Hiányzó értékek problémái a relációs adatbázisokban

Dr. Keszthelyi András László

Egyetemi docens, Óbudai Egyetem, Keleti Károly Gazdasági Kar
Keszthelyi.Andras@kgk.uni-obuda.hu

Az adatbáziskezelés relációs elvű megközelítése több, mint ötven esztendős. Edgar Frank Codd a relációs algebrára alapozva gyakorlatilag tökéletes munkát végzett, nem véletlen, hogy a relációs elvű adatbáziskezelés máig egyeduralkodó. Ennek ellenére a hiányzó értékek problémaköre mindmáig kissé problémás. Az SQL szabványban van egy (vagy inkább: a) NULL jel a hiányzó adatelemek jelzésére. Konkrét érték azonban két, minőségileg különböző okból vagy esetben hiányozhat: a) az adatelem létezik, csak értékét – még – nem ismerjük, b) az adott esetben az adatelem nem létezik, értelmetlen. A klasszikus iskolapélda szerint: Ki hányszor szült? Évát még nem kérdeztük meg, nem tudjuk, hogy hányszor szült, az érték létezik, csak mi nem ismerjük. Papírnyomtatványon üresen hagyjuk a rovatot, ez az inszignifikáns nulla esete. Ádámot meg sem kérdezzük, esetében a kérdés nem értelmezhető (n.a. – not applicable), az érték nem is létezik. A probléma az, hogy a két minőségileg különböző esetre csak egy darab NULL jelünk van. A jelen tanulmányban ezen probléma lehetséges megoldásait vizsgálom.

Kulcsszavak: relációs adatbázisok, hiányzó értékek, NULL

Történeti előzmények

1.1 Hardver

Az elektronikus számítógépek első generációja elektroncsövekből épült föl, a fő feladat a puszta működőképesség megteremtése volt. Jellemző sebessége: 300 szorzás másodpercenként. Ez az időszak – lazán fogalmazva – a második világháború környékére tehető.

A második generáció (kb. 1955-1965) már mai szemmel nézve is számítógépszerű: méretei emberi léptékűek, fogyasztása jelentősen lecsökken, megbízhatósága megnövekszik. Tranzisztorokból épül fel. Sebessége 200.000 szorzás másodpercenként. Háttértárak: lyukkártya és mágnesszalag.

A harmadik generáció (kb. 1965-1974) már integrált áramköröket tartalmaz. Sebesség: 2 millió szorzás másodpercenként. Háttértár: ferritgyűrűs tár.

A negyedik generáció (kb. 1971-től) nagy bonyolultságú integrált áramkörökből épül fel, jellemző sebessége 20 millió szorzás másodpercenként (és fölötte). Háttértár: mágneslemez.

Az adatbáziskezelés szempontjából kritikus fontosságú elem a háttértár, viszonylag gyors blokkos szervezésű lemezegység nélkül nincs adatbáziskezelés. 1973-ban jelent meg az IBM 3340 típusjelű lemezegysége (35, illetve 70 MB, 25 msec átlagos elérési idő), a „winchester" (Marci K. Sun, Devon Prince, 1980) és adott jelentős lökést a (relációs) adatbáziskezelésnek.

1.2 Szoftver, operációs rendszerek

A '60-as évek közepétől kerül előtérbe a szoftverek fejlődése, mivel a számítógépek ekkorra már eljutottak az üzembiztos, sorozatban gyártható szintre. Megvolt a működő „vas", lehetett, mi több, kellett foglalkozni a szoftverrel, hogy legyen, ami működteti.

1964-ben jelent meg az IBM OS/360 nagygépes operációs rendszere (a System/360-hoz). Az évtized végére születik meg a UNIX operációs rendszer, egy máig tartó másik sikertörténet. A '70-es évek közepére már megjelennek az olyan lemezegységek, amelyek mind tárolókapacitásuk, mind sebességük alapján már általános célú érdemi munkára is alkalmasak.

