Az adatbázistervezés alapjai és titkai

Avagy az út az adattól az adatbázison át az információig

Halassy Béla

© Dr. Halassy Béla, 1994.

Emőkének és Melindának

TARTALOMJEGYZÉK

ELŐZMÉNYEK **8**
ELŐSZÓ **9**
BEVEZETÉS - I **12**
1. ADAT ÉS INFORMÁCIÓ **14**
1.1 Az alapfogalmak szemléletmódja 14
1.2 Az adat - első megközelítésben 14
1.3 Az ismeretszerzés momentumai 15
1.4 Az adat és az információ lényege illetve viszonya 16
1.5 Szintaktika, szemaktika, pragmatika 18
1.6 Miért hibádzik a kommunikáció? 19
1.7 Mit lehet megtudni? 20
Ellenőrző kérdések - 1 21
2. AZ ISMERETKEZELÉS KÉT MÓDJA **22**
2.1 Az ismeret hordozó közegei 22
2.2 Az ismeret négy dimenziója 22
2.3 Történeti kitérő: A „szemantikai adatbázis" 24
2.4 Kétféle mondat 25
2.5 Adatszerű ismeretkezelés 27
2.6 Szövegszerű ismeretkezelés 28
2.7 A kétféle ismeretkezelés viszonya 29
Ellenőrző kérdések - 2 30
3. AZ ADATBÁZIS LÉNYEGE **32**
3.1 Állomány, adatbank, adatbázis 32
3.2 Egyedtípus, -előfordulás, -halmaz 33
3.3 Tulajdonságtípus, -érték, -értékhalmaz 34
3.4 Két relativitás 36
3.5 A bináris relációk modellje 37
3.6 Az azonosítás problémaköre 39
3.7 Kapcsolattípus, -előfordulás, -(al)halmaz 40
3.8 Az adatbázis 42
3.9 Három további kérdés 44
Ellenőrző kérdések - 3 45
4. AZ ADATBÁZIS HÁROM SZINTJE **46**
4.1 Egy alapvető probléma: az eszközorientáltság 46
4.2 Az adatok két aspektusa 47
4.3 Az adatbázis kétféle tartalma 48
4.4 Az adatbázis fogalmi és logikai szerkezete 49
4.5 Két lényeg határán 52
4.6 Az adatbázis fizikai szerkezete 53
4.7 A fizikai függetlenség 54
4.8 A vertikális leképezés 55
Ellenőrző kérdések - 4 56

5. AZ ADATBÁZIS VETÜLETEI 58
5.1 Egy alapvető probléma: a nézetorientáltság 58
5.2 A nézetorientáltság káros következményei 59
5.3 Globális és parciális adatbázisszemlélet 60
5.4 Az „adatmodell" jelentése és tartalma 63
5.5 Az ANSI-SPARC architektúra 64
5.6 A szintek megfeleltetése, modellek és sémák 65
5.7 Adatmodell-elmélet és adatbázis-gyakorlat 66
5.8 Adatmodell-reprezentációk 67
Ellenőrző kérdések - 5 69
6. A MODELL ALAPVETŐ SZERKEZETE 71
6.1 Az adatmodell, mint rendszer 71
6.2 Az egyedek kétféle struktúrája 71
6.3 A tulajdonságok alapvető szerepei 73
6.4 Abszolút és relatív szerep 74
6.5 Hierarchikus inhomogén kapcsolatok 76
6.6 Hálós egyedviszonyok - 1 78
6.7 Hálós egyedviszonyok - 2 81
6.8 Újabb tulajdonság szerepek 83
6.9 A kölcsönös egyedviszony 84
6.10 Gyorsmérleg az adatszerkezetről 86
Ellenőrző kérdések - 6 86
7. SZERKEZETI FINOMSÁGOK 88
7.1 Az adatmodell sokszínűsége 88
7.2 A „száz-százalékos elv" 88
7.3 Az értéktartomány 90
7.4 A szerepnév 91
7.5 Ismétlődés és szerepnév 93
7.6 A visszamutató egyedviszony 94
7.7 A hierarchikus homogén viszonyok 95
7.8 Családfa és házastárs viszonyok 96
7.9 Háromféle „üres" érték 99
7.10 Egyedaltípusok 100
7.11 Szerkezeti összefoglalás 103
Ellenőrző kérdések - 7 104
8. ADATBÁZISKEZELÉS 107
8.1 Az ismeretek természete 107
8.2 Az adatkezelés szintjei 107
8.3 Történeti kitekintés: Az adatbázisgépek 110
8.4 Adatkezelés és adatfeldolgozás 110
8.5 Szabad-e ...? 112
8.6 Állomány- és adatbáziskezelés 113
8.7 A természetes adatfeldolgozási lánc 115
8.8 Technikai adatok 117
8.9 Az adatbázis alapvető titka 118
Ellenőrző kérdések - 8 119
9. A METAADATBÁZIS 121
9.1 Az adatbázisok két szemléleti síkja 121
9.2 A metaadatbázis lényege 122

9.3 Történeti kitekintés: Az adatszótár ... 124
9.4 A metarendszerek természete ... 126
9.5 A metaismeretek fajtái ... 127
9.6 Passzív, félaktív és aktív adatszótár ... 128
9.7 Tervezés ábra alapján ... 129
10. ADATBÁZIS-MENEDZSELÉS ... 131
10.1 Mitől lehet rossz az adatbázis? ... 131
10.2 Hány az adatbázis? ... 132
10.3 Alkalmazási adatszabványok ... 133
10.4 Fejlesztési adatszabványok ... 135
10.5 Fejlesztési eljárászabványok ... 136
10.6 Menedzselési szabványok ... 137
10.7 Változásmenedzselés ... 139
10.8 Szervezeti feltételek ... 140
11. AZ ÚT ... 143
11.1 Előrehozott zárszó ... 143
11.2 Fogalom születik ... 144
11.3 Több szem 145
11.4 Adat születik ... 146
11.5 Az ismeretek szerkesztése és első betöltése ... 147
11.6 Bizonylatok ... 148
11.7 Adatbevitel ... 149
11.8 „Új" ismeret születik ... 150
11.9 Adatkimenet és behangolás ... 151
11.10 A hosszú és kanyargós út ... 152
BEVEZETÉS - II. ... 155
12. AZ ADATMODELLEK HIBÁI ... 157
12.1 Az adatbázis hibáinak a forrásai ... 157
12.2 Az adatmodell lényege ... 158
12.3 Ábrázolási konvencióink ... 159
12.4 Tipikus adatmodellezési hibák ... 160
12.4.1 Nyílt logikai átfedés ... 161
12.4.2 Látszólagos logikai átfedés ... 162
12.4.3 Rejtett logikai átfedés ... 163
12.4.4 Jelsor-ellentmondások ... 164
12.4.5 A logikai átfedés hiánya ... 165
12.4.6 Fizikai átfedés ... 166
12.4.7 Kiegyensúlyozatlanság ... 167
12.4.8 Tisztázatlan tartalmak ... 168
12.4.9 A modell kettős arculata ... 168
12.4.10 A hibák emberi következményei ... 169
12.5 Az adatmodellezési hibák gyökerei ... 170
12.5.1 Nézetvakság ... 170
12.5.2 Szintvakság ... 171
12.5.3 Szerepvakság ... 171
12.5.4 Bemenet/kimenet vakság ... 172
12.6 Az adatmodellezés céljai ... 173
Ellenőrző kérdések - 12 ... 174

13. FÜGGÉSEK ÉS NORMALIZÁLÁS 176
13.1 Egyszerű szabályok 176
13.2 Funkcionális függés 177
13.3 Tartományfüggés 180
13.4 Nem-normalizált egyedtípusok 183
13.4.1 Az ismétlődés és káros hatása 183
13.4.2 Az ismétlődés elvi alapja 184
13.4.3 A normalizálás első lépése 184
13.4.4 További tudnivalók az ismétlődésről 188
13.5 A normalizálás lényege 190
13.6 Normalizálás és struktúra 190
13.7 Ismétlődés és kétségek 192
Ellenőrző kérdések - 13 193
14. ALAPVETŐ NORMÁLFORMÁK 195
14.1 A normalizálás alapjai 195
14.2 A második normálforma 196
14.2.1 A részleges függés és következményei 197
14.2.2 A normalizálás második lépése 199
14.2.3 Normálforma és szemantika 200
14.2.4 A második normálforma és a modellstruktúra 202
14.3 A harmadik normálforma 203
14.3.1 A tranzitív függés és következményei 204
14.3.2 A normalizálás harmadik lépése 205
14.4 A normalizálás természete 206
14.5 Normalizálási sorrend 207
14.6 A dekompozíció sajátosságai 209
14.7 Az alternáló kulcs 211
Ellenőrző kérdések - 14 213
15. MAGASABB NORMÁLFORMÁK 214
15.1 Hány, melyik és milyen a kulcs? 214
15.2 A Boyce-Codd normálforma (BCNF) 215
15.2.1 A több összetett kulcsjelöltből fakadó gondok 215
15.2.2 A kulcstörő függés 217
15.2.3 BCNF problémák 218
15.2.4 A normalizálás negyedik lépése 220
15.3 A negyedik normálforma 222
15.3.1 A többértékű függés és az általa okozott problémák 223
15.3.2 A normalizálás ötödik lépése 224
15.3.3 Megjegyzések a többértékű függéshez 225
15.4 Az ötödik normálforma 227
15.4.1 A kapcsolásfüggés és az általa okozott problémák 227
15.4.2 A normalizálás hatodik lépése 228
15.4.3 Megjegyzések a végső normálformához 229
Ellenőrző kérdések - 15 230
16. A CSOPORTOK CSAPDÁI 232
16.1 Eltérő gondolkozásmódok 232
16.2 A függéstáblázat 233
16.3 Két egyszerű és három ismert függési helyzet 234
16.4 A belső kulcstörő függés 236
16.5 A metszetfüggés 238

16.6 A csoportfüggés 241
16.7 Tanulságok és a döntési táblázat kiegészítése 242
16.8 Pszeudo-tranzitivitás 243
Ellenőrző kérdések - 16 246
17. NORMALIZÁLÁSI ELJÁRÁSOK 247
17.1 A normalizálás, mint feldolgozás 247
17.2 Kapcsolathiány 248
17.3 A dekompozíció összefoglalása 251
17.4 Az „univerzális reláció” 251
17.5 Normálforma szintézis 252
17.6 Kérdezni tudni kell 254
17.7 Függések és szerkezetek 255
17.8 Két összetett szerkezeti probléma 258
17.9 Az adatmodell „fonalai” 260
17.10 Apró szerkezeti titkok 264
17.11 A kulcsmátrix 265
18. SAJÁTOS SZERKEZETI TÉNYEZŐK 268
18.1 Minőségi adatmodellezés 268
18.2 Többszörös inhomogén kapcsolatok 269
18.3 A szerepnevek függései 271
18.4 Homogén viszonyok 273
18.5 A feltételes függés és az egyedaltípus 275
18.6 Unáris egyedek 278
18.7 Szinguláris egyedek 280
18.8 Konstansok és tulajdonságstruktúrák 282
18.9 Az osztályozás kérdésköre 284
19. TIPIKUS TERVEZÉSI HIBÁK 286
19.1 Fegyelmezett fantázia 286
19.2 Általános tervezési problémák 287
19.2.1 Név-variációk 287
19.2.2 Absztrakciós gondok 288
19.2.3 A körülírás hiánya 292
19.3 A „CSAK” szindróma 293
19.4 Egyéb csacskaságok 296
19.5 Statikus szemlélet 298
19.5.1 Az idő modellezése 298
19.5.2 Esemény, változás, állapot 299
19.6 Modell-sablonok 302
19.7 Szinttévesztés 304
20. ELEMZÉS ÉS DOKUMENTÁLÁS 305
20.1 Egy mintapélda 305
20.2 A mintapélda tartalmi hibái 306
20.3 Az adatmodell dokumentálása 308
21. TERVEZÉSI ESETTANULMÁNY 313
21.1 A példa kerete 313
21.2 A kiinduló modell 314
21.3 Az esettanulmány megoldása elé 322
21.4 Ismerkedés az elemzendő tervvel 323
21.5 Az egyértelműség elemzése 324

21.6 Azonosító- és azonosságelemzés 325
21.7 A példa átmenetileg javított változata 327
21.8 A rejtett redundanciák elemzése 328
21.9 Kapcsolatelemzés 329
21.10 A példa időleges megoldása 330
21.11 Zárszó 333
FELADATMEGOLDÁSOK 335
IRODALOMJEGYZÉK 339
FOGALOMJEGYZÉK 341

ELŐZMÉNYEK

A szerző 1972-ben találkozott először az adatbázisokkal. 1974-ben irányult a figyelme az adatbázistervezés felé. A UNDP által támogatott **SZÁMOK** jó műhelynek bizonyult kutatásaihoz. Már akkor megismerhette a legkorszerűbb irányzatokat. Lehetősége nyílt arra, hogy több tízezer oldalnyi szakanyagot áttanulmányozzon és összeállítson egy terjedelmes magánkönyvtárat az adatbázisokra vonatkozó irodalmakból. Ennek alapján adta ki 1978-ban az „Adatbázisok kezelésének alapvető kérdései" című nívódíjas munkáját, amelyet 1982-ben újranyomtak. 1980-ban készült el az „Adatmodellezés, adatbázistervezés" című műve. Mindkét kiadvány kézikönyvvé vált az adatbázisok iránt fogékonyak számára.

1982-re a szerző érdeklődése mindinkább a gyakorlat felé fordult. Egyrészt az adatbázistervezést támogató szoftverek fejlesztésébe fogott, másrészt ezeket az eszközöket mindennapos munkákban alkalmazta. 1981-ben készült el ***SZIAM*** nevű első adatbázis-normalizáló programja, amelyre az IBM is felfigyelt és megvette a licencét. Ez volt a legelső szoftver, amit az IBM a kelet-európai országokból valaha is vásárolt. 1982-ben a szerzőt az IBM európai kutatóközpontjában (E.S.R.I.) tartandó előadássorozatra hívták meg. A „Gyakorlati adatmodellezés" című előadását négyszer ismételtették meg.

Ekkor már a munkát a **SZÁMALK**-ban folytatta. Az újabb könyvet („Adatmodellezés a rendszerfejlesztésben", 1983.) újabb szoftver követte. 1984-ben készült el az ***Ádám & Éva*** adat- és eljárásmodellezési segédeszköz, amelyet a BMW AG. is megvásárolt. A vonalat a ***SYDES***, majd az ***ABLAK*** adatbázistervező programok vitték tovább. Időközben az író tucatnyi adatbázistervezési projekt vezetőjeként vagy szaktanácsadójaként ütköztethette elméletét a gyakorlattal. Ennek során ipari, mezőgazdasági, pénzügyi, közlekedési, kereskedelmi, önkormányzati, múzeumi, híradási, jogvédelmi stb. területeken volt lehetősége bepillantani az ismeretek összefüggéseinek a rejtelmeibe.

1987 végén az elméleti kutatás és a gyakorlati alkalmazás egy időre megszakadt. Két kórházakban több tucat operációval eltöltött év után a szerző az **ÁB - ÁBEL Kft**-nél találkozott ismét kedvenc adatbázisaival. 1992-ben a Számítástechnika hasábjain jelent meg „Az adatbázistervezés titkai" című cikksorozata, amely ennek a könyvnek a közvetlen előfutára. Egy-egy műtét között összeállította legújabb adatmodell-elemző szoftverét, az ***AMOR***-t és az ***ÁLOM*** becenevű általánosított adatbáziskezelő rendszerét.

Több, mint húsz évnyi elméleti kutatás, szoftverfejlesztés és gyakorlati alkalmazás képezi előzményként ennek a műnek a hátterét...

ELŐSZÓ

Három kérdéskör motoszkál a fejemben. Az egyik az, hogy ***mikor*** is írom ezt a könyvet? 1983-ban jelent meg az „Adatmodellezés a rendszerfejlesztésben” című munkám. Nagyon sok barátom és kedves kollégám szerint túlzottan korán. Szerintük tíz évvel ezelőtt még nem érett meg a helyzet mondanivalóm befogadására. Bezzeg most, amikor minden vállalat átalakítja, korszerűsíti az információs rendszerét, amikor már megvehet egy csinos gépet és egy valódi relációs adatkezelőt, most szükség lenne az adatbázistervezés ismereteire. Vagyis ezzel a könyvvel szerintük végül is elkéstem.

Én pedig nagyon félek attól, hogy még mindig túl korai ez a kiadvány. Legalábbis bizonyos részeinek a befogadására, megemésztésére még nem mindenki lesz képes. A veréb, a varjú és a sas dilemmájával kell szembenéznem. Azok, akik még csak az xBASE-szerű adatkezelő verebekkel bíbelődnek, nem fogják megérteni, hogy miért van szükség ilyen „bonyolult” adatbázistervezési módszerekre. Még több bajuk lesz velem azoknak, akik láttak már karón varnyút, azaz megamini kategóriájú relációs adatkezelő rendszert. „Hát már az is gond? Húzok egy indexet, aztán annyi.” módon gondolkoznak. A valóban sokféle technikai lehetőségtől elbűvölve nem éreznek rá, hogy én miért mindig a valóság ismereteinek a hű és természetes tükrözését helyezem előtérbe a technikai megoldásokkal (pl. index) szemben. Én a jelenlegieknél sokkal, de sokkal jobb kezelőket is el tudok képzelni, amelyek úgy viszonyulnak a relációsakhoz, mint sas a varjúhoz. Mondanivalómat e legfelső szinthez szabtam és ezért félek, hogy nem fog mindenki velem szárnyalni.

A második kérdés az, hogy ***kinek*** is írom ezt a könyvet? Az adatbázistervezés komoly technikai tudást feltételez. Ezért jogosan tűnik úgy, hogy ez a kiadvány elsősorban a mai és a leendő **fejlesztőnek**, a szakembernek szól. Csakhogy az információs rendszerek fejlesztésének van két további résztvevője is. Az egyik, a kulcsfigura, a **vezető**. Ha a menedzser nincs tisztában az adatbázis tervezésének a buktatóival, ha azt hiszi, hogy egy ma megálmodott bizonylat alapján holnap már élő adatbázis fog működni, akkor a fejlesztő hiába ismeri az adattervezés titkait. Egyszerűen nem kap elég időt arra, hogy érvényesítse a tervezés minimális követelményeit. A vezető már x alkalommal tapasztalta, hogy gyorsan nem lehet jó adatbázist építeni. Ennek ellenére az y-dik helyzetben is az azonnali megoldást sürgeti. Nemigen pozitív a harmadik résztvevő, a **felhasználó** szerepe sem. Elhamarkodott igényekkel áll elő, amelyeket azután nap mint nap módosít. Pedig tudjuk, hogy az adatbázis leginkább a strukturális változásra érzékeny. Tegyük mindehhez a felhasználói önzést. Az én adatom az én váram! Abba ne szóljon bele senki, csak legfeljebb az én fejlesztőm, mert nélküle semmire sem megyek. A valójában közös adatokat alkalmazó felhasználók úgy rejtegetik egymás elől az „adatbázisaikat”, mint a legértékesebb kincset.

A jó adatbázis kialakításának a sikere a három szereplő értelmes együttműködésétől függ. Ezért ez a könyv elvileg mindhárom partinak szól. Gyakorlatilag nem bízom abban, hogy a kiadvány megtalálja a címzetteket és eléri a kívánt változást. Hiszen még saját kollégáimmal, a fejlesztőkkel is meg kell küzdenem.

Persze három réteggel nem fogok sokra menni. Az első csapatot azok az aggcsecsemők alkotják, akik még mindig a hetvenes évek számítástechnikai emlőit szopogatják. Ők már vén motorosok a szakmában, nekik ne mondjanak már semmit, mert nincs új a nap alatt. A beverklizett módon, az újabb eszközökre, módszerekre, lehetőségekre fittyet hányva tákolgatják össze ósdi „rendszereiket” a „jól bevált” sablonok szerint.

Azután itt van az ifjútitánok hada. A mindig újabb és újabb gépek és programok mellett „elkötelezetteké”. Sokszor megdöbbent, hogy velük társalogva azon kapom magam, hogy nem emberrel, hanem hardverrel vagy szoftverrel beszélek. Te, és ez a rendszer ezt csinálja, meg azt tudja... Sértődött hallgatás a válasz, amikor megkérdezem, hogy és.... És Te, fiam, személy

szerint Te mit csinálsz és mit tudsz? A drága gyermek nem érti, hogy egy dolog a technika ismerete, és teljesen más annak mindenki javára való használata.

Persze vannak olyanok is, akik mindezzel nem törődnek. A hullámlovagokról van szó. A félművelt ámítástechnikus a papa pozíciója miatt bekerül a világbanki kölcsönnel támogatott projektbe. Naponta keres annyit, amennyi egy becsületes dolgozó havi bére. Persze semmihez sem ért valójában, de ez őt nem is érdekli. Mondanom sem kell, hogy ez a könyv nem neki, nem is a tanulni képtelen aggcsecsemőnek vagy a tanulás értelmét nem ismerő ifjútitánnak - aki maga is hamar koravénné válik - szól. Hát akkor kinek?

Ezzel eljutottam a harmadik kérdéshez. ***Miért*** is írom ezt a könyvet? Azért, mert Benned bízom. A lélekben fiatalban. Aki akar és tud véresen-komolyan játszani. Akit a körülmények korlátoznak; aki nem képes mindig a legjobbat alkotni; de aki egy kártyaparti után napokkal is elgondolkozik azon, hogy hol szúrta el és miként lett volna - **jobb**. Ez a könyv a felnőtten komoly fiataloknak, a humorra képeseknek szól. Akik tudják, hogy a humor nem vicc, hanem az élet és embertársaink mélyen bölcs, megértő, megmosolygó szeretete.

Két szép lányomnak ajánlom ezt a könyvet akkor is, ha ők nem követik az én hivatásomat. Igen, hivatást mondtam, nem szakmát. Ez az adatbázistervezés legeslegfőbb titka. Nem azért kell jó adatbázist tervezni, hogy engem megdicsérjenek (szakma). Hanem azért, hogy másokat jól el tudjunk látni az élet nélkülözhetetlen kellékével, az ismerettel (hivatás). Elsősorban nem a technikát, hanem ezt a szemléletet kell elsajátítani. Ha valamit sikerül megéreztetnem ebből a lényegből, akkor ez a könyv nem íródott hiába.

Az adatbázisok tervezéséhez, felnőtté érleléséhez éppen annyi türelem szükséges, mint a fiatalokhoz. Nem mindig látod erőfeszítéseid azonnali gyümölcsét. Nem kapod meg rögtön az annyira várt köszönő szavakat. Majd egy napon, amikor a komoly, nehéz, becsületes játék a te részedről már régen véget ért, amikor már minden a helyére rendeződött és neked már nincs szereped, amikor már nem is számítasz rá, váratlanul így szólnak hozzád: „Köszönöm”. Lehet, hogy ezt a szót így soha meg nem hallod. Csak éppen a szomszéd utcában a Kovács kedves lesz a nejéhez, mert húsz helyett két másodperc alatt adták ki az információit a Te jó adatbázisterved miatt. No látod, ezért hivatás az adatbázistervezés.

Már csak annyi maradt hátra, hogy megköszönjem mindazoknak a segítségét, akik e könyv megszületéséhez hozzájárultak. Legfőképpen pedig köszönetet mondjak Annak, akitől kaptam a tudást és annak átadásának kényszerét illetve lehetőségét.

Halassy Béla

I. RÉSZ

Az adatbázistervezés alapjai

BEVEZETÉS - I.

A jó adatbázis kialakításához (is) két dologra van szükség. Az egyik az alapos ***technikai tudás***. A másik kellék, amely nélkül az előző szinte semmit sem ér, a megfelelő ***szemlélet***. Ennek a műnek az első részében - a második megalapozásaként - elsősorban az adatbázis helyes szemléletének a feltárása lesz a feladatunk. Legyen szabad egy triviális példával élve előre megvilágítani mondanivalónk tartalmát.

A lényeget az érti meg leginkább, aki látja a máriás, az ulti és a bridzs közötti különbségeket. A máriás a gyerekek játéka. Ketten játsszák. A szabályok roppant egyszerűek és előre adottak. Ebben a játékban szinte nem is lehet hibázni. Ha valaki magáncélra készít néhány egymástól valóban független számítógépes nyilvántartást, akkor az olyan, mint a máriás. Gyerekjáték. Persze az alapszabályokat ekkor is kell tudni és azokat illik is betartani.

Nálunk ma a számítástechnikai ulti a divat. Nem babra megy ez a játék: milliós alapon művelik. A szabályok már összetettebbek a máriásénál. Együtt kell működnünk a partnerünkkel és figyelnünk kell minden implicit jelzésre, akár az ellenfélére is. Bármennyire is izgalmas, az ulti rutinjáték. Ha valakinek van lapja vagy ügyesen csinál magának, akkor már nemigen tévedhet. Ebben a játékban a fantáziának nincs túl sok szerepe. Rutinosan, több évtizede megismert szabályok alapján játsszuk a mai magyar számítástechnikát.

A kártyajátékok királynője, a bridzs, teljesen más jellegű. Nem csupán az a cél, hogy nyerjünk. Úgy illik, hogy a lehető legtöbbet hozzuk ki a lapunkból. Sőt, még a vesztes is nyerhet akkor, ha minimalizálja az ellenfél nyereségét. Az alapszabályok adottak, de a bridzsben nagyon fontos szerepet játszanak az ún. konvenciók, a széleskörűen alkalmazott és az összes játékos által ismert megegyezések. Az lenne a jó, ha a számítástechnikát nem ultiként, hanem bridzsként játszanánk.

A jó játékosnak ismernie kell magát a lapot. Tehát tudnia kell azt, hogy mi az adat. Viszont nem egy, hanem tizenhárom lapot kap, amely együtt már más minőség. Tisztában kell lenni azzal, hogy a lapkombinációból mit lehet kihozni (információ). Könyvünk első fejezetében ***az adat és az információ*** alapfogalmaival ismertetjük meg az olvasót.

A társasági bridzs és a versenybridzs alapjai közösek, de szabályai mások. A számítógépes ismeretkezelésnek is több módja van, noha a lényeg közös. A második fejezetben ***a szövegszerű és az adatszerű ismeretkezelés*** azonos és eltérő vonásait tárjuk fel. Rámutatunk arra, hogy az utóbbi lényegesen feszesebb, de ugyanakkor több lehetőséget nyújtó, izgalmasabb ismeretszerkesztést követel meg.

Vannak, akik úgymond bridzselnek, de ennek a nagyon komoly játéknak az úgynevezett kávéházi változatát művelik. Az antibridzset. Mindenki a saját feje szerint mérlegel, önkényes, mások által nem ismert vagy régen túlhaladott konvenciókat használ. Sajnos ez jellemző a mai számítástechnikusok egy részére is. Ezért a könyv harmadik fejezetében el kell mondanunk, hogy mi melyik verziót játsszuk. Vagyis azt, hogy szerintünk mit is jelent maga az ***adatbázis***.

A bridzs az a játék, amit mindig lehet tanulni, de sohasem lehet teljesen megtanulni. Az ember mindig rábukkan egy újabb lapösszefüggésre, egy jobb megoldásra. A lapkombinációkat fokozatosan mérlegeli. Az első ránézésre sejti, hogy ebből a lapból egyáltalán az hozható ki, hogy... Azután mélyebben értékel. Bár ez a hasonlat nem teljesen ül, az adatbázis esetében is ***szintenként vizsgáljuk az adatainkat***. Erről szól a negyedik fejezet.

A bridzsben nincs egyéni játék. A párosoknak - sőt, még az ellenfélnek is - ugyanazokat a lapokat kell látniuk. Az egész licit arra megy ki, hogy „megbeszéljük” kettőnk együttes lapját, amit így persze az ellenfél is megismer. Az adatbázis lényege az, hogy egyéni lapjainkból indulunk ki, de a végén a lapok együttese alapján játszunk. ***Az adatbázis közös, bár annak egyes részeit másként látjuk***. Ezt a gondolatot fejti ki az ötödik fejezet.

Mindeddig a játék alapszabályairól volt csak szó. Arról nem beszéltünk, hogy milyen eltérő lapkombinációk fordulnak elő a bridzsben és azokkal mit lehet kezdeni. Vannak alapszintű szerkezetek, hiszen az egyenlőtlen és az egyenletes elosztású lapokat már minden kezdő kártyás meg tudja különböztetni egymástól. Ezeken belül vannak rafinált elrendezések. Mást jelent a hetes hosszú treff fej nélkül, mint a hatos fejjel. A könyv hatodik és hetedik fejezete ***az alapvető és a bonyolultabb adatkombinációkat*** mutatja be.

A bridzs két játékfázisból áll. A licitből és a lejátszásból. Az első szakaszban „felépítik" a játékot, vagyis meghatározzák, hogy a lapból mit akarnak kihozni. A második részben próbálják megvalósítani az elképzelést, vagyis ténylegesen kinyerni a lapból azt, amit abban láttak. Az adatbázis logikája is hasonlít erre. Van egy szakasz, amelyben az adatbázist megtervezzük és felépítjük. Majd használatba vesszük, kísérletet téve arra, hogy a lehető legtöbbet hozzuk ki belőle. Az ***adatbázis kezeléséről*** a nyolcadik fejezet szól. Itt már kimondottan versenybridzsről van szó. Vagyis különböző játékosok ugyanabból a pakliból teljesen eltérő eredményeket csiholhatnak ki. Egyazon adatbázisból ki több, ki kevesebb ismeretet nyer. Ki könnyebben, ki nehezebben - játéktudása szerint.

Minden hasonlat sántít. Bridzs-analógiánk is lassan erőltetetté válik. De talán mégsem haszontalan elmondani, hogy a versenybridzs leosztásait megörökítik, hogy a félreértéseket és az azokból fakadó vitákat elkerüljék. A konkrét lapkombinációkon kívül általános szabálygyűjtemények figyelmeztetik a versenyzőket az alapvető tudnivalókra. A verseny kezdetekor - nem közben, nem utólag - a játékosok bejelentik az általuk alkalmazott konvenciókat. Mert a játék csak akkor fair, ha azokat mindenki ismeri. Az adatbázis-játéknak is vannak általános szabályai, specifikus megegyezései és az adatbázisnak van konkrét „leosztása", szerkezete. Úgy illik, hogy mindezeket a tényezőket egy helyen, mindenki által „fellapozhatóan", megismerhetően rögzítsük. Az adatbázis általános leírására szolgál a kilencedik fejezetben bemutatott ***metaadatbázis***.

A bridzsversenyeken a tulajdonképpeni játékosokon kívül számos egyéb személy is közreműködik. A kiírók, a zsűri, a bírók, az osztók stb. Az adatbázis is ilyen összetett játék. A fejlesztő és felhasználó párosán kívül több résztvevője van, mindegyik sajátos szerepkörrel. Ezekről a ***speciális funkciókról*** szól a tizedik fejezet.

Végül is a bridzsnek az a lényege, hogy játsszák. A megfelelő előkészületek után valahol összejönnek, leülnek, osztanak és belefeledkeznek a véres-komoly játék varázslatába. A tizenegyedik fejezet ***összegzi az ismeret útját*** az adattól az adatbázison át az információig.

A mondanivaló megértését a példákon és ábrákon kívül ellenőrző kérdésekkel is segíteni próbáljuk. A feladatokra adott helyesnek tartott válaszokat a kérdésben mutatott betűvel vagy a válasz sorszámával kell megjelölni. Adott esetben több helyes és rossz válasz is létezik. A könyv végén találhatók a helyes megoldások. Ha valakinek kétségei támadnak, akkor keresse fel tanárját, barátját stb. Végső esetben pedig a szerzőt.

1. ADAT ÉS INFORMÁCIÓ

1.1 Az alapfogalmak szemléletmódja

Két dolog van a világon, amely nem fogy el soha, ha másokkal is megosztják. Az egyik a szeretet, a másik pedig az ismeret. Ez a két dolog szorosan összefügg. Az ember társas lény. Azért kapott partnereket, hogy legyenek, akikkel együttműködve meg tudja valósítani élete értelmét. Megtalálja azokat, akikre támaszkodhat és megkeresse a támogatandókat. Egész életünket a másokkal való ***kommunikáció***, vagyis az ismeretek cseréje határozza meg. A kölcsönös informálás nemcsak óriási lehetőség, nemcsak saját létünk nélkülözhetetlen feltétele, hanem egyben a szeretet által diktált kötelezettség is, mivel mások életéért is felelősek vagyunk.

A fentiekből következik, hogy a magát tudatos lénynek valló embernek el kell sajátítania a kommunikálás megfelelő módját. Ehhez pedig sokat kellene tudnia magáról az ***ismeretről***. Persze az emberek a hétköznapi érintkezés során nem boncolgatják a közlések természetét és nem is rendelkezik mindenki az ehhez szükséges felkészültséggel. Vannak azonban olyan szakemberek, akiknek mindennapos munkája valamilyen módon az információk kezeléséhez, feldolgozásához, átadásához kapcsolódik. Világos, hogy feladatukat csak akkor tudják kielégítően elvégezni, ha alaposan ismerik tevékenységük általános tárgyát. Ehhez képest eléggé elszomorító, hogy az ismeretekkel dolgozó emberek nem kellőképpen tájékozottak az adat és az információ mibenlétét illetően.

Ennek a fejezetnek az a célja, hogy mélyrehatóan feltárja az adat illetve az információ lényegét és e szavak köznapi jelentésén túlmutató értelmet adjon nekik.

Az ***adat*** és az ***információ*** fogalmak az ***ismeret*** rokonai, amennyiben a mindennapos életben e három szót felcserélhető, egymást mintegy helyettesítő kifejezésekként - szinonimákként - használjuk. Sőt, az ismeretekkel foglalkozó tudományos világ elméleti publikációiban és az ismeretek gyakorlati kezelését támogató számítógépes rendszerek kézikönyveiben is sokszor alternatív fogalomként jelenik meg az adat és az információ. A hétköznapok kellemes pongyolasága nem akadályozhat meg bennünket abban, hogy megpróbáljuk szigorú következetességgel feltárni az adat és az információ lényegét, különbségét és viszonyát.

Az adatkezelés és -feldolgozás - szolgáltatás. Ezt a szolgálatot a szakemberek csak akkor tudják mindenki megelégedésére ellátni, ha új magatartást sajátítanak el. Ha az ismeretet nem a ***számítógép***, hanem az ***ember*** oldaláról nézik. Könyvünkben arra teszünk kísérletet, hogy bemutassuk ezt a ma még sokak számára szokatlan és újszerű szemléletmódot.

1.2 Az adat - első megközelítésben

Induljunk ki az adat szó eredeti latin jelentéséből. A „***datum***” szót a latinok főnévként és igenévként is használták. Előbbi formájában jelentése „adomány, ajándék” (vö. az előző pont első bekezdésével). Egyébként „adottat” jelent - és innen származik a mi „dátum” szavunk is. Az igenév többes száma „data”, vagyis „adottak”, az adott dolgok. Ezzel a formával sűrűn talál-

kozunk az angolszász szakmai irodalomban. Alapvető jelentőségét jól mutatja, hogy az IBM adatfeldolgozási szótárában [1] több mint 100 szószedet kezdődik a „data” szóval.

A hivatkozott szótár három meghatározást is ad magára az adat fogalomra (103. o.). A kitételek közül kettő korlátos, mert az adatot annak valamilyen megjelenítési formájához köti. Ezért csak az első definíciót idézzük:

> *„Az adat tények, fogalmak, eligazítások olyan formalizált reprezentációja (megjelenítése, tükörképe), amely alkalmas az emberi vagy az automatikus eszközök által történő kommunikációra, értelmezésre vagy feldolgozásra.”*

Mivel a fogalmak és az eligazítások (az eredetiben: instructions) maguk is tények, a meghatározás kicsit terjengős. Viszont éppen ezért jól utal arra, hogy nemcsak a konkrét dolgokat kell tényekként felfognunk. Jól kapcsolódik ide a hazai Idegen szavak szótárának [2] a megfogalmazása. E szerint a „data” „ismert tényeket, adatokat, dolgokat” jelent. Tehát az adat ismert tény és az ismert tény adat.

Az IBM meghatározása formai elemet is tartalmaz, amennyiben a reprezentációra, az ismeret megjelenítési módjára utal. Az ember képtelen megismerni azokat a tényeket, amelyeket nem a megfelelő, az általa is emészthető formában közölnek vele. Éppen ezért először azon kell elgondolkodnunk, hogy milyen módon jutunk ismereteinknek a birtokába.

1.3 Az ismeretszerzés momentumai

Az ismeretet mindig valamilyen közeg hordozza. Ezért az ismeret megszerzésének első momentuma az ***észlelés***, vagyis a szembesülés a közeggel. Itt van az újság: olvassuk el. Most mindegy, hogy közöl-e velünk valami lényegeset, vagy sem; ha nincs kéznél, nem tudjuk átböngészni. Ha az számunkra fontos, akkor érte kell mennünk. Mivel az ismeret tükrözi a tényeket, ha nincs például készletkimutatásunk, vagy az nem áll rendelkezésünkre amikor és ahol igényeljük, az csaknem annyit jelent, mintha készletünk sem lenne. Az ismerethordozó közeg aktuális jelenléte az észlelés alapvető feltétele.

Az ismeretekkel foglalkozó szakemberek máris levonhatják az első tanulságot: az ismeretközlés sikerének legelső feltétele az ismerethordozó közeg aktualitása. Mit ér az a levél, amely egy mai fontos eseményre hívna meg, de csak holnap kapom meg?

Az ismeretszerzés második momentuma az ***érzékelés***. Bármennyire is fájdalmas, vannak embertársaink, akik nem látnak és/vagy nem hallanak. A csökkentlátó számára az írás, a kép, a csökkenthalló részére a hang, a zaj nem érzékelhető. Az ismeret mindig valamilyen formát ölt. Az IBM fenti meghatározásának megfelelően olyat, amely alkalmas az emberi kommunikációra. (A gépet egyelőre mellőzzük.) Az érzékelés feltétele, hogy a forma megfeleljen az ismeretet fogadó ember fizikai képességeinek.

A fentiekből következik a második tanulság. Az ismeretközlésnek számos formája van. A szakembereknek törekedniük kell arra, hogy a formát a szó szoros értelmében az ismeret címzettjének a testére szabják. Az IBM további meghatározásait éppen azért nem idéztük, mivel azok az adatot a szemmel látható tényekhez kötötték. Holott ismeret számos más módon is átadható - és nem csak csökkentlátó embertestvéreinknek.

A harmadik momentum a ***felfogás***. Most ide „írunk” két jelet, kicsit elnagyoltan:

Az olvasók mindegyike észleli és érzékeli e jeleket, de csak páran fogják fel a jelentésüket. Hiába adnánk meg latin betűs változatukat - „nyippon” vagy „nyihon” - is. A megfejtést csak az tudja, aki ért japánul. Mert a jelek valójában betűk, az ábra valóban írás, és a „nyippon” illetve „nyihon” szó ennyit jelent: „Japán”.

Az ismeretekkel dolgozó szakemberek már láthatják a harmadik tanulságot is. A tényeket felfoghatóan, azaz érthető jelekkel kell ismertetni. A közlésekben mellőzni kell az általunk használt, de az ismeretet fogadó fél számára felfoghatatlan sajátos jeleket.

A negyedik ismeretszerzési momentum a ***megértés***. A felfogás és a megértés nagyon közel állnak egymáshoz, de nem teljesen azonos lényegek. Vegyük csak például a következő három jelsort: „Béla” - „aléB” - „bela”. Az első minden érett magyar ember számára érthető ismeretet közöl, de egyáltalán nem biztos, hogy mond valamit a latin betűket amúgy ismerő kínainak. A rejtvényeket kedvelők felfedezik, hogy a második jelsor ugyanazt a tartalmat hordozza, mint az első. A harmadik szóról adott környezetben néhány bennfentes tudja, hogy valójában azonos a „Béla” karaktersorral, csak bizonyos technikai körülmények miatt így ábrázolták az ismeretet. Viszont egy szláv népbe tartozó valaki a jelsort látva így szól magában: „Jé, itt valami fehérről van szó.” Az ismeretet felfogja, de annak valódi lényegét nem érti meg.

A felfogás és a megértés eltérése két dologra hívja fel a szakemberek figyelmét. Egyrészt az ***ízlésességre***. Megengedhetetlen és ízléstelen az ismeretek formailag pontatlan közlése. Ugyan ki szeretné becsületes „Álmos” nevét „almos” módon írtan olvasni? Másrészt arra, hogy a közlés legfontosabb hordozóközege az ***emberi nyelv***, amely népekhez, nemzetekhez kötött. Tehát a tényeket ízlésesen (formailag pontosan), az alkalmazott közeg (esetünkben: a nyelv) szabályait betartva kell tudatni az ismeret megcélzott fogadójával.

Az észlelhető, érzékelhető, felfogható, sőt megérthető közlés, - amelynek tartalmát meg is akarjuk ismerni, mert számunkra elfogadható, ízléses formában tálalták - már alkalmas az ***értelmezésre*** (a kommunikációra, a feldolgozásra - vö. az IBM definíciójával). Most már módunk van az adat meghatározására és arra, hogy elgondolkodjunk az értelmezés lényegén.

(Megjegyzés: Az ismeretszerzésnek van egy másik módja is. Új tudásra a már meglévő ismeretek kezelésével és feldolgozásával is szert tehetünk. Ezekről a lehetőségről majd könyvünk 8.5 pontjában emlékezünk meg.)

1.4 Az adat és az információ lényege illetve viszonya

A fentiekben leírtuk - azaz sokak számára észlelhetően, érzékelhetően, felfoghatóan és érthetően közöltük - e jelsort: „Béla”. Bizonyára akadtak, akik a karaktersorozat olvastán így szóltak magukban: „Jé, hiszen itt valakinek a keresztneve Béla.”

Pardon, álljunk csak meg! Mindeddig szó sem volt *keresztnévről*! Nem mi jelentettük ki, hogy a „Béla” keresztnév, hanem az olvasó vonta le ezt a következtetést. Az ismeretet fogadó a jelsor láttán önkéntelenül is elvégzett magában egy gondolkodási műveletet. Ha felfedezi ennek a tevékenységnek a lényegét, akkor máris közel jut az adat és az információ meglehetősen pontos elhatárolásához.

D 1/1 **Az adat értelmezhető (észlelhető, érzékelhető, felfogható és megérthető) ismeret.**

A „Béla” jelsor észlelhető (itt van), érzékelhető (írva van), felfogható (ismert jelekkel írtuk le), megérthető (a leírt jelsor tartalma nem idegen) ismeretet hordoz. Nem pusztán a ‘B’, az ‘é’, az ‘l’ és az ‘a’ betűk egyvelege. Meghatározásunk az általánosan megszokott definícióktól két lényegben különbözik:

- Nem hangsúlyozza a tényszerű lényeget. Az adat közölhet tényt, ám utalhat elképzelésre, tervre, szándékra, fikcióra is.
- Nem köti meg a megjelenítési formát. Az írás, a kép, a hang, a szag, az érintés, a mozdulat stb. egyaránt hordozhat ismeretet.

Most térjünk át a fenti ‘jé’-re! Ez a meglehetősen vulgáris (közönséges) kitétel igen jól tükrözi az információ lényegét. Egy szóval sem állítottuk, hogy a „Béla” jelsor keresztnevet takar. Az olvasó mégis ezt a következtetést vonta le - helyesen. Észlelte, érzékelte, felfogta, megértette a jelsort, amit ekként már értelmezni is tudott. Az ***értelmezés*** lényege, hogy az újonnan kapott közlést régi tudásunkhoz kötjük.

Tudjuk, hogy a „Béla” keresztnév. Ezért a „Béla” jelsor olvastán azonnal keresztnévre asszociálunk, azaz összekötjük az új értelmezhető ismeretet a már értelmezettel. Ha nem értenénk, hogy mit is jelent a „Béla” jelsor és mi a keresztnév, nem volna mit mihez kapcsolnunk. Nem születne bennünk - információ.

Az információ szokványos meghatározásai többnyire féloldalasak. Az Idegen szavak szótára [2] szerint az információ „felvilágosítás, tudósítás, tájékoztatás, hírközlés” illetve „értesülés, adat”. Ez a definíció összemossa az értelmezhető dolgot (adat) az értelmezés eredményével. Az IBM [1] megfogalmazása sokkal jobban megközelíti a lényeget. Az információ „Az a jelentés, amelyet az ember tulajdonít az adatnak ...”. Saját, látszólag sokkal egyszerűbb meghatározásunk közlése után kifejtjük, hogy a két definíció miért féloldalas. Szerintünk:

D 1/2 **Az információ új ismeretté értelmezett adat.**

Az információ nem azonos az adattal, hanem az adatnak az a jelentése, amit az ember annak tulajdonít az értelmezés által. Ennyiben az IBM meghatározása teljesen korrekt. Viszont az információ mindig hír, mindig valamilyen újdonság, új ismeret, felvilágosítás. Az IBM definíciója ezt a tényezőt elhanyagolja. Pedig az „újdonság” momentum nagyon is fontos. Ha ötvenszer elmondjuk ugyanazon személyeknek, hogy e könyv szerzője „Béla”, azzal semmi újat sem közlünk. Az olvasóban nem születik új ismeret - információ.

Az Idegen szavak szótára helyesen emeli ki az információ újszerűségének a vonását, viszont nem utal arra, hogy az ismeretért értelmezéssel kell megdolgoznunk. Így elrejti előlünk azt a tényt, hogy ugyanaz a „tudósítás” teljesen eltérő információkra vezethet attól függően, hogy a felvilágosításnak milyen jelentést tulajdonítunk. Lássuk csak a következő példamondatot:

1.1 példa

„Az X agrárvállalatnál a tengeri hozama az idén Z mázsa régi magyar holdanként.”

A közlést sokan nem értik, mert nem tudják, hogy a „tengeri” az a törökbúza, vagyis a kukorica. Mások fennakadnak a „régi magyar hold” kifejezésen. Mi az, hogy „magyar” és „régi”? Miért nem hektár? Ismét mások figyelmeztetnek, hogy a „mázsa” már nem bevett mértékegység. Mármost akik értik a mondatot és értelmezik is annak mondanivalóját, azok is eltérő következ-

tetésre jutnak. Van, aki a „Z” hozamot kevesli. Más szerint elegendő vagy jó. Ezek az emberek már további háttérismerethez is tudnak kapcsolni: a tengeri hozama annyi szokott lenni... Azon a tájon és olyan időjárás mellett... - teszik hozzá a még értőbbek. Milyen volt a tengeri fajtája? Milyen technológiával vetették, gondozták, aratták?

Az egyetlen ismeret akár száz kérdést is szülhet. A kérdésekre adott válaszok eredményezik a végső következtetést. Kérdezni viszont csak az tud, aki az eredeti közlést is információvá tudta értelmezni.

Amint látjuk, az adat egy, míg a belőle születő információ számos lehet. Az adat személytelen, tényt közlő ***objektív*** ismeret. Ezzel szemben az információ mindig személyes, az adatot fogadó ***szubjektumához*** kötődő lényeg. Ebből a kettősségből az adat kezelésére és -feldolgozására vállalkozó gyakorlati szakembereknek és az ismeretekkel elméletileg foglalkozó tudósoknak több következtetést kellene levonniuk.

1.5 Szintaktika, szemantika, pragmatika

Az alábbi elmélkedés tájékoztató jellegű. Nem tartozik szorosan az adat és az információ elvi lényegének a kifejtéséhez. Csupán a mai ismeretkezelési gyakorlatot és elméletet kívánja megvilágítani. Fantasztikus számítógépekkel és csodálatos ismeretkezelő szoftverekkel rendelkezünk. Akkor miért van az, hogy mégis állandó ismeretínségben szenvedünk? A következő bekezdések adnak magyarázatot erre a kérdésre.

A ***gyakorlati*** szakemberek általános hibája, hogy nem ismerik fel az adat és az információ elméleti különbségét. Nevezetesen azt, hogy az előbbi tényt közöl, tehát objektív valami, míg az utóbbi személyesen értelmezett dolog, vagyis szubjektív. Ezért nem alkalmazzák azt az elvet, hogy az ismeretet mindig úgy kell „tálalni”, hogy az a megcélzott felhasználó által könnyen értelmezhető legyen. Szelektív, emberre szabott adatszolgáltatásra, személyes jellegű ismeretközlésre lenne szükség. Ezzel a követelménnyel szemben a mai felhasználó gyakran kap elérhetőségében (észlelés), megjelenítésében (érzékelés), formájában (felfogás) és tartalmában (megértés) korlátos, elnagyolt, tömegszerű, nem-személyes és így általa információvá nehezen értelmezhető adathalmazokat. Olyan ismeret-egyvelegeket, amelyek sokkal inkább a szakember képességeit, semmint a felhasználó igényeit tükrözik.

A fentiek mentségére szoltál, hogy az ***elméleti*** szakemberek nagy része saját elefántcsonttornyába zárkózik és nem képes elfogadható fogódzót nyújtani a gyakorlat számára. A soksok elvi irányzat közül most csak egynek a példáját említjük. Az információelméletről (angolul: information theory) van szó, amely az ismereteknek pusztán csak a ***mennyiségi*** oldalával foglalkozik. Az információelmélet szerint a „Szereti-e János Lujzát?” és a „Szereti-e János a karalábét?” kérdésekre adott igen-nem válasz ugyanannyi „határozatlanságot szüntet meg”. A teoretikusoknak talán Jánost vagy főleg Lujzát kellene megkérdezniük arról, hogy valóban ugyanannyi határozatlanság szűnik-e meg bennük a két ismeret kapcsán. Ez a példa a teóriák általános hiányosságára akarja felhívni a figyelmet. Az információelmélet, az információalgebra, a hierarchikus-hálós-relációs adatmodellezési koncepciók mind-mind mennyiségorientáltak és a lehetségesnél sokkal kevesebbet törődnek az ismeretek ***minőségi*** oldalával. Kicsit pongyolán, de talán kifejezően fogalmazva: az információ praktikumával.

A jó elméleten alapuló gyakorlat és a gyakorlatias elmélet hiányát jól mutatja az a tény, hogy az ismeret három aspektusa közül az első kettővel jóval többet foglalkozunk, mint a harmadikkal.

Ugyanis minden adatnak három szemléletmódja van. Az ismereteket valamilyen formai módon jelenítjük meg. Jelsoraink értelmezhető tartalmakat hordoznak. Végül - amiről sokszor

megfeledkezünk - az ismeretközlésnek meghatározott célja van. Az alábbiakban röviden áttekintjük ezt a három nézőpontot.

A ***szintaktika*** az ismeret közlésére szolgáló jelek és/vagy jelcsoportok általános elrendezését jelenti, amely független az ismeret tartalmától, a megjelenítés módjától és használatától (ld. [1], 413. oldal). Tehát a szintaktika az érzékelés és a felfogás tartományába tartozik. Olyan jeleket használunk és azokat úgy rendezzük el, hogy az ismeretet fogadó magát a jelsort érzékelhesse és felfoghassa.

A ***szemantika*** a jeleknek és/vagy jelcsoportoknak a közlendő tartalomhoz való viszonyát jelenti, függetlenül a megjelenítés módjától (ld. [1] 375. oldal). A „Béla", „béla", „bela" stb. az általunk ismert szabályok szerint itt és most ugyanazt a tartalmat hordozza, írásmódtól függetlenül. Tehát a szemantika a megértés tartományába tartozik. Olyan jelsorokat állítunk össze, amelyek együttes tartalmát az ismeretet fogadó megértheti.

Sajnálatos módon a szintaktikát és a szemantikát a mai ismeretkezelési gyakorlatban sokszor félreértelmezik. A számítógépes programok utasításaira szűkítik le ezt a két lényeget. (Arról most ne is beszéljünk, hogy a két tényezőt gyakran összekeverik. Szintaktikainak mondják a szemantikai hibát - vagy éppen megfordítva - aszerint, hogy azt a fordító/szerkesztő milyen alapon fedezi fel.) Teszik ezt annak ellenére, hogy a szintaktika és szemantika az ismeretközlés általános aspektusai. Tehát éppen úgy, sőt talán még inkább vonatkoznak az adatra, mint a programutasításra.

Az adatoknak van egy harmadik nézete is, amelyről talán éppen az előbbi hibás szemlélet, a szintaktika és a szemantika programozásbeli túldimenzionálása miatt feledkezünk el. Az IBM szótára szerint a ***pragmatika*** a karaktereknek illetve jelsorozatoknak az azok értelmezéséhez és használatához való kapcsolata (ld. [1], 317. oldal). Ez a meghatározás helytálló, de kicsit üres, azaz némi magyarázatra szorul. A pragmatika lényegében a közölt ismeret gyakorlati használhatóságát jelenti. Magyarul azt, hogy partnerünknek olyan ismeretet adjunk át, amelyet ő könnyen és gyorsan azzá az információvá tud értelmezni, amelyre éppen szüksége van.

A szintaktika és szemantika objektív jellegű, különben aligha vizsgálhatnánk számítógéppel a programok formális megfelelőségét. Ezzel szemben a pragmatika a szubjektum szférájába tartozik. Nincs olyan számítógép, amely el tudná dönteni, hogy számunkra hasznos adatot közölt-e velünk. Ezért van szükség az emberre, a szakemberre, aki érti az adat és az információ különbségét, vagyis tisztában van a pragmatika lényegével.

A mai adatbázisok tervezői túlságosan el vannak foglalva az adatok fizikai tárolásával és megjelenítésével, vagyis a tisztán szintaktikai aspektusokkal. Már kevesebbet törődnek az egyedi adatok jelentésével és az adatok elemi összefüggéseivel, a szemantikával. Az ismeretek pragmatikai hasznosságát az adatbázis fogalmi szerkezete nagymértékben befolyásolja. Ez a terület pedig sokak számára végleg ismeretlen.

1.6 Miért hibádzik a kommunikáció?

A következő gondolatokat egy 1972-es halvány xeroxmásolatból idézzük, ezért a forrást nem tudjuk pontosan megadni. S. J. Harris maga is egy 1920-as írásra hivatkozik, amelyben W. James a kommunikációról elmélkedik. E rejtélyek dacára is érdemes megfontolni, amit mond.

„Ha Jancsi és Lajos beszélget, akkor a diskurzusnak valójában hat résztvevője van. Jancsi, ahogyan ő magát látja. Jancsi, ahogyan őt Lajos látja. Jancsi, amilyen a valóságban. Lajosra ugyanez a három szemlélet vonatkozik. A kommunikáció emiatt a többszörösség miatt veszik el. Jancsi valamit mond, amin Lajos teljesen mást ért. Sőt, két dolgot ért. Mert más az, amit Lajos

úgy fog fel, mint a valódi Lajos és más az, amit Lajos úgy ért meg, mint ahogyan magát elképzeli. Persze mielőtt Lajos bármit is hallana, Jancsi ugyanilyen zavarban van saját magával.

A feleség a férjének esik, miközben nem is vele, hanem valami egészen mással van baja. A disputa végén már egyikük sem érti, hogy egyáltalán miről is volt szó. A drága nej kijelent valamit úgy, ahogyan az saját magából fakad. Nem figyel arra, hogy közben mindez a férjére teljesen másként hat, mert ura máshogyan szemléli őt, mint ő saját magát. A hölgy azt sem veszi észre, hogy valójában nem azt mondja, amire gondol, mivel nem ismeri önmagát és kifejezési eszközei is szűkösek.

Hasonló a helyzet a gyerekekkel. A szülő beképzeli magának, hogy azt teszi, ami a kölyöknek jó. A picur mindezt másként látja: számára az ős arra való, hogy kiszolgálja az ő igényeit... Ha két emberi lény nem tud kommunikálni, akkor vajon mi a helyzet a szervezetek, a csoportok és az egész társadalom szintjén...?

> ***Az emberiség még nem jutott el arra a fokra, hogy megoldja az üzenetek kódolásának, küldésének, fogadásának és dekódolásának elsődleges problémáját.***"

Eddig az idézet. A hetven évvel ezelőtti gondolatok ma is érvényesek. Sohasem lehet jó informatikus abból, aki nem érti, hogy a küldött és a fogadott szavak nem azonosak. Aki nem látja az adat (az üzenet) és az információ (a hatás) különbségét. Aki öncélként a számítógéppel, a programmal, a tárolt adattal törődik, nem pedig azzal, hogy legyen mások által fogadható üzenete.

Más elfakult másolatból csak a gondolatcsírát kívánjuk elültetni. Egyes fensőbbséges emberek sajátja, hogy összetévesztik a ***kommunikálást*** és az ***informálást***. Az előbbi kétoldalú tevékenység, amelyben mindkét fél egyenlő szerepet játszik. Az utóbbi egyirányú folyamat, amelyben az ismeretküldő a meghatározó.

El kellene elmélkednünk azon, hogy vajon mi, mai számítástechnikusok, melyik gondolatkörben mozgunk... Csak közöljük a saját elképzeléseinket a felhasználóval, azaz informálunk, vagy nagyon is odafigyelünk arra, amit az alkalmazó mond és logikusan érvelve kommunikálunk. Persze ugyanezek a kérdések a felhasználói oldalon is felvetődnek. Mert a kedves alkalmazó lehetetlent akar, de azt viszont azonnal. Nem figyel a fejlesztő nehézségeire. Nem kommunikál, hanem informál.

1.7 Mit lehet megtudni?

A kilencvenes évek elején tudományos rangra emelték a ***káosz-elméletet***. A világot nem a vakvéletlen uralja, minden törvényszerű. Csakhogy az ember képtelen felismerni a szabályokat. Elkésve és utólagosan reagál a tényekre. Mire a törvényt felfedezi, már az nem is él. A hetven évvel ezelőtti, az előző pontban idézett kitétel - „Az emberiség még ..." - tökéletesen illeszkedik a káosz-elmélet megismerhetetlenségi axiómáihoz.

Mit jelent mindez az informatikusnak? Talán azt, hogy Madáchot feledve adjuk fel a reményt és a küzdelmet? Éppen ellenkezőleg. Csak azért is bele fogunk pislantani az ismeretközlés rejtelmesnek tartott titkaiba. A tényeket rögzítő adatokat adatbázisokban fogjuk tárolni és megpróbáljuk azokat onnan úgy előbogarászni, hogy embertársaink információkat nyerhessenek.

Ebben a könyvben nyomon fogjuk követni az adat útját az adatbázison keresztül az információig. Megpróbálunk megküzdeni a káosszal. Bizakodóan, de nem elbizakodottan. Már tudjuk, hogy a fizikai ősrobbanás szellemi detonációval is járt. Választ adunk sok kérdésre, de mindegyi-

ket nem fogjuk tudni megoldani és minden egyes megoldott probléma a felvetések tucatjait vonja majd maga után.

Tudásunk, erőnk, akaratunk és ezeknek megfelelően könyvünk is véges. Az adatbázis viszont végtelen. Ha ezt az olvasó megérti, akkor nem hiába harcolunk az informatikai káosz ellen.

Ellenőrző kérdések - 1

1/01 Információt (I) vagy adatot (A) takar-e a következő mondat: „Halassy Béla e mű szerzője”.

1/02 Információt (I) vagy adatot (A) rejt-e a következő leírt mondat: „A szerzőnek BMZ 873 rendszámú 1300-as Ladája van.”.

1/03 Információt (I) vagy adatot (A) hordoz-e a következő mondat, vagy netán egyiket sem (S): „Sári 6 éves.”.

1/04 Ez a jelsor „navigadnoj” észlelhető-e (1), érzékelhető-e (2), felfogható-e (3) és értelmezhető-e (4) az Ön számára? Adja össze válaszának megfelelően a pontokat.

2. AZ ISMERETKEZELÉS KÉT MÓDJA

2.1 Az ismeret hordozó közegei

Az ismeret átadásának és átvételének számos közvetítő eszköze van. Minden érzékszervünket az ismeretszerzés szolgálatába állíthatjuk. A kép, a hang, a mozdulat, a hőérzékelés, a zene mind-mind ismereteket közvetítenek felénk. Nem információkat, hanem adatokat. Ezeket az adatokat másokkal összegyúrva mi értelmezzük információvá. Például ugyanaz a mozdulat két különböző arckifejezéssel párosulva teljesen más hírtartalmú. A vállvonást kísérő mosoly „sajnos nem"-et jelent. A vállrándítással párosuló fintor viszont azt mondja, hogy „nem, és nem is érdekel".

Bármennyire is érdekesek az ismeret átadásának és átvételének ezek a módozatai, nekünk egy közegre kell szorítkoznunk. Ahhoz az eszközhöz fordulunk, amely az ember sajátja és amely az embert azzá teszi. Ez a közeg pedig nem más, mint a ***nyelv***, a természetes emberi nyelv. Ezen belül is alkalmazunk egy megkötést: csak az írásos ismeretekkel törődünk annak dacára, hogy közeleg a hangokat is megértő számítógépek korszaka. (Persze amit itt elmondunk, az jórészt a beszélt nyelvre is vonatkozik.)

Ennek a fejezetnek az a célja, hogy bemutassa a természetes emberi nyelv és a számítógépes ismeretkezelés összefüggéseit úgy, hogy párhuzamot von a kettő alkotóelemei között.

A fejezet során először az adat négy „dimenzióját" fogjuk ismertetni a mindennapos mondatok két alapvető alkotóeleméhez, az alanyhoz és az állítmányhoz kapcsoltan. Rá fogunk mutatni arra, hogy a számítógépes ismeretkezelés során is valójában alanyban és állítmányban gondolkozunk. Csak éppen a számítógépes ismerethordozó közegek természete miatt ezeket a tényezőket merev egységekbe kell kényszerítenünk. Így szembesülünk az „adat" fogalom egy szűkebb értelmével. Ez nem jelenti azt, hogy korábbi adatmeghatározásunkat ne tartanánk továbbra is érvényesnek.

Már a dimenziók megismerése során is érdemes felfigyelni az alany és az állítmány relativitására. Az egyik mondat állítmánya a másik alanya lehet. A mindennapos életben éppen ez a tény ad lehetőséget arra, hogy a különböző dolgokra vonatkozó ismereteinket összefűzzük. Ez a kapcsolódás az alapja az adatbázisnak is, amelynek lényegét a következő fejezetben fejtjük ki.

A fejezet második felében az ismeretkezelés kétféle módjáról lesz szó. A fentiekben a nyelvet jelöltük meg, mint bennünket érdeklő közeget. Azonban nem mindegy, hogy ***természetes*** emberi nyelvben fogalmazzuk-e meg az ismeretet, vagy erre a célra ***mesterséges*** „nyelvet" használunk. Ezért rá kell mutatnunk a szöveg- és az adatszerű ismeretkezelés közös alapjain túl azok eltérő lehetőségeire és konvergálásukra is.

2.2 Az ismeret négy dimenziója

Az emberi közlés legtermészetesebb egysége a (kijelentő) ***mondat***. A mondat ***szavakból*** áll. Ezek különböző szerepeket töltenek be a mondatban és ezeknek megfelelően ***mondatrészeket*** alkotnak. A teljes, de minimális kijelentő közlésben ***alany*** és ***állítmány*** szerepel. Ez idáig

közismert, elemi szintű tudás. Ássunk azonban kicsit mélyebbre, komolyabban megvizsgálva a mondatrészek szerepeit. Előbb nézzük meg, hogy miért is szükséges ez a vizsgálódás. Ehhez vegyük alapul a következő két példamondatot:

2.1 példa

„Rózsa kocsija fehér."
„Gabi kocsija Lada."

A fenti mondatokat bármilyen szövegszerkesztővel be lehet vinni a számítógépre. A szöveget el lehet tárolni; azt ki lehet írni a nyomtatóra; ki lehet jeleztetni a képernyőn; át lehet másolni stb. Kikereshetjük a mondat bármelyik szavát vagy jelsor-részletét. Azonban a szövegszerkesztővel tárolt ismeretek tartalmára nem tudunk okosan rákérdezni. Nem tudunk feltenni ilyen kérdéseket:

- „Milyen színű Rózsa kocsija?"
- „Kinek a kocsija fehér?"
- „Milye fehér Rózsának?"
- „Van-e Rózsának kocsija?"

A ***szövegszerkesztővel*** történő ismeretkezelés nem adhat mást, mint lényege. Ezek az eszközök nem arra készültek, hogy segítségükkel információvá értelmezhető adatokat kezeljünk. Ezért az ismeretkezelésnek más módját kell keresnünk. Olyant, amivel a fentiekhez hasonló kérdések is megválaszolhatók. Ehhez pedig ízekre kell szednünk a kijelentő mondatokat, hogy azután az azok részeivel kapcsolatos kérdéseinket jól tudjuk megfogalmazni.

A mondatok ***alanya*** mindig két részből tevődik össze. Ezek hivatalos neve „legközelebbi nem" (genus proximum) és „megkülönböztető jegy" (differentia specifica). Példáink esetében a legközelebbi nem a KOCSI, a megkülönböztető jegyet pedig a „Rózsa" és a „Gabi" jelentik. A most elmondottakat azonnal ki kell egészítenünk pár megjegyzéssel. A legközelebbi nemnek nem kell expliciten, tehát láthatóan szerepelnie a mondatban. Az lehet rejtett, implicit is. Pl. a „Rózsáé fehér." kijelentéshez is kötődik legközelebbi nem. Hiszen evidens, hogy ennek a mondatnak a megértéséhez kellett lennie egy korábbi közlésnek is, amelyben utaltunk arra, hogy miről (kiről) van szó. Ugyanez vonatkozik a megkülönböztető jegyre is: „A kocsija fehér." (Más jellegű probléma, hogy valóban behatárol-e a megkülönböztető jegy. Ha több Rózsa vagy Gabi is szóba jöhet, akkor a mondat alanyát pontosítanunk kell.)

A fenti kettősség a mondatok ***állítmányára*** is vonatkozik. Az első mondat állítmányában a legközelebbi nem a Szín, a megkülönböztető jegy pedig a „fehér". Itt is érvényesek a fentebb tett megjegyzések. A generikus állítmány (Szín illetve Típus) lehet implicit, szerepelhet más mondatban és behatárolónak kell lennie. Például nagyon is elképzelhető, hogy a „Lada" megkülönböztető jegy nem egészen pontos, mert többféle Lada típusú gépkocsi létezik.

Az ismeretkezelés nagy mestere, C. Bachman már a hatvanas években az adat három ***dimenziójának*** nevezte [3] az ismeret specifikus alanyát („Rózsa" kocsija), generikus (Szín) és specifikus („fehér") állítmányát. Abban a korban még nem léteztek adatbázisok. Egyszerre csak egyféle jelenséggel törődtek az ismeretkezelés során. Ezért nem utalhatott Bachman akkor a negyedik tényezőre, a generikus alanyra (KOCSI), amelyet az ő nyomdokán joggal nevezhetünk most már az ismeret első dimenziójának.

A négy dimenziót ábrán is szemléltethetjük. Ebben a könyvben a négy dimenzió jelölésére a következő egyezményes írásmódokat (konvenciókat) fogjuk használni: Nagybetűvel írjuk az általános alanyt (KOCSI), vastag dőltbetűvel a specifikusat (***Rózsa***). Nagybetűvel kezdjük az általános állítmány nevét (Szín) és kiemelés nélküli, kisbetűs a specifikus állítmány (fehér).

	ALANY	ÁLLÍTMÁNY
Generikus	KOCSI	Szín
Specifikus	***Rózsa***	fehér

2.1 ábra: Az adat négy dimenziója

A tényeket közlő kijelentések persze lehetnek bonyolultabbak is. Például: „Rózsa kocsija fehér és Lada típusú." vagy „Rózsa Lada kocsija fehér." vagy „Rózsa fehér kocsija Lada.". Az összetett mondatok is négy dimenzióra bonthatók a fentiek szerint, hiszen mindig visszavezethetők egyszerűekre. Például: „Rózsa kocsija fehér." és „Rózsa kocsija Lada típusú.". Az alany ismétlésével takarékoskodunk a közléseinkben is, meg az ismeretek dimenziókba való besorolásánál is. Az összetettség nem jelent új dimenziót, csak az állítmány tartalmának a többszörözését. Jól mutatja ezt a következő ábra:

	ALANY	ÁLLÍTMÁNY1	ÁLLÍTMÁNY2
Generikus	KOCSI	Szín	Típus
Specifikus	***Rózsa***	fehér	Lada

2.2 ábra: Összetett dimenziók

A természetes nyelvű mondatokban nemcsak alany és állítmány, hanem más mondatrészek is szerepelhetnek: tárgy, határozó, jelző. Ezekről azt kell tudni, hogy maguk is állítások. Ezért a mi felfogásunkban azokat is a generikus és specifikus állítmány két dimenziójába soroljuk. Az előző bekezdés elején lévő három mondat ezért nemcsak a mindennapos nyelvben hordoz azonos ismeretet, hanem a mi taglalásunk szerint is azonos képet ad. Mindhárom mondat a 2.2 ábra sematikus szerkezetére képezhető le.

Az ábrákat a mondatokkal összevetve több következtetést vonhatunk le. Egyrészt a kétféle módon közölt ismeret tényleges tartalma pontosan lefedi egymást. Másrészt a mondatok kerekebbek, szebbek - míg a sémák kattogósabbak, üresebbek. Harmadrészt viszont a sémák pontosabbak, mert - szemben a mondatokkal - expliciten tartalmazzák mind a négy dimenziót. Márpedig lényeges megjegyeznünk, hogy az ismeretközléshez mind a négy dimenzióra szükség van. Tegyünk csak ellenpróbát a következő mondatokkal:

- A rózsa ára 50 Ft. Krumplifajtáról vagy virágról van szó? Nincs generikus alany.
- A virág ára 50 Ft. Melyik virágé? A rózsáé, a liliomé? Nincs specifikus alany.
- A rózsa 50. Milye 50? Ennyi darab? Ennyi kiló? Nincs generikus állítmány.
- A rózsa színe. Mi van vele? Hiányzik a specifikus állítmány.

2.3 Történeti kitérő: A „szemantikai adatbázis"

A természetes nyelvi ismeretközlés jól beilleszkedik a sematikus négy dimenzióba. Ez a tény a 70-es években néhány kutatót arra ösztönzött, hogy megalkossák a ***szemantikai adatbázis*** koncepcióját [[4]]. Ebben az elképzelésben a számítógépen tárolt ismeret alapvető egysége a

mondat (angolul: sentence). Ez majdnem azonos a természetes nyelvi mondattal: kerek, emberi, érthető kijelentés. Lássunk néhány példát:

2.2 példa

„X. Rózsa 1957-ben született."
„Y. Gabi 1946-ban született."
„X. Rózsa Szegeden lakik."
„Y. Gabi Pécsett lakik."

A mondatokban nem kellett expliciten utalni a generikus alanyra (SZEMÉLY) és állítmányra (Születési dátum, Lakóhely). A dimenziókat úgy adták meg, hogy ***mondattípusokat*** kreáltak. Példánk esetében a SZEMÉLY SZÜLETETT és a SZEMÉLY LAKIK mondattípusokat (angolul: sentence type) fogalmazhatjuk meg. Persze akkor, ha a személyeknek mindig csak az aktuális lakcímére vagyunk kíváncsiak, összetett mondatot is alkothatunk: „X. Rózsa 1957-ben született és Szegeden lakik.".

Az elgondolás humánus célokat, emberibb ismeretkezelést kívánt szolgálni és egy ideig megvalósíthatónak látszott. Az ember ne sematikus formákkal, hanem kerek mondatokkal kommunikáljon a számítógéppel. Az ismeretkezelő programnak kell kihámoznia a mondat lényegét, ami a mondat elemeinek a mondaton belüli elhelyezése által úgymond adott. Vagyis a tartalom a mondat struktúrájából következik. Innen kapta ez a koncepció a „szemantikai" jelzőt.

Kétségtelen, hogy a viszonylag egyszerű angol nyelvi szerkezetek megfelelni látszottak az előzetes feltételezéseknek. Arról most ne beszéljünk, hogy a magyar nyelvben ez a megoldás egyelőre kizárt a ragok és a ragozások kavalkádja miatt. A kísérlet végül angolszász területen is csődöt mondott. Az angol nyelvben sem mindent ugyanúgy és ugyanolyan sorrendben írunk le. Külön gondot okozott a felhasználó számára annak megjegyzése, hogy az összetett mondatokat alkotó tagok miképpen követik egymást és annak emlékezetben tartása, hogy milyenek a különböző mondattípusok közötti kapcsolódások. Ezért a felhasználónak olyan „sémákat" kellett volna adni, amelyek már az ismeretkezelés alábbi adatszerű módjának a sajátjai.

Végeredményben az akkori próbálkozás meghiúsult. Ennek nem kis részben az is oka volt, hogy az ismeretrögzítők - pusztán fizikai okokból - kerek mondatok helyett jobban szeretnek szavakkal dolgozni. Ragok, töltőszavak és egyebek nélkül kevesebb jelet kell leírni. A kísérlet annyiban mégis figyelemre méltó, hogy a jövő század egyik lehetséges megoldását vetíti előre. Ha majd hanggal, újság-, könyv-, iratletapogatással úgy lehet ismereteket vinni a számítógépbe, hogy a programokkal a mondatok struktúráit és kapcsolatait is meg tudjuk állapítani, akkor ismét napirendre fog kerülni a szemantikai adatbázisok időlegesen elvetélt koncepciója.

2.4 Kétféle mondat

A természetes nyelvi vagy ahhoz közeli ismeretkezelés sok problémát vet fel. Egy személyre vonatkozóan hányféle mondattípust célszerű alkotni? Állítmányonként (született, lakik) egyet? Ez óriási redundanciával jár, hiszen mindig le kell írni például azt, hogy „X. Rózsa". Egyetlen egyet? Ebben az esetben a mondat kígyónyi hosszúságú lesz. További kérdés, hogy miként kapcsolódnak egymáshoz az azonos alanyra vonatkozó eltérő típusú mondatok illetve a különböző generikus alanyú mondatok? Például hogyan köthetők össze a személy és az általa birtokolt gépkocsi(k) ismeretei?

A szemantikai adatbázis koncepciója ezeket a görcsöket nem tudta feloldani. Nézzük meg, hogy mire jutunk egy másik fajta megközelítéssel. Vegyük csak alapul a következő pár mondatot (ld. 2.3 példa). A gondolatjáték kedvéért ezeket a kijelentéseket sematikus formában is megadjuk (ld. 2.3 ábra).

2.3 példa

„Gabi kocsija Lada típusú."
„Gabi kocsija ötszemélyes."
„Laci kocsija Lada típusú."
„Laci kocsija ötszemélyes."

	ALANY	ÁLLÍTMÁNY1	ÁLLÍTMÁNY2
Generikus	KOCSI	Típus	Férőhely
Specifikus	***Gabi***	Lada	5 személy
Specifikus	***Laci***	Lada	5 személy

2.3 ábra: A kocsik ismeretei sematikus formában

Ha a mondatokat páronként (a Gabi és a Laci alanyok szerint) vesszük, akkor azok nagyon értelmeseknek tűnnek. Ám a négy mondat együtt némi ellenérzést kelt bennünk, amiben megerősítve érezzük magunkat, ha a 2.3 ábra sematikus képét nézzük. „Maga mindent kétszer mond?" - vetődik fel bennünk a kérdés (Lada - 5 személy).

Tudomásul kell vennünk, hogy a természetes nyelvű közlések igen komoly ***redundanciát***, ismeretismétlést hordoznak magukban. (Ez az x+1-dik oka a szemantikai adatbázisok kudarcának.) Az ismeretátfedés gondjaival-bajaival most ne törődjünk. Csak annyit jegyezzünk meg, hogy az a mai számítógépes ismeretkezelésben nemkívánatos jelenség. Ezért nagyon sokan foglalkoznak a redundancia elkerülésének a módjaival. A megoldás pedig kézen fekszik. Nézzük csak meg a következő három kijelentést:

2.4 példa

„Gabi kocsija Lada típusú."
„Laci kocsija Lada típusú."
„A Lada típusú kocsik ötszemélyesek."

Mint látjuk, úgy takarítottunk meg egy mondatot, hogy közben ismeretet nem vesztettünk. Továbbra is tudjuk, hogy Gabi és Laci kocsija öt férőhelyes. Csak a megfelelő mondatokat kell összekötnünk. Eközben felfedezzük, hogy a mondatoknak kétféle fajtája van:

2.5 példa

„Gabi kocsija fehér."
„Gabi kocsija Lada típusú."

A két mondat látszólag teljesen azonos jellegű. Csakhogy már tudjuk, hogy az első kijelentés lezárt, befejezett egy gondolatfonalat. A második pedig éppen megnyitott egy újat és a 2.4 példa harmadik kitételéhez juttat bennünket.

Most kap igazán értelmet az ismeret első dimenziója, a generikus alany, mert már kétfélét is meg kell különböztetnünk (SZEMÉLY, KOCSITÍPUS). Újabb megoldásunk sematikus képét a

2.4 ábra mutatja. Az ábrán dőlt nemkövér szedet mutatja az egyik mondattípusból a másikba átvezető állítmányt.

A képet vegyes érzelmekkel fogadjuk. Az ALANY-ÁLLÍTMÁNY és a Generikus-Specifikus megjelölésektől eltekintve a 2.3 ábrán 9, ezen pedig 10 bejegyzés található. Mit nyertünk? Inkább veszítettünk, mert most is dadogunk a Típus kétszeri feltüntetésével. Ámde számoljunk utána, hogy négy Ladát birtokló személy esetében a 2.3 ábra szerinti séma 15, a 2.4 ábrának megfelelő már csak 14 bejegyzést tartalmazna. 10000 Lada-tulajdonosnál pedig az arány már 30003/20006 lenne stb.

Nem véletlenül játszunk itt a számokkal. A 2.3 ábra esetében egyetlen dologgal kell foglalkoznunk, míg a 2.4 ábránál két dolgot össze kell tudnunk kapcsolni. Világos, hogy ennek az összefüggésnek a megteremtése a számítógépen időt, helyet - pénzt - igényel. Akkor fizetődik ki, ha ellentételként nyerünk valamit. A példa esetében a nyereség világosan látszik, noha egyelőre a többszörös ismerettárolásnak a mellékköltségeiről még nem is beszéltünk.

	ALANY1	ÁLLÍTMÁNY1	ALANY2	ÁLLÍTMÁNY2
Generikus	KOCSI	Típus	TÍPUS	Férőhely
Specifikus	***Gabi***	*Lada*	***Lada***	5 személy
Specifikus	***Laci***	*Lada*		

2.4 ábra: A kocsiismeretek átalakított sémája

Most, hogy már tudjuk ***mit*** kellene a számítógépeken tárolni az ismeret dimenzióinak megfelelően, rátérhetünk a ***hogyan*** kérdésére.

2.5 Adatszerű ismeretkezelés

A számítógépeket a kezdetek kezdetén elsősorban könyvelési, nyilvántartási feladatok támogatására használták. Mégpedig úgy, hogy ezek a nyilvántartások egymástól teljesen függetlenül kerültek számítógépes tárolásra és feldolgozásra. A kimutatásokat irattartókban, magyarul dossziékban vezették, amelyek angol neve ***file*** (ejtsd: fájl). Ezt ma az előkelő adatállomány névvel illetjük. Egyféle dossziéban voltak az egyfajta jelenségre vonatkozó ismeretek. Az irattartóféle tehát pontosan megfelel a generikus alanynak, az ismeret első dimenziójának. Volt SZEMÉLY, ANYAG, ESZKÖZ, KOCSI, KOCSITÍPUS stb. fájl. Mint említettük, ezek között nem volt semmi összefüggés.

A fájlban vezették a feljegyzésre méltó tényeket. Feljegyzésre méltó az olimpiai csúcs, amit a latin „emlékezés” szóból származó angol „feljegyzés” - record - kifejezésen át a magyar ***rekord*** fogalom is tükröz. Persze a vállalatokban más dolgokat tartanak feljegyzésre méltóaknak. Ettől függetlenül rekordnak nevezték a valakiről, valamiről feljegyzett ismeretek együttesét. Így Kovács úrnak a SZEMÉLY fájlban volt egy személy mintájú, „Kovács” címkéjű rekordja. A rekord tehát megfelel a specifikus alanynak, az ismeret második dimenziójának.

A dossziék papírjain előre behatárolt, névvel nevezett dobozkák szerepeltek. Olyan „rovatok”, amelyekbe be kellett vezetni a megfelelő ismereteket. A rovatok a papír adott területét foglalták el, ezért hát elnevezték e területeket ***mezőknek*** (angolul: field). A mező két dolgot jelentett: egy rovatfajtát (pl. Név), azaz mezőtípust és egy konkrét tartalmat („Kovács”). Itt találkozunk az ismeret további két dimenziójával, a generikus és specifikus állítmánnyal.

A későbbiekben ezeket az ismereti egységeket így vagy úgy átkeresztelték. A rekord helyett például használták a ***mondat***, a mező helyett a ***szó*** kifejezést is. (Arra most nem térhetünk ki, hogy ezeket a megjelöléseket váltogatva alkalmazták az ismeret logikai tartalmára és az annak a számítógépes tárolására szolgáló belső fizikai egységekre.)

A lényeg - névtől függetlenül - ugyanaz maradt. Az ismereteket a rendes gyűjtők módjára előre megcímkézett „dobozkákba" helyezték el, pontosan az eddigi ábráinknak megfelelő módon. A KOCSI címkéjű dobozokba kerültek a tulajdonosok nevei, a „Gabi", a „Laci" és a többiek. A címke és a tartalom együttesét pedig már nem mezőnek, hanem ***adattételnek*** vagy egyszerűen csak ***adatnak*** kezdték hívni. Így kapott az adat az eredeti, tág jelentése (értelmezhető ismeret) mellett egy szűkebbet - „címkézett ismeretdoboz" - is.

A fentiek miatt nem tautológia, ha „adatszerű ismeretkezelés"-ről beszélünk. Ez azt jelenti, hogy az adatokat dobozokba rendezzük el. Ami kényelmetlen, picit inhumánus, sok gondot okoz. Viszont a címkézés miatt már választ tudunk adni azokra a kérdésekre, hogy „Van-e kocsija Rózsának?", „Milyen színű a kocsija?" stb. A dobozok ügyes összekapcsolásával azt is megismerjük, hogy hány férőhelyes Gabi Lada típusú kocsija. Mivel ugyanahhoz a rekordhoz elvileg tetszőleges számú adattételt rendelhetünk, az ismereteket nem kell ismételgetnünk. Nem kell többször tárolnunk azt, hogy „Gabi kocsija...".

2.6 Szövegszerű ismeretkezelés

Viszonylag hosszú idő telt el addig, amikor a számítógépek szövegtárolási képességét azok számítási képességeivel egyenrangúnak, sőt adott esetben fontosabbnak kezdték látni. Az adatszerű nyilvántartások mellett kezdtek megjelenni a természetes nyelvű szövegeket tároló és kezelő rendszerek.

A kétféle megoldást nagyon könnyű összehasonlítani. Csak képzeljünk egymás mellé pár „berovatozott", rubrikázott és kitöltött bizonylatot, meg pár egyszerű könyvoldalt vagy néhány folyamatosan teleírt A4-es papírt.

A szövegszerű ismeretkezelés során természetes emberi nyelvben megírt szavakból alkotott természetes mondatokat tárolunk. Ezek a szavak nincsenek megcímkézett, névvel ellátott dobozokhoz rendelve. A leírási sorrend kötetlen. Írhatjuk azt is, hogy „Rózsa kocsija fehér.", de azt is, hogy „Fehér a Rózsa kocsija.".

A szövegszerű ismeretkezelést nem szabad összetéveszteni a szövegszerkesztéssel. Az utóbbi esetében minden egyes szöveg külön állományt képez, tehát az állomány és a feljegyzés, a rekord lényegében ugyanaz. Ezzel szemben a szövegszerű ismeretkezelés a kezdetek kezdete óta megkülönbözteti az ismeret két dimenzióját, az általános és a specifikus alanyt. A szövegkezelővel ismereteket rögzíthetünk általában a személyekről, a kocsikról stb. és ezeken belül a konkrét X és Y személyről illetve a Z és Q kocsiról. Tehát ezek az állományok több „rekordból" állnak.

Az ismeretek természetes nyelvben történő bevitelénél kötetlen szórendet és eltérő ragozási formákat alkalmazhatunk. Ez a természetes rugalmasság kétségtelen előnyt jelent. Viszont az ismeretek visszakeresésénél éppen ez okozza a gondokat. A ragozások miatt sajátos szótárakat kell fenntartani és/vagy a szótöveket felismerő programokat kell alkalmazni, például a „fehéret", „fehérben", „fehérnek" stb. lényegi azonosságának a feltárására. Szinonimatárakra van szükség például a „piros" és a „vörös" rokonságának a felismerésére. A legnagyobb gondot pedig a homonimák jelentik. Mert honnan tudná a kezelő, hogy a mondat elején nagybetűvel írt „Rózsa" virágot, személyt vagy krumplifajtát jelent-e? A „Gabi" fiúnak, lánynak vagy az összerakós gyerekjátéknak a neve-e?

Az adatszerű ismeretkezelésben „egy pillanat” alatt megtalálhatjuk, hogy melyek a „fehér” kocsik, mert csak a Szín nevű doboz tartalmait kell végigvizsgálnunk. Ezzel szemben a szövegszerű ismeretkezelés kezdetén a tartalmak ilyen elkülönítésére nem volt mód. A kezelőnek minden egyes karaktersort össze kellett vetnie a „fehér” jelsorozattal. Ennek a teljes - és így természetesen lassú - átböngészésnek az elkerülése érdekében, a keresés felgyorsítására hamarosan „szókincstárat”, ***tezauruszt*** kezdtek alkalmazni. Ez a segédtár olyan index, amely őrzi a szövegelőfordulás tartalmát („fehér”) és állománybeli megjelenési helyeit. Ezzel támogatja az ismeret negyedik dimenzióját, a specifikus állítmányt.

Viszont továbbra is megoldatlan maradt a jelsorozat értelmezésének a problémája. A „Rózsa” szövegrészt már gyorsan ki tudtuk keresni, de nekünk kellett eldöntenünk a szöveg környezete alapján, hogy a jelsor személyre vagy virágra utal-e. Ezt a gondot a ***deszkriptor*** konstrukcióval szüntették meg. A „leíró” az ismeretrészeket kategorizáló ismérv. A „Rózsa” szó bevitelekor kijelenthetjük, hogy az most a virág vagy a keresztnév deszkriptorhoz kötődik-e. Ezzel a szövegszerű ismeretkezelés is kikerekedett az ismeretkezelés harmadik dimenziójával, a generikus állítmánnyal (pl. Keresztnév, Szín stb.).

Ettől még a szövegszerű ismeretkezelés nem csapott át az adatszerűbe. A terjedelmes szövegek bevitelekor nyilván nem minden szót fogunk deszkriptorokhoz rendelni, mert ez nemcsak fáradságos, hanem felesleges is lenne. Ezzel szemben az adatszerű ismeretkezelésben minden ismeretdarabot a megfelelő adatdobozhoz kell illesztenünk.

Mivel a kétféle kezelés egyaránt fontos a számunkra, érdemes pár gondolattöredék erejéig áttekinteni azok kapcsolódását és együttes jövőjét. Mielőtt ezt megtennénk, el kell mondanunk, hogy a szöveges ismereteket esetleg több állományt is tartalmazó ***könyvtárakban*** tárolják. Az ilyen könyvtárak együttesét nevezzük ***adatbanknak***.

2.7 A kétféle ismeretkezelés viszonya

Az adat- és a szövegszerű ismeretkezelésnek más a célja. Ezért balgaság lenne azt mondani, hogy az egyik jobb, fontosabb, szebb, mint a másik. Nyilvánvaló, hogy a könyveket nem adatmezőkre szabdalva fogjuk a számítógépben tárolni, mert az effajta műveket folyamatosan akarjuk olvasni. Ezzel szemben a vállalat főkönyvi adatait nem ódában írjuk le, hanem roppant szigorúan meghatározott rovatokhoz illesztjük.

Amióta az ismeret négy dimenzióját a szövegszerű kezelés is támogatja, a kétféle kezelési mód egyre inkább konvergál, azonos irányba tart. A fejlesztők felismerték, hogy az adatszerű ismeretkezelés során is szükségesek a kötetlen szövegek. A tárgyak formai leírásakor, a növények lelőhelyének a közlésekor, a személyek életrajzának a megadásakor kötetlen méretű és tetszőleges tartalmú „adatdobozokra” van szükség. A dBASE-féle kezelők „memo”-típusú adatmezői még csak gyatra próbálkozások ennek a feladatnak a megoldására, de már a helyes irányba mutatnak. Megfordítva: a szövegkezelőket egészen komoly adatszerű kezelési képességekkel bővítették ki. Hiszen nem elég magát az írásművet számítógépen tárolni, hanem ki kell tudni keresni a mű szerzőjét, címét, kiadóját, kiadási dátumát, formátumát stb. Ezeket az ismereteket pedig adatszerűen érdemes megfogalmazni és kezelni.

Valamikor majd megszűnik a kétféle ismeretkezelés mai különbsége. Egyelőre azonban a két megoldás párhuzamosan él és fejlődik, mert a köztük lévő legnagyobb különbség ma még feloldhatatlannak látszik.

Az ***ismeretek kombinálásának*** a lehetőségéről van szó. Tegyük fel, hogy arra vagyunk kíváncsiak, hogy mikor, hol, milyen káreseményekben vett részt Rózsa Lada típusú fehér kocsija! Itt négy eltérő jelenség (kocsi, kocsitípus, tulajdonos és káresemény) adatainak az együttes

kimutatásáról van szó. Adatszerű kezelés esetén a négy dologra vonatkozó ismereteket egy-egy - egymással összekapcsolható - állományban tároljuk. Ezen állományok adatmezőit tetszőleges kombinációkban előkereshetjük.

Ezzel szemben szövegkezelés esetében egyelőre nincs lehetőségünk a különböző állományok ismereteinek az összekapcsolására. Ha a kocsi, kocsitípus, tulajdonos és káresemény ismereteit együtt akarjuk látni, akkor azokat egy szövegben kell együttesen megjelenítenünk. Ez ismeret-többszörözéshez és/vagy nehézkes kezeléshez vezet. Mert például egy káreseményben több kocsi is részt vehet és így az esemény körülményeit minden kocsinál le kell írnunk. Ha viszont csak a kocsik adataira vagyunk kíváncsiak, a károkra nem, akkor vagy készítünk egy rövidebb, csakis a kocsikra vonatkozó szöveget (ami redundancia), vagy számunkra felesleges ismeretek oldalai között kell tallóznunk.

