások) között különféle kapcsolatok lehetnek. Az egyedtípusok közötti összefüggéseket kapcsolattípusoknak, az egyedelőfordulások közötti viszonyokat kapcsolatoknak nevezik.

Példa. A SZEMÉLY és a SZEMÉLYGÉPKOCSI közötti kapcsolattípus megfogalmazási formája a következő:

TULAJDONOS(SZEMÉLY, SZEMÉLYGÉPKOCSI).

Példa. A konkrét személy és a tulajdonában lévő személygépkocsi között fennálló konkrét kapcsolat leírása pedig a következőképpen néz ki:

((1234,Imre Petković), (SU 372-66,TMBEEA614TO334911,Škoda Felicija GLS, trula višnja).

A fentebb említett és a példákban is kihangsúlyozott eltérő absztrakciós szintet gyakran hívják intenziónak és extenziónak. Az intenzió fogalma alatt egy halmaz definíciós leírását értik (lásd SZEMÉLY egyedtípus), az extenzió pedig az előfordulás-halmaz legalább egy konkrét előfordulását (lásd SZEMÉLYGÉPKOCSI előfordulási modell).

Az adatmodell integritási része a valós rendszer diktálta megszorításokat, illetve azok leírását tartalmazza. A relációs meodell ismertetésekor ezek a korlátok az üzletszabályzati megszorítások csoportjban jelentek meg:

- doméjnértékekre vonatkozó megszorítások,
- a tulajdonságtípus értékhalmazára vonatkozó korlátok,

- a konkrét egyedtípus-előfordulás tulajdonságértékeinek doméjn-értékhalmazból való leképzésének megszorításai,
- a kapcsolattípusokra és kapcsolatokra vonatkozó megkötések.

A különböző megszorításokat az adatbázistervezés folyamatában csak integritási feltételeknek (sértetlenségi feltételek) nevezik. A fenti megszorítások egy leírási formája az alábbi példákban.

Példa – a doméjnértékekre és a tulajdonságértékekre vonatkozó megszorításokra:

$d_1 \leq D \leq d_2$, illetve $t_1 \leq T \leq t_2$

Itt a doméjnértékek és a tulajdonságértékek értékkorlátozását lehet tetten érni.

Példa – a konkrét egyedtípus-előfordulás tulajdonságértékeinek doméjn-értékhalmazból való leképzésének megszorításokra:

T(oktató_születési_éve) ← dátum {(folyó_év-67) ≤ év ≤ (folyó_év- 22)}

A beírásra kerülő oktatónál az oktató_születési_éve tulajdonságtípus-értékek a dátum doméjnból merítődnek. A legfiatalabb iktatható oktató 22, a legidősebb 67 éves lehet.

Példa – a kapcsolattípusokra vonatkozó megkötésekre

TULAJDONOS(SZEMÉLY(0,N):SZEMÉLYGÉPKOCSI(1,1))

A példában felírt TULAJDONOS kapcsolattípusra vonatkozó megszorítás mindkét egyedtípus szemszögéből értelmezendö/értelmethető. A valós rendszerben nyilvántartott személyek közül nem feltétlenül kell mindenkinek autótulajdonosnak lennie, de a tulajdonos akár több személygépkocsival is rendelkezhet egyidőben. A személygépkocsik a nyilvántartásban mindenképpen személyhez kötötten jelenhetnek meg, viszont egyidőben csak egyetlen személy lehet a tulajdonosuk.

Példa – a konkrét kapcsolatokra vonatkozó megkötésekre

Álljon itt egy mondattal megfogalmazott megszorítás is a példának okáért. Kiss János mozgássérült személynek engedélyezett a speciális típusú (mozgássérültek számára készült) gépkocsi nyilvántartásba vétele.

Amennyiben az adatbázis adatai eleget tesznek a sértetlenségi (integritási) feltételeknek, akkor azt mondják rá, hogy konzisztens (ellentmodásmentes).

Az adatmodell műveleti része (a dinamikus tulajdonságokat leíró komponens) a használatban alkalmazható operációkat/műveleteket írja le. A műveletek végrehajtásakor az adatbázis állapota megváltozik. Ez nem jelenti szükségszerűen (automatikusan) azt is, hogy a

felhasználói adatokban változások történnek. Az adatbázis, ugyanis, a felhasználói adatok mellett nagyszámú metaadatot is tartalmaz (pl. hozzáférési jogosultságok az egyes táblákhoz). Figyelni kell a műveletek végrehajtásának következményeit és megtiltani azon műveletek végrehajtását, amelyek megsértik az integritási feltételeket (nem felelnek meg valamelyik megszorításnak) az adatbázis épségének megóvása érdekében.

A műveletek nagy általánosságban az adatbázis adatainak csak egy kis részét érintik, így szükségszerűen tartalmaznak egy adatprojekciós részt, amely kiválasztja/kijelöli azt az adathalmazt az adatbázisból, amelyre a művelet érvényesül (végrehajtódik). A művelethez tartozó adathalmaz három módon jelölhető ki:

1. az adatok egymás közötti viszonya alapján
2. a tulajdonságtípus értéke alapján
3. az aktuális sormutató értéke alapján (túlhaladott-nem használatos).

A műveleteknek öt fajtáját különböztetik meg:

1. adatok olvasása az adatbázisból
2. új adatok beszúrása/beírása
3. meglévő adatok törlése
4. adatok módosítása
5. az aktuális sormutató értékének beállítása (túlhaladott-nem használják).

Ha az adathalmaz kiválasztásában több szerkezeti elem (egyedtípus) is részt vesz, akkor navigációról van szó, ilyen esetben a kiválasztásokat deklaratív nyelv segítségével írják le. A deklaratív (nemprocedurális) nyelvek nem tartalmazzák azokat a (procedurális) műveleteket, amelyek a kiválasztás »hogyan«-jára vonatkoznak, csupán a kiválasztásra kijelölt adatokat és azok szerkezeti elemekbeni helyét (hol helyezkednek el) fogalmazzák meg.

Az adatmodellek kiépítésekor szükségszerű az absztrakció (elvonatkoztatás) alkalmazása. Az absztrakció egy intellektuális erőfeszítés/folyamat a valós rendszer leképzésekor keletkező összetett adatmodell-elemek közös és fontos tulajdonságainak feltárására és megnevezésére/elnevezésére, ugyanakkor a lényegtelen tulajdonságok elhanyagolására.

5.2. AZ ADATMODELLEK FEJLŐDÉSE

Az első adatmodellek a múlt század hatvanas éveiben jelentek meg. Az első adatmodell-próbálkozások hiányosságai motiválták a következő, jobb minőségű modellek megjelenését. Azóta sok adatmodell jelent meg, de csak az alábbiak bizonyultak maradandónak és

alkalmazhatónak az adatbázisok tervezésénél és alkalmazásánál/használatánál(Mogin Principi baza podataka):

1. hálós adatmodell,
2. hierarchikus adatmodell,
3. relációs adatmodell,
4. egyed-kapcsolat modell (egyed-attribútum-kapcsolat modell),
5. funkcionális adatmodell,
6. szemantikus hierarchiák modellje,
7. szemantikus adatmodell,
8. objektumorientált adatmodell
9. logikai adatmodell.

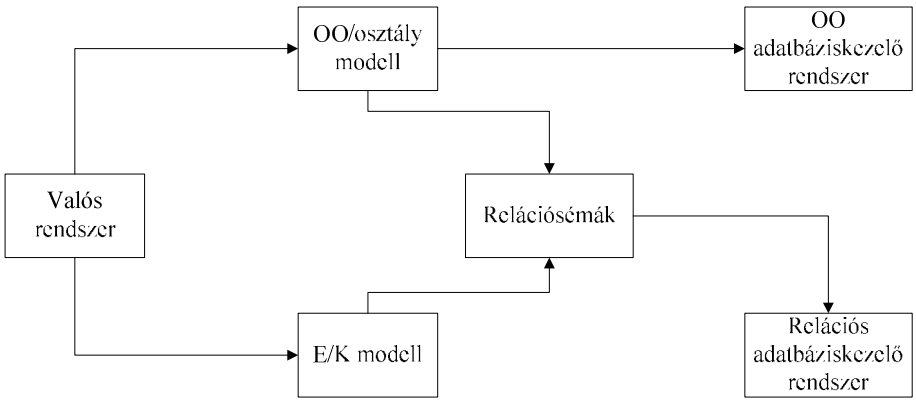
Az adatbázistervezés folyamán az egyed-kapcsolat modellt és az objektumorientált adatmodellt használják, a gyakorlati megvalósítás és alkalmazás terén pedig a relációs adatmodellt alkalmazzák.

A gazdag szemantikával (jelentéstartalommal) rendelkező adatmodellek elsősorban a tervezési folyamatban alkalmazhatók sikeresen, mert segítségükkel könnyen ábrázolhatók a valós rendszer adatainak ismérvei.

5.3.AZ ADATMODELLEK GYAKORLATI MEGKÖZELÍTÉSE

Gyakorlati szempontból elmondható, hogy nem mindegyik adatmodell alkalmazható a tervezésben és megvalósításban/használatban egyaránt. Azok az adatmodellek, amelyek kiválóan alkalmazhatók a tervezésben, bonyolultságuk miatt nem rendelkeznek saját adatbáziskezelő szoftverrel (számítógépen ezek az adatmodellek nem implementálhatók). A mai alkalmazások túlnyomó többsége relációs adatbázisokat használ, amelyek a relációs adatmodellen alapulnak, nagyon egyszerű szerkezetűek és szemantikailag sokkal szegényebbek. A tervezés eredményeképp kapott adatmodellek nem valósíthatók meg közvetlenül a számítógépeken, ezeket át kell alakítani relációs modellé. Egy gazdag jelentéstartalommal bíró adatmodellt egy egyszerűbb modellé. Az átalakítás nem túl bonyolult, az átalakítás maga, mára már automatizált is a CASE eszközökben. Az objektumorientált adatmodell ugyan rendelkezik számítógépen futó adatbátiskezelő rendszerrel (vannak már objetumos/objetumorientált adatbáziskezelő rendszerek), de teljesítményben még messze nem közelítik meg a relációs adatbáziskezelőket.

Az adatbázistervezés, –kialakítás, és a használat feljebb leírt folymatát a következő módon lehet ábrázolni (5.1. ábra).



5.1. ábra

Az adatbázistervezésnél az információs rendszerek tervezéséhez hasonlóan nem két, de háromfajta adatmodell alkalmazásáról kell beszélni. Ezt a háromfajta adatmodellt tulajdonképpen a következő három (adatokra is vonatkozó) absztrakciós szint kényszeríti ki:

1. fogalmi (konceptuális) szint,
2. logikai szint és
3. fizikai szint.

A fogalmi szinten a valós rendszer lényege, tartalma a legfontosab és ez kerül megfogalmazásra a fogalmi szintű adatmodellekkel: az egyed-kapcsolat modell-lel és az objektumorientált adatmodellel. A valós rendszert kiszolgáló megoldás a logikai és fizikai szinten kerül megvalósításra (dokumentálásra). A logikai szint fogalmán nagyjából az adatbáziskezelő rendszert kell érteni, a fizikai szinten pedig magát a vasat (a hardvert), amin az alkalmazások és a kiszolgáló szoftver fut. Mivel a három szint teljesen különválasztható, nem csak ajánlatos, de kötelező is őket külön dokumentálni. Egy cég életében az adatok változnak legkevésbé, a szoftverek gyakrabban, a számítógépes hardver pedig a három lényeg közül a leggyakrabban. Hardvercsere esetén így csak a fizikai szintre vonatkozó dokumentáció kerül felülvizsgálatra. A szoftverváltoztatás a logikai és fizikai szint újraelemzését-tervezését (reineneering) vonja maga után. A valós rendszer adatainak megváltozásakor pedig minden szinthez hozzá kell nyúlni. A fogalmi szintű adatmodell a valós rendszer adatainak és azok összefüggéseinek csobítatlan képét rögzíti. A logikai adatmodell kialakításánál az elemzők/tervezők a felhasználó/megrendelő igényeit tartva szem előtt lerontják a fogalmi modell által leírt hű képet a valós rendszerről, és kialakítanak egy a felhasználó szempontjából kívánatos logikai adatmodellt. A számítógépen ez az adatmodell általában csak átalakításokkal valósítható

meg, ami tovább rontja a már egyébként is (a logikai adatmodell által) lerontott képet a valós rendszerről.

5.4. AZ EGYED-KAPCSOLAT MODELL

Az egyed-kapcsolat modell (E/K modell) a múlt század hetvenes éveinek közepén P. P. Chen által kialakított adatmodellje (ER modell az angol Entity Relationship szavak rövidítése alapján). A grafikus formában (ábrán) dokumentált modell közérthető formában rögzíti a dolgokat és könnyen áttekinthető. Az E/K modell egyszerűsége egyben azt is jelenti, hogy a bonyolultabb helyzetekben hiányos dokumentációt nyújt. A model első leírásában csak a strukturális és az integritási rész (megszorítások leírása) kapott helyet, amit egy későbbi munkában pótoltak a műveleti résszel. A bonyolultabb esetek ábrázolhatóságát egy kiegészítés nyomán az E/K model bővítményei teszik lehetővé. A model szemantikus értelemben gazdag (kevés szimbólummal sok mindent leír), mindamellett, hogy nem tökéletes (Codd nem is tekintette igazi adatmodellnek, hanem csak egy adattervezést támogató modellnek). Az E/K modellben használt szimbólumok, jelölések nem egyöntetűek, inkább azt lehetne mondani, hogy igen változatosak, eltérőek (iskoláktól, szakirodalomtól függően), viszont könnyen megérthető a kisszámú jelölést tartalmazó szimbólumrendszer.

5.4.1. AZ E/K MODELL STRUKTURÁLIS RÉSZE

Az egyesek szerint két, mások szerint három alapelemre épülő modell hárompillérű: egyedet, tulajdonságot és kapcsolatot ír le.

Egyed, egyedhalmaz és egyedtípus. A valós rendszer adatokkal leírni kívánt fontos jelenségét, lényegét, szereplőjét egyednek nevezik. Az egyedeknek megkülönböztethetőeknek kell leniük. A hasonló jellegű, azonos tulajdonsággal/tulajdonságokkal rendelkező egyedek egyedtípust (egyedhalmazt) alkotnak. Egyedtípus minden lehet, amit adatokkal kell leírni: valós vagy képzelt személy, tárgy, esemény, jelenség vagy fogalom. Az egyedtípus tulajdonképpen a valós rendszerben fellelhető lényegek fogalmi képe. Az egyed pedig az egyedtípus egyedelőfordulása. Az egyedtípust téglalappal jelölik az E/K diagramon.

Tulajdonság, tulajdonságtípus, tulajdonság-érték (tulajdonság-értékhalmaz v. doméjn v. domain). A valós rendszer leírásra kerülő elemei különböző tulajdonságokkal rendelkeznek, melyeknek fogalmi tükörképét tulajdonságtípusnak nevezik. A tulajdonságtípus jele az ellipszis, vagy a lekerekített sarkú téglalap. Az azonos egyedtípusba sorolt egyedeknek

legalább egy közös tulajdonságtípussal kell rendelketniük. Az egyes egyedelőfordulás tulajdonságtípusai egy-egy tulajdonság-értéket vesznek fel. Maga a tulajdonságtípus így egy tulajdonság-értékhalmazból (doméjnből v. domain-ből) vesz fel értékeket. A tulajdonságtípus lehet egyszerű (amikor a tulajdonságtípust nem lehet vagy nem szükséges részekre bontani, pl. lakcím, amely irányítószám, helység, utca, házszám, bejárat, emelet, ajtó részekből áll) vagy összetett (amikor több egyszerű tulajdonságtípushoz külön megnevezést rendelnek). Nem egyértelmű, hogy mi lehet egyedtípus, és mi tulajdonságtípus (pl. szín – gépkocsié vagy szín – a festékgyárban). A tulajdonságtípust (vagy a tulajdonságtípusok halmazát), amely egyedi (különböző) értéket vesz fel minden egyedelőfordulásra azonosítónak (egyesek kulcsnak vagy kulcsjelöltnek) nevezik. Az azonosítót alkotó tulajdonságtípusok folytonos vonallal vannak aláhúzva (az ellipszisben vagy lekerekített sarkú téglalapban) az E/K ábrán/diagramon. Az azonosítóra (kulcsra) két megszorítást szabtak ki:

1. egyedi (különböző) értékeket vehet fel és
2. minimális felépítésű.

A csak az első feltételnek megfelelő atulajdonságtípus halmaz szuperkulcsnak tekinthető. A minimális felépítés azt jelenti, hogy a kulcs egyetlen valódi részhalmaza sem tölthet be kulcsszerepet (ennek a megszorításnak az elenőrzése csak több tulajdonságtípus alkotta kulcsok esetében van értelme. Ritkán fordul elő egy olyan természetes tulajdonság, amelynek értékei különböznek minden egyedelőforduláson. Ilyenkor vagy mesterségesen kialakított tulajdonságtípusokat használnak azonosítóként vagy addig gyűjtik a tulajdonságtípusokat egy halmazba, amíg a mérlegelések és becslések alapján már bizonyosan megfelel a fenti két megszorításnak. A leírtak kétfajta azonosítást tükröznek:

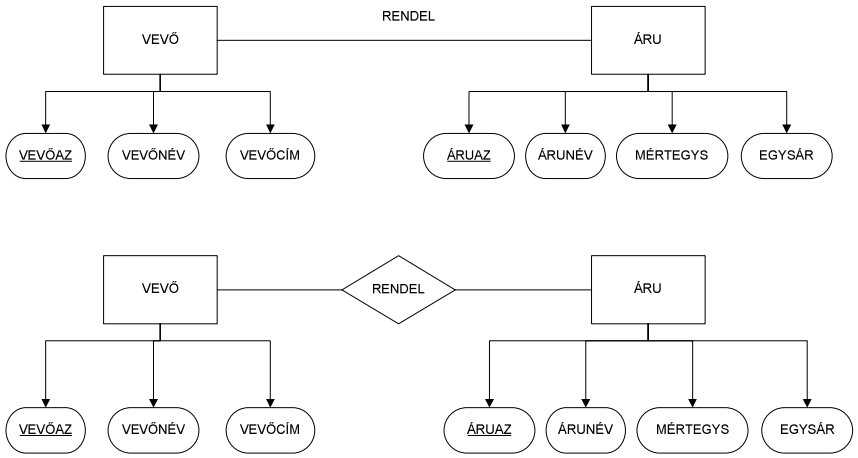
1. nominativ azonosítás (amikor egy, legtöbbször mesterségesen kialakított tulajdonságtípus az azonosító) és
2. deszkriptív azonosítás (amikor a természetes tulajdonságtípusok halmaza képezi az azonosítót).

A gyakorlatban előfordulhat, hogy több kulcsjelölt (azonosítójelölt) is megjelenik egy egyedtípusban. Ilyenkor, tetszőleges választás alapján az egyiket elsődleges kulcsnak (azonosítónak) választják.

Kapcsolat, kapcsolattípus. Az ismeretek intenzionális, extenzionális és transzverzális jellegének (vetületének) ismerete megkönnyíti a kapcsolatok és kapcsolattípusok megértését.

Az egyedtípus **intenzió**ja az egyedtípust/egyedtípusokat, és a hozzá/hozzájuk tartozó tulajdonságtípusokat tartalmazó vetület. Dokumentálni kétféleképpen lehet: kifejezéssel és grafikusan. Az $E(T_1,T_2,T_3,...,T_n)$ kifejezés intenzió. Az egyedtípus kialakítása során absztrakció

segítségével csak a fontos tulajdonságtípusok kerültek rögzítésre az E tulajdonságtípusról. Az intenzió grafikus ábrázolása az 5.2. ábrán látható.



5.2. ábra

Az egyedtípus **extenzionális jelleg**ét (vetületét) az egyedelőfordulások halmaza képezi. Dokumentálni az előforduláshalmaz elemeinek felsorolásával szokás (gyakran táblába/táblázatba helyezve az értékeket, lásd 5.1.Tábla). A tábla/táblázat rendszerint az egyedtípus (RENDELÉS) nevét is tartalmazza az oszlopnevekként feltüntetett tulajdonságtípusok mellett. Az extenziót tehát a tábla sorai (maguk a tulajdonságértékek) jelentik a fejléc nélkül. A fejléc viszont nem más, mint az intenzió.

RENDELÉS

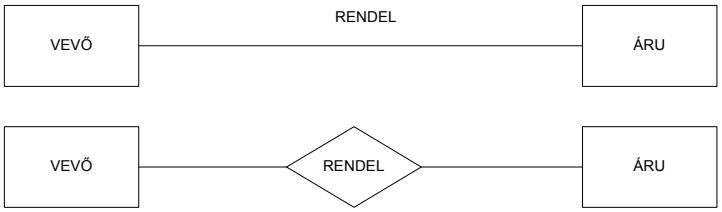
Rendelésazon	Vevőazon	Renddátum
17494	112	12.09.2000.
15944	147	15.09.2000.
13675	174	05.10.2000.
19395	174	14.07.2000.
25495	112	15.09.2000.
...	...	...

5.1. Tábla 1

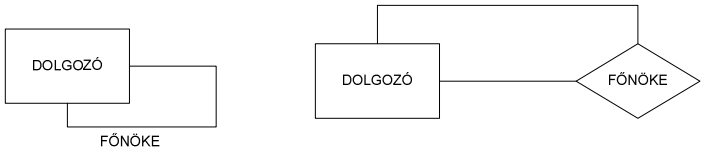
Transzverziónak az egyedtípusokban, illetve egyedekben rejlő konkrét ismeretek összekapcsolását hívják, illetve az ismeretváltást (az egyik egyedtípusban/egyedben található ismeretekről történő való áttérést a másik egyedtpusban/egyedben található ismeretekre) az egyedtípusok között.

A két konkrét egyedelőfordulás közötti ideiglenes vagy tartós viszonyt kapcsolatnak nevezik, a két egyedtípust pedig kapcsolatípus/kapcsolattípusok köthetik egymáshoz. Az E/K modellben használt kapcsolatnak asszociáció jellege van (bővebben erről később). A kapcsolat a legnehezebben felfogható fogalom az ismeretek világában, de érdemes megharcolni érte, mert ő a hatékony adatkezelés legfontosabb pillére. A két különböző egyedtípust összekapcsoló

kapcsolattípust inhomogén kapcsolattípusnak tekintik (5.3. ábra). A kapcsolattípus, azonban egy egyedtípust saját magával is összeköthet, ekkor a kapcsolattípusra azt mondják, hogy visszamutató, homogén vagy rekurzív (5.4. ábra). A kapcsolattípust is többféle jelöléssel illetik. A leggyakrabb jelölések a két egyedtípus közötti szakasz vagy rombusz. A szakasszal jelölt kapcsolattípus esetében a szakaszt ábrázoló vonal folytonos vagy szaggatott jellege utal a kapcsolat kötelező vagy opcionális/esetleges létére (kapcsolaterősség), a vonalvégek (nem elágazó vagy elágazó/varjúláb) pedig a kapcsolatfokra (egy vagy több). Ez utóbbi jelölésmódok a későbbiekben válnak majd teljesen érthetővé.



5.3. ábra



5.4. ábra

5.4.2. AZ E/K MODELL INTEGRITÁSI RÉSZE

Az E/K modell fogalma alatt a gyakorlatban csak az E/K diagramokat értik Ezeken a diagramokon csak egyedek és kapcsolatok (esetenként tulajdonságtípusok/attribútumok) jelennek meg. A modell fogalma messze túlnő a diagram fogalmán, és a diagramok mellett még az egyed- tulajdonság- és kapcsolattípusok, valamint kulcsok, külső kulcsok és doméjnek pontos szemantikus szöveges leírását is magába foglalja. Az E/K diagramokon a következő megszorítások ábrázolhatók:

1. kulcsmegszorítások/kulcsok
2. hivatkozásiépség megszorítások és
3. egyéb strukturális/kardinalitási/kapcsolatfoki/opcionalitás-megszorítások.