1.3 Relációs kezelés

A relációs adatbáziskezelés kezdete – mint oly sok találmányé – nehezen köthető egy adott naphoz. Ha mindenképpen meg akarjuk nevezni a kezdetét, 1969-re tehetnénk. Ekkor jelent meg a témában az első tanulmány, Edgar Frank Codd, közismertebb nevén Ted Codd tollából „Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks" címen (Codd E. F, 1969). Ebben a cikkében a matematikai relációk adatbáziskezelésben való alkalmazhatóságát tárgyalja egy IBM kutatási beszámoló keretében. Codd 1970-es tanulmányát szokták még emlegetni mint első publikációt.

Ebben a tanulmányban Codd hivatkozik D. L. Childs 1968-ban megjelent, „Description of a Set-Theoretic Data Structure" c. tanulmányára (Childs D. L., 1968) akit tehát ötletadóként tarthatnánk számon.

Érdekes technikatörténeti kérdés lenne, hogy a relációs adatbáziskezelés kezdetét végülis mikortól számíthatjuk: az első olyan publikációtól, amely kifejezetten és részletesen erről szól (a lehetőség vagy a konkrétumok szintjén), vagy az első olyan kezelő megjelenésétől, amely az elmélet támasztotta követelményeknek

legalább nagyjából megfelel. Mindenesetre Codd a kezdet kezdetén túlmegy a relációs adatbáziskezelés technikai lehetőségének fölvetésén, már ekkor hangsúlyozza pl. a szerkezeti épség megőrzésének fontosságát, pedig még nem is létezik relációs elvű adatbázis, sőt relációs adatbáziskezelő sem.

Codd ekkor 46 éves. Gyakorlatilag tökéletes alapozást csinál, a relációs adatbáziskezelést érdemi előzmények nélkül, szinte a semmiből teremti meg. Munkája értékét csak fokozza, hogy ekkor még az esetleges megvalósításhoz a technikai feltételek nem állnak rendelkezésre, nincs módja a gyakorlatban is kipróbálni elgondolását, kísérletezni.

Viszonyításképpen: ekkoriban (1969-70) még használható operációs rendszer is alig van. Ekkor még, illetve már dolgoznak a Bell Laborban a UNIX operációs rendszer ősváltozatán Ken Thompson és Dennis Ritchie. Az első számítógépes hálózat 1969-70 fordulóján 4 (négy) géppel indul. 1973-ban jelenik meg az IBM 3340 lemezegység (erről kapta máig tartóan a „winchester" nevet a merevlemez) (Marci K. Sun, Devon Prince, 1980) és kapott így a számítástechnika világa egy nagy lökést a további fejlődéshez. A (relációs elvű) adatbáziskezelésnek ugyanis nélkülözhetetlen technikai kelléke a blokkos tárolási elvű – azaz blokkonként címezhető – háttértár (mágneslemez). Szalagos (szekvenciális) háttértárral gyakorlatban működő relációs elvű adatbáziskezelést csinálni ugyanis nem lehet. Szerintem másmilyen elvűt sem.

Az IBM eleinte nem is alkalmazza Codd kutatási eredményeit. Tízéves küzdelmébe kerül, hogy elfogadtassa a szakmai közvéleménnyel. Könyvének ajánlása: „A Királyi Légierő II. világháborúbeli pilótáinak és legénységének és az oxfordi tanároknak. Tőlük ered elhatározásom, hogy harcoljak azért, amit igaznak hittem, azalatt a tíz-tizenvalahány esztendő alatt, amíg a kormány, az ipar és a kereskedelem erőteljesen ellenezte az adatbáziskezelés relációs elvű megközelítését." (Codd E. F, 1969)

Az általa lefektetett, sőt részletesen kidolgozott alapelvek máig érvényesek, egészen a lekérdezések kezelő általi optimalizálásával bezárólag. Természetes, hogy finomítások történtek az eltelt évtizedek során, sőt bővítési kísérletek is. Maga a lekérdező nyelv a hajdani SEQUEL-től az SQL mai állapotáig is komoly fejlődésen és szabványosítási lépéseken ment keresztül.

A más szemléletű módok nem igazán terjedtek el, mondhatjuk, hogy máig egyeduralkodó a relációs elvű adatbáziskezelés. A megjelent publikációk alapján azonban a relációs elvű megközelítés az első, amely esetben az elmélet kidolgozása – teljes értékű módon – megelőzte a megvalósítási kísérleteket. Ebben az értelemben a relációs modell megelőzi a hálós és hierarchikus modellt.