A szövegszerű ismeretkezelés nem ad módot az adatok átalakítására; az adatfeldolgozásra. A felhasználónak egyszerűen csak el kell olvasnia a tárolt szöveget. Ezután vagy nyer abból információt, vagy sem. Nem vezet különösebb út az adattól az információig. Ezért bár a szövegkezelő rendszerek érdemeit elismerjük, azokról nincs egyéb fontos mondanivalónk. A továbbiakban csakis az adatszerű ismeretkezeléssel foglalkozunk. Legelőször arra teszünk kísérletet, hogy tisztázzuk magának az adatbázisnak a fogalmát.

Ellenőrző kérdések - 2

2/01 A következő kijelentések közül melyik teljes (T) és melyikből hiányzik az általános (1) vagy specifikus (2) alany, az általános (3) vagy specifikus (4) állítmány?

- A kocsi típusa Lada.
- A szerző kocsijának a típusa Lada.
- Pityu másfél méter magas.
- Sára Katié kilencvenes.

2/02 A következő mondatok közül melyik sejtet viszonyt (V) újabb közlés felé és melyik tovább nem vezető leíró (L) ismeret?

- Pista kocsija ötszemélyes.
- A rendeléstételben a cikkszám 1234.
- Gábor szervezői tanfolyamra jár.
- A híd hossza 236 méter.

2/03 A következő állítások közül melyik igaz (I) és melyik hamis (H)?

- A könyveket szövegszerűen szerkesztjük.
- A könyvek ismereteit szövegszerűen kezeljük.
- A könyveket nem lehet adatszerűen kezelni.
- Magát a könyvszöveget nem célszerű adatszerűen kezelni.

2/04 Az alábbi kitételek közül melyik igaz (I), melyik hamis (H)?

- A szövegszerű kezelés nem alkalmas több állomány kezelésére.
- A szövegszerű kezelés nem alkalmas az ismeretkapcsolásra.
- A készletkimutatást adatszerűen érdemes kezelni.
- A múzeumi tárgyak leírására a kétféle kezelés együttese a célszerű.

3. AZ ADATBÁZIS LÉNYEGE

3.1 Állomány, adatbank, adatbázis

Nem csak a számítástechnikai szakma sajátja, hogy az alapfogalmakat nagyon lazán használja. Az azonban kétségtelen, hogy ***divatszakmáról*** van szó, aminek következménye, hogy a silány „árukat" is hivalkodó és hívogató köntösbe burkolják. Azaz másnak nevezik, mint ami valójában. Manapság nagy divat adatbázisról beszélni, ezért minden gyenge kis programot, amely adatokkal pepecsel, „adatbáziskezelő"-nek titulálnak, megtévesztve a felhasználót. Ideje ezért végre tisztázni, hogy mi is valójában az adatbázis.

Az ismeretkezelés egyik átfogó lehetőségének megfelelően ***adatállományokat*** hozunk létre. Az „állomány" megjelölés azt mutatja, hogy nem egyedi dologról, hanem egymással valamilyen szempontból összetartozó ismeretek együtteséről van szó. Az „adat" jelző az ismeretkezelés módjára utal, és elválasztja az ilyen állományokat a program-, rendszer-, szöveg-, kép- stb. fájloktól. Tehát ebben az esetben a szűkebb adatfogalmat jelöli. Léteznek olyan programozási nyelvek, programok, azokból összeállított rendszerek, amelyek az állományok *egymástól független* kezelésére használatosak. Ezeket az eszközöket ***állománykezelőknek*** (angolul: file management system) hívjuk. Minden ellenkező híresztelés dacára ide tartoznak a még ma is nagy népszerűségnek örvendő dBASE-szerű rendszerek, amelyek nincsenek sokkal jobban felkészítve több állomány együttes kezelésére, mint például a COBOL nyelv.

Az ismeretkezelés másik módjánál ***szövegállományokat*** hozunk létre. A jelző mutatja, hogy az alapismeretek kötetlen, természetes - és nem-adatszerű - megfogalmazásúak. Ez nem zárja ki, hogy a szövegekhez adatszerű ismeretek is kapcsolódjanak. Például megadjuk a könyv címét, szerzőjének a nevét, kiadásának dátumát stb. Az ilyen állománykezelő eszközöket ***szövegkezelőknek*** (angolul: text management system) illenék hívni. Nálunk - másutt nemigen - elterjedt a „szöveges adatbáziskezelő" kifejezés is. Ez a megjelölés pontatlan, amint azt a továbbiakból látni fogjuk. A szöveges ismereteket tároló rendszereket ***adatbankoknak*** hívjuk. Itt az „adat" szó nem adatszerűt jelent, hanem a fogalom tágabb értelmének felel meg.

Az ismeretkezelésnek van egy harmadik módja is: az adatbáziskezelés.

Ennek a fejezetnek az a célja, hogy kifejtse az adatbázis mibenlétét, bemutatva az egyéb ismeretszerkesztési megoldásokkal való rokonságát és az azoktól való eltérését.

A gyakorlatban használt adat(bázis)kezelőkben százféle eltérő módon nevezik az ismeretek alapvető egységeit. Ezek a megjelölések tárolási és kezelési módszerekhez kötöttek. Nem igazán fejezik ki az ismeretek közös lényegét. Ezért tájékoztatásunkat néhány fogalom tisztázásával kell kezdenünk. Ezeket másutt már széleskörűen használják, viszont nálunk még ismeretlenek vagy nem az eredeti, nem a valódi értelmükben alkalmazzák őket.

3.2 Egyedtípus, -előfordulás, -halmaz

Az ismeretnek négy dimenziója van a közlés generikus és specifikus alanyának illetve állítmányának megfelelően. Az alany írja le azt a valamit, amire a közlés vonatkozik. A latinban „entitas", az angolban az előbbi alapján „entity" a neve annak az önálló lényeggel bíró dolognak, amiről ismeretet közlünk. Az lehet bármi: személy, más élőlény, tárgy, fogalom, elképzelés stb. A latin és az angol szó magyar fordításának megfelelően

D 3/1 **Azt a dolgot-izét-valamit, amit ismeretekkel akarunk leírni, egyednek nevezzük.**

Korábbi példamondatunkban - „Rózsa kocsija fehér." - egy tárgyról volt szó, azt akarjuk ismerettel leírni, jellemezni. Tehát a Rózsa kocsija - egyed. Önálló lényeggel bíró valami. Ennek a jelenségnek kettős természete van. Egyrészt egy ***konkrétumról*** beszélünk; Rózsa kocsijáról és nem Gabi kocsijáról. Másrészt a konkrét dolgot ***absztrakt*** osztályba soroljuk. Kocsiról, nem pedig ruháról, szobáról, könyvről stb. van szó. Másik volt mondatunkban - „Gabi kocsija Lada." - ugyanebbe az absztrakt osztályba sorolt másik konkrétumot említettünk.

D 3/2 **Az ismeretekkel leírandó jelenségek absztrakt osztályait egyedtípusokkal tükrözzük. A típusba sorolt konkrét egyedeket egyedelőfordulásoknak hívjuk. Az egyedtípus minden időpontban az előfordulások adott halmazával rendelkezik.**

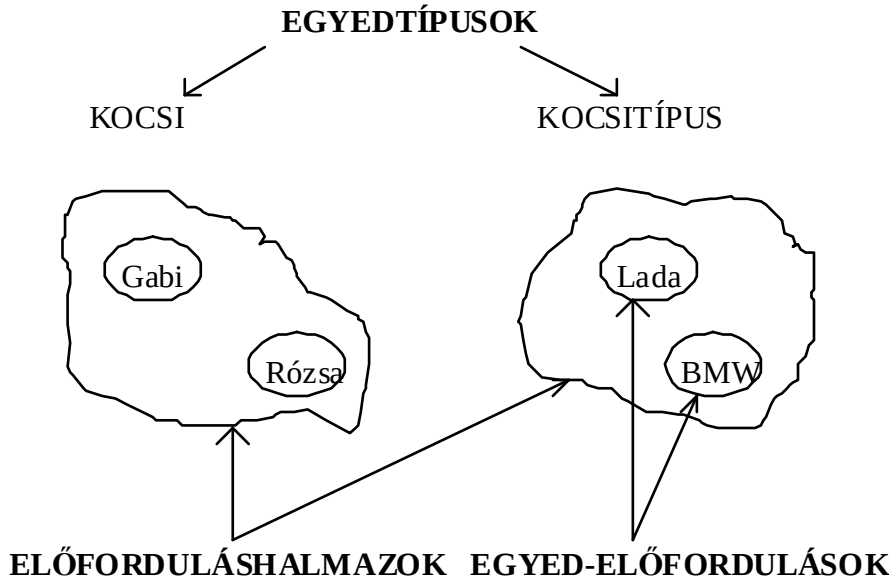
Első megközelítésben könnyűnek tűnik a dolgok elrendezése a meghatározás szerint. Mivel több kocsiról is szót ejtettünk, a KOCSI - egyedtípus (angolul: entity type). Ennek két egyedelőfordulása (angolul: entity occurrence vagy instance) a „Rózsa" és a „Gabi" kocsija. Az előfordulások együttese alkotja a „Rózsa, Gabi" egyedelőforduláshalmazt (angolul: entity occurrence set).

A valóságban a helyzet sokkal bonyolultabb. Miért nem a járműveket vagy még általánosabban a Rózsa és Gabi által birtokolt tárgyakat tekintettük egyedtípusnak? Vagy miért nem a személykocsikat, tehát egy specifikusabb kategóriát? Nyilván azért nem, mert az adott helyzetben céljainknak a KOCSI egyedtípus felelt meg a leginkább. Ezért sohasem szabad elfeledkezni arról, hogy az egyedelőfordulás nem azonos magával az egyeddel, hanem annak ismereti ***tükörképe***. Ugyanígy az absztrahált egyedtípus is tükörkép, a valóság informatikai mása. A tükrözés mindig a szubjektív céloknak felel meg. Ezért többféle módon hajtható végre, idővel változhat és - mivel a tükörkép nem azonos magával a jelenséggel - sokszor nem lesz tökéletes.

Azután meg kell fontolnunk a „konkrét" jelzőt is. Ez nem feltétlenül „egy darabot" jelent! Ha például csavarokat jellemzünk ismeretekkel, akkor nem fogjuk az összes tényleges M6-os csavart külön ismeretsorral leírni, hanem magát az M6-os lényeget tekintjük egy „konkrétumnak", szemben például az M8-as csavar lényeggel. Azt is meg kell érteni, hogy a „Lada típusú gépkocsi" is konkrétum, jóllehet X millió ilyen gépkocsi futkározik az utcákon. Hiszen maga a típus minden tényleges „fizikai" kocsitól függetlenül is ismeretekkel leírható, tükrözhető lényeg.

Amint látjuk, az egyedtípus és -előfordulás elvileg könnyen megfeleltethető az ismeret első két dimenziójának, az általános (típus) és a specifikus (előfordulás) alanynak (egyed). Ám maga a gyakorlati tükrözés - az, hogy mit tekintsünk típusnak és előfordulásnak egy adott helyzetben - nem mindig triviális feladat.

Az eddigi fogalmak összefüggéseit a 3.1 ábra szemlélteti. Megjegyezzük, hogy a továbbiakban az egyedtípusokat csupa nagybetűvel (KOCSI), az egyedelőfordulásokat dőlt kövér betűvel (***Rózsa*** kocsija) fogjuk jelölni.



3.1 ábra: Az egyedekkel kapcsolatos tényezők

3.3 Tulajdonságtípus, -érték, -értékhalmaz

Az ismeret másik két dimenziója az általános és specifikus állítmány. A „Rózsa kocsija fehér.” közlésben az általános állítmány (Szín) megjelölése implicit, a specifikusé („fehér”) explicit. Latinul „attributum”-nak, angolul ebből származóan „attribute”-nak hívják a dolgok általános, elválaszthatatlan sajátosságait, jellemzőit. Az „elválaszthatatlan” jelző hétköznapian szólva a következőket jelenti: Mindegy, hogy fehér-e vagy piros-e Rózsa kocsija, de „nincsen kocsi szín nélkül” és annak valamilyen konkrét megjelölése (fehér, piros stb.) nélkül.

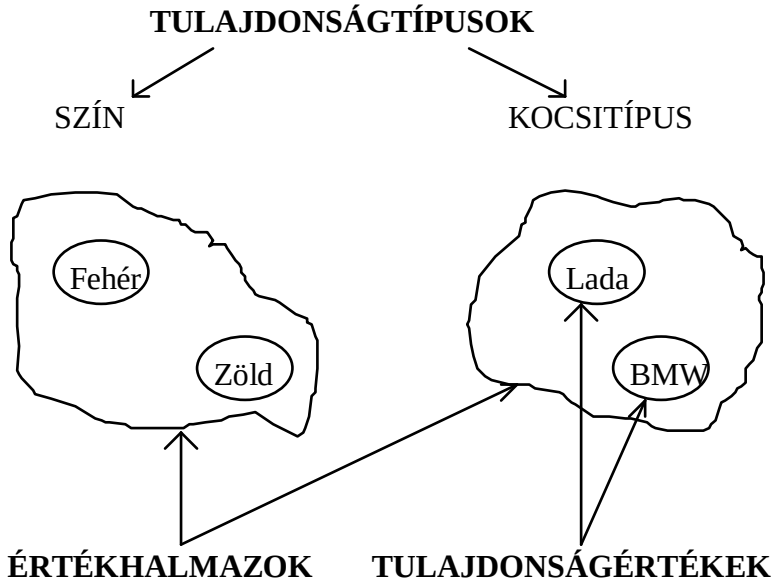
D 3/3 **Azt a dolgot-izét-valamit, amivel leírjuk a bennünket érdeklő jelenséget, tulajdonságnak nevezzük.**

Az egyedhez hasonlóan a tulajdonság is lehet absztrakt illetve konkrét. A Szín absztrakt fogalmat a hozzá képest konkrét „fehér”, „piros” stb. színekből vonatkoztattuk el. Ezért a tulajdonságnak is három aspektusára kell figyelnünk.

D 3/4 **A jelenségeket leíró ismeretek absztrakt osztályait tulajdonságtípusokkal tükrözzük. A típusba sorolt konkrét ismeretet tulajdonságelőfordulásnak vagy -értéknek hívjuk. A tulajdonságtípus minden időpontban az értékek meghatározott halmazával bír.**

A Szín tulajdonságtípus, a „fehér” annak értéke. Függetlenül attól, hogy éppen használatos-e - vonatkozik-e konkrét kocsira - a „fűzöld” érték is a Szín típus értékhalmazába tartozik. Ezzel szemben világos, hogy az „50 kiló” nem lehet ezen értékhalmaznak a része. Erre a bennfoglalásra illetve kizárásra vonatkozik a definíció „meghatározott” jelzője. Amin tehát nem azt kell érteni, hogy eleve felsoroljuk az összes lehetséges színt.

A tulajdonságokkal kapcsolatos fogalmak összefüggéseit a 3.2 ábra mutatja. A továbbiakban a tulajdonságtípusokra kezdő nagybetűvel (Szín), az értékekre idézőjelben („fehér”) hivatkozunk.



3.2 ábra: A tulajdonságokkal kapcsolatos tényezők

A világos elmélet érvényesítése sokszor nem könnyű gyakorlati feladat. „Gabi kocsija Lada.” típusú. De vajon a „Lada” érték valóban megfogható típus? Nem „Lada 2104”-et kellett volna mondanunk? Hogyan fejezzük ki, hogy az „1993. dec. 4.” keltezés, de ezen belül megrendelési- vagy szállítási-dátum? A tulajdonságérték nem azonos a leírt dolog tényleges sajátosságával: annak csak tükörképe. Ugyanígy a tulajdonságtípus is absztrakt informatikai tükörkép. A célon, a szubjektumon és a képességen múlik a tükrözés mikéntje és az sohasem lesz teljesen tökéletes.

3.4 Két relativitás

A fentiekben nem véletlenül hangsúlyoztuk, hogy az egyed- és a tulajdonságtípus tükörképek. Nem mondhatjuk tehát azt, hogy a kocsi egyedtípus, míg a szín tulajdonságtípus. Csak azt jelenthetjük ki, hogy amennyiben a kocsikat ismeretekkel akarjuk leírni, úgy azokat a KOCSI egyedtípus előfordulásaival tükrözzük. Ha a színt a kocsi fontos sajátosságának tartjuk, úgy azt a KOCSI egyedtípus Szín tulajdonságtípusának értékeiként reprezentáljuk. Mivel céljainkon múlik a tükrözés mikéntje, ugyanazt a lényeget kétféle módon is megfoghatjuk. Lássunk csak egy példát!

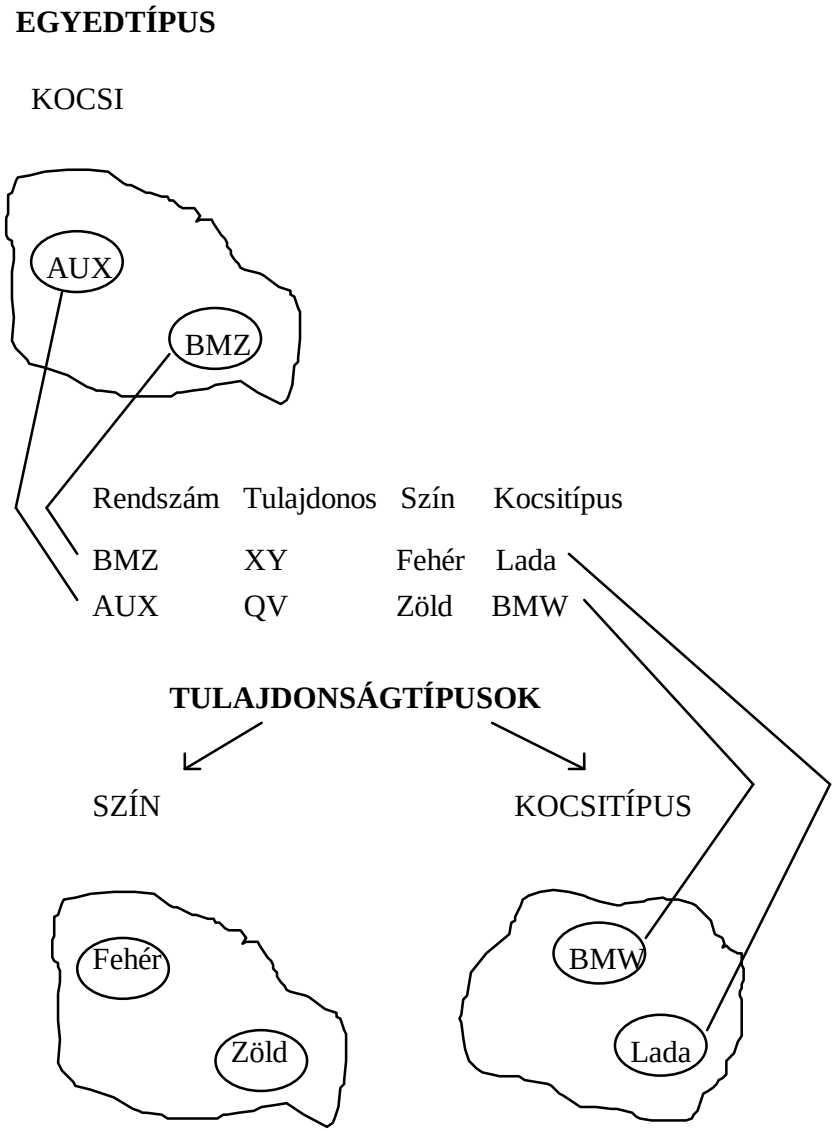
A kocsi vevője számára a szín valóban csak attribútum. A kocsi gyártója viszont elgondolkozik azon, hogy milyenre, miért és hogyan is fesse a kocsikat. Ehhez ismernie kell magának a színnek a jellegzetességeit. Ezért számára a szín „ismeretekkel leírandó dolog", tehát egyed is lesz. A gyártó adatbázisában szerepelni fog a saját ismeretekkel (pl. hullámhossz) leírt SZÍN egyedtípus és a Szín tulajdonságtípus, amely a KOCSI egyedtípushoz kapcsolódik. Volt más ilyen esetünk is. A Kocsitípus a KOCSI egyedtípus tulajdonságtípusa, de ugyanakkor lehet külön KOCSITÍPUS egyedtípusunk is, amely saját tulajdonságtípusokkal (pl. Férőhely) jellemezhető. Lásd a 3.1 és 3.2 ábra együttesét.

Másik érdekes tény, hogy az ember egyszerűen képtelen úgy beszélni a dolgokról, az egyedekről, hogy ne említse azok valamilyen tulajdonságát. Hiszen amikor azt mondjuk, hogy „Rózsa kocsija fehér.", a kocsit annak egy sajátosságával határoltuk be: megadtuk tulajdonosának a nevét. Ezen az alapon jogosan gondolkodhatnánk egy Tulajdonosnév nevű tulajdonságtípusban, amelynek értéke „Rózsa". (Másik mondatunkban ennek a tartalma „Gabi" volt.)

A „Rózsa" és a „Gabi" megjelöléseket persze lecserélhetnénk az „X rendszám" és az „Y rendszám" behatárolásokra is. Az „X rendszámú kocsi fehér." és az „Y rendszámú kocsi Lada." megfogalmazások mit sem változtatnak azon a tényen, hogy az egyedekre azok valamilyen tulajdonságtípusának (Rendszám) az értékeivel utalunk. Tehát vannak olyan tulajdonságok, amelyek az egyedeket mintegy képviselik, az ismeretközlésben azokat helyettesítik. Ezért le kell szögeznünk, hogy

Az egyed- és a tulajdonságtípus relatív (viszonylagos) fogalmak.

A kétféle viszonylagosság egymással összefügg. Mint majd kiderül, az első is a másodikon alapul. A 3.3 ábra mutatja az egyedekkel és tulajdonságokkal kapcsolatos fogalmaknak az összefüggéseit.

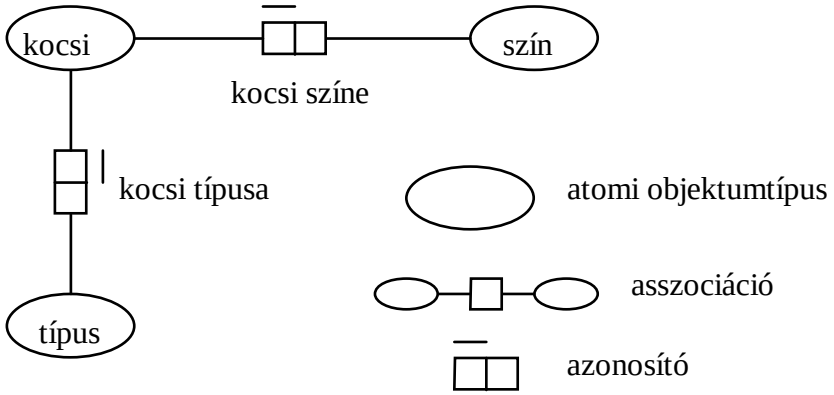


3.3 ábra: Az egyedek és a tulajdonságok összefüggései

3.5 A bináris relációk modellje [5]

Teljesen kézenfekvő, hogy a szín egyetlen lényeg. Ezért létezik olyan adatbáziskoncepció, amelyben nem alkalmazzák az egyed/tulajdonságtípus párost a jelenségek tükrözésére. A reprezentálandó dolgokat ***objektumoknak***, azok absztrakt osztályait objektumtípusoknak hívják. A Kocsi, a Szín, a Kocsitípus egy-egy objektumtípus. Ezek között párosával - innen a bináris, a másodfokú jelző - határoznak meg összefüggéseket, amelyeket ***asszociációknak*** neveznek. A két

objektumtípust és a közöttük lévő viszonyt sokszor atomi - tovább nem bontható - ***mondattípusnak*** is hívják. Ilyen mondattípus pl. a „kocsi - színe”. Az atomi mondat a mondattípus előfordulása. Pl. „X kocsi - fehér”. (Persze a „mondatok” a valóságban adatszerűen - és nem szövegszerűen - kerülnek tárolásra.) A fogalmak összefüggéseit a 3.4 ábra mutatja.



3.4 ábra: A bináris modell tényezői

Bár a koncepció mögötti elmélet teljesen világos, a bináris relációk modellje nem örvend nagy népszerűségnek a gyakorlatban. Ennek három oka van. Az egyik ok igen hétköznapi. Ha a kocsikat harmincféle sajátosság jellemzi, akkor harmincegy objektumunk és - feltételezve, hogy a sajátosságok egymáshoz nem kapcsolódnak - harminc asszociációnk van. Mindezeknek külön nevet kell adni, ami az adatbázis általános képét elbonyolítja. Ráadásul igen nehéz összeszedni egy gépkocsi sajátosságait, mert hiszen végig kell menni a harminc összefüggésen. Ez a megoldás hatalmas redundanciával jár, mert harmincszor kell elmondani azt, hogy „az X kocsi...”. (Az X kocsi fehér; az X kocsi Lada típusú; az X kocsi tulajdonosa Rózsa stb.)

Másrészt kiderül, hogy az egyed mégsem vehető egy kalap alá a tulajdonsággal. Mert mit is jelent „az X kocsi”? Nyilván a rendszámról van szó. Azt viszont nem jelenthetjük ki a bináris koncepcióban, hogy a rendszám nem azonos a kocsival, hanem annak csak reprezentánsa. Ezek szerint a bináris reláció elmélete nem tiszta, mert impliciten kénytelen elismerni az egyed (kocsi) és a tulajdonság (rendszám) különbségét, de expliciten nem támogatja ezt a megkülönböztetést.

Még világosabbakká válnak az elvi korlátok a harmadik probléma kapcsán. A bináris relációban mindig egy objektum reprezentánsa kapcsolódik annak valamelyik egyéb sajátosságával. Például: rendszám-szín, rendszám-tulajdonoskód stb. Azonban sokszor előfordul, hogy a valódi objektumot helyettesítő tényező maga is összetett. Vegyük csak alapul a rendeléstételek mindennapi példáját! A rendeléstétel eleve a rendelés és a cikk viszonyát fejezi ki. Ezért reprezentánsa már eredetileg is összetett (Rendelésszám és Cikkszám). Ha e pároshoz egy további objektumot (Mennyiség) kapcsolunk, akkor a reláció már nem bináris, hanem harmadfokú lesz. Ha nem illesztjük a mennyiséget a relációba, akkor pedig nem tudjuk megmondani, hogy a rendeléstételben mennyi árut igényeltek.

Ennek az ellentmondásnak a feloldására vezették be az ún. ***lebonthatatlan*** (angolul: irreducible) reláció konstrukcióját [6]. Az ilyen relációban a valós objektumot helyettesítő tétel összetett is lehet. A rendeléstétel reláció képe a következő: ({Rendelés, Cikk}, Mennyiség). Ez a hármas nem bontható meg, mert az eredményezett objektumpárosok egyike sem közvetít valódi ismeretet. Ezzel szemben a ({Rendelés, Cikk}, Mennyiség, Ár) négyes lebontható az előbbi és a ({Rendelés, Cikk}, Ár) reláció párosára ismeretveszteség nélkül.

A lebonthatatlan relációk koncepciója bölcsen hallgat arról, hogy a „jobboldali” objektum is lehet összetett. Ezt a problémát most ne feszegessük. Láthatjuk, hogy a redundancia gondját a bináris irányzat fejlettebb változata sem oldja meg és az sem képes különbséget tenni a valódi objektum és annak reprezentánsa között. A valódi objektum - pl. a rendeléstétel - ismeretei relációkra tördelődnek szét. Mindennek az az alapvető oka, hogy a bináris és a lebonthatatlan relációk elmélete nem tesz különbséget az egyed és a tulajdonság között. Ennek következtében nem ismeri fel, hogy a tulajdonságok nem egyenrangúak. Márpedig nem azok, amint azt a következő pontban kifejtjük.

3.6 Az azonosítás problémaköre

Ahhoz, hogy ismeretet tudjunk átadni, egyértelműen meg kell jelölnünk azt a dolgot, amiről szó van. Ez a behatárolás két momentumból áll. Először is utalnunk kell magára a jelenségcsoportra (egyedtípus). Tehát arra, hogy kocsikról és nem könyvekről beszélünk. Ezzel a tényezővel most többet nem foglalkozunk. Másodszor a jelenségcsoporton belül be kell határolnunk magát a konkrét jelenséget (egyedelőfordulás). Azaz meg kell mondanunk, hogy melyik kocsira vonatkozik az ismeret.

D 3/5 **Az egyedelőfordulás egyértelmű megjelölését azonosításnak hívjuk.**

Az azonosítás kérdésében ismét a nyelvtanhoz célszerű fordulni. Az azonosításnak a nyelvtanban két módja van. Az egyik az ún. ***nominális*** (azaz: név szerinti) meghatározás. Ez azt jelenti, hogy megadjuk a legközelebbi nemet és a megkülönböztető jegyet. Például: „Rendszáma X”. A „Rendszám” adja meg a hivatkozási alapul szolgáló általános jellemzőt, az „X” pedig annak specifikus értékét. Ha létezne több olyan gépkocsi is, amelynek a rendszáma azonos, akkor a Rendszám nem lehetne az egyértelmű kijelölés eszköze. Vegyük észre, hogy a nominális azonosítás egyáltalán nem jelenti azt, hogy a hivatkozásra szolgáló tényező valóban „névszerű” név. Példánk esetében a Rendszám is - „név”. Gyakran használjuk ezt a kifejezést: „A dolgokat a nevükön kell nevezni”. Ez a kitétel az adatmodellezésben a pontos hivatkozás követelményével azonos.

A jelenségekre vonatkozó ismeretek kezelésénél viszonylag ritkán van olyan szerencsénk, hogy a nominális azonosításhoz tudunk folyamodni. Nem gyakori, hogy a jelenségeknek van olyan egyetlen és ***természetes*** sajátossága, amely a jelenségcsoport minden egyes elemére eltérő tartalmú. Ilyenkor két lehetőségünk van. Az egyik az, hogy ***mesterséges*** „nevet” vezetünk be. Ilyenek a különböző azonosítókódok illetve -számok, mint például a szerződésszám, a vevőkód, a rendelésszám, a cikkszám stb. Azért, mert a szerződéseknek, vevőknek stb. nincs olyan egyetlen eredendően természetes jellemzője, amely minden konkrét szerződésre, vevőre stb. biztosan eltérő értéket venne fel.

Az „egyetlen” jelző sejteti velünk a hivatkozás másik lehetséges módját, amit a nyelvtanban ***deszkriptív*** (leíró) azonosításnak neveznek. A deszkriptív behatárolás során a jelenség nemegyedi tartalmú sajátosságait addig soroljuk fel, addig bővítjük újabb leírással, ameddig a felsorolt sajátosságok kombinációja egyértelműen nem azonosítja a kérdéses valamit. Egy példabeszélgetés: „Kinek a kocsija?” „A Rózsáé.” „A Kovács Rózsáé?” „Nem, a Szabó Rózsáé.” „Aki az X utcában lakik?”. „Igen, róla beszélek.” A keresztnevet a vezetéknévvel, majd a lakcímmel pontosították a felek. Végeredményben a tulajdonságtípusok értékét addig adták meg, ameddig a behatárolás meg nem történt.

A hétköznapi életben a deszkriptív meghatározás ad-hoc jellegű és sokszor redundáns. Alkalomszerű, mert párbeszédeinkben a „körülírt” valaminek nem mindig ugyanazokat a sajátosságait soroljuk fel és főleg nem ugyanolyan sorrendben. Ebből következik a redundancia is. Lehet, hogy ha a fenti párbeszéd során az „X utca” említése előbb történik, akkor már a vezetéknévre nem is kellett volna rákérdezni.

Az adatbázisokban az egyértelmű és minimális hivatkozás a célszerű. Ezért előre ki kell jelölnünk a jelenségeknek azt a természetes vagy mesterséges, egy vagy több tagból álló sajátosságát, amelynek tartalma a jelenségcsoport minden egyes tételére nézve egyedi.

D 3/6 **Az egyedtípus azon tulajdonságtípusát vagy tulajdonságtípuskombinációját, amely minden egyedelőfordulásra eltérő értéket vesz fel, vagyis azokkal kölcsönösen egyértelmű viszonyban áll, az egyedtípus azonosítójának nevezzük.**

Az egyértelmű megjelölés, hivatkozás kérdése az azonosító által elméletileg megoldódik. Gyakorlatilag az azonosítás sem könnyű feladat. Ha nincs természetes és egyetlen sajátosságot tükröző nominatív azonosítónk, akkor nehezen megjegyezhető mesterségeset kell bevezetnünk vagy hosszú közlést igénylő deszkriptív azonosítót kell alkalmaznunk. Gondoljunk csak a Személyiszám alkalmazásának a „betiltása” utáni helyzetre! Ez a félig természetes, félig mesterséges ismeret prímán bevált a személyek azonosítására. Ma vagy egy mesterséges Törzsszámot kell kitalálnunk, amelynek értékét senki sem tudja megjegyezni. Vagy a Név/ Születési-dátum/Születési-hely/Anyja-neve/Lakcím ismeretkombináció méretével, helyes és egyértelmű leírásával stb. kell megkínlódnunk.

3.7 Kapcsolattípus, -előfordulás, -(al)halmaz

„Gyuri kocsija kis Polski. Hány személyes?” Aki ismeri ezt a kocsitípust, az azonnal kivágja a feleletet: „Gyuri kocsija négyszemélyes.” A jártas olvasó a megadott közléshez (Gyuri kocsija kis Polski) hozzákapcsolta a saját háttértudását, amely így hangzik: „A kis Polski típusú kocsik négyszemélyesek.”

A fentiekből látszik, hogy közléseink nem teljesen egyenértékűek. Egyes ismeretátadások kapcsán az ismeretet fogadó továbbgondolkozik, az ismereteket összefűzi, konkatenálja. Más esetekben a közlés nem von maga után ilyen logikai váltást. A „Gyuri kocsija piros.” közlésre csak nagyon kevesen fognak elgondolkodni azon, hogy a piros szín mit is jelent.

Ismereteink általános és specifikus állítmányai kétféle jellegűek. Vannak köztük lezártak és vannak logikai ugrásokra késztetőek. A jelleg attól függ, hogy érdekeltek vagyunk-e a jelenségek közötti összefüggésekben vagy sem.

D 3/7 **A bennünket érdeklő jelenségviszonyokat kapcsolatoknak hívjuk.**

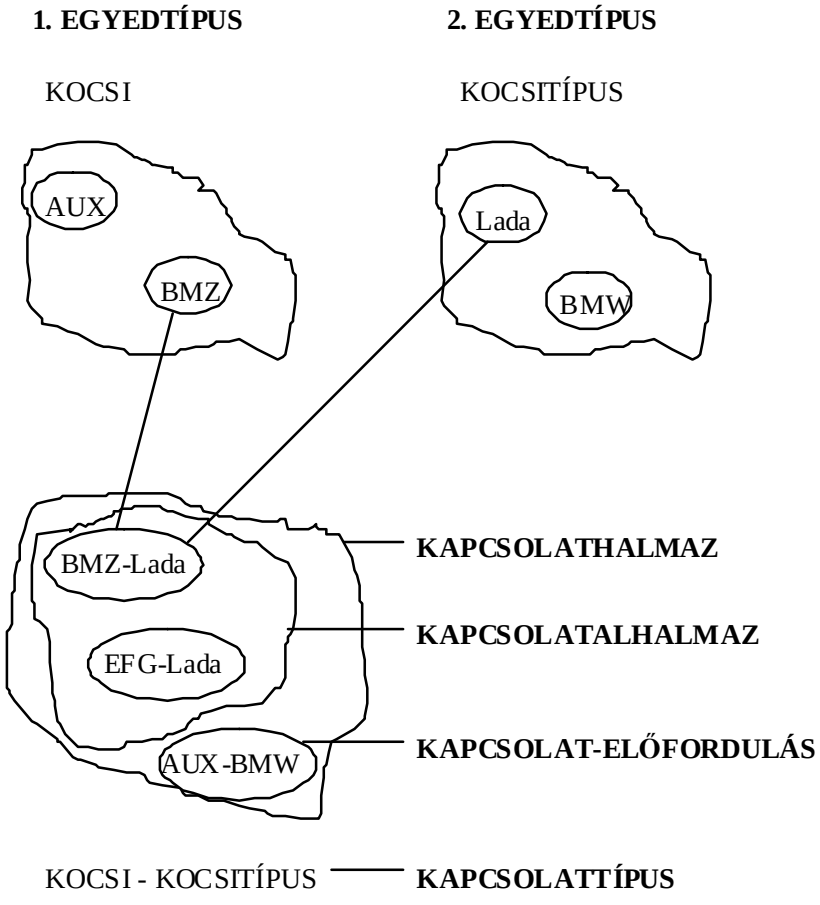
A kapcsolattal (angolul: relationship) összefüggésben is - az egyedhez és a tulajdonsághoz hasonlóan - megkülönböztetjük a konkrét illetve az absztrakt vetületet. Gyuri kocsija kis Polski, tehát négy személyes. Ezt az ismeretet a konkrét kocsi és kocsitípus közötti konkrét viszony (Gyuri kocsija - kis Polski) alapján nyertük. Általában tudjuk, hogy minden kocsi valamilyen kocsitípusba tartozik, és annak megfelelő a férőhelye. Ez a tudásunk absztrakt, a kocsitípus és a kocsi közötti generikus viszonyból következik. Mindebből levonhatjuk a tanulságot, hogy a kapcsolatok is több aspektussal rendelkeznek.

D 3/8 **A jelenségek közötti viszonyok absztrakt osztályait kapcsolattípusokkal tükrözzük. A típusba sorolt kapcsolatot annak előfordulásának nevezzük. A kapcsolattípus minden időpontban az előfordulások általános halmazával és egyedenkénti alhalmazaival rendelkezik.**

A kocsik mindig kocsitípusokba sorolhatók. Tehát általánosan létezik a KOCSITÍPUS - KOCSI kapcsolattípus. Gyuri kocsija kis Polski („Polski" - „Gyuri" kapcsolatelőfordulás). Másnak is lehet kis Polskija („Polski" - „Lackó" kapcsolatelőfordulás). Ezért egyedenként („Polski") beszélhetünk a kapcsolatelőfordulások („Gyuri" és „Lackó") alhalmazáról. Azért alhalmazról, mivel a kapcsolattípus másféle kocsikra is érvényes (pl. „Lada" - „Gabi" kapcsolatelőfordulás). Tehát a kapcsolatnak létezik az egyedektől („Polski" és „Lada") független teljes halmaza is.

Már szinte felesleges ismételnünk, hogy a kapcsolatelőfordulás is csak tükörkép. Nem azonos a dolgok tényleges viszonyával. Ezért a kapcsolattípusok gyakorlati meghatározása sokszor nem könnyű feladat. Ráadásul ez az adatbázistényező számos elméleti problémát is felvet, amelyekre majd csak a részletesebb tárgyalás során lesz módunk kitérni.

A kapcsolatokkal összefüggő ismereteket a 3.5 ábra foglalja össze. Ennek áttekintése után már kísérletet tehetünk az adatbázis lényegének a meghatározására.



3.5 ábra: A kapcsolatokkal összefüggő tényezők

3.8 Az adatbázis

Egyszerűen szólva az adatbázis a bennünket érdeklő különbözőféle jelenségek ismereteinek a szervezett együttese. A „különbözőféle” kifejezés a fentiekből már remélhetőleg érthető. Egyfajta jelenség a kocsi, másik annak tulajdonosa, harmadikféle a kocsitípus. Ismereteket közölhetünk magáról a kocsikról, azok tulajdonosairól és általánosan a kocsitípusokról. Bennünket igazán a „szervezett” jelző izgat. Mi annak a tartalma?

A későbbiek során több fejezetet fogunk szentelni ennek a kérdésnek. Jelenleg csak első, közelítő és átfogó választ adhatunk rá. Csak annyi a célunk, hogy az adatbázist elhatároljuk az adatbanktól és az állományok szervezetlen együttesétől (vö. 3.1 pont). Vegyük példaként a következő mondatot: „A Forint utcában lakó Rózsa kocsija fehér, Lada típusú, ötszemélyes.”. Most pedig ismételten gondolkodjunk el azon, hogy milyen módokon lehet ezt az ismeretet a számítógéppel által is kezelhető módon megfogalmazni.

Az első lehetőség az ***adatbanki***, a szövegszerű megformálás. A fenti mondatot minden további nélkül eltárolhatjuk gépünkön. „Szervezettségről” sok szó nincs, mert hiszen az ismeret közlésének a módjára semmilyen megkötés sem vonatkozik. A fenti mondatot a következő kitételekkel helyettesíthetnénk: „Rózsának van kocsija. A színe fehér. Típusa Lada, ezért ötszemélyes. Rózsa a Forint utcában lakik.”. Vagy: „Itt van egy kocsi, Lada. Színe fehér. Tulajdonosa Rózsa, akinek állandó lakhelye a Forint utca.”. Az ismeret tartalma mindkét esetben ugyanaz és még kitalálhatnánk több ezer, ugyanazt a tényt közlő megfogalmazást.

A második lehetőség az ***állományi***. A fenti példamondatot toldjuk meg egy újabbal: „A Fillér utcában lakó Gabi kocsija fehér, Lada típusú, ötszemélyes.” A két ismeretsor alapján nincs akadálya annak, hogy meghatározzuk a 3.6 ábra által mutatott állományképet:

Cím	**Név**	**Szín**	**Típus**	**Férőhely**
Forint utca	Rózsa	fehér	Lada	ötszemélyes
Fillér utca	Gabi	fehér	Lada	ötszemélyes

3.6 ábra: Hagyományos állományi szerkezet

Az adatszerűen megcímzett, „dobozokban” tárolt ismeretek immár egyértelműek és ezért a szövegszerű megfogalmazással szemben „szervezettek”-nek tűnnek. Egyetlen, kötött felépítésű állományt alkotnak, amelyből mindent megtudhatunk.

Azonban ezzel az elrendezettséggel nem lehetünk teljesen elégedettek. Mert miért kell kétszer leírnunk azt, hogy a Lada ötszemélyes? Gyanúnk tovább fokozódik, ha a 3.6 ábrát újabb ismeretekkel egészítjük ki azzal a természetes feltételezéssel élve, hogy egyvalakinek több kocsija is lehet. Az eredményt a 3.7 ábra mutatja:

Cím	**Név**	**Szín**	**Típus**	**Férőhely**
Forint utca	Rózsa	fehér	Lada	ötszemélyes
Fillér utca	Gabi	fehér	Lada	ötszemélyes
Forint utca	Rózsa	piros	Polski	négyszemélyes
Fillér utca	Gabi	zöld	Polski	négyszemélyes

3.7 ábra: Egy rosszul szerkesztett állomány

Ezzel a példával már több bajunk van. Már nemcsak azt ismételgetjük, hogy a Lada öt fős, hanem azt is, hogy a (kis) Polski négy férőhelyes. Ismételten megtudjuk, hogy hol lakik Rózsa illetve Gabi. A többszörös ismeretmegadás nemcsak a tárolót fogyasztja feleslegesen, hanem adatbeviteli, -módosítási és -törlési erőforráspazarlással is jár. (Gondoljuk meg például, hogy mi történik, ha Rózsa átköltözik a Pengő közbe.) A redundancia miatt nagy az inkonzisztencia veszélye is. Elég egyetlen helyen Babi-t írnunk a Gabi helyett, és máris hamisak az ismereteink. A hatékonyság pedig végleg csorbát szenved. Ha csak Rózsa lakóhelyére vagyunk kíváncsiak, miért kell átböngésznünk kocsijainak az adatait is? Ezzel a problémák sora még nem zárult le: a 3.7 ábrában ugyancsak nehezen találunk kedvünkre való azonosítót...

Az ismeretek redundanciája, azok esetleges ellentmondása, a felesleges adatok kezelésének a kényszere és az azonosítás nehézsége el kell, hogy gondolkodtasson bennünket. Az állományi típusú kezelés egyáltalán nem zárja ki a 3.7 ábrának megfelelő „szervezést", de evidens, hogy az nem felel meg a számunkra. Ismereteink elrendezésének a célszerű képét a 3.8 ábra mutatja:

TULAJDONOS		KOCSI			KOCSITÍPUS	
Cím	Név	Szín	Típus	Név	Típus	Férőhely
Forint utca	Rózsa	fehér	Lada	Rózsa	Lada	ötszemélyes
Fillér utca	Gabi	fehér	Lada	Gabi	Polski	négyszemélyes
		piros	Polski	Rózsa		
		zöld	Polski	Gabi		

Kapcsolatok: TULAJDONOS - KOCSI a Néven keresztül.
KOCSITÍPUS - KOCSI a Típuson át.

3.8 ábra: A kocsiismeretek ésszerű elrendezése

Első ránézésre ez a kép a korábbiaknál sokkal bonyolultabb. Hiszen egy helyett már három állományt mutat és adatféléket (Név, Típus) ismétel. Azonban gondolja meg az olvasó, hogy nem a tulajdonságtípusok, hanem a tulajdonságértékek többszörözése jelenti a valódi gondot. A 3.7 ábrával kapcsolatos tartalmi redundanciaproblémákat a 3.8 ábrának megfelelő szerkezet sorra kiküszöböli. Egyszeresen tároljuk a kocsik típusfüggő férőhelyeit és a tulajdonosok címeit. Ha csak az utóbbiakra vagyunk kíváncsiak, nem kell kezelnünk a kocsik adatait stb.

A 3.8 ábra az ismereteket a maguk helyére rendezi. Egyelőre még nem tökéletesen, mivel az egyedazonosítókat nem jelöltük ki. Azok meghatározásához sokkal többet kellene tudnunk az ismeretek szervezettségéről. Itt és most elégedjünk meg azzal, hogy a 3.8 ábra szerinti ismeretszerkesztés sokkal jobb, mint a 3.7 ábrán bemutatott. Próbáljuk meg összegezni, hogy ez minek köszönhető.

Az ismeretek megszervezése három dolgot jelent. Először meghatározzuk az egyed-típusokat (TULAJDONOS, KOCSI, KOCSITÍPUS). Azután megadjuk a tulajdonságtípusokat, azokat gondosan a megfelelő egyedtípus(ok)hoz rendelve. Tudatosan tesszük ezt, és nem csak úgy átabotában definiálunk negatív következményekkel járó adatsorokat, mint a 3.7 ábra esetében. Végül ismét figyelmesen ügyelünk arra, hogy az egyedtípusok között megfelelő átmeneteket biztosítsunk. Tehát megadjuk az azok közötti kapcsolattípusokat.

Az állománykezelők alkalmazása esetén is lehetőség van arra, hogy több állományt jelöljünk ki és az azokban tárolt ismereteket egy programmal kezelhessük. Azonban az állományközti összefüggéseket nem tudjuk előre meghatározni. Ezekre a viszonyokra a programjainkban nem tudunk hivatkozni. A programozó kénye-kedvére van bízva a kapcsolódó ismeretek megfelelő kezelése. Ezzel szemben az ***adatbázis*** módú adatkezelésben az állományok közötti általános logikai összefüggések ***absztrakt*** képét előre meghatározhatjuk, sőt, meg kell határoznunk. Ezek

után a ***konkrét*** ismereteket az előbbi képnek megfelelően lehet, sőt kell kezelnünk. Az adatbáziskezelő rendszer ezt nemcsak lehetővé teszi, hanem ki is kényszeríti. A programozó nem a saját ízlése szerint, hanem az előre megadott KOCSITÍPUS - KOCSI viszonynak megfelelően kell, hogy kezelje az ismeretek együttesét.

A fentiek szerint az adatbázisszerű ismeretkezelésben megkülönböztetjük az adatbázis általános elvi felépítését (absztrakt kép) és az abban őrzött aktuális ismereteket (konkrét tartalom). Az adatbázis általános struktúráját ***adatmodellnek*** nevezzük.

D 3/9 **Az adatmodell véges számú egyedtípusnak, azok egyenként is véges számú tulajdonság- és kapcsolattípusának a szervezett együttese.**

Ha a 3.8 ábra konkrét kocsikra vonatkozó adatait figyelmen kívül hagyjuk, akkor megkapjuk feladatunk adatmodelljét. A konkrét ismereteket csakis ennek az általános képnek megfelelően tárolhatjuk és kezelhetjük. Ezért eljutottunk az adatbázis lényegének a megfogalmazásához:

D 3/10 **Az adatbázis véges számú egyedelőfordulásnak, azok egyenként is véges számú tulajdonságértékének és kapcsolatelőfordulásának az adatmodell szerint szervezett együttese.**

3.9 Három további kérdés

Az adatbázis nem adatbank, mert nem szövegszerűen és nem külön könyvtárakban tárolja az ismereteket. Viszont nem is pusztán az adatállományok halmaza. Hiszen az állományok között tartalmi összefüggéseket, kapcsolatokat határozunk meg és az állományok adattételeit ezekhez igazítjuk. Nem véletlen, hogy a Szín tulajdonság csak a KOCSI egyedben szerepel, míg a Név tételt ezen kívül a TULAJDONOS egyedben is feltüntettük. Persze egyelőre az olvasó számára rejtély, hogy pontosan mit is fed a „nem véletlen” kitétel.

Annyi bizonyos, hogy az egyedtípusok tulajdonság- és kapcsolattípusait el kell rendezni.

De miként? Az egyed/tulajdonság/kapcsolat-hármas egzakt összefüggéseit majd a 6. és 7. fejezetben világítjuk meg. Előbb a szokatlan terminológiát kell megmagyaráznunk. A legtöbb felhasználó és fejlesztő eddig rekordokban, relációkban, szegmensekben, adatmezőkben, oszlopokban, adattételekben stb. gondolkodott. Miként viszonyulnak ezek az ismerethordozó egységek az adatmodell hármasához? A 4. fejezetben fogjuk kifejteni, hogy nem terminológiai zűrzavarról van szó. Az adatbázisokat a tükrözendő valóságtól a számítógépes megvalósításig többféle szemléleti ***szinten*** kell vizsgálnunk.

A harmadik kérdéskör arra vonatkozik, hogy „hány” az adatbázis. A mindennapi életben az X felhasználó az Y felhasználónak a saját adatbázisával dicsekszik - és megfordítva. Majd némi beszélgetés után felfedezik, hogy a számítógépen kezelt adataiknak vannak közös ismeretelemei is. Ezért a gondolkodó emberekben megfogalmazódik a sejtés: lehet, hogy nekünk valójában csak egy adatbázisunk van? Az 5. fejezetben mutatjuk be, hogy az adatbázisnak különböző szemléleti ***vetületeit*** kell és lehet megkülönböztetni.

Ellenőrző kérdések - 3

3/01 Melyik mondat rejt egyedtípust (T), -előfordulást (E) és -halmazt (H)?

- A Fillér utcai lakók nem szeretik a karalábét.
- Kíváncsiak vagyunk a Fillér utcai lakók ismereteire.
- Gabi a Fillér utcában lakik.

3/02 Nyilvántartást kell vezetnünk a betegellátás finanszírozásáról. Adatokat kell tárolnunk az orvosokról, a betegekről és a társadalmi biztosítottakról, mint lehetséges betegekről. Ön hány (n) egyedtípust lát e feladat mögött?

3/03 Az egészségügyi dolgozók pótlékot kapnak. Ön miként tükrözné a pótlékra vonatkozó ismereteket? Egyedtípussal (E), tulajdonságtípussal (T) vagy mindkettővel (M)?

3/04 Lát-e különbséget a RENDELÉS egyedtípus Rendelésdátum és Vevőkód tulajdonságtípusa között? Fejtse ki a gondolatait.

3/05 Ön szerint melyik kitétel igaz (I) és melyik hamis (H):

- Az adatbázis a számítógépen tárolt adatok együttese.
- Hanglemezeim adatait egy dBASE-állományban tartom. Ez adatbázis.
- Van egy HANGLEMEZ és egy KIADÓ állományom. A kettő között a közös Kiadókód adattal tartok kapcsolatot. Ez adatbázis.
- Az adatbázis a felhasználó számára fontos ismeretek halmaza.

4. AZ ADATBÁZIS HÁROM SZINTJE

4.1 Egy alapvető probléma: az eszközorientáltság

A számítástechnikai fejlesztők és rajtuk át a felhasználók is ***eszközorientáltak***. Ez azt jelenti, hogy az éppen rendelkezésükre álló rendszer által támogatott adatszerkezetekben gondolkodnak. Ha ma dBASE-szerű kezelővel dolgoznak, akkor adataik struktúráját annak képességeinek megfelelően határozzák meg. Ha holnap megamini kategóriájú relációs rendszert alkalmaznak, akkor a szerkezeteket annak lehetőségei szerint alakítják át.

Természetesen a leendő rendszerekben a kezelő által hatékonyan támogatott struktúrákhoz célszerű igazodni. Adataink szerkezetét a kezelőrendszer képességeinek megfelelően kell kialakítani. Azonban ez a mondat is mutatja, hogy valamilyen már meglévő lényeg (adataink) és egy másik - esetleg változó - tényező (a kezelő) közötti megfeleltetésről van szó. Ebben a viszonyban nyilván a cél az elsődleges és nem az eszköz. Ezzel szemben a mai fejlesztők az adatstruktúrákban eleve összekeverik a felhasználó számára fontos dolgokat (adattartalom) és a saját szempontjukból lényegeseket (a kezelő korlátai). Így az adatszerkezetben menthetetlenül összecementezik a „mit” és a „hogyant”, a kezelendő adatokat a kezelési móddal.

Az eszközorientált szemlélet következményei károsak. Az említett beágyazás miatt a kezelő lecserélésekor újra kell elemezni a „mit” körét is, holott elegendő lenne csak a „hogyan” gondjával törődni. Az egybeépítés miatt igen nehezen oldható meg az adatok átvitele az egyik kezelő alól a másikba. Ezért annyira nehézkes ma az inhomogén rendszerek kiépítése, vagyis ugyanazon adatoknak eltérő gépeken és eltérő szoftverekkel való közös, egyidejű kezeltetése is.

Az eszközorientáltság szükségszerűen ***eszközfüggőséghez*** vezet. A rendszer hosszabb használata után a fejlesztő és a felhasználó az adatokat nem a maguk természetes módján, hanem a kezelő által korlátozott módon fogja látni. Szentül meg lesz győződve arról, hogy az adatbázisok úgy néznek ki, ahogyan ő azt megszokta. Pedig ez tévedés: az adatbázisok már ma is nagyon sokfélék és állandóan fejlődnek. Az eszközorientált környezet képtelen nyomon követni ezt a változást, mert nem nyitott rá.

Ennek a fejezetnek az a célja, hogy az eszközfüggetlenség jegyében ismertesse az adatbázisok három szerkezeti szintjét.

Az adatbáziskutatók már régen rádöbbentek arra, hogy az adatbázisok szerkezetét több síkon kell vizsgálni. Ezért megkülönböztetik az adatbázis fogalmi, logikai és fizikai aspektusát. A „szintek” kifejezés az adatbázis vertikális tagolására utal. Nem azt jelenti, hogy a konkrét adatbázis valamilyen módon szét van darabolva - bár az sem kizárt. Hanem azt, hogy az adatbázisok szerkezetét általában három lépésben kell meghatározni úgy, hogy mindegyikben más-más tényezőkre ügyelünk.

4.2 Az adatok két aspektusa

A következő példa alapján igen könnyen meg lehet érteni az adatok két szemléleti tényezőjét:

4.1 példa

„Béla születési dátuma 1946. május 14."
„Béla születési dátuma 1946.05.14."
„Béla születési dátuma 05/14/1946."
„Béla születési dátuma 1946.05.14."
„Béla 1946. május 14-én született."
„Meli születési dátuma 1973. március 16."

Az első öt mondat ugyanazt az ismeretet tudatja velünk. A kitételek mégsem azonosak. Nem nehéz rájönnünk arra, hogy az ismeretközlésben meg kell különböztetnünk a mondanivalót, a ***tartalmat*** és a kifejezésmódot, a ***formát***. A hatodik mondat tartalma az első ötétől eltérő, formája viszont az első mondatéval azonos.

Felesleges bárkit is győzködni arról, hogy a két tényező közül melyik a fontosabb. Evidens, hogy a tartalom a lényegesebb. Ami nem azt jelenti, hogy a forma közömbös; aki nem ismeri az amerikai keltezést, az a harmadik kitétel kapcsán zavarba jöhet. A példával nem a fontosságot akarjuk hangsúlyozni. Sokkal inkább a tartalom és a forma ***számszerű*** viszonyára kívánjuk felhívni a figyelmet. Nevezetesen arra, hogy ugyanaz a tartalom nagyon sokféle formában fejezhető ki.

Most vonatkoztassuk a legutóbbi kitételt a számítógépes és nem-számítógépes adatkezelésre! A Születési-dátum nevű adattételnek egy adott személyre nyilván csak egy értéke lehet (feltételezve az általunk használt naptárat). Ugyanakkor az értéket százféle módon jeleníthetjük meg papíron és számítógépes adattárainkon.

Következtetés: az adatbázisok tekintetében is különbséget kell tennünk az adatok tartalma és formája között. Meg kell határoznunk az adatbázis tartalmi szerkezetét, majd - tekintettel az adatkezelő képességeire és hatékonyságára - ki kell jelölnünk az adatok formai jellemzőit.

Az előző mondatban a „majd" szó rettentően fontos. A mai gyakorlatban az adatbázis-tervezők a tartalmat és a formát egy lépésben, együttesen határozzák meg. Vagyis a tartalmat a formába betonozzák. Ez azért hibás felfogás, mert ugyanazt az adatot papírra is leírjuk, továbbá többféle formában is kezeltethetjük a számítógépen. Ha az adatbázisról nem készítünk külön - formától mentes - tartalmi tervet, akkor nem látjuk át a változások következményeit. A formai változás ugyanis nem hat ki a tartalomra, de a tartalmi szintű módosítás mindig formai átalakítással is jár.

Azt már világosan látjuk, hogy az adatbázisok tartalmi és formai aspektusát meg kell különböztetnünk. Joggal merül fel a kérdés, hogy amennyiben két lényegről van szó - tartalom és forma -, úgy miként beszélhetünk az adatbázisok három szemléleti szintjéről? Erre a felvetésre a következő két pontban találjuk meg a választ.

4.3 Az adatbázis kétféle tartalma

A számítógépes adatkezelésben jártas fejlesztők és felhasználók jó része csak egyféle kezelőrendszert alkalmazott és ma még kevesen találkoztak a valódi adatbáziskezeléssel. Ezért nem figyelhettek fel arra, hogy az adatbázis tartalmával két síkon kell foglalkoznunk. Mivel a két szemléletmód egymás nélkül nehezen lenne megérthető, ebben a pontban együtt ismertetjük az adatbázisok fogalmi és logikai szerkezeti aspektusát.

Elsőként az adatbáziskezelők adatszerkezeti korlátait érdemes ecsetelnünk, mivel ez a könnyebben érthető tényező. Vegyük például a következő pár mondatot és az azoknak megfelelő adatszerű megfogalmazásokat:

4.2 példa

„Béla születési dátuma 1946.05.14."
„A BMZ 873 rendszámú kocsi Béla tulajdona."
„A ZBM 378 rendszámú kocsi Béla tulajdona."

TULAJDONOS

Személy	**Születési-dátum**
Béla	1946. 05. 14.

KOCSI

Rendszám	**Személy**
BMZ 873	Béla
ZBM 378	Béla

4.1 ábra: Kocsik és tulajdonosaik

Egyes adatkezelő rendszerekben lehetőség van ún. ***adatcsoportok*** megadására. A Születési-dátum adattételről kijelenthető, hogy az valójában év, hó és nap együttese. A rendszer az évet, a hónapot és a napot együttesen és külön-külön is tudja kezelni. A relációs rendszerekben erre általában nincs mód. Vagy egyetlen Születési-dátum adatot alkalmazunk, vagy az Év/Hó/Nap hármasát vesszük fel. Tehát döntenünk kell az azonos tartalmak kétféle szerkesztése ügyében. Ennek a döntésnek nincs köze a formához: a három tétel sorrendje tetszőleges lehet és mindegyiken belül eltérő írásmódokat is alkalmazhatunk. A tartalom kétfélesége a lényeg. Az, hogy egy vagy három adatmezőt alkalmazunk-e.

Vannak adatkezelők, amelyek megkövetelik a különböző jelenségek (pl. személyek és kocsik) adatainak a szétválasztását, ha a jelenségek közötti összefüggés többszörös (egy személynek több kocsija lehet). Mások minden további nélkül támogatják az ún. ***ismétlődő adatokat***. Az olyan megoldásokat, amikor egy adatsoron belül valamelyik ismeretféle tartalma többszörös. Az ilyen kezelőkben felvehető a következő összetett adatsor: (Béla, 1946.05.14, {BMZ 873, ZBM 378}). Ebben a Rendszám értéke többszörös.

Az adatkezelők a többféle jelenség közötti ***kereszthivatkozás*** szempontjából is megoszlanak. Egyesek megkövetelik, hogy a jelenségek (pl. személy és kocsi) összefüggéseit közös adattétel (esetünkben: Személy) fejezze ki. Mások ehhez nem ragaszkodnak: a megfelelő személy és kocsi összetartozásának a tényét valamilyen belső technikai mutatóval rögzítik. A 4.2 példa tartalma - adatszerűen - a következő módokon is megfogalmazható:

TULAJDONOS

Személy	**Év**	**Hó**	**Nap**
Béla	1946	május	14

KOCSI

Rendszám	**Személy**
BMZ 873	Béla
ZBM 378	Béla

TULAJDONOS

Személy	**Születési-dátum**	**Rendszámok**
Béla	1946. 05. 14.	{BMZ 873, ZBM 378}

TULAJDONOS KOCSI

Személy	**Születési-dátum**	**Rendszám**
Béla	1946. 05. 14.	BMZ 873
		ZBM 378

4.2 ábra: A kapcsolódó ismeretek különféle kezelési módja

Az utolsó táblázatban gondként merül fel a tulajdonosok és kocsik adatainak az összekötése. Vannak olyan kezelőrendszerek, amelyek használatakor ez az összekapcsolás nem tartalmi, hanem fizikai alapon (pl. pointerrel, azaz mutatóval) történik. A technikai megvalósítással mi most nem kívánunk törődni, mert éppen azt akarjuk elválasztani a tartalomtól.

Éppen ezért elmondhatjuk, hogy mindeddig nem beszéltünk az adatoknak a tényleges formai megjelenítéséről. Például arról, hogy a Születési-dátum értékeit milyen jelsorozatok képviselik a számítógépen. Nem hoztuk szóba azt sem, hogy Béla adatai milyen eszközön, milyen sorrendben, az adattételek milyen elrendezésében stb. kerülnek tárolásra.

Viszont ezzel szemben bemutattuk, hogy ugyanazt az ismerettartalmat a kezelőrendszerek saját képességeiknek és korlátaiknak megfelelően különböző módokon rendezhetjük el. Az egyik oldalon van egy közös tartalmú ismeretünk. Már mindenki ismeri Béla születési dátumát és kocsijainak a rendszámát. A másik oldalon (a két ábra együttesében) négy olyan elrendezésünk is van, amely - az adathordozón való tényleges elhelyezéstől függetlenül - ugyanazt a tartalmat mégiscsak másképpen tálalja a részünkre.

Ha kezelőt váltunk és áttérünk a 4.2 ábra egyik megoldásáról a másikra, akkor bizony igen jelentősen meg kell változtatnunk az adatbázis tartalmi szerkezetét, jóllehet a „mondanivalónk” - a végső tartalom - ugyanaz marad. Tehát az adatbázisok tartalmát valóban két síkon kell szemlélnünk. Ezt a kitételt még jobban igazolja a következő pont.

4.4 Az adatbázis fogalmi és logikai szerkezete

Ebben a könyvben nincs módunk a különböző adatkezelő rendszerek által támogatott tartalmi adatszerkezeti egységek hallatlan különbségeinek a bemutatására. Erről a témáról külön értekezést kellene írni. Az egyik rendszerben rekordnak, a másikban relációnak, a harmadikban szegmensnek (stb.) nevezik azt az adatsort, amely adott esetben tartalmilag pontosan ugyanazt az ismeretet hordozza. (A különböző elvű adatkezelőkre nézve a [7] ad jó eligazítást.) Amint láttuk, a különbségek nemcsak terminológiai jellegűek, hanem bizonyos helyzetekben valóban eltérő tartalmi adatbázisszerkesztési megoldásokat is eredményeznek (vö. 4.2 ábra).

Ebben a pontban azt fogjuk kimutatni, hogy az egyazon elvi alapú adatkezelők használata esetén is különböző módokon határozhatjuk meg az azonos tartalmú adatbázisok szerkezetét. A 4.3 ábra szemlélteti a mondanivalónkat. (N.B.: Az ilyen ábrákban a „...” azt mutatja, hogy olyan egyéb ismereteket is tartalmaz az adatbázis, amelyek bennünket mondanivalónk szempontjából nem érdekelnek.)

Személy	**...**	**Páratlan**	**Páros**
Kovács	...	X forint	Y forint
Szabó	...	Z forint	Q forint

4.3 ábra: Egy „ravasz” megoldás

A példa a bérelszámolásról szól. Az egyedileg pontosan behatárolható személyek (Kovács, Szabó) béreit kell kezelni. Ehhez mindig csak az aktuális és az azt befolyásoló - előző havi - bér ismeretére van szükség. A 4.3 ábra megoldásának a tervezője ezért így gondolkozott: „Januárban a Páratlan adattételbe viszem a bér adatát, amelyet márciusig megőrzök. Februárban felülírom a Páros adattétel tartalmát az akkori bérrel. Márciusban - az akkor már felesleges januári - Páratlan tétel tartalmat aktualizálom. És így tovább...”

A megoldás „zseniális”. Takarékos és egyelőre működik. Először akkor robban le, amikor a 13. havi kifizetések kerülnek szóba. Mert ez a bér a páratlan hónapba tartozik, csakúgy, mint az azt követő januári. Másodszor akkor, amikor egyszerre kettőnél több havi bért kívánnak kezelni. Ez ki van zárva, mert hiszen a Páratlan/Páros adattételek tartalmai havonként felülíródnak. Tehát az újabb igények felmerülése esetén az adatbázis szerkezetét át kell alakítani; az adatokat át kell tölteni az új adatbázisba; az összes addigi kezelőprogramot újra kell írni stb.

Adatbázisaink a fenti ötletszerű megoldások miatt használhatatlanok. A tervezők olyan adatszerkezeteket találnak ki, amelyek az általuk az adott időpontban fontosnak vélt ismeretigényeket jól kiszolgálják, de ezek ***változása*** esetén összeomlanak. A kérdés az, hogy miképpen lehet elkerülni az ilyen változások kellemetlen és „előre nem látható” hatásait?

A válasz egyszerű. Csak az adatbázisok kétféle tartalmi szerkezetével kellene tisztában lenni. A 4.4 ábra mutatja a megfelelő megoldást.

Ha a 4.4 ábra dőltbetűs sorától eltekintünk, akkor annak ismerettartalma a 4.3 ábráéval egyezik meg. Mivel a 4.4 ábra példájában a személy nevét - vagy annak valódi azonosítóját - a két állományban meg kell ismételni az összekapcsolhatóság kedvéért, az első pillanatban úgy tűnik, hogy a 4.3 ábra megoldása kedvezőbb. Azonban a dőltbetűs sor világosan mutatja a két adatbázisterv közötti valódi különbséget. A BÉR állományban a személyekhez akárhány havi bérismeretet köthetünk anélkül, hogy a korábbi ismereteket a felülírás miatt elveszítenénk, mint a 4.3 ábra példájában. Tehát a 4.4 ábra szerinti szerkezet stabilabb.

SZEMÉLY

Személy	**...**
Kovács	...
Szabó	...

BÉR

Személy	**Hónap**	**Forint**
Kovács	1	X
Kovács	2	Y
Szabó	1	Z
Szabó	2	Q
Kovács	*3*	*W*

4.4 ábra: Az okos megoldás

Az összevetés kapcsán két gondolat vetődhet fel bennünk. Az első egy kérdés ez: Miképpen találjuk meg a változástűrő adatbázisszerkezeteket általában? (Azért általában, mert példánk esetében a megoldás kézenfekvő volt, viszont más helyzetekben több fejtörésre lehet szükség.) A második egy megállapítás: A fentiek szerint a 4.4 ábra megoldása a jobb. A két felvetés közül először a kérdést fogjuk körüljárni.

Azok a tervezők, akik az állományszerű adatkezeléshez szoktak, egy alkalmazási feladat felmerülésekor azonnal „rekordképekben” gondolkodnak. Ezzel szemben azok, akik már találkoztak valódi adatbázissal, először eltöprengenek a jelenségek valós összefüggésein. Pontosabban szólva

a jelenségeket tükröző ***fogalmak*** mögötti viszonyokon. Valahogyan így: „Lássuk csak! Vannak személyek (egyik fogalom), akik béreket (másik fogalom) kapnak. Egy személynek 0,1 ... N bére lehet az idő során. 0 akkor, amikor még nem kap fizetést, mert új belépő. Később havonta kap bért. Tehát a személy és a bér fogalom két eltérő lényeg. Ezért két állományt fogok tervezni."

A 4.4 ábra adatbázisának a felépítése egy az egyben megfelel a valós jelenségeket tükröző fogalmak közötti viszonyoknak. Ezért általánosan is megfogalmazhatjuk a definíciót:

D 4/1 **Fogalminak nevezzük a jelenségeket, azok sajátosságait és viszonyait a valóságnak megfelelően és természetes fogalmakban tükröző adatszerkezetet.**

Most pedig térjünk át a minősítés problémájára! Hajlamosak vagyunk túl hamar kijelenteni, hogy a 4.4 ábra terve „jobb" a 4.3 ábra által mutatott megoldásnál. Az ilyen elhamarkodott ítélet helyett inkább azon kellene elgondolkoznunk, hogy az adatbázisszerkezeteket két különböző szinten kell értékelnünk. Ha kicsi a tárkapacitásunk, nincs megfelelő adatkezelőnk, hosszú ideig valóban csak kéthavi bér nyilvántartására van szükségünk, akkor a 4.3 ábra megoldása jó. Nem „jobb", mint a 4.4 ábráé, hanem másképpen jó.

Ennek a „másképpen jó" kitételnek a megértéséhez vegyük sorra, hogy milyen indokok alapján térhetünk el a fogalmi adatszerkezettől. Az első tényező, amit számításba kell vennünk ***technikai***, és erről éppen az előző pontban beszéltünk. Sajnálatos tény, hogy az adatkezelő rendszerek képességei nem tökéletesek. Ezért sokszor az adatkezelő adatstrukturálási korlátai miatt nincs lehetőségünk arra, hogy a valóságot híven tükröző fogalmi adatszerkezetnek megfelelően alakítsuk ki az adatbázis felépítését. A fogalmi szerkezetet „el kell rontanunk", megalkudva a kezelő aktuális képességeivel.

A második ok ***hozzáférési***. A személy egyetlen lényeg, ezért a fogalmi adatszerkezet szintjén a közvetlenül a személyre vonatkozó ismeretek egy elvi állományt alkotnak. Azonban a személyeknek lehetnek titkos, nem mindenki által kezelhető adatai is. Igaz ugyan, hogy a modern adatkezelők hozzáférési zárakkal tudják védeni az adatmezőket, de a legbiztonságosabb megoldás nyilvánvalóan az érzékeny adatok elválasztása a többiektől. Ezért a tervező a személyre vonatkozó ismereteket esetleg több állományban fogja elhelyezni. Vagyis az egyetlen fogalmi adatsorból több valóságosat hoz létre. Például a személyek ismereteit „általánosakra" és „titkosakra" darabolja.

A harmadik ok ***hatékonysági***. Ennek a tényezőnek a megértéséhez tekintsük át a 4.5 és 4.6 ábrákon mutatott ismerethalmazok különbségeit:

KOCSIK

Rsz	Szin	Tipnev	Fero	Tukod	Tunev
X	Fehér	Lada	5	1	Rózsa
Y	Zöld	Lada	5	2	Gabi
Z	Piros	Polski	4	3	Lajos
Q	Piros	BMW	5	1	Rózsa

4.5 ábra: Hagyományos szerkezet

TULAJDONOS

Tulajdonoskód	Tulajdonosnév
1	Rózsa
2	Gabi
3	Lajos

KOCSI

Rendszám	Szín	*Típusnév*	*Tulajdonoskód*
X	Fehér	*Lada*	*1*
Y	Zöld	*Lada*	*2*
Z	Piros	*Polski*	*3*
Q	Piros	*BMW*	*1*

KOCSI-TÍPUS

Típusnév	Férőhely
Lada	5
Polski	4
BMW	5

4.6 ábra: Adatbázis szerkezet

Kétségtelen, hogy az utóbbi ábra tükrözi helyesen a valóságot, vagyis a 4.6 ábra fogalmi adatszerkezetet mutat. A 4.5 szerinti szerkezet alkotója viszont így gondolkozott: „Igaz, hogy a kocsi, a kocsitípus és a tulajdonos három eltérő dolog. Ezért ha adataikat összevonom egy állományba, redundánsan fogom tárolni a férőhelyet és a tulajdonos nevét. Ám ezek közül csak a nevet kell többszörösen karbantartanom változás esetén, hiszen a férőhely nem módosul. Külön nem vagyok kíváncsi sem a kocsitípus, sem a tulajdonos adataira: kereséseim kizárólag a kocsikra vonatkoznak. Ha a típust és a tulajdonost külön tárolnám, a keresési idő megnőne. Ezért mindent egybevetve az egyetlen állomány mellett döntök. Abban a tudatban, hogy az általam figyelembe vett feltételek megváltozása esetén az adatbázis átszerkesztésére és a programok újraírására lesz szükség."

Most már közeledünk a tartalom kettősségének a megértéséhez és megfogalmazhatjuk az adatbázisok második szerkezeti szintjének a lényegét:

D 4/2 **A technikai, hozzáférési és hatékonysági követelményeknek ill. korlátoknak megfelelően meghatározott tartalmi adatszerkezetet nevezzük az adatbázis logikai szerkezetének.**

A fentiek alapján beláthatjuk, hogy az adatbázisoknak valóban kétféle tartalmi szerkezete van. Az egyik a valóságot ***elvileg*** pontosan tükröző fogalmi, a másik a ***gyakorlati*** körülményeknek jól megfelelő logikai. A kettő közül a fogalmi az elsődleges, amelyet mindenképpen meg kell határozni. Ha a tervező nem alkotja meg a fogalmi adatbázistervet, akkor nem lesz tisztában a változások következményeivel. Ha a technikai, hozzáférési és hatékonysági körülmények módosulnak, teljesen elölről kell kezdenie a tervezést akkor is, ha maguk az ismeretek nem változnak. Ugyanakkor annak semmi akadálya sincs, hogy a fogalmi és a logikai szerkezet azonos legyen. Vagyis a fogalmi adatbázistervet minden átalakítás nélkül egy az egyben átvegyük az adatbázis logikai felépítéseként.

4.5 Két lényeg határán

A 4.1 példában többféle formában adtuk meg a „Születési dátum" ismeret tartalmát. A harmadik kitételben az amerikai hó/nap/év alak szerepelt, amely gondokat okozhatott a nap/hó/év formát ismerők számára. 14-dik hónap? Olyan nincs. Mint tudjuk, félreértés történt. A „14" az amerikai dátumformában a napot, nem a hónapot jelenti. Viszont ez az eset felhívja a figyelmünket az adattételek (tulajdonságtípusok) lehetséges tartalmainak (tulajdonságértékek) a problémájára.

A számítógépes adatkezelésben nagy hangsúlyt fektetünk arra, hogy az adatbázis tartalma a valóságot hűen tükrözze. A papírra bármit leírhatunk, de a számítógépbe bevitt ismereteket megvizsgáljuk abból a szempontból, hogy az adat értéke megfelel-e a várt tartalomnak. Ezt a műveletet ***érvényesítésnek*** (a latin/angol kifejezés alapján: validálásnak) nevezzük. A validálás elvégezhető saját programmal is. Azonban az adatkezelők megszabadítanak bennünket az ilyen procedúrák programozásától. Minél fejlettebb az adatkezelő, annál többféle érvényesítési támogatást nyújt. Ezt annak alapján teszi, hogy az adatszerkezetben meghatározzuk a validálási szempontokat.

A számos érvényesítési szempont közül most csak kettőt emelünk ki. Minden adatkezelő használatakor meg kell adnunk az adattétel ***hosszát***, ami nyilvánvalóan a bevihető jelsor méretét korlátozza. Ezen túlmenően ki kell jelölnünk az adattétel ***típusát***, amely - sokszor a hosszal

összhangban - az ismeret jellegének megfelelően korlátozza az értékhalmazt. Például a dátum típusú adatban nem szerepelhet május harmincötödike. A hossz és a típus korlátok ellenőrzését nem kell magunknak programoznunk: ezek validálását minden kezelőrendszer automatikusan elvégzi. Ehhez az adatszerkezetben meg kell adnunk az adatok ábrázolásának a módját.

D 4/3 **Az adat típusát és méretét együttesen adatábrázolásnak nevezzük.**

Az ***adatábrázolás*** (angolul: data representation) az első megközelítésben kimondottan fogalmi szintű tényező. A (Gergely-naptár szerinti) dátum fogalma kizárja a május harmincötödikét. A Rendelésszámnak már a neve is mutatja, hogy ez a tulajdonság csak számban megfogalmazott értékeket vehet fel. A második megközelítésben az ábrázolás logikai szintű tényező. Ugyanazt a tartalmat többféle módon fejezhetjük ki a korlátoknak megfelelően. Ha 99 évnél idősebb személyekről nem vezetünk adatokat, akkor a születési dátumból elhagyhatjuk az évszázad megjelölését anélkül, hogy ismeretet veszítenénk. Sőt, annak sincs akadálya, hogy az elvileg numerikus tartalmú és ekként validálandó Rendelésszám adatot karakteresen tároljuk és kezeljük. Ezért az adatábrázolásról nemcsak tartalmi, hanem mélyebb szinten is beszélnünk kell.

4.6 Az adatbázis fizikai szerkezete

Az adatbázis tervének a kialakításakor nemcsak az ismeret tartalmát, hanem az adat számítógépen való tárolásának a módját is figyelembe kell vennünk. Tekintettel kell lennünk a legkisebb természetes-nyelvi ismeretegység, a ***jel*** és a számítógépes elemi tárolási egységek közötti összefüggésekre. A számítógépek ***bitekkel*** dolgoznak úgy, hogy azok egybefűzésével fejezik ki a természetes jeleket. Ma már általában a ***byte*** (8 bit) a jel tárolóegysége. Ezért azt gondolhatnánk, hogy egy öt természetes jelből álló ismeret tárolásához öt bájtra van szükségünk.

Ez azonban így nem teljesen igaz. Mindenféle trükkökkel elérhető, hogy a természete szerint X hosszú adatot ennél kevesebb bájtban tároljuk. Az adat típusának a helyes megválasztásával az adat tárolási méretet, helyigényét lecsökkenthetjük. A trükkökre, a bájtok belső felépítésére itt nem óhajtunk kitérni. Csak arra akarunk rámutatni, hogy az adatábrázolás olyan tényező, amely egyrészt a fogalmi és logikai szerkezetre tartozik (a validálás miatt), másrészt viszont azon már túlmutat, mert átvezet bennünket az adatbázis tárolási aspektusához.

A korszerű adatbáziskezelők használata esetén valójában nem is tudhatjuk, hogy adataink az adattárolókon hol és milyen elrendezésben találhatók. Mi több, a valóban fejlett eszközök még azt is elrejtik előlünk, hogy az adat melyik tényleges tárolóegységre került. A mai adatkezelők adatállományaink ismerettartalmát „horizontálisan” és „vertikálisan” darabokra szabdalva helyezik el a tárakon. Horizontálisan - az általunk egymás mellett látott Típusnév és Férőhely (vö. 4.6 ábra) adat a tár két szélső részére kerül. Vertikálisan - mi Rózsa és Gabi adatait egymást követőeknek tartjuk, holott azok a táron szétszóródnak.

Mindehhez tegyük hozzá, hogy az ismeretek ilyen szétszabdalását a különböző adatkezelők eltérő módokon, számunkra teljesen ***transzparens*** módon végzik. Ezért hajlamosak vagyunk arra a következtetésre jutni, hogy egyáltalán nem érdemes foglalkoznunk az adatok tárolókon történő elrendezésével.

A felhasználónak valóban nincs köze a tárkezeléshez (angolul: storage management). Viszont tudnia kell, hogy a fizikai adatszerkezet a legkevésbé sem közömbös a számára.

D 4/4 **Fizikai adatszerkezetnek nevezzük az ismeretek tárolón való elhelyezésének, hozzáférésének és ábrázolásának a tudatosan meghatározott rendjét**.

Az adatbázis fizikai tervezője határozza meg azt, hogy a kezelőrendszer tárolási egységeinek megfelelően miként rendezi el adatainkat a számítógépen. Blokkokat, lapokat, adattereket, adatklasztereket - és még ki tudja mi mindent - kell meghatároznia. Mindezek persze „ál" fizikai egységek, mert arra még a fizikai tervezőnek sincs módja, hogy azokat a tár tényleges fizikai egységeihez (kötet, cilinder, sáv stb.) rendelje. Egyes adatkezelők használatakor még azt sem teheti meg, hogy az adatsor (egyed, rekord) és a „tárolódoboz" közötti összefüggést kijelölje. Más rendszerekben meghatározhatja a logikai adatsor fizikai helyének a címét, azaz megadhatja az adatok ***elhelyezési módját***.

Vegyük észre, hogy ez a lehetőség egyben kényszer is. Egyes adatkezelők alkalmazásakor meg kell adni az elhelyezési módot. Arra az adatkezelő programokban is utalni kell. Ha a mód változik, a programokat át kell írni. Viszont így hatékony kezelőprogramokat tudunk készíteni. Más adatkezelők használatakor az elhelyezési módot nem kell - nem lehet - kijelölni. A mód nem változik, tehát a programokat nem kell változtatni. Azonban arra sincs lehetőségünk, hogy a program hatékonyságát befolyásoljuk az elhelyezési mód okos megválasztásával.

Az elhelyezés az állományaink (pl. KOCSI) tételeinek az egymásutániságát illetve a logikai tétel és annak fizikai címe közötti elvi megfeleltetést jelenti. Az elhelyezés mindig egyszeres. (Most ne beszéljünk arról, hogy a KOCSI állományt több példányban, eltérő elhelyezési módokon is tárolhatjuk.) A KOCSI állomány „rekordjai" vagy a Rendszám, vagy valamelyik más adattétel elvi sorrendjében tárolódnak. Lehet, hogy ilyen sorrend egyáltalán nincsen, hanem a Rendszámot és a tárolási címeket véletlen módon feleltetjük meg egymásnak. Sőt, még az is elképzelhető, hogy a kocsik adatai mindenféle általunk látható rendszer nélkül követik egymást, mivel az elhelyezési módot nem tudjuk meghatározni.

Ismereteinket - az elhelyezési módtól függetlenül - hatékonyan és meghatározott rendben kívánjuk előkeresni. Ennek érdekében mindenféle másodlagos, technikai adatokat alkalmazunk az ismeretek összerendezésére. Az adatkezelő lehetőségeinek megfelelően indexeket, mutatóláncokat, mutatótömböket jelölünk ki. Vagyis az egyféle elhelyezési módhoz kapcsoltan egy vagy több ***hozzáférési módot*** adunk meg.

A hozzáférési mód is kétarcú. Lehetővé teszi adataink hatékony elérését. Ha a programban az adatállományt például indexszel nyitjuk meg, akkor gyorsabb lesz a visszakeresés. Csakhogy elérési igényeink változásakor átalakítjuk a hozzáférési módot - és ezért át kell írnunk korábbi programjainkat. Minél több lehetőséget nyújt az adatkezelő rendszer (pl. index mellett mutatókat is), annál jobban tudunk ügyelni az időleges hatékonyságra és annál keservesebb a változtatás.

A fizikai adatszerkezet harmadik tényezője az ***adatábrázolás***. Ez a mai adatkezelők leggyengébb pontja. Az ábrázolás nemcsak tartalmi validálást, hanem fizikai tárolási megoldást is jelent. Az adatkezelő programokban utalni kell a fizikai ábrázolásra. Következésképpen annak változásakor jelentős programmódosításokra lesz szükség.

4.7 A fizikai függetlenség

A hagyományos - állományi - adatkezelés időszakában ***még*** nem törődtünk a fogalmak valódi összefüggéseivel. A fogalmi adatszerkezet mellőzésével olyan adatbázisterveket alkottunk, amelyekben az ismereteket eleve tartalmilag korlátosan, az adatkezelő fizikai adatszerkesztési képességeinek megfelelően határoztuk meg. „Itt nincs szükség szervezésre; majd felveszünk egy indexelt szekvenciális cikk fájt" - mondta egy prominens szervezési osztályvezető annak idején. Nem gondolkodott a fogalmak összefüggésein („nincs szükség szervezésre") és eleve eldöntötte

az adatok elhelyezési-hozzáférési módját („indexelt szekvenciális”), szintén minden alaposabb megfontolás nélkül.

A mai adatkezelők egy része az adatok tényleges tárolásának a mikéntjét elrejti a felhasználó elől. Ezért a mostani tervezők némelyike ***már*** nem foglalkozik az adattárolás mikéntjével. Nagyon sokakat megtéveszt az adatkezelők forgalmazói által reklámozott ún. fizikai adatfüggetlenség elve. Ezért egy kicsit körül kell járnunk ezt a fogalmat.

D 4/5 **Fizikai adatfüggetlenségről beszélünk akkor, ha az adatok elhelyezési, hozzáférési és ábrázolási módjában bekövetkező változáskor nincs szükség a programok módosítására**.

Ezzel a szép elvvel szemben a mai adatkezelő rendszerek csak nagyon alacsony fokú fizikai adatfüggetlenséget biztosítanak. Az adat ábrázolási módjának a változtatásakor a programokat át kell írnunk. Sőt, azt is el kell viselnünk, hogy az adatkezelő mindenféle fizikai átrendezésbe fog, hiszen az ábrázolás szerencsétlen módon nemcsak fogalmi/logikai, hanem egyben tárolási tényező is. Ezért a típus és/vagy a hossz változásakor az adatbázis fizikai átalakításáig adataink nem elérhetőek. Az átalakítás - például a hossznövelés - következtében a korábban ténylegesen egymás melletti adataink a táron egymástól távolra kerülhetnek és a kezelési hatékonyság leromolhat.

Tapasztalhattuk, hogy ugyanazon ismeretek visszakeresése tízszer lassabb vagy gyorsabb, ha index alapján vagy éppen annak mellőzésével történik a kezelés. Programjainkban utalnunk kell az indexekre. Csakhogy a hatékonysági követelmények idővel megváltoznak. Ezért az új igények felmerülésekor át fogjuk írni programjainknak a technikai segédadatok (indexek, mutatók) kezelésére vonatkozó részeit. Tehát a hozzáférési mód változtatása esetén aligha érvényesül a fizikai függetlenség elve.

Végül tudomásul kell vennünk, hogy az egyetlen elhelyezési módot nyújtó rendszerek sem végső megoldások. Az X relációs kezelő alkotói a szoftver N. változatáról az N+1. verziójára való áttéréskor a korábbi fixhosszúságú invertált állományszerkezet helyett kötetlen méretű nem-invertált rekordstruktúrát vezettek be. A „közös platform” és a „hordozhatóság” szépen csengő jelszavai ellenére az ilyen váltás adatbázisaink hosszadalmas átformálásával, az ismeretek kezelésének időleges leállásával, a programok alapos átalakításával jár.

A fizikai adatfüggetlenség fennen hirdetett elve a gyakorlatban nem mindig állja meg a helyét. Tegyük mindehhez, hogy nemcsak a fizikai tárolási módok, hanem a logikai szerkezeti egységek is mozgásban vannak. Pár éve még a relációs kezelők nem engedték meg az ún. ismétlődő adatok használatát (ld. 4.3 pont). Ma már sok rendszer támogatja ezt a logikai szintű strukturális tényezőt. Aligha kérdéses, hogy az új lehetőség kapcsán adatbázisunkat át fogjuk szervezni és programjainkat módosítjuk.

Végeredményben tökéletes fizikai adatfüggetlenség nincs. Nem ez a baj. Hanem az, hogy ezt a tényt nem akarjuk észrevenni. Nem figyelünk kellőképpen az adatbázis három szintjének a tudatos meghatározására. Erről a tudatosságról lesz szó a következő pontban.

4.8 A vertikális leképezés

Az egyszerű állománykezeléshez szokott fejlesztők az adatbázis szerkezetét „középutasan” határozzák meg. Nem végeznek fogalmi elemzést. Az ismereteket azonnal a logikai és fizikai aspektusokat vegyítő adatsorokba szervezik az úgymond jól bevált COBOL, PL/1, dBASE vagy egyéb „rekordképek” mintájára.

Ez az út sokáig nem járható. Nem lehet figyelmen kívül hagyni az adatbázisok három szerkezeti szintjét. Először az eredeti tartalmat, a fogalmi szerkezetet kell átgondolni. Másodszor az aktuális igényeknek megfelelő logikai szerkezetet kell kialakítani. Csak ezek után szabad rögzíteni az ismeretek fizikai ábrázolását és tárolási struktúráját.

Az adatbázisalkotás mai rossz gyakorlatának két oka van. Az egyik az ***ismerethiány***. A fejlesztők és felhasználók egy része nincsen tisztában a három szinttel és különösen a fogalmi szerkezettel kapcsolatosan hiányosak az ismereteik. Ezért a jelenségek tényleges összefüggéseinek az elemzését elhanyagolva az igényeket az első pillanatban kiszolgálni látszó, de roppant rugalmatlan adatbázisterveket készítenek (vö. 4.3 ábra).

A másik okot a ***mai adatkezelőkben*** kell keresni. Ezek egyáltalán nem támogatják a fogalmi szerkezetet. Amikor egy mai relációs rendszer „fogalmi" szintet említ, akkor valójában csak a saját - néha igen korlátos - logikai struktúráját vetíti fel a fogalmi szintre. Semmi akadálya sincs annak, hogy a 4.3 ábra struktúráját egy mai CASE-ben „fogalminak" titulálják. Ráadásul a mai adatkezelők használatakor az adatbázis szerkezetének a leírásában eleve összekeveredtek a logikai és fizikai szintű tényezők. Hiszen a tartalommal együtt kell megadni a hozzáférési módot (pl. index) is. Nem a tartalmi, hanem a fizikai ábrázolásmódot kell kijelölni. Például a Rendelésszámot karakteresnek jelöljük meg és nincs módunk annak közlésére, hogy ez az adat logikai szinten valójában numerikus.

Vegyük észre, hogy az adatbázis három szintje hierarchikus, 1:N fokú viszonyban áll egymással. Ugyanaz a fogalmi szintű struktúra több logikaiban testesülhet meg, akár egyidejűleg is. A 4.1 ábra adatbázisa az egyik részlegnél ilyen, a másiknál olyan logikai szerkezetet vehet fel (ld. 4.2 ábra). Ugyanazt a logikai adatszerkezetet több fizikaiban valósíthatjuk meg, akár egyidejűleg is. A központban nagy gép van, a fiókban kicsi. Az utóbbi helyen trükkösebb fizikai megoldásokat kell alkalmazni, miközben mindkét gépen ugyanazokat az ismereteket kezelik.