A kulcsok és a kulcsmegszorítások jelölése. Minden egyedtípusnak kell, hogy legyen kulcsa, függetlenől attól, hogy milyen egyedtípusról van szó (gyönge egyedtípus, egyedtípus az öröklési vagy »AZ EGY«-»IS A« hierarchiában – lásd később az E/K modell bővítményeinek leírásában). A kulcsot folytonos vonallal való aláhúzással jelölik. Ha kettő vagy több tulajdonságtípus rendelkezik aláhúzással, az nem több kulcsot, hanem összetett kulcsot jelent. Több kulcs (kulcsjelöltek) megjelölésére a grafikus megjelenítési forma korlátai miatt nincs lehetőség. A kulcsjelöltek közül kell választani egyet, és az az egy lesz az egyedtípus azonosítója (elsődleges kulcsa).

Strukturális/kardinalitási megszorítások az E/K modellben. A **hivatkozásiépség- és** egyéb más, **kapcsolat fokára és a kapcsolaterőre (a kapcsolat opcionalitására) vonatkozó megszorítások** bemutatása az alábbiakban kifejtésre kerülő, strukturális/kardinalitási megszorítások keretén belül fog megtörténni.

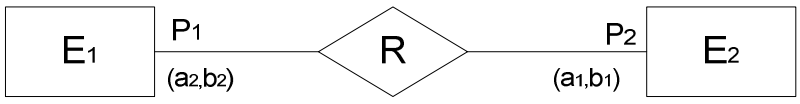
A kapcsolattípus legfontosabb legfontosabb jellemzőjeként a kapcsolat fokát (számossági jelleg szerinti típusfelosztását) tekintik, de nem szabad megfeledkezni a kapcsolaterőről sem a bináris kapcsolatban lévő egyedtípusokra vonatkozóan. Egyes vélemények szerint a kapcsolatokat adatokkal is jellemezni kell, de ezt sokan elvetik (ebben a szellemben készül ez a bemutatás is), mert az adatokkal jellemezni kívánt lényeget, az eredeti megegyezés szerint egyedtípusnak hívják. Ha egy kapcsolatot mégis szükséges adatokkal leírni, vagy a kapcsolatok között kell megfogalmazni kapcsolatot/kapcsolatokat, akkor ezek a kapcsolatok kapcsoló egyeddé/egyedtípussá (szerb nyelven gerund) változnak.

A **számossági jelleg szerint** a kapcsolatok **a)** 1:1, **b)** 1:N és **c)** M:N típusúak lehetnek (ez a kapcsolatba lépő elemek/egyedtípusok maximális értékére vonatkozó leírás/korlát/megszorítás). A **kapcsolaterőt** (**a kapcsolat opcionalitását**) **illetően** egy konkrét egyedtípus egy adott kapcsolatban **a)** opcionálisan (nem kötelezően) vagy **b)** totálisan (kötelezően) vesz részt. A mindkét jelleget dokumentáló jelölés bemutatása következik az alábbiakban.

A kapcsolattípus általános leírása a következő:

$$R\,(E_1\,(a_2\,,b_2):E_2\,(a_1\,,b_1))$$

A leírás magyarázatához készült grafikus ábrázolás a 4.5. ábrán látható.



Sl. 5.5.

Az előző kifejezésben az R a kapcsolattípus nevét jelöli, az E_1 és az E_2 az egyedeket (egyedtípusokat) képviselik, az a_1 és a_2 a minimális kardinalitás-értéket (ami az opcionalitás-jelleget tükrözi), a b_1 és b_2 pedig a maximális kardinalitás-értéket (ami a kapcsolat számossági jellegét mutatja) ábrázolják. Amennyiben csak általánosan kapcsolat-fokról van szó, akkor rendszerint a maximális kardinalitás-értékre kell gondolni. Minden kapcsolat/kapcsolattípus két leképezést takar. Az általános kapcsolatleírás alapján ez a két leképezés (a P_1 és a P_2) a következő:

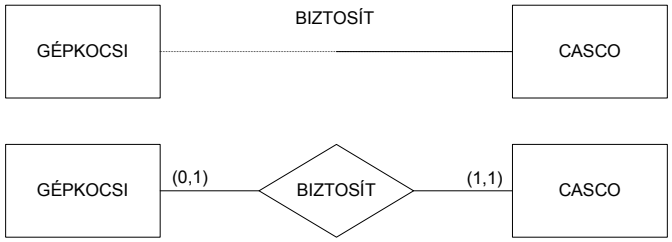
$$P_1 : E_1(a_2, b_2) \rightarrow E_2$$
$$P_2 : E_2(a_1, b_1) \rightarrow E_1$$

A leképzések értelmezése egyszerű.

Az első (P_1) leképzés értelmében az E_1 egyedtípus mindegyik egyedelőfordulásának legalább a_2 (minimális kardinalitás) számú egyedelőfordulással kell kapcsolatban lennie az E_2 egyedtípusból, persze attól több kapcsolatot is létesíthet, de a kapcsolatainak száma nem haladhatja meg a b_2 (maximális kardinalitás) értéket.

Az második (P_2) leképzés értelmében az E_2 egyedtípus mindegyik egyedelőfordulásának legalább a_1 számú egyedelőfordulással kell kapcsolatban lennie az E_1 egyedtípusból, azzal, hogy a nagyobb kapcsolatszám megengedett, de nem haladhatja meg a b_1 értéket.

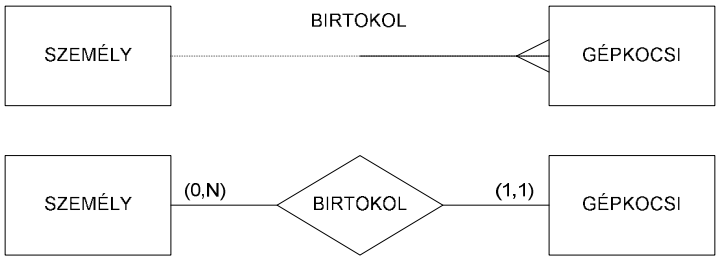
Példa. A személygépkocsi (GÉPKOCSI) és a CASCO viszonyának példája - a biztosít kapcsolattípus – 5.6. ábra.



5.6. ábra

Minden személygépkocsinak nincs szükségszerűen CASCO biztosítása, ha viszont van, akkor legfeljebb egy van. A CASCO szükségszerűen kapcsolódik egy személygépkocsihoz, és legfeljebb is csak egy autóhoz kapcsolódhat. A biztosít kapcsolat kötelező a CASCO oldaláról.

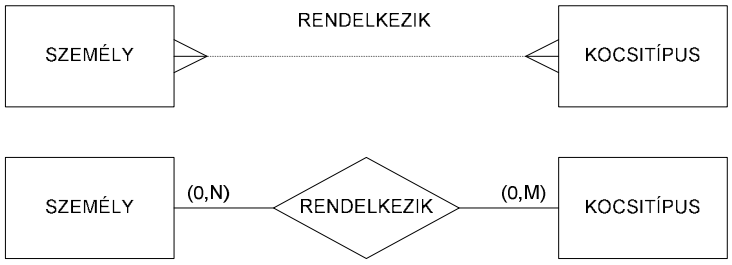
Példa. A személy és a személygépkocsi (GÉPKOCSI) viszonyának példája – a birtokol kapcsolattípus (5.7. ábra).



5.7. ábra

A személynek nem létszükséglete a birtokol kapcsolat (nincs szüségszerűen autója), de akár több szemlygépkocsit is birtokolhat. A személygépkocsi (forgalomban) viszont csak akkor létezhet, ha egy egy személyhez kötött (a birtokol kapcsolattípus alapján), de egytől több személyhez nem is kapcsolódhat. A birtokol kapcsolat kötelező a személygépkocsi oldaláról (a személygépkocsi egzisztenciálisan gyönge egyedtípus).

Példa. A személy és a kocsitípus kapcsolatának példája – a rendelkezik kapcsolattípus (5.8. ábra).



5.8. ábra

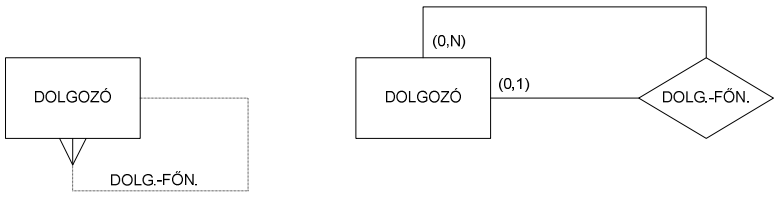
A személy számára nem kötelező, hogy rendelkezzen kocsitípussal, de akár több kocsitípussal is rendelkezhet (nincs korlát, hogy mennyivel). A kocsitípus egyedelőfordulásai számára nem kötelező a kapcsolat léte egyetlen személy felé sem, de konkrét kocsitípussal akár több személy is rendelkezhet.

A **visszamutató/homogén kapcsolattípus**ok esetében a kapcsolatípus önmagával kapcsoja össze ugyanazt az egyedtípust, vagyis ugyanannak az egyedtípusnak különböző egyedelőfordulásai vannak egymással kapcsolatban. Elvben minden számossági jelleg szerinti kapcsolatfok elképzelhető rá. Ami a kapcsolaterőt vagy függéserőt illeti, csak elvétve található totális (kötelező) kapcsolat (vagyis a minimális kardinalitásérték a legtöbb esetben 0).

Példa. A dolgozó egyedtípus dolgozó-főnöke kapcsolattípusának példája

A dolgozó számára nem kötelező, hogy legyen közvetlen főnöke (ez az egyetlen vezérigazgató), de ha van, akkor cask egy közvetlen főnöke lehet. A másik oldalról viszont nem

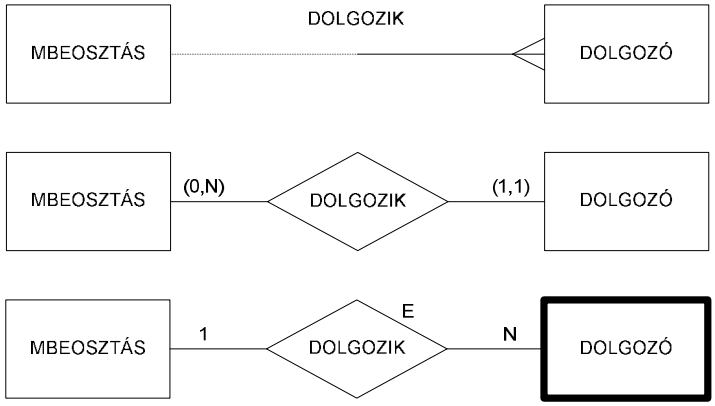
kötelező, hogy minden dolgozónak legyen beosztottja (ezek a beosztott nélküli dolgozók – belőlük sok van), viszont ha van nekik, akkor akár több beosztottjuk is lehet.



5.9. ábra

A gyenge (az **egzisztencionálisan gyenge**) **egyedtípus**. Ezek az egyedtípusok látszatra ugyanolyanok, mint a többi egyedtípus, csupán a függéserő nullától különböző. Ez azt jelenti, hogy az ilyen egyedtípushoz tartozó egyedelőfordulásokat nem lehet beszúrni az adatbázisba, ha a beszúrással egyidőben (ugyanazon tranzakción belül) nem történik meg a függéserő (minimális kardinalitás) értékének megfelelő számú kapcsolat beszúrása is. Sok esetben az egzisztenciálisan gyönge egyedtípust a reguláris/független egyedtípushoz kapcsoló kapcsolattípust is a "gyönge" jelzővel illetik. A gyenge egyedtípusok egyedelőfordulásainak léte a függéserő számának megfelelő kapcsolatok meglététől függ.Ilyen szabályos

Példa. A cégben minden dolgozónak kell, hogy legyen munkahelye (MBEOSZTÁS), ahol dolgozik. Ennek a megszorításnak az 5.10. ábrán látható modell-részlet felel meg

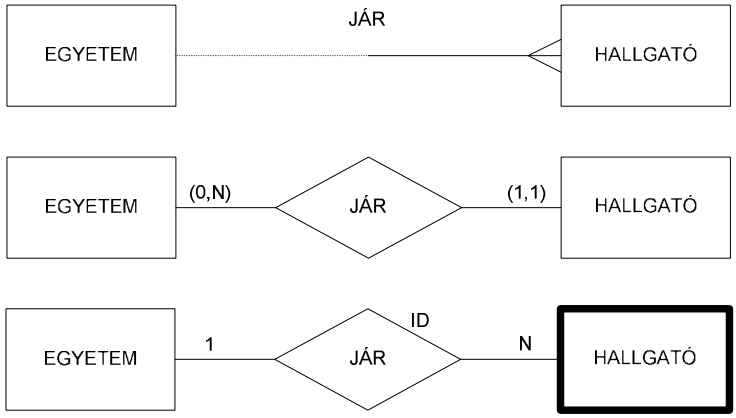


5.10. ábra

Az azonosító-függő gyenge (identifikációsan gyenge) egyedtípus. Amennyiben az egyedtípusnak nincs saját azonosítója (tulajdonságtípushalmaza, amely kulcsként szolgálhatna), akkor identifikációsan gyönge kapcsolattípussal/kapcsolattípusokkal összeköthető a vele kapcsolatban álló független egyedtípussal/egyedtípusokkal, amely/amelyek rész vesznek a gyönge egyedtípus azonosítójának kialakításában. Az identifikációsan gyönge egyedtípust

szokás gyerek egyedtípusnak, az azonosításban (kulcskialakításban) részt vevő egyedtípusokat pedig szülő egyedtípusnak is nevezni. Az identifikációs függés mindig egzisztencionálisat is jelent, fordítva viszont nem érvényes a megállapítás.

Példa. A hallgatók országos szintű megkülönböztetéséhez nem elegendő a »Leckekönyvszám« tulajdonságtípus, amely egyébként egy-egy egyetemen vagy egyetemi karon belül teljesen megfelelő azonosító (kulcs). Ennek az elvárásnak az 5.11 ábrán feltüntetett modell-részlet eleget tesz (feltételezve, hogy egy hallgató csak egy egyetemre járhat).



5.11. ábra

5.4.3. AZ E/K MODELL MŰVELETI RÉSZE

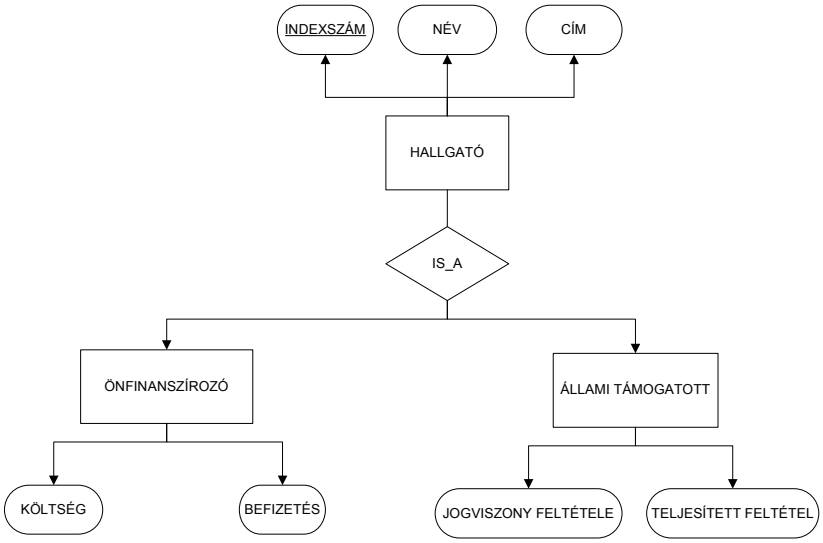
Az E/K modell eredetileg csak a szerkezeti és megszorítási részeket tartalmazta, csak később jelent meg hozzá a műveleti rész. Mivel a gyakorlatban teljesen mellőzik az E/K modell műveleti részének alkalmazását, ezen a szinten szükségtelen annak ismertetése is. Az irodalomban persze könnyen utána lehet keresni (Lazarević) ha valakinek szüksége lenne rá.

5.4.4. AZ E/K MODELL BŐVÍTMÉNYEI

Az E/K modell széleskörű elterjedésével újabb igények fogalmazódtak meg a felhsználók részéről a modellel szemben a valós rendszer minél jobb tükrözése érdekében. Többfajta EER (Extended ER model) modell is megjelent, amelyekben az általánosítások, specializációk és kategorizációk, valamint tartalmazási kapcsolatok és kapcsolatok közötti kapcsolatok leírására/modellezésére van lehetőség.

Specializáció/általánosítás esetén egy egydtípusból az egyedelőfordulások csoportosítása olyan módon történik, hogy a mindig (minden egyedelőfordulásra) értékkel (NOT NULL) renedelkező, közös tulajdonságtípusokat (beleértve az azonosítót/kulcsot is) a fő egyedtípus (egyedfőtípus-superklasa) fogja tartalmazni, egy-egy egyedaltípusba (alosztály-potklasa) pedig csak azok a tulajdonságtípusok kerülnek (az azonosítóval egyetemben), amelyek csak rájuk jellemzőek, a többi egyedaltípusban pedig nem értelmezhetők (NULL értéket vesznek fel). Az egyedaltípus egyedelőfordulásai persze megöröklik az egyedfőtípus összes tulajdonságértékeit (az azonosító értékének megegyezése alapján). Ez az ún. IS_A kapcsolat (»az-egy« típusú kapcsolat), amelyet más néven öröklési kapcsolatnak is hívnak. Az egyedaltípust az egyedfőtípussal egyetemben lehet csak vizsgálni, elemezni. Az IS_A hierarchia top-down kiépítését specializációnak, a bottom-up kialakítását pedig általánosításnak (generalizacija) tekintik. A kapcsolatfokot és –erőt illeti az egyedaltípus oldaláról mindig (1,1), és a diagramon fel sem tüntetik. Az egyedfőtípus oldaláról pedig az (a_2, b_2) páros az előzőekben ismertetett szokványos értelmezéssel bír.

Példa. A hallgatók finanszírozási szempontból kivetített hierarchiája: állami támogatott és önfinanszírozó egyedaltípusokkal rendelkező IS_A (AZ_EGY) hierarchia (5.12. ábra).



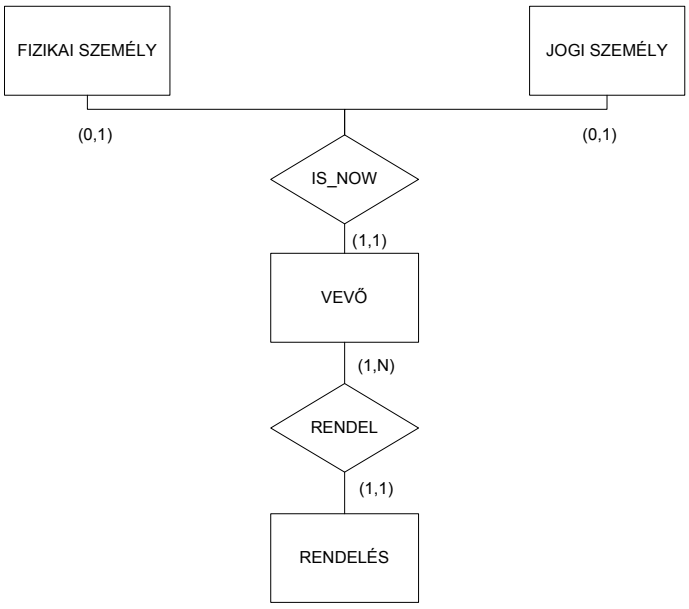
5.12. ábra

A hallgató minden egyedelőfordulása bizonyosan megjelenik az egyik vagy másik egyedaltípus előfordulásaként (ha nincs harmadik finanszírozási lehetőség), vagyis a minimális kardinalitás, a kapcsolaterő értéke 1. Más részről a hallgató csak egyik altípusban jelenhet meg, mert ha önfinanszírozó, akkor azért az, mert nem kapott állami támogatást, ha meg állami

támogatásban részesült, akkor nem fog önszántából, a saját zsebéből még fizetni is. A maximális kardinalitás, a kapcsolatfok értéke tehát ugyancsak 1.

A **kategória** mint fogalom grafikusan, kinézésre »fejreállított« IS_A hierarchiának tűnhetne első pillantásra, de lényegbeli különbségről van szó a két fogalom között. A kategória olyan egyedaltípus, amely különböző egyedtípusok (egyedfőtípusok) egyedelőfordulásait gyűjti egybe (egy kategóriába) annak okán, hogy ezek a különböző egyedtípusok egyedelőfordulásai azonos szerepkörben jelennek meg. A kategória megörökli az egyedfőtípusok összes tulajdonságtípusait (természetesen nem mindegyikét egyszerre) a külső kulcson kerersztül. A kategória-egyedtípus szerkezeti felépítésében helyet kapnak az egyedfőtípusok kulcsai külső kulcsként, de csak egy külső kulcs értéke különbözik NULL-tól. A kategóriának egy mesterségesen bevezetett kulcsa van (szerb nyelven surogat). A kapcsolatfok és –erősség értelemszerűen (1,1) a kategória oldaláról és (0,1) az egyedfőtípusok felöl.

Példa. A vevő kategória/egyedaltípus példája arra a helyzetre, amikor a vevő lehet fizikai és jogi személy is (5.13. ábra).



5.13. ábra

A vevő kategória a fizikai és jogi személyek uniójának valódi részhalmaza, és a saját, mesterségesen kialakított kulcsa mellett tartalmazza a fizikai személy és jogi személy azonosítóját (kulcsát) külső kulcsként. A kategóriát létrehozó, általam önkényesen IS_NOW-nak jelölt (AZ_MOST_EGY) kapcsolattípus a megnevezésben is próbál utalni arra, hogy a vevő kategória a fizikai és jogi személyeknek csak egy szerepkörére utal (pl. a FIZIKAI SZEMÉLY AZ MOST EGY VEVŐ). Valamennyi vevő kategória egyedelőfordulásra érvényes, hogy az

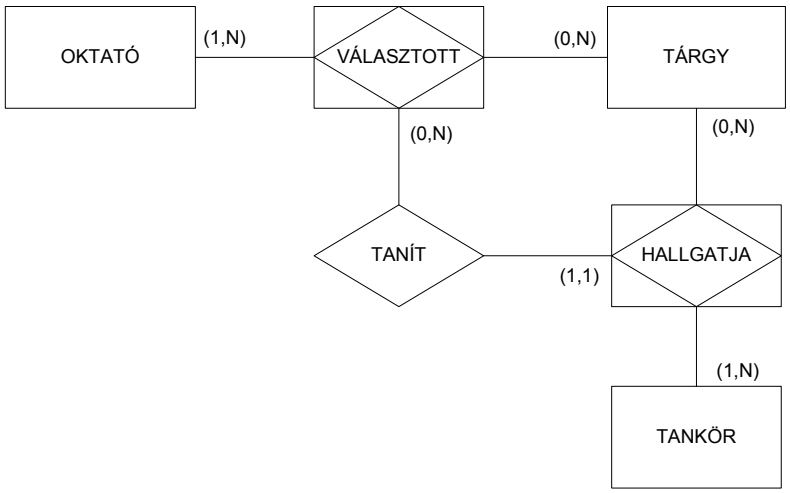
egyik külső kulcsa NULL érték lesz, hiszen ha a vevő fizikai személy, akkor a jogi személyre mint egyedtípusra mutató idegen kulcs értéke mindenképpen NULL értékű kell legyen.

Kapcsoló egyedtípus (szerb nyelven gerund, jelentése főnévi igenév) akkor keletkezik, ha egy kapcsolattípust tulajdonságtípusokkal kell jellemezni, vagy szükség van két kapcsolattípus összekapcsolására. Mivel a megegyezés szerint a tulajdonságokkat/adatokkal leírt lényeg neve egyedtípus, így az adatokkal leírni kívánt kapcsolattípust kapcsoló egyedtípussá kell átalakítani, amihez új jelölés is jár: a romusz, és a sarkait érintő (legkisebb) téglalap. Ugyancsak alapelv, hogy kapcsolattípus csak egyedtípusok között fogalmazható meg, így itt is a kapcsoló egyedtípussá való átalakítás a járható út. A kapcsoló egyedtípus kulcsa vagy összetett vagy hierarchikus.

Példa. Az oktató, a tárgy és a tankör viszonya a tárgyak előadásával és hallgatásával kapcsolatban (5.14.ábra) kapcsoló egyedtípus alkalmazásával.