Ez nem is csoda. Az első adatbáziskezelők ugyanis a gyakorlatias előzmények talaján bukkantak föl, a fájlkezelőkből alakultak ki. A fájlkezelés alapú rendszerek azonban nem felelnek meg az adatbáziskezeléssel szemben támasztott követelményeknek.

1.4 Matematikai alapok

Codd halmazelméleti, relációs algebrai alapokra helyezte az adatbáziskezelés elméletét egy történelmi pillanatban. Nagy előnye, hogy aránylag egyszerű és problémamentes, „letisztult" matematikai alapozást jelent.

A kulcsfogalom az értéktartomány. Értéktartománynak egy tulajdonságtípus általánosan felvehető értékeinek a halmazát nevezzük, függetlenül attól, hogy az adatbázisban pillanatnyilag található-e példa (előfordulás) minden lehetséges értékre, avagy sem.

A reláció formális matematikai meghatározását véve: az S1, S2, ..., Sn – nem feltétlenül különböző – halmazokon (értéktartományokon) R egy reláció, ha ez egy halmaza azon rendezett n-eseknek, amelyek első eleme az S1, második eleme az S2, n-edik eleme az Sn halmaz eleme. Másképpen mondva az S1, S2, ..., Sn fölötti R reláció egy részhalmaza az S1xS2x...xSn Descartes-szorzatnak.

Úgy is mondhatjuk, hogy a reláció nem más, mint egy kétdimenziós tábla (táblázat), melynek csak elemi értékei lehetnek.

Az értéktartomány (domain) alapvető jelentőségű – lenne – a relációs adatbáziskezelésben. Ismét Coddot idézem, amikoris arról ír, hogy az értéktartományok jelentősége még az elsődleges és az idegen kulcsokénál is nagyobb: „A relációs modell számos tulajdonságának teljes támogatása az értéktartományok fogalmának teljeskörű támogatásától függ. Az értéktartományok teljes támogatásának néhány előnye következik. (...) az értéktartományok a ragasztó, amely egyben tartja az adatbázist. Jegyezzék meg: értéktartományról beszélek, nem elsődleges és idegen kulcsokról. A relációs modellben a kulcsok fogalma fontos további és szakosított ragasztót jelent." (Codd E. F, 1969)

A táblázat oszlopainak nevei felelnek meg a modellezett jelenség tulajdonságainak, sorai pedig a nyilvántartásban szereplő tényleges előfordulásoknak.

2 Hiányzó értékek

Azért az SQL, illetve a relációs adatbáziskezelés még ma sem tökéletes. Lássunk ízelítőül egy problémát, a NULL jelek problémakörét.

2.1 Iskolapélda

Egy adatbázisból kétféle módon hiányozhat ismeret. Vagy egész sor hiányzik egy táblából, azaz egy (vagy több) egyedelőfordulás nem szerepel, holott annak

szerepelnie kellene. Ez az eset kívül esik a relációs elvű kezelés határain, modellezési-tervezési vagy üzemeltetési hiba eredménye lehet. („Az egyetlen mód, hogy ne hiányozzon a névsorból: ő olvassa.") (Rejtő, J., 1964)

A másik eset, amikor a sor a „helyén van", csak egyes adatelemei hiányoznak, nem kap értéket minden tulajdonsága, másképpen mondva vannak üres cellák a sorban. Ez az eset különféle problémákat vet föl, amelyeket valamilyen módon kezelni kell.

Az adatmodell fogalmának beletartozik, hogy „megnevezzük a fontos tulajdonságokat (...) leírjuk (...) tartalmukat". Ebből az következne, hogy minden tulajdonságnak minden esetben van értéke, holott ez nem biztos, hogy így van. Ha pontosabban végiggondoljuk a fenti fogalommeghatározást, akkor világossá válik, hogy nem erről van szó benne. A „leírjuk (...) tartalmukat" kitétel nem szükségképpen jelenti azt, hogy minden egyes esetben ismerjük az adott tulajdonság értékét, sőt még azt sem, hogy annak minden esetben lenne egyáltalán értéke.