A fejlesztők nem ismerik, a kezelők nem támogatják a ***vertikális leképezést***. Ennek lényege az, hogy nem egy, hanem mindig három - egymásra alapozott - adatbázistervet kell készíteni ugyanarra az adatbázisra. Először megalkotjuk a fogalmi tervet. Az alkalmazási körülmények mérlegelése után a fogalmi tervet nem átalakítjuk, hanem azt leképezzük a megfelelő fogalmi szintű tervre. A technikai körülmények értékelése után a logikai tervet nem átalakítjuk, hanem azt leképezzük a célszerű fizikai szintű tervre.

Az ***átalakítás*** és a ***leképezés*** között lényeges a különbség. Az átalakítással elveszik a korábbi szintű terv és a végén egyetlen tervünk marad. A leképezéssel megmarad a magasabb szintű terv is és minden időben három tervünk van. Ez azért fontos, mert mindig szem előtt kell tartanunk a ***változásokat***. Ha a fogalmi, logikai és fizikai szerkezet egyetlen tervben keveredik, akkor roppant nehéz bemérni a változások hatásait. Gyakorlatilag az adatbázis tervezését elölről kell megismételni. Ha három tervünk van, akkor könnyen felmérhetjük azt, hogy a változás melyik szintet érinti és az átalakítást csak az adott szinttől kell kezdeni. Ha a 4.3 ábra megoldása már nem jó és új logikai szerkezetre van szükség, akkor a tervezőnek újra el kell(ene) gondolkodnia a személyek és bérek fogalmi összefüggésein és mindent elölről kellene kezdenie. Viszont a 4.4 ábra tervezőjének rendelkezésre áll a fogalmi terv, ezért ő az áttervezést már a logikai szinten indíthatja.

Ellenőrző kérdések - 4

4/01 Az SQL nyelvben létezik a „duplapontosságú" numerikus adattípus. Az Ön véleménye szerint ez fogalmi (F), logikai (L) vagy fizikai-tárolási (T) szintű tényező?

4/02 A SZEMÉLY állományban szerepel a nyelvtudásra vonatkozó ismétlődő csoport (Nyelv, Vizsgadátum, Nyelvpótlék). A tervező három ilyen csoportot alkalmaz. Véleménye szerint ez fogalmi (F), logikai (L) vagy fizikai (T) szintű szerkezeti megfontolás?

4/03 Az előző feladatra vonatkozóan döntse el, hogy igazak (I) vagy hamisak (H) a következő megjegyzések:

- A baj az, hogy negyedik nyelvet nem lehet megadni.
- A fogalmi szintű tervben külön NYELV egyedet illene tervezni.
- Ha biztos, hogy további nyelvre soha nincs szükség, akkor a terv elfogadható.
- A kezelés mindenképpen nehézkes, mert melyik csoport az angol nyelvé?

4/04 A tervező meghatároz egy külön Telefonszám adattípust. Ennek tartalmát az előre elképzelt felépítés szerint validálja, de sima karakteres módon tárolja. Fogalmi (F), logikai (L) vagy fizikai (T) megfontolásról van-e itt szó?

4/05 Egy jó adatkezelő használata esetén Ön szerint az alábbi tényezők közül melyeket kell változtatni (V), melyeket esetleg (E) és melyeket nem (N), ha a Telefonszám adat méretét egy karakterrel megnöveljük?

- az adatot tartalmazó bemeneti képek
- az adatot tartalmazó kimeneti képek
- az adatot kezelő programok
- az adatbázis fizikai szerkezete
- az adatbázis logikai szerkezete

5. AZ ADATBÁZIS VETÜLETEI

5.1 Egy alapvető probléma: a nézetorientáltság

Az alcímben jelzett baj megértéséhez ismerni kell az eddigi hazai számítástechnikai fejlődés bizonyos nemkívánatos vonásait. Ugyanannak a szervezetnek az eltérő részlegei a saját maguk részére, a többiektől szeparáltan alakították ki „rendszereiket". Más és más hard- és szoftvereszközöket vásároltak; de bennünket most nem a szervezeti szintű technikai káosz káros kommunikációs következményei érdekelnek. A cégek nem alkalmazták a megfelelő rendszertervezési módszertani megoldásokat. Ennek következtében az eltérő részlegek sokszor részben ugyanazokat az adatokat vitték számítógépre. Viszont egyazon ismeretfélének más-más nevet és tartalmat adtak (pl. másféle kódolást használtak) vagy különböző ábrázolási módot választottak. Ha pedig egy szakterületnek, funkciónak a gazdája nem lelkesedett a számítástechnikáért, akkor bizonyos fontos adatok egyáltalán nem kerültek gépre.

Az információs fejlesztésnek ez a módja szervezeti szinten adattöbbszörözésre, adatellentmondásra és adathiányra vezetett. Az igen vegyes eszközpark és a már számítógépen tárolt ismeretek összehangolatlansága a szervezetek informatikai fejlődésének ***objektív*** akadályai. Ezek nem kis gondok, de ha lenne rá szándék és erőforrás, ez a probléma viszonylag „könnyen" feloldható lenne. Azonban a valóban integrált információs rendszer kialakításának van egy sokkal nehezebben leküzdhető ***szubjektív*** gátja is.

Az integráció legnagyobb akadálya a ***nézetorientáltság***. A fejlesztők és a felhasználók egyaránt hozzászoktak egy adatkezelési szemléletmódhoz. Az Y fejlesztő létrehozott néhány állományt; meghatározta az abban lévő ismeretek és a képernyők tartalmát illetve formáját stb. Az X felhasználó ettől fogva meg volt győződve arról, hogy úgy néz ki az „adatbázis" ahogyan azt ő elképzelte, és más képet már nem is ölthet. A felhasználó ugyan sűrűn változtatja a nézetét a saját „adatbázisáról", de azt - a fejlesztővel együtt - nehezen viseli el, hogy más kívülről szóljon bele az ő adatbázis magánügyeibe. Márpedig világos, hogy a szervezeti szinten teljes, egyértelmű és nemredundáns ismeretkezelés megvalósítása érdekében az egyes részlegek által tárolt ismereteket és azok kezelését össze kellene hangolni. Ez pedig bizony „külső" beleszólással is jár.

Ebben a fejezetben az adatbázis egyéni és közös nézeteire, vetületeire hívjuk fel a figyelmet. Ezen túlmenően, de ezzel a témával kapcsolatosan bemutatjuk az adatbázis sokoldalú képét.

A teljes szervezet szintjén az ismeretek integrálása komoly objektív előnyökkel - az adattárolási illetve -kezelési költségek csökkenésével és minőségileg jobb ismeretszolgáltatással - jár. Ennek érdekében le kell küzdeni a szubjektív akadályokat. Ehhez pedig meg kell ismerni az adatbázis különböző lehetséges nézetmódjait.

5.2 A nézetorientáltság káros következményei

A magyarázat kedvéért tegyük fel, hogy ugyanabban a szervezetben a felhasználók egyik körét a gépkocsikra és az azok tulajdonosaira vonatkozó ismeretek érdeklik. Ők nem kíváncsiak a kocsitípussal kapcsolatos adatokra (5.1 ábra). Az alkalmazók másik csoportjának nincs szüksége a tulajdonos adataira, viszont ők kezelni szeretnék a kocsitípussal kapcsolatos ismereteket is (5.2 ábra). A kétféle felhasználói kör egymástól függetlenül kialakítja magának a két ábra által mutatott „adatbázisok” szerkezetét.

KOCSIK-1

Rendszám	Szín	Típusnév	Tulajnév	Tulajcím
X	F	Lada	Rózsa	C1
Y	Z	Lada	Gabi	C2
Z	F	Polski	Lajos	C3
Q	P	BMW	Rózsa	C1

5.1 ábra: A tulajdonosban érdekelt felhasználó ismeretei

KOCSIK-2

Rendszám	Szín	Típusnév	Férőhely	Fogyasztás
X	Fehér	Lada	5	A
Y	Zöld	Lada	5	A
Z	Fehér	Polski	4	B
Q	Piros	BMW	5	C
W	Piros	Polski	4	B

5.2 ábra: A típusban érdekelt felhasználó ismeretei

1. probléma: ***Redundancia***. A két állományban külön-külön rögzítették a Rendszám, a Szín és a Típusnév adatot. Az átfedés a szükségesnél nagyobb együttes tárigénnyel jár. Az adatokat többszörösen kell bevinni, módosítani, törölni stb. - kezelni - a szervezet egészének a szintjén.

2. probléma: ***Inkonzisztencia***. Az első állományban a színeket kódolták, a másodikban nem. Ezért kétféle programrészlet szükséges. Továbbá semmi nem garantálja, hogy az egyik felhasználó által készített kimutatás ugyanazokat az ismereteket fogja felsorolni a közös adatok tekintetében, mint a másik által összeállított. Például már amiatt sem, hogy a kétféle állománynak eltérő lehet az aktualitása, a karbantartási ciklusa. Így fordulhat elő a terjedelmi eltérés is. Az, hogy az egyik „adatbázis” már mutatja a W rendszámú gépkocsi ismereteit, a másik még nem.

3. probléma: ***Ismerethiány***. Egyik állományban sem szerepel a biztosítási díjjal kapcsolatos adat. Emiatt az abban érdekelt felhasználói kör létre fog hozni a maga részére egy harmadik „adatbázist”. Ebben természetesen megismétli a gépkocsira, kocsitípusra és tulajdonosokra vonatkozó adatok jó részét, ami által a redundancia és az inkonzisztencia tovább fokozódik.

Felmerül tehát a kérdés, hogy miként lehet olyan közös adatszerkezetet kialakítani, olyan együttes adatbázist létrehozni, amivel a fenti problémákat elkerülhetjük, de úgy, hogy mindegyik felhasználót kiszolgáljuk. Azaz mindegyik számára a korábban megszokott képet nyújtsuk.

Mielőtt az előbb feltett kérdésre válaszolhatnánk, rá kell mutatnunk az 5.1 és 5.2 ábra által jelzett adatszerkezetek egy további problémájára is. A két feltételezett állomány önmagában is redundáns, mert az első táblában a tulajdonosok, a másodikban a kocsitípusok adatrészsorai többszörösen azonos tartalommal fordulnak elő. Ez azért van, mert a két állomány tervezői nem

adatbázisban gondolkodtak - ezért szerepelt fentebb idézőjelben az „adatbázis” szó. Az adatbázis ugyanis több egyedtípust és azok között meghatározott kapcsolattípus(oka)t feltételez. Ezek alkalmazása esetén az említett redundancia elkerülhető, amint azt az alábbiakban bemutatjuk.

5.3 Globális és parciális adatbázisszemlélet

Az előző példa esetében a fejlesztők nem látták meg a valós egyedtípusokat, mert nem ismerték az adatbázis lényegét. Az is gyakran előfordul, hogy több alkalmazás mindegyikében adatbázisszerűen gondolkodnak, csak éppen az egyazon jelenségekre vonatkozó ismereteket eltérő módon meghatározott egyedtípusokhoz kötik. Ha pedig egyedtípusok nincsenek vagy azokat másképpen látják a különböző alkalmazásokban, akkor nagyon nehéz a közös adatbáziskép kialakítása.

Ezért az egyetlen közös adatbázis tervének a meghatározása két feladatot ölel fel. Az elsőben az egyes felhasználók nézeteit kell adatbázisszerűvé átformálni.

Az 5.3 és a 5.4 ábrák mutatják az első lépés eredményeit. Az adatbázisszerű átalakítás látszólag mindig gondokkal jár. Mindkét esetben be kellett vezetnünk egy-egy plusz adatfélét (Tulajkód, Típuskód), tehát növelnünk kellett az állományok „szélességét”. Csakhogy fontoljunk meg két dolgot. Az egyik az, hogy amennyiben a tulajdonos megjelölése az első esetben, a kocsitípusé a második esetben egyértelmű lett volna, úgy az újabb adatfélére nem lenne szükség az adatbázisban. Ellenkező esetben viszont maguk az eredeti megoldások is rosszak, mert a tulajdonosra/kocsitípusra nem lehet egyértelműen hivatkozni. A másik az, hogy általában a valós állományok „szélessége” (az adatféleségek száma) lényegesen kisebb (tízes, százas nagyságrendű), mint azok „hosszúsága” (a tárolt tételek száma, amely ezres, sőt milliós nagyságrendű).

KOCSIK-1

Rendszám	Szín	Típusnév	*Tulajkód*
X	F	Lada	*T1*
Y	Z	Lada	*T2*
Z	F	Polski	*T3*
Q	P	BMW	*T1*

TULAJDONOS

Tulajkód	Tulajnév	Tulajcím
T1	Rózsa	C1
T2	Gabi	C2
T3	Lajos	C3

5.3 ábra: A tulajdonosban érdekelt felhasználó adatbázisrészlete

KOCSIK-2

Rendszám	Szín	*Típuskód*
X	Fehér	*T1*
Y	Zöld	*T1*
Z	Fehér	*T2*
Q	Piros	*T3*
W	Piros	*T2*

KOCSITÍPUS

Típuskód	Típusnév	Férőhely	Fogyasztás
T1	Lada	5	A
T2	Polski	4	B
T3	BMW	5	C

5.4 ábra: A típusban érdekelt felhasználó adatbázisrészlete

Az 5.3 és 5.4 ábra által mutatott megoldások lényegesen jobbak a korábbiaknál. A belső redundancia megszűnt, és ezért csökken az adattárigény. Az adatkezelési idő töredékére csökken, de ennek belátását az olvasóra bízzuk. (Fontolja meg, hogy a méretcsökkenés mennyire redukálja

a lemezelérési időt; a redundancia megszűnésével miképpen csökken pl. a lakcímkarbantartás ideje; végül milyen többletlehetőségeket nyújt az új szerkezet, amelyben a kocsikra nem is kell rákérdeznünk, ha csak a tulajdonosok neveire és címeire vagyunk kíváncsiak.)

A második lépésben az új képeket össze kell hangolni. Ekkor már csak arra kell ügyelnünk, hogy egyeztessük az ismeretábrázolást. Az senkit sem zavarhat, hogy a kocsitípus és a tulajdonos kódjában egyaránt „Tx" értékek szerepelnek, mint ahogyan világos, hogy az adatbázisban lehet „12345" értékű cikkszám és ugyanilyen értékű rendelésszám. Nem fogjuk őket összetéveszteni. Nem baj, ha az eltérő ismeretfélék tartalma azonos. Viszont probléma, ha az azonos ismeretfélék tartalma eltérő. Pl. ha a Szín adat a két adatbázisban más tartalmú. Itt megegyezésre van szükség.

Feltételezve, hogy a Szín adatot mindkét alkalmazásban a természetes érték szerint vezetik, a célszerű közös adatbázist az 5.5 ábra mutatja. Az ábra adatbázistervében szerepelnek mind az első, mind a második felhasználó által igényelt adatok. Mi több - feltételezve, hogy a díjtétel a kocsitípus függvénye - a harmadik felhasználónak is tudunk ismereteket nyújtani.

Elképzelhető, hogy a kérdéses szervezetben egyetlen olyan felhasználó sincs, aki az 5.5 ábra három egyedtípusának valamennyi tulajdonságtípusát egyidejűleg kezelni óhajtja (az ábrán fel nem tüntetett, az egyedtípusok közötti kapcsolattípusok szerint). Továbbmegyünk: ha egy szervezetben létrehoznak egy valódi adatbázist, az az egyed- és kapcsolattípusok százait, a tulajdonságtípusok ezreit ölelheti fel. Pl. a kép kiegészülhetne a gépkocsik által okozott károkkal, a kárkifizetésekkel, a biztosítási díj befizetéseivel stb. Tehát egészen biztos, hogy nincs olyan alkalmazás, amelyben szükség lenne egy élő adatbázis valamennyi adatára egyidejűleg (egy program által kezelten, egy kimutatásban).

KOCSI

Rendszám	Szín	*Típuskód*	*Tulajkód*
X	Fehér	*T1*	*T1*
Y	Zöld	*T1*	*T2*
Z	Fehér	*T2*	*T3*
Q	Piros	*T3*	*T1*
W	Piros	*T2*	*?*

TULAJDONOS

Tulajkód	Tulajnév	Tulajcím
T1	Rózsa	C1
T2	Gabi	C2
T3	Lajos	C3

KOCSITÍPUS

Típuskód	Típusnév	Férőhely	Fogyasztás	Díjtétel
T1	Lada	5	A	II
T2	Polski	4	B	I
T3	BMW	5	C	IV

5.5 ábra: Végsőre formált közös adatbázisrészlet

Ezért az adatbázis szerkezetét - pl. az 5.5 ábrának megfelelő módon - csakis egyetlen „felhasználó" látja annak teljes összetettségében: az adatbázis tervezője. Az ő feladata a korábbiakban jelzett problémák elkerülése, az integrált adatbázis kialakítása. Ő egyetlen adatbázisképpel dolgozik; a teljes szervezet szempontjából egységessel és általánossal. Egységes egyetlennel, mert a korábbiakkal szemben az 5.5 ábra már csak egy szerkezetet mutat. Általánossal, mert ez a szerkezet minden felhasználót kötelez. Minden felhasználó minden kért adatot megkap, de csak akkor, ha az beleilleszkedik az összképbe, az adatbázisnak az összes ismeretet átfogó és éppen ezért ***globálisnak*** nevezett nézetébe.

Kérdés: Az 5.5 ábrának megfelelő szerkezetű adatbázis kiszolgálja-e azt a három felhasználót, akiről az előző pontban emlékeztünk meg? Válasz: Az 5.1 és 5.2 ábrának megfelelő ismeret az 5.5 ábra szerinti adatbázisból könnyen levezethető, sőt a díjtétel ismeretet is visszanyer-

hetjük abból. A már meglévő és megfelelően szerkesztett adatokat tetszőleges kombinációkban kereshetjük ki az adatbázisból. Ekkor - mint arra fentebb utaltunk - az adatbázisnak csak egy részletében vagyunk érdekeltek, és ezért beszélhetünk az adatbázis ***parciális*** nézeteiről. Az 5.3 és a 5.4 ábra felfogható a 5.5 ábra adatbázisa két részleges nézeteként.

Szemben az 5.1 és 5.2 ábra által sugallt hagyományos megoldásokkal, az adatbázis parciális nézete valóban tetszőleges lehet. Az 5.5 ábra adatbázisszerkezete alapján az 5.6 ábra kicsit furcsa, de nem megvalósíthatatlan nézetét is képezhetjük. Erre a korábbi változatokban nem nyílt lehetőségünk:

KOCSIK

Rendszám	Szín	Férőhely	Tulajcím	Díjtétel
X	Fehér	5	C1	II
Y	Zöld	5	C2	II
Z	Fehér	4	C3	I
Q	Piros	5	C1	IV
W	Piros	4	?	I

5.6 ábra: Egy „vegyes” felhasználói szemlélet

Tehát az egyetlen globális adatbázisnézetből elvileg tetszőleges számú, az adatbázisban tárolt ismereteket különböző módon kombináló parciális nézet vezethető le. Az 5.5 ábrán mindössze tíz eltérő ismeretféle szerepel. Az olvasó kiszámolhatja, hogy ezeknek az egy-egy parciális képbe való bevételével vagy onnan való elhagyásával hányféle részleges nézet generálható ennek a picinyke mintapéldának az esetében is.

A globális és parciális nézettel a mindennapi életben találkozunk. Figyeljük csak meg a következő közlések közti különbségeket:

5.1 példa

„Az X rendelésben az A cikket kérték.”
„Az Y rendelésben a B cikket kérték.”
„Az A cikket az X rendelésben igényelték.”
„A B cikket az Y rendelésben igényelték.”

Az első és a harmadik illetve a második és a negyedik mondat lényegében ugyanazt az ismeretet közli. Azonban az alany és az állítmány felcserélése nem puszta játék. Az első két kitétel az értékesítési részleg munkatársától származik, aki a rendelések és cikkek viszonyát az előbbi felől „nézi”. A második két mondat a raktárgazdálkodási csoport tagjának tulajdonítható. Ő a rendelések és a cikkek kapcsolatát az utóbbiak felől „szemléli”. A két nézet eltérésének dacára a valóság egy. Tudniillik az, hogy a rendelések és a cikkek összefüggenek egymással.

Amint látjuk, a „valódi” adatbázisszerkezet megalkotásával egyetlen felhasználó sem veszít ismeretet. Sőt, mindegyik többféle ismerethez juthat. A következő pontokban azt fogjuk bemutatni, hogy a globális és parciális nézet egyeztetésének nemcsak az elméleti alapja erős, hanem gyakorlatilag sem éri sérelem a közös adatfelhasználástól esetleg ódzkodó felhasználókat. Előbb azonban meg kell világítanunk az „adatmodell” szó kétféle jelentését, mivel ezt a fogalmat eltérő tényezőkre alkalmazzák.

5.4 Az „adatmodell” jelentése és tartalma

Az informatikában az „adatmodell” szót kétféle értelemben használják: specifikusan és generikusan. A ***specifikus*** jelentés szerint az adatmodell az adott alkalmazási környezet közös adatbázisának az absztrakt tükörképe (vö. D 3/9). A szó szigorú értelmében az adatmodell mindig fogalmi szintű adatbázistervet jelent. Ezen belül az adatmodellnek két tényezőkörét kell megemlítenünk.

Az adatmodellnek van felépítési, ***szerkezeti*** (angolul: structure) vonzata. Az adatmodell véges számú jelenségnek, azok véges számú sajátosságának és viszonyának a fogalmi mintája. A jelenségek, sajátosságok és viszonyok tükörképei - például az egyed-, tulajdonság- és kapcsolattípusok - egymással meghatározott összefüggésekben állnak, vagyis ezek a tényezők szerkezetet alkotnak. Mivel elemek és viszonyaik szervezett összefüggéséről van szó, az adatbázist egészen nyugodtan ***rendszernek*** is tekinthetjük. Az adatbázis az adott alkalmazási környezet ismeret-rendszere.

A specifikus értelemben vett adatmodellnek van egy másik ún. ***korlát*** (angolul: constraint) vonzata is. A korlátok az adatbázis tényezőire és azok viszonyaira vonatkozó meghatározások, feltételek, megkötések, kitételek, kijelentések stb. halmazát jelentik. A korlátok egymással is kapcsolatban állhatnak, ezért rendszert alkotnak. Minden adatbázishoz a szerkezeti rendszer mellett tartozik egy korlátrendszer is, amely az adatmodell része. Példák a korlátokra: Minden gépkocsinak kell, hogy legyen ismert típusa. A gépkocsik díjtétele X és Y forint közé esik. A kocsik fogyasztása nem lehet nulla vagy negatív szám.

Vegyük észre, hogy az adatbázis szerkezete önmagában is hordoz egy sor korlátot. Példák: minden gépkocsinak csak egy típusa és egy tulajdonosa lehet; a típus csak olyan értéket vehet fel, ami létezik a KOCSITÍPUS egyedtípus előfordulásai között; a kocsi rendszáma nem lehet ismeretlen. Mindezek, mint majd látni fogjuk, az adatbázis szerkezetéből következnek és ezért nem kell őket külön leírni. Éppen ezért korlátnak valójában csak az adatszerkezetben meg nem adható feltételeket, kikötéseket tekintjük.

Az adatmodell ***generikus*** értelemben egy meghatározott „adatbázis-filozófiát” jelent. Általános elképzelést, elcsépelt szóval: megközelítést arra nézve, hogy konkrét alkalmazásoktól függetlenül az adatbázisok szerkezetét általában milyen tényezők alkotják, azok között általában milyen összefüggések léteznek és általában miképpen kell megadni a korlátokat. Az 5.5 ábrában három egyedtípus 10 eltérő tulajdonságtípusát tüntettük fel, elhanyagolva a kapcsolattípusokat és a korlátokat. Az 5.5 ábra tehát egy konkrét adatmodell részlete. Viszont az a koncepció, amely szerint az adatbázis fogalmi szerkezetét általában egyed-, tulajdonság- és kapcsolattípusok szervezett együtteseként kell megadni, általános felfogásmód, „adatbázis-filozófia”.

Többféle ilyen filozófia létezik. Korábban már említettük a *bináris relációk* modelljét (ld. 3.5 pont). Az is egy elvileg lehetséges - bár gyakorlatilag nem túl sikeres - generikus adatmodell. Egészen más az *n-fokú relációk* modellje. (Ezt nevezik ma röviden relációs modellnek, elfeledkezve az n-fokú jelzőről.) Ismét más tényezőkkel és szerkezetekkel operálnak a *hálós és hierarchikus* modellek [7]. Mi több, az adatbázisoknak létezik *matematikai-logika*-alapú modellje is, amelyben az adatbázis szerkezeti tényezőit nem adatszerűen, hanem matematikai-logikai formulák segítségével adják meg. Ezen filozófiák leírására, elemzésére és összehasonlítására itt nincs módunk; ahhoz egy külön könyvre lenne szükség. Ezt az elemzést elvégezték helyettünk mások (pl. [8]) bár úgy, hogy az értékelésben még nem szerepelhetett a legújabb, az *objektum-orientált* irányzat [9].

A generikus adatmodellkoncepciókra általában két dolog jellemző. Az egyik az, hogy valójában ***nem fogalmi szintű*** tényezőkkel operálnak. Például a relációs modell „reláció” egysége se nem tárolási (fizikai szintű), se nem valós (fogalmi szintű) tényező, hanem valahol a kettő között elhelyezkedő tartalmi, tipikusan logikai szintű alkotóelem. Például az 5.1 ábra állománya

felfogható relációként is, amelynek tárolásáról mit sem tudunk és az a reláció nem a valóságnak megfelelően tükrözi a jelenségeket (mert nem választja szét a kocsi és a tulajdonos valós lényegeket).

A nem fogalmi szintű modellkoncepcióknak az is a sajátja, hogy az adatbázis modelljét elválaszthatatlanul összekapcsolják az ***adatbáziskezelés modelljével***. Tehát egyben programnyelvi modellek is, amelyek nemcsak az adatbázis szerkezetét korlátozzák, hanem azt is megszabják, hogy milyen parancsokkal lehet az adatbázist kezelni. Az adatmodell és a kezelési modell ilyen egymásraépülése a koncepciókészítők szerint előnyös. Gyakorlatilag részben az is. Viszont ez az „árukapcsolás" szükségszerűen azzal a negatív jelenséggel jár, amit a korábbiakban eszközfüggőségnek neveztünk (ld. 4.1 pont).

A következő pontban egy olyan generikus adatmodellkoncepciót, adatbázisfilozófiát mutatunk be, amely elméletileg majdnem tökéletesen összehangolja az adatbázis szinteket és nézeteket, de nem kényszerít ránk az adatmodellel együtt egy kezelési megközelítést is.

5.5 Az ANSI-SPARC architektúra [10]

Az ANSI (American National Standards Institute) már a hetvenes évek közepén meghatározta az adatbázisok általános elvi felépítésének - a generikus adatmodellnek - a célszerű képét, amit azóta is ANSI-SPARC adatbázis-architektúrának nevezünk. Az elképzelést időközben az ISO (International Standards Organisation) elfogadta és továbbfejlesztette illetve finomította [11]. Mi itt csak a lényeget akarjuk összegezni.

Az ANSI architektúra szerint minden adatbázisnak három „szintje" van. Nem az általunk már ismertetett fogalmi, logikai és fizikai szintekről van szó. Az azoknak való megfeleltetésre majd alább kitérünk. (Az ANSI egyik apró tévedése a „szint" szó használata, mivel nem egymás alatti három réteget alkotnak az ismertetendő tényezők.)

Az első szint a ***fogalmi*** (angolul: conceptual level). Ezen a szinten egy alkalmazási környezet valamennyi ismeretét és azok valamennyi összefüggését egyetlen közös adatmodellben tükrözik. Ez a reprezentáció mentes a kezelőrendszer típusától és az amögött álló „filozófiától". A fogalmi adatmodell nem relációs, nem bináris, nem hálós stb. Kategóriái valóban fogalmi tényezők. Mára már mindenütt szinte egyöntetűen Chen kezelőfüggetlen egyed-tulajdonság-kapcsolat hármasát alkalmazzák a fogalmi adatmodell alapkategóriáiként [12]. A fogalmi adatmodell tehát a szó generikusan értelmében is egy, hiszen egységes tényezőkben fogalmazódik meg. Az adott alkalmazási környezetben vett specifikus értelmében is egy, mivel egyetlen integrált, tetszőlegesen összetett adatbázist feltételez.

A második szint a ***belső*** (angolul: internal level). Minden fogalmi szintű adatbázisnak van legalább egy, de lehet tetszőlegesen több belső reprezentánsa. (A „belső" jelző itt a számítógépen belülire utal.) Magyarul: ugyanazon tartalmú adatbázist illetve annak megfelelő részeit a különböző gépeken vagy akár egy számítógépen belül is eltérő módokon lehet megszervezni. Igény szerint az adatbázisnak lehet több, másféle módon szerkesztett példánya vagy részlete is. Történeti okai vannak annak, hogy a belső szintet az ANSI miért nem tagolta logikaira és fizikaira. Az ANSI koncepció születésekor dúlt a nevezetes vita a hálós és a relációs filozófiák között [13]. A két irányzat eltérő logikai adatbázisszerkezetekkel dolgozott, a relációsnak pedig még nem is volt fizikai szerkezeti elképzelése. Az ANSI szerint helytelen, ha a felhasználónak törődnie kell azzal, hogy adatait milyen kezelővel manipulálják. Ez számára belső, transzparens ügy kellene, hogy legyen. Nekünk viszont nem szabad elfeledkeznünk arról, hogy a relációs, a hálós, a hierarchikus stb. logikai szerkezet egymástól teljesen eltérő. Tehát azokat másként kell megtervezni, és éppen ezért a logikai és a fizikai adatbázisszerkezet kialakítása kettős feladat.

A harmadik szint a ***külső*** (angolul: external level). Minden fogalmi szintű adatbázisnak több felhasználója van. Az egyes felhasználók az adatbázis különböző részleteiben érdekeltek. Sőt, ugyanaz a felhasználó is többféle módon akarja látni az adatbázis tartalmát. Az 5.5 ábra által tükrözött fogalmi adatbázisnak többféle látásmódja lehet. A fenti összes többi ábra megfelelhet egy-egy külső látásmódnak. A „külső" jelzőt az indokolja, hogy a felhasználók egyike sem tudja, hogy belül a számítógépen milyen az adatok szerkezete.

D 5/1 **Az adatmodellnek a felhasználó által kezelésre kiválasztott részét nézetnek vagy szemléletnek nevezzük.**

A ***nézetnek*** (angolul: view) levezethetőnek kell lennie az adatmodellből. Ez azonban nem jelenti azt, hogy a nézetben foglalt ismeretek szerkezete meg kell, hogy egyezzen az adatmodell struktúrájával. Az 5.1 ábra tartalma visszakapható az 5.5 ábrából, de az előbbiben az utóbbi több egyedtípusának a tulajdonságtípusait egyetlen egységbe vonták össze.

D 5/2 **Az adatmodellben lévő tulajdonságtípusokból mesterségesen - vagyis nem a tényleges egyedszerkezeteknek megfelelően összeállított - tulajdonságsorokat virtuális egyedeknek nevezzük.**

A felhasználó tetszőleges számú és szerkezetű virtuális egyedet képezhet. Sőt, azt is megteheti, hogy ugyanazt a nézetet több kezelőprogramban alkalmazza. Attól tehát nem kell félnie, hogy a közös adatbázis nem fogja őt kiszolgálni. Attól sem kell tartania, hogy mások illegálisan fogják használni az adatait, mert az adatbázisban tetszőleges kombinációjú ***zárakat*** (angolul: lock) helyezhet el azokra az egyed- és tulajdonságtípusokra, sőt -előfordulásokra is, amelyek ilyen vagy olyan kezeléséből ki akar rekeszteni másokat.

Az ANSI a külső szint esetében is elhanyagolt egy dolgot. Nevezetesen azt, hogy ennek a szemléletnek is van logikai és fizikai szintje. Elvileg nincs akadálya annak, hogy a valójában X hosszban tárolt Y típusú adatot Z hosszban és Q típusúként „lássa" a felhasználó. Ezért a nézet megtervezésekor is kettős feladat vár ránk: a tartalmi és formai kialakításé.

5.6 A szintek megfeleltetése, modellek és sémák

A fogalmi, belső és külső szintek között az adatbáziskezelő rendszer teremt kapcsolatot. Amikor a felhasználó megadja az igényét (külső szint), azt a kezelő leképezi az általános képre (fogalmi szint), hogy megvizsgálja a kérés teljesíthetőségét és behatárolja az adatbázis érintett részeit. Amikor pedig tényleges kezelésre kerül sor, akkor az adatmodell vonatkozó elemeit mintegy leképezi a tárolási szerkezetre (belső szint), vagyis meghatározza, hogy hol találhatók a keresett adatok. Az utóbbi művelet a fizikai adatfüggetlenség miatt rejtett a felhasználó számára. A korszerű kezelőknél az előbbi is az, ami alapot ad egy újabb fogalom megvilágításának.

D 5/3 **Logikai - más néven adat-program - függetlenségről beszélünk akkor, ha az adatok szerkezetében bekövetkező változás nem hat ki a kezelőprogramokra.**

Mivel az 5.2 ábra nézetét képviselő felhasználó nem érdekelt a kocsitulajdonos adataiban, a tulajdonos címe adat méretének a változása, a tulajdonos foglalkozása adat felvétele, vagy akár a tulajdonos nevének a törlése nem érinti az általa használt kezelőprogramokat. Sőt, a díjtétel változtatása sem fogja érdekelni. Ennek az igen egyszerű oka az, hogy - szemben a hagyományos

adatkezeléssel - a modern adatkezelő programokban nem kell definiálni az adatbázis valamennyi érintett egyedtípusának az összes tulajdonságtípusát. Csak a program által ténylegesen kezelt tényezőkre kell hivatkozni. Így a programban meg nem jelölt adatfélék törlése vagy módosítása illetve bármilyen új adatféle felvétele az adatbázisba nem igényli a program átírását.

Arról viszont nem szabad elfeledkezni, hogy a program által ténylegesen kezelt tényezők együttese nem azonos a felhasználói nézettel (view). Ezért az ANSI architektúrából hiányzik egy igen fontos tényező, amelynek lényegét egy példa segítségével mutatjuk be.

Tegyük fel, hogy valaki az 5.2 ábrának megfelelő nézetet kívánja generálni az 5.5 ábra adatbázisának a tartalmából úgy, hogy csak a budapesti tulajdonosok kocsijaiban érdekelt. Bár az illető nem akarja látni a tulajdonos címét, a megfelelő kocsik kiválasztásához a programnak ezt a tulajdonságtípust is kezelnie kell. A felhasználó nem érdekelt a tulajdonos kódjában sem, de a gépkocsikat és tulajdonosokat csak ezen tulajdonságtípus alapján tudja összekötni a program. Ezért a felhasználó nézete és a felhasználást kiszolgáló program szemlélete nem mindig ugyanaz.

D 5/4 **Az adatmodellnek a kezelőprogram által érintett részét adatalmodellnek nevezzük.**

Az adatalmodell megegyezhet a nézettel, de lehet annál bővebb is. Az almodell kétféle tényezővel szokott kiegészülni a nézethez képest. Előfordulhat, hogy a nézet több egyedtípus tulajdonságaiból összevont úgy, hogy azok valamelyikének azonosítóját a virtuális egyed nem tartalmazza. Ekkor a valós egyedek között ***kapcsolatot teremtő*** tulajdonságtípus (példánkban: a Tulajkód) is az adatalmodell része. Ha a felhasználó a teljes egyedelőforduláshalmaz (minden gépkocsi) helyett csak részhalmazt kíván kezelni (budapesti kocsik), akkor a ***kiválasztási ismérvként*** szolgáló tulajdonságtípus is (Tulajcím) az adatalmodellbe kell, hogy tartozzon, miközben nem része a virtuális egyednek. (N.B.: Valójában a kapcsolatot teremtő tulajdonság is kiválasztási ismérv. Csak nem külső, a felhasználó által látott, hanem belső, a kezelő által az egyedek közötti átmenetre használt tétel.)

A külső szint tényezőit az adatbázist kezelő programokban kell megadni. Az adatbázis belső szerkezetét az adatok tényleges betöltése előtt kell közölni az adatkezelővel. Mindkettőt megelőzően, az adatbázis használatának a megkezdése előtt le kell írni a kezelő számára az adatbázis fogalmi felépítését. Most a hangsúly a „leírni" lényegen van.

D 5/5 **Az adatmodell formális leírását fogalmi sémának hívjuk.**

A leírás formája a kezelőrendszertől függ. Ezért különböző adatkezelők együttes használata esetén nem elképzelhetetlen, hogy egyazon adatbázisnak több fogalmi sémája is van, noha maga a fogalmi adatmodell - egy. A fogalmi séma mintájára beszélünk az adatbázis belső és külső sémáiról is. Már csak arra kell visszautalnunk, hogy az adatmodell a struktúra és a korlátok együttese. Ezért a sémák nem pusztán szerkezeti leírások, hanem a korlátokat is felölelik.

5.7 Adatmodell-elmélet és adatbázis-gyakorlat

A fentiekben felvázoltuk, hogy az adatbázisok általában miként épülnek fel elméletileg. Az adatbázisnak van egy és csakis egy - minden alkalmazás adatait átfogó - fogalmi képe. Ehhez több - logikai és fizikai szinten meghatározandó - belső szerkezet tartozhat. Az adatbázis használatakor több - logikai és fizikai szinten kijelölendő - külső struktúrát (nézetet illetve almodellt) fogunk meghatározni. Az adatkezelő feladata, hogy a külső, parciális igényeket az egyetlen közös, globális nézeten át a belső lehetőségek szerint kielégítse. Ez a kijelentés nemcsak állítást,

hanem tagadást is tartalmaz. Az elméletileg jó adatbázisarchitektúrában nincs helye annak, hogy a felhasználó a fogalmi séma megkerülésével a külső és a belső szint között közvetlen kapcsolatot létesítsen.

A valós adatbáziskezelő rendszerek még nem jutottak el erre az elméleti szintre. Sok kezelőben még arra sincs mód, hogy az egyetlen globális fogalmi sémát megadjuk. A teljes adatbázisnak nincs egyetlen sémája; a programokban jelöljük ki az adatbázis érintett részeinek a leírásait. Tehát valójában eleve csak „alsémákkal" dolgozunk. A gyengébb kezelőkben még alséma megadásra sincs lehetőség; az adatbázis érintett része implicit marad (vö. dBASE és társai). Más esetekben meg kell adni az adatbázis egyetlen leírását, de ez a séma nem fogalmi szintű, hanem logikai és fizikai szintű tényezők egyvelege. Ezek után már nem csoda, hogy a programokban az adatokra direkt is lehet hivatkozni. Vagyis a „mindent a fogalmi sémán keresztül" elvvel ellentétben a külső és a belső szint között közvetlen kapcsolatot is létre lehet hozni.

Amint látjuk, a meglehetősen tiszta elmélettel szemben a gyakorlatban káosz uralkodik. Ez pedig baj, mert az adatbáziselméletek nem véletlenül tartalmaznak bizonyos - a gyakorlatra nagyon is kiható - elveket. Láthattuk, hogy milyen következményekkel jár a fizikai (vö. D 4/5) és a logikai (vö. D 5/3) adatfüggetlenség követelményeinek a megsértése. Ezért az adatkezelő kiválasztásakor érdemes figyelni arra, hogy a rendszer mennyire tesz eleget az ebben a fejezetben felvázolt elveknek. Bizonyosak lehetünk abban, hogy hiányosságok esetén meglehetősen komoly következményekkel kell számolni.

Mivel a mai kezelők még nem tökéletesek, adatbázisaink kiépítésében és használatában nem szabad csakis a ***technikára***, a szoftverre hagyatkozni. Nagyon-nagyon sok tennivaló vár magára az ***emberre***, ha az amúgy sem olcsó adatbáziskezelés költségeit nem akarjuk értelmetlenül megnövelni. Magyarul: a szakembereknek kell pótolniuk az eszközök hiányosságait. Ehhez pedig az eddigieknél sokkal alaposabban meg kell ismerniük az adatbázisok lelkivilágát.

5.8 Adatmodell-reprezentációk

A következő két fejezetben majd látni fogjuk, hogy az adatmodell szerkezete és az ahhoz tapadó korlátok néha meglehetősen összetettek. Emiatt felmerül a kérdés, hogy milyen módon célszerű megfogalmazni az adatmodellt. Ezen a téren ugyanis sok visszásságot tapasztalunk. Ez azért van, mert sem az adatbáziskezelő gyártók, sem az adatkezelők felhasználói nem látják az adatmodell többszörös célját.

Az adatmodell végső célja az, hogy a leendő adatbázis létrehozásához szolgáló műveleteket és a programozást elősegítse. Az adatmodell alapján definiáljuk az üres számítógépes adatbázist és a modell mutatja, hogy a programozó milyen ismereteket és milyen összefüggésben kezelhet. Azonban az adatbázis bevezetéséig még sok teendő van. Az előzetesen felvázolt modell szolgál arra, hogy a felhasználó igényeit illetve maguknak a valós jelenségeknek az összefüggéseit jobban megismerhessük. Az adatmodell nem azonnal, hanem több kísérlet eredményeképpen születik meg. Az adatbázisalkotás nélkülözhetetlen lépése a ***modellelemzés***. Ennek során a felhasználói igényeket amúgy már kiszolgálni képes modellt optimalizáljuk. Vagyis megkeressük azt az adatmodellt, amely nem csak teljes, hanem egyben egyértelmű és minimális is.

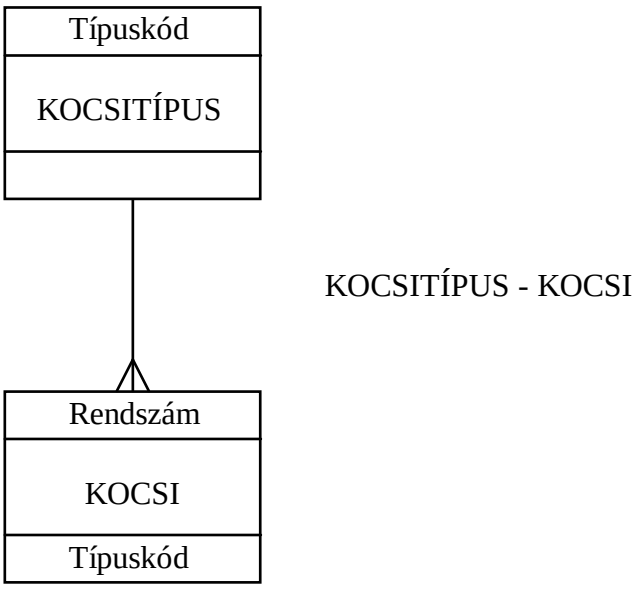
A fenti három cél közül a fejlesztők az elsőre - a szinte legkevésbé fontosra helyezik a hangsúlyt. Ennek közvetlen következménye, hogy az adatmodellt a rendelkezésre álló kezelő képességeinek és konvencióinak megfelelően ***listák*** halmazában fogalmazzák meg. A listák szükségesek és hasznosak, de messze nem elegendők az adatmodell hű tükrözésére. Számos olyan korlát van az adatmodellben, amelyet az adott kezelő nem tud figyelembe venni. Ezek rendre

kimaradnak a modellből és csak programspecifikációk szintjén kerülnek meghatározásra. Így a felhasználó azokat szinte nem is látja, nem ellenőrizheti.

Köztudott, hogy a fejlesztők nem szeretnek írni. Ennek ellenére igen nagy szükség lenne az adatmodell tartalmas, de nem bő - mert akkor nem olvassák el - verbális ***leírására***. A szöveges modell-leírás alapján győződhetne meg a felhasználó arról, hogy ismeretigényeit - főleg a listákban nem megadható korlátokat - helyesen értelmezték a fejlesztők.

Azt viszont nem árt tudni, hogy a viszonylag egyszerű szerkezetű adatbázisokban is több tucat egyedtípus szerepel. Összetettebb alkalmazásokban az ismeretekkel tükrözni kívánt jelenségfélék száma százakra, sőt ezrekre tehető. Nincs olyan fejlesztő vagy felhasználó, aki át tudná látni a modell lényegét és képes lenne firtatni annak valamelyik kényes pontját pusztán listák és leírások alapján. Az adatmodell áttekintő elemzéséhez másféle megfogalmazási forma is szükséges.

Ezt a tényt már a hetvenes években felfedezték. Amiként az építészek és a majdani lakók sokkal jobban megértik a leendő ház képét pár ábra alapján, úgy a leendő adatbázis lényege is tömörebben és világosabban megfogalmazható az ***adatmodelldiagramok*** segítségével. Ezek az rajzok Bachman nevéhez fűződnek [14] és az ő nyomdokán hívjuk őket Bachman-diagramoknak. A következő fejezetekben mi is alkalmazni fogjuk ezt az adatmodell-tükrözési formát. Ezért most, előzetesen, bemutatjuk annak legfontosabb konvencióit, vagyis megegyezéses jeleit az 5.7 ábra segítségével.



5.7 ábra: Az adatmodell-diagram alapvető konvenciói

Az ábrán téglalapok jelölik az ***egyedtípusokat***. A téglalap mérete mindkét irányban tetszőleges. Az eredeti konvenció szerint a téglalap közepén található az egyedtípus neve. (Mai divat, hogy az egyedtípusok jelölésére legömbölyített sarkú téglalapokat használnak. Ez azért jó, mert sokkal nehezebb rajzolni - egyébként az égadta világon semmi jelentősége sincs. Ezen kívül mindenki mást ért a lekerekítésen.)

A téglalapon belül lehet feltüntetni a ***tulajdonságtípusokat***. Azonban a hosszú tulajdonságlisták eltorzítanák a diagramot és megnehezítenék az áttekintést. Ezért általában csak az azonosítót és az értelmezési/kapcsolati szempontból fontos tulajdonságok nevét szokták bejegyezni a

dobozba. Az azonosítót külön is megjelölik. Mostanában ezt egy tömör fekete ponttal teszik. Ennél jobban bevált a régi metódus, amelynek alkalmazása során az azonosítót a többi tulajdonságtól vonal választja el. A legáttekinthetőbb az 5.7 ábrán mutatott megoldás. Felül szerepel az azonosító, alul a kapcsoló és az egyéb fontos tulajdonság.

A téglalapokat összekötő vonalak tükrözik az egyedek közötti ***kapcsolattípusokat***. A vonal mellé írjuk a kapcsolattípus nevét. A kapcsolattípus jellegzetességeiről, így például a fokáról is, majd csak a következő fejezetben lesz szó. Itt előre jelezzük, hogy a vonal végén lévő „varjúláb" (angolul: crowsfoot) mutatja, hogy az egyik egyedtípus (KOCSITÍPUS) egy előfordulásához a varjúlábas végen lévő egyedtípusnak (KOCSI) több előfordulása tartozhat. A vonal sima vége az egyszeresség jele. Magyarul: egy kocsitípushoz több kocsi tartozhat (varjúláb), de minden kocsinak csak egy típusa lehet (sima vonalvég).

Nem fejezhetjük be ezt a témát két fontos megállapítás nélkül. Az adatmodell reprezentáló erejétől elbűvölt kezdők gyakori hibája, hogy egy diagramon nagyon sok tényezőt tüntetnek fel. Pedig a zsúfolt rajz nem segíti, hanem megnehezíti az értelmezést. Ezért az összetettebb ábrákat részekre kell bontani és ki kell dolgozni az ábralapok közötti összefüggések kifejezésének az egyezményes jeleit. A másik kérdés az irányokkal kapcsolatos. Az európai ember felülről-lefelé és balról-jobbra ír. Általában ennek megfelelően tekinti át az ábrákat is. Ezért a józan ész azt diktálja, hogy a másoknak fölérendelt illetve a „fontos" egyedeket valahol felül, a tőlük függőeket alul helyezzük el az ábrán.

Egy kocsitípushoz több kocsi tartozik, és ezért a fenti diagramon a KOCSITÍPUS doboza a KOCSI téglalapja felett látható. Ez az ábrázolás fejezi ki a két jelenség hierarchikus kapcsolatát akkor is, ha a KOCSI a „fontosabb" egyed. A szervezeti ábrákban a főosztályt jelenítjük meg az osztály felett, nem pedig megfordítva. Mindezt azért említjük, mert egyes mai tervezési segédletek alkalmazási kézikönyveiben pontosan a fordított logikájú reprezentálást ajánlják. Mivel több rendeléstétel van, mint rendelés, szerepeljen az előbbi egyed doboza az utóbbié felett. Ez a megfontolás számunkra természetellenes.

Mindezen persze nincs sok értelme vitatkozni. A lényeg az, hogy meg kell találni az egyedeket reprezentáló dobozok jól áttekinthető elhelyezését. Ha például a SZEMÉLY egyedtípushoz sok alárendelt egyed (NYELVTUDÁS, PÓTLÉK stb.) kapcsolódik, akkor a „baloldalt, alul" ma hirdetett előírás helyett mi a „középen, felül" megoldást javasolnánk. Mert nem a merev fent-lent, baloldalt-jobboldalt megfontolás, hanem az áttekinthetőség a lényeges.

Az adatmodell ábrázolási alapjainak a bemutatása után ideje, hogy áttérjünk magának az adatmodellnek a részletesebb taglalására.

Ellenőrző kérdések - 5

5/01 Van egy kárrendező és egy biztosítási-díj megállapító részlegünk. Az egyik arra kíváncsi, hogy egy káreseményben milyen bajok érték az abban résztvevő kocsikat. A másik abban, hogy egy kocsit milyen károk értek és emiatt miként számolja ki a bonus-malus százalékát. Ön hányféle ismeretekkel leírandó jelenséget lát e feladat mögött? Adjon meg egy számot.

5/02 Mondja meg, hogy melyik megoldás a helyes (H) és melyik a rossz (R) az előző feladatok támogatására:

- A két részlegnek nincs köze egymáshoz. Itt két adatbázis szükséges.
- A feladatok részben közös ismeretekre támaszkodnak. Legjobb lenne két részben átfedő adatszerkezetet kreálni, ügyelve arra, hogy a közös KÁR egyed karbantartása egyszerre történjen meg a két részben.
- Itt csak egyetlen adatbázisnak a két nézetéről van szó.

5/03 Iskolai adatbázist tervezünk az ott oktató tanárokról, a tanulókról és a bedolgozókról. Van aki a tanár által oktatott tárgyakra, más a tanuló által tanulandókra, ismét mások a tantárgyakhoz a segédeszközöket szállító személyekre kíváncsiak. A tanárt a képzettséghez, a diákot a jegyekhez, a bedolgozót az eszközökhöz kötődő ismeretek jellemzik. Ezek az ismeretek kölcsönösen kizáróak (pl. a tanár nem kap jegyet). A kérdés az, hogy milyen módon lehet az egyetlen SZEMÉLY egyedhez kapcsolódó különböző ismereteket mindenki részére a saját maga által igényelt módon megjeleníteni? Adja meg az alábbi változatok közül a helyesnek tartott válasz sorszámát:

- Három különböző SZEMÉLY rekordképet kell tervezni.
- A sajátos igénynek megfelelő szemléletet virtuális egyedben fejezzük ki.
- Saját programmal összebogarásszuk az ismereteket.

5/04 A tanárokról három listát készítünk. Mindegyik a tanár nevét és az általa oktatott szaktárgyak megnevezéseit tartalmazza. Az egyiken a tanár, a másikon a tantárgy sorrendjében. A harmadik a másodikkal azonos, de a tantárgy teljes neve helyett azon csak a név első x jele szerepel. Ön szerint az alábbiak közül melyik állítás igaz (I) és melyik hamis (H):

- Az ANSI szerint három külső nézetet kell megadni.
- Logikai szinten nézve a dolgot valójában csak egy listáról van szó.
- A tantárgynév eltérő hossza csak fizikai szintű változatot jelent..

5/05 Az alábbi kitételek közül melyik igaz (I), melyik hamis (H):

- Az adatmodell nem más, mint a Bachman-diagram.
- Az adatbázis tervét rekordképekben kell megadni.
- Az adatbázis tervét jó bő szövegben le kell írni, hogy a felhasználó értse.
- Lista, szűkszavú leírás és pár áttekinthető rajz a modell veleje.

6. A MODELL ALAPVETŐ SZERKEZETE

6.1 Az adatmodell, mint rendszer

Az adatbázis az alkalmazási környezet egyed-, tulajdonság- és kapcsolat-előfordulásainak az adatmodell szerint szervezett együttese (vö. D 4/8). Mindeddig a „szervezett” jelzőn csak valamiféle tudatos elrendezést értettünk, de nem magyaráztuk el, hogy az valójában mit is takar. Ha az adatbázis lényegét fel akarjuk tárni, akkor az említett meghatározásnak arra a kitételére kell támaszkodnunk, amely szerint az adatok együttese „az adatmodell szerint” szervezett. Ebből pedig az következik, hogy az adatbázis megértéséhez az adatmodellt kell jobban megismernünk.

Az adatmodellről jelenleg csak két dolgot tudunk. Először is azt, hogy a fogalmi modellt egyed-, tulajdonság- és kapcsolattípusok alkotják (vö. D 4/7). Másodszor pedig azt, hogy ezek a tényezők egymással viszonyokban állnak. Azonban ezeknek az összefüggéseknek a természetét egyelőre még nem ismerjük. Ezért az egyetlen biztos támpontból kell kiindulnunk: mivel az adatmodell elemekből és azok viszonyaiból áll, az adatmodellt ***rendszerként*** kell felfognunk. A rendszereket pedig általában úgy ismerhetjük meg, hogy megvizsgáljuk azok elemeinek és az elemek viszonyainak a természetét.

Ebben a fejezetben az adatmodell mint rendszer tényezőivel és e tényezők összefüggéseivel kapcsolatos alapvető tudnivalókat ismertetjük. Végeredményben azt fejtjük ki, hogy mit kell érteni az adatbázis szervezettségén.

Az adatmodellt, amelyről a következő pontok szólnak, típusok alkotják. Ezért a továbbiakban - a rövidség kedvéért - el fogjuk hagyni a „típus” és az „előfordulás” megjelöléseket. Ezeket csak az olyan összefüggésekben fogjuk alkalmazni, amelyekben az egyértelműség azt megköveteli. Tehát az alábbiakban például az „egyed” mindig „egyedtípust” fog jelenteni.

Az adatmodell ebben a könyvben mindig ***fogalmi szintű*** modellt takar. Ezért nem törődhetünk a mai adatkezelő rendszerek logikai illetve fizikai szintű adatszerkezeti korlátaival és nem vehetjük át azok terminológiáját. Így előfordulhat, hogy egy konkrét adatkezelő ismereteiben jártas olvasó olyan megoldásokkal és fogalmakkal találkozik e fejezet során, amelyek az általa alkalmazott rendszerben nem használatosak. Sőt az is megtörténhet, hogy saját kezelőjének a képességei az általános adatmodellezési elveknek ellentmondanak. Ilyenkor a hibát nem a fogalmi modellben, hanem az adott rendszer specifikus korlátaiban kell keresni.

Az adatmodellezésben járatlanok azt hiszik, hogy az adatmodell egy az egyben megvalósítandó strukturális terv. Ez tévedés. Az adatmodell rengeteg nem-szerkezeti korlátot is rögzít. Ezt a tervet le kell képezni logikai, majd fizikai szintűre a tényleges megvalósítás előtt. Előbb azonban látnunk kell, hogy mi is az, amit meg kellene tudnunk valósítani.

6.2 Az egyedek kétféle struktúrája

Az adatmodell háromféle tényezője (egyed-tulajdonság-kapcsolat) egyenrangú lényegek. Egyikről sem jelenthetjük ki, hogy fontosabb a másiknál. Ezért ha az alábbi kitételekben az adatmodellt az egyed oldaláról nézzük, annak egyetlen oka az, hogy valamelyik tényezőnél meg kell ragadnunk ezt a rendszert. Vizsgálataink szemléltetése céljából érdemes visszatérnünk a nyelvtani analógiához. Vegyük példaként a következő három mondatot:

6.1 példa

„Az X rendszámú fehér, Lada típusú kocsi Rózsáé."
„A Lada típusú kocsik ötszemélyesek."
„A kis Polski típusú kocsik négyszemélyesek."

Most pedig elemezzük együtt a fenti kijelentéseket! Azonnal észrevesszük, hogy a második és a harmadik kitétel azonos, az első viszont azoktól eltérő mintájú. Ezzel egy lépéssel máris közelebb jutottunk az adatmodell megértéséhez, hiszen

a modell minta, az egyedi, konkrét jelenségek közös, absztrakt vonásainak az együttese.

A második és harmadik példamondatban ugyan eltérő konkrét szavak is szerepelnek, de a szavak általános elrendezése azonos. Úgy is mondhatnánk, hogy a két kitétel azonos „mondattípusba" tartozik. Ezzel szemben az első közlésben megtaláljuk a másik kettő néhány konkrét szavát, de más a szófűzés. Tehát az első mondat a másik kettőtől különböző szerkezetet képvisel.

E felfedezéstől már csak egy lépés az adatmodell első szerkezeti aspektusának a megértése. Vonjunk párhuzamot a mondat és az egyed illetve a szó és a tulajdonság között! A mondat szavakból áll, míg az egyedet tulajdonságok jellemzik. A mondattípusok feltárásához ismernünk kell a szavak mondaton belüli általános rendjét. Ugyanígy az egyedek meghatározásakor ki kell jelölnünk az azokhoz tartozó tulajdonságok elrendezését.

D 6/1 **Az egyed tulajdonságainak a sorát az egyed belső szerkezetének nevezzük.**

Mindennapi közléseinkben az azonos vagy különböző jelenségekre vonatkozó ismereteinket következetesen egymáshoz kapcsoljuk. Ennek során meghatározott logikát követünk. Ellenpéldaként vegyük csak alapul az alábbi két mondatot:

6.2 példa

„Az X rendszámú fehér, Lada típusú kocsi Rózsáé."
„Az X típusú fehér gázkonvektor ára 36000 forint."

Ennek a két kitételnek együtt semmi értelme sincs, bár vannak bennük közös szavak. Viszont a korábbi példánk első két mondatának együttese alapján az olvasó pontosan tudhatja, hogy az X rendszámú gépkocsi ötszemélyes és nem négy férőhelyes. Amint látjuk, nem mindig az önmagában vett mondat hordozza a közlendő ismeret teljes tartalmát. A mondatokat egymásba fűzzük. Ezt nem véletlenszerűen, nem össze-vissza módon tesszük. A teljes ismeret átadásának az érdekében „mondattípusainkat" logikusan kapcsoljuk egymáshoz.

Ez a tény arra int bennünket, hogy az adatmodell tekintetében sem szabad megállnunk az egyedek és a tulajdonságok viszonyainak a vizsgálatánál. Az egyedek közötti összefüggésekre is figyelnünk kell, vagyis meg kell határoznunk az egyedek kapcsolatainak a rendjét.

D 6/2 **Az egyed kapcsolatainak az együttesét az egyed külső szerkezetének hívjuk.**

A fentiekből látható, hogy az adatmodell ***hiperstruktúra***, vagyis szerkezetek szerkezete: az egyedek belső és külső felépítésének a szervezett együttese. Később majd kimutatjuk, hogy az egyed belső és külső szerkezete kölcsönösen egymáson alapul. Ennek az összefüggésnek a feltárásához először az egyedek belső szerkezetét kell alaposabban megismernünk.

6.3 A tulajdonságok alapvető szerepei

Hasonlítsuk össze a következő két kitételt formai és tartalmi szempontból, majd vizsgáljuk meg bennük az egyes mondatrészek feladatát!

6.3 példa

„Az X rendszámú kocsi színe fehér."
„Fehér az X rendszámú kocsi színe."

A két mondatban más a szórend (forma). A mindennapi közlésekben a szórend hangsúlyozási célokat szolgál, de ez az aspektus egyelőre kívül esik érdeklődési körünkön. Ezért elmondhatjuk, hogy a két kijelentés azonos ismeretet közvetít (tartalom). A mondatokban van alany és állítmány (mondatrészek). Ezek mondaton belüli helye, amint látjuk, nem befolyásolja a tulajdonképpeni ismerettartalmat.

Most pedig térjünk át az adatmodellre! Mondanivalónk szemléltetéséhez a korábbi 5.5 ábrát használjuk fel. Lásd a 6.1 ábrát.

A példa szerint a KOCSI egyed ***belső szerkezetét*** a Rendszám-Szín-Típuskód-Tulajkód tulajdonságsor alkotja. A 6.3 példa értelmében a tulajdonságok egyeden belüli sorrendje közömbös, és ezért például a Szín-Rendszám-Tulajkód-Típuskód tulajdonságsort az előzővel azonosnak kell tekinteni a fogalmi modellezés szintjén. Bár a sorrendet tetszőlegesen határozhatjuk meg, egyféle sorrendet le kell rögzítenünk, hogy a tulajdonságértékeket a megfelelő tulajdonságtípusokhoz tudjuk rendelni (vö. adatszerű adatkezelés).

KOCSI

Rendszám	Szín	*Típuskód*	*Tulajkód*
X	Fehér	*T1*	*T1*
Y	Zöld	*T1*	*T2*
Z	Fehér	*T2*	*T3*
Q	Piros	*T3*	*T1*
W	Piros	*T2*	*?*

TULAJDONOS

Tulajkód	Tulajnév	Tulajcím
T1	Rózsa	C1
T2	Gabi	C2
T3	Lajos	C3

KOCSITÍPUS

Típuskód	Típusnév	Férőhely	Fogyasztás	Díjtétel
T1	Lada	5	A	II
T2	Polski	4	B	I
T3	BMW	5	C	IV

6.1 ábra: Mintaadatbázisunk

Mivel a sorrend lényegtelen, a tulajdonságoknak nem az egyeden belüli helyére, hanem az ismeretközlésben ellátott funkciójára kell ügyelnünk (vö. mondatrészek). A tulajdonságnak a belső egyedszerkezeten belüli feladatát a tulajdonság ***szerepének*** nevezzük. Az alany és az állítmány természetes mondatbeli funkciójának megfelelően kétféle szerepről beszélhetünk.

Feltételezésünk szerint a Rendszám tulajdonság értékei a szóbanforgó gépkocsikat egyértelműen elkülönítik egymástól. Az ilyen hivatkozási feladatra képes tételt, amely minden egyedelőfordulásra eltérő értéket vesz fel („X”, „Y” stb.) és így az egyedelőfordulásokat mintegy helyettesíti, az egyedtípus ***azonosító*** szerepű (vö. D 4/5) tulajdonságának nevezzük. Az azonosítót más néven ***elsődleges kulcsnak*** (angolul: primary key) is hívjuk, bár a „kulcs” kifejezés nem éppen fogalmi szintű megjelölés.

Formai megegyezésünk (konvenciónk) szerint az azonosítót kövér dőlt szedettel mutatjuk. Általános szokásként - ez nem szigorú előírás - az azonosítót a tulajdonságsor elejére helyezzük. Ez megfelel annak a mindennapos gyakorlatnak, hogy általában az alannyal kezdjük a mondatot. (N.B.: Az itt elmondottakkal szemben a *fizikai adatszerkezetben* nem közömbös az adattételek sorrendje és abban az azonosítót szinte mindig a tulajdonságsor első tagjaként célszerű felvenni.)

A korábbiakban utaltunk arra, hogy az adatszerkezet mindig bizonyos korlátokat is hordoz magában (vö. 5.4 pont). Ezek a megkötések a szóbanforgó adatmodellezési koncepció sajátjai, így tehát más-más korlátokat támasztanak az eltérő adatbázisfilozófiák. A ma általánosan elterjedt nézetek szerint az azonosító tulajdonságra a következő feltételek vonatkoznak:

K1 **Minden egyednek kell, hogy legyen azonosítója.**
K2 **Az azonosító értéke egyetlen egyedelőfordulásban sem lehet üres/ismeretlen.**
K3 **Minden egyednek csak egy azonosító tulajdonsága lehet.**
K4 **Ugyanaz a tulajdonság csak egyetlen egyednek lehet az azonosítója.**

Hangsúlyozzuk, hogy a felsorolt korlátok a fogalmi szintű adatmodellezés megkötései; a logikai és fizikai adatszerkezet szintjén nem kell feltétlenül érvényesülniük. Az első két kitétel voltaképpen azt mondja ki, hogy minden egyedelőfordulást meg kell tudni különböztetni egymástól. A K2 korlát valójában önmagától értetődő, hiszen az azonosító meghatározásából egyenesen következik. A K3-K4 megkötés az egyedtípusok és az azonosító tulajdonságtípusok kölcsönös és egyértelmű megfeleltetésének a szabályát rögzíti. A többszörösség (redundancia) és a következetlenség (inkonzisztencia) elkerülésének az érdekében a valós jelenségek egyszeres és ellentmondásmentes tükrözését szolgálja.

Most pedig térjünk át a másik alapvető szerepre. Az „X” és a „Z” kocsi színe fehér, a „Q” és a „W” kocsi színe piros. Tehát a KOCSI egyed Szín tulajdonsága több egyedelőfordulásban is azonos értékű lehet. Azokat a tételeket, amelyeknek az értéke az egyedelőfordulásokra nézve nem egyedi, ***leíró*** szerepű tulajdonságoknak hívjuk. Minden egyedhez tetszőleges számú leíró tulajdonság köthető. A leírók sorrendje a fogalmi modellezés szintjén teljesen közömbös. (N.B.: A fizikai adatszerkezet szintjén ezt a sorrendet is meg kell fontolni.) A leíró tulajdonság értéke üres vagy ismeretlen is lehet. Példánkban, első megközelítésben, a Típuskód és a Tulajkód tulajdonság ugyancsak leíró. Az utóbbi értéke a „W” egyedelőfordulásban ismeretlen.

Összegezzük az eddig elmondottakat: Az egyedtípus belső szerkezete nem homogén. A természetes mondatokban megkülönböztetjük az alanyt illetve az állítmányt és egy mondaton belül lehet több állítmányunk is (vö. a 6.1 példa első mondatával). Ugyanígy az egyed tulajdonságsorában különbséget teszünk azonosító és leíró szerepű tulajdonságok között. Az egyednek mindig van azonosítója, csak egy azonosítója van, viszont lehet több leírója is.

6.4 Abszolút és relatív szerep

A 6.1 ábra alapján könnyen meg tudjuk határozni az „X” rendszámú kocsi tulajdonosának a címét. Ehhez a Tulajkód értékét használjuk fel. Megállapítjuk, hogy a tétel tartalma a KOCSI egyed „X” sorában „T1”. Most átlépünk a TULAJDONOS egyedbe. Kiválasztjuk azt a sort,

amelyben a Tulajkód értéke szintén „T1”. Kikeressük az adott sorban a Tulajcím tartalmát. Most már tudjuk, hogy a vonatkozó kocsi tulajdonosának a címe „C1”.

Az előző bekezdésben utánoztuk az adatbáziskezelő működését. Képzeletbeli kezelőnkkel hasonló módon tudnánk kielégíteni a fordított irányú kérdéseket is. Például kikereshetnénk azt, hogy milyen rendszámú kocsik vannak Rózsa birtokában. Ennél a feladatnál is arra a tényre alapozunk, hogy adatbázisunkban a KOCSI és a TULAJDONOS egyednek van olyan közös tulajdonsága (Tulajkód), amely a két egyed előfordulásaiban azonos értéket („T1”) vesz fel.

Az előző két feladatot azért tudtuk könnyen megoldani, mert már az adatbázis felépítésénél gondoskodtunk arról, hogy a kocsik és a tulajdonosok adatai között átmenetet biztosítsunk a Tulajkód tételen keresztül. Azonban az adatbázisok kialakításánál aligha tudunk előre megfontolni minden lehetséges kérdésváltozatot. Még szerencse, hogy erre nincs is szükség. Az adatbázist a kérdések ismeretének a hiányában is tökéletesen meg tudjuk szerkeszteni. Ehhez csak az egyedek külső szerkezetének a természetét kell ismernünk.

Bizonyára már eddig is feltűnt, hogy a Tulajkód egyszerre két funkciót lát el. A KOCSI egyedben leíró, a TULAJDONOS-ban viszont azonosító szerepű. Most vessük össze a kocsik Tulajkód és Szín tulajdonságát! Eddigi ismereteink szerint mindkét tétel leíró, ámde a Tulajkód tételnek a Színnel szemben egyéb feladata is van. A Szín értékei nem vezetnek bennünket sehová, míg a Tulajkód értékei alapján közlekedni tudunk a KOCSI és a TULAJDONOS egyed között.

Az olyan tulajdonságot, amely az egyik egyedben azonosító, a másikban leíró feladatot tölt be, az utóbbi egyedben ***kapcsoló*** szerepű tulajdonságnak nevezzük. Az ilyen tételt külső vagy ***idegen kulcsnak*** (angolul: foreign key) is hívjuk, mert az adott egyedben nem elsődleges kulcs, de valamelyik másik egyedben az. A kapcsoló tulajdonságokat ábráinkon dőltbetűvel mutatjuk.

Most már részben érthető az a fentebbi kitétel, amely szerint a Tulajkód a KOCSI egyedben „első megközelítésben” leíró szerepű. Hiszen ez a tulajdonság a valóságban, a „második megközelítésben” kapcsoló. Mindez nem puszta játék a szavakkal. A Szín tétel ma leíró. Ha holnap létrehozunk egy SZÍN nevű egyedet, amelynek kulcsa a Szín tulajdonság, akkor ennek szerepe a KOCSI egyedben kapcsolóvá válik. Megfordítva: Ha megszüntetjük a TULAJDONOS egyedet, akkor a Tulajkód szerepe a KOCSI-ban leíróvá minősül vissza.

A fentiekből kitűnik, hogy a szerepek nem azonos fontosságúak. Az adatmodell tervezésekor gondolnunk kell arra, hogy az azonosítók nemcsak ezt a funkciót látják el, hanem az egyedek közötti átmenet eszközei is. Ha a leíró tulajdonságokat megszüntetjük vagy sajátosságaikat módosítjuk, akkor ez a beavatkozás csak az érintett egyed adott tulajdonságát kezelő programokra lesz kihatással. Ezzel szemben ha azonosító tulajdonságot számolunk fel vagy változtatunk, akkor megszűnnek illetve módosulnak az egyedek közötti viszonyok is. Ezért nemcsak az azonosítót kezelő, hanem minden olyan programot át kell írni, amely kapcsolóként használja a kérdéses tulajdonságot.

Ezért az adatmodell szilárdságának az érdekében meg kell különböztetnünk a tulajdonságok szerepeinek a minőségét:

D 6/3 **A tulajdonságnak az egyeden belül ellátott funkcióját relatív szerepnek, legfontosabb relatív feladatát pedig abszolút szerepnek nevezzük.**

A Szín tulajdonság relatív szerepe a KOCSI egyeden belül leíró. Mivel ez a tulajdonság a modellben másutt nem fordul elő, az egyetlen szerepe egyben a legfontosabb is. Tehát a Szín abszolút szerepe is leíró. Ezzel szemben a Tulajkód tételnek két relatív szerepe van. A TULAJDONOS egyedben azonosító, a KOCSI egyedben kapcsoló. A fontosabb szerep az azonosító, és ezért a Tulajkód abszolút szerepe is az. A Szín abszolút leíróval bármit tehetünk. Viszont a Tulajkód abszolút azonosító bármilyen megbolygatása az adatbázis komoly átalakítását és a programok tekintélyes módosítását vonja maga után.

A kapcsoló relatív szerepű tulajdonságok fejezik ki az egyedek közötti viszonyokat. Tehát korábbi sejtésünknek megfelelően az egyedek belső és külső szerkezete szorosan összefügg. Az adatmodell három tényezője egymásba fonódik, amit jól mutat a következő (korlátjellegű) meghatározás is:

D 6/4 **Két egyed akkor és csak akkor áll kapcsolatban egymással, ha az egyik kapcsoló szerepű tulajdonságként tartalmazza a másik azonosító szerepű tulajdonságát.**

Ezt a definíciót a következő fejezetben majd módosítani - pontosítani - fogjuk. Most arra hívjuk fel a figyelmet, hogy az egyed belső és külső szerkezete kölcsönös viszonyban áll egymással. Nem mondhatjuk, hogy a belső szerkezet határozza meg a külsőt, mert sokszor éppen a külső szerkezet elemzése vezet a belső átalakítására. Az adatbázisalkotók néha elfeledkeznek erről a fontos tényről. Egy példával világítjuk meg tévedésüket.

Tételezzük fel, hogy a tervező elfelejtette betenni a Tulajkód tulajdonságot a KOCSI egyed tulajdonságsorába! Ha tudatos modellező, akkor az egyed külső viszonyainak az elemzésénél rádöbben, hogy a kocsik tulajdonosokhoz tartoznak. Következésképpen a két egyed között kapcsolatot kell teremteni. Ezt úgy tesszük, hogy az egyik egyed azonosítóját kapcsolóként a másik egyed tulajdonságsorába illesztjük. Tehát a KOCSI egyed belső szerkezetét a külső szerkezetnek megfelelően a Tulajkód tétellel bővítjük.

Lám, a fentebb kétellyel fogadott kitételt máris igazoltuk. Korábban azt mondtuk, hogy nem kell előre ismerni minden lehetséges kérdést az adatbázis helyes felépítéséhez. Ez így is igaz. Ha megtaláljuk a jelenségek valós összefüggéseit, akkor azokat konkrét kérdések nélkül is kifejezhetjük az adatmodellben. A nem-valós viszonyok pedig sohasem lesznek kezelhetők. Erre a kérdésre majd a 6.6 pontban térünk vissza. Most meg kell fogalmaznunk egy korlátot:

K5 **Az adatmodellben kell, hogy legyen kapcsoló tulajdonság.**

Ez a korlát axiomatikus, mert az adatbázis és a kapcsolat definíciójából következik. Ugyanis nem tekintjük adatbázisnak az olyan adatok halmazát, amelyen nem tudunk meghatározni legalább két olyan egyedtípust, amely egymással kapcsolattípust létesít a kapcsoló tulajdonságtípuson keresztül. Ha van KOCSI és TULAJDONOS állományunk, de a kocsik nem köthetők a tulajdonosokhoz és fordítva, akkor a két állomány együtt nem tekinthető adatbázisnak.

6.5 Hierarchikus inhomogén kapcsolatok

Mint a későbbiekben látni fogjuk, az egyedviszonyoknak számos válfaja van. Ezek közül a legegyszerűbbek és egyben leggyakoribbak a hierarchikus inhomogén kapcsolatok.

Az „inhomogén” (külön nemű) jelző könnyebben érthető. A 6.1 ábra alapján két kapcsolatot tudunk meghatározni. A TULAJDONOS - KOCSI és a KOCSITÍPUS - KOCSI viszonyokról van szó. (N.B.: A kapcsolatokat úgy fogjuk jelölni, hogy kötőjellel fűzzük össze a kapcsolt egyedek neveit. Ez a konvenció csak a jelenlegi ismertetésben érvényes. Egyébként a kapcsolatoknak bármilyen nevet lehet adni.) A két viszonyban különnemű dolgokat hozunk összefüggésbe: a kocsikat a tulajdonosaikhoz és típusaikhoz rendeljük.

A „hierarchikus” megjelölés kapcsán már több sajátosságra kell rámutatnunk. Ilyen viszonyról akkor beszélünk, amikor az egyik egyedtípus egy-egy előfordulásához a másik egyedtípusnak több előfordulása tartozhat, de ez fordítva nem igaz. A „hierarchikus” jelző egyszerre fejez ki

mennyiséget és minőséget. Az egyik egyed ***1*** előfordulásához a másik egyed több, ***N*** előfordulása kötődhet, és emiatt az ilyen kapcsolatokat ***1:N fokúnak*** („egy-az-enes”-nek) is nevezzük. Azt az egyedet, amelyben a kapcsolatot kifejező közös tulajdonság azonosító szerepű ***fölérendeltnek***, azt, amelyben kapcsoló szerepű ***alárendeltnek*** hívjuk. Végül az ilyen összefüggésre sokszor használjuk a ***birtoklási*** (angolul: „has a”) megjelölést is.

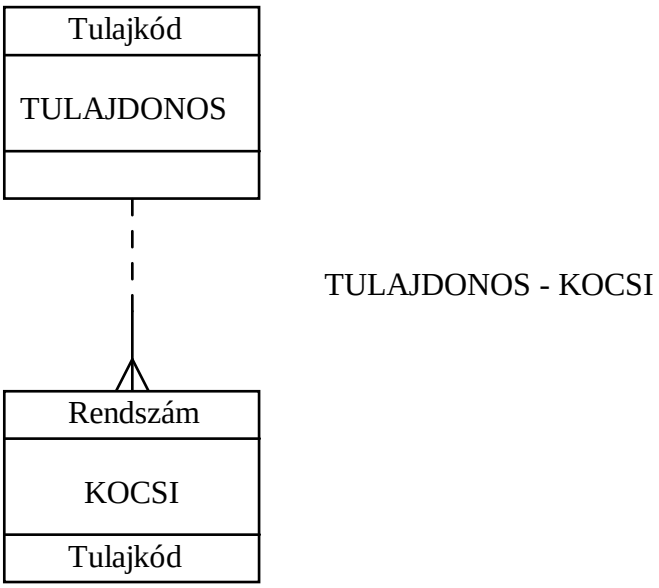
Példánk feltételezése szerint egy tulajdonoshoz több kocsi tartozhat, de egy kocsinak csak egy tulajdonosa lehet. Tehát ez a viszony kétségtelenül 1:N fokú. A kapcsolatban a TULAJDONOS egyed a fölérendelt, a KOCSI az alárendelt. (N.B.: A kapcsolat megnevezésében a fölérendelt egyed nevét szokás előre helyezni, így: TULAJDONOS - KOCSI.) Ebben a példában még a birtoklási megjelölés is érthető, hiszen a tulajdonos birtokolja a kocsit. A KOCSITÍPUS - KOCSI viszonnyal kapcsolatosan a fenti kitételeket megismételhetnénk. Ez az összefüggés is hierarchikus. Legfeljebb azt furcsálljuk, hogy miért „birtokolja” a kocsitípus a kocsit. Azonban ezzel a szakzsargonnal ki kell békülnünk.

A kapcsolat fokára még vissza kell térnünk. A fok nem kötelező, hanem megengedő korlát a viszony mindkét irányát tekintve. Az „egy tulajdonoshoz több kocsi tartoz***hat***” kitétel nem jelenti azt, hogy minden tulajdonosnak kell, hogy legyen kocsija. Az N értéke ***0,1 ... X*** lehet. Ez azt jelenti, hogy az adatbázisban nyilvántarthatunk olyan „tulajdonosokat” is, akik időlegesen nem rendelkeznek kocsival. (Ha viszont az adatbázisban sohasem lehet olyan tulajdonos, akinek a birtokában egynél több kocsi van, akkor nyilvánvalóan értelmetlenség 1:N fokú - az N lehet egynél több is - kapcsolatról beszélni.)

A fokot a fordított irányból is nézhetjük. A „minden kocsit csak egy tulajdonos birtokol***hat***” kijelentés gyakorlatilag és ebben a példában valóban így is értendő, mert nincs kocsi tulajdonos nélkül. Általánosan és elvileg az 1 valójában ***0 vagy 1*** értéket jelent. Ezt a kitételt a példa kibővítéséből érthetjük meg. A kocsik adott biztosítónál kötött CASCO kötvényéről van szó. A BIZTOSÍTÓ - KOCSI viszony hierarchikus, mert egy társaságnál több kocsit biztosítanak, de egy kocsinak *legfeljebb* egy CASCO kötvénye lehet. Viszont vannak olyan járművek is, amelyekre nem kötöttek CASCO-t. Tehát ezekhez „0” biztosító tartozik.

A fentiek szerint a viszony foka nem tévesztendő össze az ún. ***opcionalitással***. A kapcsolat mindig kétirányú összefüggés, ezért az opcionalitást is két oldalról kell meghatározni. A viszony fölülről (a fölérendelt felől) és alulról (az alárendelttől nézve) opcionális vagy kötelező lehet. Minden kocsinak van tulajdonosa (alulról kötelező), de nem mindegyiknek van CASCO-ja (alulról opcionális). Csak olyan kocsitípus ismereteit vezetjük, amelyhez konkrét kocsi tartozik (fölülről kötelező), ezzel szemben azoknak a tulajdonosoknak az adatait is kezeljük, akiknek nincs jelenleg kocsijuk (fölülről opcionális).

Az opcionalitás az adatmodell kapcsolati tényezőjének olyan szerkezeti jellemzője, amely egyben sajátos modellkorlátként is szolgál. A 6.1 ábra példájának a tervezője erre a tényre nem figyelt, hiszen a „W” rendszámú kocsi tulajdonosát nem ismerjük. A tervező elfelejtette, hogy az opcionalitás a majdani tényleges adatkezeléssel függ össze. Ha a kapcsolat kötelező, akkor nem illeszthető az adatbázisba olyan kocsi (bevitel), amelyben a Tulajkód értéke ismeretlen vagy nem felel meg egy tulajdonos egyedelőfordulás Tulajkód értékének. Ha a kapcsolat kötelező és kivesszük az adatbázisból a „T1” tulajdonost, akkor meg kell szüntetnünk minden olyan KOCSI egyedelőfordulást is, amelyben a Tulajkód értéke szintén „T1” (törlés). Végül a kötelező kapcsolatot hordozó tulajdonság értéke nem változtatható üresre (módosítás).



6.2 ábra: Hierarchikus inhomogén kapcsolat

A hierarchikus inhomogén kapcsolat általános modelljét a 6.2 ábra mutatja. A két egyedtípust jelképező dobozt összekötő vonal tükrözi magát a kapcsolattípust. A vonal egyik végén van csak „varjúláb" - innen tudjuk, hogy a kapcsolat 1:N fokú. A viszony opcionalitását folyamatos illetve szaggatott vonallal mutatjuk. A folyamatos oldalon a kapcsolat kötelező (minden kocsinak van tulajdonosa), a szaggatott oldalon opcionális (nyilvántarthatunk olyan tulajdonost is, akihez nem tartozik aktuálisan ismert konkrét kocsi). Az általunk alkalmazott konvencióban (ld. 5.8 pont) az azonosító tulajdonság neve az egyedé fölött, a kapcsolóé az alatt szerepel. Így a kapcsolat vonalát követve egyértelműen megállapíthatjuk, hogy melyik tulajdonság hordozza a viszonyt és modellünk teljes-e. Egyszerűen nem feledkezhetünk meg arról, hogy a KOCSI egyedben fel kell tüntetni a Tulajkód tulajdonságot.

6.6 Hálós egyedviszonyok - 1

Egy gondolatkísérlettel folytatjuk az adatmodell szerkezetének a megvilágítását. Mi lenne, ha a kocsik adataira egyáltalán nem lennénk kíváncsiak és csak azt akarnánk megtudni, hogy a tulajdonosok milyen típusú kocsikat birtokolnak illetve megfordítva, adott típusú kocsik milyen tulajdonosoknak a birtokában vannak? Miképpen kellene átalakítani a 6.1 ábra adatbázisának a szerkezetét?

A dolgok természete szerint egy tulajdonosnak különböző típusú kocsijai lehetnek, mint például Rózsának. Megfordítva: egy kocsitípus több tulajdonoshoz kapcsolódhat, hiszen többeknek lehet például Lada kocsijuk. Számszerűen fogalmazva: a TULAJDONOS és a KOCSITÍPUS egyed viszonya mindkét oldalról nézve 1:N fokú. Mivel nem valószínű, hogy az „N" mértéke a két irányban azonos, az ilyen összefüggéseket ***M:N fokú*** viszonyoknak nevezzük. Még véletlenül sem hívjuk őket kapcsolatoknak, mert ezt a fogalmat az 1:N fokú viszonyok megjelölésére

tartottuk fenn. Az „em-az-ennes” viszonyokban nem léteznek fölé- és alárendeltek. Az összefonódások többszörösek. Ezért az ilyen összefüggéseket ***hálós*** viszonyoknak hívjuk.

A kérdés az, hogy az adatmodellben milyen tényezőre alapozzuk az ilyen viszonyt? A hierarchikus viszonyoknál (pl. TULAJDONOS - KOCSI) a közös tulajdonság (Tulajkód) testesítette meg a kapcsolatot. Vajon működik-e ez a megoldás a hálós viszonyok esetében is? Mi lenne, ha például a TULAJDONOS egyedbe felvennénk a Típuskód tulajdonságot azért, hogy kikereshessük a tulajdonosok által birtokolt kocsitípusokat?

Ez a megoldás elméletileg hibás és gyakorlati megvalósítása is nehézségekbe ütközik. Elvileg rossz, mert felvetődik a kérdés, hogy miért a TULAJDONOS egyedhez kötjük a Típuskód tulajdonságot és miért nem a KOCSITÍPUS egyedhez a Tulajkód tételt? Bármelyik megoldást is választjuk, az adatkezelés „előítéletes” (angolul: biased) lesz, azaz az egyik irányú kezelést a másik rovására kedvezményezi. Vagy a tulajdonosok kocsitípusait, vagy a kocsitípusok tulajdonosait tudjuk hatékonyan kikeresni; a másik irányú keresés csorbát szenved. Ha viszont mindkét megoldást alkalmazzuk, akkor az nyilvánvaló redundanciára vezet, mert kétszer tükrözzük azt az egyszeres tényt, hogy a tulajdonos adott típusú kocsival rendelkezik.

A gyakorlati gondot a számok okozzák. Mivel egy tulajdonosnak különböző típusú kocsijai lehetnek, a Típuskód tulajdonság ugyanarra az egyedelőfordulásra több értéket vehetne fel, amint azt a 6.3 ábra mutatja.

TULAJDONOS

Tulajkód	Tulajnév	Tulajcím	Típuskód
T1	Rózsa	C1	T1
			T3
T2	Gabi	C2	T1
T3	Lajos	C3	T2

6.3 ábra: Ismétlődő értéket tartalmazó egyed

D 6/5 **Azokat a tulajdonságokat, amelyek egy egyedelőfordulásra több értékkel is rendelkezhetnek, ismétlődő tulajdonságoknak hívjuk.**

A bajt a „több” okozza, mert „több”-ként a maximumot kell megadni. Ez pedig pazarlásra vezet. Annál a tulajdonosnál, akinek csak egy kocsija van, a második és a további ismétlés tartalma üres lesz. Az adat feleslegesen foglal tárolóhelyet és növeli a feldolgozási időt. (N.B.: Egyes szoftverekről azt állítják, hogy az üres adat nem igényel tárt és nem befolyásolja a kezelés időtartalmát. Ez egész egyszerűen általtás.) Ráadásul ha a maximumot alábecsültük, akkor az adatbázis átszervezésére lesz szükség. Tegyük mindehhez, hogy egyes rendszerek egyáltalán nem tudják kezelni az ismétlődő adatokat. Olyan pedig végleg nincs, amely az ismétlődés szélsőséges értékeivel megbirkózna. Például azzal a konkrét problémával, hogy csak egy tulajdonosnak van Lamborghini kocsija, míg pár százezernek Ladája. Miképpen lehetne így a Tulajkód adatot felvenni a KOCSITÍPUS egyedbe?

Az ismétlődő tulajdonság (angolul: repeating attribute) által okozott elméleti és gyakorlati problémák miatt fogalmazzuk meg a következő korlátot:

K6 **Az egyedekben nem szerepelhetnek ismétlődő tulajdonságok.**

Most talán már érthető, hogy a jelenségek közötti M:N fokú viszonyokat miért nem nevezzük kapcsolatoknak. Az ilyen összefüggések nem fejezhetők ki közös tulajdonság segítségével. Más megoldáshoz kell folyamodnunk. Egészen kézenfekvőnek tűnik például egy újabb egyed alkalmazása a 6.4 ábrának megfelelő módon.

A 6.4 ábra BIRTOKOLJA egyede remekül megoldja kapcsolási gondjainkat. A KOCSI-TÍPUS felől indulva a Típuskód segítségével kikeressük a megfelelő BIRTOKOLJA előfordulást. Megállapítjuk a Tulajkód értékét, amelynek alapján kikereshetjük a vonatkozó TULAJDONOS előfordulást. Analóg módon tudjuk kikeresni a bennünket érdeklő tulajdonos kocsitípusainak az adatait. Tehát ez az adatmodell egyensúlyos, nem előítéletes. Nincs gondunk az ismétlődő értékek darabszámának a megállapításával sem. A BIRTOKOLJA egyednek tetszőleges számú előfordulása lehet, abban csak a tényleges összefüggéseket őrizzük és az üres értékek nem foglalnak helyet. Végül a 6.4 ábra modellje minden normális rendszerrel kezelhető.

KOCSI-TÍPUS

Típuskód	Típusnév	Férőhely	Fogyasztás	Díjtétel
T1	Lada	5	A	II
T2	Polski	4	B	I
T3	BMW	5	C	IV

TULAJDONOS

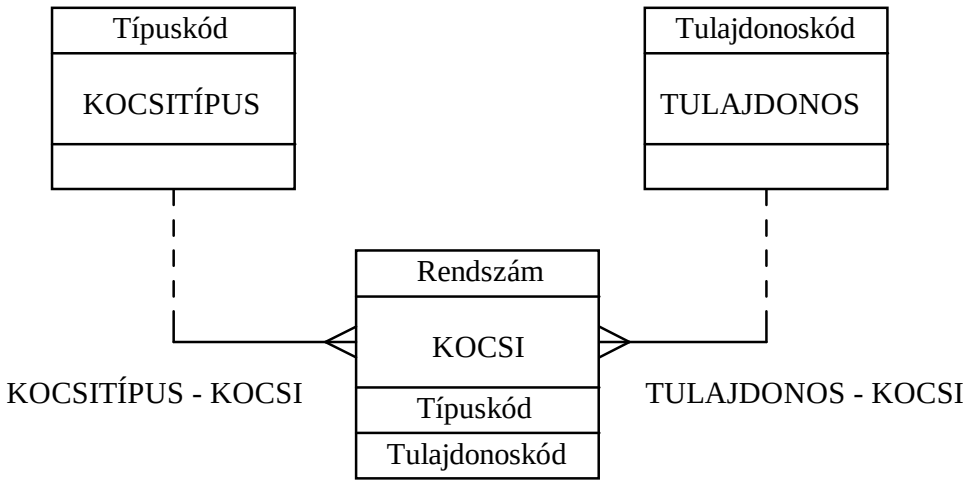
Tulajkód	Tulajnév	Tulajcím
T1	Rózsa	C1
T2	Gabi	C2
T3	Lajos	C3

BIRTOKOLJA

Típuskód	***Tulajkód***
T1	***T1***
T1	***T2***
T2	***T3***
T3	***T1***

6.4 ábra: Az ismétlődés feloldása új egyedtípussal

A 6.5 ábra mutatja az M:N fokú viszonyok célszerű modellezését. Amint látjuk, az „em-az-ennes” viszonyt egy harmadik egyed közbeiktatásával két 1:N fokú kapcsolattal valósítottuk meg. Ez a végső és helyes megoldás.