A 4.23.ábra szemantikája (jelentése):

1. az **o** oktató a **t** tárgyra a **választott** oktató,
2. a **t** tárgyat a **tk** tankör **hallgatja** és
3. az **o** oktató a **t** tárgyat (amelyre választva van) **előad**ja a **tk** tankörnek (amelyik a tárgyat hallgatja).



5.l4. ábra

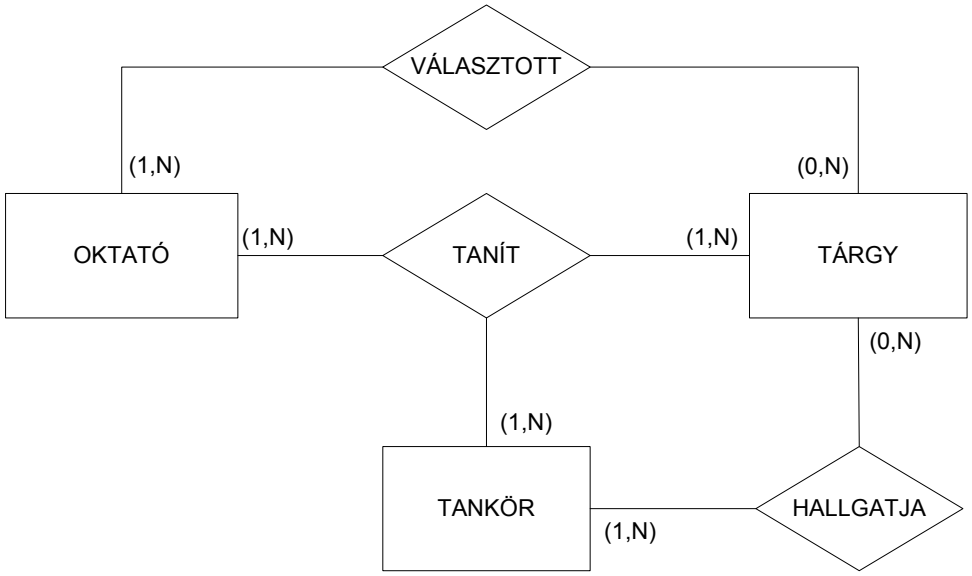
Példa. Az oktató, a tárgy és a tankör viszonya a tárgyak előadásával és hallgatásával kapcsolatban kapcsoló egyedtípus alkalmazása nélkül.

A kapcsoló egyedtípust nélkülöző megoldás szemantikája (5.15. ábra):

1. az **o** oktató a **t** tárgyra a **választott** oktató,
2. a **t** tárgyat a **tk** tankör **hallgatja** és

3. az **o** oktató a **t** tágyat **előad**ja a **tk** tankörnek.

Észrevehető, hogy a második megoldásban nincsenek megszorítások arra, hogy ki, mit fog előadni melyik tankörnek. Itt előfordulhat, hogy az **o** oktató olyan **t** tárgyat ad elő, amelyikre nincs is választása egy olyan **tk** tankörnek, amelyik nem is hallgatja a tárgyat.



5.15. ábra

5.4.5. MODELLEZÉS AZ E/K MODELLEL

Az E/K modell a valós rendszer statikus leírására szolgál: a strukturális rész teljes egészében, az integritási rész csak részlegesen kerülhet leírásra. A modelllezés az egyedtípusok megfogalmazásával kezdődik: a nevük megfogalmazásával és a szemantikájuk, jelentésük leírásával. Az egyedtípusok a valós rendszer adatokkal leírni kívánt fontos dolgainak fogalmi tükörképe. Az egyedtípusok számbavétele után következik a kapcsolattípusok kijelölése és jellemzése, amivel párhuzamosan már a grafikus kép, a diagram kialakítása is elkezdődik. Ez a két dolog együttesen képezi az E/K modell lényegét, és a későbbiekben létrehozandó adatbázis gerincét is. A harmadik lépésben pontosítandó, hogy mit is kell tudni a valós rendszer fontos lényegeiről: ez a lépés a tulajdonságtípusok, azok értékhalmazának, doméjnjeinek a meghatározásáról szól.

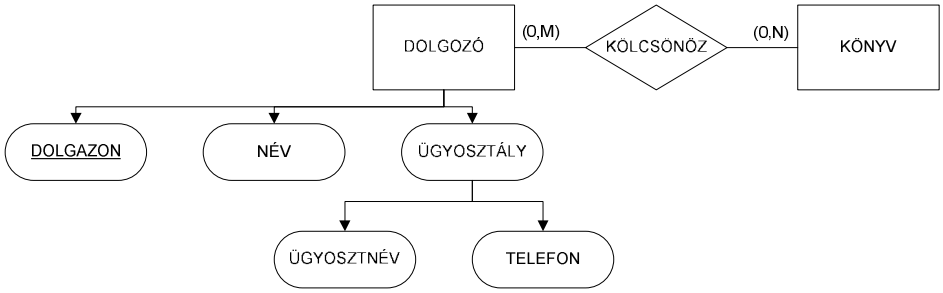
Az informatikában (a modellezésben is) a megoldások meg egyediek, egyformák. Több különböző megoldás is kielégítheti az elvárásokat, követelményeket. A valós rendszer fogalmainak leképzése nem egyértelmű. Ugyanazon fogalomnak különböző ábrázolása, tükrözése, modellezése nem jelent szükségszerűen rossz megoldást.

Példa. Egy fogalom egyedtípusként és tulajdonságtípusként is modellezhető.

5.16. ábra

Az 5.16. ábrán egy cég-struktúra modellezésénél az ügyosztályok, a dolgozók és a projektek jelennek meg (több, egyéb más lényeg mellett), amelyeket adatokkal kell jellemezni.

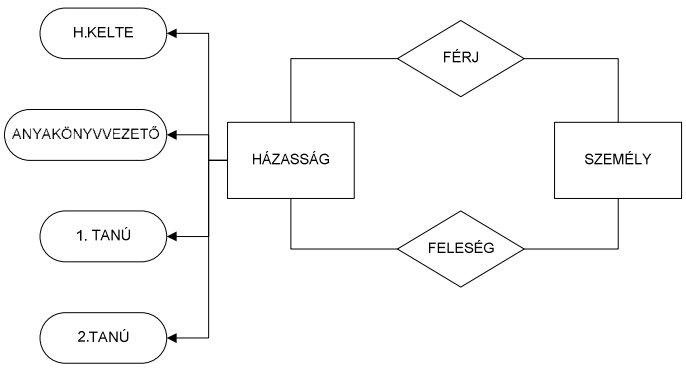
A 5.17. ábrán a céges könyvtár modell-részlete látható, amelyen az ügyosztály a dolgozó egyedtípus tulajdonságtípusaként, a dolgozó munkahelyének leírásaként jelenik meg. Az ügyosztály a két különböző valós rendszerben az önállóságban, fontosságban különbözik, ezért eltérő az ábrázolása.



5.17. ábra

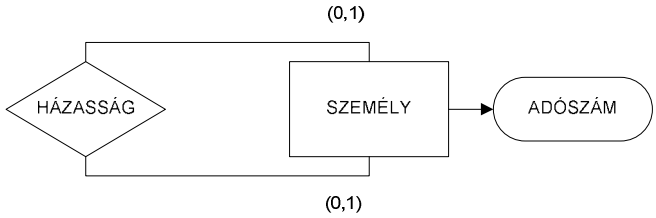
Példa. Egy fogalom egyedtípusként és kapcsolattípusként is modellezhető.

Az anyakönyvi hivatal nyilvántartásába tartoznak a házasságok is, amelyekről különböző adatokat is nyilvántartanak. A leírásból egyértelműen következik, hogy itt a házasság egyedtípusként jelenik meg (5.18. ábra).



5. 18. ábra

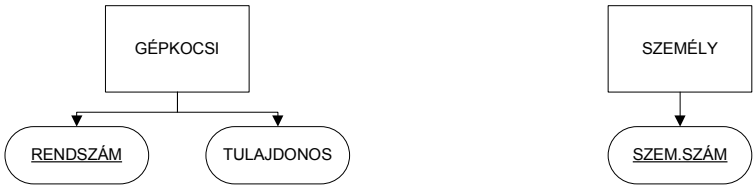
Az adóügyösztályon a házasság csak az egy családba való tartozás megállapítását kell, hogy lehetővé tegye, az adatok magáról a házasságról nem érdekesek, hanem csak az általa megvalósuló kapcsolatok (5.19. ábra).



5.19. ábra

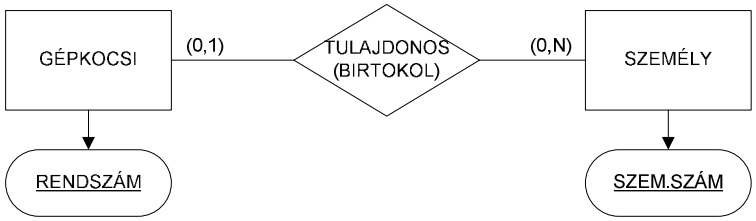
Példa. Egy fogalom tulajdonságtípusként és kapcsolattípusként is modellezhető.

A tulajdonosi viszonyt a forgalmi engedélyben a 5.20. ábra szerint tartják nyilván: a személygépkocsi leírásaként, jellemzéseként (annak ellenére, hogy a tulajdonos nem a gépkocsi jellemzője.



5.20. ábra

Szakmai körökben a 5.21. ábrán látható megoldás az elfogadott: a tulajdonos kapcsolatként ábrázolva két lényeget: a személyt és a személygépkocsit kapcsolja össze. Ez az ábrázolás szemantikailag jobban, pontosabban tükrözi az általa képviselt valós rendszerrészt. Az 5.21.



5.21. ábra

ábrán látható megoldásrészlet eltér az 5.7. ábrán látható modellrészlettől, ugyanis itt nemcsak a forgalomban lévő gépkocsik kerülnek be a nyilvántartásba, hanem a különböző autószalonokban tulajdonosra váró új és használt (külföldről behozott) gépkocsik is (nulla értékű kapcsolaterősség a gépkocsinál).

5.5.OBJEKTUMORIENTÁLT ADATMODELL

Az objektumorientált adatmodell igénye a mérnöki tervezésben jelentkező bonyolult adatstruktúrák leírásánál és kezelésénél fogalmazódott meg. Az ilyen fajta igények hatékony kezelésére a relációs (és az előtte kialakított bármelyik) adatmodell már nem alkalmas. A mérnöki tervezés igényelte nagyszámú tulajdonságtípus, kisszámú tulajdonságértékekkel párosulva, nagy- és változó hosszúságú értéksorok kezelése, a mérnöki tervezésben igényelt fejlesztési változatok nyomonkövetésének igénye, és a többi mérnöki alkalmazásban megfogalmazott elvárások mind-mind túlmutatnak a relációs adatmodellek lehetőségein.

Az objektumorientált adatmodell az objektumorientált programnyelvek igényeire/tapasztalataira és a szemantikus adatmodellekre vezethető vissza. Az alapgondolat az, hogy a valós rendszer egymással kapcsolatban lévő és együttműködő objektumok halmazaként is felfogható. A végső cél az, hogy az adatbázisban alkalmazott objektumok teljes mértékben megfeleljenek, „egybe olvadjanak" az objektumorientált gazdanyelvvel, amely az objektumos adatbázist használja. Hogy a program által létrehozott és az adatbázisban meglévő objetumok között csak az „élettartamban" legyen különbség: az alkalmazások objektumai rövid életűek (tranziens objektumok-csak az alkalmazás futásideje alatt élnek), az adatbázis-objektumok pedig hosszú életűek (perzisztens objetumok-az adatbázis élethossza a korlát).

Az objektumorientált szemléletmód legfontosabb jellegzetességeinek ismertetése nem maradhat ki egyetlen objektumos modellezési technika ismertetéséből sem, így az objektumorientált adatmodell bemutatásánál sem. A legtöbbet emlegetett, és legfontosabbnak tartott jellegzetességek a következők:

1. egységbezárás (encapsulation)
2. öröklődés (inheritance)
3. polimorfizmus
4. újrafelhasználhatóság

Az egységbezárás az a mechanizmus, amellyel az objektum a szerkezetét, az attribútumait és a működését, műveleteit összefogja és elrejti a környezete elől. Kívülről az objektum nem befolyásolható (adatai, eljárásai nem érhetők el) közvetlenül, hanem csak azokkal a nyilvánossá és elérhetővé (láthatóvá) tett műveletekkel, amelyeket a létrehozásnál definiáltak.

Az öröklődés osztály szinten megfogalmazott jellegzetesség, és azt jelenti, hogy minden osztály örökölheti teljes számban vagy csak szelektíven a hierarchiában felette lévő osztály vagy osztályok jellegzetességeit (attribútumait, műveleteit).

A polimorfizmus vagy többalakúság azt jelenti, hogy egy konkrét objektum, ugyanarra a kérésre (üzenetre) akár különbözőképpen is reagálhat, ugyanis nemcsak az üzenet (kérés), hanem az üzenet küldője is befolyásolhatja a reakció-művelet kiválasztását. A műveletek mellett az adatok is eltérő formában jelenhetnek meg ugyannannál az objektumnál, a hatásmechanizmus (a kérés-kérésküldő) függvényében.

Az újrafelhasználhatóság olyan modell-elemek, modell-egységek kidolgozását jelenti, amelyeket a fejlesztés későbbi szakaszaiban, vagy akár más fejlesztésekben újra hasznosíthatnak. Ez a komponens alapú fejlesztés alapötlete.

Az objetumorientált adatmodell két alapfogalma az **objektum** és a **literál**. Érdekes azonban megjegyezni, hogy ezekből az alapelemekből nem épül ki összetettebb elem az objektumorientált adatmodellekben. Viszont az objektumok és literálok, típusaiktól függően lehetnek több fajták, akár nagyon összetettek is, anélkül, hogy ezeknek az összetett objektumstruktúráknak külön megnevezést láttak volna elő. Másszóval az objektumnak, mint strukturális/szerkezeti elemeknek az összetettségére a típusmegjelölése utal. Ez teljesen érthető, hiszen amíg, mondjuk a relációs modellben pontosan előírt szerkezetű/felépítésű strukturális elemek jelenhetnek meg csak a modellben (attribútum, relációséma, adatbázisséma), addig az objektumos környezetben, az objektumok szerkezeti bonyolultságának a határa a csillagos ég.

Az **objektum**ok fogalma alatt bármit lehet érteni/értelmezni (szubjektum, objektum, fogalom, elképzelés, esemény, vagy ezeknek egy csoportja), amely egyértelműen megfogalmazható, körülhatárolható és megkülönböztethető más objektumoktól a rendszerben. A leírás nagyon emlékeztet az egyedtípus megfogalmazására, azzal a különbséggel, hogy az objektumoknak az attribútumaik és kapcsolataik mellett metódusaik (eljárásaik, viselkedésük) is vannak.

Az objektumok tehát olyan egyedtípusok, amelyek képesek megőrizni az állapotukat (jellemzőiket), emellett műveletekkel (eljárásokkal, operációkkal) is rendelkeznek, közöttük olyanokkal is, amelyeknek a végrehajtását a környezetben lévő más objektumok kérhetik. Egy objektum úgy tud hatni egy másik objektumra, hogy üzenetet küld neki (kommunikál vele), amelyben a célobjektum neve, és a küldő objektum számára is látható művelet (viselkedés) áll. A műveletek láthatósága korlátozott: egy adott objektum összes műveletét nem látja mindegyik objektum a környezetéből. Az üzenetet fogadó objektum az üzenet hatására elvégez egy műveletet, vagy megváltoztatja az állapotát. Az objektumok állapotát a tulajdonságainak értékhalmaza szabja meg. Az objektum tulajdonságai közé az attribútumai és a kapcsolatai tartoznak. Az objektum viselkedése a műveletein keresztül rögzített. Minden objektum egyedi azonosítóval (identifier) rendelkezik, amely megkülönböztetésre szolgál, és független magától az

objektumtól (a struktúrájától, állapotától és metódusaitól). Az objektumnak nevet is kell választani, leírását pedig attribútumokkal és műveletekkel megadni. Az attribútumok függőségi viszonya az objektumon belül legalább a harmadik normálformának megfelelő szinten kell legyen. Az objektumok csoportosíthatóak

1. *a valós rendszerben betöltött szerepük alapján*
2. típusuk szerint.

A valós rendszerben betöltött szerepük alapján az objektumok lehetnek: a) **entitás-jellegű objektumok**, b) **interfész-objektumok** és c) **vezérlő objektumok**. Az **entitás-jellegű objektumok** a problématér elemzésekor (üzleti modell kialakításakor) bukkannak elő és kerülnek megfogalmazásra (ezek általában hosszabb élettartamú objektumok). Az **interfész-objektumok** az elnevezésüknek megfelelően a rendszert (a rendszer belső működését) kapcsolják össze a környezettel (kommunikáció a környezettel). Ilyenek pl. a különböző jeletésformák, képernyőképek, stb. A **vezérlő objektumok** a feladatok végrehajtásának felügyeletében, irányításában, vezérlésében vesznek részt, akár konkrét befolyás gyakorlásával egyes objektumok felett.

Az objektumok típus szerinti csoportosítása a következő: a) atomi objektumok (nincs beépített/előre definiált, csak felhasználó által létrehozott/definiált), b) kollekciók, és c) strukturált objektumok. A kollekció-objektum több különálló (atomi vagy kollekció) objektumból épül fel. Fajtái: halmaz (az elemek rendezetlen halmaza-ismétlődésmentes), multihalmaz/táska (rendezetlen elemgyűjtemény-ismétlődésekkel), lista (elemek rendezett gyűjteménye), tömb (programnyelvekből ismert típus-változó hosszúságú), szótár (kulcs-érték párosok gyűjteménye-kollekciók indexelésére használatos típus). A strukturált objektumok fajtái: dátum, idő, időbélyeg és itervallum. Minden típusnak egy leírása (specification) és egy vagy több megvalósítása (implementation) van. A típus specifikációja (leírása) a típus használója számára is látható, külső jellegzetességeket tartalmazza: a kívülről végrehajtásra kérhető műveletek és a kívülről hozzáférhető tulajdonságok (adatok, attribútumok), valamint a kivételek (exceptions, hibák), amelyek a végrehajtás folyamán keletkezhetnek. A típus megvalósítása (implementation) az adott objektum belső felépítését, működését, jellegzetességeit rögzíti. A leírás és a megvalósítás elkülönítése az objektumorientált hozzáállás nagy vívmánya, hiszem így valósul meg az egységbezárás (lásd később): a környezet csak használja az objektum szoláltatását (a műveletet, amit nyújt), anélkül, hogy ismerné a megvalósítás módját. A megvalósítás tehát rejtve marad a szolgáltatás használói előtt.

A **típusleírások** (specification) **interész** (interface, nem tévesztendő össze az interfész-objektumokkal) vagy **osztály** segítségével adhatók meg. Az interfész nem példányosítható, de a

benne megfogalmazott művelet-leírásokat az felhasználói objektumok örökölhetik. Az osztály állapot- és viselkedés jellezgetességeket is tartalmaz, és példányosítható. Az állapotleírás fogalma alatt az attribútum- és kapcsolatleírásokat kell érteni, a viselkedés alatt pedig a műveletleírásokat.

Állapotmodellezés – Egy objektumpéldány állapotát az attribútumértékek és a konkrét kapcsolatok határozzák meg. Az attribútum típusa bármilyen literál- vagy objektumtípus lehet, másszóval az objektumok attribútumértékei literálértékek vagy objektumidentifikátorok lehetnek. Az utóbbi eset felfogható kapcsolatként is (hiszen egy attribútumérték-a relációs modellben ezt hívják külső/idegen kulcsnak- egy másik objektumra mutat) , de az objektumos megközelítés rendelkezik külön mechanizmussal/leírással a kapcsolatok megfogalmazására. A kapcsolat csak bináris (két objektum közötti) lehet, és implicit megfogalmazású (attribútumszinten, az attribútumfelsorolás egy soraként specifikált). A kapcsolat párokban kerül leírásra a specifikációban, a kapcsolat inverze is rögzítésre mindkét oldalon. A kapcsolat kardinalitása aszerint van meghatározva, hogy a kapcsolatban atomi objektum („egyes" oldal) van-e feltüntetve vagy kollekció („többes" oldal). Az integritásmegőrzés az objektumos környezetben a törlésre kerülő objetum felé mutató kapcsolatok törlésében nyilvánul meg.

Viselkedésmodellezés. Az viselkedésleírások csak az interfész- és objektumtípusban jelenhetnek meg. A viselkedésmodell műveletleírásai szabványosak, a műveletnevek (műveletmegnevezések) egy-egy típuson belül egyediek. Különböző típusokban előfordulhatnak azonos megnevezésű, különböző tartalmú (eltérő) műveletek, az ilyen neveket „túlterhelt" neveknek hívják (overloaded). Ilyen esetekben a műveletvégrehajtás során fel kell oldani a „túlterheltséget", és a megszólított objektumban kell keresni a végrehajtásra kért műveletet. A műveletek elvégzése közben különböző hibák történhetnek. Ezen előrelátható hibák kezelésére kivételek (exception) fogalmazhatók meg, amelyek aztán akkor élesednek (hajtódnak végre), amikor bekövetkeznek a hibák.

Öröklődés. Az öröklődés két fajtáját különbözterti meg az objetumorientált adatmodell: a) az állapot-öröklődést és b)a viselkedés-öröklődést. Az állapot-öröklődést tükröző kapcsolat az **extends,** de ezzel a kapcsolattal a gyerekobjektum nemcsak az állapotot (attribútumokat) örököli meg a szülőobjektumtól, hanem a viselkedést (műveleteket) is. A viselkedés-öröklődés megvalósítására az IS_A (generalizáció/specializáció) kapcsolat szolgál, azzal, hogy a gyerekobjektum a viselkedését (műveleteket) több szülőobjektumtól is örökölheti.

Az objektumorientált adatmodell másik alapfogalma a **literál**. A literál egy állandó értékű „objektum" (tehát valamilyen érték, vagyis adat), ami alkalmazásra kerül az adatmodellben. Az idézőjel azért szükséges, mert a literál mégsem objektum: az objektumnak van egyedi

azonosítója (identifier), a literálnak viszont nincs (emellett értéke is megváltoztathatatlan). A literálok típus szerinti csoportosítása megegyezik az objektumok típus szerinti csoportosításával. Az atomi literálok fajtái: long, long long, short, unsigned long, unsigned short, float, double, boolean, octet, char (character), string, enum (enumeration). A literálok másik két csoportja teljesen megegyezik az objektumok típus szerinti csoportosításának második, illetve harmadik csoportjával.

Az objektumorientált adatmodellben megtalálható ugyanaz a kettősség, mint az E/K modellben, tudniillik, hogy a modellelemek két absztarkciós szinten közelíthetők meg: fogalmi (absztrakt) és logikai (előfordulási vagy értékszint) szinten. A fizikai (a fizikai tárolókon való elhelyezés formája) szint ismeretlen marad az átlagfelhasználó számára, hiszen azt az objektumorientált adabáziskezelő rendszer elrejti. Foglami vagy absztrakt szinten osztályokról, attribútumokról és műveletekről szól a diskurzus, míg a logikai (előfordulási) szinten az objektum, attribútumérték és metódusok fogalmak használatosak.

5.5.1. OBJEKTUMORIENTÁLT ADATMODELL LEÍRÁSA ODL-BEN

Az adatbázisszerkezet leírása az objektumorientált technológia fogalmaival a leggyakrabban ODL (Object Definition Language) nyelv segítségével történik. Az alábbiakban a legalapvetőbb fogalmak leírásának/megfogalmazásának ismertetése következik, úgy mint az osztály, attribútum, kapcsolat (kapcsolattípusok), inverz kapcsolat, alosztály, kulcs és attribútumtípus (az objektum/osztálytípusokról már volt szó az előző pontban). Az ODL, az objektumorientált megközelítés bármelyik megfogalmazási módjához hasonlóan az osztály fogalma köré épül. A kapcsolat fogalma nem önálló, mint az E/K modellben, hanem speciális tulajdonságként jelenik meg (a tulajdonságok közé van beágyazva).