Egy példával megvilágítva: egy háziorvosi nyilvántartásban elképzelhető olyan rovat, hogy a páciens (adott pillanatig, összesen) hányszor szült. Ez nyilvánvalóan egy nemnegatív egész számmal leírható ismeret. Értéke különböző esetekben lehet: a) pozitív szám, amikor tudjuk, hogy az adott hölgy hányszor is szült (pl. négyszer); b) nulla, amikor tudjuk, hogy az adott hölgy (még) nem szült egyszer sem (a szignifikáns nulla példája). Eddig problémamentes. Probléma azokkal a hölgyekkel van, akikről nem tudjuk, hogy hányszor szültek. Esetükben a megfelelő rovat üres marad (az inszignifikáns nulla esete), míg férfiak esetében szintén üres marad, de a két „üres" állapot nem egyenértékű. Férfiak esetében ugyanis a „szülések száma" nem értelmezhető („n.a." – not applicable, és nem pedig „nincs adat"). E két utóbbi esetet azonban mindenképpen szükséges megkülönböztetni egymástól, a valósághű modellezés követelményén túl csak egyetlen példával érzékeltetve: ha születésszám-statisztikát kell készíteni, nyilván nem számolhatjuk bele nemcsak a férfiakat, de azon hölgyeket sem, akiknél az adat értéke – egyelőre – ismeretlen.

Erre a problémára született az a megoldási kísérlet, hogy az adatbáziskezelők egy különleges, NULL elemmel jelzik azt, ha az adott érték nem létezik. Ez mutatja azt, hogy az adott helyzetben az adat értéke ismeretlen, de nem különbözteti meg az „ismeretlen" esetét a „nem értelmezhető" esetétől, azaz két elvileg különböző esetet egybemos.

2.2 Típusfüggő jelenség

A hiányzó értékek problémája adattípusfüggő. Numerikus adatok esetében ugyanis a hiányzó érték és a „nem értelmezhető" nem különböztethető meg, ugyanis nincs üres numerikus adat az üres szövegessel ellentétben. (Bárki

kipróbálhatja: egy táblázatkezelő cellájába beírandó tetszőleges képlet hivatkozzon egy üresen hagyott cellára. A képlet nulla értékkel fog számolni az üres cella helyén.) Szöveges típusú adat esetében ugyanis van „üres" adat, az ún. üres string, amelyet olyan szövegkonstansként lehet megadni, amely semmit nem tartalmaz (insert into <táblanév> set <mezőnév>="";), azaz valóban az „ürességet" képviseli. Numerikus esetben azonban a nulla (0) az mindig szignifikáns. Így szöveges típusú adatelem esetében meglenne az „üres" („pillanatnyilag nem ismerem az adatot") jelentéstartalom is és a „nem értelmezhető" (NULL) jelentéstartalom is, viszont ugyanez numerikus adatelemek esetében így nem áll fenn, tehát a helyzetet súlyosbítjuk avval, hogy a jelenség függ az adatelemek típusától.

Ezen problémakör a kezdet kezdete óta fennáll, és máig nincs rá megnyugtató és egyértelműen világos megoldás. Ezt jelzi már az is, hogy Ullman és Widom azt írják kitűnő könyvükben, hogy „Ugyan a nullértékek nem képezik részét a hagyományos relációs modellnek, de azért nagyon hasznos és kiemelkedő szerepet játszanak az SQL lekérdezőnyelvben". [6]

2.3 Az A és I jelekről

Állításukkal ellentétben már az „alapító atya", Codd meglepően terjedelmesen és részletesen foglalkozik a problémával, idézett könyvében a teljes 8. fejezet (Chapter 8. Missing Information) erről szól, míg a 9. fejezet az evvel kapcsolatos technikai kritikákkal foglalkozik (Chapter 9. Response to Technical Criticisms Regarding Missing Information). Ezen kívül a 13. számú követelményben (RS-13) foglalkozik a hiányzó ismeret kezelésének módjával: „Azt a körülményt, ha egy érték nem áll rendelkezésre, az egész adatbázisban egységesen és módszeresen kell jelölni, függetlenül a hiányzó érték adattípusától. Erre a célra jelek szolgálnak." Azaz a hiányzó adat jelzése nem érték, hanem valamilyen más módon megvalósított jelzés. (Codd E. F, 1990)