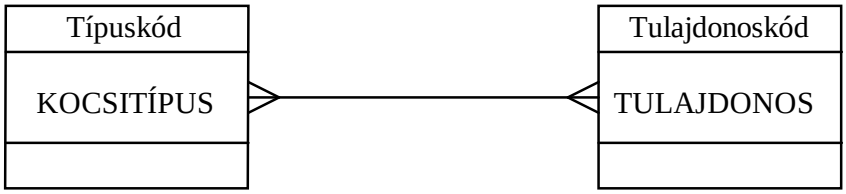


6.5 ábra: Hálós egyedviszony

A fentiekből azt a következtetést vonhatjuk le, hogy az adatmodellben lévő egyedek között kétféle viszony létezhet. Vannak tulajdonságon alapuló ***közvetlen*** kapcsolatok és vannak harmadik egyeden keresztül megvalósuló ***közvetett*** viszonyok. Tehát amennyiben két egyed M:N fokú viszonyban áll és összefüggésüket ki akarjuk fejezni, akkor esetleg egy további egyedet kell

meghatároznunk. Azért „esetleg”, mert ha már létezik a két egyed viszonyát megteremtő harmadik egyedünk - mint esetünkben a KOCSI -, akkor persze az új tényező felesleges.

Azonban az olvasóban nyilván felmerül a kérdés, hogy amennyiben a fogalmi adatmodell hálós viszonyt nem tartalmazhat, úgy miért beszélünk egyáltalán erről a tényezőről? És miként ábrázolnánk - ha szabadna - az „em-az-ennes” összefüggéseket?



6.6 ábra: A hálós viszony ábrázolása

A végső adatmodellben nem szerepelhet hálós viszony. Azonban az előzetes adatmodell-vázlatokban még nem ismerhetjük minden tényező pontos összefüggését. Azt viszont tudhatjuk, hogy két egyedtípusnak majd kapcsolatban kell állnia egymással. Az előzetes modellben ezt úgy fejezzük ki, hogy mindkét irányban varjúlábas vonallal kötjük össze a két egyedet reprezentáló dobozokat (ld. 6.6 ábra). Vegyük észre, hogy ilyenkor nincs lehetőség a kapcsoló tulajdonságtípus feltüntetésére, azaz bármelyik egyed azonosítójának a másik egyedben leíróként való megismétlésére. Ez logikus is, hiszen a hálós viszonyokat nem tulajdonságtípus, hanem külön egyedtípus reprezentálja (ld. 6.5 ábra) a végső adatmodellben.

6.7 Hálós egyedviszonyok - 2

Az egymással M:N-es összefüggésben álló egyedek az előző pontban elmondottak szerint sohasem hozhatók közvetlen kapcsolatba. Mindig közvetett, harmadik egyeden keresztül érvényesített viszony testesíti meg az ilyen egyedek kapcsolódását.

Ennek a kitételnek ellentmondani látszik a relációs adatkezelők azon képessége, amely szerint az egymással M:N fokú elvi viszonyban lévő egyedeken is végre lehet hajtani az összekapcsolás (angolul: join) műveletét. Ennek lényegét a 6.7 ábra szemlélteti. A művelet során az egyik egyed (DOLGOZÓ) sorait egyenként kiegészítjük a másik egyed (GÉP) azon sorával vagy soraival, amelyben a közös tulajdonság (Költséghelykód) azonos értéket (pl. K1) vesz fel. Természetesen a közös tulajdonságot az eredmény-egyedben csak egyszer szerepeltetjük.

DOLGOZÓ

Törzsszám	Költséghelykód
T1	K1
T2	K2

GÉP

Gépazonosító	Költséghelykód
G1	K1
G2	K1
G3	K2
G4	K2
G5	K1

6.7 ábra: 1. rész. Két hálós viszonyú egyedtípus

DOLGOZÓ/GÉP

Törzsszám	Gépazonosító	Költséghelykód
T1	G1	K1
T1	G2	K1
T1	G5	K1
T2	G3	K2
T2	G4	K2

6.7 ábra: 2. rész. Az összekapcsolás eredménye

A „T1” törzsszámú dolgozónál a költséghely értéke „K1”. Ezért a „T1-...” értéksort kiegészítettük a gép egyed minden olyan előfordulásának az adataival, amelyben a költséghely értéke szintén „K1”. Ugyanígy járunk el a DOLGOZÓ többi sorával. Világos, hogy az eredmény, a DOLGOZÓ/GÉP egyed nem része a valódi adatbázisnak. Annak tartalma ugyanis teljes egészében megtalálható a másik két egyedben.

Az összekapcsolás látszólag kitűnően sikerült. Látszólag. Ám aki az összekapcsolás ilyen módját is megengedi, az csak a mennyiségekkel van elfoglalva, a minőségekkel nem törődik. Kétségtelen, hogy ***mennyiségileg*** - kombinatorikusan - semmi akadálya sincs annak, hogy a dolgozók és a gépek adatait a Költséghelykód közös értékei alapján egymáshoz kössük. Azonban ***minőségileg*** az ilyen kapcsolás megengedhetetlen, mert félrevezető, hamis ismereteket eredményez. Sajnálattal kell ugyanis közölnünk, hogy a „T1” dolgozónak semmi köze sincs a „G2” géphez, a „T2” dolgozó pedig az életében nem látta a „G3” gépet. Márpedig az eredményegyed összefüggéseket sejtet a megjelölt jelenségpárosok között.

A példa kapcsán levonhatunk egy általános, korlátszerű következtetést. Két egyed közös ***leíró*** szerepű tulajdonságára alapozva sohasem határozhatunk meg egyedviszonyt. A Költséghelykód révén a DOLGOZÓ és a GÉP egyed nem kapcsolható egymáshoz.

Ha és amennyiben valóban kíváncsiak vagyunk a dolgozók és a gépek tényleges viszonyaira, akkor adatmodellünket át kell alakítanunk. A helyes megoldást a 6.8 ábra mutatja.

DOLGOZÓ

Törzsszám	Költséghelykód
T1	K1
T2	K2

GÉP

Gépazonosító	Költséghelykód
G1	K1
G2	K1
G3	K2
G4	K2
G5	K1

6.8 ábra: 1. rész. Az eredeti két hálós viszonyú egyedtípus

HASZNÁLJA

Törzsszám	***Gépazonosító***	...	Műszak
T1	***G1***		M1
T1	***G5***		M3
T2	***G4***		M2

6.8 ábra: 2. rész. Az eredeti két egyedtípus korrekt összekapcsolása

Szemben a fenti álmegoldással, most már tényleg hű képet nyertünk a dolgok valódi összefüggéseiről. Mi több, a DOLGOZÓ és a GÉP M:N-es viszonyát megtestesítő HASZNÁLJA egyed az eredetieken kívül további tulajdonságokkal is leírható (Műszak), ha erre szükség van. Ezért a közvetett viszony bizonyos esetekben még kedvezőbb is, mint a közvetlen. A közvetett viszonyt egyed valósítja meg, tehát lehet saját tulajdonsága. A közvetlen kapcsolatot pedig tulajdonság hordozza, és a tulajdonságnak nem lehet tulajdonsága.

6.8 Újabb tulajdonság szerepek

A 6.4 ábra BIRTOKOLJA és a 6.8 ábra HASZNÁLJA egyedeinek a belső szerkezete eltér a korábban, a 6.1 ábra példájában megszokott biztos, egyszerű és áttekinthető struktúrától. Ott világosan ki tudtuk jelenteni, hogy a KOCSI egyed Rendszám tulajdonsága azonosító, Szín tulajdonsága leíró, míg Tulajkód tétele első megközelítésben leíró, de végül is kapcsoló relatív szerepű. Viszont a HASZNÁLJA egyed esetében csak a Műszak tulajdonságról tudjuk, hogy leíró. Zavarban vagyunk az azonosítási és a kapcsolási funkciót, valamint a másik két tulajdonság relatív szerepét illetően.

Az adatmodellezési gyakorlatban sokszor alkalmazunk két egyed közt összefüggést teremtő ún. ***viszonyegyedeket***. Ilyenkor előfordul, hogy az egyednek nincs olyan tulajdonsága, amely betölthetné az azonosító szerepkört. A K1 korlát viszont előírja számunkra, hogy minden egyednek kell, hogy legyen azonosítója. Az ilyen esetekben a nyelvtani deszkriptív azonosítás (ld. 3.6 pont) mintájára igyekszünk megkeresni a tulajdonságoknak azon kombinációját, amely minden egyedelőfordulásra nézve egyedi értékű.

A BIRTOKOLJA és a HASZNÁLJA egyedek esetében megtaláltuk ezeket az együtteseket. Az előbbiben a Tulajkód és a Típuskód értékeinek az összefűzésével az egyedelőfordulásokat pontosan behatároló azonosítóértékeket kaptunk. Már csak az a kérdés, hogy akkor melyik tulajdonság a BIRTOKOLJA egyed azonosító szerepű tétele? A Tulajkód? A Típuskód? Mindkettő?

Feledkezzünk el a relációs adatkezelők egyik otromba megoldásáról! Az ilyen rendszerekben mindkét tulajdonságot elsődleges kulcsnak kell kijelölni, ami többszörös elméleti hibát jelent. Először: a Tulajkód nem lehet azonosító, mert hiszen értékei nem különbözőek az egyedelőfordulásokra nézve. Másodszor: minden egyednek csak egy azonosító tulajdonsága lehet. Harmadszor: egyes relációs kezelőkben feltételezik, hogy az ilyen módon, a konkatenáció részeként azonosító tulajdonság mindig kapcsoló. Ez pedig súlyos elméleti és gyakorlati tévedés. Igenis létezhetnek olyan azonosítókombinációk, amelyek valamelyik tagja nem mutat másik egyedre.

A fenti zűrzavarral ellentétben a modellelmélet nagyon pontosan és tisztán határozza meg a tényezőket. A több tételből álló ***tulajdonságcsoportokat*** kezelni nem tudó rendszerekkel szemben elismeri az ilyen tényezők jogosultságát. Ennek megfelelően az azonosítókat két osztályba sorolja. Vannak ***elemi*** (angolul: elementary), azaz egyetlen tulajdonságnak megfelelő azonosítók, amilyen például a KOCSI egyed Rendszám tétele. Ezek mellett léteznek olyan ***összetett*** kulcsnak (angolul: compound vagy composite key) nevezett azonosítók is, amelyek több tagból állnak.

Az ilyen tulajdonságokat az összefűzött tételek neveinek az együttesével jelöljük úgy, hogy a nevek közé „+” jelet teszünk. Tehát a BIRTOKOLJA egyednek nem a Tulajkód és nem a Típuskód, hanem a Típuskód+Tulajkód összetett tulajdonság az azonosítója. (N.B.: Az ilyen neveken belül a tagok sorrendje közömbös. Csak a papír természete miatt kényszerülünk arra, hogy a konkatenált tényezők valamelyikét előre írjuk. Tehát a Típuskód+Tulajkód és a Tulajkód+Típuskód ugyanazt az összetett kulcsot jelenti.)

A fentiek szerint a KOCSI azonosító szerepű tulajdonsága a Típuskód+Tulajkód együttese. Az azonosítót alkotó tényezőket az ilyen esetekben, az első megközelítésben, ***kulcsrész*** (angolul: key-part) relatív szerepűeknek mondjuk. Tehát a BIRTOKOLJA egyedben a Típuskód és a Tulajkód relatív szerepe kulcsrész. Azonban észrevesszük, hogy mindkét tulajdonság más egyed felé biztosít kapcsolatot. Ennélfogva, második megközelítésben, kapcsoló relatív szerepet kellene nekik tulajdonítanunk. Ennek a kettősségnek a feloldására az adatmodellezésben két eszközünk is van. Egyrészt emlékezzünk vissza az abszolút szerepre. Mivel a Tulajkód abszolút szerepe azonosító, már tudhatjuk, hogy a BIRTOKOLJA egyedben kapcsolási feladatot is ellát. Másrészt azért, hogy a szerepek megadása ne legyen ilyen implicit, a szerepeket minősítjük. A Tulajkód a KOCSI egyedben ***leíróként-kapcsoló***, a BIRTOKOLJA egyedben ***kulcsrészként-kapcsoló***. Így már a kép teljesen kerek és világos.

Nem bonyolult és felesleges az ilyen sokféle szerep alkalmazása? Nos, egyrészt azért még igen távol állunk a bonyolultságtól, mert egy adatbázis helyes felépítése és működtetése érdekében ezt a pár szerepet igazán megjegyezhetjük. Másrészt a megkülönböztetés egyáltalán nem felesleges. Ezt a következő, axiomatikus (a K2 kitételből következő) korlát is mutatja:

K7 **Az összetett azonosító részei egyetlen egyedelőfordulásra nézve sem lehetnek üres vagy ismeretlen értékűek.**

Ebből a megkötésből természetesen következik egy további modellezési összefüggés is. A 6.1 ábra KOCSI egyedében a Tulajkód leíróként-kapcsoló volt. Értéke lehetett ismeretlen is. Ezért nem volt módunk pontosan megállapítani, hogy a „W” rendszámú kocsinak ki a tulajdonosa. A 6.4 ábra BIRTOKOLJA egyedében a Tulajkód kulcsrészként-kapcsoló. Értéke nem lehet ismeretlen. Általánosabban fogalmazva: a leíróként-kapcsoló tulajdonságokon alapuló kapcsolatok alulról lehetnek opcionálisak is, viszont a kulcsrészként-kapcsoló tételekre épülő viszonyok alulról mindig kötelezőek.

Már csak annyit kell megemlítenünk, hogy néha egy egyedet csak azért határozunk meg az adatmodellben, hogy közvetett viszonyt teremtsünk két másik egyed között. Lehet, hogy az ilyen egyednek egyáltalán nem lesz saját leíró tulajdonsága (ld. BIRTOKOLJA). Azokat az egyedeket, amelyekben csak azonosítórész relatív szerepű tulajdonságok vannak, ***csupa-kulcs*** (angolul: all-key) egyedeknek hívjuk. A gyakorlati tapasztalatok szerint ez a jelleg előbb-utóbb elveszik, mert találunk valamilyen az egyedhez kapcsolandó többletismeretet. Ez történt a HASZNÁLJA egyed esetében.

6.9 A kölcsönös egyedviszony

Az eddigiekben a hierarchikus (1:N fokú) kapcsolatról és a hálós (M:N fokú) viszonyról volt szó. Most már csak a kölcsönösnek (angolul: mutual) nevezett egyedösszefüggést, az ***1:1 fokú viszonyt*** kell bemutatnunk. Feladatunk nem könnyű, mert az ilyen kapcsolódás a gyakorlatban meglehetősen ritka, de ugyanakkor sok elméleti fejtörést okoz.

Gondjaink megértéséhez módosítsuk korábbi példánkat. Éljünk azzal a meglepő feltételezéssel, hogy minden tulajdonosnak legfeljebb csak egy kocsija lehet, miközben továbbra is igaz, hogy minden kocsi csak egy tulajdonoshoz tartozhat. A kérdés az, hogy miképpen kapcsolódik egymáshoz az így módosított TULAJDONOS és KOCSI egyed? Milyen tényezővel fejezzük ki a viszonyukat? Három megoldás kínálkozik.

Néhány modellezési koncepcióban az 1:1 fokú viszonyt úgy, ahogy van, kizárják. Tessék ***összevonni*** a két egyed tulajdonságait egyetlen egyedbe - hirdetik sokan. Gyakorlatilag - az alább

ismertetett gondok miatt - ez bizony vonzó megoldásnak tűnik. Ámde a kocsi és a tulajdonos mégsem azonos lényegek, ezért ez az egyszerű út elméletileg helytelen. Problémát jelent az is, hogy ha a tulajdonosba tesszük a kocsi adatait vagy megfordítva, akkor képtelenek leszünk olyan kocsit illetve tulajdonost nyilvántartani, amelynek nincs tulajdonosa illetve akinek nincs kocsija. Tehát az összevonást csakis két esetben érdemes alkalmazni. Elméletileg akkor, ha két egyedet határoztunk meg, a kettő lényegében azonos jelenséget tükröz és rájöttünk induló tévedésünkre. Gyakorlatilag esetleg akkor, ha egészen biztosak vagyunk abban, hogy a viszony mindkét irányban kötelező és a következő megoldások nem célszerűek.

Vannak, akik azt javasolják, hogy a „fontosabb" egyed azonosítóját kapcsolóként tegyük a másik egyedbe. Például a KOCSI tulajdonságsorát bővítsük a Tulajkód tétellel. Ez biztos átmenetet jelent a két egyed között. Nincs gond a többszörös értékekkel, mint az M:N fokú viszonyoknál, tehát a kezelési „előítélet" - a kocsi tulajdonosát könnyebb kikeresni, mint a tulajdonos kocsiját - nem jár komoly hátránnyal. A dolog veleje az, hogy a két egyed között olyan ***mesterséges fölé-alárendelt***, 1:N fokú kapcsolatot létesítünk, amelyben az alárendeltek számát (N) eleve egyre (1) korlátozzuk. Ez a változat roppant gyakorlatias. Viszont elméletileg hibás, mert ebben a tekintetben az adatmodellezés nem ismeri a „fontosabb" minőséget.

A harmadik irányzat szerint a KOCSI és a TULAJDONOS éppen úgy ***mellérendelt*** viszonyban áll, mintha összefüggésük hálós természetű lenne. Az M:N-es kapcsolódások esetében külön egyedet szoktunk alkalmazni (vö. HASZNÁLJA), amelynek még az az előnye is megvan, hogy külön leíró tulajdonságok köthetők hozzá. A Vétel-dátuma tulajdonság azt fejezné ki, hogy milyen keltezéssel került a kocsi a tulajdonos birtokába. Ez az ismeret nem a tulajdonost és nem a kocsit, hanem a kettőjük viszonyát jellemzi. Ezért elvileg az a helyes, ha a TULAJDONOS és a KOCSI egyedet a TULAJDONA egyeddel kapcsoljuk össze, ahhoz kötve a Vétel-dátuma tulajdonságot.

Sajnos ez a megközelítés sem tökéletes. Gyakorlatilag végképpen nem az, mert megnöveli az elérést a tulajdonostól a kocsiig és megfordítva, hiszen egy külön egyedet kell kezelni, amely kulcsának a részei, mint ismeretek, redundánsak és fogyasztják a tárat. Elméletileg pedig órákat lehetne azon vitatkozni, hogy a Vétel-dátuma tényleg a tulajdonos és a kocsi viszonyát jellemzi-e, vagy pedig valójában magát a kocsit. Hiszen a kocsi korábbi életére nem vagyunk kíváncsiak. Ha azok lennénk és a kocsihoz több Vétel-dátum érték is kapcsolódhatna, akkor a tulajdonos-kocsi viszony eleve nem lehetne 1:1 fokú.

Be kell vallanunk, hogy az inhomogén 1:1 fokú viszonyok megfelelő tükrözése az adatmodellezési elmélet egyik gyenge pontja. Egyetlen fogódzóként csakis a viszony fokának az időbeli változására tudunk támaszkodni. Ha és amennyiben egy tulajdonosnak a későbbiekben több kocsija is lehet, akkor a második változat a helyes megoldás. Ha pedig egy kocsinak több tulajdonosa is lehet majd, akkor a harmadik utat célszerű követni. Mivel meglehetősen ritka, hogy az 1:1 fokú viszony M:N fokúra változik, viszont az 1:N fok igen gyakori és sokak számára természetes, elméletileg nem tökéletesen megalapozva, de a gyakorlatnak megfelelően ebben a könyvben általában a második változatot tudjuk ajánlani.

A kölcsönös viszonyok az adatbázis előzetes tervezési fázisában ugyanúgy tükrözhetők az adatmodellben, mint a mellérendelt M:N fokú viszonyok (ld. 6.5 ábra). A különbség csak annyi - ezért nem is mutatunk itt külön ábrát -, hogy a két egyed dobozát összekötő vonal mindkét végén sima lesz. Tehát nem alkalmazunk egyik irányban sem többszörösséget mutató „varjúlábat".

6.10 Gyorsmérleg az adatszerkezetről

Jól működő (ismeretigényeinket kiszolgáló), hatékony (tárban és kezelési időben takarékos), rugalmas (a szerkezeti módosításokat könnyen elviselő), robusztus (a kezelt tételek számának a drasztikus változásaitól nem befolyásolt) adatbázist kell építenünk. Erre nem lehetünk képesek akkor, ha nem határozzuk meg igen nagy tudatossággal az adatbázis általános elvi szerkezetét, az adatmodellt.

Az adatbázis nem az ösztönösen megfogalmazott rekordképek szerint állományokba zúdított ismeretek kazla. Az adatbázis szervezettsége azt jelenti, hogy roppant figyelemmel, fáradságos mérlegeléssel meghatározzuk az egyedtípusokat. Kijelöljük azok belső szerkezetét (a tulajdonságtípusok sorait) és külső szerkezetét (a kapcsolattípusok együttesét). A két szerkezet összefügg. Ha két egyedtípus kapcsolódna egymáshoz (külső szerkezet), de ez nem fejeződik ki a modellünkben, akkor új egyed- vagy tulajdonságtípust kell bevezetnünk (belső szerkezet). Ha egy egyedtípusnak van egy kapcsoló tulajdonsága (belső szerkezet), de nem határoztuk meg az általa hordozott kapcsolatot (külső szerkezet), akkor azt pótolnunk kell. Mindig abban a tudatban kell eljárnunk, hogy az egyed-, a tulajdonság- és a kapcsolattípusok egymástól elválaszthatatlanok. Csakis ezen tényezőknek a fegyelmezett elrendezése eredményez jó adatmodellt és annak megfelelő adatbázist.

Ellenőrző kérdések - 6

6/01 Ön szerint az alábbi állítások közül melyik igaz (I) és melyik hamis (H):

- Van egy MAGNÓSZALAG és egy HANGLEMEZ állományom. Mivel mind a kettőben szerepelnek ugyanazok a zeneszámok, van egy jó adatbázisom.
- A fenti két állományomat bármikor összeválogathatom pl. a „Richard” név szerint és ezzel megfelelő ismeretekhez jutok.
- A két egyedtípus belső és külső szerkezete összefügg. Ha van egy külön SZEMÉLY állományom, akkor abból indulva lekérdezhetem azokat a szalagokat és lemezeket, amelyeken Cliff Richard énekel.

6/02 Ön szerint milyen viszonyban állnak egymással a következő jelenségek? Hierarchikus (H), hálós (S) vagy lineáris (L) megjelölést alkalmazzon a viszony 1:N, M:N illetve 1:1 foka szerint.

- Kocsi és CASCO-biztosítása.
- Személyek és nyelvek.
- Számla és az azon lévő tételsorok.

6/03 Adott a RENDELÉS egyedtípus a Rendelésszám, -dátum és Számlaszám tulajdonságtípusokkal, valamint a SZÁMLA egyedtípus a Számlaszám és Számlaegyenleg tulajdonságtípusokkal. Adja meg a tulajdonságok relatív és abszolút szerepeinek a párosait. Az azonosítót A, a leírót L, a kapcsolót K jelölje. Például a SZEMÉLY egyedben a Személynév szereppárosa LL lenne.

- RENDELÉS/Rendelésszám
- RENDELÉS/Rendelésdátum
- RENDELÉS/Számlaszám
- SZÁMLA/Számlaszám
- SZÁMLA/Számlaegyenleg

6/04 Ön szerint milyen a számla és az azon lévő tételsorok közötti viszony? Az alábbi számok egyikét adja meg:

- A számla felől kötelező, a sor felől nem az (1).
- A sor felől kötelező, a számla felől nem az (2).
- Mindkét oldalról kötelező (3).
- Mindkét oldalról opcionális (4).

6/05 Ön szerepeltetné-e a SZEMÉLY egyedtípusban a Képzettség tulajdonságot az alábbiak szerint? A helyes válasz sorszámát adja meg.

- Nem, mert egy személynek több képzettsége is lehet.
- Ismétlődő csoporttal megoldható a dolog.
- Mindez nem probléma, ha csak a legmagasabb képzettséget veszem fel.

6/06 Adja meg a válasz sorszámával, hogy milyen következtetést vonna le Ön abból, ha a SZEMÉLY egyedet annak Dátum (értsd: születési dátum) tulajdonságán összekapcsolva a RENDELÉS egyedet (Rendelésdátum) kiderülne, hogy mindig Kovács születésnapján rendelnek eperfagyit:

- Kovács szereti a fagylaltot.
- Kovács csakis az eperfagylaltot szereti.
- Lehet, hogy Kovács epres, de két egyedet leírón nem szabad kapcsolni.

6/07 Adott a SZÁMLA egyedtípus Számlaszám azonosítója, amely voltaképpen a Vevőkód értékéből és egy Sorszám-ból áll. Legyen kedves és adja meg a három tétel abszolút és relatív szerepjelét az azonosító (A), kulcsrész (R) és kapcsoló (K) megjelölésekkel. Például a RENDELÉSTÉTEL Rendelésszám és Cikkszám összetett azonosítójában az utóbbi jele AK lenne, mivel azonosító a CIKK egyedben, de ugyanakkor kapcsol is felé a RENDELÉSTÉTEL-ben.

6/08 Mit tenne Ön akkor, ha olyan számla érkezne be, amelyen a Számlaszám értéke részben elmosódott? A helyes válasz sorszámát adja meg.

- Kidobnám a számlát.
- A felismerhető karakterekkel vinném be a számlát.
- Mesterséges azonosítót- például napi dátumot és sorszámot - alkalmaznék.
- Külön állományba tenném a kétséges tételt annak tisztázásáig.

7. SZERKEZETI FINOMSÁGOK

7.1 Az adatmodell sokszínűsége

Az előző fejezet során megismertük az adatmodellek általános felépítését. Az adatmodellt egyedtípusok alkotják úgy, hogy meghatározott tulajdonságtípus-sorokkal rendelkeznek. A kapcsoló szerepű tulajdonságokon keresztül az egyedtípusok között kapcsolattípusokat jelölünk ki. Az adatbázisok ennek a szerkezetnek megfelelően épülnek fel. Az egyedelőfordulásokat tulajdonság-értéksorok írják le. A kapcsolók értékein át az egyedelőfordulások egymással kapcsolatelőfordulásokat létesítenek.

Az adatmodell tényezői közül a tulajdonságokat a szerepeikkel, az egyedviszonyokat az összefüggés fokával és opcionalitásával ***minősítettük***. Amint láthattuk, az osztályozás mindig valamilyen modellszerkezeti korláto(ka)t eredményezett. Általánosabban fogalmazva úgy is mondhatjuk, hogy a tényező minősége mindig valamilyen mennyiségi vonzattal is jár. A besorolás miatt az adatmodell némiképpen összetettebbé válik, viszont ennek árán sokkal de sokkal pontosabb képet tudunk rögzíteni a valóságról. Ez pedig a korrekt adatkezelés feltétele, amint azt például a kapcsolatok opcionalitásával összefüggésben tapasztaltuk (ld. a 6.5 pontot).

Az adatmodell tényezőit érdemes még erősebb nagyító alá venni a valóság még pontosabb tükrözése érdekében. Erre ösztönöz bennünket az a tény is, hogy az előző fejezetben nyitva hagytunk néhány kérdést. Az ismétlődés kapcsán néhányan esetleg úgy gondolják, hogy ők ismernek általunk eddig be nem mutatott megoldást is az adatértékek többszörözésének a problémájára. Tehát ezt a kérdéskört újra elő kell vennünk. A 6.5 pontban „inhomogén" kapcsolatról beszéltünk. Az olvasó joggal várja, hogy indokoljuk a megkülönböztető jelző használatát, azaz kitérjünk a „homogén" kapcsolatokra is. Végül az előbbieknél rejtettebben és csak általánosságban vetődött fel az „üres vagy ismeretlen" érték problémája. Nem ugyanazt jelentené a két dolog? Ezt a dilemmát is fel kell oldanunk.

Ennek a fejezetnek az a célja, hogy az adatmodellel kapcsolatos alapvető ismereteinket tovább finomítsuk és újabb szerkezeti tényezők feltárásával gazdagítsuk.

Az előző fejezetben bemutatott strukturális összefüggésekre alapozva gyakorlatilag bármilyen alkalmazási környezet adatmodelljét össze tudjuk állítani. Ezért egyesek megelégednek az egyed-tulajdonság-kapcsolat hármasra vonatkozó alapszintű tudással. Ez nem is csoda, mivel egyes kezelőrendszerek még az alapvető struktúra lehetőségeit sem aknázzák ki. Mi viszont nem állhatunk meg az adatmodellezés alfájánál. Bár az omegáig ebben a könyvben egyáltalán nem fogunk eljutni, a fejezet végére már igen gazdag adatmodell-kép fog összeállni bennünk. Először azt vizsgáljuk meg, hogy miért szükséges többet tudnunk az adatmodell szerkezetéről.

7.2 A „száz-százalékos elv"

Eddig nem beszéltünk egy nagyon mindennapinak (triviálisnak) látszó korlátról, amelynek számos vonzatát jelenleg nem óhajtjuk taglalni és amelyet éppen ezért szigorú kitételként most nem is fogalmazunk meg. Arról van szó, hogy az adatmodellben minden tényezőnek a saját

fajtáján belül egyértelmű nevet kell adni. Magyarul: nem lehet két egyed-, tulajdonság- vagy kapcsolattípusnak azonos a neve. Viszont az nem kizárt - legfeljebb zavaró -, hogy például egy egyedtípusnak és egy tulajdonságtípusnak ugyanaz legyen a megjelölése.

A jelzett megkötés szigorát a mindennapi ismeretkezelés során nem igazán érezzük. Az alábbi példa igazolni látszik gondtalanságunkat:

7.1 példa

„Az N. számú megrendelés X keltezéssel érkezett. Y dátumra kell teljesíteni."
„Q napon fogadtuk az M. rendelést. A szállítást Z időpontra kérik."

A ténylegesen használt szavak és azok sorrendjei nem tévesztik meg az adatbázistervezőt. Felfedezi a két kijelentésben a közöset és előáll a következő modellrészlettel:

RENDELÉS

Rendelésazonosító	Rendelésdátum	Szállításdátum
M	X	Y
N	Q	Z

7.1 ábra: Egy megfelelőnek látszó modellrészlet

Első ránézésre a példa megoldása tökéletes. A tervező tudta, hogy az egyedhez nem köthet két azonos (Dátum) nevű tulajdonságot. Ezért - helyesen - két eltérő megjelölést alkalmazott a keltezésekre. Az átlagos szemlélő semmi kivetnivalót sem talál a 7.1 ábra szerkezetében. Nem is azzal van a baj, ami az ábrán látható, hanem azzal, ami azon nem szerepel. Próbáljuk csak meg mélyebben elemezni ezt a minimodellt!

A RENDELÉS egyed két leíró tulajdonságának a minősítő jelzői (Rendelés és Szállítás) jól mutatják, hogy nem azonos lényegekről van szó. Ugyanakkor a közös szórész (dátum) azt sejteti velünk, hogy mégiscsak valamilyen módon rokon jelenségeket jelölnek e tulajdonságok nevei. A két tétel nem pénz, nem név, nem azonosító, nem kód, hanem keltezés jellegű értéket tartalmaz. Csakúgy, mint például az itt nem mutatott SZEMÉLY egyed esetleges Születés- vagy Házassági-dátum nevű tulajdonsága. Az azonos jelleg miatt a Rendelésdátum értékét össze lehet vetni a Szállításdátum-éval, sőt - bár ennek nincs sok értelme - a Születésdátum-éval is. A Rendelés-azonosító értékével viszont soha.

Nagy ügy, mondják erre a tapasztalatlanok. A tulajdonságok adattípusát (vö. D 4/4) „dátum"-nak deklaráljuk a vonatkozó tételek esetében, és ezzel már le is zártuk a kérdést. Az adatkezelő az adattípus alapján el tudja végezni a szükséges műveleteket (validálás, hasonlítás stb.). Csakhogy a megoldás nem ilyen egyszerű. Tegyük fel, hogy a Rendelés- és a Szállításdátum nem eshet vasár- és ünnepnapra. Viszont a születés és a házasodás ilyen napokon sem tiltott... Amint látjuk, az adattípus (dátum) ugyan jó gyakorlati fogódzó, de nem határolja be kellő pontossággal a tulajdonságok megengedett értékeit.

Miért gond ez? A választ akkor kapjuk meg, ha ismerjük az adatkezelés kétféle elvi módját. A fejletlen adatkezelésre az volt a jellemző, hogy a programozó által írt programsorok szolgáltak az adatbázis-korlátok (vö. 5.4 pont) érvényesítésére. A kezdetek kezdetén a programnak kellett ellenőriznie az azonosítók egyediségét; azt, hogy az adat tartalma üres-e; mivel pedig még nem volt dátum-adattípus, azt is, hogy a keltezésnek szánt adatban nincs-e 13. hónap. Mivel a felhasználói programba illesztett eljárások végezték ezeket a műveleteket, az ilyen adatkezelést ***procedurális*** jellegűnek mondjuk.

Az ilyen kezelés nem kívánatos, mert adat-program függőséget okoz (vö. D 5/3). Ha húsz programban kezeljük a Rendelésdátum adatot, akkor ennyiben kell leírni azt a procedúrát, amely

kizárja a vasár- és ünnepnap dátum-értékeket. Változás esetén - a vasárnap megengedett érték lesz - ennyi programot kell átírni. A közösen használt - egyszer megírt, de több programhoz kötött - eljárás némileg javít a helyzeten. Csakhogy a procedúrát akkor is a programozónak kell megírnia; nem szabad elfelejtenie a megfelelő programokhoz kötnie; változás esetén pedig továbbra is elkerülhetetlen a programmódosítás. A közös eljárásnak az is alapvető feltétele, hogy az adat - esetünkben a Rendelésdátum - formáját mindig azonos módon határozzák meg. Ehhez pedig valahol - mégpedig csakis egy helyen - le kellene rögzíteni ezt a formát.

Ma már a legtöbb kezelő támogatja a dátum adattípust. A 13. hónap kizárására nem kell programsort írni: az ellenőrzési eljárást maga az adatkezelő tartalmazza. Azt akkor hajtja végre, ha az adatról (Rendelésdátum) tudja, hogy az dátum típusú. Ezt a tényt előre ki kell jelenteni, meg kell határozni az adatkezelő részére. Ezért az ilyen jellegű adatkezelést deklaratív vagy ***definitív*** jellegűnek nevezzük. Deklarálhatjuk, hogy egy adat értéke nem lehet üres; az értéknek egyedinek kell lennie stb. Az adatkezelőbe beépített eljárás automatikusan elindul, ha az adatkezelő a definiált korláttal találkozik.

A korlátokat csak egyszer, a sémában (vö. 5.6 pont) kell megadni. Változás esetén nem kell a felhasználói programokhoz nyúlni; csak a sémát kell módosítani, egyetlen egy helyen. Ezért a definitív adatkezelés a procedurálisnál sokkal nagyobb adat-program-függetlenséget biztosít. Tegyük mindehhez, hogy a korlát eleve generikus, tehát egyetlen rutint feltételez minden dátum típusú adat ellenőrzésekor, függetlenül az adatnevektől. (Ezzel szemben procedurális kezeléskor adatnevenként külön-külön kell végrehajtatni a kapcsolódó ellenőrzési eljárást.)

A definitív kezelés előnyei miatt fogalmazták meg az ún. ***száz-százalékos elvet*** [15]. Ennek értelmében - az adatfüggetlenség érdekében - minden az adatokra vonatkozó ismeretet az adatbázis fogalmi sémájában - tehát a felhasználói programon kívül - kell megadni. Az elv célját jól mutatja példánk folytatása.

A Rendelés- és Szállításdátum adattípusa dátum. A kezelő ellenőrizni tudja, hogy az érték valós keltezésnek felel-e meg. Arra azonban nem képes, hogy kizárja a vasár- és ünnepnap értékeket. Tehát erre a célra felhasználói procedúrát kellene készíteni, megsértve a száz-százalékos elvet (angolul: 100% principle). Nincs valamilyen célszerűbb megoldás? A következő pont erre a kérdésre ad választ.

7.3 Az értéktartomány

A korszerű adatbázisok alapvető strukturálási tényezője az ***értéktartomány*** (angolul: domain, ejtsd: doméjn).

D 7/1 **Az értéktartomány a tulajdonságtípus általánosan felvehető értékeinek a halmaza.**

A meghatározásban az „általános” szó mutatja a lényeget. A tulajdonságtípusok nem pusztán az egyedtípusok tartozékai, hanem önálló fontosságú tényezők. Ez abból a tényből is következik, hogy ugyanaz a tulajdonságtípus több egyedtípushoz kapcsolódhat. A doméjn a tulajdonságtípus egyedtípustól független - általános - értékeinek a halmazát jelenti. Példa: A Dátum általános értékhalmaz minden egyedtől függetlenül az ismert keltezéseket tartalmazza, miközben számos egyedben szerepelhet Dátum nevű tulajdonságunk.

Csak röviden térünk ki arra, hogy az értéktartomány mint halmaz tételeit eltérő módokon lehet meghatározni. Vannak ***viszonyító*** jellegű tartományok. Például: A Rendelésazonosító értéke nagyobb mint 1, de kisebb mint 999999. ***Minta*** fajtájúak. Például: A Telefonszám általános

felépítése feleljen meg az (X99)-9-999-999 etalonnak. Gyakori a ***lista*** típusú doméjn. Például a Személy-nem csak „Nő” vagy „Férfi” értéket vehet fel. Végül a tartomány ***algoritmus*** alapú is lehet. Fenti példánk esetében a doméjnhez köthetjük azt az eljárást, amely kizárja a vasár- és ünnepnap értékeket a Rendelésszám tartalmának a kezelésekor.

A jó adatkezelők megengedik, a még jobbak megkövetelik azt, hogy az egyedtípusok tulajdonságtípusait értéktartományoknak feleltessük meg. Így az értékkorlátokat egy helyen kell csak definitív módon megadnunk, és nincs szükség változásnak kitett programsorok írására. Csak a tulajdonság értéktartományba való besorolásának a problémájával kell szembenéznünk.

A Rendelésdátum és a Születésdátum egyaránt keltezés. Ha mindkét tulajdonságot az általános Dátum doméjnhez rendeljük, akkor nem zárjuk ki a vasár- és ünnepnapi rendeléseket. Ezért a Születésdátum-ot a Dátum tartományhoz kötjük, viszont a Rendelésdátum esetében meghatározunk egy - algoritmus jellegű - szigorúbb Munkanap értéktartományt. Ebbe a doméjnbe fog tartozni a Szállításdátum is. Ha a rendelések/szállítások korlátait feloldjuk, akkor nincs más dolgunk, mint a tulajdonságok áttétele a Munkanap doméjnből az általánosabb Dátum-ba.

Az elméletileg igazán fejlett adatkezelés esetében még azt is ki kellene tudnunk jelenteni, hogy a Munkanap a Dátum alhalmaza. Mivel a gyakorlati rendszerek többsége még az előző igényeknek sem tesz eleget, a doméjn további elvi lehetőségeinek a boncolgatását itt lezárjuk.

7.4 A szerepnév

A tiszta adatmodellezési elmélet szerint vannak egyedek (RENDELÉS), tartományok - azaz egyedtől független általános tulajdonságok - (Munkanap) és a kétféle tényező közötti viszony a tulajdonképpeni attribútum (Rendelésdátum). Egyes gyakorlati adatkezelőkben nem tesznek különbséget az általános (doméjn) és az egyedspecifikus (attribútum) tulajdonság között. Mások a két tényezőt elismerik ugyan, de nem követelik meg azok megfeleltetését. Vagyis például a Rendelésazonosító attribútumhoz nem kötelező kijelölni a megfelelő tartományt.

Most ne firtassuk tovább a gyakorlati kezelők korlátait! Inkább csak azt tekintsük át, hogy milyen körülmények között van mindenképpen szükség az általános és a specifikus tulajdonság meghatározására és egymáshoz rendelésére. A magyarázat kedvéért feledkezzünk el példánk korábbi speciális korlátozásáról és tételezzük fel azt, hogy a Rendelés- (és a Szállítás-) dátum bármelyik napot jelölheti. Gondolatmenetünk a következő:

Szükségünk van a RENDELÉS egyedtípusra. Kijelöljük annak elsődleges kulcsát, a Rendelésazonosító-t. (Ennek értéktartománya mondanivalónk szempontjából lényegtelen, ezért arra nem térünk ki.) Felfedezzük, hogy a rendeléseket adott napon adják fel. Ezért felvennénk a Dátum tulajdonságot, amely értékei a Dátum doméjnből származnak. Majd szükségünk van a szállítás idejének a megjelölésére is. Ezért ismételten a Dátum tulajdonságot alkalmaznánk, hiszen a szállítási keltezések is a Dátum doméjnhez kötődnek.

Csakhogy egy egyed tulajdonságsorában nem szerepelhet két azonos nevű tényező. Miként oldhatjuk meg úgy ezt a problémát, hogy a két egyedi, specifikus tartalmat elkülönítjük, de mégis utalunk a közös, generikus dátum lényegre? Ennek a kettősségnek a feloldására szolgál a ***szerepnév*** fogalma.

D 7/2 **Az értéktartomány szerepneveinek hívjuk azokat a tulajdonságokat, amelyek specifikus, egyeden belüli tartalma a tartomány általános korlátainak felel meg.**

Már az adatmodellezési elmélet gyermekkorában felfedezték ezt a tényezőt. Azért hívták szerep***név***nek, mert kezdetben valóban a megnevezés formájához kapcsolódott. Az egyedhez kapcsolt tulajdonság nevét (Rendelésdátum) úgy alakították ki, hogy az általános tulajdonság, a doméjn megjelölését (Dátum) elölről kiegészítették a specifikus tartalmi értelmezés (Rendelés) minősítő jelzőjével.

Megjegyzés: A tulajdonságnak az egyeden belüli szemantikai szerepét, vagyis mondanivalóját nem szabad összetéveszteni a korábban tárgyalt szerkezeti szereppel (ld. 6.3 pont). A Rendelés- és a Szállításdátum értelme és ezért szerepneve más, míg strukturális szerepe azonos, egyaránt leíró.

Az ember nem a szavak, hanem az azok mögötti fogalmak alapján szerez ismeretet. Felismeri a „dátum" és a „keltezés" azonosságát. Ennyire még a számítógépet is meg lehet tanítani. Ezért már csak az elavult rendszerek ragaszkodnak ahhoz, hogy a tulajdonságok speciális lényegét a generikus tartalomra utaló név jelzős szerkezetével jelöljék (Rendelés - Dátum). A korszerűbb elméletben és gyakorlatban nem a név írásmódja a fontos.

Most ismét tételezzük fel, hogy rendelést nem lehet feladni és szállítást nem lehet teljesíteni vasár- és ünnepnapokon. A modern adatkezelőkben semmi akadálya sincs, hogy kijelentsük: a Rendelés- és Szállításdátum a Munkanap doméjn szerepneve. Ezzel a formális megkötés - a szerepnév a doméjn jelzős neve - megszűnt. A lényeg megmaradt: a szerepnév az általános tulajdonság értékét az egyeden belül specifikusan, tartalmilag ***minősíti***.

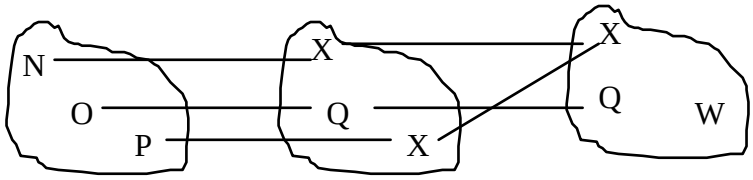
Az adatmodellek változnak. Ma a RENDELÉS egyedben csak egyetlen mennyiségre utalunk, feltételezve, hogy a kért mennyiséget le is szállítjuk. Ezért Mennyiség néven vesszük fel a vonatkozó tulajdonságot. Holnap kiderül, hogy a szállított és az igényelt mennyiség nem mindig azonos. Ezért szükségünk lenne egy újabb mennyiség tulajdonságra, amit viszont már csak más névvel jelölhetünk. Az egyedben lesz egy Mennyiség és egy Szállított-mennyiség nevű adat. Ez nem elegáns, sőt - annak számára, aki csak az első adatot látja, de tud a másodikról is - megtévesztő is lehet. Ezért vannak olyan modellezési irányzatok [16], amelyek minden esetben előírják a szerepnevek használatát.

Utolsó gondolatként felhívjuk a figyelmet a szerepnév-doméjn viszony egyoldalúságára. Az adott értéktartományhoz kapcsolódó szerepnév-tulajdonság csak olyan értéket vehet fel, amely a doméjn értékhalmazának a része. Arra nézve viszont nincs megkötés, hogy az értékhalmaz összes tétele valóban alkalmazásra is kerüljön. Magyarul: nem biztos (sőt!), hogy minden munkanapon fel is adnak rendelést. Ezzel szemben aligha fogjuk alkalmazni a Személy-neme tulajdonságot illetve doméjnt, ha olyan egyedtípust tervezünk, amelyben vagy csak nők, vagy csak férfiak adatait vezetjük. Végeredményben az egyed-specifikus tulajdonság értékkészlete megegyezhet a generikus doméjn értékeinek teljes halmazával, de lehet annak valódi alhalmaza is.

A szerepnév segítségével a valóságot pontosabban reprezentáló adatmodelleket tudunk építeni a fejezet első pontjában felvázolt célnak megfelelően. Ezért kell visszatérnünk a 7.1 ábra példájára. Az ott ismertetett adatmodell jó, de nem tökéletes. Azért nem az, mert nem tükrözi azt a valóságbeli tényt, hogy a Rendelésdátum és a Szállításdátum ugyanabba az értéktartományba tartozik, de mégis eltérő lényeg. Tehát az ábrából hiányzik a Munkanap doméjn és a két dátum annak való megfeleltetése.

Az egyedtípus, a generikus (doméjn) és specifikus (attribútum) tulajdonságtípus összefüggéseit a 7.2 ábra mutatja.

EGYEDTÍPUS	ATTRIBÚTUM	DOMÉJN
RENDELÉS	Rendelésdátum	Munkanap



7.2 ábra: Három ismereti tényező összefüggése

7.5 Ismétlődés és szerepnév

A 6.6 pontban az ismétlődő adatok gondjával kellett szembenéznünk. Ott csak egyetlen megoldást találtunk. Ha egy egyedtípus (TULAJDONOS) adott tulajdonságtípusa (Kocsitípus) az egyedelőfordulásokra több értéket is felvehet, akkor külön egyedtípust kell alkalmaznunk. Viszont a most tárgyalt szerepnév konstrukció lehetősége egyeseket arra ösztönözhet, hogy egy másféle ismétléssel kerüljék meg az értéktöbbszörözés problémáját.

Példaként tegyük fel, hogy a személyek nyelvtudásait kívánjuk rögzíteni! Azt már beláttuk, hogy az egyed (SZEMÉLY) tulajdonságának (Nyelvtudás) az értéke nem lehet ismétlődő. Azonban mi történne, ha a SZEMÉLY egyedben - mondjuk - három tulajdonságot alkalmaznánk a Nyelvtudás közös doméjn szerepneveiként? Az egyednek lenne Nyelvtudás-1, -2 és -3 nevű tulajdonsága úgy, hogy ezek mindegyike csakis a Nyelvtudás tartományból vehetné az értékét.

Az olvasó máris érzékeli a csapdák sorozatát. Nem tudjuk kezelni a személy által ismert esetleges negyedik nyelvet. A háromnál kevesebb nyelvet értőknél fellép az üres érték gondja. Hiányzik az elrendezettség is, mert az egyik személynél az angol a Nyelvtudás-1, a másiknál a Nyelvtudás-2 tartalma lesz. Ha viszont a szerepneveket adott nyelvekhez kötjük, akkor nemcsak személyenként, hanem generikusan is csak három nyelv kimutatására lesz lehetőségünk.

A tanulság egyértelmű. A szerepnév eltérő szemantikai tartalmú (értsd: az egyedtípus szempontjából más-más jelentésű), de ugyanakkor az értékek szerint azonos doméjnbe tartozó tulajdonságok (vö. Rendelés- és Szállításdátum) megkülönböztetésére szolgál. Sohasem alkalmazható az azonos szemantikai lényegű - azonos tartalmi jelentésű - ismeret (Nyelvtudás) többszörösségének a feloldására. Az adatbázis „mélysége” - egy személy több nyelven beszél, tehát a Nyelvtudás adatnak több tartalma van - az ilyen esetekben sohasem váltható ki az adatbázis „szélességének” - a tulajdonságtípusok számának - a növelésével, álszerepnevek alkalmazásával. A személy és a nyelv viszonya - a 6.6 pontban jelzett módon - csakis egy további egyed (SZEMÉLY NYELVTUDÁSA) bevezetésével fejezhető ki.

A szerepnév nem alkalmas az ismétlődés gondjának a megoldására. Viszont ugyanakkor két nagyon fontos adatbáziskonstrukció elvi alapjaként szolgál. A következő pontokban ezeket fogjuk áttekinteni.

7.6 A visszamutató egyedviszony

Az előző fejezetben az egyedek ***inhomogén*** összefüggéseit boncolgattuk. Az egyedek olyan viszonyaival foglalkoztunk, amelyekben két eltérő egyedtípus (pl. KOCSI és TULAJDONOS) előfordulásai kapcsolódtak egymáshoz. Ezzel szemben a gyakorlatban sűrűn megesik, hogy ismeretközléseink alanya és állítmánya ugyanabba a jelenségkörbe tartozik. Lássuk csak a következő három példamondatot:

7.2 példa

„Jóska főnöke Laci."
„Feri főnöke Laci."
„Laci főnöke Peti."

A három kitétel ugyanabba a „mondattípusba" tartozik. Ezért a 7.2 példa megfelelő adatmodelljét látszólag igen könnyű meghatározni. Íme az eredmény:

SZEMÉLY

Személynév	Főnöknév
Jóska	Laci
Feri	Laci
Laci	Peti

7.3 ábra: Beosztottak és főnökök ismeretei

A szerkezet az első pillanatban korrektnek tűnik. Ám az adatkezelési műveletek átgondolása után kezdeti biztonságunk elveszik. Mert mi is történik akkor, ha a „Laci" azonosította sort töröljük azért, mert Laci kilép a cégtől? Nyilvánvaló, hogy távozása után „Laci" nem lehet a másik két munkatárs főnöke. Tehát az első két sorral tennünk kell valamit azzal kapcsolatosan, hogy a harmadikat megszüntetjük.

Azonban honnan tudjuk azt, hogy mi legyen az első két tétel sorsa és miből is sejtjük, hogy azokkal egyáltalán törődnünk kell, amikor a harmadik egyedelőfordulást kezeljük? A gyakorlati adatkezelők többsége csak procedurális megoldást kínál. Ezért a programozónak vagy eszébe jut a harmadik sor törlése kapcsán üríteni a főnök nevét az első két sorban, vagy sem. A műveletet vagy jól programozza, vagy sem. Az adatbázis konzisztenciája teljes mértékben rajta múlik - ami igen veszélyes.

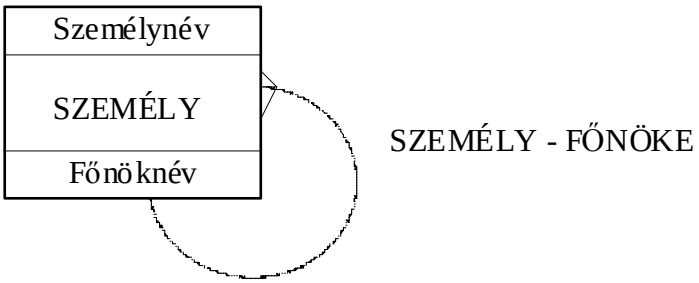
Mi tehát a célszerű megoldás? A definitív adatkezelés. Az adatkezelővel előre tudatnunk kell, hogy a „Laci" adatsor törlése esetén mit tegyen az érintett többiekkel. Ehhez pedig még előbb azt kell leszögeznünk az adatmodellben, hogy a SZEMÉLY egyedtípus előfordulásai egymáshoz kapcsolódnak.

Ezen a ponton zavarba jövünk. Mindeddig úgy sejtettük, hogy a viszony két eltérő egyedtípust (pl. GÉPKOCSI és TULAJDONOS) kapcsol össze. (Annak ellenére, hogy az eddigi fejezetek során a megfogalmazásokban igyekeztünk kerülni ezt a látszatot.) Most rá kell döbbennünk arra, hogy van olyan viszony is, amely egyazon egyedtípus különböző előfordulásait köti egymáshoz. A fentebb jelzett adatkezelési problémát nem tudjuk kikerülni a SZEMÉLY - FŐNÖKE viszony meghatározása nélkül. Hiszen Jóska és Feri Lacihoz, Laci pedig Petihez kapcsolódik.

D 7/3 **Visszamutató viszonyról beszélünk akkor, ha egy egyedtípus előfordulásai ugyanannak az egyedtípusnak más előfordulásaival állnak összefüggésben.**

A visszamutató (angolul: involuted vagy recursive) viszonyokat ***homogén*** összefüggéseknek is nevezzük, mert a kapcsolt egyedelőfordulások egyneműek. Példánkban személyek kötődnek más személyekhez. Azt tehát már elkönyvelhetjük, hogy az inhomogén viszonyokon kívül figyelnünk kell a homogén jellegűekre is. Már csak arról nem szabad elfeledkeznünk, hogy az utóbbiak esetében is tekintetbe kell venni a viszony fokát, opcionalitását és természetét. Erről a következő pontokban lesz szó.

Az adatmodell-diagramokban a visszamutató viszonyokat úgy tükrözzük, hogy az egyedtípust jelölő dobozból kiinduló vonalat oda hajlítjuk vissza. Lásd a 7.4 ábrát.



7.4 ábra: A visszamutató kapcsolat ábrázolása

7.7 A hierarchikus homogén viszonyok

A homogén viszonyok mind birtoklási („van neki” - angolul: „has a”) természetűek. A munkatársnak van főnöke, a vezetőnek van beosztottja. Ezért csak a fokkal és az opcionalitással kell foglalkoznunk.

A 7.4 ábra ***1:N fokú*** viszonyt mutat, hiszen azzal a feltételezéssel élünk, hogy egy vezetőnek lehet több beosztottja is, viszont minden munkatárs csak egy főnökhöz tartozhat. Könnyen belátható, hogy az ilyen viszonyok mindig törvényszerűen alulról és felülről is ***opcionálisak***. Az ugyanis elképzelhetetlen, hogy minden munkatársnak legyen beosztottja és minden vezetőnek legyen főnöke. A SZEMÉLY - FŐNÖKE munkatársi viszonyban a szó szoros értelmében joggal beszélhetünk fölérendeltről és alárendeltről, az 1:N fokú viszonyok két tényezőjéről. Szemben az inhomogén viszonyokkal, a homogénekben ugyanaz az egyedelőfordulás egyidejűleg fölé- és alárendelt is lehet (ld. „Laci”).

Az inhomogén kapcsolat (pl. TULAJDONOS - KOCSI) két eltérő egyedtípus olyan közös tulajdonságtípusán (Tulajkód) alapult, amelyek közös értékekkel („T1”) rendelkeznek. Viszont a jelenlegi példában csak egy egyedtípusunk van és ezért nem beszélhetünk „közös” tulajdonságról. Ezért felmerül a kérdés, hogy milyen tényezőre épül a SZEMÉLY - FŐNÖKE kapcsolat?

A 7.3 ábrában felfedezzük, hogy a Személynév és a Főnöknév ugyanabból a Személynév értéktartományból származik. Hiszen a főnök neve is egy személy neve. Ezért a két tulajdonság értékei közösek, és ez alapján bizonyos személyek adatai más egyénekéhez köthetők. Tudjuk, hogy Jóska főnöke Laci, aki viszont Peti beosztottja.

A specifikus példából általános következtetést is levonhatunk:

D 7/4 **Az egyedtípus kapcsolatban áll saját magával, ha van olyan leíró tulajdonságtípusa, amely az azonosító korlátozó szerepneve.**

A SZEMÉLY egyed azonosítója a Személynév. Ennek szerepneve a Főnöknév. Minden főnök személy, de nem minden személy főnök. Ezért ebben az esetben a szerepnév nemcsak minősítő, hanem egyben ***korlátozó*** jellegű is. A Főnöknév a Személynév valódi alhalmaza. Újabb ismereteinket a 7.5 ábra mutatja:

SZEMÉLY

Személynév	*Főnöknév*
Jóska	*Laci*
Feri	*Laci*
Laci	*Peti*

7.5 ábra: Beosztottak és főnökök kapcsolt ismeretei

Ez az ábra tartalmában nem tér el a 7.3 ábrától. Azonban a dőltbetűs szedet mutatja, hogy a Főnöknév az adott egyedben nem pusztán leíró, hanem kapcsoló relatív szerepű tulajdonság, amelynek alapján a megfelelő sorokat tudatosan kötjük egymáshoz.

Ma még kevés gyakorlati adatkezelő ad módot a visszamutató kapcsolatok automatikus érvényesítésére. Azok a rendszerek, amelyekben ez lehetséges, konzisztensebb adatbázisokat biztosítanak a számunkra. Ugyanis nem a programozón múlik a korlátok érvényesítése. Maga az adatkezelő garantálja a szabályok betartását. Az adatbázis tervezője kijelöli a visszamutató kapcsolatot és megadja annak opcionalitását. Ha a SZEMÉLY - FŐNÖKE kapcsolat opcionális, akkor „Laci” törlése esetén a másik két sorban üríti a Főnöknév tartalmát. Ha kötelező lenne - mint tudjuk, nem lehet az - akkor azokat is törölné. Sőt, ha Feri valakinek a főnöke, akkor az ő tételét is megszüntetné. (A változások összetett hatásainak az automatikus érvényesítését ***továbbvezetésnek*** - angolul: propagation - nevezzük.)

7.8 Családfa és házastárs viszonyok

Két egyedtípus előfordulásai egymással nemcsak 1:N fokú alá-fölérendeltségi, hanem M:N vagy 1:1 fokú mellérendeltségi viszonyban is lehetnek. Az inhomogén M:N-es egyedviszonyok áttekintése során (ld. 6.7 pont) egyértelműen a további egyedtípus bevezetése mellett tettük le a voksot. Az 1:1 fokú kölcsönös viszonyokra (ld. 6.9 pont) többféle megoldást kínáltunk, de végül is a kvázi-hierarchikus megoldást tartottuk célravezetőnek.

Most tehát az a feladat vár ránk, hogy áttekintsük a mellérendelt homogén viszonyokat és megnézzük, hogy milyen módon tudjuk azokat célszerűen tükrözni az adatmodellben. Először a ***homogén hálók*** esetét fogjuk vizsgálni.

Szemléltetésre a termelésirányítás egyik ismert adatszerkezetét fogjuk alkalmazni. Egy-egy termék - például a kocsi - több másikból épül fel. A kocsinak vannak főszerelvényei, azoknak szerelvényei, amelyek viszont alkatrészekből állnak. A termékek ilyen felépítését ***családfának*** (angolul: family-tree) vagy darabjegyzéknek (parts-list) hívjuk. A moduláris termékfelépítés érdekében ugyanazt a mélyebb szintű „darabot” - amit (termelési) tételnek nevezünk - többféle magasabb szintűben használják fel. Például több karosszériában is azonos ajtót alkalmaznak; az ajtócsavarok szabványosítottak stb. A tételek felhasználásának a listáját ***beépülési jegyzéknek*** (angolul: goes-into list) nevezzük. A 7.6 ábra mutatja példa-adatbázisunkat.

TÉTEL CSALÁDFA

Tételkód	Tételnév	.	***Tételkód-1***	***Tételkód-2***	Darab
T1	A		***T1***	***T2***	2
T2	B		***T1***	***T3***	2
T3	C		***T2***	***T4***	12
T4	D		***T3***	***T4***	8

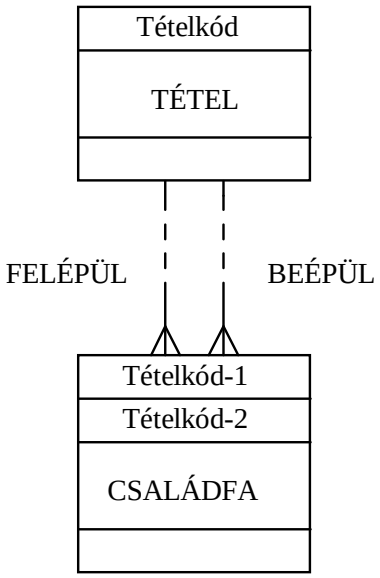
7.6 ábra: Tipikus családfa modellrészlet

A TÉTEL egyed írja le önmagukban a kocsi alkotórészeit, ideértve magát a kocsit is. Mivel egy tételbe több másik épülhet be és egy tétel több másiknak lehet a része, a viszony M:N fokú. Ezért ennek reprezentálására külön egyedet kell alkalmaznunk. A homogén viszonyokat tükröző egyedeket általában családfa jellegűeknek nevezzük. Nem véletlenül. Ha a TÉTEL helyett SZEMÉLY lenne az első egyedünk, akkor a szülők és gyerekek kapcsolatait (akár több generáción keresztül is) a szó szoros értelmében a személyek családfájával tudnánk kifejezni.

A családfa jellegű egyedek igen gyakoriak. Ilyenekkel tükrözzük a nem-hierarchikus szervezetek felépítését, az egymáshoz hálózatosan kapcsolódó homogén objektumok viszonyait (út-, csatorna-, villany- stb. hálózatokat) és így tovább. A családfa egyedek két 1:N fokú, alulról-felülről egyaránt opcionális kapcsolattal kötődnek az alapegyedhez. A kapcsolatok két oldalról opcionálisak, mert maga a végtermék már semmibe sem épül bele (nincs fölérendeltje), alul pedig az alapanyagoknak már nincs további alkotóelemük (alárendeltjük).

A családfa egyed azonosítója mindig összetett. Értékei az alapegyed (TÉTEL) elsődleges kulcsa (Tételkód) értékeinek a párosai. Tehát ezeknek az adott egyeden (CSALÁDFA) belüli relatív szerepe (kulcsrészként) kapcsoló. Mivel egy egyedhez két azonos nevű tulajdonságot nem köthetünk, a családfa egyed azonosítóját az alapegyed azonosítójának a szerepnevei (Tételkód-1 és Tételkód-2) alkotják. Itt ismét találkozunk azzal a problémával, hogy ezek a szerepnevek nemcsak minősítő, hanem egyben korlátozó jellegűek is. A Tételkód-1 nem vehet fel minden Tételkód értéket a viszony opcionalitása miatt.

A 7.7 ábra mutatja a családfa-összefüggések tipikus adatmodellezési részletét:



7.7 ábra: A családfa-viszony modelldiagramja

A homogén hálós viszonyoknál több gondot okoznak az 1:1 fokú összefüggések. Tipikus ilyen viszony a személyek közötti házasság, és ezért az ilyen kapcsolódásokat ***házastársi*** (angolul: marriage) viszonyoknak is hívjuk. Hiszen - európai környezetet feltételezve - egy férjnek csak egy felesége lehet egyidőben és megfordítva. Szemben az 1:N és az M:N fokú homogén viszonyokkal, az 1:1 fokú lehet kötelező is. Kijelenthetjük, hogy csak házas személyekről vezetünk ismereteket. Az összefüggés elvileg egyszerű, de gyakorlati megvalósítása problémákat okoz.

A legrosszabb megoldás, ha létrehozunk egy-egy külön FÉRJ és FELESÉG egyedet. Ebben az esetben csak nehézkesen tudjuk kikeresni a személyekre az azok nemétől független közös ismereteket. Például azt, hogy kik - férj és feleség egyaránt - születtek adott napon. Ráadásul a homogén viszony inhomogénné válna, mert hiszen már két egyedtípus kapcsolódna egymáshoz. Ezért válogatnunk kellene az inhomogén viszonyok megoldásai (ld. 6.9 pont) közül, amelyek egyike sem felelne meg igazán céljainknak.

SZEMÉLY

Személynév	*Házastársnév*
Jóska	*Panni*
Feri	*Kati*
Panni	*Jóska*

7.8 ábra: A házastárs-viszony rossz megoldása

A második út az, hogy egyetlen SZEMÉLY egyedet képzelünk el és a személyek közötti házastársi kapcsolatokat az 1:N fokú homogén viszonyok módjára fejezzük ki. Ez azt jelenti, hogy a SZEMÉLY egyedben felvesszük az azonosító (Személynév) szerepnevét (Házastársnév), amely kapcsoló tulajdonságként szolgál, egymáshoz kötve a házastársi viszonyban álló személyeket. Ezt a megoldást szemlélteti a 7.8 ábra.

A megoldás tökéletlenségét két tényező is mutatja. Egyrészt azonos ismeretsor-részeket kell megismételnünk. (A sorrend a fogalmi szinten közömbös. Ezért a Jóska-Panni ill. a Panni-Jóska részleteket azonosnak kell tekintenünk.) Másrészt éppen ez a kettősség adatkezelési gondokhoz és inkonzisztenciához vezet. A 7.8 ábra hiányos, mert nincs benne a Kati-Feri páros. Mi több, a házasság megszűnése esetén két tételt kell karbantartani.

A fentiek miatt az 1:1 fokú homogén kapcsolatoknál leginkább a harmadik változat a célszerű. A családfa egyed mintájára létrehozunk egy házastárs egyedet a 7.9 ábra szerint.

SZEMÉLY			HÁZASTÁRS	
Személykód	Személynév	...	***Férjkód***	***Feleségkód***
S1	Jóska		***S1***	***S2***
S2	Panni		***S3***	***S4***
S3	Feri			
S4	Kati			

7.9 ábra: A házastársi viszony célszerűbb megoldása

Mit vesztünk és mit nyerünk a legutóbbi megoldással? A két egyed kezelése miatt némi kezelési időtöbblettel kell majd számolnunk. (Nem sokkal, mivel az egyszeres tárolás miatt a kezelési idő csökken.) Ugyanakkor adatstruktúránk biztosítja a konzisztens összefüggéseket; nem

követhetünk el beviteli és egyéb kezelési hibákat. Például nem felejthetjük el, hogy amennyiben Feri felesége Kati, úgy Kati férje Feri.

A 7.9 ábra megoldásának legnagyobb előnye a rugalmasság. Szemben a másik két változattal ez a szerkezet kétféle bővítést is elvisel. Ha nemcsak a jelenleg érvényes, hanem a mindenkori házasságokat is ki akarjuk mutatni, a kapcsolat M:N fokúvá válik. Ez a 7.9 megoldás esetében nem jelent gondot. Ha új tulajdonságot akarunk bevezetni, az sem lehet akadály. Vegyük például a Házassági-dátum adatot. Ez a keltezés nem az egyik vagy másik félre, hanem magára a viszonyra jellemző. A 7.9 ábra HÁZASTÁRS egyede ezzel a tulajdonsággal minden gond nélkül kibővíthető. Viszont a 7.8 ábra esetében a SZEMÉLY egyed kiegészítése az adattétellel redundanciát és adatkezelési problémákat okozna. A fentiek következtében az 1:1 fokú homogén viszonyoknál a házastárs jellegű egyed tűnik a legcélszerűbb megoldásnak.

Ezzel a homogén viszonyok tárgyalását majdnem befejeztük. Már csak két momentumra kell felhívnunk a figyelmet. Az egyik az, hogy a fogalmi modell szintjén kell meghatározni a viszony inhomogén-homogén típusát illetve az egyed sajátos (családfa, házastársi) jellegét. Ha az adatkezelő képtelen a típus és a jelleg érvényesítésére, akkor magunknak kell procedurális programsorokat írnunk az inkonzisztens adattartalom feltárására. Például arra, hogy kizárjuk a következő eseteket: Feri Ferinek a főnöke; a T1 tétel a T1 tételbe épül be; Feri Ferinek a házastársa. (Ilyen tartalmi ciklusok az inhomogén viszonyoknál nem léphetnek fel.)

A másik tényező a szerepnevekkel kapcsolatos. A homogén viszonyokban az összefüggéseket mindig szerepnév kapcsoló tulajdonságok tükrözik (Főnöknév, Tételkód-1, Férjkód). Ezek a szerepnevek mindig korlátozó jellegűek. Ezért magában az alapegyedben (pl. SZEMÉLY) fel kell venni azt a tulajdonságot (pl. Nem), amely a kapcsolódást behatárolja. Csak így kerülhetjük el a téves adatviszonyok kialakulását. Ha nem volna a SZEMÉLY egyedben a Nem tulajdonság és nem kötnénk ahhoz a Férjkód és Feleségkód szerepnév korlátozását, akkor a HÁZASTÁRS egyedben feltűnhetne a Feri - Jóska páros.

7.9 Háromféle „üres” érték

Az adatbáziskezelés egyik leginkább elhanyagolt és legjobban félreértett tényezője az üres (numerikus adatoknál: nulla) tulajdonságérték. A kapcsolódó problémákat a következő négy kérdés-válasz párossal világítjuk meg:

7.3 példa

„Hányszor szült Panni? - Kétszer.”
„Hányszor szült Zsóka? - Egyszer sem.”
„Hányszor szült Kati? - Nem tudom.”
„Hányszor szült Feri? - Hahaha...”

Az első párbeszéd nem szorul magyarázatra. Önmagában véve a második sem. Azonban az utóbbit a harmadikkal összevetve már gondolataink támadhatnak. A mai korszerűnek mondott adatkezelők használatakor ki lehet jelenteni, hogy az adat értéke lehet-e üres vagy sem. Viszont a kétféle üres tartalom („egyszer sem” - „nem tudom”) szétválasztására nincs mód. Ezért az adatbázisban lévő Szülésszám tulajdonság nulla értéke jelentheti azt is, hogy Zsóka még fiatal, nem volt ideje, lehetősége stb. a szülésre, no meg azt is, hogy Kati adatainak a bevitelekor nem ismerték az adat tartalmát vagy elfeledték azt kitölteni.

A fentiek alapján kétféle „üreset” kell megkülönböztetnünk. A valódit, amit ***szignifikánsnak*** nevezünk („egyszer sem”) és az ismeretlen értéket, az ***inszignifikáns*** nullát („nem tudom”). Gondoljuk csak át a kétféle üres hallatlan tartalmi különbségét! Áttérve egy másik példára: érkezik a számlánkra X forint. Ezt az összeget Y különböző partnerünk fizette a részünkre. Egyesek átutalásain a forint-rovat elmosódott, ezért nem tudtuk az adattartalmat kitölteni. Az ilyenkor üresen maradt mező nem nullát jelent - hiszen a forint már nálunk van -, hanem azt, hogy nem ismeretes annak pontos tartalma. Az pedig teljesen világos, hogy másként kell azt az ügyfelet kezelni, aki valóban nem fizetett és azt, aki azt megtette, de akárki hibájából a konkrét érték rejtve maradt.

Mivel a mai adatkezelők és adatkezelési módszerek nem nyújtanak lehetőséget a szignifikáns és inszignifikáns üresek szétválasztására (például úgy, hogy az utóbbit egy külön karakterrel jelölik), adatbázisainkban és adatkezelő programjainkban kisegítő adatokat kell alkalmaznunk, amelyek mutatják az üres jellegét. Ha ezt nem tesszük, helytelen ismereteket kapunk és a kommunikáció is tévútra vezet. Például megsértünk egy fontos ügyfelet.

A negyedik párbeszéd még tovább vezet bennünket. Feri esetében nem lehet a szülések száma adott érték, de szignifikáns vagy inszignifikáns nulla sem. Feri ugyanis nem nőnemű. Ezért az adat tartalma az ő esetében nem ilyen vagy olyan üres, hanem egyáltalán nincs értelme. A tulajdonság Ferire nézve ***nem-értelmezhető***, nem-alkalmazható (angolul: not-applicable, n. a.). A statisztikai gyakorlatban sokszor találkozunk ezzel az „n. a.” megjelöléssel, amely eredetileg valóban az ismeret lehetetlenségére (férfi nem szülhet) utalt. Később azok, akik keveset értenek az ismeretekhez, nálunk összemosták a két lényeget. Szellemesnek vélve az átfordítást, az „n. a.” betűpárost a „nincs adat” rövidítésére használják. Ez nagy hiba.

Statisztikai szempontból valóban csak az a fontos, hogy a nem-valódi üreseket meg tudjuk különböztetni a valódiaktól. Ha átlagos szülésszámot számítunk, kiemeljük a személyek közül azokat, akiknél a Szülésszám tartalma ismeretlen és azokat is, akikre nézve nem értelmezhető. Ezért a statisztika szempontjából lényegtelen az inszignifikáns nulla és a nem-értelmezhető érték szétválasztása.

Arról viszont szinte teljesen felesleges győzködni bárkit is, hogy három különböző jelenség a „nő, aki nem szült”, a „nő, akiről nem tudjuk, hogy hány gyermeket szült” és a „férfi”, aki eleve képtelen erre a kemény és szép feladatra. Márpedig az adatbázis szerkezeti kialakításakor nem az időben és térben korlátos (pl. statisztikai) igények, az ún. külső szemléletek (ld. 5.3 pont) a meghatározóak. A cél a valós jelenségek lehető legtermészetesebb tükrözése. Ezért a fogalmi adatmodellben nagyon pontosan szét kell választani a szignifikáns, inszignifikáns és nem-értelmezhető üres értékeket felvevő tulajdonságtípusokat. Erre azért is szükség van, mert a harmadik fajta „üres” egy igen fontos modellezési tényezőnek az alapja.

7.10 Egyedaltípusok

Az 1:1 fokú homogén viszonyok áttekintése kapcsán láttuk, hogy nem célszerű külön FÉRJ és FELESÉG egyedet kijelölnünk az adatbázisban. Az egyetlen elvi SZEMÉLY egyed ilyesféle megbontásának számos negatív következménye van. Viszonyt kell meghatározni és kezelni a két egyedtípus között. Bonyolultabbá válik az emberekre - a férjekre és a feleségekre egyaránt - vonatkozó ismeretek kikeresése például a születési dátum, lakhely, foglalkozás stb. szerint. Meg kell duplázni a karbantartási funkciókat stb.

Az egyedekhez számos leíró tulajdonság kapcsolódhat. Ezek az egyedelőfordulásokra eltérő értékeket vehetnek fel. A konkrét egyedek nem egyformák; sokféle módon osztályozhatók. Ez a

tény még nem elég alap az egyedtípus ilyen vagy olyan szétdarabolására. Mert ha valaki ma férj és feleség egyedtípust kreál, akkor holnap másnak eszébe juthat kovács, színész, kiskereskedő egyedtípusokat alkotni. Holnapután pedig valaki a januári, februári stb. születésű egyének egyedtípusait fogja meghatározni. Mindebből óriási káosz keletkezhet.

Az egyedtípusok megbontásával csínján kell bánnunk. Vannak azonban olyan helyzetek, amelyekben az előre elképzelt egyedtípus tulajdonságtípusait mégiscsak érdemes felosztani több egyedtípus között. A 7.10 ábra mutat egy ilyen példát.

RENDELÉS

Rendelésszám	Rendelésirány	Devizanem	Devizaszorzó
1	hazai		
2	export	dollár	X
3	export	márka	Y
4	hazai		

7.10 ábra: Egy megbontható egyedtípus

Amint látjuk, vannak hazai és export rendelések. Az előbbieknél a devizával kapcsolatos adatok nem üresek - mint már tudjuk -, hanem értelmezhetetlenek. Az adatkezelő ezek helyére „nullát” ír, ami - a híreszteléssekkel ellentétben - tárt foglal és növeli a kezelési időt.

Ha egy egyednek sok olyan tulajdonsága van, amely valamilyen további tulajdonságtól függően nem-értelmezhető tartalmú, akkor megfontolhatjuk az egyed megbontását. Tudjuk, hogy a „sok” nem éppen egzakt megjelölés. Azonban a megbontás nem kényszerű, hanem csak lehetséges adatmodellezési megoldás. Gyakorlati útmutatás: Megbontást akkor szoktak alkalmazni, ha az egyed tulajdonságainak a száma meghaladja a tízet és azoknak legalább egyharmada nem-értelmezhető jellegű.

D 7/5 **Az egyedtípusnak a nem-értelmezhető tulajdonságtípusok szerinti megbontását specializációnak, az eredmény-egyedtípusokat az eredeti egyed altípusainak nevezzük.**

A 7.10 ábra megbontásának az eredményét a 7.11 ábra mutatja. A specializált EXPORT-RENDELÉS egyedtípus a RENDELÉS altípusa. (Az altípus-egyed nevének nem kell az eredeti egyed nevéből származnia, bár az segíti az értelmezést.) Az egyedaltípus az eredeti egyed nem-értelmezhető tulajdonságait tartalmazza leíró tételekként úgy, hogy azok tartalma már mindig értelmezhető. A megbontás módja egyszerű, csak a két egyed viszonyát kell jól meghatározni.

RENDELÉS

Rendelésszám	Rendelésirány
1	hazai
2	export
3	export
4	hazai

EXPORT-RENDELÉS

E-Rendelésszám	Devizanem	Devizaszorzó
2	dollár	X
3	márka	Y

7.11 ábra: Egyedtípus és -altípus

A specializáció értelméből következik, hogy az egyedaltípus ugyanazt a lényeget fejezi ki, mint a megbontott egyed. Ezért törvényszerű, hogy az új egyed elsődleges kulcsának az eredetiével elvileg azonosnak kellene lennie. Azonban az azonosítókra vonatkozó korlát (vö. K4) ezt a

lehetőséget kizárja. A problémát ismét a szerepnév oldja fel: a specializált egyed azonosítója az eredeti egyed kulcsának a ***tipizáló*** szerepneve. Az ilyen szerepnév minősít, korlátoz az alapegyed valamelyik tulajdonsága szerint (itt: Rendelésirány) és egyben szétválasztja az altípusokat.

Az utóbbi kitétellel kapcsolatosan megjegyzendő, hogy a specializáció általában több altípust eredményez. Például: vállalatunknak vannak partnerei, akiket a standard adatokon kívül olyan tulajdonságok is jellemeznek, amelyek csak a vevőkre vagy csak a szállítókra vonatkoztathatók. Ekkor a Vevő-partnerkód és a Szállító-partnerkód az eredeti PARTNER egyed két altípusát valóban szétválasztja.

A specializáció feltétele, hogy az egyedaltípusok egyedelőfordulásainak halmazai egymást kizárják. Ellenkező esetben ugyanis redundancia lépne fel. Ez a korlát sokszor kellemetlen. Ha valamelyik partnerünk egyszerre vevő és szállító, akkor bizony háromféle altípust kellene kijelölni: vevő, szállító és vevő/szállító. Ebben az esetben a kezelés igen nehézkessé válna. Ezért általában csak akkor alkalmazunk specializációt, ha az altípusok tényleg külön valós jelenségeket tükröznek.

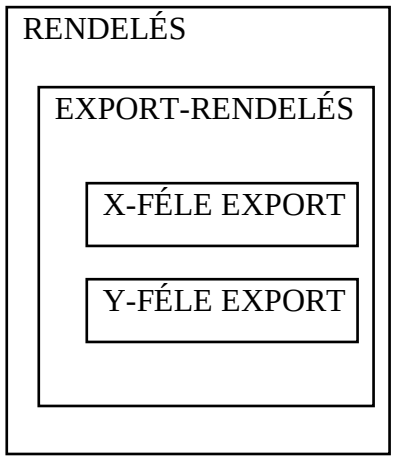
Csak röviden térhetünk ki arra a korlátra, amely szerint az altípus csak egy eredeti egyedhez kapcsolódhat. Számos ok miatt a mai modellezési elmélet kizárja, hogy egy jelenséget több másik altípusaként határozzunk meg. Emiatt a motorcsónak vagy csak a motoros-, vagy csak a vízi-járművek altípusa lehet, a kettőé egyszerre nem. (Pedig a valóságban mindkettő altípusa.)

Az egyedaltípus az eredeti egyedtípushoz 1:1 fokú, alulról kötelező, felülről opcionális kapcsolattal kötődik. Minden rendeléshez csak egy export-rendelés kiegészítés tartozhat és megfordítva. Minden export-rendelés meg kell, hogy feleljen egy rendelésnek, de ez fordítva nem igaz. Az ilyen viszony nem birtoklási, hanem ***altípus*** természetű, amit ***„is-egy”*** (angolul: is-a) kapcsolatnak hívunk. Mert hiszen az export-rendelés is egy rendelés.

Az altípus természetű kapcsolatokat nemcsak a sajátos kapcsolódás, hanem az ún. ***öröklődés*** (angolul: inheritance) miatt is meg kell különböztetnünk a birtoklásiaktól. Az egyedaltípust nemcsak a saját tulajdonságsora jellemzi, hanem az eredeti egyedtípusé is. Példa: a VEVŐ a PARTNER altípusa. Ezért speciális tulajdonságain kívül jellemzi a vevőre és szállítóra egyaránt vonatkozó közös Partnernév is.

A fentiekben a ***részleges*** specializációról volt szó, amelyben az eredeti egyed „leadja” egyes tulajdonságait egy vagy (általában) több altípus egyednek, miközben saját maga is fennmarad. Ritkán, de előfordul a ***teljes*** specializáció is. Ebben az esetben az eredeti egyedben csak az azonosító és a tipizáló tulajdonság maradna meg. Az ilyen egyednek nincs létjogosultsága, tehát a specializáció következtében az eredeti egyed megszűnik. Ettől fogva nem is beszélhetünk egyed-altípusokról, kapcsolatról, öröklésről stb. Néha mesterségesen is alkalmazzuk a teljes specializációt. Például a gépkocsik tulajdonosainak az esetében. Természetük szerint CÉG és SZEMÉLY egyedtípusokat képezünk az eredeti egyetlen TULAJDONOS egyedből annak ellenére, hogy a cégeknek és a személyeknek egyaránt van például neve.

Az ilyen mesterséges specializációval vigyázni kell, amint a FÉRJ és FELESÉG példáján láttuk. Nagyon sokszor éppen az ellentétes műveletet kell végrehajtanunk. Ha azt látjuk, hogy két egyedtípus tulajdonságsora nagymértékben átfed, akkor meg kell fontolnunk azok összevonását. A műveletet ***generalizációnak*** nevezzük. Ez is lehet teljes vagy részleges. A FÉRJ és a FELESÉG egyed teljesen megszűnik a SZEMÉLY-be való összevonáskor. Viszont a VEVŐ és a SZÁLLÍTÓ közös adatainak a PARTNER-be való emelése után azokban is maradhatnak tulajdonságok, tehát a részleges generalizációval valódi egyedaltípusok képződnek.

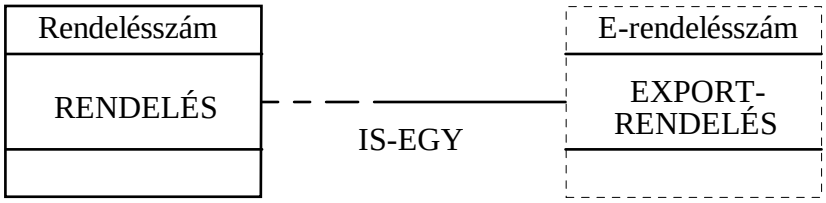


7.12 ábra: Az egyedtípus és -altípus egyik ábrázolása diagramon

Az adatmodelldiagramokon külön illik jelölni az egyedaltípusokat. Erre két konvenciót alakítottak ki. Újabban az a szokás, hogy a specializált alapegyedet (pl. RENDELÉS) jelképező dobozon belül tüntetik fel az altípus-egyedét (pl. EXPORT-RENDELÉS). Lásd a 7.12 ábrát.

Ennek a megoldásnak számos hátránya van. Egy típusnak számos altípusa lehet. Altípusonként nyilván eltérőek a fontos tulajdonságok. Nagyon gyakran az altípus egyednek más a külső szerkezete - kapcsolattípusainak az együttese - mint a típusé. Az altípusnak is lehet altípusa. Mindezeket a viszonyokat egy dobozon belül áttekinthetően ábrázolni gyakorlatilag lehetetlen. Arról nem is beszélve, hogy miként tükröznénk azt a - jelenleg meg nem engedett - struktúrát, amelyben egy altípus több egyednek lehetne a specializáltja.

A fentiek miatt mi egy másik konvenciót részesítünk előnyben. Az egyedaltípusok dobozait szaggatott vonallal jelöljük, egyébként pedig az altípust a szokásos módon tükrözzük. Lásd a 7.13 ábrát.



7.13 ábra: Az egyedtípus és -altípus másik ábrázolása diagramon

7.11 Szerkezeti összefoglalás

Az adatbázisok szerkezeti felépítését lassan kezdjük megismerni. Egyrészt láthattuk, hogy az adatmodell valóban az egyedek, tulajdonságok és kapcsolatok tudatosan szervezett együttese. Másrészt viszont rá kellett döbbennünk arra, hogy a „tudatos szervezés” rengeteg tényező közös átgondolását és meghatározását igényli. Az egyed tulajdonságsorát nem kényünk-kedvünk szerint

állítjuk össze. A tulajdonságokat általános érték-tartományoknak kell megfeleltetnünk, ha a tulajdonságokban nemcsak a különbségeket, hanem az azonosságokat is látni akarjuk. Kiderült az is, hogy az egyedviszonyok meglehetősen bonyolult tényezők. Már egyáltalán nem arról van szó, hogy két egyed bármilyen közös tulajdonsága szerint összekapcsolható (amint azt a relációs modellben feltételezik). Két egyed viszonyát sokszor egy harmadik valósítja meg. Egy egyedtípus saját magával is kapcsolatba hozható. Nem közömbös a viszony foka, opcionalitása és természete.

Mivel időközben újabb összefüggéseket is felfedeztünk, úgy illik, hogy a kapcsolat lényegét végleg tisztázzuk:

D 7/6 **Két egyedtípus kapcsolattípusban áll, ha az egyik azonosító tulajdonsága vagy annak szerepneve a másik egyed kapcsoló tulajdonsága. Kapcsolattípust létesít az egyedtípus saját magával, ha kapcsolóként tartalmazza azonosítója szerepnevét.**

A korábbiakban láttuk, hogy a kapcsoló tulajdonság eredetileg leíró vagy kulcsrész szerepű lehet. Az altípus természetű kapcsolatban a kapcsoló tulajdonság azonosító szerepű és - szemben a birtoklási kapcsolattal - mindig szerepnév.

Az utolsó kitétel is mutatja, hogy az adatmodell struktúrája igen pontos szabályokon alapul. Még számos összefüggést lenne érdemes megvilágítanunk, de a fárasztó ismertetést ideje abbahagynunk. A fejezetre szánt helyünkből csak egy utolsó gondolatfutamra telik.

Az előző fejezetben és a jelenlegiben az adatmodell elméleti szerkezeti tényezőivel és azok viszonyaival foglalkoztunk. Gyakorlatilag a mai adatbáziskezelők zöme a bemutatott struktúrák felét sem tudja kezelni. Mi értelme van akkor a „bonyolult” elméletnek? Erre a kérdésre többféle választ kell adnunk.

Először: Nem hisszük, hogy az eddig elmondottak olyan nagyon bonyolultak lennének. Ne vállalkozzon adatbázistervezésre az, aki ezt a pár alapvető összefüggést sem látja át, mert rossz adatbázist fog építeni.

Másodszor: A gyakorlati adatkezelésben időnként történtek elméleti visszalépések. Például a relációs modell elvi képességei messze „alulmúlják” a sokkal korábbi hálós modell ugyancsak elvi adatszerkesztési lehetőségeit. Ennek dacára a gyakorlati kezelők fejlődnek és egyre újabb meg újabb vonásokat vesznek át a fentiekben felsorolt képességekből. Tehát az elmélet elhanyagolása egyet jelentene azzal, hogy nem készülünk fel a jövőbeli gyakorlatra.

Harmadszor: ***A fogalmi adatmodell nem (csak) arra való, hogy azt valamilyen adatkezelő alatt egy az egyben megvalósítsák.*** Hanem arra (is), hogy feltárjuk ismereteink természetét. A tények pedig makacs dolgok. Ha az adatkezelő nem képes automatikusan érvényesíteni egy-egy korlátot (például a visszamutató kapcsolatot), akkor magunknak kell megírnunk a megfelelő programrészletet. Ám miképpen tehetnénk ezt, ha nem ismerjük magukat a jelenségeket és nem tudjuk, hogy egy tipikus vonás esetében általában mi a teendő? Ezért a fogalmi adatmodell pontos leírására akkor is szükség van, ha az adatkezelőnk korlátos képességű.

Ellenőrző kérdések - 7

7/01 Helyes (H) gyakorlatot jelentenek-e vagy sem (N) az alábbi megoldások:

- A RENDELÉSTÉTEL-ben a rendelésszámnak T-Rendelésszám a neve.
- A RENDELÉSTÉTEL-ben a mennyiséget az R-Mennyiség, a SZÁLLÍTÁSTÉTEL-ben az S-Mennyiség névvel illetjük.
- A RENDELÉS és a SZEMÉLY egyedben egyaránt van Dátum nevű adat.

7/02 Melyik igaz (I) és melyik hamis (H) megállapítás:

- A Rendelésdátum és a Szállításdátum egymás szerepnevei.
- Ha létezik a Dátum tartomány, akkor az előző két adat annak a szerepneve.
- A Születés-kelte lehet a Dátum tartomány szerepneve.
- A Hossz nem biztos, hogy a Méret tartomány szerepneve.

7/03 Melyik igaz (I) és melyik hamis (H) kitétel:

- A Dátum validáló rutinját a séma Dátum tartományához kell kötni.
- A Dátum validáló rutinját az egyedi programokban kell megadni.
- A Dátum validáló eljárását rutinként megírjuk, majd a programokba visszük.

7/04 Az alábbiak közül melyik jó (J) és melyik rossz (R) megoldás. Van szállás-költség, napidíj és dologi költség.

- KIKÜLDETÉS (***Személy***, Költség-1, Költség-2, Költség-3)
- KIKÜLDETÉS (***Személy***, Szállásköltség, Napidíj, Dologi)
- KIKÜLDETÉS (***Személy+Költségtípus***, Költség)

7/05 Hogyan tükrözné az osztályfőnök és a diákok viszonyát? Az alábbiak közül melyik megoldás jó (J) és melyik rossz (R)? Vigyázzon!

- SZEMÉLY (***Személy***, Tanár/Diák-jel, *Ofő*) az Ofő a Személy szerepneve
- TANÁR (***Tanárazonosító***, ...)
DIÁK (***Diákazonosító***, ...)
OSZTÁLY (***Tanárazonosító+Diákazonosító***, ...)
- SZEMÉLY (***Személy***, Tanár/Diák-jel)
OSZTÁLY (***Tanárazonosító+Diákazonosító***, Ofőjel)
A két azonosítórész a Személy szerepneve.