Az osztály és attribútumleírás az osztálydeklarációban található (az előzőek szerint a kapcsolatok/inverz kapcsolatok is):

```
Class <osztálynév> {
        <jellemzők listája>
};
```

Példa:

```
Class Hallgató {
        attribute integer leckekönyvszám;
        attribute Struct Név {string vezetéknév, string keresztnév} hallgatónév;
        attribute boolean neme;
        attribute integer születésiév;
        attribute Struct Cím {string város, string utca, string házszámlakás,
                              integer irányítószám } lakcím;
        attribute string telefon;
        relationship Set<Tárgyak> felvette inverse Tárgyak::felvették
};
```

Attribútumtípusok:

a) atomi:integer, float, character, string, boolean és enum
b) rekordstruktúra: Struct S{T_1 N_1, T_2 N_2, …, T_n N_n}, ahol T a típusmegnevezést, az N pedig a mezőnevet jelöli
c) osztálynevek (az osztályok az előző pontban ismertetett öt típusúak lehetnek: **halmaz** (Set<T> - a T típusú elemek rendezetlen halmaza-ismétlődésmentes), **multihalmaz/táska** (Bag<T> - a T típusú rendezetlen elemgyűjteménye-ismétlődésekkel), **lista** (List<T> - a T típusú elemek rendezett gyűjteménye/listája), **tömb** (Array<T,i> - a T típusú elemek i elemű tömbje, programnyelvekből ismert típus), **szótár** (Dictionary<T,S> - T kulcsértékből és S kiterjedéstípusból álló párosok nem ismétlődő/egyedi gyűjteménye-kollekciók indexelésére használatos típus)

Kapcsolat és inverz kapcsolat ábrázolására az ODL az attribútumok között adja meg a lehetőséget: erre a példa utolsó sorában található leírás vonatkozik. Szemantikája a következő: a felvette nevű kapcsolat (relationship) alapján a hallgató osztály minden objektuma (eleme/minden hallgató-objektuma) a tárgy osztály egy-egy objektumhalmazával lehet kapcsolatban. A kapcsolatfokra a kapcsolat megfogalmazásának (definiciójának) formája utal: ha a relationship kulcsszó után osztálynév található, akkor a kapcsolatfok 1, ha viszont a relationship kulcsszó után kollekciótípus (Set<osztálynév>, Bag<osztálynév>, List<osztálynév>, Array<osztálynév> vagy Dictionary<osztálynév>), akkor a kapcsolatfok értéke N vagyis több.

A kapcsolatnak megfelelő inverz kapcsolat csak jelölésre kerül, mégpedig közvetlenül a kapcsolatdefiníció után. Az inverz kapcsolat tényének/létének megfogalmazásában nincs utalás az inverz kapcsolat kapcsolatfokáról. Erről csak az inverz kapcsolat megfogalmazásának helyén (abban az osztályleírásban, amelyben definiálásra került) található információ az előzőekben leírt formában.

Alosztályok természetesen ODL-ben is megfogalmazhatók. A megfogalmazás helye az alosztály ODL-deklarációjában helyezkedik el. Az 5.12. ábrán látható osztályhierarchia egy részének (önfinanszírozó hallgató) ODL leírása következik most:

```
Class ÖnfinanszírozóHallgató extends Hallgató {
        attribute integer költség;
        attribute integer eddigi_befizetések;
};
```

A többszörös öröklés lehetősége is adott: az extends kulcsszó után több, egymással kettősponttal elvásztott osztály is megjelenhet. A szemléltető példa felírásához szükség van egy második osztályra is a többszörös (kétszeres) örökléshez:

```
Class Dolgozó {
        attribute integer személyi_szám;
        attribute Struct Név {string vezetéknév, string keresztnév} dolgozónév;
        attribute boolean neme;
        attribute integer születésiév;
        attribute Struct Cím {string város, string utca, string házszámlakás,
                              integer irányítószám } lakcím;
        attribute string telefon;
        attribute integer fizetés;
        attribute integer a_fizetés_terhelhető_része;
};

Class DolgÖnfinanszHallgatóFizetésiÁtutalással extends Hallgató : Dolgozó {
        attribute integer havirészlet;
        attribute integer kezdőrészlet_év_hó;
        attribute integer részletek_száma;
};
```

A kulcsmegszorítás leírása ODL-ben nem kötelező, hiszen az objektumtermészetű ODL alapértelmezettnek veszi, hogy minden objektum születésétől rendelkezik azonosítóval. Az ODL-ben akár több kulcs is deklarálható közvetlenül az osztálynév feltüntetése után: zárójelbe téve és a **key** vagy **keys** kulcsszó megadása után, a kulcsokat egymástól vesszővel elválasztva. Összetett kulcs esetén a kulcsattribútumokat zárójelben kell felsorolni:

```
Class Osztályzat (key (leckekönyvszám, tárgyazon)) {
…
};
```

5.6. AZ UML OSZTÁLYDIAGRAMJA MINT ADATBÁZIS-SPECIFIKÁCIÓ

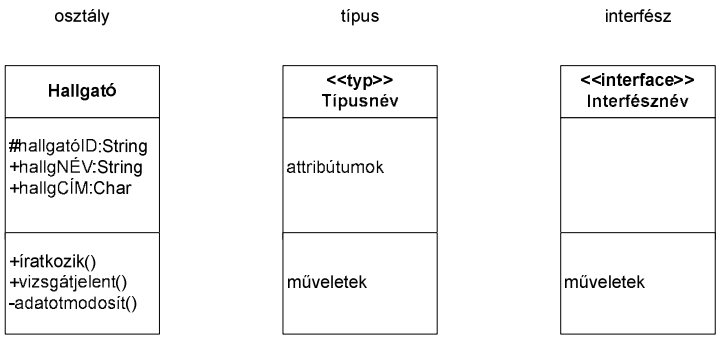
Az Egységesített Modellező Nyelv (Unified Modeling Language – UML) az alkalmazásfejlesztés (információs rendszerek tervezésének) segédeszköze, amely egyben a rendszerdokumentáció rögzítésének mergfogalmazási módja (»nyelve«) is.

Az osztálydiagram a rendszer szerkezetét/felépítését, a statikáját rögzíti/dokumentálja, és ehhez rendelkezik az összes szükséges szerkezeti alapelem-jelöléssel: osztályok, interfészek, attribútumok, kapcsolatok és műveletek is leírhatók vele. Az osztály és az interfész fogalma az UML-ben megfelel az objektumorientált adatmodellnél megismert osztály és interfész fogalmával (az UML és az E/K modell legfontosabb fogalmainak megfeleltetése az 1. Táblázatban található.

1. Táblázat

UML	E/K modell
Osztály	Egyedtípus
Társítás/asszociáció	(bináris) Kapcsolattípus
Társításosztály	A kapcsolattípus attribútumai
Alosztály	Egyedaltípus
Aggregáció	Egy-sok v. sok-egy kapcsolattípus
Kompozíció	Egy-sok v. sok-egy kapcsolattípus hivatkozásiépség-megszorítással

A típusok definiálására nincs külön fogalom/jelölés (koncept), hanem az osztály sztereotípiájaként ábrázolhatók. A sztereotípia teszi lehetővé új fogalmak bevezetését a már meglévők segítségével (összekombinálásával). Az UML-ben már vannak előre definiált sztereotípiák, pl. az interfész és a típus (5.22. ábra). Interfész esetén attribútumokat általában nem tüntetnek fel.



5.22. ábra

Az UML osztályleírásában a láthatóság a legfontosabb újdonság, melynek értékei: a) »-» privát, csak osztályon belüli műveletek számára látható, b) »+« nyilvános, minden más

osztályból, sőt, a rendszeren kívülről is elérhető, és c) »#« védett, csak bizonyos objektumok férhetnek hozzá. Egyetlen kulcs (egyszerű vagy összetett) megfogalmazására van csak lehetőség az UML-ben is, hasonlóan az E/K modellhez. A jelölés egy »PK« (Primary Key) rövidítés a kulcsot alkotó attribútumok neve után. A külső kulcsok is jelölhetők egy »FK« (Foreign Key) rövidítéssel a külső kulcsban szereplő attribútumok neve után (az 5.22. ábrán nincsenek feltüntetve ezek a jelölések).

Az asszociáció névvel azonosítandó társítási kapcsolat, amelyeknél a minimális és maximális kardinalitás jelölésében eltérések mutatkoznak az E/K modellhez képest: a »*« jel a maximális kardinalitás értékeként korlátmentességet jelez, ha a minimális és maximális kardinalitás értékeként csak egyetlen »*« jel jelenik meg, az »0..*« jelentésű, ha egyáltalán nincs érték feltüntetve, akkor azt »1..1«-nek kell értelmezni. A visszamutató társítások (társítások önmagával) semmiben sem különböznek a különböző osztályok társításától: az osztály itt két különböző szerepben jelenik meg a társításban, és ezek a szerepek névvel jelölhetők a társítás két végén. Az egy-több típusú asszociációk (rész-egész kapcsolat) egyes oldalán előforduló két variációja külön nevet kapott az UML-ben:

(a) aggregáció – az egyes oldalon »0..1« jelölés található, vagyis a résznek nem kell kötelezően kapcsolódnia egyetlen egészhez sem, de ha kapcsolódik, akkor csak egy egészhez kapcsolódhat; jele egy üres rombusz az egyes oldalon (a »0..1« jelölése nélkül)

(b) kompozíció – az egyes végen »1..1« típusú jelölést tartalmazó asszociáció, amelynél a résznek tartoznia kell legalább egy egészhez, és legfeljebb is csak egy egészhez tartozhat; jele egy fekete rombusz az egyes oldalon (a »1..1« jelölése nélkül); ez az egzisztenciálisan gyenge egyedtípusok megfogalmazási módja az UML-ben.

Az identifikációsan gyenge egyedtípusok jelölésére nincs külön szimbólum az UML-ben, de a kompozíció bizonyos jelölésmódosítással jól szemlélteti a kulcsköltöztetést a gyenge oldal felé: a gyenge (a fekete rombusszal szemben) oldalon a kapcsolat vonala nem az osztálhoz kapcsolódik közvetlenül, hanem egy, az osztályhoz illesztett „PK” feliratot tartalmazó kis négyzethez, utalva ezzel arra a tényre, hogy a gyenge oldal a kompozíció kapcsolatot kulcs előállítására használja.

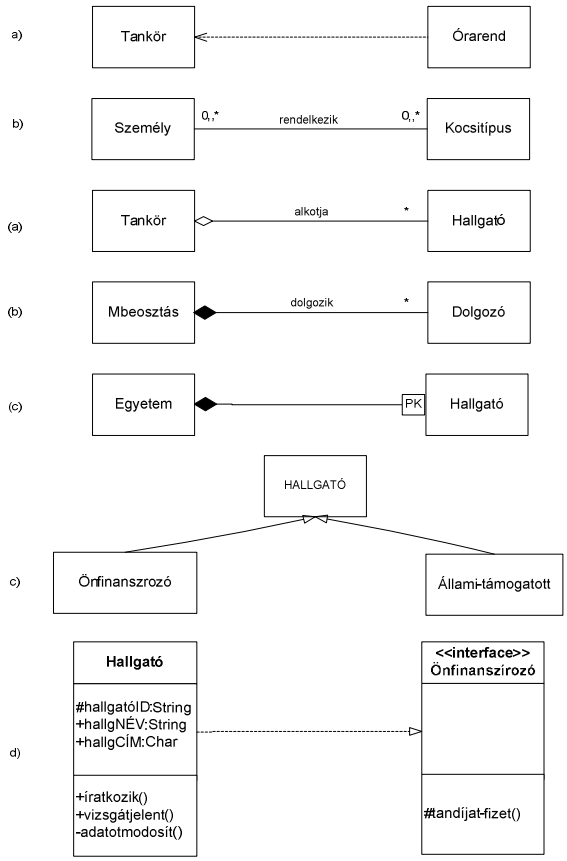
Kapcsolatok. A teljesség kedvéért következzen az UML nyelvben megfogalmazható hat, illetve hétféle kapcsolat az osztályok között (Lazarevic et al.5.23. ábra):

a) függőségi kapcsolat (amikor bizonyos változások az egyik osztályban kiváltják a változásokat a másik osztályban) – a Tankör változásának következtében (a hallgatók

számának növekedésével) megváltozhat az Órarend (pl. újabb labi-csoportot kell az órarendbe iktatni),

b) asszociáció (az E/K modellben ismertetett kapcsolatnak felel meg, a kapcsolaterő/ függéserősség és a kapcsolatfok –minimális és maximális kardinalitás-is kötelezően feltüntetendő) – a (egy konkrét) Személy rendelkezhet akár több kocsitípussal is, és a (egy konkrét) Kocsitípus akár több személy birtokában is lehet,

   (a) aggregáció, gyakran speciális asszociációnak (rész-egész kapcsolatnak) tekintik (az egész megszűnésével a rész nem szűnik meg automatikusan, a rész számára **nem kötelező** a kapcsolat az egész felé) – a Hallgató »alkotórésze« a Tankörnek,



5.23. ábra

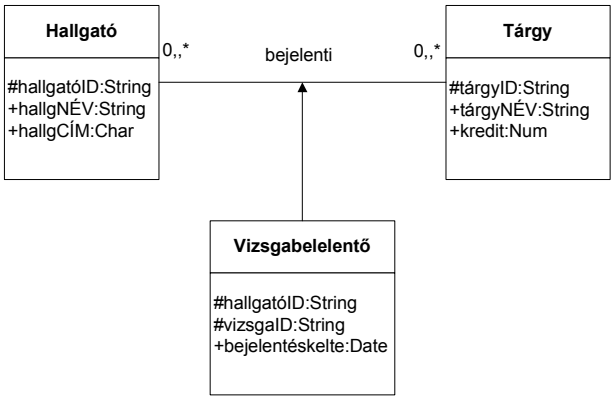
   (b) kompozíció vagy szigorú aggregáció (szoros rész-egész kapcsolat, az egész megszűnésével a rész is eltűnik, a rész számára **kötelező** a kapcsolat az egész felé) – egzisztenciálisan gyenge osztály ábrázolására – a Dolgozó kötelezően egy Mbeosztásban (munkabeosztásban) dolgozik (a munkabeosztás megszűnésével a dolgozó is megszűnik-esetleg más beosztásba kerül),

(c) kompozíció identifikációsan gyenge osztály ábrázolására (szoros rész-egész kapcsolat, az egész megszűnésével a rész is eltűnik, a rész számára **kötelező** a kapcsolat az egész felé) – a Hallgatónak, mint gyenge osztálynak az Egyetem (erős osztály) felé mutató kapcsolata kulcsképzésre alkalmazott/használt,

c) általánosítás (generalizáció) és specializáció (öröklődési kapcsolat) és

d) megvalósítási viszony (kapcsolat) – a Hallgató megvalósítja az Önfinanszírozó interfészt.

Egyes E/K modellértelmezések megengedik, hogy a kapcsolattípusok tulajdonságtípusokkal legyenek leírva. Ebben a tankönyvben nem támogatott ez a fajta szabad értelmezése az E/K modellnek, hanem a kapcsolattípusok tulajdonságtípusokkal való leírása helyett kapcsoló egyedtípusokat ajánl a leírás. Ha tehát kapcsolattípusokat tulajdonságtípusokkal kell jellemezni, akkor előzőleg a kapcsolattípusok kapcsoló egyedtípusokká való átalakítását szorgalmazza a leírás. UML-ben szokványos fogalom a társításosztály (kapcsolatosztály), amelyben a kapcsolat jellemzői kapnak helyet. Az 5.24. ábrán a Hallgató és a Tárgy osztály között megfogalmazott „bejelenti” asszociációs kapcsolatot leíró társítási osztály szerepel**.**



5.24. ábra

Az öröklődés az E/K modellhez hasonlóan négy fajta lehet, azaz négy különböző fajta osztályhierarchia alkalmazható az UML-ben:

5. részleges osztályhierarchia – a szuperosztálynak lehet olyan eleme/objektuma, amelyik nem tartozik egyetlen alosztályba sem (az E/K modellben a részleges osztályhierarchia olyan IS_A kapcsolatnak felel meg, ahol az egyedfőtipus kapcsolaterő – minimális kardinalitás – értéke 0).

6. teljes osztályhierarchia – a szuperosztálynak minden eleme/objektuma beletartozik valamelyik alosztályba (az E/K modellben a teljes osztályhierarchia olyan IS_A kapcsolatnak felel meg, ahol az egyedfőtipus kapcsolaterő – minimális kardinalitás – értéke 1).
7. diszjunkt osztályhierarchia – a szuperosztálynak minden eleme/objektuma legfeljebb egy alosztályba tartozik (az E/K modellben a diszjunkt osztályhierarchia olyan IS_A kapcsolatnak felel meg, ahol az egyedfőtipus kapcsolatfok – maximális kardinalitás – értéke 1).
8. átfedő v. átlapolt osztályhierarchia – a szuperosztálynak lehet olyan eleme/objektuma, amelyik több alosztályba is tartozik (az E/K modellben az átfedő/átlapolt osztályhierarchia olyan IS_A kapcsolatnak felel meg, ahol az egyedfőtipus kapcsolatfok – maximális kardinalitás – értéke N).

Az UML jelölésrendszere gazdagabb az objektumorientált adatmodellétől. Az utóbbi években széleskörűen használatosak az UML-en alapuló alkalmazásfejlesztést támogató szoftverrendszerek (pl. SAP Sybase PowerDesigner, Sparx Systems Enterprise Architect, az IBM Rational több termékcsaládja, stb.), ezért került sor erre a rövid ismertetőre. Az adatbázisok ma jobbára relációsak, de ezek az alkalmazásfejlesztést támogató szoftverrendszerek, felajánlják az elkészített UML modell automatikus leképzését relációs modellre, sőt, az adatbázist megvalósító SQL utasításokat is biztosítják a legtöbb használatban lévő konkrét adatbáziskezelő szoftverre.

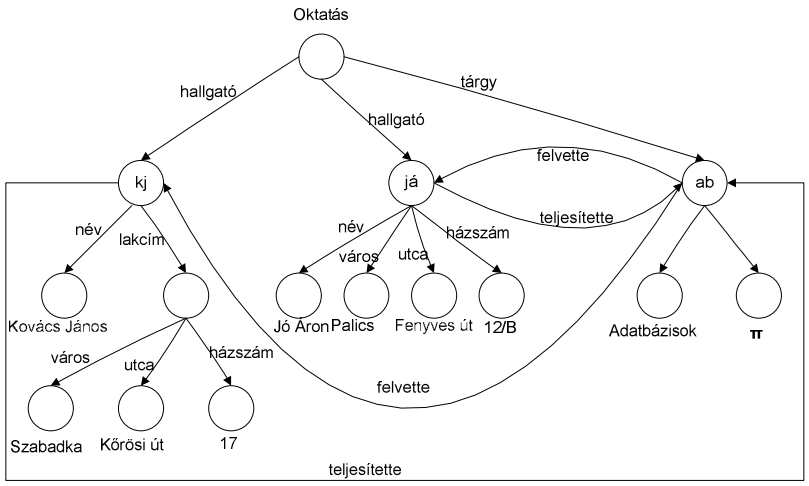
5.7.A FÉLIG-STRUKTURÁLT ADATMODELL

Az eddigiekben megismert adatmodelleknél az adatbázis szerkezete (az adatbázis sémája) már jó előre, és a felhasználói adatoktól teljesen függetlenül, elkülönülten/elkülönítetten kerül rögzítésre. A felhasználói adatok bevitelekor már minden szerkezeti elem (tábla, mezők, azok típusai és az össz megszorítás) ismert az adatbáziskezelő rendszer számára (mindent előzőleg le kellett írni és az adatbáziskezelő rendszerrel utasítások formájában meg kellett ismertetni). Az ilyen adatmodelleket szokás jól strukturált adatmodelleknek nevezni. Ezeknél a jól strukturált adatmodellek szerint működő adatbázisoknál az adattárolás fogalma csak a tényleges felhasználói adatok rögzítését, tárolását kell érteni (az adat/jelsor szemantikája/jelentése/értelme a modellben elfoglalt helye alapján értelmezett). A jól strukturált adatmodellek hatékony adatbáziskezelő rendszerek kiépítését teszik lehetővé.

A félig-strukturált adamodellben nincs előre meghatározott és elkülönítetten rögzített szerkezet (séma), itt a séma kikövetkeztethető az adatokból (az adatok önleírók, magukban

hordozzák az információt a sémáról). Ez egyrészt azt jelenti, hogy nemcsak a tényleges felhasználói adatok kerülnek rögzítésre, hanem a sémára vonatkozóak is, többszörös tárhelyet foglalva el a felhasználói adatok tárigényéhez viszonyítva. Másrészt, az említett tárigény-töblet mellett bonyolultabá és hosszadalmasabbá válnak a lekérdésfeldolgozások is. A felhasználó szempontjából azonban kedvező, hogy például saját igényei szerint bővítheti és használhatja a félig-strukturált adatmodellt (felvehet új attribútumokat, kapcsolatokat stb.), és ez nem lesz kihatással a többi felhasználóra. A félig-strukturált adatmodell lehetővé teszi a hasonló, de eltérő adatmodell szerint tárolt adatok transzparens összekapcsolását (két vagy több adatbbázis úgy illeszthető félig-strukturált adatmodell segítségével egymáshoz, mintha egy adatbázisban lennének) , és az adatok interneten történő megjelenítését, megosztását.

A félig-strukturált adatmodell grafikus ábrázolására fa jellegű (nem szükségszerűen szabályos fa szerkezetű) irányított gráfot használnak (5.25. ábra). A gráf csúcsai/csomópontjai levelek vagy belső csomópontok lehetnek. A leveleknek nincsenek gyerekei (kimenő élei): ezek tartalmazzák az atomi típusú adatokat. A belső csomópontoknak vannak kimenő élei (gyerekei): az élek névvel (cimkével) rendelkeznek, amelyek a struktúra leírására szolgálnak. A levelekre mutató élek az attribútum nevét tartalmazzák, a többi él pedig a két csomópont kapcsolódási módját (a csomópontok közötti viszonyt) írja le. A bemenő éllel nem rendelkező egyetlen belső csomópont a gráf gyökere (root, amely a az egész adatbázist képviseli), amelyből az összes csúcs elérhető.



5.25. ábra

5.7.1. XML – EXTENSIBLE MARKUP LANGUAGE

Az XML a HTML-hez hasonló cimke alapú jelölésrendszer, és akárcsak a HTML-t, dokumentumok leírására tervezték, de a félig-strukturált adatok leírására is alkalmas, és

manapság csak arra használják. A HTML-ben a dokumnetumrészek megjelenítési formáját rögzítik, az XML jelölők (tag-ek) a dokumentumrészek szemantikájáról, jelentéséről szólnak. A jelölők csúcsos zárójelben elhelyezett szövegtartalmak (a HTML-hez hasonlóan), melyek párban helyezkednek el (nyitó, pl. <valami> és zárójelölő, pl. </valami>). Egy nyitó és zárójelölő közötti részt (amely szöveget, HTML-részletet és tetszőleges számú, akár egymásba ágyazott jelölőpárt tartalmazhat) a jelölőkkel együtt XML-elemnek nevezik. Megengedett a zárójelölő nélküli elem is, amely csak attribútumokat tartalmazhat.