A hiányzó, de az adott körülmények között értelmezhető eset jelölésére szolgálna az ún. A-jel (applicable), míg a nem értelmezhetősége okán való hiányt az I-jel mutatja (inapplicable). A máig általánosan meglévő és használt NULL nem szerencsés megoldás és nem szerencsés elnevezés, mert valójában nem érték (ez az SQL szintaktikán is meglátszik, mert a 17 éveseket úgy kell lekérdezni, hogy „...where Kor=17", míg azokat, akiknek az életkora hiányzik, azokat nem úgy, hogy „...where Kor=NULL", hanem így: „...where kor is NULL"). Továbbá fennáll az a probléma, hogy két különböző esetet kellene megkülönböztetni az egyféle NULL használatával, ami nem megy.

Általánosabban fogalmazva, adatbázisok esetén a kétértékű logika nem elegendő. Az általánosan alkalmazott háromértékű logika is kevés, négyértékű logikára van szükség. Ezt Codd már 1986-87-ben, lassan negyedszázada leszögezte. A négyértékű logika implementálása nem lenne nehezebb vagy időigényesebb, mint

a háromértékűé. Codd rámutat arra is, hogy ennek precíz megoldása fölöslegessé tenné a NULL használatát az SQL-ben. A NULL kiküszöbölése pedig egyszerűbbé és egyértelműbbé tehetné a relációs adatbáziskezelés egyes területeit.

Date ugyancsak a NULL jel használata ellen érvel a relációs adatbáziskezelés kapcsán (Date, C. J., 2005).

2.4 A modellezés mnősége

Lehetne azonban másképpen is érvelni, a valóság modellezésének oldaláról közelítve a problémához. Ebben az esetben viszont, a hasonlóság követelményéből kiindulva megkockáztathatjuk, hogy nem is lenne szükség a „nem értelmezhető" jelentéstartalomra, amikor a NULL jel maradna az „adat még ismeretlen" állapot jelzésére. Ha ugyanis a valóságot modellezem, akkor kiragadom a számomra fontos jelenségeket, továbbá azok tulajdonságait. Egy létező jelenség létező tulajdonsága pedig nehezen elképzelhető, hogy értelmezhetetlen legyen. Másképpen fogalmazva, ha fölmerül adott helyzetben az – eseti – értelmezhetetlenség problémája, az valamilyen modellezésbeli pontatlanságra vezethető vissza.

Ez azonban egy olyan terület, ahol az elméleti teljesség és hibátlanság csak jelentősen nagyobb bonyolultság árán lenne elérhető. Célszerű lehet tehát kisebb engedményt tenni az elméleti teljesség kárára a könnyebb és hatékonyabb modellezés érdekében, nem tévesztve szem elől azt, hogy az adott esetben értelmezhetetlen tulajdonságok gyakorisága a modell jóságának mértékére is utalhat.

Az üres/hiányzó értékek eddig vázolt problémája gyakorlatias eszközökkel kezelhető minden olyan esetben, ahol az múlhatatlanul szükséges, megfelelő további tulajdonságok beiktatásával. Példának okáért – iskolapéldánknál maradva – ha a páciens neme férfi és a szülésszám NULL-jeles, akkor az a nem értelmezhető eset ("n.a."). Ha viszont a páciens neme nő, és a szülésszám ugyancsak NULL-jeles, az meg a létező, de még nem ismert értékű adat esete.

Vannak azonban további problémák is a NULL használata körül, és ezeket máig sem sikerült megnyugtató módon tisztázni. Márpedig nagyon kellemetlen, hogy egy közel negyven éves eszközben, amelynek szilárd matematikai alapjai vannak, ilyen tisztázatlan, nem egyértelmű, nem megnyugtató módon rendezett megoldások (illetve megoldatlan problémák) legyenek.

2.5 Lehetséges megoldások

A legalább háromféle lehetséges megoldás van. Az első lehetőség a valószínűségi adatbázisok használata (többértékű logika), amely a matematikusok számára

vonzó lehet ugyan, de az emberiség nagyobbik hányada nehezen emésztené meg, különösen hétköznapi helyzetekben.