7/06 Az alábbi két megoldás közül melyik tükrözi helyesen a fix szervezetek (S) és az egymásba fonódó projektek (P) ismereteit. Mindkettő részeit egységnek nevezzük.

- EGYSÉG (***Egységkód***, ..., *Fölérendelt-egységkód*)
- EGYSÉG (***Egységkód***, ...)
KAPCSOLAT (***Egységkód-1+Egységkód-2***, ...)

7/07 Mi a teendő a következő tervvel? A helyes válasz számát adja meg. Két jó válasz is van.

ZENESZERZŐ (Szerzőkód, Szerzőnév, Szerzőcím, Sz-adatok)
SZÖVEGÍRÓ (Írókód, Írónév, Írócím, Í-adatok)
Az Sz-adatok csak a szerzőt, az Í-adatok csak az írót jellemzik.

- A terv így kifejező, ezért jó.
- A két egyed első három adata generalizálható a SZEMÉLY egyedbe.
- A két egyed mindig teljesen generalizálható.
- A két egyed teljesen generalizálható a SZEMÉLY egyedbe, ha az Sz- és Í-adatok száma nem nagyobb háromnál.

7/08 Ön tanár és nyilvántartást vezet a diákjairól illetve a barátairól. A diákok között is vannak barátai ill. megfordítva. Adja meg a helyes válasz(ok) számát:

- Készítek egy DIÁK és egy BARÁT állományt.
- Egyetlen SZEMÉLY egyedet képzelek el n.a. értékekkel.
- Az előbbit teszem, ha a diákspecifikus jellemzők száma nem jelentős.
- Készítek egy SZEMÉLY egyedtípust, meg egy DIÁK és BARÁT altípust, ha a specifikus jellemzők száma jelentős.
- Három állományom lesz: DIÁK, BARÁT és DIÁKKÉNT-BARÁT.

8. ADATBÁZISKEZELÉS

8.1 Az ismeretek természete

Az előző fejezetekben számtalanszor használtuk az „adatkezelés” és „adatkezelő rendszer” kifejezéseket anélkül, hogy megmagyaráztuk volna azok mibenlétét. Úgy gondoljuk, hogy ezen fogalmak lényege mindenki számára ismert, és ezért pontos definíciójuk hiánya nem zavarta az eddigiek megértését. Most azonban rá kell mutatnunk az ***adatkezelés*** és az ***adatfeldolgozás*** különbségére, viszonyára illetve az ismeretszerzésben betöltött szerepére.

Az 1. fejezetben részletesen taglaltuk az ismeretszerzés közvetlen módját (észlelés, érzékelés stb.). A második mód az eddig elmondottakból adódik: ismeretre tehetünk szert az adatbázisban tárolt adatok lekérdezésével is. Ehhez viszont arra van szükség, hogy az adatokat betöltsük, azokat frissen tartsuk, elrendezzük, szelektáljuk stb. - egyszóval: kezeljük. Azonban az ismeretszerzésnek van egy harmadik mindennapos módja is: a már meglévő adatainkon műveleteket hajtunk végre, vagyis adatfeldolgozást végzünk.

A fentiekből következik, hogy az adatoknak két fajtáját kell megkülönböztetnünk. Vannak ún. ***alapadatok***, mint például a termék megrendelt mennyisége és egységára. Ha az alapadatokon műveleteket hajtunk végre - például megszorozzuk a mennyiséget és az egységárat - akkor új, ún. ***származtatott*** adatot kapunk. Az eléggé világos, hogy az alapadatoknak az adatbázisban van a helye. Viszont kérdés, hogy miként viszonyul egymáshoz az adatbázis és a származtatott adat.

Ennek a fejezetnek az a célja, hogy az adatkezelés és az adatfeldolgozás elhatárolása után bemutassa a származtatott adatok és az adatbázis viszonyát.

A mindennapos gyakorlatban sokszor előfordul két hiba. A fejlesztők arra kényszerítik a felhasználókat, hogy olyan adatokat vigyenek az adatbázisba, amelyek természetük szerint származtatottak. Magyarul: a számítógép helyett kézzel kell elvégezni az adatokon bizonyos előzetes műveleteket. Ennek általában az az oka, hogy az adatbázis építését nem a megfelelő ponton kezdték el és így nem állnak rendelkezésre a számítógépen a kellő alapadatok. A téma bővebb kifejtése nélkül most csak utalni tudunk arra, hogy az adatbázis kialakításának van egy ésszerű sorrendje.

A másik gyakori hiba az, hogy az adatfeldolgozást nem visszük végig a megfelelő pontig. Nem dolgoztatjuk meg eléggé a számítógépet és nem produkálunk könnyen információvá értelmezhető ismeretet a felhasználó részére.

A fejezetben arra is rá fogunk mutatni, hogy a fejlesztők előszeretettel használnak ún. ***technikai*** adatokat. Ezek nem a felhasználó számára fontos ismereteket tükröznek. Ezért kérdéses, hogy szabad-e őket alkalmazni az adatbázisban és ha igen, akkor miként.

8.2 Az adatkezelés szintjei

Az adatszerű ismeretkezelés alapja az, hogy adatainkat címkézett - tulajdonságnévvel ellátott - „dobozokban” tároltan képzeljük el. Egy-egy jelenségcsoporthoz - egyedtípushoz - több ilyen

skatulya tartozhat. Például a KOCSI egyed két tulajdonsága a Szín és a Kocsitípus. A „fehér” ismeretet az előbbi dobozba tesszük, a „Lada” értéket pedig az utóbbihoz rendeljük. Így olyan táblázatot kapunk, amely oszlopokból (tulajdonságtípusok) és sorokból (egyedelőfordulások) áll. Egy ilyen táblarészletet mutat a 8.1 ábra a korábbi konvencióinknak megfelelő formában:

KOCSI

Rendszám	Szín	Kocsitípus
X	fehér	Lada
Y	zöld	Lada
Z	fehér	Polski
Q	piros	BMW
W	piros	Polski

8.1 ábra: Egy teljesen szokványos adattáblázat

A táblákban vertikálisan helyezkednek el a tulajdonságtípusok. Ezek sorozatát (Rendszám, Szín, Kocsitípus) a tábla ***fogalmi tartalmának*** (angolul: intension) nevezzük. A tábla sorainak a horizontális együttesét ***fogalmi kiterjedésnek*** (angolul: extension) hívjuk. Természetesen a két irányt csak képszerűen kell érteni; azokat fel is lehetne cserélni. Mivel a kiterjedés mérete (az egyedelőfordulások száma) általában nagyobb, mint a tartalomé (a tulajdonságtípusok száma), az adatmodellezési szóhasználatban a fenti irányjelzés terjedt el. Mert hiszen a papír hossza is általában nagyobb, mint a szélte.

A 8.1 ábra által mutatott kép csak az emberben él, és mint az előbbi meghatározások is mutatták, fogalmi szintű elképzelés. A tényleges adatbázis elrendezése sokszor még csak nem is emlékeztet az itteni struktúrára. Gyakran előfordul, hogy a logikai tervezés során a táblázatokat ilyen vagy olyan okok miatt (ld. 4.4 pont) szétdaraboljuk. Ezt a megbontást ***logikai particionálásnak*** nevezzük. Ez a megbontás tartalom és/vagy kiterjedés szerint történhet.

A 8.2 ábra a ritkább intenzionális, a 8.3 ábra a gyakoribb extenzionális particionálást mutatja. Az is előfordul, hogy a két megoldást kombinálják. A megbontás a típus szinten adatismétléssel (angolul: data replication) jár, hiszen az azonosítót (Rendszám) már több állomány tartalmazza. Az intenzionális particionálásnál viszont kerülni kell az érték szintű adatreplikálást, mert az kellemetlen kezelési gondokat okozó redundanciát jelent (ld. a Z rendszámú kocsi adatait a 8.3 ábrában).

A tulajdonságsor megbontásának egyik gyakori oka az, hogy az adatkezelő rendszer „nem szereti” a széles - sok tulajdonságtípust tartalmazó - táblát, kezelési hatékonysága romlik egy bizonyos méret felett. A másik ok, amely az előfordulások szerinti megbontásnak is a fő indoka, a (földrajzi) elosztottság. Egyes gépkocsik adatait az „X”, másokét az „Y” helyen kell kezelni.

KOCSI-1

Rendszám	Szín
X	fehér
Y	zöld
Z	fehér
Q	piros
W	piros

KOCSI-2

Rendszám	Kocsitípus
X	Lada
Y	Lada
Z	Polski
Q	BMW
W	Polski

8.2 ábra: Tartalom szerinti megbontás

KOCSI-3

Rendszám	Szín	Kocsitípus
X	fehér	Lada
Y	zöld	Lada
Z	fehér	Polski

KOCSI-4

Rendszám	Szín	Kocsitípus
Z	fehér	Polski
Q	piros	BMW
W	piros	Polski

8.3 ábra: Terjedelem szerinti megbontás

A táblázatok tartalma a valóságban sem horizontálisan, sem vertikálisan nem úgy kerül elrendezésre a számítógépen, mint ahogyan azt látjuk. Az adatkezelő rendszerek a táblákat mindkét irányban megbontják. A tárolón a Kocsitípus tartalmának karaktersora a Szín értékétől távol is elhelyezkedhet és egyáltalán nem biztos, hogy a Q kocsi adatai a Z kocsi ismereteit közvetlenül követik. Tehát a táblák nemcsak logikailag - tartalmilag - bonthatók meg, hanem a konkrét ismeretek is szétdaraboltan foglalnak helyet a táron, azaz ***fizikailag particionáltak***.

Végeredményben van egy fogalmilag egységes adatbázisunk (8.1 ábra), amelyet logikailag megbontunk (8.2 és 8.3 ábra) - többnyire a földrajzi körülmények miatt - és amelynek részei szanaszét szórva találhatók a tényleges fizikai adathordozókon. Ha tehát az adatkezelés lényegét akarjuk firtatni, akkor egy percre sem szabad elfeledkeznünk a három szintről és arról a tényről, hogy ismereteink kezelésében két vagy három alaprendszer működik közre.

Adatainkat a fizikai tárakon ténylegesen az ***operációs rendszer*** helyezi el és onnan az keresi vissza. Ha nem használunk adatkezelőt, akkor az elhelyezési- és hozzáférési mód (ld. 4.6 pont) kiválasztásával magunk utasítjuk programjainkban a működtető rendszert az adatok adott módon látszódó elhelyezésére. Ezzel a mélyszintű adatkezeléssel nem foglalkozunk.

Szerencsére ma (speciális feladatoktól eltekintve) nincs szükségünk arra, hogy közvetlenül érintkezzünk az operációs rendszerrel. Az ***adatkezelő rendszerek*** (angolul: data management system) garmadája áll a rendelkezésünkre. Programjainkban ezt az eszközt kérjük fel az ismeretek elhelyezésére és visszakeresésére. Az adatkezelő utasítja az operációs rendszert igényeink teljesítésére.

Amennyiben adataink földrajzilag elosztottak - vagyis nem egy gépen találhatók -, úgy szükség van a harmadik eszköz közreműködésére is. Az ***adatátviteli rendszer*** (angolul: data communication system) segít bennünket az adatok „távoli” elhelyezésében és onnan való visszakeresésében. Ezzel a szoftverrel sem érintkezünk közvetlenül: ez az adatkezelő feladata. Könyvünkben az adatátviteli rendszer által végrehajtott adatkezelési funkciókról nem lesz szó. Végeredményben a továbbiakban csak azt fogjuk vizsgálni, hogy miként használjuk az adatkezelő rendszert és az milyen feladatokat végez el a számunkra.

8.3 Történeti kitekintés: Az adatbázisgépek

A korszerű adatkezelő rendszerek ***osztott*** (angolul: shared) használatúak, vagyis egyidejűleg több felhasználót is kiszolgálnak. A legmodernebbek ***többfonalas*** (angolul: multi-thread) módon működnek. Ez azt jelenti, hogy nem a kezelési igényeket (a tranzakciókat) állítják sorba, hanem az elemi be/kimeneti műveleteket. A W gépkocsi adatait az Y felhasználó azonnal (a másodperc töredéke alatt) elérheti, miután azokat az X felhasználó kezelte feltéve, hogy az utóbbi nem alkalmazott valamilyen speciális zárat (ld. 5.5 pont).

Ily módon a mai adatkezelők megismétlik az operációs rendszer feladat-, tár-, erőforrás- stb. menedzselési funkcióit a maguk szintjén. Ez nyilvánvalóan pazarló megoldás. (Nem szólva arról, hogy a rosszul tesztelt adatkezelők ilyen replikált funkciói összeakadhatnak az operációs rendszerével.)

Ezért a hetvenes évek vége óta kísérleteznek olyan rendszerekkel, amelyekben a két alapszoftver - az operációs rendszer és az adatkezelő - funkcióit egyesítik. A számítógépnek nincs külön operációs és adatkezelő szoftvere: az utóbbi látja el az előbbi feladatait is. Az ilyen módon működő technikát nevezik ***adatbázisgépnek***. Feltehetően piaci okok miatt nem terjedt el ez a megoldás, hiszen csak igen nagyméretű és egy helyen tárolt adatbázis esetén fizetődik ki. Azonban számítanunk kell arra, hogy nem is hosszú időn belül az operációs, az adatkezelő és a kommunikációs szoftverek képességeit egyesíteni fogják, mert e tényezők összehangolatlansága lassan már világra szóló skandalum.

8.4 Adatkezelés és adatfeldolgozás

A mindennapos felhasználó és a kezdő szakember sok bajjal kell, hogy megküzdjön, amíg meg nem érti az alcímben jelzett két lényeg különbségét. Megveszik valahol az X adatkezelő rendszert, majd megdöbbennek azon, hogy a beszerzett szoftverrel nem képesek új „információ” létrehozására többé-kevésbé összetett alkalmazási programok megírása nélkül. Ezért nagyon fontos, hogy világosan lássuk az adatkezelés és adatfeldolgozás különbségét és összefüggését.

Az adatkezelő rendszerek az ismeretek dimenzióival (ld. 2.2 pont) foglalkoznak és nincs közük az ismeret tartalmi mondanivalójához. Képletesen szólva az adatkezelőnek „mindegy”, hogy Rózsa kocsija fehér vagy rózsaszínű. Csak annyit kell tudnia, hogy van KOCSI egyedtípus, amelynek a Szín az egyik tulajdonságtípusa, ami bizonyos értékeket vehet fel. Vagyis az adatkezelő kizárólag ***adatszerkezeteken*** dolgozik.

Adatkezelési művelet az ismeret bevitele, módosítása, törlése, képernyőn való megjelenítése, nyomtatóra történő kiírása, válogatása, rendezése, másolása, kimentése, áttöltése stb. Az ilyen műveletek során nem születik új ismeret, csak a meglevők elrendezésére kerül sor.

Néhányan megütköznek ezen a kijelentésen és felvetik a kérdést: „Ha beviszem egy új kocsi adatsorát, módosítom a Szín tartalmát, törlöm az egyik kocsi adatait - ez nem jelent új ismeretet?” Nem! A felhasználó már birtokában volt az ismeretnek, különben nem tudta volna azt bevinni, módosítani vagy törölni. Nem az adatkezelő „szülte” az ismeretet. Ő csak elrendezte azt a négy dimenziónak megfelelően.

D 8/1 **Adatkezelési műveleteknek hívjuk azokat az adatokon végrehajtott (számítógépes) tevékenységeket, amelyek során új ismeret nem születik.**

A fentiekben nem véletlenül használtuk az „ismeret” szót. Ha valaki lekérdezi az adatbázist - adatkezelési művelettel -, akkor információt nyerhet pusztán a megjelenítés, az elrendezés, a válogatás következtében. Az egyén információra tehet szert az ismeretek pusztán formai tálalási módján keresztül, mindenféle tartalmi adatátalakítási művelet nélkül is.

Most vessünk egy pillantást a 8.4 ábrára!

RENDELÉSTÉTEL

Rendelésszám	***Cikkszám***	Mennyiség	Ár
X	***A***	80	50
X	***B***	70	25
Y	***A***	30	50

8.4 ábra: RENDELÉSTÉTEL egyedtípus

A kérdés az, hogy mennyit kell fizetni az első rendeléstételért? A Tételérték nevű tulajdonság nem szerepel az adatbázisban. Ezért tartalmát nem kaphatjuk meg az adatbázis átrendezésével és valamilyen módon való megjelenítésével, vagyis adatkezelési művelettel. Az első rendeléstétel Mennyiség és Ár tulajdonságának a tartalmát össze kell szoroznunk ahhoz, hogy a kívánt ismerethez jussunk. A Tételérték az adott esetben 4000 (Ft).

Most már lehetőségünk van az adatfeldolgozás meghatározására:

D 8/2 **Adatfeldolgozási műveleteknek hívjuk azokat az adatokon végrehajtott (számítógépes) tevékenységeket, amelyek során új ismeret születik.**

A fentiek értelmében az ismeretek két osztályba sorolhatók azok megszerzési módja szerint. Egyrészt vannak megfigyelés vagy közlés által nyert ismeretek. Ránézünk a kocsira és megállapítjuk, hogy a színe fehér. Beállít az ügyfél és közli, hogy a neve Rózsa. Másrészt vannak feldolgozási művelet eredményeképpen kapott adatok, mint a Tételérték.

D 8/3 **Az észlelés vagy közlés által nyert ismereteket alapadatoknak, a matematikai és/vagy logikai műveletekkel kapottakat származtatott adatoknak hívjuk.**

Az itteni alapadat meghatározás nem azonos a programozásban megszokottal. Nem az adott program szempontjából tényezőnek tekintett tételt nevezzük alapadatnak. Nem a műveleti szerep, hanem az ismeret megszerzési módja az osztályozás szempontja. Ezért a Rendelésszám is alapadat, noha nem hajtunk végre rajta feldolgozási műveletet. A Tételérték pedig származtatott adat annak dacára, hogy például a Rendelésérték kiszámítási műveletében alaptényező. Ez a megkülönböztetés a következő pontban felvetett kérdéskör miatt fontos.

Az adatkezelés és az adatfeldolgozás viszonya egyértelmű az adatbázisok használatakor. Adatkezelés létezhet feldolgozás nélkül, de ez fordítva nem igaz. Vannak olyan adatbázisok, amelyeknek egyetlen célja a nyilvántartás és semmiféle származtatási műveletet nem végeznek az adatbázis tartalmán. Ilyen adatbázisok a tág értelemben vett „lexikonok”: a nyelvi szótárak, a lelőhely kimutatások, a könyv-, hanglemez-, szalag- regiszterek, a címjegyzékek stb. Viszont a Rendeléstétel értékét sohasem tudjuk kiszámítani, ha az Ár és Mennyiség adatot nem tároljuk az adatbázisban és nem keressük elő a szorzási művelet végrehajtása előtt.

A fenti gondolatmenetből egy roppant fontos következtetést kell levonnunk.

Az adatbázison alapuló információs rendszerben nem az adatfeldolgozás miként je, hanem az adatkezelés módja a fontos.

Ha az adatbázis jól szerkesztett, akkor abból minden ismeret előkereshető és szükség szerint feldolgozható. Ha az adatbázis szerkezete rossz, akkor bármilyen ragyogóak is a feldolgozási elképzeléseink, azokat nem fogjuk tudni megvalósítani.

8.5 Szabad-e ...?

Az adatbázisokkal kapcsolatosan mindig felvetődik az a visszatérő kérdés, hogy szabad-e az adatbázisban származtatott adatokat tárolni? Megengedhető-e a 8.5 ábra szerinti megoldás?

Amennyiben az ár nem alku és egyéb körülmények tárgya, hanem rögzített, úgy az Egységár már nem a RENDELÉSTÉTEL, hanem a CIKK tulajdonsága. Hiszen a 8.4 ábra jól mutatja, hogy ellenkező esetben redundanciával kellene számolni: többször tárolnánk az „A cikk ára 50” sort.

Mármost ha a RENDELÉSTÉTEL nem tartalmazná a Tételérték tulajdonságot és ki akarnánk számolni annak értékét, akkor a megfelelő tételből a Cikkszám alapján el kellene mennünk az azonos cikkszámú CIKK előforduláshoz és ki kellene keresnünk az egységárat. Az egyedtípusok közötti átmenet - amelyről alább még bővebben is szólunk - nyilván időt igénylő adatkezelési művelet. Ezért jogos a pont elején felvetett kérdés.

RENDELÉSTÉTEL

Rendelésszám	***Cikkszám***	Mennyiség	Tételérték
X	***A***	80	4000
X	***B***	70	1750
Y	***A***	30	1500

CIKK

Cikkszám	Egységár
A	50
B	25

8.5 ábra: Rendelés adatbázis részlete

A válasz az első megközelítésben teljesen határozott. A származtatott adat redundanciát jelent és ezért elvileg nem tárolható az adatbázisban. Gondoljuk csak meg, hogy visszamenőlegesen végrehajtunk egy áremelést az „A” cikk árát 50-ről 60-ra növelve! A redundancia miatt ekkor az adatbázis többszörös karbantartása szükséges: a RENDELÉSTÉTEL minden „A” cikkszámú sorában módosítani kell a Tételérték tartalmát.

Az elv világos, de a gyakorlat másként fest. Tegyük fel, hogy a Főnök tíz másodpercen belül tudni akarja megrendeléseink összértékét. Ehhez ki kell számítani minden rendeléstétel értéket és azt summázni kell rendelések szerint. Az adatbázis méretétől és a gép képességeitől függően ez a feldolgozás tíz másodpercnél hosszabb időt is igényelhet. Ezért nincs más út, mint tárolni a Rendelés-összérték adatot, amit minden rendeléstétel karbantartáskor újra ki kell számítanunk. Persze magát a Tételértéket nem fogjuk tárolni, aminek az okát mindjárt megvilágítjuk.

Mivel az adatbázisok mérete és a gépek teljesítménye különböző, nem lehet általános szabályt megfogalmazni arra nézve, hogy melyik származtatott adatot érdemes az adatbázisban tartani. Viszont meglehetősen világos, hogy melyik származtatott adatot nem érdemes tárolni. Erre a ***„plusz egy”*** szabály ad eligazítást. Nem célszerű a származtatott adatot az adatbázis részének tekinteni, ha alaptényezői az amúgy is kezelt tételben vannak vagy „plusz egy” hozzáféréssel elérhetők. A 8.4 ábra esetében a Tételérték nem szerepelhet a RENDELÉSTÉTEL egyedben, hiszen az abban lévő Mennyiség és Ár adatokból azonnal kiszámítható. A 8.5 ábra megoldása rossz, mert a tétel elérése után a vonatkozó cikk egy plusz hozzáféréssel megkapható.

A „plusz egy vagy több” hozzáférés kérdése figyelmünket az adatbáziskezelés sajátos vonásaira irányítja. Mielőtt rátérnénk a hagyományos állománykezelés és az adatbáziskezelés

különbségeinek a megvilágítására, felhívjuk a figyelmet a tárolt származtatott adatok egyik különleges vonására.

A 8.4 ábra példájában nem volt lehetőség a Tételérték adatkezelésére. Az ismeretet csakis adatfeldolgozás útján szerezhettük meg. Ha a Tételérték az adatbázis része, mint a 8.5 ábra példájában, akkor az ismeret adatfeldolgozás nélkül, egyszerű adatkezeléssel is megkapható. Arra azonban gondosan ügyelni kell, hogy a származtatott ismeretek adatkezelését a megjelenítés különböző módozataira korlátozzuk. Káoszra vezetne, ha a Tételérték tulajdonság tartalmát a felhasználó közvetlenül - a mennyiségtől és az ártól függetlenül - bevihetné, módosíthatná, törölhetné. Ezért gyenge adatkezelőnek kell titulálnunk azt a rendszert, amelyben nem lehet deklarálni a tulajdonság származtatott voltát és a származtatás automatikus módját, ezzel megakadályozva a származtatott adat karbantartását.

Itt említjük meg, hogy már a 70-es évek elején létezett olyan adatbáziskezelő (lásd pl. [17]), amelyben deklarálni lehetett származtatott adattípust is. A „függvénynek” hívott adattípushoz ki kellett jelölni a műveletet: például Tételérték = Mennyiség*Ár. A Tételérték nem került tárolásra. Az ismeret lekérdezése során az adatkezelő a függvény alapján mindig automatikusan számolta ki a megfelelő értéket.

8.6 Állomány- és adatbáziskezelés

Az adatkezelésnek két módja van. Erről már tettünk említést a 3.1 pontban. Most a kétféle kezelés különbségeit alaposabban is megvilágítjuk.

Az ***állománykezelés*** során az egy egyedtípusba tartozó, egy logikai állományt képező tételek halmazán dolgozunk. Ha az áruk ismereteire vagyunk kíváncsiak, akkor a CIKK állományban bogarászunk; ha a rendelt mennyiségek adatait kezeljük, akkor a RENDELÉSTÉTEL állomány előfordulásaival csinálunk valamit. A lényeg az, hogy a kezelés mindig egyféle táblát érint. Képletesen szólva az ilyen kezelés „vertikális”, az adott tábla kiterjedése (extension) mentén halad. Kikeressük az első cikket, majd a másodikat stb.

A hagyományos állománykezelés általában ***kötegelt jellegű***. Ha arra vagyunk kíváncsiak, hogy mennyi az „X” rendelés összértéke, akkor a RENDELÉSTÉTEL állományt végigtallózva kikeressük azokat a sorokat, amelyekben a Rendelésszám értéke „X”. Készítünk egy ún. átmeneti állományt, amit majd a CIKK állománnyal összeválogatunk a Cikkszám szerint és kibővítünk az Egységár és Tételérték adattal. Lásd a 8.6 ábrát. Ismételt állománykezeléssel az átmeneti fájl tételein végigballagva összesítjük a Tételérték tartalmát. Esetünkben 5750 lesz a Rendelésérték származtatott tulajdonságának az összege.

RENDELÉSTÉTEL-1

Rendelésszám	***Cikkszám***	Mennyiség	Egységár	Tételérték
X	***A***	80	50	4000
X	***B***	70	25	1750

8.6 ábra: Egy átmeneti állomány

Megjegyzés: Ha a Tételértéket tároljuk, akkor megtakarítjuk a kötegelt kezelés első műveletét és az átmeneti állományt. Ezen takarékosság jegyében hajlamosak tervezőink a származtatott adatok tárolására.

Lehet, hogy a fenti feladatot ma már csak nagyon kevesen hajtanák végre a leírt módon. Azonban még ma is gyakran alkalmaznak kötegelt feldolgozást és átmeneti állományokat olyan környezetben is, ahol már adatbáziskezelővel dolgoznak. Teszik ezt annak ellenére, hogy az adatbáziskezelés logikája nem azonos az állománykezelésével. Az előbbi több lehetőséget nyújt úgy, hogy magában foglalja az utóbbi képességeit is.

A valódi adatbáziskezelés ***tranzakcióorientált***. Ez azt jelenti, hogy a bemenetből kiindulva átmeneti állomány képzése nélkül vezetjük végig a kezelést a kimenetig. Példánk esetében a feldolgozás logikája a következő: Az „X" rendelés összértékére van szükség. Ezért keresni kezdjük a RENDELÉSTÉTEL állományban azokat a sorokat, amelyekben a Rendelésszám értéke „X". Ezt a műveletet állománykezelési módon, extenzionálisan végezzük. Amikor elérjük az első megfelelő sort, azt nem másoljuk ki átmeneti állományba, hanem felhasználjuk a CIKK - RENDELÉSTÉTEL kapcsolatot, amely a közös Cikkszám kapcsoló tulajdonságon alapul. A RENDELÉSTÉTEL állományból a CIKK-be átlépve kikeressük a cikk egységárát és rögtön megállapítjuk az adott rendeléstétel értékét. Ismereteinket azonnal a kimenetbe visszük és csak azután megyünk a következő rendeléstételre.

Tehát az állománykezeléssel szemben, amely extenzionális, az állomány kiterjedése szerint - „vertikálisan" - halad, az ***adatbáziskezelés*** intenzionális. A közös tartalmú tulajdonság mentén „horizontális" - különböző állományok közötti - irányt vesz.

Az „ál"-adatbáziskezelők használatánál ezt az állományok közötti kimutatást külön kell programozni. Az adott rendeléstételhez érve nekünk kell megállapítanunk a Cikkszám értékét, majd azt felhasználva mi programozzuk a megfelelő cikk kikeresését (miután magunk nyitottuk meg a CIKK állományt). Ezért ebben az esetben valójában állománykezelésről van szó, amely procedurális jellegű (vö. dBASE). Ezzel szemben a valódi adatbáziskezelőnek csak annyit kell mondani, hogy keresse ki az aktuális rendeléstétel fölérendeltjét a CIKK - RENDELÉSTÉTEL kapcsolat mentén. A Cikkszám értékét nem a program, hanem az adatbáziskezelő határozza meg. Az nyitja meg a CIKK állományt és az keresi ki automatikusan a megfelelő cikket. Tehát az ilyen kezelés deklaratív, az előre meghatározott tényezőn - a kapcsolaton - alapul.

Mindenesetre az „ál"-adatbáziskezelésre is jellemző az, hogy az állományon belüli kezelést állományok közötti kezelési műveletek szakítják meg.

D 8/4 **Az egyedtípusok közötti, a közös tulajdonságon alapuló (automatikus) átmenetet navigálásnak nevezzük.**

A „navigálás" kifejezés régi keletű és eredeti értelmében az ún. hálós adatbázisokhoz kötődik. Ha kikeressük az első céljainknak megfelelő rendeléstételt (Rendelésszám=„X") és onnan átváltunk a vonatkozó cikk (Cikkszám=„A") kezelésére, akkor „jelzőbóját" kell elhelyeznünk. Ez mutatja, hogy melyik RENDELÉSTÉTEL sorhoz kell majd visszatérnünk a CIKK megfelelő sorának a kezelése után. Összetettebb, többféle egyedet érintő kereséseknél egyedtípusonként kell kijelölnünk a bójákat. Például akkor, ha nemcsak az „X", hanem az összes rendelés értékét akarjuk kiszámítani. Ekkor az ábrákon nem mutatott RENDELÉS egyedből indulunk el. Keressük az első („X") rendelés első tételét, de megjegyezzük, hogy melyik rendelésnél tartunk, mert majd rá kell térnünk a másodikra („Y"). A RENDELÉSTÉTEL egyedből a cikk felé kilépve elkönyveljük, hogy melyik tételnél vagyunk („XA"), hogy majd a folytatásban átléphessünk a következőre („XB"). E bóják mentén haladva - vagyis navigálva - találjuk meg a visszafelé vezető helyes utat az adatbázis szerkezetében.

Valamikor régen a „bója" az adat fizikai elhelyezési módjának megfelelő fizikai mutatót jelentett. Tehát ha az elhelyezési mód (ld. 4.6 pont) megváltozott, akkor át kellett írni azt a programrészt, amely a bóják mentén az állományok között navigált. Ez a tény sértette a fizikai adatfüggetlenség elvét (ld. 4.7 pont). Ezért a relációs adatkezelőkben kiküszöbölték a navigáció és az azzal szorosan összefüggőnek hitt kapcsolat tényezőjét. Ma már tudjuk, hogy ez az

előrehaladás valójában két lépést jelentett hátra. Lehetetlenné tette a kapcsolattípusokon alapuló logikai szintű deklaratív adatkezelést, amelynek lehetősége már régóta ismert. Kikényszeríti a procedurális programozást, amely ráadásul igen sokszor fizikai függőséggel jár (indexelt/nem-indexelt-e a kapcsoló adat).

8.7 A természetes adatfeldolgozási lánc

Most érkeztünk el az adat, az adatbázis és az információ összefüggésének a megvilágításához. Az adatbázisok előtti korszakban a fejlesztők és felhasználók a számítógépes alkalmazást a ***bemenet*** (input), a ***feldolgozás*** (processing) és a ***kimenet*** (output) hármasaként képzelték el. Ez az IPO-nézet szorosan kapcsolódott azzal a felfogással, amely szerint az alkalmazásokat a kimenetből kiindulva kell megtervezni. Bár ez a szemlélet már réges-rég elavult, még mindig akadnak olyan környezetek, amelyekben az IPO-megközelítés uralkodik.

Az IPO-felfogás azért káros, mert egoista, azaz önérdekű. Egy felhasználó (vagy egy szűk alkalmazói csoport) szempontjából értelmezi a bemenetet és a kimenetet. Teljesen figyelmen kívül hagyja azt a tényt, hogy az adatbázis ***közös*** tulajdon. Az adatbázisra, mint az információs rendszer legfontosabb tényezőjére még maga az IPO betűszó sem utal.

Miért veszélyes ez a szemlélet? Azért, mert természetellenessége miatt felesleges munkákhoz vezet mind a bemeneti, mind a kimeneti oldalon. Ezt az állítást két példával szemléltetjük. Először a bemeneti oldalt vizsgáljuk meg.

Valamikor a hetvenes években építőipari termelésirányítási rendszert akartak készíteni valahol. A „felsőbb szervek" ukáza szerint az adatfeldolgozás gépesítését az ún. durva-programozással - vagyis az építőipari erőforrásoknak a megalkotandó épületekhez való globális hozzárendelésével - kellett volna kezdeni. A „volna" mutatja, hogy - szerencsére - ebből a projektből sem lett semmi... Ám mi történt volna, ha megvalósul?

Egy négyzetméternyi X típusú falhoz mondjuk Y darab falazóanyag szükséges. Az ilyesfajta ismeretet normának szoktuk nevezni. Ha egy épületben Z négyzetméter fal van, akkor ahhoz Z*Y darab X típusú anyag szükséges. Vagyis a célmértéket (Z) meg kell szoroznunk a norma mértékegységével (Y). Ez evidens. Viszont az adott példa esetében a „felsőbb szervek" nem rendelkeztek sem a célmérték, sem a norma adatbázisban tárolt adataival. Mi következik mindebből a ***bemenetre*** nézve?

A felhasználónak minden egyes esetben külön bemenetben kellene megadnia a célmértéket. Vagyis a falazóanyag-szükséglet, a habarcs-mennyiség, a munkaóra stb. megállapításához külön-külön be kellene ütnie a „Z" számot. Ez nyilván felesleges munka és igen nagy a hiba esélye. Az adatrögzítő Q-t fog ütni Z helyett. Ugyanez a megállapítás a normára is vonatkozik. A Z és a W célmérték esetén külön-külön elő kell venni a kézi normakönyvet és abból kilesni, hogy Y a norma. Az pedig teljesen világos, hogy Szabó a normakönyv A, Kovács pedig annak B változatát fogja használni.

Az eset a 70-es években történt. Ám nehogy azt higgyük, hogy ma más a helyzet. Az önfejű vezetők nem ismerik az információs rendszerek lényegét. Akarnok módon az általuk megálmodott részrendszerrel kezdik a „gépesítést". Ezzel iszonyatos többletmunkára kényszerítik saját beosztottjaikat és súlyosan veszélyeztetik az ismeretek konzisztenciáját.

Az első tanulság az, hogy a természetes adatfeldolgozási láncot nem szabad ***elölről*** megtörni. Ha a norma és a célmérték természetesen, lényegüknél fogva megelőzik a végső mennyiséget, mert annak alapadatai, akkor a „gépesítést" csakis ezen adatok számítógépes kezelésével szabadna kezdeni.

A ***kimeneti*** példa szintén az akkori korból származik. A számítógépet is gyártó (!) cégben a „termelésirányítási” program ontotta a tonnányi tablókat. Például május 2-án közölte, hogy április 3-án elromlott az meg az a gép. Persze április 3-án ugyanezt nem tette meg. Természetes, hogy erre a megkésett „információra” már senki sem volt kíváncsi. Ráadásul pontosan ugyanazt a tablótömeget kapta a mérnök, a könyvelő, a műszakvezető, a karbantartó, a raktáros stb.

Az ilyen tablókban mindenki találhat magának hasznos ismeretet. Például a karbantartó kiböngészheti, hogy mindig az X-típusú gép romlik el és többnyire olyan módon, hogy ... Csakhogy a mindennapi élet hajtásában kinek van ideje tablókat lapozgatni? Így hát - idézet - „A tablókat a sarokba toltuk, amíg csak el nem vitték a kukába.”.

Ma már szupermodern adatbáziskezelőket használunk. Ennek ellenére nap mint nap találkozunk a „tabló-szindrómával”. A legtipikusabb példa az ***egyeztetés***. A tervező rossz adatbázisszerkezetet készít, amely emiatt képtelen az önellenőrzésre. Emlékezzünk csak arra, hogy az adatmodell szerkezete egyben korlát is, tehát alkalmas bizonyos fokú automatikus ellenőrzésre. A másik hiba az, hogy nem használják ki a mai kezelők validálási képességeit. A hibás adatok tömegét engedik be az adatbázisba. Emiatt nélkülözhetetlen a bevitt adatok „egyeztetése”. Sikerült-e helyesen bevinni az adatot vagy sem?

Bár a jó adatbázisszerkezettel, a megfelelően alkalmazott validálási rutinokkal és főleg a beviteli fegyelem szigorításával a mai hibás adatbevitel mértékét a tizedére lehetne csökkenteni, valamilyen egyeztetésre mindig szükség lesz. Csak az a baj, hogy nem Irénkének és Marikának adjuk az őket illető pártucatnyi tételt egyeztetésre, hanem több száz oldalnyi tablót zúdítunk egy-egy szervezeti egységhez. A szelektivitás teljes hiánya jellemzi a mai gondolkodásmódot.

Az egyeztetés csak példa volt arra, hogy a természetes adatfeldolgozási láncot ***hátulról*** is megtörjük. Nem testreszabott ismereteket adunk a felhasználónak. Sajnos nagyon is jellemző, hogy a tervező nem Marikát és Irénkét látja a szeme előtt, amikor valamit kifundál, hanem egy jellegtelen szürke tömeg számára készít leporellókat. Megfordítva: Irénke és Marika már annyira megszokta a tablólapozást, hogy szentül hiszi: ez a számítógépes adatfeldolgozás lényege.

A vezetőknek, fejlesztőknek és felhasználóknak végre tudomásul kellene venniük, hogy

az adatok feldolgozásának van egy optimális lánca.

Ez az adatok természetes összefüggésein alapul. Az ismeretek logikailag és időben elrendezettek. Azaz az egyik ismeret a másiknak az alapja. A normát (Y) már tudni kell, amikor a célmérték (Z) szerinti téglaszükségletet ki akarjuk számolni. Ezt a vastörvényt nem lehet büntetlenül megszegni. A számítógéppel foglalkozók gyakran elfeledkeznek arról, hogy az adatbázis mindig ***alapadatbázist*** jelent. A származtatott adatok tárolásának a fentebb tárgyalt eseteit kivéve az adatbázisban csak alapadatokat tárolunk, viszont azokat tárolnunk is kell.

Az adatfeldolgozás természetes fonalának a kezdetét viszonylag könnyű felismerni. A végét sokkal nehezebb megtalálni. Ahhoz alaposan ismerni kellene a felhasználót. Mi több, a felhasználónak is összhangban kellene lennie saját magával. Tudnia kellene, hogy melyek a valós feladatai és azok teljesítéséhez milyen információkra van szükség. A bajok ott kezdődnek, hogy maga a felhasználó nincs tisztában a saját funkciójával, a számára fontos ismeretekkel és azok összefüggéseivel. Mivel előre nem is tudja, hogy mit keres, ilyen meg olyan vetületű tablókat kér. Majd csak rábukkan valami fontosra. Amennyire a vezető és a tervező a felelős az optimális feldolgozási lánc elölről való megtöréséért, ugyanannyira a felhasználó a ludas abban, hogy a számítógépet nem lehet okosan megdolgoztatni és a valódi végéig vinni a feldolgozást.

8.8 Technikai adatok

Mint már ismeretes, az adatbázisnak három szintje van: fogalmi, logikai és fizikai. Az adatbázisban az általunk fontosnak tartott jelenségek lényeges ismereteit és azok viszonyait akarjuk rögzíteni (fogalmi szint). Az alkalmazási/technikai környezet miatt távlati igényeinkkel megalkuszunk és tartalmilag ilyen-olyan módon elrendezett korlátos adatbázist hozunk létre (logikai szint). Elkönyveljük, hogy a számunkra fontos ismerettartalom mindenféle számunkra rejtélyes adattárolási és -ábrázolási módokon valósul meg (fizikai szint).

Mindeddig csak azokról az adatokról szóltunk, amelyek az alkalmazási környezet számára lényeges ismereteket tükrözik, amelyek ***a felhasználó számára fontosak***. Magában a fizikai adatbázisban lesznek olyan belső technikai adatok (indexek, mutatók, belső kulcsok), amelyeket a felhasználó nem lát, de a számára lényeges ismeretek kezelése szempontjából végül is az ő számára lényegesek.

Az alcímben jelzett „technikai adat" kifejezés nem ezekre a titkos, belső, szükséges adatokra vonatkozik. A kívülről is látható, másodlagos célú adatokról van szó. Mindezt persze leginkább egy példán keresztül lehet megérteni.

A végső-felhasználó számára fontos - elsődleges - adat a Szerződéskötés-dátuma. Ő adja meg ennek tartalmát, ő kezeli az adatot, amit akkor és úgy lát, ahogy akar. Viszont itt van a Szerződésrögzítés-dátuma. Ez azt mutatja, hogy mikor vitték be sikeresen a szerződés adatait a számítógépre. Ha tetszik, ha nem, be kell látnunk, hogy ehhez a végső-felhasználónak elvileg semmi köze sincs. A szerződés adott naptól fogva él függetlenül attól, hogy a „gépesek" annak adataival miként bűvészkednek. Ezért a Szerződésrögzítés-dátuma másodlagos adat. A felhasználó által nem kezelhető, sőt esetleg nem is látható. Nos, ezeket a másodlagos adatokat nevezzük technikaiaknak.

A mai adatbázistervezők kedvelt szokása, hogy az adatbázis állományait teletűzködik olyan másodlagos adatokkal, amelyek csakis ***a fejlesztő számára fontosak.*** Bevittem, töröltem, elküldtem a csekket, értesítettem az ügyintézőt, átadtam stb. Amint látjuk, ezek az ismeretek többnyire arra szolgálnak, hogy a fejlesztő és a feldolgozó igazolja magát a felhasználóval folytatott - sajnos egyre gyakoribb - vitákban. Bár szomorúan, de be kell látnunk, hogy az ilyen másodlagos ismeretek tárolására is szükség van. A kérdés az, hogy hol van a technikai adatok megfelelő helye?

A rossz tervező a „rekordképekben" - az elsődlegesekkel teljesen összekeverten - tünteti fel a saját másodlagos technikai adatait is. (Mentségére szolgál, hogy a mai adatkezelők egyáltalán nem támogatják az elsődleges/másodlagos minősítést. Számukra a Szerződéskötés-dátuma és a Szerződésrögzítés-dátuma ugyanolyan lényeg.)

Ez a pont a fejlesztők számára két üzenetet akar közvetíteni. Egyrészt azt, hogy ne éljenek vissza helyzetükkel. Azzal, hogy ők közelebb állnak az adatbázishoz, mint a felhasználók. Erkölcstelen dolog a másodlagos adatoknak az elsődlegesekkel együtt történő tárolása, mert lelassítja a felhasználói adatkezelést. Másrészt azt, hogy a közös tárolás amúgy is csacsiság. A technikai, másodlagos adatok általában többszörös értékűek. (Ebben az időszakban is küldtem csekket, az előzőben is stb.) Ezért az a célszerű megoldás, ha a technikai adatokat külön technikai egyedtípusokban tárolják.

8.9 Az adatbázis alapvető titka

A bemenet-feldolgozás-kimenet egyszerű hármasában való gondolkodás ideje rég lejárt. Persze még írhatunk programot, amelyek a közvetlen bemenetből némi feldolgozási műveletsor után azonnal produkálja a kimenetet. Azonban ma már - az adatbázis korszakában - a „korai bemenet" és a „késleltetett kimenet" kellene, hogy jellemző legyen.

Korai bemenet: Amint tudomást szerzünk egy tényről, annak ismereteit azonnal rögzítjük az adatbázisban. Akkor is, ha nem tudjuk, hogy ki és mikor fogja azt felhasználni. Tehát az alapadatot nem akkor visszük a számítógépre, amikor abból majd konkrét kimenetre lesz szükségünk. El kell könyvelnünk, hogy minden adatbázis telis-tele van sohasem értékesített ismeretekkel. Ez a felfogás az alapja annak, hogy amikor komoly információkra van szükségünk, nem kell a bevitellel bajlódni. Minden alapadat előre „tálalva van".

Késleltetett kimenet: Mivel az adatbázisban minden fontos ismeret rendelkezésre áll, nincs szükség származtatott adatok bevitelére, előfeldolgozásokra, kapkodásokra. Majd (de csakis akkor), ha a vonatkozó ismeret fontossá válik, azt előkeressük az adatbázisból. Így nincs szükség elhamarkodottan újravariált bemenetekre és előre át nem látott feldolgozásokra. Nem a bemenet, nem a feldolgozás, hanem a szilárdan meghatározott adatbázisszerkezet a jó kimenetek alapja.

A fentiek után megfogalmazhatjuk az adatbázis egyik legfontosabb törvényét:

Az információs rendszereknek nem az adatfeldolgozás, hanem a gondosan meghatározott adatbázisszerkezeten alapuló adatkezelés a motorja.

Azt az ismeretet, amit egyszer - helyesen - az adatbázisban rögzítettünk, bármikor és bármilyen variációban elő tudjuk keresni és össze tudjuk illeszteni a többi adattal. Ha viszont az adatbázisunk tartalma rossz vagy az ismeretek a helytelen szerkezet miatt nem hozhatók összhangba, akkor bármennyire is kínlódunk, nem teszünk szert információra.

A hetvenes években a jelszó a „gépesítés" volt. A számítógép majd megoldja az információ igényeinket. A számítógéppel előre „bedrótozott" adatfeldolgozási folyamatokat végeztettek. Így a bemenet, a kimenet, az összefüggések legkisebb változása esetén azonnali újradrótozásra volt szükség. Azért, mert az akkori fejlesztők még nem láthatták, hogy a feldolgozás roppant változékony, az adatbázis viszont stabil. Végeredményben a számítógépeket ***osztó-szorzó, adatfeldolgozó masinákként*** képzelték el és használták. Sajnos ez a szemlélet még sok mai fejlesztőt és felhasználót is megfertőz.

Pedig a mai igazi jelszó az „alapismeret-tárolás". A számítógépet arra illik használni, hogy jól meghatározott szerkezetben - egyértelműen, egyszeresen és teljesen - tárolja alapvető ismereteinket. Ezek egy jó adatkezelő segítségével bármikor elővehetők, megnézhetők, összedolgozhatók. A feldolgozás szinte mellékes; a megfelelő ismeretszerkesztés a lényeg. Ezért ma a számítógépeket az ***ismereteket adatbázisszerű módon tároló, elsősorban adatkezelési feladatokat ellátó eszközökként*** kellene elképzelnünk.

Tegyük fel, hogy a rendeléstétel vagy a cikk adatait helytelenül visszük be, mert nem tudjuk érvényesíteni az adatkezelési szinten meghatározható korlátokat. Tegyük fel, hogy a Rendelt-mennyiség és az Egységár tartalmát helyesen tároljuk, csak éppen a kapcsoló tulajdonság bajai miatt a rendeléstétel nem kapcsolható a megfelelő cikkhez. Végül tegyük fel, hogy alapvetően rosszul határoztuk meg a rendelés és a cikk viszonyát. Mi hasznunk van abból, ha elvileg tudjuk, hogy a Tételérték a Rendelt-mennyiség és az Egységár szorzata? A szorzás legfeljebb az általános iskolás alsós gyerek gondja. A megfelelő adatbázisszerkezet kialakítása, a korlátok helyes meghatározása és érvényesítése - az adatmodellezés - a valóban felnőtt informatikus feladata. Mert az adatkezelés az adatmodellben rögzített szerkezeten és korlátokon alapul.

Ellenőrző kérdések - 8

8/01 Az alábbi állítások közül melyik igaz (I) és melyik hamis (H)? Mindegyik állítás ezzel a kitétellel kezdődik: „Vannak X, Y és Z típusú szerződéseink”.

- A háromféle szerződésnek sok közös adatfélesége van. Ha három állományt kreálunk, az nem-tudatos particionálást jelent.
- Az X típusú szerződéseket ötven különböző fiókban kezelik. Ez az adatbázis fizikai particionálását jelenti.
- A háromféle szerződés közös adatairól tudunk. Fogalmi szinten azokat együtt határozzuk meg. Majd az egyetlen SZERZŐDÉS egyedet a szerződésfajta speciális tulajdonságainak megfelelően három állományban valósítjuk meg. Ez az adatbázis logikai particionálását jelenti.

8/02 Az alábbiak közül melyik adatkezelési (K) és melyik adatfeldolgozási (F) művelet?

- Kikeressük a Lada típusú kocsik tulajdonosait.
- Módosítjuk a cikk árát 50-ről 60-ra.
- Megállapítjuk a számla végösszegét a számlatételek tárolt értéke alapján.
- A Vevőkód tárolási sorrend helyett Vevőnév sorrendű listát készítünk.

8/03 Az alábbi kitételek közül melyik igaz (I) és melyik hamis (H)? A bevételi számlák nyilvántartásáról van szó úgy, hogy a számlák többtételesek.

- A Számlatétel-érték a Mennyiség és Ár alapján származtatott adat.
- Ezt az adatot mindig a SZÁMLATÉTEL egyedben kell tárolni.
- Ez az adat mindig a SZÁMLATÉTEL egyedet jellemzi.
- Ezt az adatot sohasem kell tárolni.
- A Számlaérték adatot tároljuk, ha a főnök azt gyorsan akarja visszakeresni.
- Az Össztartozás (Σ Számlaérték) adatot tároljuk, ha a főnök azt gyorsan meg akarja ismerni.

8/04 Hol tárolná Ön a szerződésekre vonatkozó Változás-oka és a Változás-dátuma adatokat, ha a változás technikai jellegű (bevitel, törlés stb.)? Számmal adja meg a helyes választ.

- Sehol. A változás tényét papíron kell rögzíteni.
- A SZERZŐDÉS egyedben, hiszen a változás arra vonatkozik.
- Mivel több változás is vonatkozhat egy szerződésre és ez csak technikai adat, terveznék egy külön SZERZŐDÉSVÁLTOZÁS technikai egyedet.

8/05 Mondja el szavakban saját magának, hogy miért helytelen, ha a Főnök a gépesítést a könyvelés „gépreültetésével” kezdi.

8/06 Marcsa és Julcsa ugyanazon a részlegen dolgozik. Marcsa az X, Julcsa az Y körzetért felelős. Feltételezhetően hibás ismereteket visznek be a számítógépre. Ön mit tenne? Adja meg a teendők sorszámát az alábbiak szerint. Ha valamit nem tenne, akkor 0 legyen a szám.

- Átnézném a beviteli programok validálási rutinjait.
- Ellenőriztetném „szemmelveréssel” Marcsa és Julcsa adatbevitelét.
- Felülvizsgálnám az adatbázis szerkezetét és korlátait.
- A részlegnek adnék egy egyeztető tablót.
- Marcsának és Julcsának külön-külön adnék egyeztető listát.

9. A METAADATBÁZIS

9.1 Az adatbázisok két szemléleti síkja

Az adatbázis az ismeretek tudatosan szervezett együttese. A 4. fejezetben rámutattunk, hogy az adatbázisokat három - fogalmi, logikai és fizikai - ***szemléleti szinten*** kell meghatározni. Ez a záloga az ún. vertikális adatfüggetlenségnek (ld. 4.8 pont). Annak, hogy az alsóbb szinteken bekövetkező változások következtében ne kelljen teljesen újratervezni az adatbázis szerkezetét. Az 5. fejezetben az adatbázisok - globális és parciális - ***szemléleti vetületeivel*** foglalkoztunk. Az egyéni felhasználói nézetek definiálásának a képessége ad alapot az ún. horizontális vagy logikai adatfüggetlenségnek (ld. D 5/3). Az adatbázis szerkezetének a változásai nem hatnak ki az alkalmazásra, amennyiben azok nem érintik az adott nézetbe tartozó tényezőket.

A változásokra nem érzékeny adatbázis kiépítésének a jól meghatározott fogalmi adatmodell az alapvető feltétele. Ezért két fejezetet szenteltünk az adatmodell, mint rendszer tényezőinek és azok összefüggéseinek az ismertetésére (ld. 6. és 7. fejezet). Ennek kapcsán talán már rádöbbentünk arra, hogy a szinteken és vetületeken kívül az adatbázisoknak egyéb vizsgálati aspektusai is vannak. Többször használtuk azt a kitételt, hogy „az adatbázis az adatmodell szerint szervezett". Ez azt mutatja, hogy nem egy lényeggel, hanem két - egymásnak megfeleltetett - rendszerrel kell foglalkoznunk.

A felhasználó számára lényeges ismeretek szervezett együttesét ***alkalmazási adatbázisnak*** nevezzük. Ez tárolja a fontos valós jelenségek - például a kocsik, tulajdonosok stb. - tényleges adatait. Az adatbázis létrehozásához meg kell határoznunk annak tervét. Ha jobban meggondoljuk, a terv is ismeretek szervezett együttese, tehát valójában - adatbázis. Csakhogy ezek az ismeretek nem közvetlenül a felhasználó, hanem a fejlesztő részére lényegesek. Ezért teljesen logikus, ha az adatbázistervet ***fejlesztési adatbázisnak*** tekintjük. Tehát az adatbázisnak mindig két szemléleti síkja van: alkalmazási és fejlesztési. A kettő teljes harmóniája a sikeres adatbázisépítés és -működtetés záloga.

Ebben a fejezetben az adatbázisok két szemléleti síkjának az összefüggéseit, ezen túlmenően pedig elsősorban a fejlesztési adatbázis sajátosságait és kezelését fogjuk ismertetni.

„A vargának a cipője ...". Elgondolkodtató, hogy miközben a szakemberek mások számára számítógépes adatbázisokat építenek, a saját maguk számára fontos ismereteket „papír-adatbázisban" tartják és kezelik. Hogyan lehet képes a fejlesztő jól átgondolt alkalmazási adatbázist létrehozni, ha a saját dolgait is képtelen rendbentartani? Mert az már bebizonyosodott, hogy a kézi adatbázis-dokumentációk minősége - finoman szólva - kritikán aluli. Persze a számítógépes adatbázis-dokumentáció sem vezet sokra, hiába vannak csodálatos eszközeink, ha nem ismerjük a fejlesztési adatbázis természetét.

9.2 A metaadatbázis lényege

Az adatbázis az adatmodellnek megfelelően épül fel. Az adatmodell egyed-, tulajdonság- és kapcsolattípusok szervezett együttese. Ezért az adatbázis tervében valamilyen módon le kell írni ezeket a tényezőket, azok viszonyait és a tényezők illetve viszonyok korlátait. Ebben a pontban ennek a leírásnak az elvi alapjairól lesz szó. Mondanivalónkat a következő példamondatokkal szemléltetjük:

9.1 példa

„Rózsa kocsija Lada típusú."
„A Lada típusú kocsik ötszemélyesek."
„A Kocsitípus tulajdonság karakteres adat."
„A Kocsitípus tulajdonság a KOCSI egyedet jellemzi."

Az első két mondat a „valós" jelenségekről közöl ismereteket. A másik kettő viszont az előző adatokkal kapcsolatos ismereteket ad át. Az első kitétel elmondja, hogy milyen típusú Rózsa kocsija. A harmadik pedig leszögezi, hogy a Kocsitípus - az első mondat ismerettartalma - karakteres adattípusú.

D 9/1 **Az ismeretekre vonatkozó ismereteket metaadatoknak hívjuk.**

A definíció magyarázata előtt ki kell térnünk a fenti „valós" jelzőre. Régebben a „valós világ" és a „valós rendszer" (angolul: real-world és real-system) megjelöléssel illették az alkalmazás lényegét, hogy azt elválasszák az információs rendszertől. A jelző nem szerencsés, mert hiszen nemcsak a kocsi „valós", hanem az őt tükröző egyedtípus is az. Nemcsak a kocsi tekinthető ismeretekkel leírandó jelenségnek, hanem a KOCSI egyedtípus is. Az ellentmondás feloldására bevezették a nagyon filozofikus ***„az, amikről szó van"*** (angolul: university of discourse, amit így rövidítenek - UoD) megjelölést. Ez jobban tükrözi a lényeget. Mert „szó lehet" a kocsikról, tulajdonosaikról stb. éppen úgy, mint az egyed-, tulajdonság- és kapcsolattípusokról. Mindez nézőpont - szemléleti sík - kérdése.

Most térjünk vissza a metaadathoz! Emlékezzünk arra, hogy az adatnak négy dimenziója van (ld. 2.2 pont). Az alkalmazási ismereteknél az egyedtípus (KOCSI) adott tulajdonság-típusának (Kocsitípus) a meghatározott egyedelőfordulásra (Rózsa) felvett értéke („Lada") jelenti ezt a négy dimenziót. A metaadat is adat - csak más a szemléleti síkja. Ezért a fejlesztési adatoknál is négy dimenzióban kell gondolkodnunk.

Szokásunkkal szemben most nem az egyed, hanem a tulajdonság oldaláról fogunk kiindulni. A 9.1 példa harmadik mondatában a Kocsitípus lényeg egyik jellemzőjét adtuk meg. Korábbi definíciónk szerint (ld. D 3/3) azt a valamit, amivel a jelenséget leírjuk, tulajdonságnak hívjuk. Tehát a „karakteres adat" a Kocsitípus tulajdonsága. A tulajdonságokat tipizáljuk, vagyis nevet adunk nekik (Adattípus) azért, hogy annak majd kijelölhessük az értékeit („karakteres"). Hiszen nemcsak a Kocsitípusnak, hanem minden tulajdonságnak van adattípusa.

Mivel jelen esetben a tulajdonság ismeretről szóló ismeretet hordoz, az Adattípus-t joggal nevezzük ***meta-tulajdonságtípusnak***, a „karakteres"-t pedig metatulajdonság-értéknek. Könnyű belátni, hogy az Adattípus tulajdonságnak számos egyéb metatulajdonsága lehet. Ilyen az adathossz, a validálás stb. Már kicsit nehezebb felismerni, hogy az egyedtípusok is leírhatók metatulajdonságokkal. Például a KOCSI egyedtípus jellemezhető a Darabszámmal, amely megadja,

hogy hány járműről kell nyilvántartást vezetni. A másik metatulajdonság pedig csaknem kiüti a szemünket: a „KOCSI” nem más, mint az Egyedtípus-név egyik értéke.

Az alkalmazási adatbázisban különböző jelenségek (kocsi, tulajdonos, kocsitípus stb.) adatait kezeljük. Korábbi definíciónk szerint (D 3/1) a ismeretekkel leírandó jelenségeket egyedeknek hívtuk. Ezeket egyedtípusokba (pl. KOCSI) osztályoztuk, hogy a konkrét jelenségeket majd azok előfordulásaiként (Rózsa kocsija) reprezentálhassuk. Így van ez a fejlesztési adatbázis esetében is. A metaismeretekkel leírandó jelenségeket (egyed, tulajdonság, kapcsolat stb.) ***meta-egyedtípusokba*** (pl. EGYEDTÍPUS) osztályozzuk, hogy a konkrétumokat majd azok meta-előfordulásaiként (KOCSI) adjuk meg.

Íme, előttünk áll a metaadat négy dimenziója. A meta-egyedtípus (TULAJDONSÁG-TÍPUS) adott meta-tulajdonságtípusának (Adattípus) az adott előfordulásra (Kocsitípus) felvett értéke („karakteres”) adja a négy dimenziót.

A négy tényező az adatbázisnak szükséges, de nem elégséges feltétele. Az alkalmazási adatbázis egyedtípusai kapcsolattípusokban állnak, különben nem is beszélhetnénk adatbázisról. Ehhez hasonlóan a fejlesztési adatbázis meta-egyedtípusai között ***meta-viszonytípusok*** léteznek. Jól mutatja ezt a 9.1 példa negyedik mondata. A Kocsitípus tulajdonság (metaegyed) a KOCSI egyedet (másik metaegyed) jellemzi, ezért a kettő között metaviszony áll fenn. Általában a tulajdonságok egyedekhez kötődnek, tehát a metaviszony tipizálható. Az EGYEDTÍPUS - TULAJDONSÁGTÍPUS meta-viszonytípusnak egyik előfordulása a KOCSI - Kocsitípus konkrét metaviszony.

Azok, akik eddig „rekordképben” gondolkodtak, nehezen értik meg, hogy miért van szükség három tényezőre. Példánkban a KOCSI egyedtípusra, a Kocsitípus tulajdonságtípusra és ráadásul még a KOCSI - Kocsitípus viszonytípusra is. Pedig a lényeg egyszerű: az egyed- és a tulajdonságtípus nem azonos, hanem csak kapcsolatban álló jelenségek. Az egyednek vannak tulajdonságai - és ez a kifejezés egyértelműen utal az ún. birtoklási viszonyra (ld. 6.5 pont). Sőt, a fentiekben nem véletlenül használtuk a „viszony” szót a „kapcsolat” helyett. Az egyed- és tulajdonságtípusok M:N fokú viszonyban állnak egymással. Ezért összefüggéseiket külön meta-egyedtípussal kell tükrözni (ld. 6.7 pont).

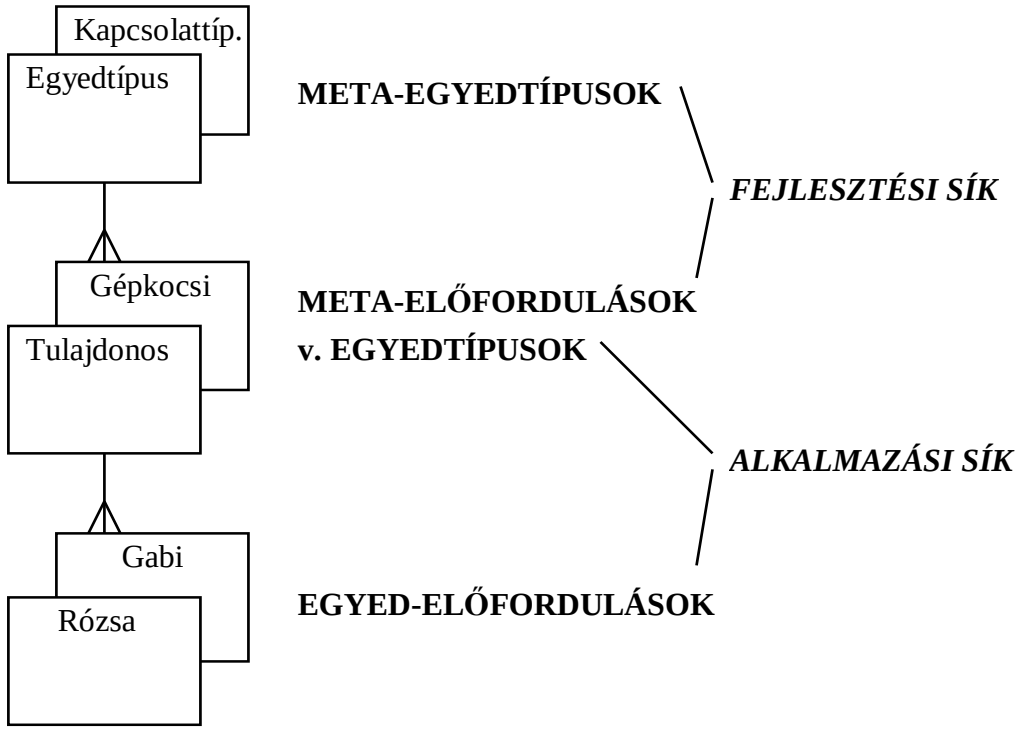
Most már összeáll a kép a metaismeretek szervezéséről. A fejlesztési adatbázis pontosan úgy épül fel, mint az alkalmazási, csak más ismereti síkot reprezentál. Ezért kijelenthetjük:

D 9/2 **A metaadatbázis a meta-egyed-, tulajdonság- és kapcsolatelőfordulások szervezett együttese.**

Eddigi ismereteink szerint a metaadatbázis nem más, mint az alkalmazási adatbázis modellje. Mert hiszen a meta-egyedelőfordulások megegyeznek az alkalmazási adatbázis egyedtípusaival. Amiként az alkalmazási adatbázis általános felépítését az adatmodell írja le, úgy a fejlesztési adatbázis általános struktúráját a metamodell, a „modell modellje” rögzíti.

D 9/3 **A metaadatmodell a meta-egyed-, tulajdonság- és kapcsolattípusok szervezett együttese.**

A fejlesztési és az alkalmazási adatbázis viszonyát a 9.1 ábra szemlélteti. Ezen jól látszik a két egymással átfedő szemléleti sík:



9.1 ábra: Az adatbázisok két szemléleti síkja

„Rózsa" a TULAJDONOS egyedtípus egyik előfordulása. Viszont a TULAJDONOS az EGYEDTÍPUS meta-egyedtípus egyik meta-előfordulása. Vagyis az alkalmazási szintű típus mindig megfelel a fejlesztési szintű előfordulásnak. Amint látjuk, a kétféle adatbázis logikája teljesen analóg, és ebből a tényből a fejezet során több következtetést fogunk levonni.

A metamodell több, mint az alkalmazási adatbázis modelljének a modellje. A meta-adatbázis az ismeretekről szóló ismeretek tárháza. Ezért nem csupán az alkalmazási adatbázis szerkezetét rögzíti, hanem az adatbázisra vonatkozó minden tudnivalót felölel. Tárolja a fejlesztéssel és a használattal kapcsolatos adatokat is. Bennünket ugyan - ebben a könyvben - elsősorban maga az adatbázis érdekel, egy rövid ismertető erejéig érdemes kitérnünk a meta-adatbázisok fejlődésére.

9.3 Történeti kitekintés: Az adatszótár

A metaadatbázis fogalmát mindeddig bizonyára kevesen ismerték. Sokkal többen találkoztak az „adatszótár" kifejezéssel. Az előző pont olvastán néhányan talán így szóltak magukban: „Hm, adatszótárról beszél, de miért emleget fejlesztési adatbázist vagy metaadatbázist?" Azért, mert nem ugyanazokról a dolgokról van szó. A korszerű adatkezelés megértéséhez nem árt ismerni a két lényeg különbségét, vagyis áttekinteni a metaismeretek kezelésének rövid történetét.

Az első adatszótár-rendszerek már a hetvenes években megjelentek. Nemcsak ***adatszótárnak*** (data dictionary), hanem adatkatalógusnak, -lexikonnak, -készletnek, sőt -kincstárnak is nevezték őket. Ezek a megjelölések nagyjából jól mutatják a kezdeti célt. A szótár az egységes értelmezés

eszköze, amely a címszóhoz kapcsoltan kötetlen szöveges leírásban adja meg az általa kifejezett tartalmat. Például: „A Kocsitípus tulajdonság azt jelenti, hogy ...”.

Az ilyen adatkincstárak - ez a kifejezés fejezi ki leginkább az adat értékét - annyiban tértek el a valódi szótáraktól, hogy a metaismereteket nem szövegszerűen, hanem adatszerűen kezelték (ld. 2. fejezet). Így lehetőség volt arra is, hogy az adat bizonyos metaismereteit (típus, hossz, ábrázolás stb.) beskatulyázzák és valamilyen kódolt formában kezeljék. Az akkori állapotnak megfelelően az adatszótárakon a műveleteket ***állománykezelővel*** hajtották végre (lásd 3.1 pont). Már ismerték a metaegyedtípus lényegét - pl. nemcsak adatokat, hanem csoportokat, rekordokat stb. is képesek voltak leírni. Viszont az ezek közötti keresgélés procedurális úton történt, hiszen nem volt még adatbázis és a metaegyedek között nem lehetett metakapcsolatokat meghatározni.

A hálós adatbázisok elterjedésével az adatszótár kifejezés új értelmet kapott. Mindaddig a szótárrendszerek függetlenek voltak az adatbáziskezelőktől. Csak a tervezésben illetve a tudatos dokumentálásban kaptak szerepet. Mi tagadás, a kimondottan szöveges részektől eltekintve úgy nézett ki az adatszótár, mint egy óriási COBOL Data Divison gyűjtemény. Viszont a hálós adatbázisokban meg kellett adni azok formalizált leírását, az adatbázis-sémát (vö. D 5/5). Nem pusztán csak a globális fogalmi sémáét, hanem - később - a belső és külső sémáét is. Ez a tény két gondolatot ébresztett a szakemberekben.

Az egyik a kettősség kiküszöbölésével kapcsolatos. Ha külön szótárrendszert alkalmaznának, akkor az adatokra vonatkozó ismereteket meg kellene duplázni, hiszen az adatszótár és a séma tartalma nagyrészt átfed. Különben is minek két szoftverrel bíbelődni, amikor ott van maga az adatbáziskezelő? Viszont ugyanakkor ezek a rendszerek nem voltak felkészítve a valóban szótárszerű szövegek kezelésére, hiszen azoknak nincs semmi keresnivalójuk a sémában. Természetesen ezt a gondot könnyen át lehetett hidalni.

A másik gondolat a sémák - akkor szinte évről-évre változó - tartalmával függött össze. A belső sémákban egy idő után meg kellett jelölni az elhelyezés paramétereiként a tárolóeszközt illetve annak egyes sajátosságait is. Ebből önként adódott a lehetőség: némi átalakítás árán általánosan is vissza lehetne keresni az adatok és az eszközök összefüggéseit. Tehát nemcsak az adatokat, hanem az azokat tároló, kezelő eszközöket is le lehetne írni az adatszótárban. Ugyanakkor a fogalmi sémában globálisan, a külső sémákban parciálisan meg kellett adni a felhasználónkénti hozzáférési jogot. Később azt konkrét feldolgozáshoz is lehetett kötni. Ezért készen állt az alap a felhasználók, az adatok és a programok egyes (összefüggő) ismereteinek a visszakeresésére is.

Itt eljutottunk arra a pontra, amikor az „adat”szótár már régen nem csak az adatokra vonatkozó ismereteket tárolta. A szótár alkalmas lett az eszközöknek, a felhasználóknak, a programoknak és ezen tényezők viszonyainak a dokumentálására is. Ettől fogva az információs rendszer mindenfajta tényezőjének a metaismereteit őrző adattárakat ***információs erőforrás szótáraknak*** (angolul: information resource dictionary, IRD) nevezték.

Az ilyen szótárakban már nem egymástól független metaismereteket tároltak. Meghatározták a „metajelenségek” - adatok, eszközök, felhasználók, feldolgozások stb. - összefüggéseit is, vagyis valódi adatbázisokat alakítottak ki. Az adatszótár kezdeti állományszerű kezelését felváltotta az adatbázisszerű kezelés, ami annál is érthetőbb, mert hiszen kéznél volt maga az adatbáziskezelő rendszer. Tekintettel arra, hogy az adatot tároló eszköz, az információt igénylő felhasználó, az ismeretet kezelő eljárás leírása maga is „ismerettel kapcsolatos ismeret”, az így kialakult szótárakat már joggal nevezhetjük meta-adatbázisoknak.

Az adatszótárak fejlődésének eddigi utolsó fejezetéhez érkeztünk. A nyolcvanas években sok olyan fejlesztőrendszer született, amelyekben nemcsak a fejlesztés eredményeit, hanem annak körülményeit is dokumentálni lehetett, sőt: kellett. Például meg lehetett adni a projekt célját, erőforrásait, ütemezését stb. Mód volt annak leírására, hogy ki, mikor, hogyan és miért úgy tervezte meg az adatbázis valamelyik részletét. Csakhogy így a fejlesztő- és az adatkezelő-rendszer szótára nagymértékben átfedett. A redundancia elkerülésére a fejlesztőrendszert az adatkezelőhöz

igazították és szótárának tartalmát beleolvasztották az adatkezelőébe. Ezzel megteremtődött a fejlesztési folyamatot és annak eredményeit együttesen dokumentáló szótár.

Már csak egy kérdésre kell választ adnunk. Az előző pontban a fejlesztési- és a meta-adatbázis fogalmát szinonimaként használtuk. Valóban azonos lényegekről van szó? Alapjában véve igen. A „fejlesztési" jelző azt emeli ki, hogy „nem alkalmazói", nem a végső-felhasználó által kezelt ismeretekről van szó. Vagyis a relativitást hangsúlyozza. Ezzel szemben a „meta" az ismeretek abszolút természetére, közvetett voltára utal. Ezért talán egy picit jobban fejezi ki a lényeget.

9.4 A metarendszerek természete

Eddig nem volt szó az adatbázis-rendszerek kétféle jellegéről. Vannak olyan alkalmazások, amelyek egy-egy előre meghatározott célfeladat kiszolgálására születtek. Ezek esetében az adatmodell tényezői és az azokat kezelő programok egymáshoz vannak cementezve. Az egyed-, tulajdonság- és kapcsolattípusok eleve adottak; azokat semmilyen módon sem lehet változtatni. Minden módosítás esetén magának az adatkezelő rendszernek az átírása szükséges. Az ilyen rendszereket ***testreszabottaknak*** (angolul: tailored) nevezzük. A feladatra való behangolás miatt az effajta alkalmazások nagyon hatékonyak, de ugyanakkor roppant rugalmatlanok. Éppen ezért - a speciális irányítási folyamatoktól eltekintve - manapság már kihalóban vannak.

A hétköznapi életben elterjedt alkalmazási adatbázisok tartalma tetszőleges. Ezen azt kell érteni, hogy a végső-felhasználó maga dönti el, hogy milyen jelenségek (kocsik, könyvek, személyek stb.) milyen ismereteit kívánja tárolni és kezelni az adatbázisban. Az adatmodellt tetszőlegesen változtathatja anélkül, hogy az kihatna magára az adatkezelő rendszerre. Az ilyen rendszereket ***általánosítottaknak*** (angolul: generalized) hívjuk. Az általános kezelőrutinok miatt ezek sohasem lesznek olyan hatékonyak, mint a testreszabott alkalmazások. Viszont roppant rugalmasak - és a mindennapi felhasználásban általában ez a fontosabb kritérium.

A kétféle jellegre nem véletlenül utaltunk gondolatmenetünknek ezen a pontján. A fejlesztői adatbázisok fejlődése elmaradt az alkalmazási adatbázisokétól. Talán azért, mert a szakemberek a végső-felhasználóknál is konokabban ragaszkodnak a megszokott megoldásokhoz. Ennek következtében a fejlesztői adatbázisok terén még ma is nagyon elterjedtek a testreszabott rendszerek. A fejlesztő számára adottak a metaadatbázis többnyire igen korlátos, a célszoftver képességeihez - és nem az igényekhez - szabott tényezői és azok összefüggései. Nincs lehetőség a meta-adatmodell változtatására.

Ezzel szemben vannak már - különböző fokon - generalizált adatszótár-rendszerek is. Ezek szinte kivétel nélkül kettős természetűek. A testreszabott szótáraknál előre kötött a metamodell felépítése, vagyis szigorúan adottak a meta-egyed-, tulajdonság- és kapcsolat-típusok. A teljesen általánosított rendszer csak „héj"; a fejlesztő maga határozza meg már a metamodellt is. Tehát eldöntheti például, hogy egyedeket, relációkat, rekordokat akar-e leírni metaegyedekként és milyen metatulajdonságokkal. A mai szótárrendszerek többsége nem ilyen. A metamodell előre adott, de a fejlesztőnek bővítésekre van lehetősége. Mit jelent ez?

A szótárrendszerek általában nem engedik meg az előre definiált metatényezők törlését vagy módosítását. Csak azt teszik lehetővé - különböző mértékben -, hogy az eredeti korlátokat a fejlesztő saját még szigorúbb szabványokkal tovább finomítsa. Viszont lehetőség van a modell bővítésére, vagyis új metatényezők meghatározására. Az már más kérdés, hogy mennyire támogatja a rendszer az ilyen speciális és a standard elemek szerves összhangját. (Általában jelenleg még semennyire sem.)

Végeredményben a szótárrendszerek ma még meglehetősen merevek. A célrendszer - a majd használt adatbáziskezelő - terminológiája és korlátai által behatároltak. Ezért felmerül a kérdés, hogy a konkrét rendszer ismerete nélkül egyáltalán érdemes-e tovább firtatni a metaadatbázisok tartalmát? A válasz feltétlen igen. Ugyanis a metaismereteknek vannak olyan sajátos csoportjai, amelyek létével és természetével nem árt tisztában lenni. Főleg azért nem, mert az alkalmazói adatbázisokkal szemben a metaadatbázis kezelésekor kisebb a szabadságfokunk.

9.5 A metaismeretek fajtái

A metaegyedek metatulajdonságait több csoportba sorolhatjuk. Ebben az összefüggésben is beszélhetünk a tulajdonságok azonosító, leíró és kapcsoló szerepeiről. E szerepeken belül pedig érdemes megkülönböztetni a további specifikus feladatokat.

A metaegyed azonosítóját nem tetszőlegesen választjuk ki. Az azonosító általában kettős. Az egyik tétel a ***név***, amit általában a metaegyed nevével minősítenek. Például: Egyedtípus-név. A név szigorúan korlátozott mind a méretét, mind a benne használható jeleket illetően. A másik tétel a tulajdonképpeni ***azonosító***, amit rövid-névnek vagy kódnak is hívnak. Ez is minősített. Például: Egyedtípus-azonosító. Ez a tényező még a névnél is jobban korlátozott. A név az emberi, kvázi-természetes hivatkozás eszköze. Nem teljesen természetes, mert a szigorú méretkorlát azt nem teszi lehetővé. Az azonosítót egyrészt majd az adatkezelő használja hivatkozásra, másrészt ezen alapul a metatényező másokkal való kapcsolata.

Mivel a név hossza korlátos, opcionálisan megadható az általában ***leírásnak*** hívott metaadat tartalma. Ezen elvileg a teljes természetes megnevezést kell érteni, de valójában bármi beírható ebbe az egy-két soros „rovatba". A valódi szöveges leírásra a ***meghatározás*** mező szolgál, az adott kezelő nagyvonalúságától függő méretben. Amint látjuk, ez a páros hordozza az adatszótár valóban szótár-jellegű ismerettartalmát: az ismeret valódi nevét és annak értelmezését.

A leíró szerepű tulajdonságok következő csoportja majdani adatkezelési ***korlátként*** szolgál. Így adhatjuk meg az alkalmazási tulajdonság (pl. Kocsitípus) adattípusát, méretét, validálását, sajátos mintáját és így tovább. Némely esetben az értéket (pl. a méretnél) viszonylag szabadon határozhatjuk meg. Más tételeknél csak szigorúan megszabott választékból válogathatunk. Ritka esetben a rendszer módot ad a készlet bővítésére; például meghatározhatunk új adattípust.

A szótárrendszerek igen kevés ***minősítő*** jellegű metatulajdonságot alkalmaznak. Ez részben érthető is. Az ilyen tulajdonság nem épül be a tényleges adatbázisba. Szerepe az, hogy segítse a fejlesztőt a helyes szerkezetek kialakításában. Egy példa: sok mai rendszer nem tudja expliciten kezelni a családfa szerkezetet (ld. 7.8 pont). Procedurálisan kell megoldani az ilyen struktúrával járó sajátos ellenőrzéseket. Hasznos lenne, ha a szótárban utalni lehetne a metaegyed családfa jellegére és ahhoz kapcsoltan le lehetne írni magát az ellenőrzési rutint is. Ilyesmire igen kevés szótárrendszer nyújt lehetőséget.

A mai szótárkezelők az ***értékelő*** metatulajdonságok terén a legfukarabbak. Az adatbázis szerkezetében a szó szoros értelmében „minden mindennel összefügg". Ezért a teljes kiépítésig a struktúrában mindig lesznek hiányok, redundanciák, ellentmondások. Ha már az adatszótárak átvették a fejlesztési adatbázisok szerepét is, elvárható lenne, hogy jelzéseket helyezzenek el az időlegesen tökéletlennek mutatkozó, később feltétlenül vagy esetlegesen javítandó tényezőknél.

9.6 Passzív, félaktív és aktív adatszótár

A metaadatbázis annyit ér, amennyire összhangban áll az alkalmazási adatbázissal. Ez pedig az adatszótár használatán múlik. Ha az adatbázis terve csak papíron létezik, akkor egészen bizonyos, hogy a tervnek csak halvány köze lesz a valóságos adatbázishoz. A számítógépes adatszótár használata a tudatos adatbázisépítés nélkülözhetetlen feltétele. Az ilyen rendszerek három különböző mértékben szolgálják ki a fejlesztési igényeket.

Vannak önálló (angolul: stand-alone), a használt adatkezelőtől független szótár-rendszerek. Néha - egyre ritkábban - az is előfordul, hogy a fejlesztési és az alkalmazási adatbázist ugyanaz a szoftver kezeli, de úgy, hogy a két adathalmaznak semmi köze sincs egymáshoz. Ilyenkor a metaadatbázis nem játszik szerepet a tényleges adatok kezelésében. Ezért ezek esetében ***passzív*** adatszótárról beszélünk. A „kézi" metaadatbázishoz képest ez is óriási előrehaladás. Minden ilyen rendszer kiszűri a nyilvánvaló átfedéseket és ellentmondásokat. Teljességre kényszerít. A jobbak intelligensebb elemzéseket is biztosítanak. Így a passzív szótár az azonos értelmezésen túlmenően a konzisztens tervezést is támogatja. Azt viszont nem tudja megakadályozni, hogy a felépített metaadatbázist semmibe véve teljesen más szerkezetű/tartalmú valódi alkalmazási adatbázist valósítsanak meg.

A mai adatkezelők többségénél a meta- és az alkalmazási adatbázist bizonyos mértékig összehangolják. Lehetőség van az alkalmazási programok olyan (elő)fordítására, amelynek során használatba kerül az adatszótár. A fordító ellenőrzi, hogy a program által kért tényezők a feltételezett névvel, formában, összefüggésben stb. kerültek-e meghatározásra az adatbázisban. Ellentmondás esetén a programot visszautasítja. Az ilyen módon működő rendszereket hívjuk ***félaktívaknak***. Azért, mert a szótár az alkalmazási program végrehajtása közben nem működik. Így a program hibátlan lefutásának az esélye nem túl nagy. Tudniillik a fordítás végrehajtása után, de a program x-dik futtatása előtt bárki átdefiniálhatja az adatbázis szerkezetét. (Arról nem is beszélve, hogy nincs mód az olyan programok futásának a kizárására, amelyeket nem fordítottak le a fenti módon.)

Az elvileg igazán jó megoldás az ***aktív*** szótár. Ennek használata szervesen beépül magába az adatbáziskezelő rendszerbe. A szótárt nemcsak a fordításnál, hanem a program futása közben is használja a kezelő. Amikor az alkalmazási program az adatbázis valamelyik tényezőjét igényli, a kezelő az óhajt a metaadatbázissal előbb egyezteti és csak azután hajtja végre a kérdéses műveletet. Így a meta- és az alkalmazói adatbázis között tökéletes az összhang. A programozási hiba egyszerűen kizárt.

Erényeik ellenére az aktív szótárak nem terjedtek el. Ennek több oka van. A metaadatbázis programfutás közbeni értelmező-módú (interpretatív) használata ma még időigényes. Az ilyesmire képes szoftvernek a kialakítása sem egyszerű feladat. A legfőbb gond azonban a változtatások menedzselése. Ha a programfordítás és -futtatás közben az adatbázis szerkezetét módosítják, akkor az intelligens szótárkezelőnek végig kellene vizsgálnia, hogy milyen programokat érint ez a változás és ki kellene kényszerítenie azok újrafordítását. Nyilván ez lenne a célszerű megoldás. Azonban megvalósításának még technikai és főleg menedzselési akadályai vannak.

9.7 Tervezés ábra alapján

A 9.3 pontban említettük, hogy a korszerű szótárrendszerek magukba olvasztották a fejlesztőrendszerek képességeit. Ezzel jelentős változás állt be a metaadatbázisok kezelésében.

Az adatszótárak kezelése kezdetben kizárólag ***formaorientált*** volt. A fejlesztőnek tervezési „bizonylatokat" kellett kitöltenie, csak éppen ezek a „formanyomtatványok" számítógépes képernyőkön jelentek meg. (Ezeknek a tipikus „rovatairól" volt szó a 9.5 pontban.) Vagyis a fejlesztő lényegében pontosan úgy kezelte a metaadatbázist, mint a felhasználó az alkalmazásit.

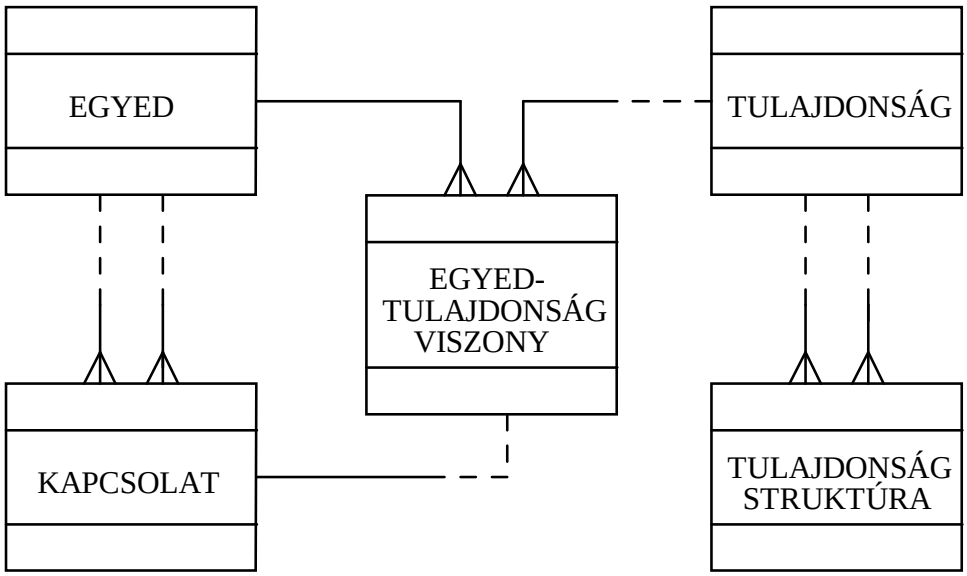
A CASE (Computer Aided Systems Engineering, azaz számítógéppel támogatott rendszertervezés) eszközök megjelenésével terjedt el az ***ábraorientált*** tervezés. Az ember hajlamos arra, hogy a szerkezeti összefüggéseket jobban átlássa egy jól szerkesztett ábra, mint akárhány sornyi jól megfogalmazott szöveges leírás alapján. A CASE eszközök lehetővé teszik, hogy az adatbázis struktúráját képernyős diagramok segítségével határozzák meg és módosítsák. (Most nem térünk ki arra, hogy nemcsak az adatszerkezetet, hanem sok egyéb tényezőt is meg lehet tervezni ezen a módon.)

Az adatbázis modellje a rajzos kezelésnél kétféle alapon kerül meghatározásra. A tervező a rendelkezésre álló ikonkészlet alapján megrajzolja az adatmodellt. A rendszer a kiválasztott ikonok és azok összefüggései alapján automatikusan összeállítja az adatmodellrészt a vonatkozó korlátokkal kiegészítve. Például: Ha két dobozt rajzolunk és azt az egyik végén „varjúlábas" vonallal kötjük össze, akkor felismeri, hogy két, egymással 1:N fokú kapcsolatban álló egyedet kívánunk kijelölni. Ezután az ábrát a diagramba, a mellé és/vagy külön megnyitott ablakokba írt nevekkel és szövegekkel egészíthetjük ki. Természetesen ennek során használhatjuk magát a már korábban feltöltött adatszótárt is.

Bár már vannak igen szép megoldások, az ábra alapján való tervezés még nem igazán terjedt el. Ennek több oka van. Az egyik az, hogy az ilyen eszközök roppant drágák és költséges hardvert igényelnek. Másrészt bármekkora képernyőt használunk is és bármennyi ablakot tudunk nyitni, az áttekintés nehézkes és a használat kényelmetlen. Végül sok rajzos tervezőnek nagyon szűkös az ikonkészlete és ezért a legtöbb metatulajdonságot - főleg a korlátjellegűeket - továbbra is szövegesen kell megadni. Az ilyen problémák dacára is az ábra alapján való tervezés a jövő útja.

A metaadatbázissal - adatszótárral - párosult rajzos tervezés lehetősége előre vetíti azt a korszakot, amelyben az adatbázis-konstruálás tényleg a mérnöki munkára fog hasonlítani. A valóban korszerű lehetőségektől elbűvölve a számítástechnikusok elfeledkeznek arról, hogy az eszközöket nem csak technikailag kell tudni használni. Amint az alkalmazási adatbázisnak is vannak menedzselési aspektusai, ugyanúgy törődni kellene a fejlesztési adatbázis használati módjával is. Erről is lesz szó a következő fejezetben.

Előbb azonban a 9.2 ábrán megadjuk az információs erőforrás szótár modelljének azt a részletét, amely kimondottan az ismeretekre vonatkozik. Tehát ezen az ábrán nem tüntetjük fel az információs rendszer egyéb tényezőit - felhasználók, eszközök, feldolgozások -, hanem csakis az adatmodell metamodelljét, az egyed-, tulajdonság- és kapcsolattípusok viszonyait mutatjuk be.



9.2 ábra: A metamodell diagramja

Az ábra magyarázata a következő: Az adatmodell egyedtípusaihoz tulajdonságtípusok kapcsolódnak. Egy egyednek több tulajdonsága lehet és egy tulajdonság több egyedet jellemezhet. Az M:N fokú viszony miatt ezt az összefüggést külön metaegyed (egyed-tulajdonság viszony) írja le. Ennek kapcsolata az egyedhez kölcsönösen kötelező. Nincs egyed tulajdonság nélkül és nincs tulajdonságviszony egyed nélkül. Viszont az adatmodellben létezhetnek olyan tulajdonságok, amelyek nem kapcsolódnak egyedekhez. Ilyen ismeretek például a konstansok. Ezért a tulajdonságoknak az egyedekhez való viszonya felülről opcionális.

Az egyedek kapcsolatokban állnak. Ez a viszony családfa jellegű, kettős kapcsolat. Mert egy egyednek lehetnek alá- és fölérendeltjei. Minden kapcsolat egyedtípusok között létezik, ezért e viszonyok alulról kötelezőek (folyamatos vonal). Nem minden egyednek van fölé- és alárendeltje, ezért e kötődések felülről opcionálisak (szaggatott vonal). Sajnos, azt nem lehet kifejezni ilyen ábrán, hogy a két kapcsolat közül ***valamelyik*** felülről kötelező, azaz nem létezhet az adatmodellben más egyedhez nem kötött egyed.