Példa:

Egyszerű XML-elem:

<hallgató> Jó Áron </hallgató>

Összetett XML-elem:

```
<hallgató>
    <vezetéknév> Jó </vezetéknév>
    <keresztnév> Áron </ keresztnév >
</hallgató>
```

Az XML nyelv sémanélküli (jólformált XML) és séma segítségével történő leírásra (érvényes XML/DTD és XML séma) is használható. A jólformált XML dokumentum nem támaszkodik semmilyen típusleírásra, amely meghatározná magának az XML dokumentumnak a felépítését: önmagában is teljes mértékben értelmezhető. A séma segítségével (XML dokumentumtípus-leírás alapján) értelmezhető XML dokumentumok számára a W3C (World Wide Web Consortium) kétfajta lehetőséget biztosított:

1. XML 1.0 DTD (Dokumentum Típus Definíció) specifikáció (érvényes XML-nek emlegetik) és
2. XML séma.

5.7.1.1. Jólformált XML

A jólformált XML dokumentum minden más segédeszköz és leírás nélkül is olvasható, értelmezhető és végrehajtható, nem támaszkodik semmilyen más leírásra, sem előírásra. Kötelezően deklarációval kezdődik és gyökérelemet is tartalmaz:

```
<? Xml version=”1.0” encoding=”utf-8” standalone=”yes” ? >
    <tetszőlegesJelölő>
        …
    </tetszőlegesJelölő>
```

Az XML dokumentum jólformáltságáról a deklarációban a *standalone=”yes”* rész tanúskodik, és azt jelenti, hogy a dokumentum önmagában is értelmezhető (önleíró). Az *encoding=”utf-8”* az alkalmazott kódolást (kódrendszert) jelöli (a nemzeti karaktereket is képes kódolni, igaz azokat két bájt hosszúságon). A *version=”1.0”* rész pedig arról tanúskodik, hogy milyen szabványnak felel meg az XML dokumentum tartalma. Az 5.24. ábrán látható félig-strukturált adatmodellt ábrázoló irányított gráf fa-szerkezetű részének (a felvette és teljesítette kapcsolatok nélküli) jólformált XML alakja a következőképpen néz ki:

```
<? Xml version=”1.0” encoding=”utf-8” standalone=”yes” ? >
<Oktatás>
        <Hallgató>
                <Név> Kovács János </Név>
                <Lakcím>
                        <Város> Szabadka </Város>
                        <Utca> Kőrösi út </Utca>
                        <Házszám> 17 </Házszám>
                </Lakcím>
        </Hallgató>

        <Hallgató>
                <Név> Jó Áron </Név>
                <Város> Palics </Város>
                <Utca> Fenyves út </Utca>
                <Házszám> 12/B </Házszám>
        </Hallgató>

        <Tárgy>
                <Tárgynév> Adatbázisok </Tárgynév>
                <Oktató> π </Oktató>
        </Tárgy>
</Oktatás>
```

5.7.1.2. Érvényes XML

Az XML dokumentumok értelmezésekor nagy könnyítés lehet, ha a dokumentum rendelkezik saját sémaleírással, hiszen abból kitűnik, hogy milyen elemekből építkezik a dokumentum, és hogyan épülnek/épülhetnek egymásba a feltüntetett elemek. Az érvényes XML esetében minden XML dokumentumhoz tartozik egy metanyelvszerű előírás- vagy szabályhalmaz, amelyet dokumentumtípus-definíciónak (Document Type Definition – DTD) hívnak. A DTD-t vagy az XML dokumentum elejére, a kötelező deklaráció elejére kell beszúrni, vagy külön állományban elhelyezni (ekkor a deklaráció utáni sorban kell hivatkozni rá). A DTD általános formája a következő:

```
<!DOCTYPE gyökérjelölő [
        <!ELEMENT elemnév (komponensek) >
        …
        <!ELEMENT elemnév (komponensek) >
 ] >
```

A hallgató egy lehetséges leírását tartalmazza a következő DTD:

```
<!DOCTYPE Hallgató [
        <!ELEMENT Hallgató (Hallgató*) >
        <!ELEMENT Hallgató (Név, Lakcím+, Tárgy) >
        <!ELEMENT Név (#PCDATA) >
        <!ELEMENT Lakcím (Város, Utca, Házszám) >
        <!ELEMENT Város (#PCDATA) >
        <!ELEMENT Utca (#PCDATA) >
        <!ELEMENT Házszám (#PCDATA) >
        <!ELEMENT Tárgy (Tárgy*) >
        <!ELEMENT Tárgy (Tárgynév, Oktató) >
        <!ELEMENT Tárgynév (#PCDATA) >
        <!ELEMENT Oktató (#PCDATA) >
 ] >
```

Jelölések:

- #PCDATA – (parsed character data) – egyszerű, nem beágyazott szöveges (karakteres) adat,
- * - az elem után írt *-jel az elem tetszőleges előfordulási számát jelöli (a nulla előfordulást is),
- + - az elem után írt +-jel az elem legalább egyszeri kötelező előfordulását jelenti

Ha most az 5.25. ábrán látható félig-strukturált adatmodellt ábrázoló irányított gráf fa-szerkezetű részének érvényes XML alakját kellene leírni a fentebbi DTD sémaleírás alapján (amely a Hallgató.dtd állományban került rögzítésre), akkor az a következő format öltené:

```
<? Xml version=”1.0” encoding=”utf-8” standalone=”no” ? >
<!DOCTYPE Hallgató SYSTEM „Hallgató.dtd”>

<Oktatás>
        <Hallgató>
                <Név> Kovács János </Név>
                <Lakcím>
                        <Város> Szabadka </Város>
                        <Utca> Kőrösi út </Utca>
                        <Házszám> 17 </Házszám>
                </Lakcím>
        </Hallgató>

        <Hallgató>
                <Név> Jó Áron </Név>
```

```
            <Város> Palics </Város>
            <Utca> Fenyves út </Utca>
            <Házszám> 12/B </Házszám>
        </Hallgató>

        <Tárgy>
            <Tárgynév> Adatbázisok </Tárgynév>
            <Oktató> π </Oktató>
        </Tárgy>
</Oktatás>
```

A két (jólformált XML és érvényes XML) leírás csupán a kötelező »fejlécben« különbözik: az érvényes XML »fejlécének« első sorában a standalone=«no« utal arra, hogy a dokumentum nem önálló, a teljes második sor pedig a DTD nevét és elhelyezését, megtalálhatóságát tartalmazza.

5.7.1.3. XML séma

A kötetlen (önleíró, előzetesen definiált forma/séma nélküli), jól definiált XML, valamint a létrehozó által előre meghatározott szabály (DTD) alapján készülő (nem önálló) érvényes XML dokumentumok mellett van egy harmadik típusú XML dokumentum-fajta is. Ezen dokumentumfajták nem saját készítésű (DTD) séma-definíciónak kell, hogy megfeleljenek, hanem egy általánosan elfogadott (szabványosított) és több lehetőséget biztosító (XML típusú) sémaleírásnak, amely a http://www.w3.org/2001/XMLSchema internetcímen található. Az alábbiakban látható XML séma alapján értelmezhető XML-séma dokumentum általános alakja nem hordoz forradalmi újdonságokat:

```
<? Xml version=”1.0” encoding=”utf-8” ? >
<xs:schema xmlns:xs=” http://www.w3.org/2001/XMLSchema”>
…
</xs:schema>
```

Az XML sémában megfogalmazható elemek általános alakja a következő:

```
<xs:element name = elemnév type = elem_típusa>
        Megszorítások és/vagy szerkezetfelépítési információk
</xs:element>
```

Példa: Az előző pontban leírt DTD-sémadefiníció egy részének (a Tárgy-leírásra vonatkozó rész)

```
<!DOCTYPE Tárgy [
        <!ELEMENT Tárgy (Tárgy*) >
        <!ELEMENT Tárgy (Tárgynév, Oktató) >
```

```
        <!ELEMENT Tárgynév (#PCDATA) >
        <!ELEMENT Oktató (#PCDATA) >
  ] >
```

egy lehetséges XML-séma leírása a következő:

```
  <? Xml version=”1.0” encoding=”utf-8” ? >
  <xs:schema xmlns:xs=” http://www.w3.org/2001/XMLSchema”>

        <xs:complexType name = “TárgyTípus”>
              <xs:sequence>
                    <xs:element name = „Tárgynév” type = „xs:string” />
                    <xs:element name = “Oktató” type = „xs:string” />
              </xs:sequence>
        </xs:complexType>

        <xs:element name = “Tárgy”>
              <xs: complexType >
                    <xs:sequence>
                          <xs:element name = „Tárgy”
                                type = “TárgyTípus”
                                minOccurs = „0” maxOccurs = „unbounded” />
                    </xs:sequence>
              </xs: complexType >
        </xs:element>

  </xs:schema>
```

A fenti példában összetett típus leírása látható és az előfordulások számának megfogalmazása is megtalálható.

5.7.1.4. XML adatbázisok

Az utóbbi időben gyakori az adatok tárolása XML formátumban. Ezen adatok adatbázisként való felhasználása (az adatoknak az adatbázis adatainak kinyerésére hasonlító eljárása) természetesen igényli a megfelelő lekérdező nyelvet. Az XML formában tárolt adatok kezelésére több, SQL-re hasonlító nyelv is megjelent. Közülük egyik sem olyan általánosan elfogadott, mint a relációs adatbázisoknál az SQL. A leggyakrabban használt nyelvek az XML-beni adatok kezelésére a következők: Xpath, az Xquery (amely az Xpath egy kiterjesztése) és az XSLT (Extensible Stylessheet Language for Transformations – Kiterjeszthető Stíluslap Nyelv). Másrészről megjelentek az SQL nyelv kiterjesztései XML adatok (adatbázisok) kezelésére, amelyeket általában SQL/XML jelöléssel illetnek, és az adatkinyerési (lekérdező) utasítások nagyon hasonlítanak a klasszikus SQL utasításokhoz.

6. MODELL-LEKÉPZÉSEK

Az adatbázisok tervezése fogalmi (konceptuális) és logikai szinten történik. A tervezés folyamata a konceptuális adatmodell kialakításával kezdődik. A leggyakrabban alkalmazott modell a fogalmi modellezésben az egyed-kapcsolat (E/K) modell. Az utóbbi időben egyre gyakrabban használják az objetummodellt és az UML modellt is a fogalmi szintű adatmodell kialakítására. Az összes fent említett adatmodell ismertetésre került az előző fejezetben. Az E/K és az UML modellhez nem készült adatbáziskezelő szoftver. Objektumorientált adatbázisok ugyan működnek már a gyakorlatban, de a relációs adatbázisok egyeduralmát egyenlőre még nem veszélyeztetik. A mai adatbáziskezelő rendszerek túlnyomó többsége relációs. Ebből egyértelműen következik, hogy bármely fogalmi szinten megfogalmazott adatmodell átalakításra/leképzésre szorul, hogy a kívánt adatbázis létrejöhessen. A konceptuális szinten megtervezett adatmodelleket relációs modellre kell átalakítani.

6.1. AZ E/K MODELL LEKÉPZÉSE RELÁCIÓS MODELLRE

Az E/K modell gazadagabb jelentéstartalmú a relációs modellnél. Az E/K modell gazdagabb szemantikájának köszönve az E/K modell fogalmai nem mindig írhatók le a relációs modell csupán egy fogalmával. A leképzés tehát nem lehet 1:1 típusú. Ez a tény ugyanúgy vonatkozik a műveletekre és a korlátokra is.

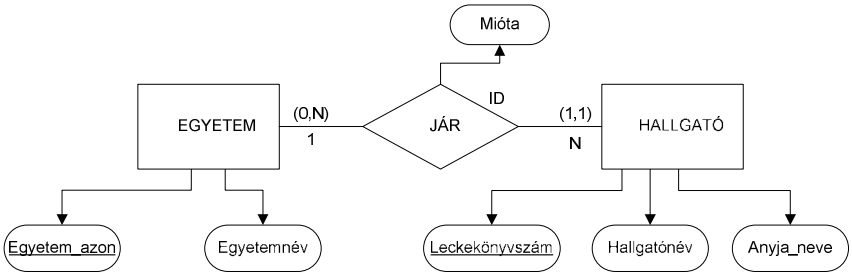
Többféle ajánlat is létezik az E/K modell relációs modellre való átalakítására. Egyesek nagyon elvont/elméleti megfogalmazás formájában adják meg az útmutatást, mások laza/szabad »lite«-os megfogalmazásban ajánlják a (nem éppen optimális) leképzést.

6.1.1. Egyedtípusok átírása relációkká

Az egyedtípusok (erős/független, egzisztenciálisan gyönge/függő és kapcsoló egyedtípusok – gerundok szerbül) átalakítása aránylag egyszerű folyamat. Minden egyedtípus relációsémává íródik át. Az egyedtípus neve a relációséma neve lesz az egyedtípus tulajdonságtípusainak nevét öröklik a reláció attribútumai. Az egyedtípus azonosítója (a nevét megtartva) a reláció kulcsává válik. A reláció attribútumai csak atomi felépítésűek lehetnek (nem lehetnek összetettek, nem lehet belső szerkezetük). Az attribútum elemi voltának eldöntésére nem elég ismerni az

attribútum nevét és jelentését: ismerni kell a felhasználásával kapcsolatosan támasztott elvárásokat is. Az attribútumok esetében ajánlatos lehet a minél részletesebb bontás.

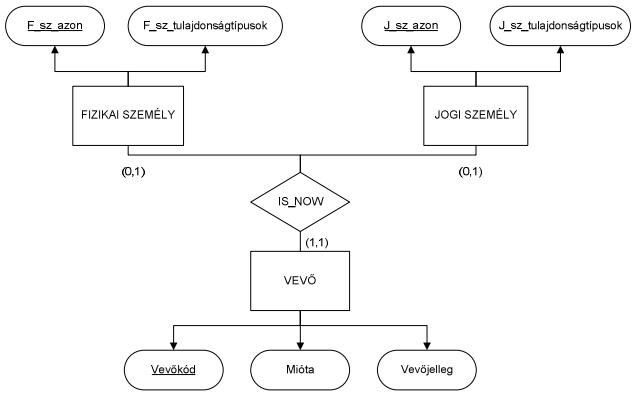
Az identifikációsan gyönge (magyar irodalomban egyszerűen csak gyenge egyedtípusnak nevezett) egyedtípusok átírása relációsémává egy árnyalattal sajátságosabb. Ezeknek a leképzése is relációsémát ad eredményként: a reláció neve megegyezik a gyenge egyedtípus nevével, attribútumnevei a gyenge egyedtípus tulajdonságtípus-neveivel, viszont a reláció (kulcs)attribútumai között megjelennek a gyenge kapcsolat(típus) túloldalán elhelyezkedő független egyedtípus kulcsattribútumai (kulcsa). Mivel a gyenge egyedtípust képviselő reláció kulcskialakításánál a gyenge kapcsolat szolgáltatta a kulcs egy részét, a teljes kapcsolatszerkezet átírásánál a gyenge kapcsolattípus konkrétan (explicite) meg sem jelenik (6.1. ábra).



6.1. ábra

EGYETEM(Egyetem_azon, Egyetemnév)
HALLGATÓ(*Egyetem_azon*, Leckekönyvszám, Hallgatónév, Anyja_neve, Mióta_jár)

A kategória, annak ellenére, hogy »fejreállított« »az-egy«, vagy inverz öröklési



6.2. ábra

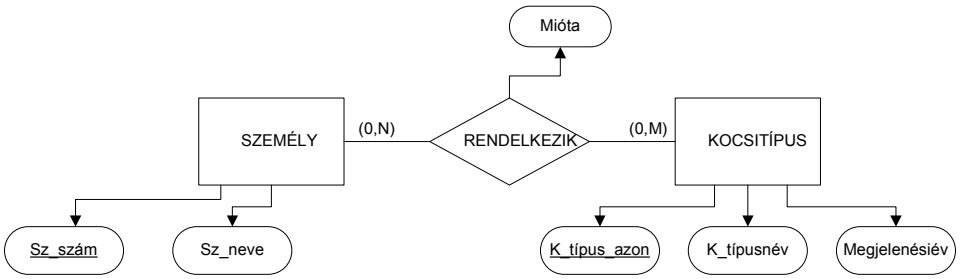
FIZIKAI SZEMÉLY(F_sz_azon, F_sz_tulajdonságtípusok)
JOGI SZEMÉLY(J_sz_azon, J_sz_tulajdonságtípusok)
VEVŐ(Vevőkód, Mióta, Vevőjelleg, F_sz_azon, J_sz_azon)

hierarchiának néz ki, mégsem sorolható az alábbiakban ismertetésre kerülő osztályhierarchiába. Attól ugyanis szerkezetben és lényegileg is nagyon eltérő. A kategória azonos szerepkörben

megjelenő, nagyon különböző felépítésű egyedfőtípus-előfordulásokat gyűjt egy csoportba és csak mintegy »összekötő« szerepet játszik. Az őt képviselő relációséma felépítésében csak egy mesterségesen kialakított kulcs és a különböző egyedfőtípusok számával megegyező külső kulcsa van (6.2. ábra). Értelemszerűen ezekre a külső kulcsokra nem alkalmazható (tilos) a kötelezően kitöltendő megszorítás (NOT NULL).

6.1.2. Kapcsolattípusok leképzése relációkká

A legegyszerűbb esetben a kapcsolattípusnak is relációséma felel meg (bár vannak esetek, amikor ettől az elvtől eltérő megoldások jobban igazodnak az egyes kapcsolattípusok jellegéhez). A kapcsolatot képviselő reláció neve a kapcsolat nevét veszi fel. A kapcsolatot képviselő reláció attribútumait a kapcsolattípus által összekötött egyedtípusok egy-egy azonosítója/az azokat képviselő relációk kulcsa (amennyiben több is van belőlük) adja, valamint ha a kapcsolattípusnak voltak saját tulajdonságtípusai (Ullman-Stanford-iskola), akkor azok is attribútumként jelennek meg a kapcsolatnak megfeleltetett relációban. A reláció kulcsát a költöztetett relációk kulcsai alkotják, amelyek így összetett kulcsot alkotnak (6.2. ábra).



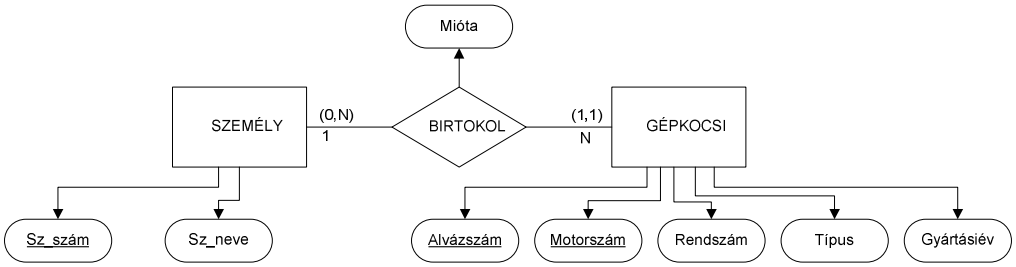
6.3. ábra

SZEMÉLY(Sz_szám, Sz_neve)
KOCSITÍPUS(K_típus_azon, K_típusnév, Megjelenésiév)
RENDELKEZIK(Sz_szám, K_típus_azon, Mióta)

Amenyiben a kapcsolattípus kapcsolatfoka (maximális kardinalitása) az 1:N típuscsaládba tartozik, akkor a kapcsolattípus nem igényel új relációsémát (persze az új relációséma megadása is elfogadható a kapcsolattípus átírására), hanem az „egyes” oldalom elhelyezkedő reláció kulcsának „költöztetése” a többes oldalon elhelyezkedő relációba (a kapcsolattípus tulajdonságtípusaival egyetemben-Stanford-iskola Ullman) is jó, elfogadható megoldásnak számít. A többes oldalra „átköltöztetett” kulcs ott egyszerű (nem kulcsjellegű attribútum – az attribútum nem lesz a kulcs része a többes oldalon) külső kulcsként fog megjelenni (6.4. ábra).

A változásokra érzékeny valós rendszerek esetében talán bölcsebb választani a kevésbé jellegzetesebb és jobb megoldást (a kapcsolattípus gondolkodás nélküli átírását relációvá). Az

ilyen megoldás ugyan több tárhelyet igényel, és a feldolgozási idő is hosszabb, ugyanakkor a változások (esetleg tervezés közbeni tévedések) által kikényszerített módosítás könnyebben



6.4. ábra

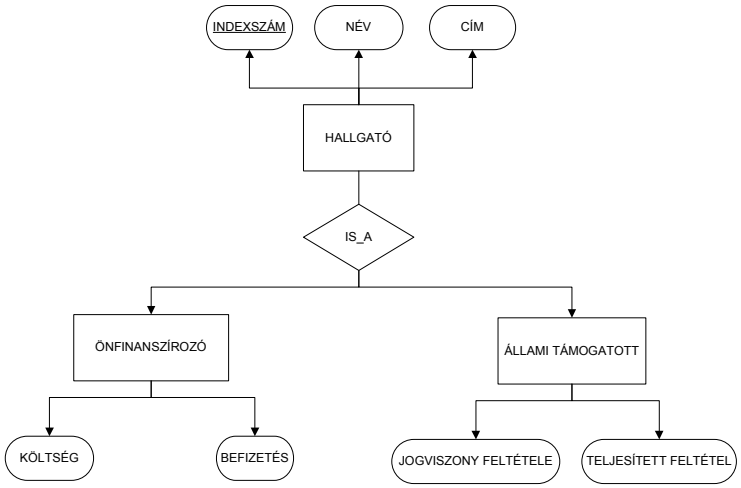
SZEMÉLY(Sz_szám, Sz_neve)
GÉPKOCSI(Alvázszám, Motorszám, Rendszám, Típus, Gyártásév, Tulajdonos-Sz_szám, Mióta)

végezhető.

6.1.3. Öröklési kapcsolattípusok vagy osztályhierarchia átalakítása

Az „IS_A" vagy „az_egy" típusú kapcsolatok esetében az egyedaltípusok identifikációsan gyönge (gyenge) egyedtípust képviselnek. A teljes osztályhierarchia átalakítása háromféle megközelítésben is elvégezhető:

1. Az E/K szempontok követésének elve (E/K elv) szerint: minden egyedaltípus (alosztály) számára készül egy relációséma, amelyben attribútumként szerepel az egyedfőtípus (szuperosztály) valamennyi tujaldonságtípusa és az egyedaltípus (alosztály) saját tulajdonságtípusai (6.5. ábra).



6.5. ábra

HALLGATÓ-ÖNFINANSZÍROZÓ(indexszám, név, cím, költség, befizetés)
HALLGATÓ-ÁLLAMI_TÁMOGATOTT(indexszám, név, cím, jogviszony_feltétele, teljesített_feltétel)

2. Az egyedek osztályhoz tartozó objektumként való kezelésének elve (objektumelv) szerint: az egyedfőtípus (szuperosztály) relációsémává íródik át (minden tulajdonságtípusa, a szerepkörének megfelelően attribútumként jelenik meg a relációban-az azonosítót alkotó tulajdonságtípusok kulcsattribútumok lesznek), ezenkívül minden saját/önálló tulajdonságtípussal rendelkező egyedaltípus (alosztály) relációséma formájában is megjelenik (az egyedfőtípus azonosítójával, mint kulccsal és a saját/önálló tulajdonságtípusokat képviselő attribútumokkal). Ez a leképzés a 6.6.

HALLGATÓ(<u>indexszám</u>, név, cím)
HALLGATÓ-ÖNFINANSZÍROZÓ(<u>indexszám</u>, költség, befizetés)
HALLGATÓ-ÁLLAMI_TÁMOGATOTT(<u>indexszám</u>, jogviszony_feltétele, teljesített_feltétel)

6.6. ábra

ábrán látható.

3. A NULL-értékek használatának elve (nullértékes elv) szerint: egyetlen relációséma keletkezik, amely tartalmazza az össz, egyedfőtípus (szuperosztály) és egyedaltípus (alosztály) valamennyi tulajdonságtípusát képviselő attribútumot (a szerepkörüknek megfelelően). A relációséma a 6.7. ábrán található.

HALLGATÓ(<u>indexszám</u>, név, cím, költség, befizetés, jogviszony_feltétele, teljesített_feltétel)

6.7. ábra

Az alkalmazott átalakítási elvek több szempontból is összevethetők, az alábbiakban két szempont szerint kerülnek vizsgálatra az átalakítás módszerei, a helyigény és a keresés szempontjából:

1. Tárolókapacitási igény szerint összehasonlítás.
    a. Az első (az E/K szempontok követésének elve szerinti) leképzés a legtakarékosabb, hiszen nincsenek adatismétlések.
    b. A második (az egyedek osztályhoz tartozó objektumként való kezelésének elve szerinti) átírás a kulcsismétlések miatt kevésbé takarékos az elsőnél. A kapcsolat erősségét/opcionalitását ekintve kulcsok minimálisan kétszer fordulnak elő teljes osztályhierarchia esetén (a kapcsolaterősség/minimális kardinalitás értéke 1), részleges osztályhierarchia esetén vannak olyan szuprosztály-előfordulások, amelyek nem jelennek meg egyetlen alosztályban sem (a kapcsolaterősség/minimális kardinalitás értéke 0), így ott a kulcsértékelőfordulás nem kétszeres. A kapcsolatfokot (maximális

kardinalitást) vizsgálva disjunkt osztályhierarchia esetén (a kapcsolatfok/maximális kardinalitás értéke 1) a kulcsértékek maximálisan kétszer fordulhatnak elő (teljes osztályhierarchia esetén, részleges osztályhierarchia esetén a duplázódás nem valósulhat meg). Az átfedő vagy átlapolt osztályhierarchia (a kapcsolatfok/maximális kardinalitás értéke N) esetében nem becsülhető egyszerűen határérték a kulcsismétlődések számát illetően: a kulcsértékek annyi plusz példányban ismétlődnek, ahány példányban a szuperosztály előfordulása megjelenik az alosztályokban.

c. A harmadik típusú (a NULL-értékek használatának elve szerinti) leképzés a tárigény szempontjából látszatra a legpazarlóbb: sok a kitöltetlen, NULL értékű mező. A ritka relációk takarékos/hatékony kezelésére már elég régóta berendezkedtek az adatbáziskezelő rendszerek, vagyis ez az átalakítási módszer sem tekinthető elvetendőnek.