A másik megoldás a NULL jel mellett az EMPTY jel bevezetése volna. Az utóbbi jelezné, hogy az adat értelmezhető, de értéke még nem ismert, míg az előbbi jelezné, hogy az adott helyen az adott adat nem értelmezhető. Ez a megoldás mind az SQL szabvány, mind az adatbáziskezelők módosítását megkövetelné.

A harmadik megoldás nem érintené az SQL szabványt, csak a kezelők módosítását követelné meg: ha a numerikus adatok is karaktersorként tárolódnának, akkor a fentebb írtak szerint lenne ténylegesen üres adat, és a NULL jel elegendő lenne a nem értelmezhető esetekre.

Igaz tehát, hogy a fentebb vázolt okfejtés alapján a hiányzó adatok problémaköre – elméletileg legalábbis – könnyen és gyorsan megoldható lenne. Mivel azonban ez a problémakör évtizedek óta jelen van, nem biztos, hogy összességében ez volna a leghatékonyabb megoldás. Már Codd fontos szempontnak tartja korai munkáiban is a kompatibilitás kérdését mint a felhasználók jogos és igen fontos érdekét (Codd E. F, 1990). Így tehát marad a fennálló állapot – legalábbis még jó darabig – és az adott helyzet megkívánta módon orvosoljuk az elméleti probléma gyakorlati előfordulását technikai eszközökkel.

2.6 Létező problémák az SQL-ben

Date már 1975-ben példát mutatott arra, hogy a NULL használata esetén az SQL esetenként hibás, vagy annak tűnő eredményeket szolgáltat. Rubinson 2007-es cikke (Rubinson, C., 2007) vitába száll Date eredeti véleményével, mire válaszként megjelenik John Grant 2008-as cikke ugyanerről a kérdésről (Grant J., 2008).

Date eredeti példája a következő: Legyen két tábla az adatbázisban: SZÁLLÍTÓ(sno, város) és ALKATRÉSZ(pno, város). Mindkét táblának egyetlen sora van, a SZÁLLÍTÓ esetében ez (s1,'London'), az ALKATRÉSZ esetében ez (p1,NULL), azaz a p1 azonosítójú alkatrész városa ismeretlen. Date kérdése: listázzuk azokat a sno-pno párokat, amelyek esetén a szállító és az alkatrész városa különbözik, vagy pedig az alkatrész városa nem Párizs. Az SQL ma megszokott formájában ez mint lekérdezés így nézhetne ki (az ékezetektől eltekintve):

```
SELECT sno,pno
  from SZÁLLÍTÓ,ALKATRÉSZ
  WHERE SZÁLLÍTÓ.város!=ALKATRÉSZ.város OR
    ALKATRÉSZ.város<>'Párizs';
```

Az SQL válasza egy üres tábla. Date viszont azt mondja, hogy az (s1,p1) páros az elvárt válasz, és úgy érvel, hogy ha p1 ismeretlen városa Párizs lenne, akkor a

feltétel második fele nyilván teljesül. Ha viszont p1 városa nem Párizs, akkor teljesül a feltétel első fele, ezeket a vagy logikai operátor kapcsolja össze, tehát a teljes feltétel a p1 bármilyen városa esetében igaz, így a válaszban benne kellene legyen az (s1,p1). De nincs. Pedig jogosan érezzük úgy, hogy a világ bármely városára igaz az az állítás, hogy az vagy Párizs, vagy nem az.

Egy másik, hasonló problémát mutat be Codd egyik példája. Legyen egyetlen táblánk, amelyik alkalmazottak születési éveit (is) tartalmazza: ALKALMAZOTT(ano,...,születés_éve). Legyenek a következő évszámok a táblában: 1939, 1940, 1940, NULL.

```
SELECT *
  from ALKALMAZOTT
  WHERE születés_éve<1940 OR
    születés_éve=1940 OR
    születés_éve>1940;
```

Ebben az esetben a válasznak mind a négy sort tartalmaznia kellene, hiszen hiába ismeretlen a negyedik dolgozó születési éve, akkor is biztosra vehető, hogy ha az nem is pont 1940, akkor azt vagy megelőzi, vagy pedig nagyobb annál. Codd azt javasolja, hogy a jövőbeni adatbáziskezelőket fel kell készíteni az efféle egyszerű tautológiák felismerésére.