A kapcsolat és az egyed-tulajdonság viszony közti vonal azt fejezi ki, hogy egyes attribútumok kapcsoló tulajdonságként szolgálnak. Minden kapcsolat ilyen viszonyon alapul, de nem minden attribútum kapcsoló tulajdonság.

Végül a tulajdonságok családfa jellegű struktúrákat alkotnak. Ha a Rendeléstétel-érték a Rendelt-mennyiség és az Egységár szorzata, akkor ebből a két tulajdonságból „épül fel”. A kapcsolat azért kettős, mert van beépülési vetület is. A Rendelt-mennyiség nemcsak a Rendeléstétel-érték kiszámításához szükséges. Felhasználhatjuk a Rendeléstétel-súly számításához is, az Egységsúly segítségével. Egy másik fajta tulajdonság-struktúra a csoport. Például a Dátum tulajdonság az Év, Hó és Nap tételekből épül fel.

10. ADATBÁZIS-MENEDZSELÉS

10.1 Mitől lehet rossz az adatbázis?

Egyre korszerűbb számítógépeket alkalmazunk. Egyre összetettebb és drágább adatkezelőket vásárolunk. Mind több és több ismeretet tárolunk „adatbázisban". Érdekes módon mintha mégsem születne több információnk. Vajon mi lehet ennek az oka?

Bár még ma is sok problémát okoz a hardver helytelen kiválasztása, az adatkezelő korlátos képessége és nem megfelelő alkalmazása, mégsem a „ver"-ekben kell keresni a hibát. Bármennyire is el tudunk képzelni tökéletesebb eszközöket, be kell látnunk, hogy még a maiak lehetőségeit sem vagyunk képesek megfelelően kihasználni. Az átlagos szakember a korszerű adatbáziskezelő lehetőségeinek mindössze 20 százalékát veszi igénybe...

Az igazság az, hogy mi magunk vagyunk a bajok okai. Nem ismerjük az adatok természetét és csak halvány elképzelésünk van arról, hogy mi is az az adatbázis. Az előző fejezetekben arra világítottunk rá, hogy az adatbázis nem pusztán ismereteink általunk ilyen-olyan módon elképzelt halmaza, hanem egy nagyon pontosan meghatározandó rendszer. Ha az adatbázis nem szolgálja ki ismeretigényeinket, annak elsődleges oka minden valószínűség szerint az az egyszerű tény, hogy ***adathalmazunk nem rendszerezett, tehát nem valódi adatbázis.***

Ám tegyük fel, hogy a szakember az ismereteket tudatosan, adatmodellt építve, az egyed-tulajdonság-kapcsolat hármasra figyelve rendezi el. A gyakorlatban mégsem minden úgy sikerül, ahogy elgondolja. A tárolás redundáns, a kezelés nehézkes és nem eléggé hatékony. Mi okozza ezt? A tervezők elfeledkeznek arról, hogy nem pusztán egy lehetséges adatbázist, hanem a lehető legjobbat kellene megtervezniük. Nagyon rossz, átgondolatlan struktúrájú adathalmazokat is lehet adatbázisszerűen kezeltetni. A mai adatbázisok nagy része azért nem-hatékony, mert a ***tervező nem kereste a strukturális optimumot***.

Az adatmodell-elemzés és -optimalizálás komoly felkészültséget igénylő rész-szakma. Ezekről a témákról a második részben lesz majd szó.. Viszont minden adatmodellel szemben vannak olyan minimális - látszólag pusztán formai - követelmények, amelyeknek az elhanyagolása komolyan megbosszulja magát. Adatbázisaink sokszor azért tökéletlenek, nehézkesen kezelhetők, mert ***tervezésük összehangolatlan és helytelen szemléletű***.

Ebben a fejezetben az utóbbi problémára irányítjuk a figyelmet és azon keresztül bemutatjuk az adatbázisok menedzselésének a szervezeti-technikai követelményeit illetve módjait.

A fejezetben arra is kitérünk, hogy a legtökéletesebb szerkezet sem garancia arra, hogy az adatbázisból megfelelő ismereteket szerezzünk. Az adatbázis leggyakrabban attól rossz, hogy nem megfelelő az abban ***tárolt adatok minősége***. Mint majd kiderül, közvetve ez a probléma is az adatmodell helytelen meghatározásával függ össze.

10.2 Hány az adatbázis?

A mindennapi életben gyakran halljuk, hogy valaki létrehozott egy ilyen, egy olyan, no meg egy amolyan adatbázist. Például az ügyfelek adatbázisát, a vállalati dolgozók adatbázisát, a szakértők adatbázisát, a pénzügyi adatbázist stb.

Aki az „adatbázis" szót adott szervezeten belül többes számban használja - „adatbázisaink" - az semmit sem tud az adatbázis lényegéről. A vezetők, felhasználók és fejlesztők nem hajlandók megérteni és felfogni a legfontosabb informatikai tételt, amely szerint

ADATBÁZIS CSAK EGY VAN!

Meglepő, kemény és az első ránézésre érthetetlen ez a kijelentés. Hiszen bárki megesküdne rá, hogy náluk több adatbázis is van. Ezért a kitétel némi magyarázatot igényel.

A vezetők, fejlesztők és felhasználók még mindig abban a tévhitben élnek, hogy az adatbázis a fizikailag együtt tárolt, az adatbáziskezelővel közösen manipulált adatok halmaza. Ezt sugallják az elméleti szakkönyvek és a szoftvereladók félszakmai szintű kiadványai is. Ha tehát két vagy több olyan adathalmazuk van, amelyeket fizikailag nem együtt tárolnak és/vagy külön kezelnek, akkor jogosan beszélnek két vagy több adatbázisról.

E téves felfogás alapja az, hogy a szakemberek nem veszik figyelembe az adatbázis három szintjét (ld. 4. fejezet), és az adatbázist csak a logikai-fizikai szinteken szemlélik. Elfeledkeznek arról az alapvető tényről, hogy fogalmi szinten

MINDEN ISMERET MINDEN MÁSIKKAL ÖSSZEFÜGG.

Persze ez a kitétel nem azt jelenti, hogy minden adat minden másikkal közvetlen kapcsolatban áll. Hanem azt, hogy az ismeretek - legalábbis közvetve - egymásra épülnek, és ezért nem szabad elfeledkezni fogalmi összefüggéseik vizsgálatáról akkor sem, ha azokat külön logikai-fizikai adatbázis-részekben tároljuk és kezeljük. Egy példával szemléltetjük, hogy mi a következménye annak, ha több adatbázisban gondolkodunk.

E pont bevezetőjében említettük az ügyfél, a dolgozó, a szakértő és a pénzügyi adatbázist. Mármost Kovács Lajos olyan dolgozónk, aki egyben ügyfelünk is és egyes, a munkakörét meghaladó feladatokban szakértői munkát is végez nálunk (pl. szuahéli tolmács). Mivel három független adatbázisról van szó, egészen természetes, hogy Kovács úr nevét mindháromban más módon írjuk. Például: KOVACS Lajos, Kovacs Lajos dr., Dr. Kovács Lajos. Kovács úr kap bért, a külön munkáért díjazást. Ügyfélként pénzügyi tranzakciókban áll vállalatunkkal. Ki fogja tudni kideríteni, hogy miként is állunk Kovács úrral anyagilag, hiszen a három adatbázis Kovács-a semmiképpen sem hozható egymással összefüggésbe?

Kovács Lajos egy - a vállalatunk számára fontos - jelenség, vagyis egyetlen egyed. Evidens, hogy fogalmilag csak egy informatikai tükörképe lehet: a SZEMÉLY egyed egyik előfordulása. Logikai és fizikai szinten a Kovácsra vonatkozó ismereteket több adategységben is tárolhatjuk (DOLGOZÓ, SZAKÉRTŐ, ÜGYFÉL), de csak azután, hogy Kovács úr önmagával való azonosságát előzetesen már tisztáztuk.

Ha így gondolkodunk, akkor ügyelni fogunk arra, hogy Kovács úr becsületes nevét minden „adatbázisban" azonos módon tüntessük fel és ezért a logikailag megosztott egyetlen adatbázis részei között mindig megtaláljuk az összhangot. Viszont akkor, ha az „adatbázisokat" egymástól függetlenül külön-külön csoportok hozzák létre, akkor halvány esély sincs arra, hogy a Kovács úrra, mint egyetlen jelenségre vonatkozó összes ismeret egymással harmonizáljon.

A fentiekből egyenesen következik, hogy egy szervezet csak akkor tudja megfelelően kezelni az ismereteit, ha a konkrét logikai-fizikai szintű adatbázisok kialakításakor nem feledkezik meg az alapvető szabályról, amely szerint fogalmi szinten csakis egyetlen adatbázis létezik. A fejezet összes többi pontja a jelzett tételre épül illetve azt támasztja alá.

10.3 Alkalmazási adatszabványok

Az „egy az adatbázis” tételt kevesen ismerik fel. A vezető egyszerre csak egy-egy terület számítógépes támogatását tudja elindítani. A felhasználót természetesen csak a saját funkciója érdekli és nem fog törődni a mások számára fontos ismeretekkel. A fejlesztő pedig örül, ha a számára kirótt feladatnak eleget tud tenni; nem lesz ideje arra, hogy széttekintsen. Ezért úgy tűnik, hogy Kovács úrra az a sors vár, hogy nevét három adatbázisban háromféle módon írják. A kérdés az, hogy amennyiben a vezető, a felhasználó és a fejlesztő egyike sem képes átlátni az egyetlen elvi adatbázist, akkor milyen módon harmonizálhatók a széttagolt adatbázisrészekben tárolt ismeretek?

A válasz egyetlen szó: adatszabvány. Az adatszabványok olyan megegyezések (konvenciók), amelyek az ismeretek tartalmát és megjelenítési formáját szabályozzák. Fontosságukat a fejlett informatikai környezetekben már régen felismerték. Magyarországon az informatika pontosan olyan mértékben marad el a korszerű színvonaltól, amilyen mértékben hiányoznak e szabványok.

Az alkalmazási adatokkal kapcsolatos szabványoknak számos válfaja van. Ezek közül most csak párat akarunk kiemelni.

Azonosító-szabványok. Könyvünk korábbi fejezeteiben számtalanszor utaltunk arra, hogy az azonosító abszolút szerepű tulajdonságok relatív szerepeiken keresztül sokszor az egyedek közti kapcsolatok megteremtésére szolgálnak. (Az alapvető szerepeket a 6. fejezetben tárgyaltuk.) Ha tudni akarjuk, hogy ki adta fel az X rendelést, akkor a rendelésben szerepelnie kell a Vevőkód megfelelő értékének. Mindez feltételezi, hogy a Vevőkód a VEVŐ és a RENDELÉS egyedekben azonos lényegű, felépítésű, formájú legyen. Saját adatbázisunk is egy pillanat alatt használhatatlanná válna, ha e szabályt nem tartanánk be.

Ámde mi a helyzet akkor, ha a szervezetben több adatbázis van? Például akkor, ha a szerződést Szegeden összetett területi jelzőszámmal, Baján kötvényszámmal, Budapesten pedig folyamatos sorszámmal jelölik? Mint tudjuk, a szerződések „mozognak”. Például az ügyfél költözése miatt átkerülnek Szegedről Bajára. Nos ekkor a bajai adatbázisban és papírokon mindenütt át kell írni a területi jelzőszámot az új kötvényszám azonosítóra. Az „ABC” ismeretből így „1234” adat lesz. Ez a munka még hagyján, de holnap jön egy bírósági per. Elő kell keresni, hogy mi történt a szerződés kapcsán még Szegeden. Csakhogy az „ABC” azonosító már nincs az adatbázisban. Az ún. történeti adatok visszakereséséhez arra lenne szükség, hogy az új azonosító („1234”) mellett megőrizzük a régit („ABC”) is. Tehát négyszeres átadás esetén már három volt azonosítót kellene vezetnünk...

Ebből a csapdából csak egy kiút van: az azonosítók tisztességes szabványosítása. A cégnek előre le kell szögeznie, hogy a szerződéseket az egész szervezetben a Szerződésszám azonosítja, amelynek lényege, szerkezete, formája ez meg ez... Mindez praktikusan azt jelenti, hogy minden azonosító bevezetését egy szabvány-körlevélnek kellene megelőznie. Ezzel szemben a magyar gyakorlat az, hogy az azonosítókat a felhasználók és a fejlesztők ízlés szerint jelölik ki és váltogatják.

Kódszabványok. Most nem a feldolgozási kódokról van szó, bár az sem egészen ildomos, hogy az egyik programban „B” (bevitel), a másikban „A” (angol „add”), a harmadikban „1” stb. jelöli az adatbetöltést. Azokról a felhasználói érdemi kódokról beszélünk, amelyek bizonyára több

adatbázisban is előfordulnak. Ilyen például a Megyekód, a Valutakód, a Mértékegységkód stb. Ma a különböző adatbázisokban teljesen eltérő kód-értékkészletekkel dolgoznak. Ezért ha az „X" adatbázisból nyert ismereteket az „Y" adatbázis adataival össze kell kapcsolni valamelyik kód alapján, akkor konvertáló-átértelmező táblázatokra és programokra van szükség. Például az „123" módon jelölt foglalkozás valójában megfelel a másutt „AB" értékkel tároltnak.

A konverzió felesleges munka. Ám nem ez az egyetlen probléma. A kódokhoz megnevezések is párosulnak. Például: „123 = cipész". A jobbik esetben a másik adatbázisban „AB = cipész" szerepel. Rosszabb esetben pedig „AB = varga". Nap mint nap tapasztalhatjuk az azonos szervezeten belüli adatbázisok ilyen ellentmondásait. Tipikus példa az „irányítószám-jegyzék", amely érdekes módon minden - más adatbázist használó - szervezeti egységnél eltérő tartalmú.

Az ilyen következetlenségek kiküszöbölésének egyetlen módja a kódszabvány. Minden egyes kódolt ismeret meghatározását meg kellene, hogy előzze a szabvány kihirdetése. Például: mától fogva az irányítószámot mindenki ilyen formában, az X felelős által aktualizált tartalommal köteles használni.

Írásszabványok. Gyakran előfordul, hogy ugyanaz az adatbázis ugyanannak a jelenségnek többszörös ismereti tükörképét tárolja. Például a SZEMÉLY adatbázisban Kovács Lajosnak több „rekordja" van. Az adatbázis-elmélet ezt a redundanciát kizárja, viszont a gyakorlatban sokszor nem tudjuk elkerülni. Mi ennek az oka és miként lehetne a nyilvánvaló bajt megelőzni? Mert ha Kovács úrnak mint ügyfélnek több ismeretsora is van, akkor abból több gond fakadhat. Vagy nemlétező tartozását fogjuk követelni tőle (Kovács úr befizetett, de a másik rekordhoz kötődik ez az ismeret), vagy valódi befizetési kötelezettségéről is elfeledkezünk (Kovács úr nem fizetett, de az aktuális rekord szerint nem is kell fizetnie, csak a másik szerint). Tehát minden attól függ, hogy melyik „Kovács-rekordra" találunk a kezelés során.

A probléma gyökerét az azonosításban kell keresnünk. A 3.6 pontban utaltunk az azonosítás nominális és deszkriptív módjára. Mindaddig, ameddig Kovácsnak volt egy kvázi-nominális személyi száma, azonosítása nem okozott gondot. A félig-természetes személyi számukat sokan meg tudták jegyezni és be tudták diktálni. E nominális azonosító szerencsétlen megszüntetése után az adatbázisokban mesterséges azonosítókat kellett felvenni. Persze az ügyfél ennek értékét nem tartja fejben, azt a papírt pedig, amire fel van írva a Törzsszám értéke, otthon felejti. Ekkor kezdődik a vadászás az adatbázisban - vagyis a deszkriptív azonosítás. „Neve? Anyja neve? Született?" stb.

Persze „dr. Kovács Lajos"-t nehezen fogjuk megtalálni, ha az adatbázisban „KOVÁCS LAJOS DR." megjelölés szerepel és nem ismerjük ezt az írásmódot. Mivel az ügyfél fontos, a pénze pedig még inkább az, nem találván a már tárolt ismeretet létrehozunk egy új rekordot új azonosítóval és a „Kovács Lajos, dr." módon írott névvel. Pár hónap múlva már öt Kovácsunk lesz az adatbázisban, mert időközben áttértünk az egyik ékezetes írásmódról a másikra ...

Ezt az egészen mindennapos, de igen komoly gondokat okozó problémát ismét csak szabványosítással lehet kiküszöbölni. A deszkriptív azonosítást támogató tulajdonságtípusokra (név, megnevezés, cím, dátum stb.) az írásmódot szabályozó konvenciókat kell kidolgozni és kötelezővé tenni.

A fentiekből látható, hogy az alkalmazási adatokra vonatkozó szabványok olyan sajátos adatmodellezési korlátok, amelyek nélkül használható adatbázis nem létezhet. Ezt a tényt egyre többen felismerik és meghatározzák az alkalmazási adatszabványokat. Majd csodálkoznak azon, hogy a recept mégsem válik be és adatbázisuk továbbra is kezelhetetlen ismereteket tartalmaz. Ennek a jelenségnek több oka van.

Az egyik az, hogy a szabvány papír-malaszt marad. A szabvány ellenőrzését nem építik be az adatkezelő programokba. A másik az, hogy a szabvány programmal nem ellenőrizhető. Tehát a vezetőnek oda kellene mennie és meg kellene néznie, hogy az adatrögzítő helyesen viszi-e be az adatot. Csakhogy erre a vezető „nem ér rá"; no meg egyébként sem szokta gyakorolni ellenőrzési funkcióját. A harmadik probléma az időfaktorral függ össze. Már van adatbázisunk - illetve adat-

masszánk, amit az adatbázisba zúdítottunk - amikor „felfedezzük” a szabványt. Az új adatok bevitelénél érvényesítjük a koncepciót, de a régi adatokat nincs energiánk átdolgozni. Ilyen esetben a szabvány persze tökéletesen felesleges. Mert továbbra is lesz egy „KOVACS LAJOS” és egy „Kovács Lajos, dr.” rekordunk ...

10.4 Fejlesztési adatszabványok

A 9. fejezetben beszéltünk a metaadatbázisokról. Remélhetőleg sikerült kimutatnunk, hogy az alkalmazási és a fejlesztési adatbázis felépítése teljesen analóg. Az alkalmazási adatbázisban meg kell különböztetni az egyik Kovács Lajos adatsorát a másik, vele nem azonos Kovács Lajos adatsorától úgy, hogy egyikükre nézve se tároljunk kettős ismeret-sorozatot. Ez a megállapítás a fejlesztési adatbázisra is vonatkozik. A Rendelésszám tulajdonságtípus megjelenéseit el kell választanunk egymástól, ha nem azonos lényegről van szó (vevői és szállítói rendelésszám) úgy, hogy a vevői rendelésszám minden adatbázisrészben azonos módon jelenjék meg.

A vezetők, felhasználók és fejlesztők nem ismerik az „egyetlen adatbázis” elvét. Az utóbbiakat kivéve nem szoktak adatkonvenciókat alkalmazni. Ez a kisebbik baj. Sokkal komolyabb gond az, hogy a fejlesztő senki más által nem érthető „magánszabványokat” kezd használni, fittyet hányva „okos” megoldásainak a következményeire. Pillantsunk csak be például az „A” és a „B” fejlesztő lelkivilágába!

Az „A” fejlesztő mániája, hogy a tulajdonságtípusnak az egyedtípus nevének a kezdőbetűjével minősített megnevezést ad. Így a KOCSITÍPUS egyedben lévő Kocsitípus tulajdonság a K_Kocsitipus nevet fogja nyerni. Csakhogy a tulajdonság a KOCSI egyedben is szerepel. Ott már nem lehet K_Kocsitipus a neve - a fejlesztő szerint. Ezért hallatlan munkába fogva az előző egyed minden viszonylatában átírja a K_Kocsitipus nevet T_Kocsitipusra (a kocsi*T*ípus egyszerű megfontolás alapján) és a KOCSI kocsitípusát fogja K_Kocsitipus névvel jelölni. Ekkor lép be a KÁR egyed, amelyben szintén szerepel a kocsitípus, amelynek neve R_Kocsitipus (ká*R*) lesz. A kifizetésé pedig F_Kocsitipus stb. A fejlesztő már rég elfeledkezett arról, hogy az egyedtípus nevének a kezdőbetűje volt a vezérlőelv. Négy K-val már ő sem tud mit kezdeni...

Ugyanakkor a „B” fejlesztő irtózik a minősítésektől. Ő a nemesen egyszerű „Nev” megjelöléssel illeti a személy, cikk, eszköz, vevő, szállító stb. megnevezését. Számára nem létezik megrendelési-, szállítási-, születési-, házassági- stb. dátum, csak „Datum”.

A fejlesztési adatbázisokban egészen kaotikus állapotok uralkodnak. Az abban tükrözött jelenségek nem azonosíthatók. Mert ugyanaz a lényeg (kocsitípus) különböző neveken „fut” (K_Kocsitipus, T_Kocsitipus stb.) és különböző jelenségeknek (házassági- és szállítási-dátum) azonos megnevezést adnak. Persze ezek után a felhasználó sem látja át az adatbázis lényegét és az adatkezelő programokban is mindenféle csavarintásokat kell alkalmazni. Hogy az X és az Y most valójában azonos, de azon feltétel esetén az X és az X teljesen más...

Ez nem informatika - ez kuruzslás. Informatika az, ha a metaadatbázisban - az adatbázis tervében - az azonos lényegeket egyetlen módon jelöljük és kizárjuk a különböző jelenségek azonos módon való hivatkozását. Ezért a metaadatbázisban is be kell vezetni az ***azonosítókra*** és a nevek ***írásmódjára*** vonatkozó szabványokat. Informatikailag gyengécske szervezet az, amelyben a programozó az adatbázis tételeit a saját maga feje szerint jelöli meg.

Az előző kitétel az adatbázisterv kódoltan minősített tételeire is vonatkozik. Lehetetlen állapot, hogy az egyik adatbázis tervében „C”, a másikban „K” jelölje a karakteres adatot. Ezért a fejlesztési adatbázisok terén is érvényesíteni kell az adatszabványok harmadik fajtáját, a ***tervezési kódszabványokat***.

Az adatbázistervezési segédeszközök - a célnak megfelelően - meglehetősen szigorúak a fenti három szabvány tekintetében. Megkötik az adatbázis tényezőinek az azonosítását, a nevek írását és a kódok alkalmazását is. Viszont ugyanakkor rugalmasak is, amennyiben nem írják elő például a nevek formáját. Ezért nem kizárt, hogy például az egyik egyedben K_Kocsitipus, a másikban Kocsitip_K nevet használjanak ugyanannak a tulajdonságnak a jelölésére. A szótárkezelők rugalmassága abban is megnyilvánul, hogy lehetővé teszik a különböző adatbázisrészek eltérő részletezettségű és mélységű dokumentálását. Ez a tervezési folyamat során így is célszerű. Viszont a végső tervnek nyilván homogénnek kell lennie.

Mivel a szótárkezelők lehetőségei tágasak, a szervezetek el nem hanyagolható feladata az ún. ***metaszabványok*** kidolgozása.

D 10/1 **A metaadatbázis tényezőire, azok viszonyaira (a szerkezetre), illetve a metaismeretek tartalmára és formájára vonatkozó egyezményeket metaszabványoknak nevezzük.**

Az alkalmazásiakkal szemben a metaszabványok általános érvényűek, vagyis azokat minden információs fejlesztésnél be kell tartani. Elő kell írni a dokumentációk szerkezetét, általános tartalmát, a listák és diagramok formáját, a metaadatok megnevezési és egyéb konvencióit. Világos, hogy a metaszabványoknak a fejlesztések megkezdése előtt kell rendelkezésre állniuk. Mivel pedig a szabvány annyit ér, amennyire betartják, gondoskodni kell arról, hogy az elkészülő adatbázisterveket ellenőrizzék a szabványosság szempontjából. A szabványok érvényesülését biztosító szervezeti megoldásról majd az alábbiakban lesz szó.

10.5 Fejlesztési eljárásszabványok

A fejlett informatikájú szervezetekben szabványok és szabványos eljárások (angolul: standards & procedures) segítik és korlátozzák a fejlesztők munkáját. Tehát nemcsak tartalmi és formai adatszabványokat, hanem a fejlesztési lépések tevékenységeire vonatkozó szabványokat is alkalmaznak.

Most nem az információs rendszerek ***fejlesztésirányítási*** (angolul: project management) módszeréről és az ahhoz kapcsolódó konvenciókról van szó. Ezek az irányítási szabványok és eljárások közismertek. Módszerekkel és eszközökkel jól támogatottak. Kiemelkedő példaként említjük az SSADM módszert [18]. A „project management” nem ennek a könyvnek a tárgya. Arra sem fogunk kitérni, hogy nálunk miért nem képesek a szervezetek elfogadni, elsajátítani és tudatosan alkalmazni egy ilyen irányítási módszert. Minket most másféle eljárások érdekelnek.

A modern fejlesztésirányítási módszerek „termékorientáltak”. Mit kell ezen érteni? Az angol informatikában ***tervterméknek*** (angolul: design product) nevezik a rendszertervet. Persze ez a termék, amely a tervezési munka eredménye, igen összetett. A módszerek pontosan előírják, hogy a tervterméknek milyen elemei vannak; azok hogyan függenek össze egymással; milyen tartalmi és formai konvenciók szerint kell dokumentálni őket stb. A tervtermék elemeit „leszállítandó dolgoknak” (angolul: deliverables) hívják. Amikor azt mondjuk, hogy a modern módszerek termékorientáltak, azon azt értjük, hogy a módszerek a „leszállítandó dolgokra” koncentrálnak. Meg kell határozni a táblaképeket, a táblák kapcsolatait, az indexeket stb.

Az előbbi állítás egy implicit tagadást is tartalmaz. Ha egy rendszertervben szerepelnie kell például a rekordképeknek, akkor a rekordképek leszállítandó, átadandó dolgok. Tehát a tervező egyik feladata (angolul: task) a rekordképek megalkotása. Ezért a módszerek előírják a feladatokat és az átadandó dolgokat (tasks & deliverables). Viszont általában nem tartalmaznak

eligazítást a ***tervezési folyamatra*** (angolul: design process). Például az SSADM kimondottan hangsúlyozza, hogy nem kíván tervezési eljárási technológiát rögzíteni. Így tehát a korszerű módszertanok nem „folyamatorientáltak”.

A tervezési módszereknek a tervezési folyamattal kapcsolatos tartózkodását sokan félreértik. Nem arról van szó, hogy ne lenne szükség egy tudatosan meghatározott tervezési eljárásra. A számítógépes eszközökkel támogatott korszerű módszerek hallatlan rugalmasságot nyújtanak. A terv részleteit szinte tetszőleges sorrendben, különböző pontossággal megadott módon lehet meghatározni. Ez a hajlékonyság két előnnyel jár. Egyrészt segíti a kísérletezést. Az ember a papíron leírt tervet nem szívesen alakítgatja, mert az munkás. Számítógéppel könnyebben lehet átalakításokat végezni a terven. Másrészt az eszközök támogatják azt is, hogy amint valami bizonyosat tud a tervező vagy mihelyst valamilyen elképzelése támadt, azt azonnal számítógépen rögzítse. Tehát a mai tervezési módszerek azért nem írnak elő tervezési eljárásokat, mert nem akarják megkötni a fejlesztők kezét. Az egyik szervezet ilyen, a másik amolyan szokások szerint fejleszt. Az egyik rendszerhez ez, a másikhoz az a módszer illik jobban.

A fentiek miatt nem szabad abban a tévhitben élni, hogy nincs szükség tervezési eljárásra. Éppen ellenkezőleg. Mivel a módszerek a tervezési folyamatot nem szabályozzák, azt magának a szervezetnek kell megtennie fejlesztési eljárásszabványok formájában. A kérdés az, hogy miért?

A korszerű módszerek eszközeinek a rugalmassága egy igen nagy veszélyt hordoz magában: az összehangolatlanságét. Ha mindenkit csak úgy odaengedünk a CASE-eszközhöz (ld. 9.7 pont), akkor abból csak káosz keletkezhet. A legkorszerűbb technikával támogatjuk a legelavultabb (individualista) fejlesztési szemléletet. A mai CASE-ek nem zárják ki az egyénieskedést. Az egyik tervező itt, a mások ott - fizikailag is más gépen - építgetheti a „saját” adatszótárát. Az se kizárt, hogy szótáraik azonos gépen vannak, de logikailag teljesen szeparáltak. Mindez teljesen ellentmond az „adatbázis egy” elvnek. Ugyanis igen kicsi az esély arra, hogy az össze nem függő szótárrészekben ugyanazt a lényeget (pl. Kocsitípus) azonos módon tükrözzék.

A végső-felhasználó nem akkor és nem úgy nyúl az alkalmazási adatbázishoz, amikor és ahogyan akar. Kovács ügyintéző és Szabó ügyintéző nem építhet két külön szerződés állományt. A végső-felhasználó műveleti lehetőségeit felhasználói eljárások szabályozzák. Miért lenne ez másként a fejlesztői adatbázis esetében? A mai módszerek nem írják elő a fejlesztési adatbázissal kapcsolatos eljárásokat. Kovács fejlesztő és Szabó fejlesztő egészen nyugodtan szerkeszthet két külön szerződés adatbázist. Egymástól függetlenül, egymásnak ellentmondóan. Nem kellene ezt megakadályozni fejlesztési szabványokkal?

10.6 Menedzselési szabványok

Az eddigi fejezetek során az adatbázis felépítésével foglalkoztunk, vagyis az adatbázis elvi szerkezetének, modelljének a megalkotására koncentráltunk. Azonban világos, hogy a modellt azért készítjük, hogy azt majd megtöltsük tartalommal, tehát életet adjunk az adatbázisnak, amit majd hosszan és kényelmesen kívánunk működtetni. Azonban - mint a fejezet első pontjában említettük - a jó struktúra nem garancia az adatbázis használhatóságára. Ha az adatbázisban tárolt adatok minősége nem megfelelő, akkor hiábavalóak a tervezési erőfeszítések. A szerkezet és a tartalom elválaszthatatlan egymástól, és ezért beszélnünk kell az adatbázis életének a veszélyes időszakairól.

Az adatbázis életének talán a legkritikusabb pillanata a megszületés, ***az adatbázis létrehozása***. Ez két momentumot ölel fel. Az első az adatbázis szerkezetének a gépen való definiálása, ami egyben fizikai helyének a kijelölését is jelenti. Magyarországon e téren nem alakult ki még a másutt már szokásos munkamegosztás. A számítógép erőforrásaival általában a programfejlesztő,

nem pedig az ismeretek valódi tulajdonosának, a felhasználónak a képviselője gazdálkodik. Így előfordul, hogy a gépeket, szoftvereket, az adatelhelyezéseket ad-hoc módon cserélgetik és ezzel rontják az adatbázis hozzáférhetőségét. Minden vállalatban illenék alkalmazni olyan szabványos eljárásokat, amelyek a felhasználó érdekében szabályoznák a ***technikai váltásokat***.

A bevezetés másik momentuma az ***adatkonverzió***. Manapság már nem az jellemző, hogy olyan új tartalmú adatbázisokat hoznak létre, amelyeket papír adathordozókon lévő ismeretekkel kell fokozatosan feltölteni. Sokkal gyakoribb, hogy már meglévő számítógépes állományokat kell az új adatbázis szerkezetének megfelelően áttölteni. Ez nagyobb méretű adatbázis esetében összetett és hosszadalmas munka. Vezetni kell, hogy milyen adatok kerültek már áttöltésre. Lehet, hogy kétszeres karbantartásra van szükség az állományok régi és új példányán. Formai, tartalmi és szerkezeti összehangolási igények is felléphetnek. Formai: A régi adatok még nem ékezetesen voltak írva, tehát az új adatbázisban módosításokat kell végrehajtani. Tartalmi: A régi ismeretet csak 3 szempontból osztályoztuk, most pedig már 5 szempontunk van. Ezért áttöltéskor szükség lehet az átkódolásra. Nagyon gyakran előfordul az is, hogy az új rendszerben más azonosítót alkalmazunk. Szerkezeti: A régi állományok adattételeit általában több egyedtípus között kell szétdarabolni az új adatbázisban.

Mindezt azért mondjuk el, hogy felhívjuk a figyelmet az adatbázis „pubertás" korára. Sokszor elfeledkeznek arról, hogy a régi ismereteket majd át kell tölteni az új adatbázisba. Csak akkor gondolkoznak el ezen a problémán, amikor maga az áttöltés aktuálissá válik. Általában az ilyen konverzió nem valósítható meg bizonyos, időlegesen vezetett, az új adatbázisban tárolt és kezelt ***technikai adatok*** nélkül (ld. 9.8 pont). Ezek nem az adatbázis szerves részei. Ha a konverziót nem gondolták át előre, akkor az amúgy jó adatstruktúrát korrumpálva (lerontva), kapkodó módon, az utolsó pillanatokban határoznak meg mindenféle technikai adatot. Nem gondolva arra, hogy ezek majd megszűnnek és csakis az adatbázis „tinédzser" korában játszanak szerepet.

Ezt a problémát elkerülendő minden szervezetben be kellene vezetni a ***konverzió menedzselési szabványait***. Ez szabályozná az adatkonverzió folyamatát. Előírná, hogy mikor, milyen formában kell elkészíteni a „konverziós térképet", amely megfelelteti a régi és az új adatszerkezetet. Rögzítené, hogy melyek azok a konverziós technikai adatok, amelyek csak az áttöltés végéig szükségesek és előírná azok kezelését illetve majdani megszüntetésének módját.

A „felnőtté vált" - a konverziós adatoktól megtisztított - adatbázissal kapcsolatosan is számos menedzselési feladat lép fel. Ezek a tevékenységek részben folyamatosak, részben időszakosak. Szerencsére még nem vagyunk teljesen gépiesek, vagyis az ismeretek jó része automatikusan nem ellenőrizhető a számítógép által. Ezért minden vállalatban szükség lenne a ***kommunikációs monitor*** funkcióra: a bemenetek, az adatbázis és a kimenetek emberi ellenőrzésére. Csakis emberi kontrollal állapítható meg, hogy helyesen viszik-e be a fáradtságuk miatt slamposságra hajlamos adatrögzítők az ismereteket. Időszakosan futtatott, netán automatikusan induló monitorprogramokkal kellene vizsgálni az adatbázis épségét, ún. integritását. Van-e az adatbázisban olyan rekord, amely a szerkezet szerint máshoz kellene kapcsolódjon, de még sincs aktuális viszonya („láncait vesztette"). Nem került-e az adatbázisba mégis a tartalmi/formai alkalmazási szabványt sértő tétel, amely esetben a bemeneti validálási programokat kell felülvizsgálni. Nem lenne szabad elhanyagolni a kimenetek „szemmelverését", emberi áttekintését sem. A formai hibákon - például a helytelen magyarsággal írt kimenetcímeken - kívül fel fogunk fedezni „számszaki" pontatlanságokat is. Mert a fejlesztők gyakran elfeledkeznek arról, hogy nemcsak a bemeneteket, hanem a kimeneteket is lehet, sőt kell validálni. Tehát minden vállalatnak feladata, hogy kidolgozza a ***monitorfunkció szabványait***.

Ezzel a menedzselési feladatok sora még messze nem zárult le. Az adatbázis épsége (angolul: integrity) mellett figyelmet kell szentelni a biztonságnak (angolul: security). Ki kell dolgozni az adatbázis mentésének, az esetleges lerobbanás utáni visszatöltésnek és újrafeldolgozásnak a szabványos eljárásait. Azután nem feledkezhetünk el a személyeket érintő adatbázisok esetében a személyiségi jog (angolul: privacy) védelméről, aminek kapcsán átfogóbban is gondolkoznunk

kell a hozzáférési jogok (angolul: access rights) kijelöléséről. (Néha - tévesen - az irodalom ezt az aspektust nevezi „biztonságnak”.) És akkor még mindig nem beszéltünk az adatok archív megőrzéséről, vagy a fejlett környezetekben szokásos - nálunk egyelőre még nem bevett - az adatbázis méretével, állapotával, használatával kapcsolatos státuszjelentésekről. Ezeket nyilván nem a végső-felhasználó, hanem az adatbázis egészét átlátó valaki készíti. Egyetlen adatbázis létrehozásába sem lenne szabad belefogni e ***menedzselési feladatok szabványosítása*** előtt.

Vegyük észre, hogy az adatbázis tartalmával - használhatóságával - kapcsolatos fenti kérdések egyike sem független az adatbázis szerkezetétől. Az adatbázist tele kell tűzködnünk technikai-menedzselési adatokkal. Ezek helyes meghatározásán és kezelésén rengeteg múlik. Azonban könyvünknek nem célja a technikai ismeretek kezelési művészetének a tárgyalása. Itt és most csak a menedzselési funkciók felismerésének és azok szabványosításának a szükségességére akartuk felhívni a figyelmet. Ezen funkciók legfontosabbikát a következő pontban ismertetjük.

10.7 Változásmenedzselés

Magyarországon az információs rendszerek fejlesztésére a fegyelem teljes hiánya a jellemző. A vezetők váratlanul, ad-hoc módon, a kapcsolódásokat át nem gondolva jelölik ki a „gépesítendő” funkciókat. Eszükbe sem jut, hogy az információs rendszer fejlesztése nem gépesítést jelent. Az ismeretek logikailag és időben meghatározott rend szerint függenek össze egymással. Ezért az alkalmazások fejlesztésének van egy célszerű, szinte kötelező erejű sorrendje. Ha valaki erre nincs tekintettel, akkor az a bemeneti és/vagy a kimeneti oldalon többletmunkára kényszeríti a felhasználót (ld. 8.7 pont). Az előbbi megállapítás a változtatásokra is vonatkozik. A változtatás beavatkozás a már nagy nehezen megteremtett rendbe. Ezért nem lenne szabad azt önkényesen, az összefüggésekre nem figyelve és azokat felborítva végrehajtani.

A felhasználókra is az ötletszerű változtatási igény a jellemző. Teljesen váratlan időpontokban kitalálnak egy-egy új bizonylatot; kicserélik a korábbi azonosítót; másféle kódok alkalmazására térnek át; a korábban X-karakteres név az új bizonylaton már Y karakter hosszú lesz stb. Senki nem törődik azzal, hogy mindez mennyi plusz papírmunkát jelent; milyen erőfeszítésbe kerül a változtatás átvezetése az adatbázison; megmarad-e vagy elveszik az egyéb alkalmazásokkal való összhang. A fejlesztő pedig időnként visszakéri a már üzemeltetésre átadott rendszerrészt és „jaj, valamit elfelejtettem” alapon abba gyorsan, elhamarkodottan belenyúlkál.

A hazai fejlesztésekre ez a kapkodás a jellemző. A bajok ott kezdődnek, hogy a szervezetek még az ***átadásmenedzselés*** lényegét sem ismerik. Nem alkotnak szabványokat arra nézve, hogy mikor, hogyan és milyen feltételek mellett tekinthető egy rendszer kvázi-késznek, üzemeltetésre átadhatónak. (Mint tudjuk, a rendszer soha sincs teljesen „kész”.) Nem ismert a ***tesztmenedzselés*** sem. A rendszert ad-hoc módon próbálgatják, mert a tesztelésre nincsenek szabványos eljárások. Az egyedi, az integrációs és az átadási teszt hármasáról pedig a legtöbben még csak nem is hallottak.

Most azonban nem a tesztelésről és átadásról van szó. Ezeket a menedzselendő, szabványokkal támogatandó eljárásokat csak a változásokkal való összefüggéseik miatt említettük. Ha senki sem tudja, hogy mikor tekintendő átadásra késznek egy rendszerrész, akkor persze azt sem lehet eldönteni, hogy egy változtatás hibajavítást, kisebb helyesbítést vagy kiegészítést, netán pedig nagyobb szerkezeti átalakítást jelent-e. Pedig a változás természete egyáltalán nem közömbös.

Üzemeltetésre már átadott rendszer esetében a javítás és kiegészítés az üzemeltető munkája. Viszont a szerkezeti átalakítás (angolul: restructuring) olyan feladat, amelyet végig kell vezetni az összes fejlesztési szakaszon. Mivel a kétféle tevékenységet más szervezet végzi, el kell tudni

dönteni, hogy a változtatásnak mi a hatása. Nálunk ezt a mérlegelést nem veszik komolyan, mert minden módosítást gyorsan akarnak végrehajtani. A következmények ismertek: egy dolgot javítanak - ötöt elrontanak. Pedig világos, hogy a változtatás hatását sem az üzemeltető, sem a fejlesztő, sem a felhasználó nem tudja igazán felmérni. Az „egy adatbázisban" minden ismeret minden másikkal összefügg. Ezért igen hamar kiderülhet, hogy az „aprónak" vélt átalakítás a végén többhónapos munkával jár.

A fentiek miatt minden vállalatban be kellene vezetni a ***változásmenedzselés szabványait***. Ezekben le kellene rögzíteni, hogy ki, milyen módon - milyen változáskérő bizonylaton - indíthat változtatást. Ki az, aki megvizsgálja a tervezett módosítás hatásait és eldönti, hogy a változtatás javítást, kisebb kiegészítést vagy strukturális átalakítást jelent-e. A szabvány leírja azt is, hogy a változás természetétől függően ki a felelős annak végrehajtásáért, az ismételt tesztelésért és adott esetben az újbóli átadásért.

A General Motors-ban történt esettel próbáljuk meggyőzni azokat, akik tiltakoznak a változtatás ilyen „nehézkes" módja ellen. Egy fejlesztő „apró" változtatást végzett egy bizonylaton. Ezzel vállalata több tízmillió dollárt takarított meg; kapott is szép summa prémiumot. Majd kiderült, hogy a módosítás átvezetése az adatokon, programokon, a felhasználói eligazításokon stb. kétszer annyiba került, mint maga a megtakarítás...

10.8 Szervezeti feltételek

Az előző pontokban csupa olyan tényezőről volt szó, amely az adatbázis egészét érinti és ezért nem tartozik a fejlesztés közismert szereplőinek a közvetlen hatáskörébe. A vezető elrendeli és betartatja a fejlesztővel és a felhasználóval a szabványokat, de azok kidolgozása és ellenőrzése nem a vezető, a felhasználó vagy a fejlesztő feladata. A fejlett informatikájú szervezetekben az adatbázis egészének a menedzselésével kapcsolatos sajátos funkciókat külön szervezeti egységek látják el. Az alábbiakban ezeket a nélkülözhetetlen menedzselési funkciókat fogjuk ismertetni. A hangsúlyt nem a külön egységre, hanem a funkcióra fektetjük. Kisebb szervezetben elképzelhető, hogy több funkciót is ugyanaz a szervezeti egység vagy személy lát el.

Adatbázist alkalmazó környezetben feltétlenül be kell vezetni az ***adatadminisztrátori*** (angolul: data administrator) funkciót. Ezt a feladatot nagyon sokan félreértik. Egyrészt nem tudják, hogy az angolban az „adminisztrálás" nem aktatologatást jelent, mint nálunk, hanem irányítást. Például a mai amerikai kormányt „Clinton-administration"-nek nevezik. Másrészt a fél-szakmai irodalomban tévesen használják az adat- illetve adatbázisadminisztrátor kifejezést: az adatkezelő rendszer bitjeivel-bájtjaival bíbelődő szoftverzsenit értenek rajta. A valóságban egészen másról van szó.

A szervezetekben minden erőforrásnak (ember, pénz, anyag, eszköz stb.) egy központi gazdája van. Az erőforrásokat mindenki felhasználhatja, de csakis a központilag szabályozott módon. Például az X-előadó nem adhat ki több milliót beruházásra. Az erőforrás gazdája az elsődleges felelős az erőforrás elosztásáért. Ezért nyilvántartást vezet magukról az erőforrásokról (vö. például anyagnyilvántartás) és azok felhasználásáról. A fejlett informatikájú szervezetekben rég tudják, hogy az adat - ***erőforrás***. Pénzbe kerül és felhasználása által pénzt hoz. Nem mindegy, hogy mi az ára és mennyit fial. Ezért kinevezik az adatadminisztrátort, a központi ***adatgazdát***.

Az adatgazdának számos feladata van. Ezek közül most csak a legfontosabbakat és röviden említhetjük. Az adatadminisztrátor nyilvántartást vezet az ismeretekről. Mint már tudjuk, ez a nyilvántartás nem más, mint a metaadatbázis (ld. 9. fejezet). Ezért az adatgazda átlátja, hogy az adatokat melyik felhasználó, milyen eszközön, milyen programmal kezeli. Ismeri az adatbázisok szerkezetét, a bizonylatokat, a kódokat, a be- és kimeneteket, a feldolgozásokat stb. Feladata ezen

tényezők harmonizálása, összhangjuk biztosítása azért, hogy ne legyen költséges átfedés vagy ellentmondás közöttük. Az adatadminisztrátor „moderátorként” is működik. Az adatbázist sokan használják. A felhasználók igényei ellentétesek, mert az egyiknek ez, a másiknak az a megoldás lenne kedvezőbb, mint ahogyan a pénz felhasználása is vitákat szülhet. Ilyenkor az adatadminisztrátor keresi azt a megegyezéses megoldást, amely a szervezet egésze számára a legkedvezőbb. Pontosan úgy, ahogyan például a főkönyvelő gazdálkodik a pénzzel.

A szervezetekben természetes ellentétek feszülnek az erőforrások használata körül. Ezeket célszerűen és okosan meg kell oldani. De ki alkalmas erre a feladatra? A vezető nem informatikai szakember. A fejlesztők és felhasználók érdekei pedig sohasem azonosak. E háromféle szereplő egyike sem képes átlátni a szervezet ismereteinek az együttesét. Mivel pedig erre az áttekintésre szükség van, meg kell teremteni az adatadminisztrátori funkciót.

A második feladatkör az előzővel rokon és feltételezi az azzal való szoros együttműködést. Az informatikai szabványok kidolgozásáról és ellenőrzéséről van szó. Ez nem a vezető, fejlesztő vagy felhasználó feladata. Ezért a fejlett szervezetekben külön ***szabvány-csoport*** működik. Ennek feladatai többrétűek. A fejlesztések megkezdése előtt meghatározza, nyilvántartja, közzéteszi az általános informatikai szabványokat és szabványos eljárásokat. A konkrét adatbázisrész előzetes tervezése - a modellezés - során az adatadminisztrátorral karöltve meghatározza a speciális adatszabványokat. A csoport a szabványos eljárásrészek letéteményese is. A jobb szervezetekben ugyanis felismerték, hogy a standard rutinokkal - például hiba-, menü-, validáláskezelés - óriási költségeket lehet megtakarítani. Végül a szabványrészleg ***minőségellenőrző funkciót*** is betölt. Ellenőrzi a tervezési folyamatot és a tervtermékeket abból a szempontból, hogy megfelelnek-e az előírt szabványoknak. Mert semmit sem érnek a szép szabványelképzelések, ha nem tartják be azokat.

Kisebb szervezetben az adatgazda és a szabványfelelős lehet ugyanaz az egység vagy személy is. Nagyobb szervezetben a két funkciót érdemes szétválasztani, mert az adatadminisztrátornak éppen elég gondja lesz az adatbázis korrekt strukturálásával.

Amint a korábbiak során láttuk, minden adatbázisnak két síkja van: a modell, a meta-adatbázis és a tényleges ismereteket tároló alkalmazási adatbázis. Az előbbiért az adatadminisztrátor felel, de ki az utóbbi gazdája? Nem a felhasználó, hiszen ugyanazt az adatbázist többen használják. Nem a fejlesztő, mert az üzemeltetésre való átadás után a felelősség már nem az övé. Nem is a szoros értelemben vett üzemeltető, amelynek a bemenetek fogadása, a futtatás és a kimenetek szétosztása - az adatbázis mindennapos használatának a biztosítása - a tulajdonképpeni feladata. Ha minden a meghatározott tervek szerint halad, akkor az üzemeltető nem felelhet például azért, hogy a felhasználók inkorrekt ismereteket visznek az adatbázisba.

A vitákat és problémákat elkerülendő a konkrét adatbázisnak kell, hogy legyen egy elsődleges felelőse. Ezt a funkciót nevezzük ***adatbázis- vagy rendszergazdának***. Nagyobb szervezetekben a két tevékenység szétválik. Az adatbázisgazda felelős az adatbázissal összefüggő menedzselési feladatokért. A kisebb módosításokért; az adatbázis integritását ellenőrző programok időszakos futtatásáért; a mentésekért és visszatöltésekért; a hozzáférési jogok kiadásáért/módosításáért; a személyi jogok védelméért; az archiválásért; a használati statisztikák készítéséért. Mindezek mellett ügyelnie kell a programmal nem ellenőrizhető bemenetek és kimenetek megfelelőségére. Végül vannak olyan ismeretigények, amelyek különleges programozási erőfeszítések nélkül is kielégíthetők például egy jelentéskészítő (angolul: report writer) segítségével. Az ad-hoc igények kiszolgálása - vagyis az adatbázis-segédletek alkalmazása - is az adatbázisgazda feladata. Tehát ez a funkció kimondottan ismeret- és alkalmazásorientált.

A mai adatbáziskezelő rendszerek igen összetettek. Az adatok fizikai helyének a kijelölése, az elhelyezésnek a hatékonyság fokozása érdekében vagy a méret változása miatti módosítása nem-triviális, speciális szakértelmet igénylő feladat. A hardvert időnként karbantartják, korszerűbbre cserélik. Ezért az adatbázist át kell plántálni másik gépre. Új szoftververziót bocsátanak ki és a megváltozott vagy újabb szoftverelemek cseréje illetve integrálása technikai átszerkesztéseket

tesz szükségessé. Az is előfordul, hogy az új kezelőváltozat kedvezőbb fizikai tárolási-hozzáférési megoldásokat kínál. Ekkor az adatbázis fizikai átszervezésére (angolul: reorganisation) kerülhet sor. A felsorolt feladatok miatt feltétlenül szükség van a szoftver- vagy (kezelő) rendszergazda funkciójára. Szemben az adatbázisgazdával, a rendszergazda kimondottan technika-orientált. Mivel igen ritkán fordul elő, hogy egy szakember éppen úgy ért az alkalmazásokhoz, mint az eszközökhöz, a nagyobb szervezetekben a két funkciót érdemes szétválasztani.

Az adat- és rendszergazda funkció kapcsán meg kell emlékeznünk a rossz hazai gyakorlatról. Nálunk igen sokszor a fejlesztők egyikét teszik meg rendszergazdának. Mivel a fejlesztés és a rendszerfelügyelet egyenként is teljes embert igényel, világos, hogy a hazai megoldásnak vagy a fejlesztés, vagy a működtetés látja a kárát.

Ezzel a szervezeti feltételek ismertetésének a végére értünk. Azonban a pontot nem zárhatjuk le két fontos, egymással összefüggő megjegyzés nélkül. Nálunk abban a tévedésben élnek, hogy akinek funkció adatott, ahhoz „ész” is jár. Ezzel szemben a gyakorlati helyzet az, hogy ma nagyon kevesen tudnák betölteni a fent ismertetett funkciókat. A szakemberek képzetlenek, és ezért nem igazán bíznak meg bennük. Emiatt a szóbanforgó kulcspozíciókat - ha azok egyáltalán léteznek - úgy töltik be, hogy nem adják meg a vonatkozó személyeknek a kellő jogkört. Ugyan ki veszi komolyan azt az adatadminisztrátort, aki a fejlesztési részleg másodrangú ügyintézője? Hogyan tud az rendelkezni, intézkedni? A felsorolt funkcióknak csak akkor van értelme, ha a feladat megfelelő jogkörrel - mondjuk ki: hatalommal, ranggal - párosul. A fejlett informatikai szervezetekben az adatadminisztrátornak minimum főosztályvezetői rang dukál...

11. AZ ÚT

11.1 Előrehozott zárszó

Barátaim, ismerőseim, tanítványaim és akikkel találkozom mind az adatbázistervezés „titkairól” faggatnak. Érdeklődésük egy bizonyos értelemben jogos. Az adatbázisok célszerű használatában, az adatbázison alapuló ismeretek megszerzésben 60 százalékos szerepet játszik az adatbázis jó megtervezése, a megfelelően kialakított adatmodell. Csak 30 százalékra teszem a valósághű adatok tárolásának a szerepét azért, mert a jó adatmodell eleve kikényszeríti a nagyrészt konzisztens ismeretbevitelt. Az adatbázissal kapcsolatos összes többi faktor legfeljebb 10 százalékban határozza meg az ismeretszerzés sikerét.

A fentiekkel szemben a mindennapi életben azt tapasztalom, hogy az adatbázis-projekt résztvevői legfeljebb 10 százalékban törődnek az adatmodellel. 20 százaléknyi figyelmet fordítanak a konzisztens adattárolására. Erőfeszítéseik 70 százaléka a körülményekre - a hardverre, a szoftverre, a fejlesztői és felhasználói lelkivilágokra illetve ellentétekre - nem pedig az adatszerkezetre (adatmodell) és az adattartalomra (adatbázis) irányul.

A legtöbben úgy érdeklődnek az adatbázistervezésről, hogy nem látják az adatmodell igen széleskörű összefüggéseit. A fejlesztő ***szintvakságban*** szenved, mert ő valójában nem fogalmi szintű adatmodellt akar. Hanem gyógyreceptet arra nézve, hogy miként is lehetne könnyen és gyorsan meghatározni egy sem logikai, sem fizikai, hanem valamiféle vegyes adatbázis-szerkezetet az ő kezelőjének megfelelően. Olyasmit akar, amit nem szabad, vagy legalábbis nincs önmagában értelme. A felhasználót ***nézetvakság*** kínozza. A fejlesztő nem foglalkozik bizonylatokkal és hasonló dolgokkal. A felhasználó viszont csak a saját - bemeneti és kimeneti - bizonylataiban gondolkozik. Eszébe sem jut, hogy nem a bizonylatokon alapul az adatbázis, hanem éppen megfordítva. Ezért születnek merev, párhuzamos, szűk célokat kiszolgáló adatbázisok. A fejlesztő és a felhasználó egyaránt nehezen fogadja el a valóban közös adatbázis gondolatát. Végül a vezető ***szerepvaksággal*** kínlódik. Nagy figyelmet szentel az eszközöknek, a feldolgozás és az adatbázis egyes részleteinek. Viszont nem eléggé törődik a valódi menedzselési funkciókkal. A szabványok kijelölésével; az adatstruktúra és az adatbázistartalom minőségének ellenőrzésével; a támogató funkciók (pl. adatadminisztrátor) megszervezésével stb.

Ennek a fejezetnek az a célja, hogy eddigi mondanivalónkat összegezve megmutassa az ismeret útját az adattól az adatbázison át az információig, és ezáltal ösztönözze az adatbázis-fejlesztés résztvevőit saját valós feladataiknak a felismerésében.

Ez a könyv nem az információs rendszerek fejlesztéséről általánosan szóló kiadvány. Ezért nem feladata a bizonylatok, kódok, bemenetek, kimenetek - az ismeretkezelésben oly fontos tényezők - tervezésének a kifejtése. Arra viszont rá kell mutatnunk, hogy az adatbázis önmagában nem fogható meg. Csakis az információs rendszer valamennyi tényezőjével összhangban képzelhető el. Ez a harmónia nemcsak szerkezeti, hanem időbeli összefüggéseket is feltételez. Ezért ebben a fejezetben az adat útját az információig mindkét szempontból nyomon fogjuk kísérni.

11.2 Fogalom születik

Bármennyire is meglepő, az adatbázis legfontosabb törvénye így hangzik:

Az a jó adatféle, ami nincs.

Ez a kitétel murisnak tűnik az első olvasásra. Azonban tessék csak meggondolni, hogy ha nem alkalmazzuk például a Cikk-rövid-megnevezés adatot, akkor azt nem kell rögzíteni, validálni (ellenőrizni egyediségét, a teljes megnevezésnek való megfelelést), tárolni, kezelni stb. Nem foglal helyet a papíron, nem kerül pénzbe és nem viszi az időnket.

Mindezt azért kell elmondanunk, mert egyes fejlesztők - talán saját fontosságukat igazolandó - azon fáradoznak, hogy minél több adatfélét gyömöszöljenek az adatbázisba. Nem egyszer hallani ilyen kitételt: „Jó lenne ide még felvenni azt az adatot ...". A tervezőket és az ügyintézőket szemmel láthatóan nem zavarja, hogy ez a plusz adat a valóban végső-felhasználónak - az állampolgárnak - mennyi fejtörést, utánjárást, kalamajkát okoz.

Az alkalmazóknak és a tervezőknek sokkal több önmérsékletet kellene tanúsítaniuk az új ismeretfélék kitalálásában és természetesen a meglévők átalakításában. Ezzel az elvvel szemben naponta azt tapasztaljuk, hogy már megint egy új formátumú és tartalmú számlát kapunk, pedig még az előzővel is alig barátkoztunk meg. A valódi problémák megoldása helyett sokan azzal bíbelődnek, hogy új bizonylatokat, rovatokat, kódokat, azonosítókat találnak ki és vezettetnek be a következmények átgondolása nélkül.

Pedig az ismeret születése hosszadalmas és fájdalmas. Nagyon sok gondosságot igényel e szülés levezetése. Példaként tételezzük fel, hogy az orvosi receptek ismereteit akarjuk számítógépes adatbázisban tárolni. Nézzük meg, hogy ebből az általános vágyunkból miként válik konkrét megoldás.

Szándékosan választottuk ezt a szélsőséges esetet, mert jól mutatja a kezdeti nehézségeket. Mert az hagyján, hogy az orvosi hieroglifákat az ördög tudja elolvasni. A baj ott kezdődik, hogy a recepten nincsenek „rovatok". (Illetve van egy-két ismeretdoboz, de általában azok létével és helyével nem szokás törődni.) A vény középső „rovata" a legtöbb esetben „természetes" - mármint az orvosnak természetes - nyelvű szöveg. „Minipress, 2*1, jelentkezni." „Minipr, reggel és este, mellékhatás esetén..." Nincs két orvos, aki ugyanúgy írná le ugyanazt az ismeretet.

Az adatbázis feltételezi az „adatszerű" kezelést. Ehhez pedig fogalmi modellt kell alkotni. Az adat valamilyen névvel ellátott „adatdoboz" tartalma lesz. Csakhogy éppen maguk a nevek, az ***alapfogalmak*** nem tisztázottak. Jó, a recepten felfedezhetjük a Gyógyszer-neve ismeretfélét, mire azonnal ránkförmedhetnek, hogy egyes vényeken nem gyógyszer szerepel, tehát az adatmegnevezés rossz. Azután mit jelent az, hogy „2*1" illetve „reggel és este"? Adagolás, dózis, használati utasítás? Szegény adat még meg sem született, máris sok bajunk van vele. A születés előtt kell tartanunk a keresztelőt. Névvel kell ellátni a leendő fogalmat. Olyan névvel, amely a tartalmat tévesztés-mentesen tükrözi.

Az ismeret születésének első momentuma a ***fogalomalkotás***. A „2*1" és a „reggel és este" ugyanazt mondja. De mit mond? A Magyar Televízióban végeztünk modellezést, amikor szembetalálkoztunk egy papíron a következő rovattal: „Színes vagy fekete-fehér?". A rovat mellett egy eligazító szöveg: „Színes = 1, Fekete-fehér = 2". Azonnal belépett a „minek nevezzelek" problémája. Mert színes/fekete-fehér lehet a készülék, a felvétel és az adás (színeset is lehet másként adni). A két érték tartozzon az Adástípus nevű rovatba? Ez a fogalom már mást jelentett. Adásfajta, adásszín stb. mind-mind rossz megnevezések. Amint ha Dózis nevet alkalmaznánk a receptnél, ötven orvos azonnal tiltakozna. A név árulná el a fogalom lényegét, de roppant nehéz jó fogalmi adatneveket találni.

A „keresztelő”, a névadás a fogalomalkotásnak csak az egyik oldala. Fel kell mérni a fogalom terjedelmét, értékkészletét is. Mit kell érteni azon, hogy a Gyógyszer-neve? A „Radipon” a gyógyszernév, vagy a „Radipon - 1.5”? A nevek nem mindig tükrözik az értékkészlet lehetséges tartalmát (vö. doméjn). Amikor a híd magasságáról van szó, valójában annak mélységéről - a vízhez legközelebbi pontjáról - beszélünk. Ezért a fogalomalkotás tényleg két dolgot jelent: a fogalom ***megnevezését és értékkészletének*** a definiálását.

A gyógyszernév („Radipon”) példájából azt is láthatjuk, hogy adott esetben az egy fogalomból többet kellhet generálnunk. Mert gondolkodhatunk úgy is, hogy a „Radipon - 1.5” a gyógyszer neve, meg úgy is, hogy a „Radipon” a medikámentum neve, az „1.5” pedig a Gyógyszerhatásfoka vagy valami hasonló elnevezésű másik tulajdonság.

Az információs rendszerek sikertelenségei vagy bosszantó hibái ott kezdődnek, amikor a tervező nem tisztázza kellőképpen a fogalmakat, illetve a saját világába zárt felhasználóra hallgatva elfeledkezik arról, hogy az adatbázis egy, felhasználói viszont többen is vannak. Mindez az ismeretek ***pragmatikai*** aspektusával (ld. 1.5 pont) függ össze.

Ha régi példánkra visszatérve ideírjuk a Kocsitípus nevet, akkor az olvasók mást és mást fognak érteni a név mögötti fogalmon. Az egyik nagy projektünkben történt, hogy az „A” felhasználó kocsitípusok szerinti kimutatást kért a járművekről. A mérnök úr teljesen használhatatlan tablókat kapott. Csak lassan derült ki, hogy a fejlesztő a „B” felhasználó - a főkönyvelő asszony - kocsitípus-fogalmát vette át. A nem műszaki, hanem pénzügyi szempontút. Nem volt eléggé körültekintő.

11.3 Több szem ...

A régi információs rendszerfejlesztési szakkönyvek igen nagy hangsúlyt fektettek a rendszerfelmérésre és azon belül is az „interjú-technikákra”. Nem akarjuk ezt a fontos megismerési módszert aláásni, viszont kételyeinket nem rejthetjük véka alá.

A személyes ***interjú*** nagyon hasznos annak feltárásában, hogy mit gondol az X megkérdezett személy a saját ismeretigényeiről. Természete miatt az interjú időigényes. A fejlesztőnek nem lesz ideje arra, hogy az adatbázis minden potenciális felhasználóját kikérdezze. Egy pár érdeklődés után abban a békés tudatban lát neki az adatbázis tervezésének, hogy ő már mindent lát. Közben csak a megkérdezettek véleményeit vette át. A többi felhasználó nézetét (ld. 5. fejezet) nem ismerheti. Beszélt az X osztályon Rózsikával, de az ugyanott dolgozó - és speciális ügyekkel foglalkozó - Julikát már nem hallgatta meg.

E könyv szerzője is beleesett ebbe a csapdába. Rózsika megesküdött, hogy minden szerződés csak egy ügyfélhez kötődik. Csak később derült ki, hogy vannak speciális család-biztosítások is, amelyek házaspárokra - tehát két személyre - vonatkoznak.

Miképpen lehet az ilyen kelepcéket elkerülni? Hiszen a fejlesztőnek nincs lehetősége - ideje - arra, hogy minden felhasználót végiglátogasson?

Nos a „huszonkettes csapdája” kivédhető a megfelelő módszerekkel. Minden új adatféleségről előzetes „születési anyakönyvi kivonatot” kell készíteni. A metaadatbázisban (ld. 9. fejezet) kell rögzíteni az új adatféle, mint fogalom nevét, megengedett értékkészletét és a szükség szerinti terjedelmű szöveges leírását. Ezt az együttest el kell küldeni minden érdekelt felhasználóhoz. Azokhoz is, akiket - idő hiányában - nem lehetett kikérdezni. Egy pillanat alatt fejünkre olvassák tévedéseinket, rámutatnak a kivételekre stb. Persze ha nincs reflexió, akkor a fejlesztő már valóban nyugodtan aludhat, mert ő megtette a magáét.

Több szem többet lát. Az viszont a fentiekből már világos, hogy az egyes adatfélék születését messze megelőzően kell általánosan felkészülni az eseményre. Ki kell alakítani a metaadatbázist.

Tömör és egyértelmű - homonima- és szinonimamentes - megnevezési konvenciókat kell bevezetni. Meg kell határozni azt, hogy milyen módon tudatják a felhasználókkal az új adat majdani születését és miképpen korrigálják az adatfélét még annak bevezetése előtt a felhasználói észrevételek alapján.

Az ismeret születésének a második momentuma az előzetes és széleskörű ***egyeztetés***. Ezt a „széleskörű" jelzőt sokan nem veszik komolyan. A tervezőt az X részleg bízta meg, tehát csakis annál érdeklődik arról, hogy mit is jelent a Befizetés. Hetek-hónapok múlva derül ki, hogy az Y részlegnél is létezik ez a fogalom, csak nüansznyi értelmezésbeli eltéréssel. A két részleg építhetett volna együtt egy jó adatbázist. Kialakíthatott volna jól használható közös bizonylatokat, adatbeviteli és -ellenőrzési, lekérdezési, könyvelési stb. eljárásokat. Olcsóbban, jobban, rendesebben. Mivel a fejlesztő és a felhasználó a fejlett technikával támogatott metaadatbázis használata esetében sem hajlandó vagy képes a valóban áttekintő egyeztetésre, az ismeretek születésénél elengedhetetlen a speciális szervezetek (ld. előző fejezet) bábáskodása.

11.4 Adat születik

A fogalomtól az adatig még hosszú út vezet. Tegyük fel, hogy a fogalomnak már van egyértelmű neve, jól meghatározott értékkészlete és kellőképpen tájékoztató leírása. A jó tervező még osztályozásokat is alkalmaz. Rámutat arra, hogy az ismeret alap vagy származtatott jellegű-e (ld. 8.1 pont). Az utóbbi esetben a leírás nyilván tartalmazza a származtatási tényezőket és magát az algoritmust. Az értékkészlet közretételében szerepelnek az érvényesítési (validálási) kritériumok is. A tervező nem feledkezik el arról sem, hogy a fogalom elemi vagy csoportos ismeretet (pl. dátum) tükröz-e.

Csakhogy a fogalmi szinten még vannak nem tisztázott részletek. Az egyik az általános (nem-fizikai) ábrázolással, a másik az ismeret szerepével kapcsolatos.

A „naponta kétszer-egy" mindig azt jelenti, hogy az orvosságot reggel és este kell bevenni? Ezt az ismeretet a „2" és az „1", a „reggel" és az „este" vagy az „R" és „E" párosaként ábrázoljuk? A Személy-neme csak nő és férfi értékű lehet. Ehhez nincs szükség különösebb fogalmi tisztázásra. Azonban a konkrét ismeret felveheti a „férfi-nő", „F-N" vagy az „1-2" alakokat is. Az elvileg tisztázott fogalmakat az ***értékek ábrázolásával*** kell párosítani ahhoz, hogy a fogalomból adattétel szülessen (vö. a D 4/3 definícióval).

Ilyenkor sokszor felmerül az a kérdés, hogy ***természetes*** vagy ***kódolt*** adatértékeket alkalmazzunk-e. A fogalmunk (Személy-neme) már létezik. Megengedett tartalmait is tudjuk - elvileg. Viszont a fogalomból akkor lesz adatszerű adat, ha tudásunk gyakorlati. Meg kell határoznunk, hogy milyen karaktersor vihető a Személy-neme adatba. Amíg ezt nem tesszük meg, addig van - feltehetően egyeztetett - fogalmunk, de nincs adatfélénk.

A fogalmak elvi ismeretek és ekként teljesen egyenrangúak. A Személy-azonosítója és a Személy-neme között nincs elvi különbség. A gyakorlati adatkezelési differencia annál nagyobb. Mielőtt megalkotnánk az adatbázis szerkezetét, már tudnunk kell, hogy egy ismeret természeténél fogva majd azonosítási vagy csak leírási szerepet (ld. 6.3 pont) fog ellátni. Egyáltalán milyen ***szerepre*** lesz majd alkalmas illetve milyent szánunk neki.

Ez a szerep többszörösen nem közömbös. Ha egy ismeretet ***azonosítási*** feladattal akarunk „megbízni", akkor gondoskodnunk kell értékeinek egyszerességéről, egyértelműségéről, közérthetőségéről és az utóbbi célt szolgáló feltűnőségéről. Az azonosítónak majd a bizonylatokon, a bevitelben és magában az adatbázisban is szembeötlőnek kell lennie. Ha viszont az ismeretnek csak ***leírási*** funkciót szánunk, akkor tudnunk kell, hogy mi lesz a várható azonosítója. Vegyük csak alapul korábbi gyógyszer-példánkat! Ha a „Radipon" név nem pontosan jelöl meg egy

gyógyszert, akkor leíró tulajdonság. Viszont ekkor mi magának a készítménynek az azonosítója? Lehet, hogy új tulajdonságot kell felvenni? Vagy a Gyógyszernév („Radipon”) és a Hatásfok („1.5”) együtt azonosítja a terméket?

Az ismeretnek négy dimenziója van (ld. 2.2 pont). Meg kell tudnunk jelölni a specifikus dolgot - a konkrét gyógyszert - amiről szó van. A fogalomból akkor lesz adat, ha a négy dimenziót világosan körvonalazzuk.

Ezért az ismeret születésének a harmadik momentuma a nem-fizikai ***ábrázolás*** és a leendő ***adatszerep*** meghatározása.

Mivel egy picit visszaéltünk az olvasó türelmével, készítünk egy gyorsösszegzést. A recepten mindenféle ismeretek találhatók. A „Radipon” és a „Bilagit” szövegeket olvasva, az adatbázis-tervező alkotott magában egy Gyógyszernév fogalmat (11.2 pont). Természetesen nem volt alkalma minden receptet végignézni. Születési anyakönyvi kivonatba vette, hogy a Gyógyszernév azt jelenti, hogy... Ezt az első megérzését közölte az érdekelt orvosokkal (11.3 pont). Mire ők megállapították, hogy a gyógyszernév ebben a formában egyáltalán nem kifejező, mert hiszen „Radipon” van ilyen, meg olyan. Tervezőnk végül megegyezik a felhasználókkal, hogy a Gyógyszernév („Radipon”) és a Hatásfok („1.5”) azonosítsa a készítményeket úgy, hogy nem alkalmaznak kódokat és - sajnos - a Hatásfok név nem igazán szerencsés. No de már mindenki tudja, hogy azon mit kell érteni. Így tehát a két fogalom - ha gyötrelmesen is - végül adattá vált.

11.5 Az ismeretek szerkesztése és első betöltése

A konkrét ismeretek adatbázisba vitele előtt meg kell határozni az adatbázis szerkezetét. Ha az adatbázisban nem szerepel a KOCSITÍPUS egyed, akkor a „Lada 1300” típusú kocsinak a férőhelyét nem ismerhetjük általánosan. A Férőhely tartalmát minden egyes kocsinál egyenként kellene megadni, ami felesleges munka, tárpazarlás és ellentmondásokhoz vezethet.

Az alkalmazási adatbázisnak az ***adatmodell*** a mintája. Az adatmodell tulajdonképpen nem más, mint egy összetett, struktúrákban és leírásokban meghatározott korlátrendszer. Ez határolja be, hogy milyen adatokat tudunk egymással összefüggésbe hozni közvetlenül vagy közvetett módon (navigálással - ld. D 8/4). Az adatbázison csak olyan nézet generálható, amely a globális struktúrából levezethető. Ezért a struktúra közvetetten meghatározza a ***kimeneti*** lehetőségeket. Ugyanakkor a korlátokon keresztül a szerkezet bizonyos mértékig behatárolja a ***bemeneti*** képeket is. Hiszen például addig nem tárolhatjuk a rendelések ismereteit, amíg nem ismerjük a vevőket.

Ezért az ismeretkezelés negyedik momentuma az ***adatmodellezés***. Ez adott esetben meglehetősen összetett feladat, amit éppen ezért külön, könyvünk második részében fejtünk ki. Itt csak a főbb aspektusokra hívjuk fel a figyelmet. Az adatmodellezés nem azt jelenti, hogy a kezelő korlátos képességeinek megfelelően összeállítunk amúgy érzés szerint pár rekordképet. Az ismeretek ***valósághű*** tükörképét kell megkomponálnunk. Ha a kezelő nem képes egy-egy korlát automatikus érvényesítésére, akkor arról magunknak kell gondoskodnunk, mert különben hamis ismeretek kerülnek az adatbázisba. Ezért az adatmodell nemcsak az adatbázis strukturális mintája, hanem egyben ***validálási tervként*** is szolgál.

A valóság hiteles tükrözésén túlmenően az adatmodellezésnek az a célja, hogy megtaláljuk az adatbázis ***optimális*** tervét. Az egyértelmű (homonimáktól és szinonimáktól mentes), teljes (a tökéletesen kapcsolható ismeretek tárolását biztosító) és minimális (redundanciákkal nem terhelt) szerkezetet. Ehhez adott esetben igen komoly matematikai és szemantikai elemzéseket kell végrehajtanunk.

Végül a szakkönyvek kevés szót ejtenek arról, hogy az adatmodellnek nemcsak statikus, hanem dinamikus aspektusa is van. Az adatbázisban tükrözött egyedtípusok ***időben*** is össze-

függenek egymással. Egymás precedensei (előzményei). Bizonyos egyedtípusok előfordulásait ismernünk kell ahhoz, hogy más egyedtípusok tartalmát kezelni tudjuk. Ezért nem ártana készíteni az adatmodell részeként egy ***precedenciatervet***, amely az egyedtípusok időbeli összefüggéseit rögzíti.

Itt már el is jutottunk az ismeret születésének az ötödik momentumához, az ún. ***adatkonverzióhoz***. Az összetettebb adatbázisok tényleges tartalmi feltöltése nem végső-felhasználói adatbevitellel kezdődik. A felhasználó csak igen ritka esetben kezd dolgozni üres adatbázissal. A legtöbb alkalommal vagy más számítógépes formátumból alakítják át (konvertálják) az adatokat vagy pedig gyorsbetöltő programokat készítenek. Az utóbbiak esetében az adatbevitel sokszor nem szép formátumú - tehát drága - bizonylatok alapján, hanem külön erre a célra készített konverziós bizonylatokról történik. Főleg a több másikkal is kapcsolódó „segédegyedeket" - irányítószám törzset, díjtáblázatokat, országlistákat stb. - célszerű az utóbbi módon bevinni, ha azok nem állnak rendelkezésre a gépen.

Az adatbázis első feltöltése ésszerűen a precedenciaterv szerint történik. Ez a biztosíték arra, hogy ne kelljen az első betöltés után a konzisztenciát helyreállító - a korlátokat érvényesítő - külön programokat futtatni. Ha például előbb vinnénk be a vevők ismereteit és csak utána a településekét, akkor utólag kellene ellenőrizni az irányítószámok helyességét a vevőcímekben.

Amint látjuk, a konverzió nem egyszerű feladat. Ha már az első feltöltésnél is rossz ismereteket engedünk az adatbázisba, akkor annak konzisztenciáját sohasem fogjuk tudni helyreállítani. Ezért a konverzió során rengeteg validáló programot kell futtatnunk. Az automatikusan nem validálható ismereteket - de csakis azokat! - konverziós listákon jelenítjük meg és „szemmel" egyeztettetjük, lehetőleg több személy által. Mindezek a feladatok csak akkor hajthatók végre sikeresen, ha nagyon tudatosan végezzük őket. Ennek pedig az a feltétele, hogy már az adatbázistervezéssel párhuzamosan elkezdjük összeállítani az ún. ***konverziós tervet***.

11.6 Bizonylatok

Az alkalmazási ismeretek általában először papírokon, ***alapbizonylatokon*** jelennek meg. Nálunk kialakult az a rossz szokás, hogy az adatbázis képét formálják a bizonylatokhoz, nem pedig megfordítva. Ebből adódik azután, hogy ugyanaz az ismeret, amely az adatbázisban csak egy egységet jelent, többféle papíron és sokszor eltérő formában tűnik fel.

E helytelen felfogás miatt a bizonylatok ***tartalmilag*** rosszul szerkesztettek. Tele vannak felesleges adatokkal. Például minden káreseménynél le kell írni a papírra, hogy hány köbcentis az X típusú kocsi motorja és mekkora az autó súlya, jóllehet ez az ismeret már az adatbázisban van és sohasem változik. Természetesen az A személy mást fog megadni a gépkocsi súlyaként, mint a B személy. Ha az ismeretet így az adatbázisba visszük, akkor abban hamis adatokat tárolunk. Ha validálással kiszűrjük, akkor feleslegesen végzünk egy ellenőrzést. Ha pedig nem visszük be, akkor minek azt papírra íratni?

A rossz bizonylat predesztinálja a helytelen adatbevitelt. A magyar bizonylatok közös jellemzője, hogy ***formailag*** is rosszul szerkesztettek. Tíz centiméteres helyet hagynak például a dátumra, hármat a névre. Világos, hogy az olvashatatlan lesz. A bizonylatokon igen rossz adatnevek szerepelnek és nagyon kevesen található értelmes kitöltési utasítás. Sem elrendezéssel, sem nyomdai megoldásokkal nem emelik ki a lényeges tételeket, például az azonosítókat.

Az ismeret születésének a hatodik momentuma a ***bizonylattervezés***. Ezt az aspektust ma teljesen elhanyagolják. A számítógépeseknek - érthetetlen okokból - a bizonylat kívül esik az érdeklődési körén. Az elsődleges adathordozókat laikusok - felhasználók - álmodják meg, holott a bizonylattervezés komoly tudást igénylő külön szakterület. Mi több, a bizonylatot előbb tervezik

meg, mint az adatbázist. Tehát a bizonylat megalkotója elvileg sem vehet részt a fogalom-, megnevezés-, értékkészlet-, ábrázolás-, méret- stb. egyeztetésekben, hiszen azokra csak később kerül sor. Ez természetesen predesztinálja a fogalmi, értékbeli stb. eltéréseket a bizonylat és az adatbázis között. Így fordulhat elő, hogy a bizonylatra a Személynév rovatba egyetlen kötőjel kerül. A mindig pontosan 24-karakteres Alvázszám rovat pedig az egyik bizonylaton 25-karakteresre sikerül. Ami azért mindegy, mert mindkettőn néha csak 10 karaktert töltenek ki...

Az ismeretkezelés alapvető szabálya, hogy a bizonylatokat csak az adatmodell megalkotása után szabad megtervezni. Ha pedig már létezik az elsődleges adathordozó, akkor azt az adatmodellnek megfelelően át kell alakítani. Az aktuális ismeret a bizonylat kitöltésével kezdi meg életét. Ha a kitöltés eleve hibákra késztet, akkor milyen lesz az adatbázis?

11.7 Adatbevitel

Elérkeztünk az ismeret születésének a hetedik momentumához, az ***adatrögzítéshez*** vagy adatbevitelhez. Ma már ezt a munkát nem annyira a külön hivatású adatrögzítők, hanem maguk a felhasználók végzik. Az adatbevitel képernyőformák segítségével történik. Sokan nem veszik észre, hogy a képernyőforma is „bizonylat". Ezért tervezésénél és kitöltésénél pontosan ugyanazokat a hibákat lehet elkövetni, amiket az előző pontban említettünk. Illetve még sokkal többet, mert az on-line kezelési eligazításokban is bőven akadnak bajok.

A képernyőforma újabb lehetőséget ad az egyértelműségi követelmények megsértésére. Ugyanannak az adatnak eddig csak két eltérő neve volt: „A" az adatbázisban és „B" - illetve sok különböző „B" - a bizonylatokon. A képernyőn - helyhiány miatt - már csak a sokkal rövidebb „C" név szerepel. Általában tervezőink nem villognak a képernyők szép, a korrekt bevitelt támogató formai elrendezésével sem.

Az adatbevitel lelke a tárolandó ismeret megfeleltetése az adatmodellben előírt korlátoknak. Az érvényesítés, vagyis a ***validálás***. A tapasztalatlan fejlesztők ezt a mozzanatot számítógépes programrészletekre redukáltan képzelik el. Holott kétféle validálás létezik: automatikus (programozható) és menedzselési. Az ügyfél címében lévő irányítószám az előbbi körbe tartozik, mert az előre tárolt irányítószám-törzs alapján validálható. Arról is tudunk gondoskodni, hogy az ügyfél nevét nagybetűvel kezdődően vigyük be. Ha származtatott adatot tárolunk az adatbázisban, akkor evidens, hogy nem követjük el azt a hibát, hogy az ismeret tartalmát a felhasználóval rögzíttetjük, hanem az adatbevitelkor automatikusan számítjuk/aktualizáljuk az értéket.

Vannak programmal nem validálható tényezők is. Ezeket ***adatszabványokkal*** (ld. 10.3 pont) kell megfogni, mert minden bevitt ismeretet validálni kell. Például az ügyfél címében mindig vagy ki kell írni az „utca" szót, vagy mindig az „u." rövidítést kell alkalmazni. A választás most mindegy; a lényeg az egyféle módon van. Nálunk képtelenek megfogni az ilyen adatok validálását, mert hiányzik az ahhoz szükséges akarat és fegyelem. A fejlesztő nem kutatja-keresi a megoldásokat, megmarad az egyszerű, szinte sablonos ellenőrzések mellett. A felhasználó fáradtság vagy egyéb ok miatt nem törekszik a pontos adatbevitelre.

A vezető pedig nem végez(tet) ad-hoc adatbeviteli felülvizsgálatot, amelynek során fény derülhetne az adatrögzítési hiányosságok típusaira és okaira.

A fentiek miatt mondtuk, hogy a validálás második fajtája menedzselési jellegű. Ha a vezetőnek valóban nincs ideje az ellenőrzésre vagy például új alkalmazás kezdetén vagyunk és még a lelkiismeretes adatrögzítők sem tudtak mindent betanulni, akkor élni kell a rég bevált ***kétszeres adatrögzítés*** technikájával, ami szintén menedzselési megoldás.

Az adatbevitellel kapcsolatosan még egy általános hibára kell felhívni a figyelmet. Nálunk elterjedt az a rossz nézet is, hogy minden adatot a számítógépre kell vinni. Azt is, amely szemmel

láthatóan helytelen, nem-teljes, ellentmondó stb. Az ügyfél ugyan nem adta meg a címét, de azért az egyéb adatokat csak bevisszük!

Mindenkit óva intünk ettől a szemlélettől. Különösen azon ismeretek esetében, amelyek valamilyen módon a jelenség azonosításában részt vesznek. Ha az ügyfél újra jelentkezik, akkor nem a régi - a rosszul bevitt - ismereteit fogjuk korrigálni, hanem új tételt nyitunk az ügyfél részére - hiszen a régit már nem is tudjuk azonosítani. Így lesz egy pillanat alatt redundáns és inkonzisztens az adatbázisunk. Utólag próbálkozunk az egyeztetéssel, holott azt megtehettük volna az első helytelen adatbevitel előtt is.

11.8 „Új" ismeret születik

Az ismeretszerzés nyolcadik momentumához érkeztünk. A 8.4 pontban már kifejtettük az ***adatkezelés és az adatfeldolgozás*** különbségét illetve viszonyát. Most röviden összegezzük azt, hogy e két műveletfélével miként tehetünk szert „új" ismeretre. Mielőtt ezt megtennénk, felhívjuk a figyelmet arra, hogy az „új" jelzőt objektív és szubjektív szempontból kell majd mérlegelnünk.

Az ismeretszerzés legalapvetőbb módja a közvetlen észlelés vagy közlés. Ránézünk a kocsira: jé, ez piros színű. Vagy Marika közli velünk, hogy az X kocsi piros. Az adatbázisban bogarászhatunk a legegyszerűbb, pusztán megtekintést szolgáló adatkezelési műveletekkel és „új" ismeretre tehetünk szert. Eddig nem sejtettük, hogy Kovács Lajosnak piros kocsija van, most már tudjuk. „Ránézünk" az adatbázisra, az adatbázis „közli" velünk ezt az ismeretet. Amely objektíven nézve egyáltalán nem új, hiszen már régen az adatbázisban tárolódott. Csak szubjektíven - vagyis a mi számunkra - új.

Kevesen veszik észre a közös adatbázisnak ezt a „lexikon" szerepét. A lexikonba X és Y is ír ismereteket - egymástól függetlenül. Ezek azonnal közös kinccsé válnak. Mert ha fellapozzák a lexikont, akkor X és Y megtalálja benne a saját tételeit is, de a másik partnerét is. Azt az ismeretet, amelynek addig nem volt birtokában. Sőt, van aki egyáltalán nem tesz ismeretet ebbe a „lexikonba", mégis mindent megtudhat belőle. A ***tallózó adatkezelés*** segítségével így tágul ki a „közvetlen észlelés vagy közlés" értelme.

Persze az adatbázis használatára nem a véletlen tallózás jellemző, mint ahogyan a lexikont sem az elejéről kezdjük olvasni. Itt ismét figyelmeztetnünk kell az adatbázis tartalmi (intenzionális) és kiterjedési (extenzionális) aspektusaira. Arra, hogy az adatbázis egyetlen nagy táblaként képzelhető el, amelyben vannak tulajdonságtípusok és egyedelőfordulások. Az adatbázis „olvasója" adott esetben a jelenségeknek csak néhány sajátosságára kíváncsi (pl. az ügyfél címére igen, de nemére nem). Máskor az ügyfelek minden tulajdonságtípusában érdekelt, de csak meghatározott egyedelőfordulások esetében (pl. a budapesti ügyfelekre kíváncsi). A két aspektus egymással összefügghet, hiszen amikor a hölgy ügyfeleket keressük, akkor az intenzió (Ügyfélnem = „Nő") alapján határoljuk be az extenziót (nem a teljes terjedelemben vagyunk érdekeltek).

Az adatbázisban tárolt, objektíven nézve egyáltalán nem új adatokból ***szelektív adatkezeléssel*** tehetünk szert szubjektíven új ismeretekre. A tallózással véletlenül szerzett ismeretekkel szemben itt tudatos keresésről van szó. A fejlesztőknek az lenne az alapvető feladata, hogy feltárják az egyes felhasználók személyes intenzionális és extenzionális adatkeresési igényeit, és azoknak megfelelő nézeteket illetve kimeneteket tervezzenek.

Az adatkezelő rendszerekkel az objektíven „régi", vagyis már tárolt adatokat mindenféle kombinációban elő lehet bogarászni a szubjektíven „új" ismeretek szolgáltatása érdekében. Feltéve, hogy az adatbázis szerkezete megfelelő és a tárolt adatok egyszeresek, nem-hiányosak, a valóságot hűen tükrözik. Ezért a hatékonyság - egyáltalán nem mellékes, viszont nem az

informatika szférájába tartozó - momentumától eltekintve valóban kijelenthetjük, hogy nem az adatkezelés mikéntje a fontos, hanem az, hogy mit kezelünk.

A származtatott ismereteket csak ritka esetben tároljuk az adatbázisban. Ha ezt tesszük, akkor a származtatott adat alapadatként viselkedik, vagyis tisztán adatkezelési művelettel nyerhető vissza. Más a helyzet akkor, ha az objektíven „régi" - már tárolt - adatokon matematikai és/vagy logikai műveletekkel kívánunk szubjektíven „új" ismeretre szert tenni. Ha pl. kíváncsiak vagyunk a Tételértékre, ami a Rendelt-mennyiség és az Egységár szorzata.

Vegyük észre, hogy a Tételérték objektíven nem „új" ismeret. Ha az adatbázisban nem is tároljuk ezt az adatot, az adatmodellben akkor is le kellett írjuk a Tételérték számítási képletét. A származtatott tulajdonságok tartalmai mindig csak szubjektíven jelentenek „új" ismeretet. Ahhoz, hogy ezeket megismertessük a felhasználóval, ***adatfeldolgozási műveleteket*** kell végrehajtanunk. Vagyis aktuálisan össze kell szoroznunk a Rendelt-mennyiség és az Egységár értékét, miután ennek a két tényezőnek a tartalmát kikeressük (adatkezelési művelettel).

Az adatfeldolgozás - gyerekjáték. Mert mikor lehet sikertelen a feldolgozás? Három esetben. Akkor, ha rossz az adatbázis szerkezete (a rendeléstételek és cikkek nincsenek kapcsolatban). Akkor, ha az adatbázisban rossz adatokat tárolunk (nem is annyi a cikk ára, vagy nem is annyit rendeltek belőle). No meg akkor, ha a felhasználó tévesen adta meg a származtatás lényegét (nem az egységárat, hanem a kialkudott árat kellene használni).

A fejlesztő a három feltétel közül csak az első kettőért felelős. Persze azért is, hogy a felhasználó által megfogalmazott származtatást jól vezesse le. Viszont azért, hogy a származtatási tényezőket és a származtatás módját helyesen határozták-e meg, csakis a felhasználó felelhet. Hiszen ő tudja, hogy szubjektíven milyen ismereteket akar látni.

Sajnos néha a felhasználók képtelenek kifejteni ismeretigényeiket. Leginkább ennek a ténynek tudható be, hogy a természetes adatfeldolgozási láncot (ld. 8.7 pont) még a tájékozott és jószándékú fejlesztő is kénytelen a végén megtörni és személytelen, nem végigvitt feldolgozású tömegkimeneteket produkálni a felhasználó számára. Ezzel pedig kis ciklusunk bezáródik, mert nem marad más megoldás, mint a gépi vagy kézi tallózás.

11.9 Adatkimenet és behangolás

A „leválogatásnak" is nevezett szelektív adatkezelés illetve a származtatási műveleteket felölelő adatfeldolgozás eredményei képernyős és/vagy nyomtatott formájú kimeneteken jelennek meg. Ha az ismeretszerzés végtelen ciklusát nem ismernénk - a kimenet megtekintése után új igények keletkeznek és az ebben a fejezetben leírt körforgás elölről kezdődik, - akkor azt mondanánk, hogy az ***adatkimenet*** az ismeretszerzés utolsó momentuma.

Igen, még mindig „ismeretszerzésről" kell beszélnünk. Kifejtettük az ismeretszerzés észlelési, érzékelési stb. aspektusait. Kimondtuk, hogy mi csak az írott nyelv szerinti ismeretekkel foglalkozunk. A „Fehér a Rózsa kocsija" és a „Rózsa kocsija fehér" példákkal arról akartuk meggyőzni az olvasót, hogy a két közlés azonos lényeget tartalmaz, a hangsúly pedig nem tükrözhető magában az adatbázisban.

A kimenet a bizonylat egyik speciális fajtája. Abban az értelemben, hogy nemcsak általános tartalmával, hanem formai aspektusaival is törődnünk kell. Jó kimenetet tervezni nem kisebb feladat, mint jó bizonylatot alkotni. Sőt, néha nehezebb.