2. Keresések szerinti összehasonlítás.
   a. Az első módszer esetében az összes relációban keresni kell az egyedelőfordulást, de már az eslő találat biztosítja az összes ismeretet (adatot) a keresett egyedelőfordulásról. Hatékonyságát tekintve átlagos időigényt véve alapul ez a módszer a középszerű (lassabb és kevésbé hatékony, mint a harmadik leképzés eredményét képviselő relációs modellrészlet).
   b. A második eljárás esetében az egyedelőfordulásról megtalálhatók az általános (a szuperosztályt képviselő relációban elhelyezkedő) adatok, de az alosztályokban elhelyezkedő adatok felkutatásához az összes alosztályokat képviselő relációkban keresni kell (esetleg több relációban is sikertelenül). Ez a megoldás a legkevésbé hatékony, egyben a legbonyolultabb is.
   c. A harmadik leképzési módszer biztosítja a legegyszerűbb keresést, hiszen csak egyetlen relációban kell keresni, és a találat az összes adatot biztosítja. Ez a módszer biztosítja átlagértékre nézve a leggyorsabb keresést.

6.2. AZ OBJEKTUMORIENTÁLT ADATMODELL LEKÉPZÉSE RELÁCIÓS MODELLÉ

Az objektumorientált adatmodell átalakítása relációs modellé hasonlóképpen történik, mint az E/K modellé, ám figyelembe kell venni az objetumorientált adatmodell sajátságait. Az osztályoknak relációkat kell megfeleltetni, ugyanúgy, mint a kapcsolatoknak.

Az osztályok átalakításánál ügyelni kell arra, hogy a relációnak legyen egy kijelölt kulcsa (az osztályok esetében nem kötelező a kulcs léte), ha az osztálynak nincs kulcsa, akkor a reláció számára mesterségesen kell létrehozni egyet. Az osztályok attribútumai/tulajdonságai lesznek a reláció attribútumai, a reláció örökölni fogja az osztály nevét. A relációattribútumok atomiságát biztosítani kell, hiszen az ODL-ben az attribútumok összetett típusúak is lehetnek (struktúra, halmaz, multihalmaz, lista). Ilyen esetben előfordulhat az, hogy a kapott reláció elfogadhatatlan normálformát ölt, ami azt jelenti, hogy utólagos ellenőrzésre (normalizációra) lesz szükség.

Az objektumos kapcsolatok leképzésénél arra kell ügyelni, hogy az ODL-ben a kapcsolat az inverzével egyetemben jelenhet meg, és a relációs modellre való leképzésnél a kapcsolatpárra (kapcsolat és inverz kapcsolat) kell létrehozni az új relációt (a kapcsolatban lévő osztályok kulcsaiból).

6.3. AZ UML OSZTÁLYDIAGRAMJÁNAK ÁTALAKÍTÁSA RELÁCIÓS MODELLÉ

Általánossában elmondható, hogy az UML osztálydiagramjának átalakításakor minden osztálynak egy reláció felel, ahol a reláció örökli az UML osztály nevét, és az osztályattribútumok relációattribútumokká válnak (az elnevezésük/nevük megtartásával). A kapcsolatok/társítások számára azonos nevű relációsémák jelennek meg a relációs modellben, és a relációséma attribútumai, mint ahogy ez elvárható, a kapcsolatban lévő osztályok kulcsattribútumai lesznek. Amennyiben a társításhoz osztály is kapcsolódik, úgy annak attribútumai is megjelennek a társítás/kapcsolat számára létrehozott relációban.

Az UML-lekézés specifikumai a következők:

1. Az osztályhierarchia átírásáról már volt szó az E/K „IS_A” vagy „az_egy” öröklési kapcsolat boncolgatásánál (leképzésénél), és az ott leírt három módszer (E/K-elvű, objektumelvű és nullértékes) itt is alkalmazható. A diszjunkt alosztályok esetében az objektumelvű módszer ajánlható, ha emellett még teljes is az osztályhierarchia, akkor csak a hierarchia levélosztályait kell átírni relációvá. Ha az osztályhierarchiában akár egy szinten is átlapolás található, akkor az E/K módszer az ajánlott, ami „csúnya” adatbázissémához vezet.
2. Aggregáció és kompozíció (mint társítástípus) átalakítása relációvá. Az „egy-több” vagy „sok-egy” típusú kapcsolatokat képviselő aggregációkat és kompozíciókat az UML-ben nem kötelező névvel illetni, így ez a feladat az átalakításkor válik kötelezővé, illetve csak válna kötelezővé, ha a társítást relációval képviseltetnénk

(ahogy ez az első bekezdésben, az általánosságokról írt megfogalmazásban áll). Itt a gyakorlatias ajánlat a „kulcsköltöztetés”, vagyis az „az_egy” típusú öröklési kapcsolat E/K elv szerinti átírása: a többes (nem rombusz végű) oldalt képviselő reláció bővítése az egyes (a rombuszvégű) oldalt képviselő reláció kulcsával. Aggregáció esetében a többes oldalt képviselő relációsémában megjelenő (nem kulcsszerepű) külső kulcs (vagyis a külső kulcs nem alkotórésze a relációséma kulcsának) a kitöltése nem kötelező, viszont a kompozíció esetében igen (NOT NULL megszorítás a külső kulcsra). Az aggregációra és kompozícióra, mint társításra nem jön létre külön relációséma.

3. A gyenge egyedtípusnak megfelelő UML-kompozíció (olyan kompozíció, amelynek a nemrombusz végű oldalán PK feliratú kis doboz található – 5.23. ábra) esetére érvényes a fenti leírás, azzal a megszorítással, hogy a többes (nem rombusz végű) oldalt képviselő reláció bővítésénél az egyes (a rombuszvégű) oldalt képviselő reláció kulcsával, a költöztetett kulcs kulcsszerepű külső kulccsá válik.

7. NORMALIZÁCIÓ

Az adatbázistervezés általánosan elfogadott (az 5.1. ábrán grafikusan ábrázolt) tervezési folyamatát követve a fogalmi tervezést (a tervezésre alkalmas fogalmi adatmodell alkalmazásával) befejezvén a kapott modell leképzését/átírását kell elvégezni relációs modellre. Az így kapott relációk (relációsémák) már megvalósíthatók bármelyik relációs adatbáziskezelő rendszerben, tehát az adatbázis már fizikailag is létrehozható. A kapott relációsémákat, esetleg a fizikailag megvalósított adatbázistáblákat vizsgálva azonban komoly hiányosságok is előfordulhatnak. Ezek a hiányosságok a relációséma belső szerkezetének vizsgálatával és javításával teljes mértékben kiküszöbölhetők. A relációk szerkezeti vizsgálatának és javításának folyamatát normalizálásnak nevezik. Példaként a következő Hallgató relációséma szolgáljon:

Hallgató (leckek_szám, név, szül_év, ir_szám, helység, utca-hsz, t_körzet, telefon)

A redundancia talán nem tűnik fel első ránézésre, de ha az extenziót (a hallgató-előfordulásokat) is ábrázoljuk (akár csak néhány táblasor-érték feltüntetésével), már nem lesz gond az adat-ismétlődések felismerésével (7.1. ábra):

Hallgató

Leckek_szám	Név	Szül_év	Ir_szám	Helység	Utca_hsz	T_körzet	Telefon
A-1	A	1991	24000	Szabadka	Zzz	/024	xxx
A-2	B	1994	24413	Palics	Vvv	024	yyy
A-3	C	1992	21000	Újvidék	Ccc	021	vvv
A-4	D	1994	24000	Szabadka	Ggg	024	hhh

7.1. ábra

Az irányítószám-értékek, a helységnevek és a telefonkörzetszámok (az előhívószámok) felesleges ismétlődése egyértelmű. A relációs adatmodellben elkerülhetetlen, sőt szükségszerű az adatok ismétlése (erről már volt szó a harmadik fejezetben), hiszen nem létezik a kapcsolat fogalma (egy olyan mechanizmus amely összekötne két relációsémát). A kapcsolat két relációséma között az egyik reláció kulcsának a másik relációban való elhelyezésével jön létre. A normalizálás célja, hogy csak ilyen elengedhetetlen adatismétlések maradjanak a relációs adatmodellben.

A felesleges redundancia a következő három, sokkal nagyobb gondot okozó (adatkarbantartási) problémával jár együtt:

1. Adatbeszúrási probléma,

2. Adatmódosítási probléma és
3. Adattörlési probléma.

Adatbeszúrási problémaként jelenik meg a fenti hallgató relációsémát megvalósító adatbázisban, hogy nincs lehetőség konkrét helységnév bevitelére egészen addig, amíg abból a helységből nem kerül a nyilvántartásba legalább egy hallgató (másszóval helységet beszúrni hallgatótól függetlenül nem lehet).

Adatmódosítási probléma akkor merül(het) fel a fenti hallgató relációsémát megvalósító adatbázisban, ha módosítani kell mondjuk a helységnevet (volt már rá példa, nem is olyan régen: Titograd-Podgorica, Svetozarevo-Jagodina, Titov Veles-Veles, Titov Vrbas-Vrbas): a javítást nem elég csupán egy helyen kell megejteni (mert akkor elentmondásossá válhat az adatbázis tartalma: pl. egyes helyeken még a helység régi neve szerepel-Svetozarevo, más helyeken már a módosított tartalom-Jagodina található), hanem át kell fésülni az egész adatbázis-táblát az össz régi tartalom módosítása érdekében.

Adattörlési probléma jelentkezik például abban az esetben, olyan sor kerül törlésre, amelyben adatbázis szinten egyszer előforduló adat/adatok találhatók. Például, ha törlésre kerül az A-3 leckekönyvszámú hallgató a fenti táblából, akkor nyoma veszik annak az ismeretnek is, hogy Újvidék irányítószáma 21000.

Ezek a fent említett problémák egyetlen okra vezethetők vissza: a relációsémában a szükséges, más relációsémák felé kapcsolatot biztosító adatismétlések mellett más fogalmakat (eltérő lényegeket) leíró adatok is előfordulnak.

A továbbiakban ismertetésre kerül azon függések (függőségek) halmaza, amely segítségével vizsgálható és minősíthető a relációsémák belső felépítése/szerkezete a redundancia és a vele együttjáró karbantartási rendellenességek (anomáliák) szempontjából. A relációk minőségének osztályozása a normálformák segítségével írható le, ahogy ez majd a későbbiekben részletezésre kerül.

7.1. FÜGGŐSÉGEK

A függőségek (az attribútumok közötti függések) csak egy adott relációsémán belül értelmezhetőek. A többfajta függőség közül a két legtöbbet használt a funkcionális és tranzitív függőség. Erre a két fajta függőségre lesz szükség a harmadik normálforma kialakításához (ezt a minőségi szintet általánosan elegendőnek tartják a relációséma belső szerkezetének meghatározásánál). A harmadik normálformát kielégítő relációsémák minőségben az egyedtípusoknak felelnek meg.

Az R relációséma Y attribútuma (attribútumhalmaza) funkcionálisan függ (FD – functional dependence) az R relációséma X attribútumától (attribútumhalmazától), abban (és csakis abban) az esetben, ha az R relációséma minden előfordulásában X minden értékét (értékhalmazát) mindig csak egy (és ugyanaz az) Y érték (értékhalmaz) kíséri. Más megfogalmazásban ha az extenzióban két sor megegyezik a X értéken (értékeken), akkor ugyanaz a két sor Y értéken (értékeken) is megegyezik. Az előbbiekben megfogalmazott funkcionális függőség az f: X→Y vagy rövidítve az X→Y jelöléssel is megadható, ahol az f a funkcionális függőség megnevezése (a funkcionális függőség nevét általában nem tüntetik fel). Maga a jelölés szavakkal leírva így szól: X funkcionálisan meghatározza Y-t, vagy Y funkcionálisan függ X-től. Általános esetben az X és az Y tulajdonságtípus-halmazokat is jelölhetnek (ahogy ez már a megfogalmazásból is kivehető volt).

A funkcionális függőség fordított irányban függetlenséget hordoz, azaz, ha X→Y, akkor érvényes, hogy Y-/->X (a függetlenséget a „-/->” jelsor ábrázolja). Könnyen belátható, hogy a funkcionális függőség a két attribútum közötti hierarchikus összefüggést hordozza magában.

In medias res. Példa: Leckekönyvszám→Hallgató_neve. A két attribútum értékkészletének értékei között N:1 típusú kapcsolat van. Az azonos nevű hallgatók különböző leckekönyvszámot kapnak (N), de minden leckekönyvszám csak egy (1) hallgatóra vonatkozik.

A funkcionális függőség lehet erős (ha minden bal oldalon lévő érték-előforduláshoz, meghatározóhoz kell kötődnie jobb oldali előfordulásnak, meghatározottnak) vagy gyenge (vannak olyan X-értékek, amelyekhez nem tartoznak Y-értékek).

Példa: a Leckekönyvszám→Hallgatónév funkcionális függőség erős, mert minden leckekönyvszámhoz tartoznia kell egy hallgatónévnek. A következő funkcionális függőség viszont gyengének tekinthető, hiszen a Leckekönyvszám→Megítélt_kölcsönfajta funkcionális függőség értelmezése alapján vannak olyan egyetemisták, akik nem igényelnek kölcsönt (ezek szerint nincs megítélt kölcsönfajtájuk).

Last but not least: a funkcionális függőség lehet teljes (full dependence) vagy részleges (partial dependence). Ha a függőség bal oldalán egyetlen (meghatározó) attribútum áll, akkor egyszerű (elemi) függőségről van szó, ha viszont a bal oldal több tényezőből (attribútumból) áll, akkor azt összetett függőségnek nevezik. A funkcionális függőségek teljes vagy részleges volta csak az összetett függőségek esetében vizsgálható/vizsgálandó. Az összetett funkcionális függőség akkor teljes, ha a jobb oldal (a meghatározott) a teljes (összetett) bal oldaltól (a meghatározótól) függ. A teljes funkcionális függőség esetén a bal oldal bármelyik összetevőjének eltávolításával a funkcionális függőség megszűnik. Amennyiben az összetett

meghatározó (bal oldal) bármely tagját elhagyva a funkcionális függőség nem szűnik meg, részleges funkcionális függőségről van szó.

Példa: Legyen adott az alábbi szerkezettel megadott SIKERES_VIZSGA relációséma, és azon belül két kiválasztott funkcionális függőség:

SIKERES_VIZSGA(Leckekönyvszám, Tantárgykód, osztályzat, ..., Tantárgy_neve)

Leckekönyvszám+Tantárgykód →osztályzat

Leckekönyvszám+Tantárgykód → Tantárgy_neve.

Az első funkcionális függőség nyilvánvalóan teljes, mert a bal oldal egyetlen tulajdonságtípusa sem határozza meg egymagában a jobb oldalt. A második funkcionális függőség azonban részleges, mert a leckekönyvszám eltávolítása a meghatározóból (bal oldal) nem szünteti meg a funkcionális függőséget (a Tantárgykód önmagában is meghatározza a Tantárgy nevét).

Szemantikailag az f: X→Y funkcinális függőség időben változó függvénynek tekinthető. Ez azt jelenti, hogy a meghatározó (bal oldal) és a meghatározott (jobb oldal) is változnak az időben, ezért más-más időpontban az X attribútumhalmaz ugyanazon előfordulás-értékhalmazához az Y attribútum különböző előfordulás-értékhalmaza tartozhat. Ehhez a ténymegállapításhoz a következő példa (funkcionális függőség) kapcsolódik: Kocsirendszám→Kötelező_szervizek_száma. A meghatározóhoz kapcsolt érték (a meghatározott értéke) változik az időben, hiszen a gépkocsi használati ideje folyamán az elvégzett kötelező szervizek száma folyamatosan növekszik.

A funkcionális függőség intenzionális megszorításnak, korlátnak tekinthető, mert egy adott relációséma attribútum-halmazára vonatkozik. A funkcionális függőségek felderítéséhez a valós rendszert (az ott használt fogalmakat) figyelmes elemzésnek kell alávetni az egyedtípusoknak megfeleltetett relációsémák attribútumainak feltárásával, adattípusainak megfogalmazásával és értéktartományaik meghatározásával.

Az attribútumot vagy az attribútumok olyan csoportját, amelyik az adott relációséma minden előfordulás-értékére különböző/eltérő értéket vesz fel, és ezzel azonosítja a reláció-előfordulásokat, kulcsnak, esetleg kulcsjelöltnek nevezzük. Az kulcsnak minimális felépítésűnek kell lennie, másszóval a kulcs egyetlen valódi részhalmaza sem töltheti be az kulcs (kulcsjelölt) szerepét. Más megfogalmazásban ez azt jelenti, hogy minden funkcionális függőségnek, amelyekben a meghatározó a kulcs, teljesnek kell lennie.

Ha az egyik (R_1) relációsémában szerepel egy olyan (akár másodlagos vagyis leíró atribútum, akár elsődleges, kulcsban szereplő) attribútum, amely egy másik (R_2) relációsémában kulcs szerepben van jelen, akkor a szóban forgó attribútumot (attribútumhalmazt) az R_1

relációsémában külső (idegen) kulcsnak nevezzük, mert az R_1-ben ugyan nem kulcs, de valahol máshol igen.

A kulcs állhat egy vagy több attribútumból. Az előbbi kulcsot egyszerű (elementary) kulcsnak, az utóbbit összetettnek (compound v. composite) nevezik. Ha a reláció kulcsa két vagy több (különböző relációból eredő) külső kulcsból és a relációséma legalább egy saját attribútumából épül fel, akkor azt hierarchikus kulcsnak nevezik. Ha a reláció kulcsát csak külső kulcsok alkotják, akkor azt a kulcsot összetettnek tekintik. (Egyesek a különböző relációból eredő külső kulcsokból álló kulcsot is hierarchikus kulcsnak tekintik.)

A tranzitív függőség lényege a következő. Jelölje A a relációséma teljes attribútumhalmazát, valamint az X és Y az olyan attribútumhalmazokat, amelyekre érvényes, hogy $(X,Y \subseteq A)$ és legyen érvényes az $X \rightarrow Y$ funkcionális függőség. Az Y attribútum(halmaz) akkor függ tranzitíven az X-től, ha létezik az attribútumoknak egy olyan Z-vel jelölt halmaza, amelyre igaz, hogy $Z \subseteq A$, és érvényes a következő két funkcionális függőség: $X \rightarrow Z$ és $Z \rightarrow Y$, de, ugyanakkor, természetesen (a funkcionális függőségek megszorításai következtében) a $Z \rightarrow X$ és $Y \rightarrow Z$ funkcionális függőségeknek nem szabad fennállniuk.

A tranzitív függőség szemantikus leírása (magyarázata) a következő: a relációséma leíró (nem-kulcs) Y attribútumhalmaza csak akkor függ tranzitíven a reláció kulcsától (X), ha őt (Y) a kulcson (X) kívül, egy másik, Z attribútumhalmaz is funkcionálisan meghatározza, amely (Z) maga is funkcionálisan függ a kulcstól (X).

A fejezet elején megfogalmazott karbantartási anomáliák nem elméleti fejtegetések, hanem mindennapi gyakorlati problémák kiváltó okai. Megszüntetésükre a relációk felbontását (dekompozíció) szokás alkalmazni. A dekompozíció az az eljárás, amelyik a relációsémákat egyenként, egymástól függetlenül vizsgálja, és a kedvezőtlen (nem megfelelő) belső szerkezetű relációkat attribútumainak szétosztásával (az eredeti reláció attribútumainak új relációsémákba – két új relációsémába – való vetítésével - projekcióval) megfelelőbb belső szerkezetűre hozza (bontja).

Algoritmus-magyarázat[1]:

Jelölje A az R_1 relációséma attribútumhalmazát, és legyen két olyan, $X \subseteq A$ és $Y \subseteq A$ attribútum-részhalmazunk, hogy $X \cup Y=A$. Az X és Y akkor alkotnak dekomozíciós párt az R_1 relációra vonatkozóan, ha az R_1 két, modjuk t_i és t_j (t_i = ext (R_1) és t_j = ext (R_1)) reláció-előfordulásának metszethalmaz-egyenlőségéből az következik, hogy létezik olyan reláció-előfordulás (t), amelyre $t=t_i$ az X (rész)halmazon, és ugyancsak létezik olyan t reláció-

[1] A dekompozíció egyszerűsített definíciójának alapjául szolgáló forrás: [Mogin???25], 82. oldal.

előfordulás, amelyre $t=t_j$ az Y (rész)halmazon. A meghatározás (definíció) formális-halmazelméleti leírása a következő: ha

$\forall t_i, t_j \in R_1$ esetén, melyre $t_i(X \cap Y) = t_j(X \cap Y)$ $\exists t \in R_1$, melyre $t(X) = t_i(X)$ és $t(Y) = t_j(Y)$,

akkor X és Y dekompozíciós párt alkotnak az R_1 relációsémára vonatkozóan.

Magyarázat-de felbontási algoristmus nélkül[2]:

Jelölje az $\{A_1, A_2, \ldots, A_n\}$ az R_1 relációséma attribútumhalmazát. Az R_1 relációséma úgy bontható fel dekompozíciós párt alkotó $X\{B_1, B_2, \ldots, B_n\}$és $Y\{C_1, C_2, \ldots, C_m\}$ relációkra, hogy

1. $\{A_1, A_2, \ldots, A_n\} = \{B_1, B_2, \ldots, B_m\} \cup \{C_1, C_2, \ldots, C_k\}$
2. $X = \pi_{B1, B2, ..., Bm}(R)$
3. $Y = \pi_{C1, C2, ..., Ck}(R)$

A felbontás megfelelő (Boyce-Codd normálformájú felbontás) módjáról/algoritmusáról a szakirodalomban találhatók leírások (Ullman, Buza), itt nem krül ismertetésre.

Példa[3]: Az R_1 relációséma attribútum-részhalmazai legyenek a következők: $X = \{A_1, A_2, A_3, A_4\}$ és $Y = \{A_3, A_4, A_5, A_6\}$, vagyis a metszet $X \cap Y = \{A_3, A_4\}$, és legyen az R_1 relációséma reláció-előfordulásainak a részhalmaza a 7.2. ábrán feltüntetett értékhalmaz.

	A_1	**A_2**	**A_3**	**A_4**	**A_5**	**A_6**
t_1	**a**	**b**	**c**	**d**	**e**	**f**
t_2	**g**	**f**	**c**	**d**	**h**	**e**
t_3	**a**	**b**	**c**	**d**	**h**	**e**
t_4	**.**	**.**	**.**	**.**	**.**	**.**

7.2. ábra

A 7.2. ábra reláció-előfordulásai alapján arra lehet következtetni, hogy az X és Y részhalmazok az R_1 relációséma dekompozíciós párját alkotják, mert:

1. $t_1 (X \cap Y) = t_2 (X \cap Y) = \{c, d\}$ és
2. $t_3 (X) = t_1 (X)$ és $t_3 (Y) = t_2 (Y)$

Heath-tétel – a gyakrolatban kiválóan alkalmazható algoritmus/eljárás[4]:

A dekompozíciós (bontási) eljárás gyakorlatban leghasználhatóbb útmutatását adja a Heath-tétel a veszteségmentes (lossless vagy nonloss) relációséma-bontás megvalósítására. A tétel nagyon egyszerű és érthető. Íme egy megfogalmazása:

[2] Ullman str.91, Buza str.89
[3] Forrás: [25], 82. oldal.
[4] Lazarevic, Wikipedia

Jelölje az A az R_1 relációséma attribútumhalmazát, és legyen érvényes az X→Y funkcionális függőség, ahol X és Y ugyancsak attribútumhalmazokat jelölnek. Ekkor bizonyosan érvényes a következő kifejezés: $R = \pi_{XY}(R) \bowtie \pi_{XZ}(R)$, ahol az $\bowtie$ operáció a természetes összekapcsolás jele (ahogy a harmadik fejezetben bevezetésre került ez az operáció), másrész érvényes, hogy a Z = A – XY .