Véleményem szerint nem az a probléma, ha egy lekérdezésre nem – a saját logikánk szerint – elvárható választ kapjuk. Az első példabeli jelenségre ugyanis az a válasz, hogy azért van így, mert az ANSI így alkotta meg az SQL szabványt – dacára Codd, Date, Maier és mások munkásságának. Azazhogy a példabeli jelenség az SQL-nek nem hibája, hanem tulajdonsága. Az igazi probléma az, ha valamitől olyasmit várunk el, ami nem lenne feladata, ha megalapozatlanul – esetleg az értelmezési tartományon túlra – extrapolálunk.

Mindkét példa esetében igaz az, hogy a természetes észjárás alapján, pontosabban az egyik lehetséges természetes észjárás alapján elvárt választ nem kapjuk meg. A másik lehetséges megközelítési módja az efféle problémáknak ugyanis az, hogy az ismeretlen érték – éppen ismeretlen mivolta okán – nem vehet részt érdemi értékelésben. Mindkét példabeli esetben ugyanis az elvárt válasz feltételezné, hogy az SQL kérés feldolgozása során a kérdés jelentésén alapuló értelmezést végezzen az SQL parancsértelmező. Ez pedig – szerintem – messze nem feladata. Már csak azért sem, mert az említett példák aránylag egyszerűek. Ha elvárnánk az SQL-től, hogy az ilyesféle eseteket a példabeli módon kezelje, akkor igen komoly tartalmi-logikai elemzést (intelligenciát) kellene beleépíteni, hogy minden lekérdezés esetén végezze el az ehhez hasonló elemzéseket – hogy milyen eredménnyel, azt előre nem lehet megjósolni. Tehát jobb, ha inkább következetesen nem teszi.

Összefoglalás

Információsnak nevezett társadalmunkban ma már alig van olyan hétköznapi tevékenység, melynek során ne találkoznánk lépten-nyomon adatbázisokkal. Ezen találkozások többségében, általában, mi vagyunk a végső-felhasználók, azaz nem mi terveztük és kezeljük azt, csak hasznát élvezzük (vagy nehézségeivel küzdünk). Az informatikának az adatbázisok tervezése és kezelése csak egy szűk szakterülete a sok közül, mégis hasznos, ha legalább valamennyire tisztában vagyunk a relációs adatbázisok működésének alapelveivel, sajátosságaival. Ezen sajátosságok közül vettünk sorra néhányat a jelen cikkben.

Hivatkozások

[1] Childs D. L.: Description of a set-theoretic data structure, Proceeding AFIPS '68 (Fall, part I) Proceedings of the December 9-11, 1968, fall joint computer conference, part I ACM New York, 1968.

[2] Codd E. F.: Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks, San Jose, IBM Research Report RJ599, 1969.

[3] Codd E. F.: The Relational Model for Database Management – version 2. Addison-Wesley Publishing Company, h.n., 1990.

[4] Date, C. J.: Database in Depth: Relational Theory for Practitioners O'Reilly, h.n., 2005.

[5] Grant J.: Null Values in SQL. In: ACM SIGMOD Record, Association for Computing Machinery Special Interest Group on Management of Data, 2008. szeptember, XXXVII. évf. 9. szám

[6] Gulyás Sándor: 50 év az adattárolás történetében, avagy miért winchester a winchester? In: Kék Rózsa - az IBM Magyarország ügyfélmagazinja, 2006/4.

[7] Marci K. Sun, Devon Prince: Development Strategy for Mechanical Evaluation Formation, in Proceedings of 5th International Conference on Industrial Application on Computational Intelligence, Budapest, Hungary, July 10-13, 1980, pp. 159-165

[8] Rejtő Jenő: Az elátkozott part. Magvető Könyvkiadó, Budapest, 1964.

[9] Rubinson, C.: Nulls, Three-Valued Logic, and Ambiguity in SQL: Critiquing Date's Critique. ACM SIGMOD Record, December 2007 (Vol. 36, No. 4)

[10] Ullman J. - Widom J.: Adatbázisrendszerek – Alapvetés. Panem Könyvkiadó, Budapest, 1998.