A kimenet címe, a kijelzett ismeretek egyértelmű neve mellett ügyelnünk kell a „vertikális" (tartalmi) és a „horizontális" (terjedelmi) elrendezésre is. A tulajdonságtípusok és az egyedelőfordulások sorrendjére. Mert - szemben az ebből a szempontból semleges bemenetekkel - a

kimeneteken maga az ***elrendezés*** is szubjektíven új ismereteket tükrözhet. Például úgy, hogy az első sorban jelenik meg a legtöbbel tartozó vevőnk.

A kimenetekben sokkal többet kellene törődnünk a kimondottan formális aspektusokkal. A helybeosztásokkal, a betűszedetekkel és az egyéb figyelemfelkeltő eszközökkel. Láthatunk olyan felmérési kimutatásokat, amelyek nem százalékok szerinti sorrendben listázzák fel a tételeket; nem is emelik ki kövér szedettel a kiemelkedő százalékokat; ráadásul a lényegtelen tényező részletezésére öt, a lényegesére csak egy sort tartanak fenn, merthogy a tördelés úgy adódik.

Az adatbázis - formasemleges. Hangsúlyok nélkül, egyszeresen kell rögzítenie az objektív tényeket. Nem lehet előre tudni, hogy ki milyen szubjektív ismeretek birtokába akar jutni az adatbázis használata által. Ha ez előre fixálva volna, akkor nem is beszélhetnénk általánosított adatbázisról. Az adatbázis felhasználásakor a megjelenítendő ismeret nem független a formától. A lényegeset és a lényegtelent alakilag is el kell választanunk egymástól, ha ezzel teljesíthetjük ki alapvető feladatunkat. Aminek a lényege az, hogy támogatjuk a felhasználót az ismeretek információvá történő értelmezésében.

Még egy tényezőt kell megemlítenünk. Bonyolultabb adatbázisok esetében nem kezdjük meg azonnal azok használatát. Egy erre a célra kijelölt felhasználói csoporttal elvégeztetjük a legtipikusabb adatkezelési műveleteket. Főleg a tömegszerűeket, amelyek az adatbázis ***robusztusságát*** tesztelik illetve azokat, amelyek kritikus válaszidőket igényelnek, tehát a ***hatékonyság*** próbájára alkalmasak.

A fejlesztők sokszor elfeledkeznek erről a tesztről. Egyedi programokat, programcsoportokat tesztelnek - ez is fontos -, de nem vizsgálják az adatbázis teljesítőképességét. Így az éles használat során lesz szükség az adatbázis átalakítására, amely sokkal több bajjal jár. Ezért az okos fejlesztők a tényleges használatbavétel előtt „megrázzák” egy kicsit az adatbázist. A nyert tapasztalatok alapján az éles igényekhez igazítják, azaz ***behangolják*** (angolul: tuning).

11.10 A hosszú és kanyargós út

Nem tudok a leendő adatbázistervezők számára könnyű munkát ígérni. Hosszú és kanyargós út vezet az adattól az adatbázison át az információig. Egy minipéldán keresztül foglaljuk össze ennek az utazásnak az állomásait. Tegyük fel, hogy a felhasználó a nagybani vevők olyan rendeléstételeinek az értékeire kíváncsi, amelyek az X cikkszámú cikkre vonatkoznak. Azért, hogy a teljes utat átlássuk, tételezzük fel, hogy nulláról kell indulni. Vagyis nincs semmi háttértudásunk. Tekintsük át, hogy mik is lesznek a teendőink.

ad 1) Tisztázni kell, hogy mit jelentenek az alapfogalmak. A „nagybani” vevő, egyáltalán a vevő, a rendelés, a rendeléstétel és a cikk, mint ismerettel leírandó jelenségek (egyedek). Mivel ezeket majd kezelni kell, fel kell tárni azonosító tulajdonságaikat. Meg kell tudni, hogy milyen ismereteket kell a vevőre, rendelésre stb. vonatkozóan megjeleníteni a tételértéken túlmenően. Persze azt is ki kell deríteni, hogy miként számítják a tételértéket és tényezőinek (ár és mennyiség) mi a tartalma. Az ár egységár-e, ami a cikket jellemzi, vagy tételenként eltérhet az értéke, mert például kialkudott ár. A fogalmaknak nevet kell adni és meg kell határozni előfordulás- illetve értékhalmazukat (11.2 pont).

ad 2) Mindezt előzetesen leírjuk a metaadatbázisban. Majd egyeztetünk minden olyan felhasználóval, aki eladással foglalkozik. Eladhatunk autókat is és azok alkatrészeit is. Az egyik kereskedő részére a „nagybani” teljesen mást jelenthet, mint a másiknak. Korrigáljuk metaadatbázisunkat a beérkező információk szerint (11.3 pont).

ad 3) Pontosítjuk fogalmaink ábrázolását, vagyis azokból adatokat alkotunk. Mindig ugyanúgy ábrázolják a rendelésszámot? Vagy netán több helyen alkalmaznak ilyen adatot eltérő ábrázolással? Melyik a helyes? Új közöset kell alkotni (11.4 pont)?

ad 4) Meg kell terveznünk az adatmodellt. Majd azt kell vizsgálnunk, hogy annak mely tényezői általános használatúak. Úgy, hogy azokat nem minden bevitelkor külön fogják beütni, hanem előre összeállítható az egyedtípus tartalma. Ha például a mértékegységet is meg kell jeleníteni, létezik-e már a számítógépen a megengedett értékek készlete és az egyértelmű-e? Ha nem, ki fogja azt nekünk átadni? Mi a konverziós teendő (11.5 pont)?

ad 5) A feladat által érintett adatok milyen meglévő bizonylatokon jelennek meg? Nevük, ábrázolásuk, megengedett értékkészletük megegyezik-e az általunk feltételezettel? Ha valamelyik tényezőben eltérés van, akkor - feltételezve azt, hogy az előző lépésekben gondosan jártunk el - át kell alakítani a helytelen bizonylatokat (11.6 pont).

ad 6) Meg kell terveznünk minden egyes beviendő adat validálását. El kell döntenünk, hogy melyik adatféleség validálható programmal, melyikre kell szabványt készítenünk. Ezt a szabványt meg is kell alkotnunk (11.7 pont).

ad 7) Összeállítjuk az adatkezelés és az adatfeldolgozás együttes tervét. Ez azt jelenti, hogy a vonatkozó lekérdezésre meghatározzuk a szelekció módját, az egyedek közötti navigálást és azt, hogy az egyedek kezelése közben mikor hajtjuk végre a származtatási műveleteket (11.8 pont).

ad 8) Prototípus kimeneteket készítünk. Egyeztetjük a felhasználóval a kimenet végleges tartalmát és formáját. A kimeneteket az igényeknek megfelelően csinosítjuk (11.9 pont).

Természetesen a felsorolt tevékenységeknek nem mindegyike tartozik a szoros értelemben vett adatbázistervezés feladatkörébe. Azonban az adatbázistervezőnek közreműködőként részt kell vennie valamennyi fejlesztési részfeladatban. Hiszen az adatbázis sikere nemcsak a korrekt terven, hanem a megfelelő használaton is múlik.

Ezzel az adatbázis kialakítására vonatkozó alapvető tudnivalók ismertetését lezárjuk és rátérünk az adatbázistervezés titkainak a feltárására.

II. RÉSZ

Az adatbázistervezés titkai

BEVEZETÉS - II.

Az adatbázistervezésnek számos titka van. Ha nem lenne, akkor mindig tökéletes vagy majdnem jó adatbázisokkal találkozhatnánk. Ezzel szemben a szomorú gyakorlat az, hogy nehezen bukkanunk egy akár csak közelítőleg elfogadható adatmodellre is. Vajon mi lehet ennek az oka? Valóban ennyire képtelenek lennénk jó adatbázisok kreálására?

Jól tervezni csak az tud, aki tisztában van a terv lehetséges általános hibáival és azok elkerülhetetlen következményeivel. A 12. fejezetben ismertetjük az ***adatmodellek tipikus hibáit***. Az adott részben leírt problémákat a mai fejlesztők és felhasználók nagy része nem veszi komolyan. Ugyan miért baj az, hogy két eltérő jelenségnek azonos nevet adnak vagy ugyanazt itt meg ott másképpen nevezik? Többszörösen tárolunk egy ismeretet? Üsse kő; elég nagy a számítógép. Ha a tervező nem ismeri fel a nagyon komoly másodlagos hatásokat, akkor szakmailag nem alkalmas a munkájára. Ha pedig tud a hibákról és mégis rossz tervet alkot, akkor erkölcsileg nem az.

Az adatbázis tervében előfordulhatnak olyan hiányosságok, amelyek némi kis utánjárással kiküszöbölhetők. A 13. fejezet mutatja be a tulajdonságok alapvető függéseit és a ***normalizálás alapjait***. A tervező által „első heves felindulásban" megalkotott kezdeti táblaképek önrevíziójának a matematikai-technikai hátterét ismerteti ez a rész, amely a következő pár fejezetet is megalapozza.

Az adatbázisok egyik leggyakoribb - noha egyáltalán nem a legsúlyosabb - hibája a felesleges adattöbszörözés. A redundancia, amely karbantartási visszásságokhoz, az adatbázis tartalmának az ellentmondásához (inkonzisztenciához) vezet. A 14. fejezet tárja fel az ***alapvető normálformákat***. Olyan táblaszerkezet-ellenőrzési és -átalakítási (normalizálási), ezen a szinten még többnyire mechanikus tervezési megoldásokról van szó, amelyek nagyon könnyen elsajátíthatók és segítenek elkerülni a legdurvább redundancia problémákat.

Még a 15. fejezet is elsősorban a feleslegesen többszörös adattárolásról szól. Rejtetten rafinált redundanciákról van szó, amelyek kiküszöbölése, a ***magasabb normálformák*** megkeresése már komoly nehézségeket okozhat. Ráadásul szembe kell néznünk a tárolási és a kezelési hatékonyság örök dilemmájával is. A tárolási szempontból kedvezőbb alakra hozott tábláink kezelése nehézkesebbé válhat.

A valódi tervezési problémákat a 16. fejezetben kezdjük el feszegetni. Egyes kezelők nem engedik meg a ***csoportok*** explicit meghatározását. Emiatt a tervezők már az elemzés szakaszában sem figyelnek az összetett tulajdonságok, a csoportok implicit alkalmazásából fakadó igen komoly problémákra. A fejezetben nemcsak ezeket a gondokat ecseteljük, hanem arra is rámutatunk, hogy azok feloldása csak szemantikai modellezéssel, a jelenségek és összefüggéseik alapos átgondolásával lehetséges.

Nem csak a csoportok modellezésének a hiánya vezet oda, hogy a hagyományos ***normalizálási eljárások*** csődöt mondottak. A széleskörűen, a mai „modern" tervezőrendszerek által alkalmazott normálforma-dekompozíció és -szintézis módszerei elméletileg hibásak. Ezért a 17. fejezetben olyan szemantikus normalizálási eljárást fogunk bemutatni, amely kiküszöböli a mai tervezési módszerek elvi és gyakorlati hiányosságait.

A mai tervezők legnagyobb hibája, hogy nem kellőképpen minősítik az adatbázisterv tényezőit. Ezt azért nem teszik, mert a kezelő úgymond úgysem képes az osztályozás automatikus validálására. Hiába is írja le a tervező a modellben, hogy az A egyedtípus családfa jellegű, ami azzal meg azzal jár, hogy..., ha az adatkezelő nem ismeri az ilyen fajtájú egyedet. Ez a felfogásmód téves. Ha a rendszer nem érvényesíti automatikusan a korlátokat, akkor nekünk kell azt megtennünk. De miképpen lennénk képesek erre, ha az adatmodellben nem határozzuk meg az

adatbázisterv tényezőinek a speciális jellegét? Ezért a 18. fejezetben az ***adatmodell speciális szerkezeteire*** hívjuk fel a figyelmet.

A következő, a 19. fejezet összefoglaló jellegű. A könyv második részének bevezető, 12. fejezetét mintegy kiegészítve a ***tipikus tervezési hibákat*** összegzi. Ez a szakasz arról is szól, hogy miképpen nem szabad és miként célszerű élni a tervezési sablonokkal.

A nagyon rövid 20. fejezet egy tervezési minipéldát mutat be. Ezzel két célunk van. Egyrészt ismertetni akarjuk az ***elemzés*** mindenkor szükséges, elemi lépéseit. Másrészt az adatmodell helyes ***dokumentálásának*** a módját szeretnénk felvázolni.

A könyv a 21. fejezet méretében kicsi, de ahhoz képest meglehetősen komplikált modellezési ***esettanulmányával*** zárul.

Most beszéljünk még egy keveset arról, hogy ***mit nem árul el ez a könyv***? Két dolgot. Az adatmodellezésnek vannak személyes apró, nehezen átadható titkai. Nagyon komplikált lenne azt elmondani, hogy a szerző miképpen következtet egyes alkalmazandó megoldásokra vagy éppen kerülendő hibákra. Egy ábrában miként fedezi fel a generalizáció vagy éppen a specializáció szükségességét. Egy leírásból miképpen következtet az egyed-, tulajdonság- és kapcsolattípusokra. Sőt, hogyan látja meg ezeket az összefüggéseket egyetlen kódolt csoportos adatban is. Egy bizonylat struktúrája alapján miként sejlik fel a számára a vonatkozó modellrészlet. Az orvosi diagnosztikát lehet általában tanítani; viszont a sajátos megérzések, amelyek alapján egy doktor jó diagnoszta lesz, nem adhatók át.

Vannak olyan nehéz-fajsúlyú titkok is, amelyekkel az író nem akarja terhelni az olvasót - és segíteni a konkurenciát. Az automatikus adatbázistervező eszközök belső lelkivilágáról és alkalmazásáról van szó. Bár maguknak az ilyen segédleteknek a tervezéséről rengeteget tudnánk mesélni, ezek a titkok nem igazán szolgálnának az olvasó üdvére. Alkalmazásuk részleteiről pedig lehetetlen és felesleges beszélni, ha nem áruljuk el a szoftver lényegét. E megjegyzéseket pedig nem bosszantásnak, hanem tájékoztatásnak szántuk. Az adatmodellezésnek vannak a profik által el nem árulandó titkai is.

A jó adatbázistervező számítógépes szoftversegédlet nagy kincset ér. De nem abban rejlik az adatmodellezés legfőbb titka. Hanem a ***szemléletben***. Nemrégiben alkalmunk volt részt venni egy „mutogatós", „önigazolós" tanácskozáson. A felhasználó a fejlesztőre mutogatott, az pedig vissza. Közös önigazolási vágyuk oda vezetett, hogy a fejlesztés harmadik résztvevője, a vezető is mentegetődzni kezdett. Nem erről a - sajnos - szokásos körképről akartunk itt szólni.

A felhasználó az adatbázis hibáiért az „informatikusokat" vádolta. Az „informatikusok" védekeztek, a vezető pedig egyensúlyozni óhajtott a „felhasználók és az informatikusok" között. Nos, itt van az alapvető baj! A három résztvevő partinak - felhasználó, fejlesztő és vezető - fogalma sem volt arról, hogy mi is az az ***informatika***. Azt következetesen összetévesztették a számítástechnikával, a számítógépes adatbázisokkal.

Az informatika az ismeretek tudománya. Ma már elválaszthatatlanul kapcsolódik a ***számítástechnikához***, de nem azonos azzal. Adatbázisaink legfőképpen attól rosszak, hogy számítástechnikai - és nem informatikai - szemléletben készülnek. Az ismeret számítógépen kívüli és belüli életét egymástól mesterségesen elválasztják. Pedig az ismeret egy, akár számítógépen, akár azon kívül kezeljük. Ezért a jó adatbázistervezés egyik legfőbb titka az ismeret kettős életének a felismerése és összehangolása. Annak elismerése, hogy jó adatbázis csak a felhasználó, a fejlesztő és a vezető korrekt együttműködésével tervezhető.

12. AZ ADATMODELLEK HIBÁI

12.1 Az adatbázis hibáinak a forrásai

A kérdés nem úgy hangzik, hogy miért nem nyerünk megfelelő ismereteket adatbázisainkból. Ha ez lenne a felvetés, akkor ki kellene térnünk az ***adatbáziskezelés*** momentumaira is. Hiszen a nem megfelelő kezelő vagy a jó kezelőrendszer ügyetlen használata is okozhatja azt, hogy ismeretéhségünk nem csillapodik. Azonban ebben a könyvben nem kívánunk a konkrét adatkezeléssel foglalkozni. Ezért kimondottan csak arra vagyunk kíváncsiak, hogy mi okozhatja az adatbázis által nyújtott szolgáltatások hiányosságait akkor, ha kezeléstechnikai problémákat nem feltételezünk.

Az adatbázis mindig kettős rendszer. A konkrét ismeretek alkotják az ***alkalmazási adatbázist***, amely egyedelőfordulások, tulajdonságértékek és kapcsolat-előfordulások szervezett együttese. Az alkalmazási adatbázis az ***adatmodell*** szerint elrendezett, amely viszont egyed-, tulajdonság- és kapcsolattípusok szervezett együttese.

E kettősségből következik, hogy az adatbázis két ok miatt lehet hibás. Előfordulhat, hogy az adatmodell jó, de a konkrét adatbázis rossz a helytelen adatbevitel, -módosítás és egyéb műveletek következtében. Most nem az adatkezelés mikéntjéről beszélünk, hanem egyszerűen csak arról, hogy az X adat helyett az Y ismeretet viszik be. Az adatbázis attól is lehet inkorrekt, hogy tévesen határozták meg általános szerkezetét, az adatmodellt.

A két hiba nem azonos súlyú. Az adattévesztés egyszeres, nem hat ki a teljes adatbázisra. Ezzel szemben a rossz adatszerkezet szükségszerűen maga után vonja azt, hogy a hibák továbbterjedjenek az egész adatbázisban, vagyis a hiba megtöbbszöröződjön. Nem szabad arról sem elfeledkezni, hogy az adatmodell nemcsak a ***struktúrát***, hanem a ***korlátokat*** is felöleli. A hibás adatbevitel egyik leggyakoribb oka az, hogy ezeket a megkötéseket vagy nem határozzák meg eléggé pontosan az adatbázis tervében, vagy elfelejtik érvényesíteni azokat a programokban.

Amint látjuk az adatbázis alapvetően attól lehet rossz, hogy rosszul alakítják ki annak általános képét, mintáját, az adatmodellt.

Ennek a fejezetnek az a célja, hogy bemutassa az adatmodell általános, tipikus hibáit, rávilágítson e bajoknak a tervezési szemléletben rejtőző okaira és ezek tükrében összegezze az adatmodellezés céljait.

Mielőtt a lényegi mondanivalóba vágnánk, összefoglaljuk az adatmodell leglényegesebb tényezőit. Az ismétlő áttekintés után ismertetni fogjuk ábráink és példáink formai konvencióit, vagyis a továbbiakban alkalmazott egyezményes jelöléseket.

A hibák részletezése és következményeik feltárása adja a fejezet lényegi mondanivalóját. Csak röviden térünk ki a problémák egyáltalán nem elhanyagolható szemléleti hátterére. A fejezet az adatmodellezés többszörös céljának a megvilágításával zárul.

12.2 Az adatmodell lényege

Az ***adatmodell*** a létrehozandó alkalmazási adatbázis fogalmi szintű szerkezeti tükörképe. A modell véges számú egyedtípusnak, valamint azok egyenként is véges számú tulajdonság- és kapcsolattípusának a szervezett együttese (ld. D 3.9). Az ***egyedtípusok*** a bennünket érdeklő jelenségeknek, a ***tulajdonságtípusok*** e jelenségek sajátosságainak, a ***kapcsolattípusok*** pedig azok viszonyainak az absztrakt fogalmi tükörképei.

Például: A TULAJDONOS és a KOCSI egy-egy egyedtípus; a konkrét tulajdonosok illetve kocsik (egyedelőfordulások) ismereti absztrakciói. A Tulajdonoskód, Tulajdonosnév, Rendszám és Kocsitípus olyan tulajdonságtípusok, amelyek a konkrét tulajdonoskódok, - nevek stb. (tulajdonságértékek) absztrakt osztályai. Végül a kocsik a tulajdonosok birtokában vannak, és ezeket a konkrét viszonyokat (kapcsolatelőfordulásokat) a TULAJDONOS - KOCSI kapcsolattípussal tudjuk együttesen és absztrakt módon kifejezni.

Az adatmodell három tényezője meghatározott logikai rend szerint függ össze egymással. A tulajdonságtípusokat általánosan is szemlélhetjük, és ekkor ***értéktartományoknak*** (doméjn) nevezzük őket. Például ilyen értéktartomány a Dátum. A tulajdonságokat az egyedekhez kapcsoljuk (attribútum), és ezzel kialakítjuk az egyed ***tulajdonságsorát***. Ezt gyakran úgy tesszük, hogy az egyedhez rendelt tulajdonságot ***szerepnévvel*** látjuk el. Például: amikor a RENDELÉS egyedhez kötjük a Dátum általános tulajdonságot, akkor annak a Rendelésdátum nevet adjuk.

Az egyed tulajdonságsorát másképpen az egyed ***belső szerkezetének*** is nevezzük. Azért lehet szó szerkezetről, mert a tulajdonságok az egyeden belül különböző feladatokat látnak el. A tulajdonságnak az egyeden belüli funkcióját ***relatív***, a modell egészét tekintve legfontosabb feladatát pedig ***abszolút szerepnek*** hívjuk. Például a Tulajdonoskód relatív szerepe a KOCSI egyedben leíró, a TULAJDONOS egyedben azonosító, és ezért abszolút szerepe is az utóbbi.

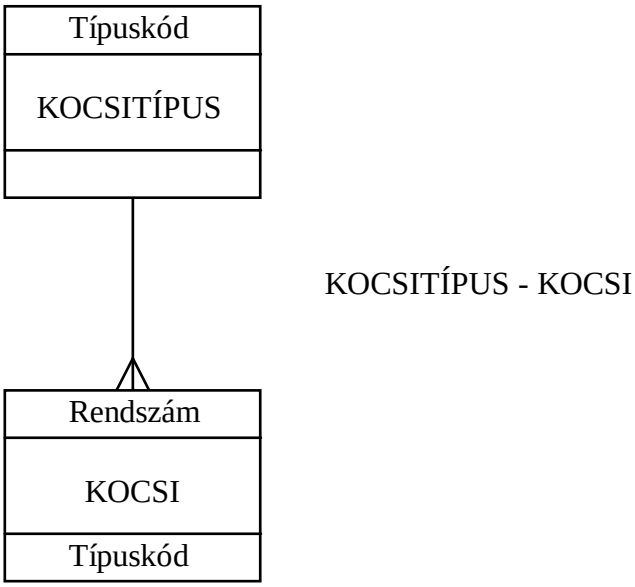
Az egyedek kapcsolatokat létesítenek egymással. Az egyed kapcsolatainak az együttesét az egyed ***külső szerkezetének*** nevezzük. Megkülönböztetünk inhomogén és homogén (más néven visszamutató) kapcsolatokat. Mindkét fajta összefüggés tulajdonságon alapul. Két egyedtípus homogén kapcsolatban áll, ha az egyik tartalmazza a másik azonosítóját vagy annak szerepnevét. Például a TULAJDONOS és a KOCSI azért kapcsolódik, mert az utóbbi leíróként tartalmazza a Tulajdonoskód tulajdonságot, amely az előbbi azonosítója. Az egyedtípus homogén viszonyt létesít önmagával, ha szerepnévként tartalmazza saját azonosítóját. Példa: a SZEMÉLY egyeden meghatározható a FŐNÖK - SZEMÉLY kapcsolat akkor, ha a Főnökkód leíró tulajdonság a Törzsszám azonosító szerepneve.

A kapcsolattípusokat alapvetően három szempont szerint osztályozzuk. A kapcsolat ***foka*** mondja meg, hogy az egyik egyedtípus adott előfordulásához tartozhat-e a másik egyedtípusnak több előfordulása vagy sem. Például a TULAJDONOS - KOCSI foka 1:N, ha minden kocsi csak egy (1) tulajdonos birtokában lehet, de a tulajdonosnak lehet több (N) kocsija is. Az ***opcionalitás*** azt mutatja, hogy az egyedelőfordulások hozzárendelése kötelező-e. Például minden kocsinak kell, hogy legyen tulajdonosa (kötelező), de nyilvántarthatunk olyan tulajdonost is, akinek nem ismerjük az aktuális kocsiját (opcionális). Végül a kapcsolat ***természete*** birtoklási vagy „is-egy” lehet. A TULAJDONOS - KOCSI kapcsolat birtoklási viszony, mert a tulajdonosoknak van kocsijuk. Viszont abban az esetben, ha a VEVŐ egyedtípus a PARTNER egyed altípusa, akkor a PARTNER - VEVŐ viszony „is-egy” természetű, mert a vevő is egy partner.

Az adatmodellnek két aspektusa van. A fentiekben az adatmodell ***szerkezeti*** nézőpontjáról volt szó. Azonban az adatmodell tényezőire és azok viszonyaira vonatkozhatnak olyan megkötések is, amelyek nem következnek magából a szerkezetből. Ezek a feltételek jelentik az adatmodell ***korlát*** aspektusát. A korlátoknak számos válfajuk van. Itt csak egy példát említünk: az adattípust. A Tulajdonoskód tulajdonságról el kell dönteni, hogy az karakteres vagy numerikus típusú adat legyen-e.

12.3 Ábrázolási konvencióink

A modell szerkezetét ***adatmodell-diagrammal*** szemléltetjük. Ebben dobozok jelölik az egyedtípusokat úgy, hogy a dobozban - az egyedtípus nevén túlmenően - megadjuk az egyed legfontosabb tulajdonságtípusainak a megnevezését is. A dobozok közötti vonalak reprezentálják a kapcsolattípusokat. A vonal „varjúlábas" vége N-es, sima vége 1-es kapcsolatot mutat (vö. fok). A folytonos vonal kötelező, a szaggatott opcionális kapcsolati oldalt jelez. Diagram-konvencióinkat az 5.7 ábra ismertette, amit itt megismételünk:



12.1 ábra: Az adatmodell-diagram alapvető konvenciói

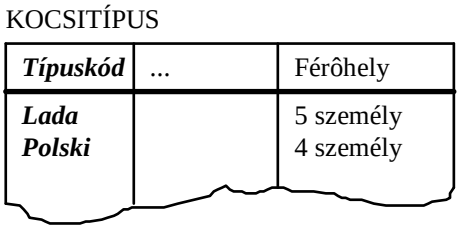
Lesznek esetek, amikor nem diagrammal, hanem ***szerkezeti leírással*** szemléltetjük majd mondanivalónkat. Az ilyen sémákban (ld. 12.1 példa) nagybetű jelöli az egyedtípust. A kapcsolatokban a két egyedtípus nevét kötőjellel választjuk el egymástól úgy, hogy elöl szerepel a főlérendelt egyed neve. A nevet követi a fok és az opcionalitás (O = opcionális, K = kötelező). Az egyed belső szerkezetét zárójelen belül mutatjuk. Az azonosító vastag és dőlt szedetű. Ha több tagból áll, akkor a tényezőket plusz jel köti össze. A kapcsoló tulajdonság dőltbetűs. Ha az egyedben más, a mondanivalónk szempontjából lényegtelen tulajdonság is szerepel, úgy azt a „..." jel mutatja. Így a 12.1 példa pontosan ugyanazt az adatmodellrészletet írja le, mint a 12.1 ábra.

12.1 példa

KOCSITÍPUS	(***Típuskód***, ...)
KOCSI	(***Rendszám***, ..., *Típuskód*)
KOCSITÍPUS - KOCSI	1:N, K-K

A jó tervező az adatmodell egyedeit ***mintával*** is reprezentálja, azaz felsorol néhány tipikus és kivételes egyedelőfordulást. Ezzel segíti az áttekintést. Meglehet, hogy a példa által olyan hibára bukkanunk, amit nem fedezhetünk fel az általános szerkezet alapján.

Az ilyen minták ábrái táblázatok. Amint tudjuk, a tábláknak van tartalmi (extenzionális) és terjedelmi (intenzionális) vetülete. Az előbbit a tulajdonságtípusok alkotják, amelyek vertikálisan egymás mellett jelennek meg a vastag vonal fölött. Az utóbbiak az egyedelőfordulásokat jelentik, amelyek horizontálisan egymás alatt találhatók a vastag vonal alatt. Ha a tábla alsó széle egyenes, akkor felsoroltunk minden egyed-előfordulást. Ha tördelt, akkor azt mutatja, hogy vannak számunkra érdektelen további tételek is a táblában. A 12.2 ábra mutat egy ilyen mintát.



12.2 ábra: Az adatmodell-minta egyezményes jelölései

Az adatmodellben a tulajdonságok értéktartományát is illenék megadni. Ezért a fenti példa és a két ábra valójában nem teljes. Példáinkban és ábráinkban a helytakarékosság miatt csak akkor tüntetjük fel a doméjneket, ha szerepnevek alkalmazására kerül sor. Vagyis ha az általános (doméjn) és az egyedhez kötött (attribútum) tulajdonság neve eltér egymástól. Egyébként nem mutatjuk az ábrán az értéktartományokat: azokat oda kell képzelni.

12.4 Tipikus adatmodellezési hibák

Az adatmodellezés és a modellelemzés valójában szinonim fogalmak. Nem tekinthetjük adatmodellnek azt az adatbázistervet, amely csak a ***formai kritériumoknak*** felel meg. Ugyanis semmi akadálya sincs annak, hogy az egyed-, tulajdonság- és kapcsolattípus hármasát pusztán formailag alkalmazva használhatatlan vagy nagyon rossz adatbázistervet készítsünk. Az adatmodellezés a pontosan meghatározható ***tartalmi követelmények*** szerinti optimális tervnek a kialakítását jelenti.

A kritériumok áttekintéséhez az szükséges, hogy megértsük az adatbázistervben fellépő hibák természetét. Az alábbiakban egy példán keresztül fogjuk bemutatni azt, hogy valóban lehet formailag megfelelő, de tartalmilag igen rossz modellt készíteni. A példa rávilágít az általános adatmodellezési problémákra és ezzel közvetve az alapvető kritériumokat is felvázolja. Lásd a 12.3 ábrát.

TULAJDONOS

Törzsszám	Típus	Tulajdonosnév	Foglalkozás	Telephely	Felügyelet
134567	Maszek	Kovács Rózsa	Keramikus 11	Budapest	-
134568	Kft	AB Kft	-	Szeged	-
136567	Maszek	Kovács Rózsa	Mérnök 32	Pécs	-
144567	Vállalat	XY vállalat	-	Budapest	Z

KOCSI

Rendszám	Típus	Tulajdonosnév	Foglalkozás	Telephely	Főhatóság	Férőhely
ABC 134	Lada	Kovács Rózsa	Keramikus K	Budapest	-	5 fő
BCD 265	BMW	AB Kft	-	Szeged	-	5 fő
DEF 896	Polski	Kovács Rózsa	Mérnök M	Pécs	-	4 fő
FGH 333	Lada	XY vállalat	-	Budapest	Z	5 fő

KÁR

Kárszám	Dátum	Típus	Tulajkód	Károsszeg
23456	05.14	Lada	134567	X
23456	05.14	BMW	134568	Y
23656	06.06	Polski	136567	Z
24456	06.31	Lada	134567	Q

12.3 ábra: „Beteg ló” minta-adatbázisunk részlete

12.4.1 Nyílt logikai átfedés

A 12.3 ábra mindhárom egyedét jellemzi a Típus nevű tulajdonság. Mégpedig olyan módon, hogy egyik egyedben sem azonosító szerepű. A KOCSI és a KÁR egyedekben a Típus értelme azonos: a kocsi fajtáját jelenti.

D 12/1 **Ha egy tulajdonságtípus több egyedtípushoz is azonos tartalommal kötődik úgy, hogy egyikben sem azonosító szerepű, akkor nyílt logikai átfedésről beszélünk.**

A meghatározás alapján a Típus tulajdonság nyílt logikai átfedésben van a KOCSI és a KÁR egyed vonatkozásában. A „nyílt” jelzőre a következő alpontok adnak majd magyarázatot. Mivel a Típus tartalma a TULAJDONOS és a KOCSI egyedben nem azonos értelmű, ez a két egyed nincs ilyen átfedésben, mert nem teljesül a meghatározás első feltétele („azonos tartalom”). Erről a jelenségről akkor sem beszélhetnénk, ha lenne egy KOCSITÍPUS egyedünk a Típus azonosítóval, mert akkor nem teljesülne a meghatározás második kitétele („egyikben sem azonosító”).

Megjegyzés: Az „azonos tartalom” kitételt csak didaktikai szempontból hangsúlyoztuk. Egyébként feltételezzük, hogy az azonos nevű tulajdonságtípus mindig azonos tartalmú. A követ-

kező alpontban mutatunk rá, hogy - sajnos - ez a teljesen normális feltételezés nem mindig érvényesül a gyakorlatban.

Az ilyen átfedés - közismert idegen szóval: redundancia - több nehézséget okoz. Mivel fizikai redundanciával (ld. 12.4.6) jár, feleslegesen fogyasztja a tárolót. Többszörösen kell az adatot bevinni és kezelni. Az átfedő adatok karbantartását egyidejűleg, többszörösen kell végrehajtani. Ha a különböző egyedtípusokban nem egyszerre történik meg az aktualizálás, akkor az adatbázis belső ellentmondást fog tartalmazni. Vagyis - idegen szóval - inkonzisztenssé válik. Ha valaki nem a „frissebb" egyedből kérdezi le az ismeretet, hamis információhoz jut.

Persze a kocsi Típus tulajdonságának az értéke nem változik, ezért annak esetében mindezek a gondok nem mutatkoznak annyira pregnánsan. Viszont a Tulajdonosnév is nyíltan átfedő tulajdonság. Ennek értékeinek többszörös tárolása, bevitele, módosítása már aligha lenne kívánatos. Ezért leszögezhetjük az első modellezési szabályt:

Sz1 ***Az adatmodellben kerülni kell a nyílt logikai átfedéseket.***

Ez a szabály nem szigorú. Az ilyen redundanciákat „kerülni kell". Ez nem azt jelenti, hogy minden ilyen jelenséget ki kell küszöbölni. Hiszen előfordulhat, hogy több egyedben is szerepel például a Mértékegység tulajdonság, mégpedig teljesen azonos tartalommal. A szabályt ezért két kitétellel finomítjuk.

Először: Azonosító szerepű tulajdonság sohasem lehet nyíltan átfedő. Azaz két egyedtípusnak nem lehet ugyanaz az azonosító tulajdonsága. Nem lehet például két Számlaszám nevű azonosítónk. Másodszor: A leíró szerepű tulajdonságok nyílt logikai átfedése nem kizárt, de nagyon ügyelni kell arra, hogy valóban nyílt átfedésről van-e szó, nehogy a következő alpontban leírt problémába ütközzünk.

12.4.2 Látszólagos logikai átfedés

A Típus tulajdonság mindhárom egyedben ugyanezzel a névvel szerepel. A KOCSI és a KÁR egyedekben a kocsi fajtáját, a TULAJDONOS egyedben viszont a birtokos természetes vagy jogi személy jellegét (az utóbbin belül is a konkrét cégformát) jelöli.

D 12/2 **Ha egy tulajdonságtípus azonos névvel, de eltérő tartalommal kötődik több egyedtípushoz, akkor látszólagos logikai átfedésről beszélünk.**

Itt nyilván az ún. ***homonimákról*** van szó. Ha két dolognak azonos a neve, de eltérő a lényege, akkor az ismeret nem kezelhető egyértelműen a név alapján. Ezt jól mutatja a következő mondat: „Készítsen kimutatást a kocsikról és tulajdonosaikról típus szerint!". Lehet ezen igény alapján tudni, hogy mi lesz a jelentés tartalma? Kocsitípus vagy tulajdonostípus szerint készül-e a kimenet?

A homonimák több bajt okozhatnak. Egyrészt az elemző a formai azonosság - közös név - alapján összetéveszthet a helyzetet a nyílt logikai átfedéssel. (Ezért adtuk a jelenségnek a „látszólagos" átfedés jelzőt.) Ebben az egyedben is szerepel egy Mennyiség nevű tulajdonság, meg abban is. Nosza, szüntessük meg valamelyiket, mert a redundancia nem jó dolog! Ekkor, ha a két mennyiség tartalma lényegében nem azonos, a tervező ismeretet veszít.

Itt egy kitérőt kell tennünk. A tervezők többsége úgy gondolja magáról, hogy ő el tudja kerülni az ilyen téves döntéseket. Nos hiszünk neki, amennyiben az adatbázis szerkezete egyszerű. Ám egy bonyolult struktúra esetében az ember képtelen átlátni a tényezők valamennyi kapcsolódását. A tapasztalatok szerint a 40-50 egyed- és/vagy 4-500 tulajdonságtípus méretheatárt túllépve az emberi tévedésnek igen nagy a valószínűsége. Ezért a problémákra vonatkozó

megállapításainkat nem pusztán a kis, direkt átláthatóvá tett szemléltető példáink vonatkozásában kell mérlegelni.

A homonima különösen veszélyes, ha valamelyik egyedben azonosítóként is fellép. Tegyük fel, hogy a Rendelésszám tulajdonság több egyedben leíró, egyben pedig azonosító. Ezért a tervező örömmel fedezi fel az egyedek közötti kapcsolatokat. Csakhogy pár egyedben a Rendelésszám a szállítói megrendelés száma, míg az azonosított egyedben a vevői rendelés számát jelenti. Ezért az az ilyen összekapcsolás teljesen téves ismeretekre vezet.

Végül gondolnunk kell arra is, hogy magának a modellezésnek az egyik részlépése, technikája a normalizálás. Ennek tárgyalásánál ki fogjuk mutatni, hogy a normalizálás feltételezi a nevek teljes egyértelműségét. Az adatmodell nem normalizálható, ha homonim neveket tartalmaz. Ezért megfogalmazhatjuk a második modellezési szabályt:

Sz2 ***Az adatmodellből ki kell küszöbölni a homonimákat.***

Ez igen szigorú megkötés. Az adatmodellben egyáltalán nem lehet homonima. Mivel a nyílt és a látszólagos logikai átfedés könnyen összetéveszthető, az adatbázis szellemi óriása E. F. Codd úr azt javasolja [16], hogy két tényezőnek sohase adjunk azonos nevet a modellben. Noha ezzel a nagyon szigorú megkötéssel nem értünk teljesen egyet, mégis fontosnak tartottuk megemlíteni. Azért, mert Codd úr kitétele jól mutatja, hogy mennyire komolyan kell venni az esetleges látszólagos átfedéseket.

12.4.3 Rejtett logikai átfedés

A példa szerkezete alapján az olvasó sohasem találta volna ki, hogy a TULAJDONOS egyed Törzsszám nevű tulajdonsága és a KÁR egyed Tulajkód attribútuma valójában ugyanazt jelenti. Pedig a minták - a konkrét értékek - egyértelműen ezt mutatják.

D 12/3 **Ha egy tulajdonságtípus eltérő névvel, de azonos tartalommal kötődik több egyedtípushoz, akkor rejtett logikai átfedésről beszélünk.**

Itt nyilván az ún. ***szinonimákról*** van szó. Ha két dolognak eltérő a neve, de azonos a lényege, akkor az ismeret nem kezelhető egyértelműen a név alapján. A következő mondat kétségeket ébreszt az emberben: „Készítsen kimutatást a károkról a tulajdonosok törzsszám sorrendjében!” De hiszen nincs is a KÁR egyedben Törzsszám nevű adat!

A szinonimák sok gondot okozhatnak. A formai eltérőség - más név - miatt a tervező nem veszi észre például azt sem, hogy a Főhatóság és a Felügyelet azonos tartalmú tulajdonságok. Nem fedezheti fel a redundanciát, mert hiszen az nem nyílt, hanem rejtett. Ezért nem szünteti meg a két tulajdonság egyikét, és így fellép a nyílt redundancia valamennyi problémája (ld. 12.4.1 alpont).

Másrészt a szinonima különösen veszélyes, ha valamelyik egyedben azonosítóként szerepel. A tervező az eltérő név miatt esetleg nem veszi észre azt, hogy a TULAJDONOS és a KÁR egyed egymáshoz kapcsolódik a közös tartalmú Törzsszám és Tulajkód tételen. Majd a normalizálásnál rá fogunk mutatni, hogy az egyértelműségnek ez a hiánya mennyire akadályozza az elemzést. Ezért közreadhatjuk a harmadik szabályt:

Sz3 ***Az adatmodell elsődleges változatából ki kell küszöbölni a szinonimákat.***

Itt meg kell magyaráznunk az „elsődleges változat” kitételt. Szemben a homonimákkal, a szinonimák terén a modellezés engedékenyebb. Az adatmodell semmilyen homonimát sem

tartalmazhat. Ezzel szemben ugyanazt a modelltényezőt különböző nevekkel is illethetjük, ha betartjuk a szinonimákra vonatkozó kiegészítő szabályokat. A tulajdonságnak lehet Mennyiségi-egység is és Mértékegység is a neve, ha valamelyikről kijelentjük, hogy az az elsődleges név, a másikról pedig tudatosan közöljük, hogy az előbbi szinonimája. Az elemzést a csakis elsődleges neveket tartalmazó - tehát egyértelmű - modellváltozaton kell elvégeznünk.

Mire jó akkor a szinonim név? A különböző megnevezések a felhasználó kényelmét szolgálják. Ha az egyik felhasználó „A”-nak, a másik pedig „B”-nek nevezi az „X” tényezőt, akkor miért ne lássák úgy, hogy a modell „A”-t illetve „B”-t tartalmaz? Eközben a tervező tudja, hogy valójában a modellben csakis „X” van. Viszont a szinonim megnevezések használatánál be kell tartani egy további fontos szabályt is: ugyanaz a név nem lehet több másiknak a szinonimája. E korlát jogosságát egy hétköznapi példával igazoljuk. A „Gabi” a „Gábor” és a „Gabriella” szinonimája. Ezért a „Gabi” hivatkozás kapcsán nem tudjuk, hogy fiúról vagy lányról van-e szó. Még világosabban: ha egy név két másiknak a szinonimája, akkor valójában homonima és azt már leszögeztük, hogy a modellben homonima nem engedhető meg.

12.4.4 Jelsor-ellentmondások

Ebben az alpontban nem egy teljesen új hibafajtáról lesz szó. Inkább „csak” arról, hogy a tervezők a homonima/szinonima kérdését nem érzékelik a maga teljes komolyságában és hajlamosak azt könnyedén venni. Néhány tipikus részhibára kell felhívnunk a figyelmet.

Az egyik probléma a homonima feltárásával függ össze. Példánkban a Foglalkozás tulajdonság neve és elvi értelme azonos a KOCSI és a TULAJDONOS egyedekben: a kocsi maszek tulajdonosának a szakmáját, hivatását jelöli. Például azért, hogy a másként adózó taxisofőrt megkülönböztesse az úrvezetőtől. Ha valaki figyelmesen nézte a 12.3 ábrát, akkor észrevehette, hogy a konkrét értékek a két egyedben eltérőek. Azok tervezője más-más osztályozási kódot alkalmazott. Tehát az elvi azonosság ellenére a két tulajdonság különbözik és valójában ***technikai homonima***. Azért csak technikai, mert a szándék közös volt; a megvalósítás ütött ki balul.

A Foglalkozás nevű tulajdonság a TULAJDONOS és a KOCSI egyedben technikai homonima, mert bár a név és az elvi tartalom azonos, más az értékkészlet. Viszont a Típus tulajdonság ugyanebben a két egyedben valós homonima, mert más az érdemi tartalom is.

Most pedig térjünk át a másik problémára, amelyre nézve nem tartalmaz külön tényezőt a „beteg-ló” mintapélda.

A tervezők bevett rossz szokása, hogy a tulajdonságok nevét valamilyen egyéni módon minősítik. Nagyon sokszor a tulajdonság által jellemzett egyedtípus nevének a rövidítését használják erre a célra. Például a Tulajdonosnév tartalmat a KOCSI egyedben a K_Tulajnev, a TULAJDONOS egyedben a T_Tulajnev megnevezéssel illetik. Ez többszörösen is csacska megoldás.

Először: A tervező az egyedtípus nevének a kezdőbetűjét akarja alkalmazni minősítőként. Azonban nem lehet következetes, mert hiszen több egyedtípusnak is lehet ugyanaz a kezdőbetűje (KOCSI, KÁR). Ezért a saját konvencióját is fel kell rúgnia. Nem jelölheti ugyanis ugyanaz a K_Tipus a KOCSI és a KÁR Típus tulajdonságát. Egy idő után pedig senki sem fogja tudni, hogy mit is jelent az R_Tipus (a KÁ*R* egyed Típus tulajdonsága).

Másodszor: Saját jelölése alapján a tervező azt fogja feltételezni, hogy a minősítő utáni név azonos lényeget takar. Ami óriási tévedés, hiszen a K_Tipus és a T_Tipus a KÁR és a TULAJDONOS egyedben teljesen mást jelent. Tehát konvenciója ismét csak nem válik be.

Harmadszor: Nincs az az automata rendszer, amely felfedezné, hogy a K_Tulajnev és a T_Tulajnev azonos lényegek. Ezért az automatával tudatni kell, hogy ezek a nevek valójában ***technikai szinonimák***.

Technikai szinonimáról akkor beszélünk, ha két tényező tartalma és értékkészlete teljesen azonos, nevük is „hasonlít”, csak éppen azok írásmódjában van „apró” eltérés. A technikai szinonima pontosan ugyanúgy viselkedik - ugyanazokat a gondokat okozza -, mint a valódi szinonima (vö. Felügyelet és Főhatóság). A technikai szinonima jelensége igen gyakori és sokszor csak a slamposságból ered. Nem emlékezve saját terveinkre ma Kocsi-tipus, holnap Kocsitipus, majd Gépkocsi-tipusa nevet adunk egyazon lényegnek.

A konkrét adatértékek és az adatnevek eltérő írásmódjából számos probléma fakad. Ezért meg kell fogalmaznunk az általános modellezési szabályt:

Sz4 ***Az adatmodell technikai homonimákat és szinonimákat sem tartalmazhat.***

E szabály indoka világos. Csakis az ember tudja megkülönböztetni a technikai és a valós homonimákat/szinonimákat. A gép erre nem képes.

Reméljük, hogy az olvasó nem felejtette el: mi a ***fogalmi*** adatmodellezésről beszélünk. Az adatkezelők semmilyen módon sem zárják ki az akár valós, akár technikai eredetű egyértelműségi zavarokat. Ezért a most nem tárgyalt ***logikai*** tervezési szinten - óvatos megfontolások után - alkalmazhatunk technikai szinonimákat (homonimákat semmiképpen sem), de nem javasoljuk az egyértelmű fogalmi modell ilyen lerontását.

Ide kapcsolódik következő megjegyzésünk is. A fentiekben nem helyesírási hibákat követtünk el, amikor a Tulajnev, Tipus megnevezéseket alkalmaztuk. A logikai tervezés szintjén a fejlesztők figyelnek az adatkezelő által a névben megengedett karakterekre és a név hosszára. Ezért nemékezethelyes és rövid neveket fognak alkalmazni, ami ismét technikai homonimákhoz és szinonimákhoz vezet. Sok mindennek van típusa és a „tulaj” rövidítés állhat a „tulajdonos” és a „tulajdonság” helyett is. Ilyen kényszerek nem határolják be a fogalmi modellezést. Ezért fogalmi szinten mindig természetes és teljes megjelöléseket (Tulajdonosnév, Típus) illik használni.

A valós és technikai homonimák illetve szinonimák elkerülésére lebegjen szemünk előtt a következő modellezési szabály:

Sz5 ***Két modellezési tényező akkor és csak akkor azonos, ha elvi értelmük, megnevezésük és előfordulás/érték-halmazuk százszázalékosan megegyezik.***

12.4.5 A logikai átfedés hiánya

A 12.3 ábra alapján senki sem tudja megállapítani, hogy egy káreseményben melyik konkrét kocsik vettek részt. Legfeljebb csak azt lehet megmondani, hogy ki volt a tulajdonos. A példaterv egyedei nem kapcsolhatók, idegen szóval ***inkonnektívek***.

D 12/4 **Ha az adatmodell két elvileg összefüggő egyedtípusa nem kapcsolható, mert nincs közös kapcsoló tulajdonságtípusuk, akkor a logikai átfedés hiányáról beszélünk.**

A TULAJDONOS és a KOCSI, a KÁR és a KOCSI elvileg összefügg. Mégsem tudjuk egyedelőfordulásaikat összekapcsolni. A „Kovács Rózsa” (tulajdonosnév) alapján nem deríthetjük ki, hogy ki az „ABC 134” rendszámú kocsi birtokosa, mert több ilyen nevű tulajdonos is szerepel az adatbázisban. Sohasem fogjuk megtudni, hogy a „23456” kárszámú káreseményben melyik konkrét kocsik vettek részt.

Adatbázisainkból sokszor a kapcsolati hiány miatt nem tudjuk előkeresni a megfelelő ismereteket, noha maguk az alapadatok - mint példánkban is - elvileg rendelkezésre állnak. Itt a modellezés egyik legkellemetlenebb problémájával kell szembenéznünk. Azzal, hogy sokkal könnyebben észlelhetők az adatmodell felesleges - redundáns - tényezői, mint az abból hiányzó

tételek. Pedig be kellene tartanunk a következő alapvető szabályt. Persze ahhoz azt kellene tudnunk, hogy két egyedtípus elvileg mikor áll kapcsolatban.

Sz6 ***Az adatmodellben nem engedhető meg a kapcsolati hiányosság.***

12.4.6 Fizikai átfedés

Az adatbázis táblázatokból áll. A táblázatoknak vannak oszlopai és sorai. A bevett konvenciók szerint a tábla „függőleges" dimenziójában helyezkednek el a tulajdonságtípusok (mert többnyire azok vannak kevesebben), „vízszintes" dimenziójában pedig az egyedelőfordulások. A tulajdonságok jelentik a tábla elvi tartalmát (intenzió), az egyedelőfordulások pedig a kiterjedést (extenzió).

Mindeddig elsősorban az adatmodell intenzionális aspektusáról volt szó. Azt vizsgáltuk, hogy az adatbázisban vannak-e valós vagy vélt általános tartalmi átfedések. Az „általános tartalom" miatt illettük az ilyen redundanciákat a „logikai" jelzővel. Azonban az adatbázisnak van konkrét tartalma is, amely az extenzionális dimenzióban - az adatbázis soraiban - fejeződik ki. Egyetlen probléma erejéig már erre is kitértünk, amikor az azonos név és elvi jelentés mögött különböző értékekre bukkantunk (ld. a 12.4.4 alpont első témáját). Rámutattunk arra, hogy az adatmodellt példákkal kell szemléltetni, amelyek felhívhatják a figyelmet a konkrét tartalmak által rejtett általános strukturális gondokra.

Ebben az alpontban a konkrét tartalmi átfedésről lesz szó, amit a „fizikai" jelzővel illetünk. Mivel 12.3 ábránk konkrét egyedsorokat is tartalmaz, rögtön szemet szúrnak az azokon belüli ismétlések. Kétszer írtuk le azt, hogy a Lada típusú gépkocsi öt férőhelyes. (Ha 100000 ilyen típusú kocsira vezetünk ismereteket, akkor az átfedés még pregnánsabb.)

D 12/5 **Fizikai átfedésről beszélünk akkor, ha az egyedtípusok előfordulásaiban azonos tulajdonságértékek vagy -sorok jelennek meg egy - nem feltétlenül explicit - további tulajdonságtípus értékétől függően.**

A fizikai átfedés lényegét nem könnyű megérteni, mert bonyolult jelenségről van szó. Az azonos tulajdonságérték nem mindig jelent fizikai redundanciát. Például több személynek is lehet Kovács Rózsa a neve. Ha viszont ez a név megjelenik a KOCSI és a TULAJDONOS egyedtípusban is a Tulajdonosnév értékeként, akkor fizikai átfedésről van szó bizonyos értelemben, de azt valójában a logikai redundancia (ld. 12.4.1 alpont) okozza.

Ezért fizikai átfedésről csak akkor beszélünk, ha egy egyedtípuson belül valamilyen elrendező elv szerint ismétlődnek az értékek vagy érték-részsorok. (Teljes értéksorok nem lehetnek azonosak, mert azt az adatbázis-elmélet eleve kizárja.) Ha a KOCSI egyedtípusban expliciten nem szerepelne a Típus tulajdonság, akkor persze nehezebb lenne feltárni a „Lada - 5 fő" páros ismétlődését, de attól még az „5 fő" redundanciája létezne.

A fizikai átfedések ugyanazokat a tárolási, beviteli, karbantartási stb. problémákat vonják maguk után, amelyeket a logikai átfedés (ld. 12.4.1 alpont) során már tárgyaltunk. Ezért a fogalmi modell szintjén ezeket a jelenségeket ki kell küszöbölni. Teljesen más kérdés az, hogy e modell megalkotása után - vagyis a hatások teljes tudatában - a logikai tervezés szintjén a fejlesztő mégis fizikai átfedéssel járó struktúrát alkot. Nyilván más dolog az, ha tudunk a redundanciáról és akarjuk is azt (például a hatékonyság érdekében), mintha a következmények véletlenül érnek bennünket a fogalmi modell nem kellő átgondolása miatt. Ezért ki kell fejtenünk a következő szabályt:

Sz7 ***A fogalmi adatmodell nem implikálhat fizikai átfedést.***

12.4.7 Kiegyensúlyozatlanság

A gondolatmenet kedvéért egy kicsit át kell alakítanunk a 12.3 ábra adatbázistervét. A KÁR egyedet kiegészítjük a Rendszám tulajdonsággal. Ezzel részben megoldjuk az inkonnektivitási problémát (ld. 12.4.5 alpont), hiszen most már a KOCSI és a KÁR egyed egymáshoz kapcsolható. Csakhogy az így kialakított modell sem lesz tökéletes. Mondanivalónkat a 12.4 ábra szemlélteti. Az ábrán csak azokat a tényezőket tüntettük fel, amelyek a most feltárni kívánt probléma megértéséhez szükségesek.

KÁR

Kárszám	Dátum	Típus	Tulajkód	Rendszám
23456	05.14	Lada BMW	134567 134568	ABC 134 BCD 265
23656 ***24456***	06.06 06.31	Polski Lada	136567 134567	DEF 896 ABC 134

12.4 ábra: Módosított KÁR egyedtípus

Az újonnan kialakított modell adatkezelési szempontból egyensúlytalan. Mivel a KÁR egyednek a Kárszám azonosítója, a Rendszám pedig leíró tulajdonsága, nagyon könnyen meg tudjuk mondani azt, hogy mely kocsik voltak egy káresemény részei. Csak a megfelelő KÁR egyedelőfordulásokat kell elérni az azonosító alapján. Ezzel szemben sokkal nehezebb annak megállapítása, hogy egy adott kocsi milyen károkat szenvedett. Ehhez végig kell keresnünk a KÁR egyed valamennyi előfordulását megnézve, hogy szerepel-e a kárban az adott rendszámú kocsi.

D 12/6 **Egyensúlytalan szerkezetről beszélünk akkor, ha a modell az elvileg egymás mellé rendelt egyedtípusok közül az egyik kezelését a másikénál jobban támogatja.**

Mi a probléma gyökere, elvi alapja? Példánkban a károk és a kocsik M:N-es viszonyban - azaz elvileg mellérendelt összefüggésben - állnak egymással. Egy kárban részt vehet több kocsi is (N) és egy kocsi több káresemény (M) áldozatává válhat. Ha tehát a kocsikat a károkhoz kötjük, de ezt fordítva nem tesszük meg (mint a 12.4 ábra megoldásában), akkor a kezelés nyilván egyoldalú lesz. Az egy kárhoz azonnal kikereshető az N résztvevő kocsi, de az egy adott kocsihoz nehezen találjuk meg az M darab kárt.

A kiegyensúlyozatlanság veszélye abban áll, hogy a mai adatkezelési igényeket preferáljuk a távlatiakkal szemben. Ha ma a kár oldaláról kell kikeresni a kocsikat és a fordított keresés nem fontos, akkor a 12.4 ábra megoldását választjuk. Viszont holnap a keresések mintája megváltozik. Az lesz a fontosabb kérdés, hogy milyen károkat szenvedett egy kocsi. Ekkor nyilván át fogjuk szerkeszteni az adatmodellt.

Vagyis a kiegyensúlyozatlan adatbázis szerkezete instabil. Ezért a fogalmi modellezés szintjén be kell tartani azt az egyszerű szabályt, amely szerint

Sz8 ***Az adatmodellben nem lehetnek kiegyensúlyozatlan részszerkezetek.***

Az ilyen szerkezetek feltárására és kiküszöbölésére a könyv további részeiben térünk ki. Itt a tervezők egyik általános rossz szokására hívjuk fel a figyelmet. „Miért beszélnek erről a problémáról, hiszen az nem is gond?" - mondanák sokan. „Húzunk egy indexet a KÁR egyed Rendszám tulajdonságára, aztán annyi..."

Egy átlagos mai magyar adatbázis mérete kétszer-ötször nagyobb a szükségesnél, ha csak a tartalmát tekintjük (vö. az előző alpontok átfedési problémáival). A tényleges méret az igényeltnél sokszor ötször-tízszer is nagyobb, ha a fizikai adatok mennyiségét nézzük. A hazai tervező túlzottan gyorsan nyúl az index nem-fogalmi, hanem fizikai szintű megoldásához. Nem tudja azt, hogy az azonosítón (Kárszám) épített index szükséges kellék, míg a leírón (Rendszám) alkalmazott mindig megfontolandó és a jó fogalmi szerkezettel mindig megelőzhető, kiváltható.

12.4.8 Tisztázatlan tartalmak

A Telephely tulajdonság a KOCSI egyedtípusban azt a földrajzi helyet jelöli, ahol a kocsit fizikailag tárolni szokták. Ezzel nem feltétlenül egyezik meg a TULAJDONOS telephelye. A budapesti telephelyű tulajdonosnak lehetnek veszprémi telephelyű kocsijai is.

Mármost az adatbázisok sokszor azért nem használhatók, mert a tulajdonságok értékeit nem azok valódi értelmének megfelelően adják meg. Például az „FGH 333” kocsi Telephely adatához a „Budapest” (ez a tulajdonos telephelye), nem pedig a valóságnak hűen megfelelő „Veszprém” (ez a kocsi tényleges tárolási helye) értéket rendelik.

Persze most joggal tehető fel a kérdés, hogy mi köze mindennek az adatmodellhez? A modellt jól megalkottuk, arról pedig nem tehetünk, hogy a felhasználó csacsiságokat ír az adatbázisba. Nincs az a program, nincs az a validálási eljárás, amivel ez a hiba kiszűrhető.

Pontosan arról van szó, hogy nincs az a program... Ezért a modellben kell tisztáznunk az ilyen kérdéseket. Erre két erőteljes eszközünk is van. Az egyik a ***név***. Ha a TULAJDONOS egyedben a Telephely, a KOCSI egyedben a Tárolóhely nevet használjuk, igen komoly mértékben csökkentjük a felhasználó tévedésének a lehetőségét. A másik eszköz a ***leírás***. Az adatmodellben a tényezőkhöz szöveges meghatározásokat kell adnunk. A Telephely azt jelenti, hogy..., viszont a Tárolóhelynek az a lényege, hogy...

Persze a felhasználói csacskaságot így sem fogjuk tudni elkerülni. Viszont ez csak akkor nem a mi gondunk, ha betartottuk a modellezési szabályt, amely szerint

Sz9 ***Az adatmodell tényezőit a beszélő nevek mellett rövid értelmező magyarázattal is el kell látni.***

A leggyakoribb adatmodellezési gond az, hogy a tervező időhiány vagy gondatlanság miatt nem tisztázza az alapfogalmak lényegét. Sohasem készít „fogalmi” adatmodellt.

12.4.9 A modell kettős arculata

A mai adatbáziskezelő rendszerek roppant szegényesek a nem-strukturális korlátok támogatásában. Persze elvileg sokféle validálást megengednek, de még nem eléggé fantáziadúsak és nem is kényszerítik ki az érvényesítés alkalmazását. Holott rengetegszer elmondtuk már, hogy az adatmodell nemcsak strukturális-, hanem egyben korlátrendszer is.

Vegyük csak alapul a 12.3 ábra KÁR egyedének a negyedik sorát. A kár dátumaként június 31-e szerepel - reméljük, hogy szemet szúrt az olvasónak -, ami tökéletes képtelenség. Merthogy a június harminc napos. Más esetekben kiköthető lenne az, hogy egy esemény csak ünnepnapokon, vagy éppen csak hétköznapokon jelentős a számunkra. A jól kifundált adatmodell tele van tűzködve a nem a szerkezet által tükrözött ***korlátokkal***.

Nézzük meg, hogy miképpen fejezhetjük ki az adatmodellben azt, hogy a kocsitulajdonosok típusa csak meghatározott értékeket vehet fel. Az egyik lehetőség az, hogy létrehozunk egy

TULAJDONOSTÍPUS (***Tulajdonostípus***) egyedet. Ez a TULAJDONOS 1:N fokú fölérendeltje lesz. Ekkor a megkötést struktúrával fejezzük ki. Azonban ezt a megoldást többnyire csak akkor fogjuk választani, ha a tulajdonosok típusáról valami többletet is el akarunk mondani. Különben a másik lehetőséghez fordulunk. Kijelöljük a Tulajdonostípus értéktartományt és listában megadjuk a megengedett értékeket. Ez már explicit korlát.

A korlátok számos válfaját itt nem ismertethetjük. Csak arra kívánjuk felhívni a figyelmet, hogy az adatmodellben a megkötéseket akkor is le kell írni, ha azok automatikus validálását a kezelő nem támogatja. Ebben az esetben ugyanis magunknak kell programozni az érvényesítést, és erről minden fejlesztőnek tudnia kell. Az adatmodell az ismeretekre vonatkozó összes ismeret tárháza, tehát a korlátokat is abban kell megadni.

12.4.10 A hibák emberi következményei

Az előző alpontokban felsoroltuk a legtipikusabb adatmodellezési hibákat. Korántsem tértünk ki minden lehetséges gondra. A könyv további fejezeteiben még számos további problémára kell felhívnunk a figyelmet. A bajok feltárásának és kiküszöbölésének a módját nem foglalhatjuk össze ebben az egy fejezetben. Ha az adatmodellezés annyira könnyű feladat lenne, hogy csak pár „rekordképet" kellene összeállítani, akkor ennek a könyvnek semmi értelme sem lenne.

Ezért egyelőre csak az adatbázistervező hallatlan felelősségére akarjuk felhívni a figyelmet. Ha egy adatmodellt rosszul szerkesztünk meg, akkor annak nem csak technikai értelemben lesznek kellemetlen következményei. Nem pusztán arról van szó, hogy a redundanciát figyelmen kívül hagyva többszörösen tároljuk az adatokat, feleslegesen növelve a tárigényt és a feldolgozási időt. Nem is csak arról, hogy a rosszul meghatározott - például egyensúlytalan - adatbázis-szerkezetet utólag majd módosítanunk kell, felesleges munkákat okozva saját magunknak.

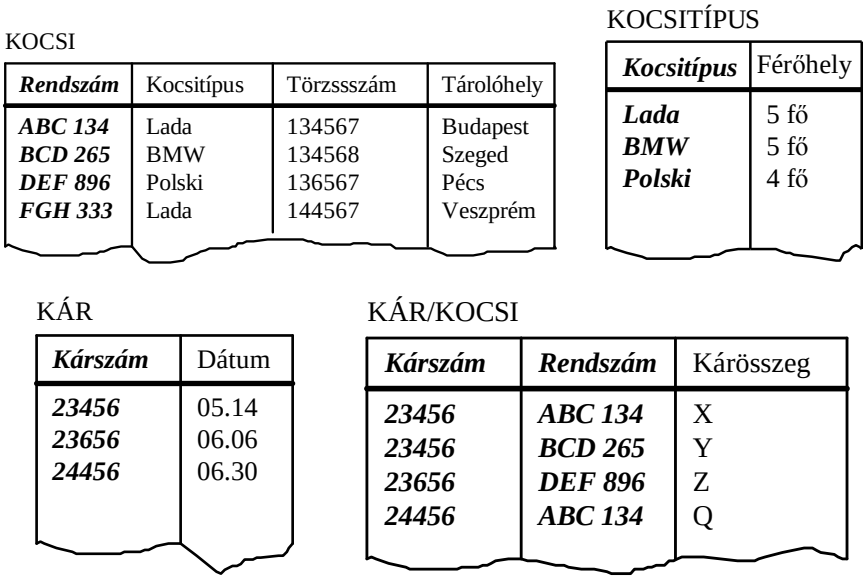
A rosszul szerkesztett adatmodell féloldalas vagy többszörös adatkezelést von maga után. Az ügyintéző a gépen babrál, miközben az ügyfél türelmetlenül veri mögötte a pultot. Mindez azért van így, mert rosszul - féloldalasan - határozták meg a szerkezetet, és ezért elfogadhatatlan az elérési idő. Az adatbázis több funkciót is kiszolgál. Ha többszörösen tartalmazza ugyanazokat az ismereteket, akkor a felhasználónak nap mint nap ki kell töltenie ugyanazokat a papírokat részben ugyanazokkal az adatokkal csak azért, mert az adatbázis szerkezete rossz. A duplikációk és kapcsolati hiányok miatt kétszer küldünk számlát ugyanarra a címre; egyszer sem küldjük el a számlát a megfelelő személynek; viszont számlát kap az, akinek ahhoz semmi köze sincs.

Rengetegszer előfordul, hogy mindezeknek az atrocitásoknak az okát nem az adatkezelésben, a téves adatbevitelben és/vagy -karbantartásban, hanem az adatmodell helytelen felépítésében lehet megtalálni. Ezért a tervezőnek nemcsak technikai, hanem egyben emberi kötelessége is az optimális adatmodell meghatározása.

A 12.5 ábrán bemutatjuk a 12.3 ábra feladatának az optimális megoldását. Az előző alpontokban ismertetett problémákat sorra véve bárki meggyőződhet arról, hogy azok egyike sem lép fel ebben az adatmodellben.

TULAJDONOS

Törzsszám	Tulajdonostípus	Tulajdonosnév	Foglalkozás	Telephely	Felügyelet
134567	Maszek	Kovács Rózsa	Keramikus K	Budapest	-
134568	Kft	AB Kft	-	Szeged	-
136567	Maszek	Kovács Rózsa	Mérnök M	Pécs	-
144567	Vállalat	XY vállalat	-	Budapest	Z



12.5 ábra: Egy hibamentes adatmodell

12.5 Az adatmodellezési hibák gyökerei

Könyvünkben el fogjuk árulni, hogy miként tudtunk optimális megoldást kerekíteni a rettenetesen rosszból. Azonban e titok feltárása előtt rá kell mutatnunk arra, hogy a rossz adatbázistervek kialakításáért nem annyira a tervezési technikai ismeretek hiánya, mint az általánosan elterjedt helytelen szemléletmód a felelős. Márpedig az adatmodellezés legalapvetőbb lényege a helyes megközelítés.

Az igen rossz adatmodelleket a megfelelő ***modellezési technikák*** alkalmazásával elkerülhetjük. Mi több, az optimálishoz közelítő adatbázisterveket is készíthetünk. Azonban a technika nem minden. Nap mint nap találkozunk olyan fejlesztőkkel, akik ismerik az adatmodell lényegét, tisztában vannak például a később tárgyalandó normálformákkal, mégis pocsék adatbázistervet állítanak össze. Az ilyen szakembereknek a ***modellezési szemléletmóddal*** van bajuk. Négy ponton kell keresni a hibák gyökereit.

12.5.1 Nézetvakság

Ez a betegség általában a felhasználókat jellemzi, akik viszont a fejlesztőket is megfertőzik. A felhasználó egy-egy adott, a kocsikat érintő kár ügyeinek az intézésében érdekelt. Ezért számára tökéletesen megfelel a 12.4 ábra megoldása. Az ő „kár-kocsi” irányú nézetét átveszi a fejlesztő is, nem gondolva arra, hogy lesznek majd „kocsi-kár” szemléletirányú alkalmazók. A felhasználó csak a mai, az általa legsürgősebbnek vélt ismeretekben gondolkozik és ez a gondolatmenete óhatatlanul átragad az őt kiszolgáló fejlesztőre is.

E nézetvakság következményei teljesen evidensek. Most már elárulhatjuk, hogy a 12.3 ábra adatbázisa azért annyira pocsék, mert három egyedtípusát három külön felhasználó három olyan „állományként” vezettette be a fejlesztőkkel, amelyeknek egymáshoz semmi közük sem volt. Persze ezek után a fejlesztő bevetheti az adatbázistervezés összes fejlett technikáját. Vizsgálhatja

az adatátfedéseket, elemezheti a normálformákat a „saját” egyedtípusain. Semmire sem fog jutni a teljes adatbázis vonatkozásában. Az továbbra is redundáns, kapcsolhatatlan és ellentmondó ismereteket fog tartalmazni.

Mert az adatbázistervezés első és legfontosabb titka az, hogy

az adatbázis - egy.

Nincs értelme egy szervezeten belül az „X” vagy az „Y” felhasználó, a „Q” vagy a „Z” funkciók adatbázisáról beszélni. Csakis „az” adatbázisban szabad gondolkodni; az egyetlenben. Különben a szervezet szintjén nem kerülhetjük el a nyílt átfedéseket, a homonimák okozta látszólagosakat, a szinonimákból fakadó rejtetteket, a technikai kétértelműségeket, a kiegyensúlyozatlanságot és a kapcsolati hiányokat.

Ezek a problémák pedig azt eredményezik, hogy a valóban végső-felhasználó - az állampolgár, az ember - többszörösen tölt ki papírokat, nem jut a megfelelő ismerethez, rossz információkat kap. Végeredményben feleslegesen szaladgál és bosszankodik. Ugyanígy túlterhelt, mert felesleges munkákat végez a minket kiszolgálni hivatott ügyintéző is.

12.5.2 Szintvakság

Most a fejlesztők tipikus betegségére térünk át. A fejlesztő az X vagy az Y adatkezelő rendszer „csodálatos” képességeiben gondolkozik. Nem tudja azt, hogy a mai „szuperkezelők” alkotói nem sokat hallottak arról, hogy az adatbázisnak fogalmi, logikai és fizikai szintje van. A fogalmi szintet egyáltalán nem támogatják, a másik kettőt pedig rendesen összekeverik. A természetes (fogalmi) adatmodellt nem lehet kifejezni a segítségükkel, mert például a kezelőben képtelenség megfogalmazni - mondjuk - a visszamutató kapcsolatot. Ezért a tervezőnek valamilyen trükkös tartalmi megoldást kell keresnie. A séma-leírásokban e tartalmat és a megvalósítási módot (például: indexek) együtt kell megadni, összezagyválva az adatbázis logikai és fizikai szintjét.

Mindennek evidens következménye, hogy a kezelőt alkalmazni kívánó tervező eleve zavaros gondolkozásra késztetődik. Az adatkezelő nem zárja ki sem a nyílt logikai átfedéseket, sem a homonimákat, sem a szinonimákat, sem a kapcsolati hiányokat. Viszont megköveteli az adatbázis indexeinek a meghatározását. Akkor pedig minek adatmodellt alkotni?

A fejlesztőket az adatkezelő-forgalmazók megtévesztik. Mivel ma ez annyira divatos kifejezés, adatmodellnek nevezik az adatbázis logikai-fizikai adatszerkezet-mixtúrájának a leírását. Így a fejlesztő nem döbben rá az adatbázistervezés másik alapvető titkára, amely szerint

az adatmodell - fogalmi.

Magyarul: Nem az a fontos, hogy az adatkezelő milyen logikai-fizikai varázslatokat kínál a számunkra. Az első lépésben nem ezekkel kell törődnünk, hanem az ismeretek valóságnak megfelelő elrendezésével. Az ezt a titkot nem ismerő fejlesztő szintvakságban szenved és így képtelen arra, hogy jó adatbázist építsen.

12.5.3 Szerepvakság

Az ember azt hinné, hogy a felhasználók és a fejlesztők éjt nappallá téve közösen azon fáradoznak, hogy kimunkálják a lehető legjobb adatmodellt. Mert hiszen ez lenne a szervezet, a cég alapvető érdeke. Sajnos a valóság teljesen másként fest.

Az információs rendszerek sikertelenségének az egyik alapvető oka az, hogy a tisztelt felhasználó nincs tisztában a saját maga ismeretigényeivel és a munkája ellátásához szükséges alapfogalmakkal sem. Például nem tudja, hogy a Kocsiminőség a kocsi fizikai állapotát jelenti és ezért a „piros" tartalmat adja ennek az adatnak. Ettől persze rossz lesz az adatbázis (ld. 12.1 pont). Még nagyobb baj, hogy az alapfogalmakat sűrűn változtatja. Ma ezt, holnap azt érti a Kocsitípuson. A változtatás sürgős. Arra nincs idő, hogy a már tárolt adatokat átdolgozzák. Így a Kocsitípus inkonzisztenssé válik - és az adatbázis rossz lesz.

A felhasználók úgy vezetnek be új bizonylatokat, kódokat, adatokat, hogy azokat nem egyeztetik a már meglévő adatmodellel. Azt hiszik, hogy az adatbázis egyetlen feladata az ő igényeiknek a mindenáron való kiszolgálása. Így fordulhat elő, hogy ugyanabban az adatbázisban hol Felügyeletnek, hol Főhatóságnak nevezik ugyanazt a lényeget. Hol karakteresen, hol numerikusan kódolják a Foglalkozást. A felhasználó sokszor nem hallgatja meg a fejlesztőt, hanem „diktál" neki.

Persze a fejlesztők is sokszor szerepet tévesztenek. Önhatalmúlag rendezgetik el az ismereteket. A természetes és egyenes megoldások helyett trükköket alkalmaznak. Most még felveszik azt az adatot, hogy ..., mert az nekik kényelmes. Eszükbe sem jut, hogy a végső-felhasználóknak - nekünk - ezzel mennyi többletmunkát okoznak. Az ismereteket megduplázzák vagy eltorzítják. Pusztán csak egy technikai jellel egészítik ki az Alvázszám-ot. Így fordulhat elő, hogy ugyanazon szervezet „két" adatbázisának egyikében az Alvázszám 24, másikában 25 karakteres. Ettől kétféle papírra, bizonylatra, nyomdatechnikai megoldásra stb. van szükség. Mert a fejlesztő - szerepet tévesztve - fantáziál.

Nálunk nem ismerik fel az adatmodell harmadik fontos szemléleti titkát, amely szerint

az adatmodell a felhasználó és a fejlesztő közös terméke.

Erről nincs több mondanivalónk. Ha a felhasználó nem tudja megmondani, hogy milyen ismeretre van szüksége, a fejlesztő pedig a saját szája íze szerint formálgatja az adatbázist - vagyis mindkét fejlesztési résztvevő szerepvakságban szenved -, akkor nincs értelme az adatmodellezésnek.

12.5.4 Bemenet/kimenet vakság

Most tegyük fel, hogy a felhasználó és a fejlesztő tisztában van az adatbázis építése során betöltendő feladatával. A fejlesztőt nem tévesztik meg az adatkezelő gyöngécske képességei és az adatbázis szintjeinek a csapdái. Tehát minden fejlesztési résztvevő tudatosan törekszik a jó adatmodell kialakítására. Bizonyosak lehetünk-e abban, hogy az ilyen igen ritka kedvező konstelláció esetén jó adatmodell fog születni?

Még mindig nem. A fejlesztőt és a felhasználót nagyon sokszor megtéveszti a bemenethez vagy kimenethez kötött szemlélet. Sokszor találkozunk olyan adatbázissal, amelynek szerkezete nem a valóság jelenségeinek a tényleges összefüggéseihez, hanem valamilyen papírhoz - bemeneti vagy kimeneti bizonylathoz - igazodik.

A 12.3 ábra KOCSI egyedtípusának a tervezője ***bemenetorientált*** adatbázistervet készített. Elétettek egy bizonylatot, amelyen feltüntették a kocsira vonatkozó adatokat. A tervező a bizonylatból egyetlen állományt kreált abban a tudatban, hogy ezzel kielégíti a felhasználó ismeretigényeit. Csak éppen a fizikai redundanciára nem gondolt. Nem számolt azzal, hogy ugyanannak a tulajdonosnak több kocsija is lehet. Ha a KOCSI egyedet jellemzi a Tulajdonosnév, akkor az nemcsak redundancia, hanem fennáll a nem azonos beírás miatti inkonzisztencia veszélye is. Több ezer Lada 2104-es típusú kocsi futkározik az utcákon. Mindegyiknél le kell írni, minde-

gyiknél külön kell tárolni, hogy a kocsi 5 férőhelyes, benzin üzemű, motorja X köbcentis, a kocsi súlya Y kiló stb.

A bemenetorientált - egy bizonylat=egy állomány - tervek mindig fizikai redundanciával és inkonzisztenciával párosulnak. A valójában X köbcenti helyett valamelyik papírra Z köbcentit fognak írni - és az adatbázis máris rossz. Mindezek a bajok megelőzhetők a 12.5 ábra KOCSITÍPUS egyedének a bevezetésével.

Vannak olyan tervezők is, akik a felhasználói igényekre alapozva ***kimenetorientált*** adatbázistervet alkotnak. Ennek tipikus példáját mutatja a 12.3 ábra KÁR egyede. A felhasználó olyan statisztikai kimutatásban volt érdekelt, amely kocsitípusok szerint összesíti a károkat. A fejlesztő ezt a szemléletet átvette. Nem nézve a dolgok valós összefüggései mögé, egy állományt készített a kimeneti igény alapján. Elfelejtette, hogy holnap a kapacitás, holnapután pedig a férőhely vagy a kocsi színe szerinti kimutatást fognak kérni. Ekkor az adatbázis teljes átstrukturálása elkerülhetetlen.

A kimenetorientált - egy kimutatás=egy állomány - tervek mindig kapcsolati hiányokat implikálnak. A KÁR egyedből nem érhető el a KOCSI egyed. Ha hozzáférhető lenne, mint a 12.5 ábra szerinti modellben, akkor a károkat ki lehetne gyűjteni típus, férőhely, kocsiszín - a kocsi minden közvetlen és közvetett (KOCSITÍPUS) tulajdonsága - alapján az adatbázis átalakítása nélkül.

A vérbeli adatbázistervező a fentebb leírt hibákat eleve elkerüli. Azért, mert ismeri az adatmodellezés negyedik szemléleti titkát, amely így szól:

az adatmodell szerkezete bemenet- és kimenetfüggetlen.

Természetesen az adatbázisnak az igényelt kimeneteket kell kiszolgálnia úgy, hogy a bemeneteken lévő adatokat tárolja. Nem azt mondtuk, hogy az adatbázis bemenet- és kimenetfüggetlen, hanem azt, hogy a szerkezete az. Magyarul: Az adatmodellben lévő egyedtípusok belső és külső szerkezete sokszor nem egy az egyben követi a papírokon lévő ismeretek szerkezetét. Az adatmodell a valóságot tükrözi. Más dolog a tulajdonos, a kocsi, a kocsitípus, a kár (mint esemény) és az eseményben az egy-egy kocsit ért kár (ld. 12.5 ábra). Ezért ezt az ötféle jelenséget öt egyedtípusban kell modellezni.

Nem mindig könnyű feltárni azt, hogy melyek a bemenetek és kimenetek mögött rejlő valós jelenségek és azok összefüggései. Könyvünk el fogja árulni a helyes struktúra megtalálásának a titkait. Amelyeket viszont csak az érthet meg, aki felismeri, hogy az adatbázis szerkezete független a bemenetektől és a kimenetektől. Mi több, nem az adatbázist kell a bemenetekhez szabni, hanem éppen megfordítva. Ha létezik KOCSITÍPUS egyedtípusunk, akkor teljesen felesleges a kárbejelentő lapokon megkövetelni a férőhely, köbcenti, súly stb. adatok kitöltését. Ha létezne egyértelmű tulajdonos Törzsszámunk... De ezt már nem részletezzük. Az olvasó már bizonyára belátja, hogy a jó adatmodellel papírt, pénzt, időt, energiát lehet megtakarítani.

12.6 Az adatmodellezés céljai

Az adatmodellezés ma divatszóvá lett. Ha valaki kitalál pár rekordképet, akkor már adatmodellt emleget. Különösen feljogosítva érzi magát e kifejezés használatára, ha a mai CASE-eszközök (Computer Aided Systems Engineering - számítógéppel támogatott rendszertervezés) valamelyikével állítja össze a tervét.

Mára az a furcsa helyzet állt elő, hogy a legkorszerűbb segédletekkel gyorsan és kényelmesen lehet igen rossz adatbázisokat készíteni. A CASE-ek nem követelik meg az egyeztetést és a 12.4 pontban felsorolt problémák feltárásában csak minimális segítséget adnak. Ezért tartottuk annyira

fontosnak a megfelelő szemlélet hangsúlyozását. Ennek elsajátításához a leendő tervezőnek tisztában kell lennie az adatmodellezés többféle céljával.

Az adatmodellezés végső, technikai célja természetesen az, hogy legyen egy megvalósítható ***adatbázistervünk***. Azonban az adatmodell és a számítógépbe beírt adatbázis-séma nem azonos. A modellben lesznek olyan korlát-kitételek is, amelyeket a kezelőrendszer nem támogat. Ezért a modell sokkal több, mint számítógépes adatstruktúra.

Az adatmodellezés másik fontos célja a felhasználói ismereteknek és azok összefüggéseinek ***az alapos megismerése***. A mindennapos életben gyakran használjuk a „fogalmam sincs" kifejezést. Ha nincs fogalmunk arról, hogy mit jelent a kocsitípus, akkor miként építhetünk jó adatbázist a gépkocsik ismereteiből? Az adatmodell a felhasználó és a fejlesztő elképzeléseinek az egyeztetésére szolgál (ld. 12.5.3 alpont). Hibásnak kell tekinteni azt a gyakorlatot, amelyben a fejlesztő a végső adatbázistervet közli a felhasználóval. Ez a terv sok olyan alkalmazás- és technikai-környezeti megfontolást tartalmaz, amely nem tartozik a felhasználóra és őt csak megzavarja. Ezzel szemben a fogalmi szintű adatmodell igen kiváló eszköz a felhasználói célok és a fejlesztői elképzelések közös nyelven történő egyeztetésére.

Az adatmodellezés harmadik célja az ***optimalizálás***. A felhasználó kimenetekben és bemenetekben gondolkozik, mert hiszen ezeken keresztül érintkezik az általa teljes egészében nem is látható adatbázissal. Fennáll az a veszély (ld. 12.5.4 alpont), hogy a fejlesztő átveszi a felhasználói nézetet és rossz adatstruktúrát alkot. Ezért a fejlesztő kötelessége az, hogy az első tervelképzeléseit a modellezési technikák „trükkjei" alapján (át)értékelje. Kiszűrje a terv egyértelműségi, átfedési és hiány-jellegű problémáit. Vagyis megalkossa az optimális adatmodellt. Tehát nem pusztán az a feladata, hogy egy számítógépen kezelhető adatbázis-szerkezetet hozzon létre (ld. első cél), hanem az, hogy a létező legjobb struktúrát alakítsa ki.

Végül az adatmodellezésnek van egy tágabb, negyedik célja is. Az ***összehangolásról*** van szó. Az ismeret a vállalat legfontosabb erőforrása. Vajon mi történne azzal a céggel, amelyikben a pénz erőforrás nincs összehangolva és mindenki úgy költ, ahogy akar? Amelyben az emberi munka nincs harmonizálva és mindenki azt tesz, amit akar? És amelyben az ismereteket csak úgy átabotában határozzák meg?

Az adatmodell a vállalat ismereteinek a térképe. Olyasmi, mint a főkönyv a pénz erőforrás vonatkozásában. A főkönyv vezetésének roppant szigorú szabályai vannak. Érdekes, hogy a pénzbe kerülő és pénzt fialó ismeretek elrendezésében nem léteznek ezek a komoly elvek. Pedig az egyetlen adatmodell szolgálhatna ismereteink magasfokúan tudatos összehangolására.

Bár az adatmodellezés utóbbi szerepéről nem hallgathattunk, könyvünk további részében elsősorban az előző két célról - a megismerésről és az optimalizálásról - lesz szó. Az adatmodellnek működő adatbázistervvé való leképezésére nem térhetünk ki részletesen. Azért nem, mert az alkalmazási környezetek nagyon eltérőek és ezért a logikai szintű tervezési megfontolások is nagyon különbözőek. A fizikai szintű tervet pedig számos technikai-környezeti tényező befolyásolja. Ezért a logikai/fizikai megfontolásokra nem térhetünk ki.

Ellenőrző kérdések - 12

12/01 Milyen problémákat fedez fel a következő adatbázis-részletben? A 12.4 pont megfelelő alpontjának a számát adja meg.

VEVŐ (***Vevőkód***, Vevőnév, Vevőcím)
RENDELÉS (***Rendelésszám***, Rendelésdátum, *Vevőkód*, Vevőcím)

12/02 Az előző módon adja meg az alábbi terv hibáinak a számát.

VEVŐ (***Vevőkód***, Vevőnév, Vevőcím)
RENDELÉS (***Rendelésszám***, Rendelésdátum, Vevő-megnevezés)

12/03 Melyik hibát mutatja az alábbi terv? A nyelv háromszorosan ismétlődő tartalmú.

SZEMÉLY (***Törzsszám***, ..., {Nyelvkód})

12/04 Tartalmaz-e hibát a következő terv? Nem (N), súlyosat (S), kicsit (K).

RENDELÉS (***Rendelésszám***, ..., Dátum)

12/05 Tartalmaz-e hibát a következő terv? Nem (N), súlyosat (S), kicsit (K).

CÉG (***Cég-adószám***, ..., Cég-név)
SZÁMLA (***Számlaszám***, ..., *Számla-adószám*)

12/06 Ön szerint milyen szemléletben készült az alábbi terv? Helyes (H), bemeneti (B), kimeneti (K)?

SZEMÉLY (***Törzsszám***, ..., Szervezetkód, Szervezetnév, ...)

12/07 Ön szerint milyen szemléletben készült az alábbi terv? Helyes (H), bemeneti (B), kimeneti (K)? A „{}” jelek közötti adatok ismétlődnek.

RENDELÉSEK (***Rendelésszám***, ..., {Cikktípus, Árfekvés, Rendelt-darab})

12/08 Milyen problémát lát a következő tervrészletben? Szóban fejtse ki gondolatait.

CIKK (***Cikkszám***, ..., Ár)
RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, ..., Ár)

13. FÜGGÉSEK ÉS NORMALIZÁLÁS

13.1 Egyszerű szabályok

Az adatmodellezés lényege az, hogy feltárjuk és megfelelően tükrözzük a valóság jelenségeit (egyedek), azok sajátosságait (tulajdonságok) és összefüggéseit (kapcsolatok). E művelet során nemcsak egy elfogadható, lehetséges adatbázisszerkezetet keresünk, hanem törekszünk az ***optimális*** modell megkeresésére. Az „optimális" kitétel itt nem a legjobb számítógépes megoldást jelenti. Persze azt is meg kell keresnünk, de az a fizikai szintű adatbázistervezés feladata. Fogalmi szinten optimális modell az, amely

- valósághű
- egyértelmű
- teljes
- és minimális.

A modellnek valósághűnek kell lennie. A KOCSI egyedtípus Telephely tulajdonságtípusa (ld. 12.3 ábra) a lényeget fejezze ki. Ne a tulajdonos címére, hanem a kocsi tényleges tárolási helyére (ld. 12.5 ábra) utaljon. Az egyértelműség követelménye kizárja a homonimákat és a szinonimákat. A 12.3 ábra modellje nem teljes, mert a KÁR nem kapcsolható a KOCSI-hoz. Nem is minimális, mert feltételezi, hogy minden kocsinál megadjuk a Férőhely értékét, holott azt elegendő lenne kocsitípusonként egyszer vezetni.

A valósághűség és az egyértelműség problémáiról kevés szó lesz ebben a fejezetben. Ezek ugyanis a nehezebben megfogható kritériumok. A teljesség követelménye sem könnyen feltörhető dió. Éppen ezért a hagyományos adatmodellezési módszerek leginkább a minimalitást - a redundancia elkerülését - célozzák meg.

Az adatmodellnek megfelelően szervezett adatbázis minimalitásának a vizsgálatára, a redundáns adattárolás és -kezelés megakadályozására jól megfogható, egyszerű szabályok állnak a rendelkezésünkre. Nem azt mondjuk, hogy ezeket a regulákat mindig könnyű érvényesíteni a gyakorlatban. Hanem csak azt, hogy az elvi szabályok tisztázottak, mivel jól érthető és egyszerű, tisztán ***matematikai*** jellegű összefüggéseken alapulnak.

Ebben a fejezetben néhány alapfogalom tisztázása után az adatmodell „gyerekbetegségeit" és azok orvoslási módját fogjuk feltárni.

A relációs adatmodell [19] elméletének a bevezetése során figyeltek fel az ismeretek bizonyos számszerű viszonyaira és az adatok eltérő (azonosítási és leírási) szerepeinek e számszerű összefüggésekkel való kapcsolatára. Ezek a felfedezések „örök érvényűek" és akkor is figyelembe veendők, ha nem relációs adatkezelőt használunk. Ezért ebben a picit unalmas fejezetben arra kényszerülünk, hogy néhány alapfogalmat zúdítsunk az olvasóra.

13.2 Funkcionális függés

Az adatmodellben résztvevő tulajdonságtípusok között többféle függvényszerű összefüggést fedezhetünk fel. Ezek közül a legismertebb az ún. ***funkcionális függés*** (angolul: functional dependence), amelynek jele ***FD***. A funkcionális függést adott egyedtípus tulajdonságtípusai között értelmezzük a következő módon:

D 13/1 **Az E egyedtípus B tulajdonságtípusa akkor és csak akkor funkcionálisan függ az egyedtípus A tulajdonságától, ha az E minden előfordulásában az A értéke minden időpontban csakis egy B értékkel társul.**

Tegyük fel, hogy a kocsik rendszáma egyedi és adott a következő egyedtípus:

13.1 példa

KOCSI (***Rendszám***, Kocsitípus, Szín, Tulajdonoskód ...)

Ha minden kocsinak minden időpontban csak egy típusa, színe és tulajdonosa lehet, amint azt feltételezzük, akkor igaz az, hogy a Rendszám ***funkcionálisan meghatározza*** az összes többi tulajdonságot. Vagy - ami ugyanaz - a Kocsitípus, a Szín és a Tulajdonoskód ***funkcionálisan függ*** a Rendszámtól. Az FD jele „→”.

A példában a következő függéseket fedezhetjük fel:

Rendszám → Kocsitípus
Rendszám → Szín
Rendszám → Tulajdonoskód

vagy

Rendszám → (Kocsitípus, Szín, Tulajdonoskód).

Az utóbbi rövidített jelölésnek a funkcionális függés ún. ***additivitási szabálya*** ad alapot, amely így szól:

Ha A → B
és A → C
akkor A → B, C.