Más megfogalmazásban, ha adott az $R_1(X,Y,Z)$ relációséma, ahol az X, Y és Z attribútumhalmazok, emellett érvényes az X→Y funkcionális függőség, akkor az R_1 reláció mindig veszteségmentesen (adat- és függőségveszteségmentesen) bontható az $R_2(X,Y)$ és $R_3(X,Z)$ relációkra.

7.2. NORMÁLFORMÁK

A normálformák, ahogy ez már az előzőekben megfogalmazásra került, a relációsémák belső szerkezetének az állapotát, minőségét fejezik ki (minősítik). Több normálforma is ismeretes (attól valamivel kevesebb használatos is), amelyeket általában az xNF jelöléssel illetnek. Az x számérték, az NF pedig a Normal Form angol szavak rövidítéséből ered. A normálformák jelölési sorában csak egyetlen tag különbözik az xNF jelöléstől, íme (a sorban az első a legkedevzőtlenebb, az utolsó pedig a legkifinomultabb normálforma): 0NF, 1NF, 2NF, 3NF, BCNF (Boyce-Codd normálforma), 4NF és 5NF.

A normalizálás a relációséma belső szerkezetének javítását célzó matematikai eljárás, algoritmus, amelynek lépéseit szükségszerűen csak szemantikai elemzés (a valós világgal való ismételt egyeztetés) kíséretében ajánlatos végehajtani. Az optimális belső relációséma-szerkezetet biztosító normalizálás, mint eljárás vagy folyamat a relációk közötti kapcsolatokra is kihatással van (hiszen relációk kívánt minőségi szintre hozása dekompozícióval történik).

A reláció ismétlődő adatot (csoportot vagy adatcsoportot) tartalmaz, ha egy (vagy több) attribútum, akár egyetlen relációelőfordulásra két vagy több értékkel is rendelkezik. Az ilyen attribútumokra mondják, hogy az értékeik nem atomiak a relációban. Ez más megfogalmazásban így hangzik: az ismétlődő adatot tartalmazó relációséma funkcionálisan független attribútumokat tartalmaz.

*A **nem-normalizált, 0NF** alakú relációsémák legalább egy funkcionálisan független, nem atomi értékkel rendelkező attribútumot is magukba foglalnak.*

Példa: HALLGATÓ (Leckekönyvszám, Hallgató_neve, ...Tantárgykód, Tantárgynév, Vizsgadátum, Osztályzat).

A hallgató relációséma relációelőfordulásai a 7.3 ábrán, táblázatban helyezkednek el. A Tantárgykód, a Tantárgynév, a Vizsgadátum és az Osztályzat attribútumok az egyes reláció-

HALLGATÓ

Leckekönyvszám	**Hallgató_neve**	**...**	**Tantárgykód**	**Tantárgynév**	**Vizsgadátum**	**Osztályzat**
E494	Beszédes Ákos		112	Matematika	2000.09.12.	9
			147	Fizika	2000.09.15.	7
			174	Elektrotechnika	2000.10.05.	6
E495	Néma Levente		174	Elektrotechnika	2000.07.14.	8
			112	Matematika	2000.09.15.	8
...	...		...	...	...	...

7.3. ábra.

előfordulásokban (a táblasorokban) több értéket is felvehetnek, vagyis ők nem függnek funkcionálisan a hallgató relációséma kulcsától, a leckekönyvszámtól. A relációséma rossz belső szerkezetét (a Heath-tétel alapján) két relációsémára lehet bontani a Leckekönyvszám→Hallgató_neve, … alapján:

1. HALLGATÓ1(Leckekönyvszám, Hallgató_neve, ...) és
2. SIKERES_VIZSGA(Leckekönyvszám, Tantárgykód, Tantárgynév, Vizsgadátum, Osztályzat).

*Az relációséma belső szerkezete tekintve bizonyosan legalább első **normálformájú (1NF)**, ha minden attribútum funkcionálisan függ az azonosítótól (minden reláció-előfordulásban minden attribútum csak atomi/egyetlen értéket vehet fel/tartalmazhat.*

A Heat-tétel alkalmazásának magyarázata: Az eredeileg megadott HALLGATÓ relációséma belső szerkezetének első normálalakra hozása a kulcstól funkcionálisan független attribútumok relációból való eltávolításából áll. Az eredeti relációséma dekompozícióval két új reláció keletkezik: az elsőben (HALLGATÓ1) az eredeti relációséma kulcstól funkcionálisan függő attribútumai maradnak (és természetesen a kulcs is), a másik relációba (SIKERES_VIZSGA) kerülnek az eredeti reláció kulcstól funkcionálisan független attribútumai az eredeti relációséma kulcsával kiegészítve. A bontás utáni két 1NF szerkezetű relációséma relációelőfordulásai a 7.4. ábrán láthatók.

A második relációséma a tartalma alapján értelemszerűen a SIKERES_VIZSGA elnevezést kapta és összetett (hierarchikus) kulcsa van: Leckekönyvszám+Tantárgykód. Az első relációséma új neve utal a felbontásban új tartalmat kapott relációra, a kulcsa azonban nem változott.

*A relációséma belső szerkezete legalább **második normálalakú (2NF)**, ha első normálformában (1NF) van, és minden leíró (nem-kulcs) attribútum teljes funkcinális függőségben van a kulcstól (funkcionális függőségben van a teljes kulcstól).*

Az előző definíció két záradéka a következő:

1. ha a relációséma kulcsa egyszerű kulcs, akkor az 1NF szerkezet megléte automatikusan és bizonyosan a 2NF szerkezetet is jelenti és
2. a csupakulcs (leíró, nem kulcsszerű attribútumokkal nem rendelkező) relációséma mindig 2NF szerkezetű.

HALLGATÓ1

Leckekönyvszám	**Hallgató_neve**	**...**
E494	Beszédes Ákos	
E495	Néma Levente	
...	...	

SIKERES_VIZSGA

Leckekönyvszám	**Tantárgykód**	**Tantárgynév**	**Vizsgadátum**	**Osztályzat**
E494	112	Matematika	2000.09.12.	9
E494	147	Fizika	2000.09.15.	7
E494	174	Elektrotechnika	2000.10.05.	6
E495	174	Elektrotechnika	2000.07.14.	8
E495	112	Matematika	2000.09.15.	8
...	...	...	...	...

7.4. ábra.

Elnézve a SIKERES_VIZSGA relációsémát és a hátrányos Tantárgykód→Tantárgynév funkcionális függőséget, ismét Heat-tétel alkalmazása merül fel, melynek eredményeként a következő két új relációséma kapható:

1. TANTÁRGY(Tantárgykód, Tantárgynév) és
2. SIKERES_VIZSGA1(Leckekönyvszám, Tantárgykód, Vizsgadátum, Osztályzat).

Szemantikailag A relációsémák második normálalakra hozása a részleges funkcionális függőségek kikeresésével kezdődik, hogy azok a következő lépésben dekompozícióval eltávolíthatók legyenek. A 2NF kialakítása három lépésből áll:

1. A relációséma kulcsából kiválasztásra kerülnek azok az attribútumok (vagy azok csoportja), amelyek maguk is funkcionálisan meghatároznak más attribútumokat,
2. az előző pontban kiválasztott attribútum(attribútmcsoport)ok a tőlük funkcionális függésben lévő attribútumokkal együtt külön relációsémába kerülnek,
3. a kulcshoz teljes funkcionális függéssel kötődő attribútumok a kulccsal együtt kerülnek az új (az eredetihez leginkább hasonló relációban.

A HALLGATÓ1 relációsémának egyszerű akulcsa, ezért a 2NF ellenőrzését (szükség szerint persze a kialakítását is) csak a SIKERES_VIZSGA reláción érdemes végezni. Itt a kulcs része, a Tantárgykód funkcionális függéssel határozza meg a Tantárgynevet, tehát ezek kerülnek az új relációsémába (TANTÁRGY). Az eredeti SIKERES_VIZSGA relációséma az "attribútumvesztés" után a SIKERES_VIZSGA1 nevet fogja viselni. A 2NF formájú relációsémák relációelőfordulásai a 7.5. ábrán találgató táblázatokban láthatók.

*A relációséma belső elrendezését tekintve bizonyosan legalább **harmadik normálalakú (3NF)**, ha 2NF-ben van, és egyetlen nem kulcsszerű (leíró) attribútuma sem függ tranzitíven a reláció kulcsától.*

Az ilyen, harmadik normálformájú belső szerkezettel rendelkező relációsémáról elmondható, hogy minden leíró attribútuma funkcionálisan függ a kulcstól (1NF), teljes funkcionális függéssel függ az (összetett) kulcstól (2NF), viszont a többi leíró attribútumtól funkcionálisan független (3NF).

HALLGATÓ1

Leckekönyvszám	**Hallgató_neve**	**...**
E494	Beszédes Ákos	
E495	Néma Levente	
...	...	

TANTÁRGY

Tantárgykód	**Tantárgynév**
112	Matematika
147	Fizika
174	Elektrotechnika
...	...

SIKERES_VIZSGA1

Leckekönyvszám	**Tantárgykód**	**Vizsgadátum**	**Osztályzat**
E494	112	2000.09.12.	9
E494	147	2000.09.15.	7
E494	174	2000.10.05.	6
E495	174	2000.07.14.	8
E495	112	2000.09.15.	8
...	...	...	...

7.5. ábra.

A harmadik normálalak elérése a relációsémában felfedezett tranzitív függőség meghatározó és meghatározott (bal és job) oldali attribútumainak egy új relációba történő kiválasztásával történik. Így az eredeti relációsémából, a felbontás után keletkező új reláció nem fogja tartalmazni a tranzitív függőség meghatározott (job oldali) attribútumait.

A 3NF alakra hozás példájaként egy kereskedelemben használt modellrész fog szolgálni:

RENDELÉS(Rendelésszám, Vevőkód, Vevőnév, Rendelésdátum).

Rendelésnyilvántartási problémáról van szó. A Heat-tétel alkalmazását itt a 2NF-ben levő RENDELÉS relációra és a Vevőkód→Vevőnév funkcionális függőségre lehet értelmezni (a Vevőnév attribútum tranzitívan függ a reláció kulcsától (Rendelésszám), aminek következtében a következő két relációséma keletkezik:

VEVŐ(Vevőkód, Vevőnév) és

RENDELÉS1(Rendelésszám, Vevőkód, Rendelésdátum).

A második normálformában lévő RENDELÉS relációséma, majd a dekompozíció eredményeképpen létrejött RENDELÉS1 és VEVŐ követően létrejött relációsémák relációelőfordulásai a 7.6. ábrán találgató táblázatokban láthatók.

Az előző normalizációs folyamatok (lépések, amelyek a relációséma belső szerkezetének javítását célozzák meg) dekompozícós eljárással bonyolódtak, az eredeti relációséma (és a folyamat lépéseiben létrejövő újabb relációk) két vagy több relációra bontásával. A normalizálás a relációséma belső szerkezetét javítja, finomítja.

RENDELÉS

Rendelésszám	**Vevőkód**	**Vevőnév**	**Rendelésdátum**
17494	112	Carnex	2000.09.12.
15944	147	Pionir	2000.09.15.
13675	174	Sever	2000.10.05.
19395	174	Sever	2000.07.14.
25495	112	Carnex	2000.09.15.
...	...		...

RENDELÉS1

Rendelésszám	**Vevőkód**	**Rendelésdátum**
17494	112	2000.09.12.
15944	147	2000.09.15.
13675	174	2000.10.05.
19395	174	2000.07.14.
25495	112	2000.09.15.
...	...	...

VEVŐ

Vevőkód	**Vevőnév**
112	Carnex
147	Pionir
174	Sever
...	

7.6. ábra.

A dekompozíciós eljárás az attribútumok kivetítésével (eltávolításával, vagyis a relációk felbontásával) éri el a kívánt eredményt (a belső szerkezet javítását): az 1NF-re hozáskor a kulcstól független attribútumokra, a 2NF esetében az összetett kulcstól funkcionálisan nem teljesen (vagyis részlegesen) függő attribútumokra, illetve a 3NF megvalósításánál a kulcstól tranzitívan függő attribútumokra értendő a kivetítés (eltávolítás új relációsémába). A kivetítés mindig új relációba történik, olyan módon, hogy a meghatározó attribútumok is az új relációsémába kerülnek (a zavart vagy rossz belső szerkezetet okozó meghatározott attribútumokkal együtt). A normalizálási lépések elvégzésekor (befejezésekor) szükségszerű a relációsémák ellenőrzése, esetenként reláció-összevonások is történhetnek. Egy feltételnek azonban mindig teljesülnie kell a dekompozíciós eljárás alkalmazásánál: biztosítani kell a

bontási folyamat visszafordíthatóságát, vagyis, hogy az új relációsémákból mindig visszaállítható legyen az eredeti (kiindulási) reláció. Az ilyen dekompozíciós eljárás neve veszteségmentes (kapcsolat- és adatvesztésmentes) dekompozíció (nonloss vagy lossless decomposition).

A kivetítés vagy szétválasztás ellentétes művelete az összekapcsolás (join). Két relációséma összekapcsolásához szükségszerű az azonos tartalmú (esetleg azonos nevű is lehet, de ez nem feltétel) attribútum(ok) jelenléte mindkét relációban. Az összekapcsolás veszteségmentessége ugyancsak kikötés, amelyet be kell trtani: az új, összekapcsolással létrejött relációknak hordozniuk kell a szülő-egyedek minden információját (nonloss vagy lossless join).

8. AZ SQL TOVÁBBI LEHETŐSÉGEI

A tankönyben ismertetett deklaratív nyelv, az SQL, sokkal több mindenre képes, mint amennyi lehetőség ismertetésre került. Ebben a rövid fejezetben két olyan kiragadott, és eddig bemutatásra még nem került SQL-alkalmazási terület kerül a terítékre, amelyek heterogén eredetűek, így »az SQL további lehetőségei« gyűjtőcím alá kerültek. Első témakörként a triggerek (kioldók) felvázolására kerül majd sor. Egyes vélemények szerint a kioldók a megszorításoknak speciális fajtái, mások szerint viszont az aktív adatbázisok szabályrendszerének (az ECA – Event-Condition-Action szabályrendszernek) az egyik csoportját alkotják. A második érintett terület/témakör az SQL egy bővítési lehetőségét, egy szerveroldali alkalmazását mutatja be. Konkrétan a sémában tárolt eljárásokról lesz szó. Ezeket az eljárásokat az adatbáziskezelő rendszer a relációséma részeként rögzíti (a nézettáblákhoz hasonlóan), és bármikor, bárhonnan (SQL lekérdezésekben, beágyazott SQL-ekben) meghívhatók.

8.1. AZ ECA SZABÁLYRENDSZER ÉS A TRIGGEREK (KIOLDÓK)

Az ECA (Event-Condition-Action) szabályrendszer *esemény-feltétel-művelet* formájú szabályok gyűjteménye, ahol az esemény fogalma alatt a következő egyszerű és összetett eseményeket kell érteni:

1. egyszerű események:
    a. adatkarbantartási műveletek,
    b. adatlekérési/listázási/SELECT műveletek,
    c. időpont/időesemény bekövetkezése,
    d. alkalmazásban/programban megfogalmazott esemény bekövetkezése,
2. összetett események – az egyszerű és előzőleg már megfogalmazott összetett események kombinációja a következő operátorok alkalmazásával:
    a. logikai operátorok (AND, OR, NOT, stb.)
    b. szekvencia (adott sorrendben bekövetkező események lefolyásának esetén)
    c. időeseményt is tartalazó összetett eseménysor (kompozíció), pl. “5 percenként az ébresztési időpont bekövetkezése után”.

Az ECA szabályrendszer alapján működő aktív adatbázisoknál egy konkrét szabályban megfogalmazott esemény bekövetkezésekor kiértékelésre kerül az adott szabályban található

feltétel, és amennyiben a feltétel kiértékelési eredménye “igaz”, úgy végrehajtásra kerül a feltétel után következő művelet.

A triggerek esetében csak egyszerű, egy adatbázistábla adatkarbantartásra vonatkozó esemény alkalmazható (sorbeszúrás, adatmódosítás vagy sortörlés). A triggereket gyakran használják a naplózás (naplóvezetés) megvalósítására: így a kritikusan fontos adatkat tartalmazó táblázatok adatainak mindennemű változása naplózásra kerülhet (ki végezte az adatkarbantartást, mikor történt az esemény, és a karbantartás előtti és utáni állapot leírása – beszúrásnál az új, beszúrt sor, törlésnél a régi törölt sor, módosításnál a régi és az új értékek rögzítése).

A triggerek osztályozása három szempontból is elvégezhető:

1. a hívás (a feltétel ellenőrzésének és a művelet végrehajtásának) ideje alapján a triggerek lehetnek:
    a. BEFORE triggerek és
    b. AFTER triggerek
2. a triggert aktiváló/kiváltó esemény (művelet) alapján:
    a. INSERT trigger,
    b. UPDATE trigger,
    c. DELETE trigger és
3. a trigger által elvégzendő művelet érvényesítése alapján:
    a. utasítás szintű (statement-level) trigger (a karbantartás által érintett összes sorra vonatkozóan csak egyszer hajtódik végre a trigger művelete)
    b. sor szintű (row-level) trigger (minden karbantartást érintő sorra egyszer végrehajtásra kerül a trigger művelete).

A trigger általános alakja:

CREATE TRIGGER *triggernév*
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF *mezőnév*$_1$, *mezőnév*$_2$, ..., *mezőnév*$_n$] }
ON *táblanév*
[REFERENCING {OLD [ROW] [AS] *régisor*]
 | NEW [ROW] [AS] *újsor*
 | OLD TABLE [AS] *tábla_a_régi_sorokkal*
 | NEW TABLE [AS] *tábla_az_új_sorokkal*}
[FOR EACH {ROW | STATEMENT}]
[WHEN *feltétel*]
{*SQL_utasítás*
 | BEGIN ATOMIC
  {*SQL_utasítás*$_1$; *SQL_utasítás*$_2$[; …*SQL_utasítás*$_n$]
  END}

Példa. A következő kioldó (trigger) az önfinanszírozó hallgató táblában tárolt összbefizetésTandíjra mező értékének csökkentését teszi semmissé. Nem logikus ugyanis, hogy a hallgató által tandíjra befizetett összeg csökkenthető legyen (legfeljebb csak növekedhet egy újabb befizetésnek köszönve). A Hallgató-önfinanszírozó relációsémája a következő: **HALLGATÓ-ÖNFINANSZÍROZÓ(<u>indexszám</u>, név, cím, költség, összbefizetésTandíjra)**

```
1)  CREATE TRIGGER összTandíj
2)  AFTER UPDATE OF összbefizetésTandíjra on Hallgató-önfinanszírozó
3)  REFERENCING
4)      OLD ROW AS Régisor,
5)      NEW ROW AS Újsor
6)  FOR EACH ROW
7)  WHEN (Régisor.összbefizetésTandíjra > Újsor.összbefizetésTandíjra)
8)      UPDATE Hallgató-önfinanszírozó
9)      SET összbefizetésTandíjra = Régisor.összbefizetésTandíjra
10)     WHERE indexszám = Újsor.indexszám;
```

A trigger leírása/értelmezése:

- Az 1) sor a trigger-deklaráció, amelyben a trigger neve is megtalálható (összTandíj).
- A triggert kiváltó esemény leírása és az aktiválás ideje a 2) sorban került megfogalmazásra: a Hallgató-önfinanszírozó tábla összbefizetésTandíjra attribútumának módosítása után kerül működési állapotba a trigger.
- A 3), 4) és az 5) sor definiálja a hivatkozási lehetőséget a módosítás előtti értékekre (Régisor) és a módosítás utáni értékekre (Újsor) is. Beszúrás esetén az OLD ROW AS értelmezhetetlen, törlési eseménynél viszont a NEW ROW AS, értelemszerűen.
- A 6) sor a trigger által elvégzendő művelet érvényesítésére vonatkozik: a példában, a 6) sorban sor szintű trigger definíciója/deklarálása (FOR EACH ROW) olvasható (ha a FOR EACH STATEMENT tartalom lenne feltüntetve, akkor az utasítás szintű triggert jelölne, és akkor az OLD ROW AS helyett az OLD TABLE AS, illetve a NEW ROW AS helyett a NEW TABLE AS kifejezésnek kellene állnia).
- A 7) sorban a trigger által kiértékelendő (szokványos WHEN) feltétel helyezkedik el, aminek hiányában a következő sorokban leírt műveletek feltétel nélkül végrehajtódnának. A feltétel akkor ad vissza »igaz« értéket, ha a módosítás következtében/után az összbefizetésTandíjra attibútum értéke kisebb, mint a módosítás előtti érték
- A maradék sorok, a 8), 9) és 10) a végrehajtandó műveletet tartalmazzák: a példában egy WHERE záradékkal rendelkező módosítási művelet helyezkedik el,

amelyik csak az érintett sorra végzi el a módosítás előtti érték visszaállítását. Egynél több végrehajtandó művelet esetén, azokat ponosvesszővel kell elválasztani egymástól, és a BEGIN (ATOMIC)-END kulcsszók közé kell beágyazni.

8.2.A RELÁCIÓSÉMÁBAN TÁROLT ELJÁRÁSOK

Az SQL:2003 jelölésű szabvány, egyéb más dolgok mellett az SQL nyelv PSM (Persistent, Stored Modules – Tartós, Tárolt Modulok) bővítését is tartalmazza. A PSM-ek a relációsémában tárolódnak, és függvényeket-FUNCTION és eljárásokat/procedúrákat-PROCEDURE tartalmazhatnak (akár nagyobb számban is, vegyesen). A függvények és az eljárások nagyon hasonlítanak a programnyelvekben használatosakhoz: a függvények sok (és kizárólag) bemenő paramétere lehet, és egy visszatérési típusa, az eljárás számára pedig tetszőleges számú bemeneti és kimeneti/visszatérő paraméter engedélyezett.

Az eljárásdefiníció szintaktikája:

CREATE PROCEDURE *eljárásnév* (paramfajta$_1$ paramnév$_1$ paramtípus$_1$, ...,
paramfajta$_n$ paramnév$_n$ paramtípus$_n$)
[*lokális_deklarációk*]
BEGIN
utasítások
END;

- a paramfajta$_i$ a paraméter fajtáját fogalmazza meg, és a következő három értéket veheti fel:
  (1) IN – bemenő paramétert jelöl, amelynek értéke nem változhat az eljárástörzsben (a BEGIN és az END között)
  (2) INOUT – bemenő-kimenő paraméterre utal, az eljárás bemenőként fogadja, és visszatérőként kezeli, értéke megváltozhat az eljárástörzsben
  (3) OUT – kimeneti (az eljárás szempontjából)/visszatérő (az eljárást meghívójának szemszögéből) paraméter leírása, amely híváskor érték nélküli, értéket az eljárástörzsben kap.
- *lokális_deklarációk* – az eljárástörzsben használt változókat itt kell deklarálni – ha vannak
- *utasítások* – az eljárástörzs utasításai között BEGIN-END párossal határolt beágyazott utasításblokkok is szerepelhetnek.

A függvénydefiníció szintaktikája kicsit eltér az eljárásétól:

CREATE FUNCTION *eljárásnév* ([paramfajta$_1$] paramnév$_1$ paramtípus$_1$, ...,
[paramfajta$_n$] paramnév$_n$ paramtípus$_n$) RETURNS *típus*
[*lokális_deklarációk*]
BEGIN
utasítások
END;

- a paramfajta$_i$ a függvények esetében csak IN lehet, így a függvénydefinícióban a megadása is tetszőleges,
- a függvény visszatérési típusát kötelezően meg kell adni.

Példa: A következő eljárás a hallgató lakcímváltozását végzi el, a relációséma a következő:

Hallgató (<u>leckek_szám</u>, név, szül_év, ir_szám, helység, utca-hsz, t_körzet, telefon)

```
CREATE PROCEDURE Lakcímváltozás (IN leckek_száma CHAR(8), IN új_ir_szám
                                 INTEGER(5), IN új_helység VARCHAR(30), IN
                                 új_utca-hsz VARCHAR(30))
    UPDATE hallgató
        SET ir_szám=új_ir_szám
        SET helység=új_helység
        SET utca_hsz=új_utca_hsz
      WHERE leckek_szám=leckek_száma;
```

- az eljárás nem igényel saját változókat – a *lokális_deklarációk* rész szükségtelen
- az eljárástörzs egyetlen SQL utasítást tartalmaz, így a BEGIN-END páros is fölösleges.

8.2.1. Utasítások a tárolt eljárásokban

Az alábbiakban a tárolt eljárásokban legyakrabban alkalmazott utasítások kerülnek bemutatásra, röviden, tömören, helyenként példával illusztrálva.

1.) A lokális változó deklarációjának alakja:

DECLARE *lokális_változó_neve típus*;

- Csak az eljárás/függvény futásidejében élő változó

2.) Az értékadó utasítás szintaktikája:

SET *változó = kifejezés*;

- A *kifejezés* (SQL lekérdezés is lehet, ha egy értéket ad vissza) kiértékelése során kapott érték a *változó*-ba kerül,
- Az értékadó utasítás bárhol előfordulhat az eljárástörzsben (a fenti példában a SET utasításrész nem önálló utasítás, hanem a a módosító/UPDATE utasítás szerves része),

- Amennyiben a *kifejezés* hiányzik a *változó* NULL értéket vesz fel.

3.) Utasításcsoportok létrehozása

- A függvény, illetve eljárás törzse csak egyszerű utasítás lehet, ezért ha több utasítást kell alkalmazni a törzsben, akkor azokat BEGIN-END kulcsszavak közé kell helyezni. Az utasításokat az utasításcsoporton belül (a BEGIN és az END között) pontosvesszővel kell elválasztani.

4.) Utasításcimkék használata

- Az utasítások elé cimke helyezhető, amit kettőspont választ el az utasítástól.

5.) Eljáráshívási utasítás (függvényhívásra nem alkalmazható)

CALL *eljárásnév* (*argumentumlista*);

- Az eljáráshívás kiadható egy általános SQL felületen, SQL parancsként:

CALL Lakcímváltozás (12212212, 24413, „Palics", Horgosi út 91/A);

- Az eljáráshívás alkalmazható más függvények vagy eljárások utasításaként
- Hívható eljárás egy befogadó nyelvi programból is, pl. így:

EXEC SQL CALL Lakcímváltozás (12212212, 24413, „Palics", Horgosi út 91/A);

6.) Függvényhívási utasítás

- Sok más nyelvhez hasonlóan a függvényt csak kifejezésekben lehet használni (a nevével lehet hivatkozni rá a kifejezésekben), külön hívási utasítása nincs és nem megengedett!

Példa: Tételezzük fel, hogy van egy Településneve függvénytípusú tárolt eljárás, amely az irányítószám IN paraméterrel rendelkezik, és a településnevet adja vissza visszatérő értékként. A hallgató relációséma pedig **Hallgató (leckek_szám, név, szül_év, ir_szám, helység, utca-hsz, t_körzet, telefon).** A hallgató tábla új hallgatójának beszúrása ilyen alakot is ölthet:

```
INSERT INTO hallgató
        VALUES (11211212, Mekk Elek, 1994, 24413, Településneve(24413),
                Horgosi út 91/A, 024, 024 752 257);
```

7.) Visszatérési utasítás a függvényeknél – szintaktika:

RETURN *kifejezés*;

- A programnyelvekben megszokott visszatérési (RETURN) utasítástól eltérően, a vezérlés itt, a visszatérési utasítás végrehajtásával nem kerül vissza a hívó eljáráshoz,

hanem a függvény végrehajtása folytatódik a további utasítások végrehajtásával (ha léteznek).

8.) Elágazásutasítások – az IF utasítás általános alakja:

IF *feltétel* THEN
    utasításlista
ELSEIF *feltétel* THEN
    utasításlista
ELSEIF
    . . .
ELSE
    utasításlista
END IF;

- az utasításlista pontosvesszővel ellátott utasításait nem szükséges BEGIN-END kulcsszavak közé kell helyezni.

9.) Ciklusutasítások

A ciklusutasítások közül minden ismert, más programozási nyelvben is használatos utasítás alkalmazható a PSM-ben: LOOP, FOR, WHILE és a REPEAT. A ciklusdefiníciós (cikluskezdő) és cikluszáró utasítások is rendelkezhetnek cimkével, megkönnyítve ezzel az áttekintketőséget (az ember számára) és az értelmezhetőséget (és szintaktikai hibák kiszűrését a PSM értelmező számára). A cimke a cikluskezdő utasítás elé, illetve a cikluszáró utasítás mögé írandó.

a) A LOOP utasítás formája:

loopcimke:LOOP
        utasításlista
    END LOOP;

- a LOOP ciklusnak nincs saját tesztelési lehetősége a végrehajtás számának korlátozására, ezért az utasításlistában alkalmazni kell a következő cikluselhagyó utasítást:

LEAVE *loopcimke*;

b) A FOR utasítás alakja:

FOR *ciklusnév* AS *sormutatónév* CURSOR FOR *lekérdezés*
DO
    utasításlista
END FOR;

- a FOR utasítás csak sormutató-értéken való lépegetésre használható, a *lekérdezés* eredménytáblájában való léptetésre alkalmas, és az aktuális sorra értelmezve hajtódnak végre az *utasításlista* utasításai,
- gondoskodik a sormutató inicializásáról és lezárásáról, ellenőrzi, hogy van-e következő sor és bekéri ha van, ha nincs, akkor pedig kilép a ciklusból (nincs szükség kilépési feltételere),
- nincs szükség változó/változók deklarálására, amibe az aktuális sor elemeit elhelyezi a rendszer, hanem az eredeti (*lekérdezés*ben, SELECT-ben használt) attribútumneveket lehet használni az *utasításlistá*ban.

c) A WHILE utasítás szintaktikája:

WHILE *feltétel* DO
utasításlista
END WHILE;

- a WHILE elöl tesztelő ciklus, így ha a feltétel a ciklus végrehajtásának pillanatában nem teljesül, az utasításlista egyszer sem kerül végrehajtásra.

d) A REPEAT utasítás alakja:

REPEAT
utasításlista
UNTIL feltétel
END REPEAT;

- a REPEAT hátul tesztelő ciklus, így ha a feltétel a ciklus végrehajtásának pillanatában nem is teljesül, az utasításlista egyszer mindenképpen végrehajtásra kerül.

10.) Kivételek kezelése/hibakezelés

Az SQL utasítások végrehajtásának sikerességéről/sikertelenségéről az adatbáziskezelő rendeszer az SQLSTATE változó értékének beállításával értesíti a felhasználót/alkalmazást. Az SQLSTATE változó öt karakter hosszúságú, és az értéke minden SQL utasítás végrehajtása után lekérdezhető. Erre természetesen minden utasítás végrehajtása után nem biztos, hogy szükség mutatkozik, de a lehetőség adott. A kivételek kezelése alapvetően kétféleképpen történhet:

a) gyalogos módszerrel, amikor csak a feltétel kerül meghatározásra, a bekövetkezéséről és feldolgozásáról az eljárástörzsben kell helyet és figyelmet fordítani. Ekkor elég csupán a feltételnévre hivatkozni, és nem szükséges az SQLSTATE változó értékét vizsgálni. A feltételdeklaráció általános alakja a következő:

DECLARE *feltételnév* CONDITION FOR SQLSTATE *érték*;

b) kivételkezelő kódrészlet definiálásával, amikor az adatbáziskezelő rendszer figyeli a feltétellistában megfogalmazott feltételeket, és amennyiben szükség van rá, automatikusan végrehajtja a hibakezelő kódrészletet. A kivételkezelő kódrészlet deklarációja a következő szintaktikával rendelkezik:

DECLARE *hol_folytassa* HANDLER FOR *feltétellista utasítás*;

- a *hol_folytassa* a következő három értéket veheti fel:
  - CONTINUE – a hibát kiváltó utasítás utáni utasítás kerül végrehajtásra, miután a deklarációban leírt *utasítás* lefutott,
  - EXIT – a deklarációban leírt *utasítás* lefutása után a kivételkezelő deklarációt tartalmazó BEGIN... END blokkot követő utasítás kerül végrehajtásra és
  - UNDO – megegyezik az EXIT változattal, azzal a különbségel, hogy a BEGIN... END blokkban történt változásokat „visszagörgeti", visszavonja.
- a *feltétellista* vesszővel elválasztott feltételek sorát jelenti, ahol akár szimbolikus, akár „SQLSTATE 'xxxxx' " alakú feltételek is előfordulhatnak,
- az *utasítás* a feltételek bekövetkezésekor végrehajtandó utasítás, ha több utasítás végrehajtására van szükség, akkor azokat BEGIN... END blokkba kell elhelyezni.

11.) Lekérdezések a tárolt eljárásokban

Tárolt eljárásokban több helyen és módon is megfogalmazhatók lekérdezések:

a) feltételekben,
b) egyeteln értéket visszaadó lekérdezések az értékadó utasítások jobb oldalán is alkalmazhatók,
c) egysoros eredménytáblát létrehozó lekérdezések megfogalmazhatók az INTO záradék (közvetlenül a SELECT záradék után és a FROM záradék előtt) segítségével, ahol a lekérdezés eredménye (a sor) egy előzőleg deklarált változóba (változókba) tölthető,
d) táblázatot visszaadó eredménytábla esetén a feldolgozást lehetővé tevő sormutatót kell deklarálni, ami csak szükséges, de nem elégséges feltétel a feldolgozáshoz. A feldolgozás a sormutató megnyitásával (OPEN) kezdődik el, amikor a nyitás pillanatában a sormutató az eredménytábla első sorára mutat. Az ott elhelyezkedő (sor)adatok a FETCH utasítással válnak elérhetővé. Minden következő FETCH utasítás a következő sor adatait teszi a megfelelő előzőleg deklarált változó(k)ba. Az utolsó eredménytábla-sor átvétele/feldolgozása után kiadott FETCH utasításnak már nincs mit átvenni az eredménytáblából, és ezt az SQLSTATE változó '02000' értékre állításával jelzi az

adatbáziskezelő rendszer. Az eredménytábla lezárása a nyitás után bármikor megtörténhet a CLOSE utasítással, ami után az elveszik (újabb OPEN utasítás nem adható ki a már egyszer bezárt sormutatóra). A fent említett utasítások előírt alakjai a következők:

DECLARE *sormutatónév* CURSOR FOR *lekérdezés*;

OPEN *sormutatónév*;

FETCH FROM *sormutatónév* INTO *változólista*;

CLOSE *sormutatónév*;

Példa:

```
1)  CREATE PROCEDURE union (IN táblanév1 VARCHAR(100), IN oszlopnév1 VARCHAR(100), IN
    táblanév2 VARCHAR(100), IN oszlopnév2 VARCHAR(100), OUT 1union2 VARCHAR(100))
2)  BEGIN

3)     DECLARE t1 VARCHAR(100);
4)     DECLARE t2 VARCHAR(100);
5)     DECLARE vége CONDITION FOR SQLSTATE ’21000’;
6)     DECLARE v1 DEFAULT 0;
7)     DECLARE v2 DEFAULT 0;
8)     DECLARE t1t CURSOR FOR SELECT oszlopnév1 FROM táblanév1 ORDER BY 1;
9)     DECLARE t2t CURSOR FOR SELECT oszlopnév2 FROM táblanév2 ORDER BY 1;

10)    OPEN t1t;
11)    OPEN t2t;

12)    FETCH t1t INTO t1;
13)    IF vége THEN
14)       SET v1 = 1;
15)    END IF
16)    FETCH t2t INTO t2;
17)    IF vége THEN
18)       SET v2 = 1;
19)    END IF

20)    H1: LOOP
21)       IF v1 AND v2 THEN
22)          LEAVE H1;
23)       END IF

24)       IF t1=t2
25)          INSERT INTO 1union2 VALUES(t1);
26)          FETCH t1t INTO t1;
27)          IF vége THEN
28)             BEGIN
29)                SET v1 = 1;
30)                H2: LOOP
31)                   FETCH t2t INTO t2;
32)                   IF vége THEN
33)                      BEGIN
34)                         SET v2 = 1;
35)                         LEAVE H2;
36)                      END
37)                   END IF
38)                   INSERT INTO 1union2 VALUES(t2);
39)                END LOOP
```

```
40)                IF v1 AND v2 THEN
41)                   LEAVE H1;
42)                END IF
43)             END
44)          END IF
45)          FETCH t2t INTO t2;
46)          IF vége THEN
47)             SET v2 = 1;
48)          END IF;
49)          IF v1 AND v2 THEN
50)             LEAVE H1;
51)          END IF
52)       ELSEIF t1<t2
53)          INSERT INTO 1union2 VALUES(t1);
54)          FETCH t1t INTO t1;
55)          IF vége THEN
56)             BEGIN
57)                SET v1 = 1;
58)                H3: LOOP
59)                   FETCH t2t INTO t2;
60)                   IF vége THEN
61)                      BEGIN
62)                         SET v2 = 1;
63)                         LEAVE H3;
64)                      END
65)                   END IF;
66)                   INSERT INTO 1union2 VALUES(t2);
67)                END LOOP
68)                IF v1 AND v2 THEN
69)                   LEAVE H1;
70)                END IF;
71)             END
72)          END IF
73)       ELSEIF t1>t2
74)          INSERT INTO 1union2 VALUES(t2);
75)          FETCH t2t INTO t2;
76)          IF vége THEN
77)             BEGIN
78)                SET v2 = 1;
79)                H4: LOOP
80)                   FETCH t1t INTO t1;
81)                   IF vége THEN
82)                      BEGIN
83)                         SET v1 = 1;
84)                         LEAVE H4;
85)                      END
86)                   END IF
87)                   INSERT INTO 1union2 VALUES(t1);
88)                END LOOP
89)                IF v1 AND v2 THEN
90)                   LEAVE H1;
91)                END IF
92)             END
93)          END IF;
94)       END IF;
95)    END LOOP;

96)    CLOSE t1t;
97)    CLOSE t2t;

98) END
```

A példában leírt eljárás a nevéhez híven az unió műveletet végzi el a két bemeneti tábla, ugyancsak bemeneti adatként feltüntetett oszlopértékein. Kimenetként az ötödik paraméterként megadott 1union2 egyoszlopos táblát biztosítja, amely az oszlopértékek unióját tartalmazza. Az első tábla kijelölt oszlopértékeit a t1t sormutatónév „kezeli" a második tábla kijelölt oszlopelemeit pedig a t2t (8) és 9) sor). A lekérdezés eredménytábláiból a FETCH utasítások segítségével az értékek a t1 és t2 (3) és 4) sor) változókba kerülnek. Ha a FETCH nem tud már kivenni sort az eredménytáblából, akkor a *vége* (5) sor) feltétel igaz értéket kap, ilyenkor vagy a v1 vagy a v2 változó (6) és 7) sor) értéke állítódik 1-re (attól függően, hogy melyik táblából fogytak ki a sorok: v1=1 – az első tábla értékei/sorai fogytak el, illetve v2=1 – a második tábla értékei/sorai fogytak el).

A 8) és 9) sorban deklarált lekérdezések a 10) és 11) sorban hajtódnak végre, majd a t1 és t2 változók töltése történik egy-egy FETCH utasítással. Mindegyik FETCH után feltétel-ellenőrzés történik: igaz-e, hogy vége? Vagyis tudott-e a FETCH betölteni értéket a változóba? Ha nem, akkor vagy a v1 vagy a v2 kap 1-es értéket. Ha már az elején mindkét FETCH eredménytelen, akkor a H1 ciklus végre sem hajtódik, hiszen üres mindkét lekérdezés eredménytáblája (22), 23) és 23) sorok). Az értékek összehasonlításánál három eset lehetséges:

1) t1=t2 (a 24) sortól az 52) sorig elhelyezkedő eljárásrész dolgozza fel). Ekkor az egyik (tetszőlegesen kiválasztott) érték kerül az unióba, majd mindkét eredménytáblában újabb sorra kell lépni. Ha az első táblából fogyott ki az adat (27) sor) akkor az megjegyzésre kerül (29) sor) és a H2 ciklusban a második tábla adatai kerülnek az 1union2 kimenő táblába, egészen addig, amíg belőle is elfogynak a sorok (32) sor, ami feljegyzésre kerül (34) sor), majd a ciklus elhagyása következik (35) sor). A H2 ciklus elhagyása után vizsgálatra kerül, hogy van-e még egyáltalá sor, és ha nincs, az eljárás befejeződik (40), 41) és 42) sor). Happy day esetben (az esetek többségében) mindkét eredménytáblából új adat kerül a t1 és t2 változóba.
2) t1<t2 esetében (az 52) sortól a 73) sorig terjedő eljárásrész dolgozza fel) a t1 kerül az unióba (1union2) az 53) sorban, majd az első tábla eredménytábla-értéke kerül átvitelre a t1-be, aztán az algoritmus ugyanúgy folytatódik, mint a t1=t2 esetében. Ha az első táblából fogyott ki az adat (55) sor) akkor az megjegyzésre kerül (57) sor) és a H3 ciklusban a második tábla adatai kerülnek az 1union2 kimenő táblába, egészen addig, amíg belőle is elfogynak a sorok (60) sor, ami feljegyzésre kerül (62) sor), majd a ciklus elhagyása következik (63) sor). A H3 ciklus elhagyása után vizsgálatra kerül, hogy van-e még egyáltalá sor, és ha nincs, az eljárás befejeződik (68), 69) és 70) sor). Happy day esetben az első tábla eredménytáblájából új adat kerül a t1-be.

3) t1>t2 esetében (az 73) sortól a 94) sorig terjedő eljárásrész dolgozza fel) a t2 kerül az unióba (1union2) az 74) sorban, majd a második tábla eredménytábla-értéke kerül átvitelre a t2-be (75) sor). Ha a második táblából fogyott ki az adat (76) sor) akkor az megjegyzésre kerül (78) sor) és a H4 ciklusban az első tábla adatai kerülnek az 1union2 kimenő táblába, egészen addig, amíg belőle is elfogynak a sorok (81) sor, ami feljegyzésre kerül (83) sor), majd a H4 ciklus elhagyása következik (84) sor). A H4 ciklus elhagyása után vizsgálatra kerül, hogy van-e még egyáltalá sor, és ha nincs, az eljárás befejeződik (89), 90) és 91) sor). Happy day esetben a második tábla eredménytáblájából új adat kerül a t1-be.

A végén a kurzorok (lekérdezések) lezárása következik (96) és 97) sor).

10.FELHASZNÁLT IRODALOM

1. Ambler S.W., Sadalage P.J.: Refactoring Databases: Evolutionary Database Design, Addison –Wesley, 2006, (Refactoring Adatbázisok újratervezése, Kiskapu Kft., Budapest, 2009.)
2. Bódy B: Az SQL példákon keresztül, Jedlik Oktatási Stúdió, Budapest, 2003.
3. Buza A.: Az adatbáziskezelés alapjai, Dunaújvárosi Főiskola, 2012,
4. Coronel C., Morris S., Rob P.: Database Systems – Design, Implementation and Management, Cengage Learning, Boston, USA, 2012.
5. Date C.J.: "An Introduction to Database Systems", Vol. 1, Third Edition, Addison-Wesley Publ. Co., Reading, MA, 1981.
6. Đorđević B., Vodoprivredni sistemi
7. Eric J. Naiburg, Robert A. Maksimchuk: UML za projektovanje baza podataka, Addison-Wesley (2001)-CET(2002), Beograd.
8. Fehily C.: SQL, Third Edition, Peachpit Press, Berkeley, USA, 2008.
9. Fóti M., Turóczy A.:Adatkezelés otthon és a felhőben, Jedlik Oktatási Stúdió, Budapest, 2012.
10. Gajdos S.: Adatbázisok, Műegyetemi Kiadó, Budapest, 2006.
11. Garcia-Molina H., Ullman J.D., Widom J.: Database System Implementation, Prentice Hall Inc., 2000. (Adatbázisrendszerek megvalósítása, PANEM, Budapest, 2001.).
12. Garcia-Molina H., Ullman J.D., Widom J.: Database Systems: The Complete Book, Prentice Hall Inc., 2002.
13. Halassy B.: Az adatbázis-tervezés alapjai és titkai, IDG, Budapest, 1994.
14. Halassy B.: Az információs rendszerek alapfogalmai, SZÁMALK, Budapest, 1982.
15. Halassy B.: Ember – Információ – Rendszer, IDG, Budapest, 1996.
16. Kiss J.: Adatbázis-kezelés, Széchenyi István Egyetem, Győr, 2007.
17. Kuzmanov S., Mogin P., Petković I., Popović M.: "Automatizacija projektovanja šeme baze podataka", JAHORINA '88.
18. Lazarević B., Jovanović V., Vučković M..:"Projektovanje informacionih sistema" I deo, Naučna knjiga, Beograd, 1986.
19. Lazarević B., Marjanović Z., Aničić N, Babarogić S.: Baze pdataka, četvrto izdanje, FON, Beograd, 2008.
20. Lazarević B.: "Prošireni model objekti-veze" interni materijal, Laboratorija za informacione sisteme FON-a, Beograd, april 1990.

21. Lazarević B.: Baze podataka, Radni materijal za interne kurseve, Beograd, 1989.
22. Lerner A.J.: Principi kibernetike, Tehnička knjiga, Beograd, 1970.
23. Marjanović Z.: ORACLE – relacioni sistem za upravljanje bazom podataka, Breza, Beograd, 1990.
24. Masar I.: Uvod u SQL, Zagreb, 2002.
25. Mihajlović D.: Informacioni sistemi i projektovanje baza podataka, FTN, Novi Sad, 1998.
26. Mogin P., Luković I., Govedarica M.: Principi projektovanja baza podataka, FTN-STYLOS, Novi Sad, 2000.
27. Mogin P., Luković I.: Principi baza podataka, FTN-STYLOS, Novi Sad, 1996.
28. Noszkay E.: Informatikai és rendszerszervezési alapismeretek, GDF, Budapest, 1994.
29. Quittner P., Baksa-Haskó G.: Adatbázisok, adatbázis-kezelő rendszerek, DE AMTC AVK, 2007.
30. Radulović B., Kazi Z., Subić Z.: Baze podataka kroz primere i zadatke,Tehnički fakultet „Mihajlo Pupin“ Zrenjanin, 2007.
31. Raffai M.: Információrendszer-fejlesztés, Novodat Kiadó, Győr, 1999.
32. Silberschatz A., Korth H. F., Sudarshan S.: Database System Concepts, Fifth Edition, Mcgraw-Hill Companies, Inc., Boston, 2006.
33. Silberschatz A., Korth H. F., Sudarshan S.: Instrucror's Manual to Accompany Database System Concepts, Mcgraw-Hill Companies, Inc., Boston, 1997.
34. Szelezsán J.: Adatbázisok, LSI, Budapest, 1996.
35. Šuc L.: Projektovanje baza podataka, Izobrazevalni center DELTA, Ljubljana, 1985.
36. Todman C.: Projektovanje skladišta podataka (Designing a Data Warehouse), Prentice Hall PTR (2001)- CET (2001), Beograd.
37. Ullman J.D., Widom J.: A First Course in Database Systems, Third Edition, Pearson Prentice Hall Inc., 2008. (Adatbázisrendszerek – Alapvetés, második, átdolgozott kiadás, PANEM, Budapest, 2009.).
38. Ullman J.D.: "Principles of Database Systems", Computer Science Press, 1980.
39. Vardi M. Y.: Fundamentals of Dependency Theory,
40. Veljović A.: Modeliranje podataka – Erwin (materijal za kurs), CIT, Beograd, 1998.