Ez a megkötés a három Armstrong-szabály [20] egyike. A másik kettőről majd később lesz szó. Ezeken túlmenően az FD-re természetesen vonatkozik az ún. ***reflexivitási szabály*** is, amely szerint A → A, vagyis minden tulajdonságtípus meghatározza saját magát.

Példánk esetében a függések fennállnak, mivel feltételezésünk szerint minden - a Rendszám által azonosított - kocsinak csak egy típusa, színe és tulajdonosa (leíró jellemzők) lehet. Az A → B, C nem azt jelenti, hogy a B-nek és a C-nek valami köze van egymáshoz. Ez a rövidített forma csak azt mutatja, hogy adott Rendszám értékhez egyetlen Kocsitípus ***meg*** egyetlen Szín érték tartozik. (Ennek a „meg”-nek a későbbiekben még lesz szerepe.)

Az FD nem feltételezi azt, hogy a meghatározó tulajdonság értéke egyedi legyen az egyedben. Tegyük fel (persze ez nem igaz), hogy egy tulajdonosnak minden kocsija mindig azonos színű.

Ebben az esetben a KOCSI egyeden belül a Tulajdonoskód meghatározná a Szín-t. Viszont a Tulajdonoskód értéke nem egyedi, mert több kocsija is lehet valakinek. Ezért érdemes megadni a D 13/1 definíciónak egy lazább változatát is:

D 13/2 **Az E egyedtípusban az A tulajdonságtípus akkor és csak akkor határozza meg funkcionálisan a B-t, ha minden előfordulásban ugyanazt az értéket veszi fel a B, amikor az A értéke ugyanaz.**

Mivel a KOCSI egyed minden előfordulásában más Rendszám érték szerepel, fennáll többek között a Rendszám → Tulajkód függés.

Az FD mindig fordított irányú ***függetlenséggel*** jár. Ha A → B, akkor B –/–> A. Vagyis ha az A meghatározza a B-t, akkor a B nem határozhatja meg az A-t. (A függetlenség jele ebben a könyvben „–/–>" lesz.) Ugyanis abban az esetben, ha a fordított irányú függés is fennállna, akkor ***kölcsönös függésről*** (angolul: mutual dependence) lenne szó. Ennek rövid neve ***MD***, jele pedig „↔". Ha bevezetnénk a kocsikra az egyedi Kocsitípus-név tételt, akkor létezne a Kocsitípus-kód ↔ Kocsitípus-név kölcsönös függés. Végül akkor, ha két tulajdonság között egyik irányban sem tudunk megállapítani függést, akkor ***kölcsönös függetlenségről*** beszélünk. Ennek jele „<–/–>". Példánk esetében a Kocsitípus és a Szín között nincs függés, ezért Kocsitípus <–/–> Szín.

Vegyük észre, hogy az FD két tulajdonság közötti ***hierarchikus viszonyt*** mutat. Ha létezik a Rendszám → Szín függés, akkor a két tétel értékei között M:1 viszony van. Hiszen több kocsinak (M) is lehet azonos a színe, de minden kocsinak csak egy (1) színe van. Hasonló módon az MD két tulajdonság közötti egyedi, más szóval ***lineáris viszonyt*** jelent, amely 1:1 fokú. Később majd ezekre a viszonyokra fogjuk építeni az egyedek közötti összefüggéseket is.

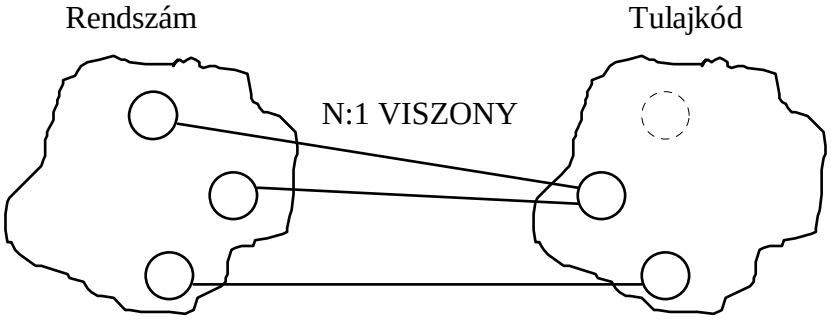
Két tulajdonságtípus funkcionális és kölcsönös függésének illetve függetlenségének az eseteit a 13.1 ábra szemlélteti.

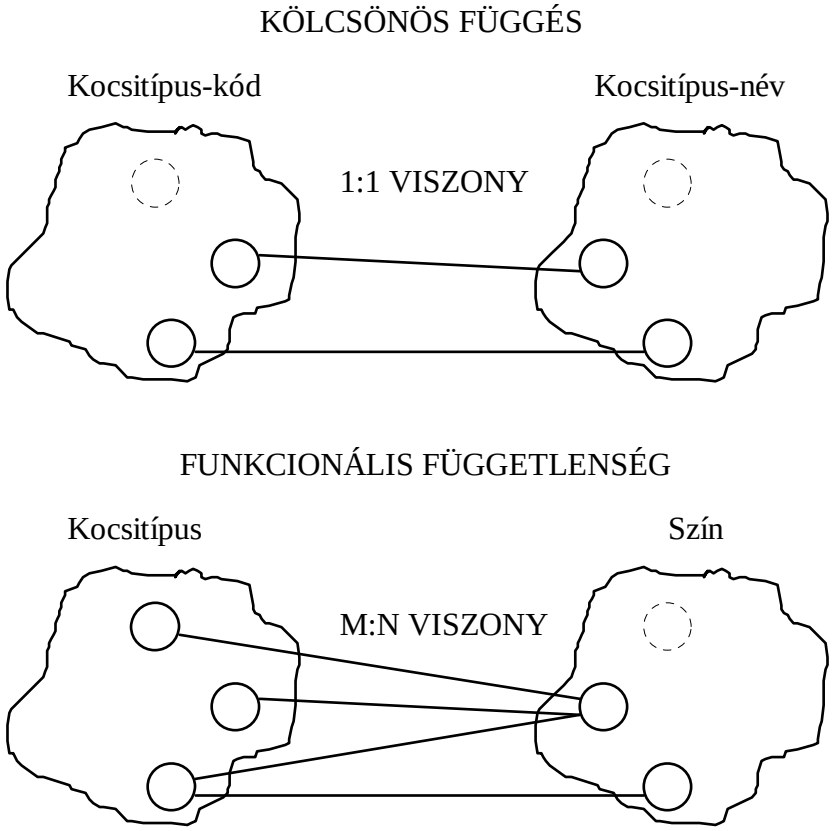
A funkcionális függéseket több szempontból is osztályozhatjuk. Az eddig bemutatott függések ***elemiek*** (angolul: elementary) voltak, mert a meghatározó tulajdonság egyetlen tagból állt. Nem ritka azonban az az eset sem, amikor a meghatározó tulajdonság több tényezőből épül fel. Ebben az esetben ***összetett*** (angolul: composite) függésről beszélünk. Ennek jele könyvünkben A+B+...+N → X lesz, vagyis a meghatározó részeit a „+" jellel fogjuk összekötni. Lásd a 13.2 példát.

13.2 példa

RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, Rendelt-mennyiség, ...)
Rendelésszám+Cikkszám → Rendelt-mennyiség

FUNKCIONÁLIS FÜGGÉS





13.1 ábra: Tulajdonság viszonyok

Az FD-k kapcsán - az ábrázolás miatt - szoktak beszélni a függés ***bal- és jobboldaláról*** is. Ekkor a baloldal a meghatározót, a jobboldal a meghatározottat, a függőt jelenti. Összetett függésről tehát akkor van szó, ha a baloldal több tagból áll.

A függéseket az ún. ***függéserő*** szerint is osztályozzuk. Az A → B függés azt mondja ki, hogy minden A-értékhez csak egy B-érték tartozhat. Azt nem köti ki, hogy minden A-értékhez kell is, hogy tartozzon B-érték. Vannak olyan esetek, amikor a B-érték nem-értelmezhető vagy ismeretlen. Vegyük csak alapul a következő példát:

13.3 példa

KOCSI (***Rendszám***, Kocsitípus, ..., CASCO-kötvényszám)

Vannak kocsik, amelyekre nem kötöttek CASCO-biztosítást, tehát a CASCO-kötvényszám értéke nem-értelmezhető. Ettől függetlenül fennáll a Rendszám → CASCO-kötvényszám FD, mert minden kocsinak csak egy CASCO-biztosítása lehet. A függést ***erősnek*** nevezzük, ha a meghatározóhoz mindig kell tartoznia meghatározottnak és ***gyengének***, ha vannak meghatározók, amelyekhez nem kötődik meghatározott. Tehát a Rendszám → Kocsitípus függés erős, míg a Rendszám → CASCO-kötvényszám FD gyenge. A 13.1 ábrán pontozott kör mutatja azt az értéket, amely nem kötődik a másik tulajdonság egyetlen értékéhez sem.

A 13.3 példa egy másik eszmefuttatásra is alapot ad. Minden kocsinak csak egy CASCO-ja lehet, de ugyanakkor minden CASCO is csak egy kocsira vonatkozhat. Ezek szerint a két tétel között kölcsönös függést kellett volna felfedeznünk? Nem. Ha a CASCO-kötvényszám értéke nem-értelmezhető, akkor maga ez a „nem-értelmezhető" valami is - érték. Ezt a továbbiakban az „NA" (angolul: not applicable) jellel fogjuk jelezni és a feltételes függésnél (ld. 18.6 pont) majd bővebben is kifejtjük. Itt csak arra utalunk, hogy valahányszor a kocsi CASCO-kötvényszám értéke NA, más és más a Rendszám értéke. Ezért nem áll fenn a CASCO-kötvényszám → Rendszám FD, tehát nem beszélhetünk kölcsönös függésről.

A funkcionális függés teljes vagy részleges lehet. Ennek az osztályozási ismérvnek csak akkor van értelme, ha a meghatározó tulajdonság - a baloldal - összetett. A függés akkor ***teljes*** (angolul: full dependence), ha a meghatározott a teljes összetett tulajdonságtól függ és amennyiben abból bármelyik tagot kiemeljük, a függés megszűnik. Ha az összetett tulajdonságból valamelyik tagot kiemelve is fennmarad a függés, akkor azt ***részlegesnek*** (angolul: partial dependence) nevezzük. A 13.4 példa mutatja a kétféle függés lényegét.

13.4 példa

RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, Rendelt-mennyiség, ..., Cikknév)
Rendelésszám+Cikkszám → Rendelt-mennyiség
Rendelésszám+Cikkszám → Cikknév

Az első függés teljes, mert a Rendelt-mennyiséget sem a Rendelésszám, sem a Cikkszám nem határozza meg önmagában. Ezért bármelyiket emeljük ki az összetételből, a függés elveszik. Ezzel szemben a második függés részleges. A Cikkszám → Cikknév függés akkor is fennmarad, ha eltávolítjuk a meghatározóból a Rendelésszámot.

A funkcionális függéssel kapcsolatosan két további dologra kell felhívnunk a figyelmet. Az FD tulajdonképpen nem más, mint az adatbázis szerkezeti ***integritási korlátja***. Ha kijelentjük azt, hogy fennáll például a Cikkszám → Cikknév függés, akkor azzal leszögezzük, hogy az egyedtípus minden egyes előfordulásában csak egyetlen Cikknév-érték szerepelhet minden időpontban, vagyis a vonatkozó egyedtípus bármilyen megvalósításában. Úgy közlünk az egyed kiterjedéséről (extenzió) valamit, hogy tartalmi (intenzió) megkötést alkalmazunk. A modellezés a tartalommal, az intenzionális vetülettel foglalkozik. Viszont igen gyakran csak példák, vagyis az extenzionális vetület segítségével értjük meg az összefüggéseket.

A modellezés feladata a valóság hű tükrözése. Ahhoz, hogy megfelelő szerkezetet alakítsunk ki, ismernünk kell a valós jelenségek ***szemantikai*** összefüggéseit. A fentiek szerint egy korlátot állapítunk meg akkor, amikor feltételezzük például a Rendszám → Kocsitípus függést. Most tegyük fel, hogy a valóságban a leszerelt rendszámot idővel kiadhatják másik kocsinak is. Semmi garancia sincs arra, hogy az új kocsi ugyanolyan típusú lesz, mint a régi. Ezért nem igaz az, hogy a KOCSI egyedtípus „bármilyen megvalósításában" (ld. előző bekezdés) fennáll ez a függés. Mindez arra int bennünket, hogy az FD-k megállapításánál igen körültekintően kell eljárnunk.

13.3 Tartományfüggés

Mint ismeretes, a tulajdonságtípusokat kétféle módon szemlélhetjük. Beszélhetünk róluk adott egyedtípussal összefüggésben, amikor is a tulajdonságot ***attribútumnak*** hívjuk. Azonban a tulajdonságokat egyedektől függetlenül is meghatározhatjuk. Ilyenkor ***(érték)-tartományról***

(angolul: domain, ejtsd doméjn) van szó. A Rendelésdátum a RENDELÉS egyedhez kötött tulajdonság, attribútum. A Dátum pedig önálló értéktartomány, doméjn.

A funkcionális függés az egyedtípus tulajdonságtípusai között értelmezett ***attribútum-függés***. Ezzel szemben a tulajdonságtípusok között az egyedtípusoktól függetlenül is feltárhatunk függési viszonyokat. Ezeket nevezzük doméjn- vagy ***tartományfüggéseknek***.

D 13/3 **A B tulajdonságtípus akkor és csak akkor tartományfüggéssel függ az A tulajdonságától, ha az A tartomány minden értékéhez minden időpontban a B tartománynak csakis egy értéke társítható.**

A tartományfüggés jele „==>", rövid neve ***DD*** (angolul: domain dependence). Az ilyen függésre mindazt elmondhatjuk, amit az előző pontban a funkcionális függéssel kapcsolatosan fontosnak tartottunk megemlíteni. A tartományok között is létezhet függetlenség illetve kölcsönös függés. A DD is lehet elemi vagy összetett, erős vagy gyenge függés. A kérdés az, hogy ha a DD ennyire hasonlít az FD-re, akkor miért van szükség erre a külön függésfajtára?

A választ egy példa segítségével fogjuk ismertetni. Tegyük fel, hogy vállalatunk rendeléseket teljesít. Például tejtermékeket szállít. Egyes vevők központi szállítást kérnek azzal, hogy majd a telephelyeikre - raktáraikba, üzleteikbe, elosztóikba - saját maguk továbbítják az árut. Más vevők azonnal a telephelyre kívánják szállíttatni a termékeket. Végül vannak olyan kis vállalkozások, amelyeknél a központ egybeesik az egyetlen telephellyel. Ilyen a „maszek kisközért". Mindezt a 13.5 példával tudjuk szemléltetni.

13.5 példa

KÖZPONT	(***Központ-azonosító***, ...)
TELEPHELY	(***Telephely-azonosító***, ..., *Központ-azonosító*)
RENDELÉS	(***Rendelésszám,*** *Központ-azonosító, Telephely-azonosító,* ...)

A példa kapcsán a következő ellentmondást fedezhetjük fel: A TELEPHELY egyedben egyértelműen fennáll a Telephely-azonosító → Központ-azonosító funkcionális függés, amennyiben azzal a feltételezéssel élünk, hogy minden telephely csak egy központhoz (vevőhöz) tartozhat. Viszont a RENDELÉS egyedben ugyanez a függés nem érvényesül. Miért nem? Itt kell visszatérnünk a ***gyenge*** függésre. A Rendelésszám → Telephely-azonosító függés gyenge, mert hiszen vannak olyan vevők, akik a központba szállíttatják az árut, mert maguk akarják azt elosztani vagy nincs is telephelyük. Következésképpen a Telephely-azonosító néhány RENDELÉS előfordulásban NA-értéket fog felvenni, vagyis nem-értelmezhető. Így a Telephely-azonosító → Központ-azonosító függés valóban nem érvényes a RENDELÉS egyedben.

Ezek után felvetődik a kérdés, hogy milyen viszonyban is áll egymással a Telephely-azonosító és a Központ-azonosító? Van vagy nincs függés? És egyáltalán miért kérdés ez?

Módosítsuk egy csöppet a 13.5 példát a 13.6 példára!

13.6 példa

KÖZPONT	(***Központ-azonosító***, ...)
TELEPHELY	(***Telephely-azonosító***, ...)
RENDELÉS	(***Rendelésszám,*** *Központ-azonosító, Telephely-azonosító,* ...)

Meg tudjuk mondani, hogy melyik központhoz tartozik az X azonosítójú telephely? Nyilván nem, mert a TELEPHELY egyedből hiányzik a központ azonosítója. Ezért a KÖZPONT és a TELEPHELY egyed nem kapcsolható (ld. átfedéshiány, 12.4.5 alpont). Vajon mi alapján fogjuk felfedezni ezt a strukturális problémát?

Mielőtt a kérdésre válaszolnánk, legyen szabad bevezetnünk a függés egy újabb osztályozását. Ha a Rendelésszám → Központ-azonosító függést a RENDELÉS egyed vonatkozásában tárjuk fel, akkor ***egyeden belüli*** (angolul: intra-entity) függésről beszélünk. Azonban nyilvánvaló, hogy a rendelésekben csak olyan Központ-azonosító érték adható meg, amely a KÖZPONT egyedben is szerepel. Ez is egy, az adatbázisra vonatkozó tartalmi ***integritási korlát***. Azonban miként tudjuk rögzíteni ezt a korlátot? Úgy, hogy alkalmazzuk az ***egyedek közötti*** (angolul: inter-entity) függés fogalmát.

Tehát a Rendelésszám és a Központ-azonosító függése nemcsak a RENDELÉS egyeden belül, hanem a RENDELÉS és a KÖZPONT egyed között is fennáll. Csakhogy az utóbbi esetben nem beszélhetünk funkcionális függésről, hiszen az - definíció szerint (D 13/1 és D 13/2) - egyedtípuson belül értelmezendő. Nos, ezért van szükség a tartományfüggés fogalmára. Ha az egyedek közti függéseket vizsgáljuk, akkor fel fogjuk fedezni a Telephely-azonosító ==> Központ-azonosító tartományfüggést. Ez alapján kiegészíthetjük a 13.6 példa TELEPHELY egyedét a Központ-azonosítóval. Így máris visszakapjuk a 13.5 példa képét.

Összefoglalóan: A tulajdonságok funkcionális függései az egyedeken belüli, a tartományok függései az egyedek közötti viszonyok meghatározásában játszanak szerepet. Gyenge függések esetén a kétféle viszony egymásnak ellentmondhat. A RENDELÉS egyedben nem érvényes a Telephely-azonosító → Központ-azonosító FD. Azért nem, mert a Rendelésszám → Telephely-azonosító FD gyenge. Ugyanakkor általánosan fennáll a Telephely-azonosító ==> Központ-azonosító DD. Ez a függés erős, mert minden telephelynek egy központhoz kell kapcsolódnia.

Mindez felveti a függés egy további osztályozásának az igényét. Erős és gyenge függésről beszéltünk akkor, ha az A meghatározó tulajdonság minden értékéhez kellett tartoznia B értéknek illetve ha voltak olyan A értékek is, amelyekhez nem kapcsolódott B. Tehát eddig a függéserőt csak a „baloldal" szempontjából vizsgáltuk. Áttekintésünk nem lenne teljes, ha az osztályozást nem terjesztenénk ki a „jobboldalra" is.

A szakirodalom általában a funkcionális függéssel van elfoglalva. Ezért a baloldalra, a meghatározóra figyel (ld. D 13/1 és D 13/2) és kevesebbet törődik a jobboldallal, a meghatározottal. A Rendszám → Kocsitípus függésnél azt a kérdést veti fel, hogy tartoznia kell-e minden Rendszám értékhez Kocsitípus értéknek vagy sem. A fordított összefüggést - kell-e minden Kocsitípus értékhez Rendszám értéknek tartoznia - nem vizsgálja. Ez érthető is: a KOCSI egyedben nem szerepelhet Rendszám-érték nélküli előfordulás. Ebben az egyedben minden Kocsitípus-hoz értelemszerűen tartozik Rendszám érték.

Viszont miért ne tarthatnánk nyilván olyan kocsitípusokat (nem a KOCSI egyedben), amelyekhez egy adott időpontban nem tartozik konkrét kocsi? Vegyük csak alapul a következő adatbázis-szerkezetet (13.7 példa).

13.7 példa

KOCSITÍPUS	(***Kocsitípus***, Férőhely, Gyártó, ...)
KOCSI	(***Rendszám***, Szín, *Kocsitípus*, ...)

Az első egyedben nyilvántarthatjuk a Lamborghini típust annak dacára, hogy ma még nincs ilyen konkrét kocsi előfordulásunk adatbázisunk második egyedében. Mivel most már két egyedtípusunk van, nemcsak a Rendszám —> Kocsitípus FD-re kell tekintettel lennünk, hanem értékelnünk kell a Rendszám ==> Kocsitípus DD-t is. Ezért fel fogjuk vetni a kérdést, hogy minden típushoz kell-e konkrét kocsinak tartoznia?

A konkrét válasz az alkalmazási körülményektől, igényektől függ. Eldönthető, hogy csak olyan típust tartunk nyilván, amelyhez konkrét kocsi is tartozik, vagy vezetünk olyan típusismereteket is, amelyek még/már nem kötődnek tényleges kocsi-előfordulásokhoz. Ezért a függés erejét nemcsak baloldalról, hanem jobboldalról is vizsgálnunk kell.

Ezek szerint a tartományfüggés jobboldalról ***erős***, ha az A ==> B függésen belül minden B értékhez kell, hogy tartozzon A érték. A DD jobboldalról ***gyenge***, ha létezhet olyan B érték, amely nem párosul A értékkel. Például a Telephely-azonosító ==> Központ-azonosító függés jobboldalról gyenge, mert létezhetnek olyan központok, amelyekhez nem kapcsolódik telephely. Viszont az A Cikkszám ==> Cikknév függés jobboldalról erős, mert nincs értelme olyan cikknév vezetésének, amelyhez nem tartozik cikkszám.

Vagy van? Az adatelemzésnek nem az a feladata, hogy a valóság értelmességét vizsgálja. Az ismeretek célszerű ***szemantikai*** viszonyait a felhasználók közösségének kell meghatározniuk. Az adatmodellező feladata az, hogy az igényeknek hűen megfelelő struktúrát alkosson. Ehhez fel kell fedeznie a tényezők összefüggéseit. Ha a felhasználók olyan cikkek neveit is nyilván akarják tartani, amelyeknek még nincs cikkszáma, akkor más modellel kell előrukkolni, mint amikor csak Cikkszámmal ellátott tételek neveit vezetjük.

13.4 Nem-normalizált egyedtípusok

A mindennapi életben gyakran van szükség arra, hogy egy adott jelenség vonatkozásában valamilyen ismeretet megtöbbszörözzünk. Álljon itt példaként az a mindennapos eset, amelyben a személyek nyelvtudására vonatkozó adatokat kívánjuk vezetni. Márpedig vannak olyan személyek, akik egynél több nyelvben is járatosak. Lásd a 13.2 ábrát.

SZEMÉLY

Törzsszám	...	Név	Nyelv	Vizsgaév	Vizsgaszint
AAA		Kovács L.	Angol Francia Olasz	1972 1978 1981	felső közép felső
BBB		Szabó P.	Olasz	1976	felső

13.2 ábra: A SZEMÉLY egyed részlete

13.4.1 Az ismétlődés és káros hatása

Az ábrából kitűnik, hogy egyes egyedelőfordulások sorában a Nyelv tulajdonságtípus több értéket is felvesz. Az olyan esetekben, amikor egy tulajdonságtípus az egyedelőfordulásokra több értékkel is rendelkezik, vagyis egy név alatt több tartalmat adunk meg, ***ismétlődő adatról*** beszélünk. Akkor pedig, ha az ismétlődő adatok egymással rendszerezett összefüggésben állnak - lásd a Nyelv, Vizsgaév, Vizsgaszint együttesét -, ***ismétlődő csoportról*** (angolul: repeating group) van szó.

Az ismétlődő adat illetve csoport számos probléma forrása. Ezek a következők:

- Meg kell adni a maximális ismétlésszámot. Ha ez a nyelv esetében három, akkor a negyedik nyelv kimutatására nincs mód egy adott személynél.
- Ha valaki a maximálisnál kevesebb nyelvet beszél, feleslegesen foglaljuk a tárolót. (N.B.: Az üres érték is tárt igényel és kezelési időtöbblettel jár.)

- Ha az egyes nyelvek fix helyhez rendeltek, például mindig az angol az első nyelv, akkor nemcsak személyenként, hanem összesen is csak három nyelvet lehet kezelni.
- Ha viszont a nyelvek nem fix helyhez rendeltek, akkor az olaszul beszélők kikeresése nehézkes.
- A kezelés egyensúlytalan. Hamar megtaláljuk, hogy ki milyen nyelveket beszél, de nehezen leljük meg az adott nyelvet beszélő személyeket.

Már a 12.4.7 alpontban felhívtuk a figyelmet az ilyen szerkezetek bajaira. A felsorolt problémák miatt az adatmodellben nem lenne szabad meghatározni olyan egyedtípust, amely ismétlődő adatokat vagy csoportokat tartalmaz. Az ilyen struktúrákat ki kell küszöbölni. A kérdés az, hogy mi alapján fedezzük fel ezeket a rossz szerkezeteket és miképpen történjék azok felszámolása?

13.4.2 Az ismétlődés elvi alapja

Az előző alpont első bekezdésével szemben az ismétlődésnek van egy olyan meghatározása is, amely a tulajdonságok közötti függésekkel kapcsolatos:

D 13/4 **Az egyedtípus ismétlődő ismeretet tartalmaz, ha létezik olyan tulajdonsága, amely funkcionálisan független az azonosítótól.**

A funkcionális függés feltételezi, hogy a meghatározó értékéhez csak egy függő érték tartozik. Ezért nyilvánvaló, hogy nem áll fenn például a Törzsszám —> Nyelv függés. A SZEMÉLY egyed nem megfelelő szerkezetű. Át kell alakítanunk. Mielőtt ennek módját kifejtenénk, megadjuk az átszerkesztés elméleti alapját.

D 13/5 **Az egyedtípus nem-normalizált, vagyis 0NF alakú, ha vannak tulajdonságai, amelyek az azonosítójától funkcionálisan függetlenek.**

Az egyedtípus szerkezeti állapotát ***normálformának*** (angolul: normal form) hívjuk. A normálalak a tervezési minőséget mutatja. Mint látni fogjuk, többféle normálforma létezik. Ezért a normálalakokat (NF) számmal minősítjük (xNF). A „nulladik" normálforma (0NF) azt jelenti, hogy az egyed egyáltalán nem normalizált, mert a normalizált állapot feltételezi, hogy az egyed tulajdonságai funkcionálisan függnek az azonosítótól.

13.4.3 A normalizálás első lépése

A feladatot már ismerjük. Az egyedből el kell távolítani minden olyan tulajdonságot, amely az azonosítótól funkcionálisan független, azaz többértékű. A SZEMÉLY egyedben a Nyelv, a Vizsgaév és a Vizsgadátum többértékű, tehát másutt van a helye.

Példánk esetében a kézenfekvő megoldás az, hogy meghatározunk egy külön NYELVTUDÁS egyedet, amely az eredeti SZEMÉLY egyed ismétlődő tulajdonságait tartalmazza. A célszerű szerkezet első változatát a 13.3 ábra szemlélteti.

SZEMÉLY

Törzsszám	...	Név
AAA		Kovács L.
BBB		Szabó P.

NYELVTUDÁS

Törzsszám	***Nyelv***	Vizsgaév	Vizsgaszint
AAA	***Angol***	1972	felső
AAA	***Francia***	1978	közép
AAA	***Olasz***	1981	felső
BBB	***Olasz***	1976	felső

13.3 ábra: Az ismétlődő tulajdonságok kiküszöbölése

Amikor az ismétlődő adatokat az eredeti egyedből az újba helyezzük, azokat mindig ***ki kell egészíteni az eredeti egyed azonosítójával*** (Törzsszám). Ennek két oka van. Egyrészt enélkül a tétel nélkül az ismétlődő adatokat tartalmazó új egyednek nem lenne egyértelmű azonosítója. Másrészt elvesznének az ismeretek közötti összefüggések. Az eredeti egyedben a személyekhez kötődtek a nyelvtudásra vonatkozó ismeretek. Ezt a kapcsolódást az új tervrészletben sem szabad elveszteni.

Az ismétlődő adatok kiküszöbölése néha új adat bevezetésével is jár. Ennek nem modell-elméleti, hanem gyakorlati okai vannak. Az új egyed (itt: NYELVTUDÁS) azonosítója ugyanis mindig összetett: az eredeti egyed elsődleges kulcsát egészítjük ki egy másik, célszerűen kiválasztott tulajdonsággal. Viszont nem szeretjük, ha az azonosító hosszú. Ezért esetleg bevezetünk egy rövidebb tulajdonságot, amint azt a 13.4 ábra mutatja. Hangsúlyoznunk kell, hogy a Nyelv maga is egyértelmű, azonosításra alkalmas tulajdonság volt. Ezért a Nyelvkód alkalmazása már nem a tulajdonképpeni normalizálás része.

SZEMÉLY

Törzsszám	...	Név
AAA		Kovács L.
BBB		Szabó P.

NYELV

Nyelvkód	...	Nyelvnév
01		Angol
04		Francia
07		Olasz

NYELVTUDÁS

Törzsszám	***Nyelvkód***	Vizsgaév	Vizsgaszint
AAA	***01***	1972	felső
AAA	***04***	1978	közép
AAA	***07***	1981	felső
BBB	***07***	1976	felső

13.4 ábra: Az új struktúra kiegészítése

Bárki meggyőződhet arról, hogy új megoldásunk a 13.4.1 alpontban ismertetett problémák mindegyikét kiküszöböli. Egy személyhez tetszőleges számú nyelvtudást lehet kötni. A NYELV egyedből kiindulva éppen ugyanolyan módon lehet visszakeresni az adott nyelvet beszélő személyeket, mint a SZEMÉLY egyedből kiindulva azt, hogy egy személy milyen nyelvek ismeretével rendelkezik. Ezért a kezelés abszolút egyensúlyos. (N.B.: A tagok sorrendje az összetett azonosítókon belül közömbös. A Törzsszám+Nyelv összetett kulcs két része elvileg ugyanúgy érhető el. Csak a papír természete miatt kell az egyik tagot a másik elé írni.)

A fenti példával szemben vannak helyzetek, amelyekben az ismétlődő adatok leválasztásával nyert új egyednek nincs olyan tulajdonsága, amely betölthetné az azonosító szerepét. Erre mutat példát a 13.5 ábra, amely a 12.3 ábra egyik részlete némileg új formában.

KÁR

Kárszám	Dátum	Típus	Tulajkód	Károsszeg
23456	05.14	Lada	134567	X
		BMW	134568	Y
23656	06.06	Polski	136567	Z
24456	06.30	Lada	134567	Q

13.5 ábra: Ismétlődést tartalmazó KÁR egyedtípus

Amint látjuk, a Típus, a Tulajkód és a Károsszeg tartalma ismétlődik. Tehát a KÁR egyed 0NF alakban van és ezért azt át kell alakítani. Ha a normalizálási szabályoknak megfelelően járunk el, akkor az ismétlődő adatokat új egyedtípusba tesszük és azt kiegészítjük az eredeti egyedtípus azonosítójával. Az időleges megoldást a 13.6 ábra mutatja.

KÁR

Kárszám	Dátum
23456	05.14
23656	06.06
24456	06.30

EGYEDI-KÁR

Kárszám	Típus	Tulajkód	Károsszeg
23456	Lada	134567	X
23456	BMW	134568	Y
23656	Polski	136567	Z
24456	Lada	134567	Q

13.6 ábra: A KÁR egyedtípus megbontása

A művelet elvégzése után megdöbbenve tapasztaljuk, hogy az új EGYEDI-KÁR tételben nem találunk megfelelő azonosítót. Az új kulcs egyik része a Kárszám, hogy tudjunk kapcsolni a KÁR egyed felé, de mi lesz a másik része? Hiszen egy káreseményben részt vehetnek azonos típusú kocsik. Az sem kizárt, hogy egy tulajdonos két azonos típusú kocsija megy egymásba. Az EGYEDI-KÁR egyedtípusban egyszerűen nem találunk megfelelő kulcsot.

Ilyenkor a hagyományos normalizálási eljárással nem jutunk sehová sem. Már pedig egyedtípus nem létezhet jó azonosító nélkül. Ezért nem normalizálási, hanem ***szemantikai megoldáshoz*** kell folyamodnunk. Mielőtt ezt kifejtenénk, rá kell mutatnunk, hogy milyen tipikus körülmények között találkozhatunk ezzel a problémával.

Ez a gond az ún. ***kimenetorientált*** (lásd 12.5.4 alpont) egyedek megbontásakor szokott fellépni. Az adott szempontra kihegyezett kimenetekben nem mindig van szükség a jelenségek egyedi behatárolására. Hiszen a kimenetnek lehet éppen az a célja, hogy valamilyen leíró jellemző szerint rendezze el az adatokat. Esetünkben valakik a károk típusok szerinti osztályozására voltak kíváncsiak. A konkrét kocsik ismereteiben nem voltak érdekeltek.

A példa világosan mutatja, hogy nem szabad az adatmodellben kimenetorientált egyedeket tervezni. A kimenetet elemezni kell és fel kell bontani az azon lévő ismereteket a valóságos jelenségeknek megfelelően. Ennek során gyakran ütközünk az azonosíthatatlanság problémájába. A 13.6 ábra szerkezetének a pontosítására nincsen semmiféle matematikai- vagy modellezéstechnikai eljárás. Az adatbázis tervezésének egyik apró titka, hogy vannak normalizálhatatlan szerkezetek. Az ilyenek esetében magunknak kell a jelenségek természetes összefüggéseinek az alapján megkeresni a megfelelő azonosítót.

Nem lesz nehéz rájönnünk arra, hogy példánkban a Rendszám tulajdonságra van szükség. Az ugyanis világos, hogy a károknak nem a tulajdonosok és főleg nem a kocsitípusok a közvetlen részesei, hanem maguk a kocsik. A kocsikat pedig feltételezésünk szerint a rendszám azonosítja. Ezért az időlegesen jó megoldást a 13.7 ábra mutatja.

KÁR

Kárszám	Dátum
23456	05.14
23656	06.06
24456	06.30

KÁR/KOCSI

Kárszám	***Rendszám***	Típus	Tulajkód	Károsszeg
23456	***ABC 134***	Lada	134567	X
23456	***BCD 265***	BMW	134568	Y
23656	***DEF 896***	Polski	136567	Z
24456	***ABC 134***	Lada	134567	Q

13.7 ábra: A kiegyensúlyozott KOCSI-KÁR modellrészlet

A 13.7 ábra terve csak „időlegesen jó”. Jó, mert kiküszöbölte az ismétlődést és egyensúlyos modellrészletet adott. A KÁR/KOCSI alapján ugyanúgy tudunk válaszolni a „Milyen kocsik vettek részt a 23456 számú kárban?” kérdésre, mint a „Melyik károk érték az ABC 134 rendszámú kocsit?” felvetésre. Csak időlegesen kedvező, mert szembetűnő a táblázatban az „ABC 134 - Lada -134567” értéksor fizikai redundanciája.

Ezzel a problémával majd a következő fejezetben foglalkozunk. Most még rá kell mutatnunk arra, hogy az ismétlődés lehet többszörös, sőt beágyazott is. Fel kell tárnunk azt is, hogy néha az ismétlődés rejtett formákat ölt.

13.4.4 További tudnivalók az ismétlődésről

A normalizálás első lépése nem mindig ilyen egyszerű. Sokszor előfordul, hogy az eredeti, a megbontandó egyedtípus többféle ismétlődést tartalmaz. Természetesen ekkor a megbontással több új egyedtípus születik. Az eredetileg elképzelt SZEMÉLY egyedben lehetnek a címre, a nyelvtudásra, a képzettségre, a pótlékokra stb. vonatkozó ismétlődő adatok. Ezek eltávolításával egymáshoz képest mellérendelt CÍM, NYELVTUDÁS, KÉPZETTSÉG, PÓTLÉK stb. egyedeket kapunk.

Az is megtörténik, hogy az eredeti ismétlődés levágásával nyert új egyed további ismétlődést tartalmaz. Tehát az eredeti egyed egy-egy előfordulása nem egyszerűen ismétlődő adatot, hanem adat-hierarchiát tartalmazott. Például a SZEMÉLY egyed Bér adatcsoportja havonta ismétlődik, de a havi bér sora is több részsorra bomlik mondjuk Jogcím szerint. Ekkor a normalizálás első lépése után a BÉR és a JOGCÍM egyedtípusokat kapjuk, amelyek közül az előbbi az utóbbi fölérendeltje.

Most pedig térjünk át a problémákra. A fejlesztők és a felhasználók egyik rossz szokása az ***ismétlések elrejtése***. Ezt különböző adatnevek alkalmazásával teszik a 13.8 ábra szerinti módon.

SZEMÉLY

Törzsszám	...	Név	Nyelv-1	Nyelv-2	Nyelv-3
AAA		Kovács L.	Angol	Francia	Olasz
BBB		Szabó P.	Olasz	-	-
CCC		Molnár E.	Spanyol	-	-

13.8 ábra: Nevek mögé rejtett ismétlés

A tervező meg van elégedve a 13.8 ábra megoldásával mondván, hogy itt nincs normalizálási probléma, hiszen minden tulajdonságtípus minden egyedelőfordulásra csak egyszeres értéket vesz fel. Nincs adatismétlődés. Persze a kicsit is járatos tervező azonnal felfedezi ezt a trükköt és ugyanúgy felszámolja a rossz adatszerkezetet, mint a nyílt ismétlések esetében. Azért, mert a rejtett ismétlés pontosan azokat a problémákat veti fel, mint a nyílt.

Az ismétlődés elrejtésének a fenti módja intenzionális, amennyiben a tulajdonságtípusok körét növelte a tervező. A másik módszer extenzionális, vagyis az egyedelőfordulások számát emeli a fejlesztő. Például a 13.5 ábra első - kétsoros! - előfordulásának a második sorát kiegészíti a Kárszám és a Dátum első sorbeli értékével. Ezzel persze visszakapja az 12.3 ábra KÁR egyedét. Amely, mint tudjuk rossz volt, hiszen nincs egyedi azonosítója. Tehát az a KÁR egyed nem reláció. Tanulság: kiegészítéssel sohasem szabad normalizálni.

Egy további rossz szokás az ***ismétlések kiiktatása***. Itt megint csak a SZEMÉLY egyedet tudjuk felhozni példaként. Nagyon sok helyen vezetik a személyek Legmagasabb-végzettség nevű tulajdonságát. Egyszeres tartalommal, ismétlődés nélkül. Holott eléggé világos, hogy egy-egy személynek lehet két vagy több teljesen azonos „rangú" legmagasabb végzettsége. Mivel a fejlesztő és a felhasználó csak egyszeres értéket enged meg, a teljes ismeret nem kezelhető és a bevitt adat is tévesen tájékoztat. Mert ha valaki mérnök is meg közgazdász is, akkor csak az egyik ismeret kerül az adatbázisba.

Ha egy tulajdonság a természeténél fogva ismétlődő tartalmú, akkor azt külön egyedtípusban kell tükrözni. Az előző példa esetében fel kell venni a SZEMÉLY-VÉGZETTSÉGE egyedet, amelyben személyenként tetszőleges számú végzettség kezelhető. Egy kis titok: Az okos tervező a minősítő jelzők alapján fedezi fel a kiiktatott ismétléseket. A „legmagasabb" jelző máris utal arra, hogy több végzettség van. Hasonló alapon bukkanhatunk rá az olyan ismételt adatokra, amelyeknek a nevében „legutolsó", „fő", „előző" stb. szerepel. Nem azt állítjuk, hogy minden ilyen esetben azonnal új egyedtípusban kell gondolkodni. Hanem azt, hogy a tervezőnek utána kell néznie annak, hogy valóban elegendő-e például a (közvetlen) Előző-munkahely vezetése, vagy pedig több korábbi munkahely adatára is szükség van.

A harmadik tervezői „gyerekbetegség" az ***értékkombinálással*** történő ismétlődés-álcázás. Mivel észrevesszük, hogy egy személynek több végzettsége is van, összetett kódokat vezetünk be. Az X kód jelenti a mérnököt, az Y a közgazdát, a Z pedig azt, aki mérnök is és közgazda is. Pedig az ilyen megoldások logikus következményei közismertek. A kódok kombinációit egy idő után már senki sem tudja nyomon követni, a programok pedig egyre bonyolultabbak és nehezebben módosíthatók lesznek. Mert ha Z mérnököt is jelent, akkor ugyebár programmal kell kikeresnünk az X és a Z (majd és a Q, később és a W stb.) végzettségű személyeket, ha a mérnökök adataira vagyunk kíváncsiak.

Ezen a ponton megfogalmazhatjuk az adatbázistervezés egyik legfontosabb titkát, sőt, ha úgy tetszik legfontosabb törvényét:

Sohasem szabad az adatszerkezetet egyszerűsíteni a programbonyolultság rovására.

Ez a törvény összefügg az ún. ***százszázalékos elvvel*** (angolul: 100% principle), amely szerint minden az ismeretekre vonatkozó ismeretet csakis az adatmodellben szabad megfogalmazni (ld. 7.2 pont). Az adatatok ugyanis sokkal könnyebben változtathatók, mint a programok. Semmit sem vesztünk azzal, ha létrehozunk a ***tiszta*** gondolkodás jegyében egy VÉGZETTSÉG egyedet, amelyben nem kell kódokat kombinálni, amely tetszőlegesen bővíthető és módosítható mindenféle programkarbantartás nélkül.

13.5 A normalizálás lényege

Ebben a fejezetben éppen csak, hogy megindultunk a normalizálás hosszú és rögös útján. Ám máris begyűjthettünk magunknak egy sor fontos tapasztalatot. Mivel jellegük szerint az első következtetéseink meg fognak ismétlődni a későbbiek során is, érdemes röviden összegezni eddigi tudásunk lényegét.

A szakirodalom jelentős része azt a látszatot kelti, hogy a normalizálás tisztán ***matematikai*** eljárás, amely szinte mechanikusan végrehajtható a megfelelő összefüggések felfedezése után. (E könyv szerzője töredelmesen bevallja, hogy néhány régebbi kiadványában saját maga is ezt a hamis nézetet sugallta. Ám az a korszaka már rég elmúlt.) A 13.6 ábra modellje illetve a 13.4.4 alpontban megvilágított esetek nagyon egyértelműen mutatnak rá arra, hogy a matematikai alapú normalizálás az adatbázistervezésnek csak az egyik - igaz: nagyon hasznos és könnyen érthető - aspektusa.

Viszont nagyon gyakran találkozunk olyan helyzetekkel, amelyekben matematika nem segít és saját ésszerű megoldást kell találnunk. Ilyenkor nincs más lehetőségünk, mint magához a valósághoz fordulni és a dolgok összefüggéseinek a természetét kutatni. Végeredményben ***szemantikai*** elemzést kell végrehajtanunk.

Az adatmodellben lehetnek az értelmezésre visszaható ***strukturálási*** és a struktúrát befolyásoló ***értelmezési*** hibák. A következő fejezetekben ki fogjuk mutatni, hogy sokszor csak szemantikai elemzéssel lehet javítani az adatmodellt, máskor pedig a matematikai elemzés hívja fel a figyelmet alapvető értelmezési problémákra. Vagyis a matematikai és a szemantikai tervezési eszköztár nem kizárja, hanem kiegészíti egymást.

Ezért ebben a könyvben a ***normalizálás*** nem azt fogja jelenteni, mint más kiadványokban. Nem pusztán a normalizálás mechanikus lépéseinek a bemutatása a célunk. Fel fogjuk tárni az emberi elemzés általunk ismert egyéb momentumait is. Ezért, úgy véljük, hogy adatbázistervezési módszerünket jogosan nevezzük ***szemantikus normalizálásnak***.

13.6 Normalizálás és struktúra

Mint fentebb is említettük, az „igazi” normalizálás még előttünk van. A 0NF alakban lévő egyedtípusok nem-normalizáltak, a szó matematikai értelmében még ***relációnak*** sem tekinthetők. A reláció ugyanis kétdimenziós tábla, amely eleve feltételezi, hogy az oszlopok és sorok találkozásánál csakis elemi értékek szerepelnek. Normalizálni csak relációt lehet. Ezért eddig tulajdonképpen csak azt mondtuk el, hogy miként tesszük egyedtípusainkat - relációkká.

Azonban már a 0NF alakú egyedtípusok felszámolása is alkalmat ad számunkra ahhoz, hogy feltárjunk bizonyos olyan szerkezeti összefüggéseket, amelyekre a későbbiek során is érdemes lesz figyelni.

Amikor az ismétlődő tulajdonságokat kiemeljük egy egyedtípusból, nem teszünk mást, mint helyreállítjuk az ismeretek természetes rendjét. Térjünk csak vissza a 13.2 ábra példájára! Az ember és a nyelv két különböző jelenség annak dacára, hogy az ember beszéli a nyelvet. A szerző bánatosan bevallja, hogy még szuahéli nyelven sem tud. Ennek ellenére nem kétséges, hogy a szuahéli nyelv létező valóság. A személyek és a nyelvek közötti összefüggés a dolgok elvitathatatlan természete szerint M:N fokú. Egy személy több (N) nyelvet tud és ugyanazt a nyelvet több (M) személy használja. Ezért akkor, amikor a Nyelv tulajdonságot a SZEMÉLY egyedhez kötjük, a jelenségek természetes hálójából mesterséges hierarchiát, tehát 1:N fokú viszonyt kreálunk.

Csak arra figyelünk, hogy egy (1) személy több (N) nyelvet beszél. A másik oldalról elfeledkezünk.

Ezért valóban igaz, hogy amikor az ismétlődő tulajdonságokat kiemeljük az egyedtípusból, akkor az ismeretek „természetes rendjét" állítjuk helyre. A szemantikus normalizálás ennél sem kevesebbet, sem többet nem jelent. Célja az ismeretek természetes rendjének a megtalálása.

A normalizálás kihat az egyedtípusok ***belső szerkezetére***, azaz tulajdonságtípusainak a sorára. Az egyik egyedtípusból elveszünk néhány tulajdonságtípust és azokat a másikba illesztjük. Ennek a műveletnek a során óhatatlanul átalakítjuk az egyedtípusok ***külső szerkezetét***, azaz kapcsolataik együttesét is. Ha a SZEMÉLY egyedből kiemeljük a tulajdonságokat és létrehozzuk a NYELVTUDÁS egyedet, akkor az eredeti és az új egyedtípus között kapcsolatot kell teremtenünk.

Vegyük észre, hogy mindez mennyire szépen elrendezett elvek szerint történik. Az ismétlődő tulajdonságok elkülönítésével nyert új egyed mindig

- összetett azonosítójú úgy, hogy az azonosító egyik része az eredeti egyed elsődleges kulcsa, a másik része pedig az ismétlődő adatok azonosítására alkalmas tulajdonság;
- az új egyedben az első kulcsrész kapcsoló szerepet tölt be, vagyis az eredeti egyedhez kapcsolja az újat;
- ezért az eredeti egyed az újjal 1:N fokú viszonyban áll;
- ez a kapcsolat birtoklási jellegű;
- a kapcsolat kötelező vagy opcionális attól függően, hogy az eredeti egyedtípus minden előfordulásában volt-e értéke az azonosító másik részének;
- amennyiben az azonosító másik részének az abszolút szerepe is azonosító (lásd 13.4 ábra), úgy az új egyed azzal is 1:N fokú kapcsolatban áll és így két, hálós viszonyban lévő jelenség között teremt természetes összefüggést;
- amennyiben az azonosító másik részének az abszolút szerepe nem azonosító, úgy az új egyed az ismeretek természetes hierarchikus elrendezését szolgálja.

A kitételek közül csak az utóbbi érdemel külön magyarázatot. Mert az világos, hogy a NYELVTUDÁS egyed azonosítója összetett (Törzsszám+Nyelvkód); a kulcs első része az eredeti SZEMÉLY egyedhez kapcsol; ez a kapcsolat 1:N fokú, mert egy személy több nyelven is beszélhet; opcionális, mert vannak személyek, akik nem beszélnek idegen nyelvet; az azonosító másik része azonosító abszolút szerepű és így a NYELV egyedhez 1:N fokú viszonnyal kapcsol, aminek következtében létrejön a SZEMÉLY és a NYELV egyedtípusok közötti, a NYELVTUDÁS által közvetített M:N-es viszony.

Néha előfordul, hogy az ismétlődő adatok leválasztásával nyert új egyed azonosítójának a másik része nem kapcsol harmadik egyed felé. Vegyük csak például az Előző-munkahely adatot, amely a SZEMÉLY egyedtípus tulajdonsága. A tervező utánanéz a dolgoknak és kideríti, hogy nemcsak a közvetlen megelőző munkahely kimutatása szükséges, hanem jó lenne adatbázisban tárolni az összes korábbi munkahely bizonyos adatait is. Ezért kreál egy ELŐZŐ-MUNKAHELY egyedet, amelyben a Törzsszám a SZEMÉLY felé kapcsol.

A felhasználók csak az előző munkahely pénzügyi adataiban érdekeltek. Nem kíváncsiak magára a(z előző) munkahelyre. Nem akarnak ismereteket vezetni arról. Tehát a munkahely nem „ismeretekkel leírandó jelenség", nem egyed. Nem lesz MUNKAHELY egyedtípusunk. A tervező egyébként sem tudná azonosítókkal ellátni a világ összes lehetséges előző munkahelyét. Ezért az ELŐZŐ-MUNKAHELY egyed azonosítójának másik részeként a foglalkoztatás kezdő dátumát választja. Ez nem kapcsol a nem-létező MUNKAHELY egyedhez. Így az új egyedtípus nem hálós viszonyt teremt, hanem kifejezi azt a természetes hierarchiát, hogy egy személynek több korábbi munkahelye is lehetett.

13.7 Ismétlődés és kétségek

Ezt a fejezetet egy ugyancsak komoly fejtörést okozó probléma ismertetésével kell zárnunk. A 13.8 ábra kapcsán említettük, hogy a tervezők nem mindig veszik észre a rejtett ismétlődéseket és így, a szerkezetet nem normalizálva, rossz adatmodellt alkotnak. Csakhogy nem mindig olyan könnyű annak feltárása, hogy rejtett ismétlődésről vagy pedig egészen más helyzetről van-e szó.

KIKÜLDETÉS-1

Kiküldetés	...	Szállás	Étkezés	Dologi
AAA		X Ft	A Ft	L Ft
BBB		Y Ft	B Ft	M Ft
CCC		Z Ft	C Ft	N Ft

KIKÜLDETÉS-2

Kiküldetés	***Költségtípus***	Költség
AAA	***Szállás***	X Ft
AAA	***Étkezés***	A Ft
AAA	***Dologi***	L Ft
BBB	***Szállás***	Y Ft
BBB	***Étkezés***	B Ft

13.9 ábra: Az „utazó ügynök” problémája

A 13.9 ábra példája ismét elgondolkodtat bennünket arról, hogy lehet-e mechanikusan adatbázistervet készíteni. A személyek kiküldetéséről van szó. A kiküldött személy három költségtérítést kap: szállásköltséget, étkezésköltséget és egyéb dologi, például utazási, költséget. Most tessék eldönteni, hogy a 13.9 ábra melyik megoldása a jobb. Az, amelyben a költségek egymás mellett, külön tulajdonságnévvel (intenzionálisan) jelennek meg, vagy az, amelyben egy név alatt, pusztán értékekként (extenzionálisan) tűnnek fel.

Mi okozza a gondot? Az egyik tervező azt gondolhatja, hogy az első megoldás extenzionálisan (nevek mögé) rejtett ismétlődést tartalmaz (ld. 13.4.4 alpont). Az ismétlődést le akarja vágni. Ekkor viszont a sorok azonosíthatatlanok lesznek. Ezért vezeti be a Költségtípus új tulajdonságot a második megoldásnak megfelelően. Végeredményben a tulajdonságtípus nevekből (Szállás stb.) tulajdonság-értékeket kreál. Viszont a másik tervező szerint az első megoldás úgy jó, ahogyan van, mert szó sincs ismétlődésről. A 13.8 ábrában a Nyelv-1, -2 és -3 értelme ugyanaz. Ezzel szemben a Szállás költség nem azonos értelmű a másik két költségfélével.

Melyik tervezőnek adjunk igazat? Annak idején egészen konkrétan ez a példa akarta megingatni a tervezőknek a normalizálásba és a modellelemzésbe vetett hitét.

Mi egyértelműen az első megoldás hívei vagyunk. A Szállás és az Étkezés költség más célt szolgál. Nem cserélhetők fel; nem vihető az egyik érték a másik helyére; nem kérdezhető le az egyik tartalom a másik helyett stb. Ezért - szemben a nyelv példájával - itt nem lépnek fel az ismétlődés által okozott problémák.

Persze a második változat hívei azzal fognak érvelni, hogy már akár holnap is felléphet egy új költségtípus. Ebben az esetben az ő megoldásukban nincs szükség szerkezeti módosításra. Hiszen az új tartalom (pl. Reprezentáció költség) értékként, sorbővítésként, extenzionálisan jelenik meg. Ugyanakkor az első verzióban a szerkezetet ki kell egészíteni az új tulajdonságtípussal, ami strukturális, tehát intenzionális módosítást jelent.

A korábbi maximálisan három nyelvvel szembeni negyedik, ötödik stb. nyelv bevezetése esetén ez az érvelés valóban meggyőző. Azonban senki sem gondolhatja, hogy a reprezentációs költséget ugyanúgy fogják kezelni, mint a szállásköltséget. Ha azt tennék, akkor nem volna értelme a különböző megjelöléseknek (Szállás szemben Reprezentáció). A tervezőknek ki kell békülniük azzal, hogy az egyedtípusok időnként új tulajdonságtípusokkal bővülnek. Ez a legegyszerűbb, legkönnyebben végrehajtható szerkezeti változtatás.

A második megoldás mellett érvelők a jelenségeket - egyedtípusokat - csupán zártan, önmagukban, egymástól szeparáltan vizsgálják. Ez a hagyományos normalizálás egyik legnagyobb hibája. Mert gondoljunk csak jobban bele a példába! Hol tükröződnek magának a kiküldetésnek az adatai? Az, hogy kit, hová, mettől-meddig stb. bocsátottak útra? Az első megoldás esetében mindezeket az ismereteket maga az egyed tartalmazhatja. A második változat esetében viszont szükségessé válik egy külön KIKÜLDETÉS egyed. Tehát nemcsak új tipizáló tulajdonságot (Költségtípus), hanem új egyedet, sőt új kapcsolatot is maga után von a második verzió. Hiszen a KIKÜLDETÉS és a KIKÜLDETÉS-2 egyed között viszonyt kell teremteni.

Az „utazó ügynök” speciális esete rávilágít egy általános adatbázistervezési problémára. A tervezők hajlamosak arra, hogy elfeledkezzenek a jelenségek összefüggéseiről és a dolgokat egymástól elkülönítve vizsgálják. Kis példánkkal attól akarjuk óvni őket, hogy a normalizálás nagyszerű technikáját mechanikusan használják. A másik fontos tanulság az, hogy egyszerű és tiszta szerkezetekre kell törekedni. Kerülni kell az új, mesterséges tényezők felvételét a modellbe. Azok nélkül is lesz elegendő elemzési gondunk, amint azt a következő fejezetből mindjárt meglathatjuk.

Ellenőrző kérdések - 13

13/01 Az irányítószámmal azonosított helységek megyékben helyezkednek el. Minden megyében lehet több település és minden település csak egy megyében van. (Most tekintsünk el Budapesttől.) Az alábbi képletek közül melyik helyes? Adja meg a helyes válasz(ok) sorszámát.

- Irányítószám → Megyekód
- Irányítószám ↔ Megyekód
- Irányítószám –\–> Megyekód
- Irányítószám <–\– Megyekód

13/02 Az Irányítószám → Településnév függés milyen jellegű a TELEPÜLÉS egyeden belül? Mindkét oldalról gyenge (G), mindkét oldalról erős (E), csak bal oldalról erős (B), csak jobb oldalról erős (J). Gondoljon arra, hogy nem csak a településeknek van irányítószáma.

13/03 Adott a Gyerekkód és az Apakód tulajdonság. Adja meg a két tétel közötti viszony jelét akkor, ha a gyerek természetes apjáról van szó (A) és akkor, ha válás vagy egyéb ok miatt a kölöknek több papája is lehet (B).

13/04 Adja meg a függési viszonyok jelét a következő tulajdonságpárosokra:

Raktárkód/Cikk-kód - adott cikk több raktárban is lehet
Raktárkód/Cikk-kód - adott cikk mindig csak egy raktárban tárolódik
Rendelésszám/Cikk-kód - a rendelés egytételes
Rendelésszám/Cikk-kód - a rendelés többtételes
Irányítószám/Kerületkód - Budapesten
Férjkód/Feleségkód - csak az aktuális házasság érdekel bennünket
Férjkód/Feleségkód - minden volt és létező házasság érdekel bennünket

13/05 Az Irányítószám azonosítja a településeket. Milyen jellegű az Irányítószám ==> Településnév doméjnfügggés? Mindkét oldalról gyenge (G), erős (E), csak jobboldalról erős (J), csak baloldalról (B) az.

13/06 Adott a RENDELÉS (***Rendelésszám***, ..., Vevőkód) egyed. Milyen természetű a Rendelésszám és a Vevőkód függése? A helyes válasz számát kérjük.

- Egyeden belüli.
- Egyedek közti.
- Mindkettő.

13/07 Tegyük fel, hogy megrendeléseink általában egytételesek. Ezer megrendelés között csak egy van, amely több cikkre is vonatkozik. Ön szerint mi a Rendelésszám és a Cikkszám viszonya? A helyes válasz számát kérjük.

- A Cikkszám meghatározza a Rendelésszám-ot.
- A Rendelésszám-tól függ a Cikkszám.
- A két tétel egymástól független.
- A két tulajdonság hol függ, hol nem függ egymástól.

13/08 Megengedhető-e (M) vagy sem (S) a következő adatbázisterv? A számla a személy által az adott hónapban befizetett összegek summája.

SZEMÉLY (***Törzsszám***, ..., Páros-havi-számla, Páratlan-havi-számla)

13/09 Adja meg az előző példa helyes megoldását!

14. ALAPVETŐ NORMÁLFORMÁK

14.1 A normalizálás alapjai

Az adatmodellezés egyik fő célja az optimalizálás (ld. 12.6 pont), vagyis az adatmodellt alkotó egyedtípusok lehető legjobb ***belső és külső szerkezetének*** a megkeresése. Könyvünk első részében rámutattunk arra, hogy ez a két struktúra kölcsönösen összefügg. Az egyedtípus tulajdonságtípusainak a sora a kapcsoló szerepű tulajdonságokon át meghatározza az egyed kapcsolattípusainak az együttesét és fordítva: a kapcsolatok befolyásolják azt, hogy az egyednek milyen tulajdonságai lesznek.

Az optimális adatmodell kialakítására - egyéb technikák mellett - a ***normalizálás*** szolgál. A normalizálás az a műveletsor, amellyel kialakítjuk az egyedtípusok lehető legjobb ***normálformáját*** (NF). Eredeti értelmében a normálforma az egyed belső szerkezetének a minőségét mutatja és csak közvetve befolyásolja a külső szerkezetet [21]. A hagyományos normalizálási eljárás során egyenként állapítjuk meg az egyedtípusok normálalakjait és nem foglalkozunk az egyedek közötti viszonyokkal. Mára már a normalizálás és a normálforma jelentése kibővült, de ezt a témát majd csak később fejthetjük ki.

Ebben a részben az eredeti normálalakok közül a legegyszerűbbeket mutatjuk be. Az 1NF, 2NF és 3NF formákról lesz szó. Ezeket majd a megfelelő pontokban részletezzük. Itt előzetesen néhány általános összefüggésre kell felhívnunk a figyelmet.

A normálformák - a felsoroltak és az azokat követő BCNF, 4NF illetve 5NF alakok is - „jósági szinteket” jelentenek és ***egymásba skatulyázottak***. Ezen két dolgot kell érteni. Egyrészt azt, hogy például a 2NF alak matematikai értelemben jobb, mint az 1NF; a 4NF jobb, mint a BCNF; az 5NF pedig a legjobb forma. Másrészt azt, hogy például a 3NF alakú egyed szükségszerűen 1NF és 2NF alakú is. Tehát ezek a normálalakok nem függetlenek egymástól, hanem logikusan egymásra épülnek.

Ebben a fejezetben az adatmodell alapvető hibáit, azok elméleti szerkezeti okait, a hibák kiküszöbölésére alkalmas normálalakokat és az ezek elérésére szolgáló normalizálási lépéseket tárjuk fel azok eredményeivel együtt.

Az alábbiakban és a következő fejezetben a normalizálás lényegét lépésenként, a normálformák „jósági szintjei” szerint mutatjuk be. Az 1NF alak után ismertetjük a 2NF formát stb. Ennek a megközelítésnek az egymásraépültség miatt csak ***didaktikai*** okai vannak. Jó, ha a tervezők egyszer elméletileg megértik az egyes alakokhoz tapadó problémákat külön-külön is. ***Pragmatikusan*** a normalizálást nem lépésenként hajtjuk végre. Nem egyenként szüntetjük meg az előzetesen elképzelt adatmodell szerkezeti hibáit. Ha egy egyedtípus egyszerre mutatja az 1NF és a 2NF alakokhoz kapcsolódó problémákat, akkor azokat igyekszünk egycsapásra felszámolni. Pontosan ehhez az azonnali gyakorlati áttekintéshez szükséges a normálalakok jó elméleti ismerete.

14.2 A második normálforma

Térjünk vissza az eredeti kocsi példához. A 13.7 ábrán mutattuk be, hogy a nem-normalizált KÁR egyedtípust miként bontottuk meg KÁR és KÁR/KOCSI egyedekre. Az utóbbi tételt fogjuk tovább vizsgálni, ezért az esetet itt megismételjük. Lásd 14.1 ábra.

KÁR

Kárszám	Dátum
23456	05.14
23656	06.06
24456	06.30

KÁR/KOCSI

Kárszám	***Rendszám***	Típus	Tulajkód	Károsszeg
23456	***ABC 134***	Lada	134567	X
23456	***BCD 265***	BMW	134568	Y
23656	***DEF 896***	Polski	136567	Z
24456	***ABC 134***	Lada	134567	Q

14.1 ábra: Egyedtípus 1NF alakban

A KÁR/KOCSI egyedtípus minden előfordulása minden tulajdonságtípusra csak egyetlen értéket vesz fel, vagyis nem tartalmaz ismétlődő tulajdonságot.

D 14/1 **Az egyedtípus akkor és csak akkor van legalább első normálformában (1NF), ha minden nem-kulcs tulajdonságtípusa funkcionálisan függ az azonosítótól.**

A KÁR/KOCSI egyedtípus Kárszám+Rendszám összetett tulajdonsága azonosítóként funkcionálisan meghatározza az összes többi tulajdonságot, hiszen a páros minden értékéhez csak egy Típus, Tulajkód és Károsszeg érték kapcsolódik. Ezért az egyed 1NF alakban van. Amint látjuk, az 1NF alakú egyedben nem lépnek fel azok a problémák, mint a nem-normalizáltban. Tehát a 14.1 ábra két egyedet felölelő megoldása jobb, mint az eredeti változat, amely egyetlen KÁR egyedtípusban vegyítette a magára a káreseményre vonatkozó egyszeres illetve a kocsikat egyenként érő és így szükségszerűen többszörös (ismétlődő) ismereteket (ld. 12.3 ábra).

A jobbítás dacára azonnal észrevesszük, hogy az új szerkezet még mindig messze nem tökéletes, mert fizikai redundanciát von maga után (ld. az „ABC 134 - Lada - 134567” értékrészsor többszörös megjelenését). Ezért az egyedet tovább kell normalizálnunk. Mielőtt ezt megtennénk a definíció két furcsa kitételére kell magyarázatot adnunk.

A „***legalább***” kitétel azt mutatja, hogy vizsgálataink adott szintjén az egyed elérte a megnevezett normálalakot. Elemzésünk jelenlegi fázisában csak annyit tudunk a 14.1 ábra KÁR egyedtípusáról, hogy az”legalább” 1NF alakú. Mivel még más összefüggéseket nem tekintettünk át, nem kizárt, hogy az egyed ténylegesen magasabb normálformájú, de arról még nem tudha-

tunk. Többet jelenleg nem is feltételezhetünk, hiszen az egyéb normálalakokat be sem mutattuk. A következő fejezet végén az olvasó konstatálhatja, hogy a most „legalább" 1NF alakúnak feltételezett KÁR valójában ötödik normálformában - vagyis optimális alakban - van. (Árulkodás egy apró titokról: a maximum két tulajdonságtípussal rendelkező egyedtípusok egy igen szélsőséges és általunk nem tárgyalt eset kivételével mindig 5NF alakúak. Ezért azok hagyományos normalizálására soha sincs szükség.)

A ***„nem-kulcs"*** (angolul: non-key) jelző eléggé félrevezető. Mégis kénytelenek vagyunk alkalmazni, mert ezt használják a szakirodalomban. A valóságban a következőt jelenti: Egy egyedtípusban lehet több olyan tulajdonság is, amely az egyed minden más tulajdonságát funkcionálisan meghatározza. Nos ezeket a tényezőket hívják kulcsnak. Ez azért félretájékoztató, mert nem csak az egyed általunk feltételezett elsődleges kulcsáról van szó. Ha a VEVŐ (***Vevőkód***, Vevőnév) egyedben a Vevőnév értéke egyedi lenne, akkor azt is kulcsnak (= meghatározónak) feltételezné a definíció. Következésképpen annak függéseit a normalizálás első lépései során nem vizsgálnánk. Ráadásul a „nem-kulcs" jelző még azt is jelenti, hogy a tulajdonság nem is része a kulcsnak. Mivel példánkban a Kárszám és a Rendszám kulcsrész, azok függései nem tartoznak az 1-3NF elemzésének a körébe.

A „nem-kulcs" jellegnek a BCNF alakig (ld. 15.2 pont) nem lesz jelentősége. Azért kellett már most beszélnünk róla, mert az 1-3NF alakok „hivatalos" definícióiban szerepel ez a kitétel. A ***kulcsrészek*** függéseinek az elemzését a hagyományos normalizálás egyébként is teljesen elhanyagolja. Ebből számos gond fakad, amelyekre majd a BCNF formánál, de még inkább a 16. fejezetben térünk ki. Itt megjegyezzük, hogy van némi elméleti alapja annak, hogy a kulcsrészek függéseire nem fordítanak kellő figyelmet.

A ***projektívitási*** Armstrong-szabály [20] szerint az összetett kulcs értelemszerűen meghatározza saját magának minden egyes összetevőjét. Tehát ha létezik egy A+B azonosító, akkor törvényszerűen fennáll az A+B $\rightarrow$ A és az A+B $\rightarrow$ B függés, ha akarjuk, ha nem. Evidens, hogy minden Kárszám+Rendszám értékpároshoz csak egy Kárszám és csak egy Rendszám érték tartozhat. Ha megadjuk az egészet, akkor ismerjük a részt is.

Még egyetlen dolgot kell megemlítenünk, mielőtt belevágnánk az „igazi" normalizálásba. Az ismétlődést tartalmazó „egységet" nem-normalizált (ld. 13.4 pont) egyedtípusnak neveztük és annak alakját a „0NF" jelöléssel illettük. Mármost az ismétlődést tartalmazó egyed a szó matematikai értelmében nem is reláció. Ezért elvileg nem lehet normálformája és elméletileg nem is normalizálható. A 14.1 ábra megoldása nem normalizálással született, ha a normalizálásnak az ebben a fejezetben használt értelmét tekintjük (ld. 14.2.2 alpont). Ezért a „0NF" formula csak analógia: az egyed 1NF alak előtti helyzetét, annál rosszabb minőségét mutatja. (N.B.: Egyesek használják az „N1NF" - not first normal form - megjelölést is.)

14.2.1 A részleges függés és következményei

A 13.2 pontban már ismertettük a részleges függés lényegét. A C tulajdonság részlegesen függ az A+B tulajdonságtól, ha az A+B $\rightarrow$ C függés mellett fennáll az A $\rightarrow$ C és/vagy a B $\rightarrow$ C függés is.

D 14/2 **Az E egyedtípus C tulajdonsága akkor és csak akkor függ részlegesen az A+B összetett azonosítótól, ha a C-t az A vagy a B is meghatározza.**

A KÁR/KOCSI egyedben az összetett azonosító meghatározza ugyan a Kocsitípus és a Tulajdonoskód tételeket, de azok függenek a Rendszámtól önmagától is.

Az ilyen részleges függés számos probléma forrása. Ezeknek a bajoknak egy része az egyedek karbantartásával függ össze. Azonban a KÁR/KOCSI egyed karbantartásáról nincs értelme

beszélni, mert az eseményeket leíró egyedek nem változhatnak. (N.B.: A hibajavítást tervezési szempontból nem tekintjük karbantartásnak.) Ezért a problémák szemléltetésére egy másik példát alkalmazunk. Lásd a 14.2 ábrát.

RENDELÉSTÉTEL

Rendelésszám	***Cikkszám***	...	Cikknév	Mennyiség
123456	***XXX***		csavar-A	X
123456	***YYY***		alátét	Y
123656	***XXX***		csavar-A	Z
123656	***ZZZ***		karmantyú	Q

14.2 ábra: Részleges függést tartalmazó RENDELÉSTÉTEL egyed

Mivel a Cikknév tulajdonság függ a teljes azonosítótól, de ugyanakkor annak Cikkszám részétől is, a Rendelésszám+Cikkszám → Cikknév függés részleges, amiből az alább leírt ún. anomáliák (visszásságok) fakadnak:

- Ha a Cikknév tulajdonságtípus más egyedet is jellemez - ezzel vagy más megnevezéssel -, akkor nyílt vagy rejtett logikai átfedést okoz (ld. 12.4.1 és 12.4.3 alpontok).
- A Cikknév tartalma fizikailag is redundáns (ld. 12.4.6 alpont) és így feleslegesen fogyasztja a tárat. Pl. többször tartalmazza az adatbázis az „XXX - csavar-A” párost.
- A redundancia miatt a Cikknév változásakor többszörös, időt rabló módosításra van szükség. Például akkor, ha a „csavar-A” név „csavar-B”-re változik.
- Ha a Cikknév más egyedet nem jellemez, akkor törlési anomália lép fel. Ha töröljük azt az utolsó rendelést, amely az adott cikkre vonatkozik, akkor elveszik a cikk nevének az ismerete is. Pl. az „123656” rendelés törlésekor megszűnik a „ZZZ - karmantyú” páros, ha csak ebben a rendelésben kértek ilyenféle cikket.
- Ha a Cikknév más egyedhez nem kapcsolódik, akkor beviteli anomáliával kell számolni. A Cikknév ismeretet nem tudjuk tárolni az olyan cikknél, amelyre nem vonatkozik rendelés. Ha nem kértek csapágyat, akkor nem vihetjük be a „QQQ - csapágy” ismeretpárost.

Az utolsó kitételt kicsit bővebben is megmagyarázzuk, felhívva a figyelmet a tervezők egy tipikus rossz szokására. A RENDELÉSTÉTEL kulcsa összetett. Az azonosítókra vonatkozó szabályok szerint az azonosító illetve annak részei nem lehetnek üres vagy ismeretlen értékűek. Ha tehát nem ismert a Rendelésszám, akkor nem vihetjük be önmagában a Cikkszám és a Cikknév páros tartalmát egy új cikkre vonatkozóan a RENDELÉSTÉTEL egyedbe.

Ezen a mindennapos problémán néhány „ravasz” tervező úgy próbál segíteni, hogy „ideiglenes” vagy „ál” azonosító-értékeket ad ki. Például kiegészíti a 14.2 ábra tábláját a „999001 - ... - QQQ - csapágy - 0” sorral. Mármost az ideiglenes értéket másként kell kezelni, mint a valódit. Tehát a „ravasz” tervező a programbonyolultság rovására próbálja meg egyszerűsíteni az adatszerkezetet, ami - mint már tudjuk - tilos. A jó adatbázis egyik alapvető titka a tisztaság. Hamis értékekkel nem pótolhatjuk a valós szerkezetet.

A fentebb felsorolt anomáliák miatt a RENDELÉSTÉTEL egyedet át kell alakítanunk a D 14/2 meghatározásra alapozva úgy, hogy kiszűrjük a bajokat okozó részleges függést.

14.2.2 A normalizálás második lépése

Ebben a fejezetben az eredeti normalizálási eljárásról lesz szó, amelyet ***normálforma dekompozíciónak*** neveznek. Ez a módszer onnan kapta a nevét, hogy az eredetileg feltételezett egyedtípus tulajdonságsorát megbontjuk. Ezt az ún. ***kivetítés*** (angolul: projection) műveletének módosított változatával tesszük.

A tulajdonképpeni kivetítés a tényleges adatbázison - tehát nem a modellen - végzett művelet. Két lépésből áll. Az első fázisban kiemeljük az eredeti egyedből (RENDELÉS-TÉTEL) a szükséges tulajdonságok (pl. Cikkszám és Cikknév) teljes (!) oszlopait egy új egyedbe. Előfordulhat, hogy ekkor az új egyedben azonos sorokat („XXX - csavar-A") kapunk. Ezért a második lépésben az azonos sorok közül csak egyet hagyunk meg, mivel egy egyedtípusnak nem lehet két teljesen egyező előfordulása. A kivetítés nem érinti az eredeti táblázatot, amely változatlan marad. Az új CIKK egyedet a 14.3 ábra mutatja.

CIKK

Cikkszám	Cikknév
XXX	csavar-A
YYY	alátét
ZZZ	karmantyú

14.3 ábra: Az alapkivetítés eredménye

Az adatmodellen végzett kivetítés némileg másként történik. Ennek a műveletnek van egy tartalmi és egy technikai mozzanata. Az első kérdés az, hogy melyik tulajdonságokat vigyük új táblázatba. A normalizálás első lépéseiben mindig a rossz függésű - esetünkben a részlegesen meghatározott - tulajdonságot (Cikknév) emeljük ki annak közvetlen meghatározójával (Cikkszám) együtt az új egyedtípusba (CIKK). Az új egyedből eltávolítjuk a dupla sorokat. Így a 14.3 ábra megoldásához jutunk. A valódi kivetítés műveletével szemben technikailag annyi az eltérés, hogy a helytelen függésű tulajdonság oszlopát végleg eltávolítjuk az eredeti egyedtípusból. Az tehát nem marad változatlan. A normalizálás eredményét a 14.4 ábra mutatja.

CIKK

Cikkszám	Cikknév
XXX	csavar-A
YYY	alátét
ZZZ	karmantyú

RENDELÉSTÉTEL

Rendelésszám	***Cikkszám***	...	Mennyiség
123456	***XXX***		X
123456	***YYY***		Y
123656	***XXX***		Z
123656	***ZZZ***		Q

14.4 ábra: A RENDELÉSTÉTEL normalizálása

Könnyű meggyőződni arról, hogy a 14.4 ábra megoldása esetében az előző alpontban felsorolt problémák megszűnnek. A Cikknév tárolása már csak egyszeres és így karbantartása is az. A cikk nevét be tudjuk vinni attól függetlenül, hogy vonatkozik-e az adott cikkre rendelés. Viszont akkor sem veszik el a cikknév ismeret, ha a cikk utolsó rendeléstételét töröljük. Tehát már csak a Cikknév esetleges logikai átfedésére - az előző alpont listájának az első problémájára - nem adtunk megnyugtató magyarázatot.

Mielőtt arra kitérnénk illik közreadnunk a 14.1 ábra példájának a helyes megoldását is. A Típus és a Tulajkód csak részlegesen függ a Kárszám+Rendszám összetett azonosítón. Mindkettőt közvetlenül meghatározza a Rendszám önmagában is. Ezért ezt a két tételt ki kell emelni a KÁR/KOCSI egyedből a Rendszám kulccsal együtt egy új egyedtípusba. A Rendszám természetesen az eredeti egyedben is megmarad. A normalizálási lépés eredményét a 14.5 ábra mutatja.

KOCSI

Rendszám	Típus	Tulajkód
ABC 134	Lada	134567
BCD 265	BMW	134568
DEF 896	Polski	136567

KÁR/KOCSI

Kárszám	***Rendszám***	Károsszeg
23456	***ABC 134***	X
23456	***BCD 265***	Y
23656	***DEF 896***	Z
24456	***ABC 134***	Q

14.5 ábra: Részleges függéstől mentes KOCSI-KÁR modellrész

14.2.3 Normálforma és szemantika

Természetesen egy egyedben többféle részleges függés is előfordulhat. Ilyenkor azok mindegyikét ki kell szűrni. Az eredményezett egyedtípusok elvileg már nem tartalmazhatnak további részleges függést. Ennek okát egy általános egyednek és feltételezett függéseinek az esetén magyarázzuk meg. A 14.1 példa mutatja a feladatot.

14.1 példa

E1 (***A+B+C***, D, E, F ...)

A+B+C $\rightarrow$ D, E, F
A+B $\rightarrow$ E, F
A $\rightarrow$ F

Érdemes elsajátítani azt a modellezési technikát, hogy a (kettőnél) többszörösen összetett kulcsok esetében mindig először az önálló résztől (A) függő tulajdonságokat emeljük ki és csak

azután térünk át a páros (A+B), hármas stb. kulcsrészektől függő tételek vizsgálatára. Tehát mindig a legközvetlenebb függést kell alapul vennünk. Ha nem így járnánk el, akkor az F tulajdonság először az A+B azonosította egyedbe kerülne. Csak akkor vennénk észre azt, hogy az F az A+B-től is részlegesen függ. Tehát a lépések szigorú betartásával meghosszabbítanánk a normalizálás folyamatát.

A normalizálás eredményét a 14.2 példa mutatja. Az eredeti egyedben csak az egyetlen nemrészlegesen függő D tulajdonság marad. A megbontással nyert második egyed nem tartalmazza az F tulajdonságot, mert azt nyilvánvalóan eleve a harmadik egyedbe visszük.

14.2 példa

E1 (***A+B+C***, D ...)
E2 (***A+B***, E ...)
E3 (***A***, F ...)

Most pedig térjünk át egy másik titokra. A normalizálási feladatot sohasem szabad önmagában, a többi tényezőtől elszigetelten értelmezni. Akkor sem, ha a hagyományos normalizálási eljárás mindig csak a rossz normálformájú egyed dekompozíciójára koncentrál. Mert az ***egész modell*** szempontjából teljesen más a teendő akkor, ha a KÁR/KOCSI megbontása (ld. 14.5 ábra) során már létezik KOCSI egyedtípus, mint akkor, ha nem létezik.

A kivetítést követően többféle helyzet fordulhat elő. Érdemes az eseteket sorra venni. A RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, ... Cikknév) példát fogjuk használni a magyarázathoz. A 14.4 ábrában alkalmazott megoldás szerint a Cikknév tulajdonságot kiemeltük az eredeti egyedből a Cikkszám meghatározóval együtt. A kérdés az, hogy mi ezek után a valódi teendő.

A ***mechanikusan*** normalizálók a következő módon gondolkoznak:

- Ha még nem létezik a kivetítéssel nyert CIKK egyedtípus, akkor azt létre kell hozni a 14.4 ábrának megfelelő módon, Cikkszám azonosító és Cikknév leíró tulajdonsággal.
- Ha már van CIKK egyedtípusunk, de abban nem szerepel a Cikknév tulajdonság, akkor nem kell létrehozni új egyedtípust, hanem a meglévőt kell a kiemelt leíró tulajdonsággal kiegészíteni.
- Ha már a CIKK egyedtípus tartalmazza a Cikknév tulajdonságot, akkor egyszerűen nincs teendő: a kivetítés csak a részlegesen függő tulajdonságnak (Cikknév) a normalizált egyedből való elhagyását jelenti.

Ez a gondolatmenet alapvetően és elméletileg helyes. Azonban feltételezi a gyakorlatilag feltételezhetetlent, miszerint a modell tényezőit eleve egyértelműen határozták meg. A valóságban többféle hiba merülhet fel a kivetítés után. A helyzetek a következők:

1.eset: Van már CIKK egyedünk, de annak nem a Cikkszám, hanem a Cikkazonosító a kulcsa. Egyszerűbb helyzetben ***azonosító-szinonimákról*** van szó. Ekkor az egyik nevet elsődlegesnek kell választani, a másikat pedig arra kell lecserélni a modell minden egyedtípusában. A bonyolultabb esetről majd a 14.7 pontban szólunk.

2.eset: Van már olyan - nem feltétlenül CIKK nevű - egyedünk, amelynek a Cikkszám az azonosítója, de az nem ugyanazt jelenti, mint a RENDELÉSTÉTEL egyedben lévő Cikkszám. (Ugyanezt a helyzetet könnyebb elképzelni a Rendelésszám esetében, mert hiszen vannak vevői és szállítói rendeléseink és - tévedésből - mindkettőnek a Rendelésszám azonosítót adhattuk.) Ekkor tehát ***azonosító-homonimákra*** bukkantunk. Ilyenkor az egyik vagy mindkét tulajdonságot át kell nevezni (Vevő-rendelésszám és Szállító-rendelésszám) és ezt az átnevezést végig kell vezetni a modell minden vonatkozó egyedtípusában.

3.eset: Cikknév nevű tulajdonság nincs a CIKK egyedben. Van viszont Cikk-megnevezés tétel. Amennyiben a kettő tartalma szemantikailag teljesen azonos, úgy ***leíró-szinonimákkal*** állunk szemben, amit az 1.esetnél ismertetett módon meg kell szüntetni. Viszont előfordulhat az is, hogy a két tényező nem teljesen azonos. Mivel valamilyen értelemben vett redundanciáról van szó, az egyik tételt valószínűleg ekkor is fel kell számolni, de ehhez felhasználói konzultáció szükséges.

4.eset: A helyzet szemléltetésére egy újabb tulajdonságot alkalmazunk. Tegyük fel, hogy az Ár nevű tulajdonság is jellemzi a RENDELÉSTÉTEL egyedet. A tervező azt is a Cikkszámtól részlegesen függőnek tartja. Ha az Ár szerepel a CIKK egyedben is, a mechanikus normalizálás jegyében azt egyszerűen elhagyja a RENDELÉSTÉTEL-ből. Ezzel pedig nagy hibát követhet el. Az Ár lehet ***leíró-homonima***, amely a CIKK egyedben egységárat, a RENDELÉSTÉTEL-ben kialkudott árat jelent. Ebben az esetben pedig egyáltalán nem szabad normalizálni, mert ismeretet vesztünk. Csak a homonim neveket kell kiküszöbölnünk egyértelmű megnevezések által.

Amint látjuk, a normalizálás nem lehet mechanikus. ***Szemantikus normalizálásra*** van szükség. Ez azt jelenti, hogy a megfelelő normálforma keresése közben - az egyedtípusok tényleges átalakítása előtt - mindig meg kell vizsgálnunk a kapcsolódó tényezők nevét és értelmét, törekedve a homonimák illetve szinonimák kiszűrésére.

14.2.4 A második normálforma és a modellstruktúra

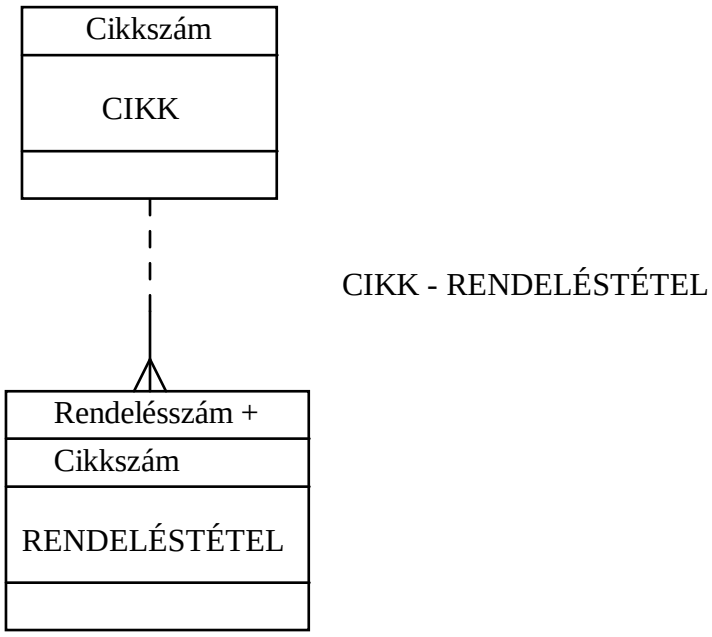
Ha egy egyedtípus már 1NF alakú, vagyis nem tartalmaz ismétlődést, akkor megvizsgálandó a részleges függés. Természetesen erre csak akkor van szükség, ha az egyed azonosítója összetett, mert részleges függés a definíció szerint (ld. D 14/2) csak ekkor léphet fel.

D 14/3 **Az egyed akkor és csak akkor van legalább 2NF alakban, ha minden nem-kulcs tulajdonsága teljes függéssel függ az azonosítójától.**

A 2NF kialakítása kivetítéssel történik (ld. 14.4 ábra). A művelet egyaránt kihat az egyed belső és külső szerkezetére. A normalizálással nyert új egyedtípus (CIKK) azonosítója (Cikkszám) mindig az eredeti egyed (RENDELÉSTÉTEL) azonosítójának a része. Ebből következően az utóbbi egyedben kapcsoló szerepű tulajdonság és a két egyed között kapcsolattípus áll fenn. Ennek jellemzői az alábbiak:

- A kapcsolatot a részleges függést okozó tulajdonság (Cikkszám) hordozza kapcsolótételként.
- Az új egyed a régivel 1:N fokú kapcsolatban áll, tehát annak fölérendeltje.
- A létrejövő kapcsolat alulról mindig kötelező, mert a kapcsoló tulajdonság kulcsrész, tehát nem lehet üres vagy ismeretlen értékű. Minden rendeléstételnek ismert cikkhez kell kapcsolódnia.
- A kapcsolat birtoklási jellegű. A cikkek rendeléstételekkel „bírnak”.

A strukturális összefüggéseket a 14.6 ábra mutatja.



14.6 ábra: A normalizálással nyert RENDELÉSTÉTEL modellrész

14.3 A harmadik normálforma

Mondanivalónk szemléltetésére most az eredeti példa (ld. 12.3 ábra) KOCSI egyedtípusát fogjuk felhasználni. Lásd a 14.7 ábrát.

KOCSI

Rendszám	Típus	Tulajdonosnév	Foglalkozás	Telephely	Főhatóság	Férőhely
ABC 134	Lada	Kovács Rózsa	Keramikus K	Budapest	-	5 fő
BCD 265	BMW	AB Kft	-	Szeged	-	5 fő
DEF 896	Polski	Kovács Rózsa	Mérnök M	Pécs	-	4 fő
FGH 333	Lada	XY vállalat	-	Budapest	Z	5 fő

14.7 ábra: 2NF alakú KOCSI egyed

Mivel a KOCSI egyedben nincs ismétlődő tulajdonság és az egyed Rendszám azonosítója nem összetett, a KOCSI a D 14/3 meghatározás szerint legalább második normál alakú.

Azonban mindenki láthatja, hogy a táblában teljesen azonos ismeret-részsorok találhatók, vagyis a KOCSI egyed fizikailag redundáns (ld. 12.4.6 alpont). Ahány Lada típusú kocsiról vezetünk ismeretet, annyiszor kell megadni a „Lada - 5 fő” párost. Ezért a tervet át kell alakítani.

14.3.1 A tranzitív függés és következményei

Ennek a fizikai átfedésnek az elvi alapját a harmadik Armstrong-szabályban [20] kell keresnünk. Ez az ún. ***tranzitivitási szabály***, amely szerint: Ha együtt fennáll az A $\rightarrow$ B és a B $\rightarrow$ C függés, akkor ezekből törvényszerűen következik az A $\rightarrow$ C függés is. Ha ezt az elvet az egyedtípus tulajdonságtípusaira alkalmazzuk, akkor megfogalmazhatjuk a tranzitivitás definícióját:

D 14/4 **Az E egyedtípus nem-kulcs C tulajdonsága akkor és csak akkor tranzitívan függ az egyed A kulcsától, ha meghatározza az azonosítótól függő B tulajdonság is.**

A 14.7 ábrában a Rendszám meghatározza a Típus és a Férőhely tulajdonságot, viszont a Típus önmagában is meghatározza a Férőhelyet. Ezért a KOCSI egyed tranzitív függést tartalmaz.

A tranzitív függés pontosan ugyanazokat az anomáliákat okozza, mint a részleges függés (lásd 14.2.1 alpont). Ezért azokat nem lenne ildomos itt újra feszegetni. Mivel azonban a következmények igen súlyosak, ismét elmagyarázzuk a problémák lényegét. Ehhez egy új példát választunk, amelyben a tulajdonságok változékonyak, szemben a 14.7 ábra Férőhely tételével. Lásd a 14.8 ábrát.

RENDELÉS

Rendelésszám	…	Vevőkód	Vevőnév
123456		111	X
123458		222	Y
123656		333	Z
123656		111	X

14.8 ábra: 2NF alakú RENDELÉS egyedtípus

A helytelen belső szerkezetű egyedek tervezésével megszegjük az ***egyértelműség*** és a ***teljesség*** követelményeit. Az „X” vevőnevet többször tároljuk. Ez egy kisebb és egy nagyobb bajt okoz. Holnap a vevő a nevét „Q”-ra változtatja. Ekkor az összes „111” vevő-kódú egyed-előfordulásban át kell írni a nevet. Ez a kisebbik baj. Ha a módosítást valamelyik előfordulásban elmulasztjuk, akkor az „111” vevőt hol „X”, hol „Q” névvel fogjuk illetni. Sőt, holnapután a „222” vevő „X-re” változtathatja a nevét. Ekkor már papírjainkon semmi sem különbözteti meg a két vevő megnevezését. Az ilyen balgaságok miatt nem jut el a csekk a címzetthez, viszont azt megkapja más, aki abban nem is illetékes.

A rosszul tervezett egyed törlési és beviteli anomáliában is szenved. Most térjünk vissza a 14.7 ábra példájára! Ha az adatbázisból töröljük az „BCD 265” rendszámú kocsit és feltételezzük, hogy az volt az utolsó BMW, akkor elveszik a „Férőhely = 5 fő” ismeret. Megfordítva: Már tudjuk, hogy az új „Rover-B” kocsi 6 fős. Ezt az ismeretet nem tudjuk az adatbázisba vinni, mert nincsen rendszámunk (kulcsértékünk).

Persze mindezeket a bajokat nem a teljes mértékükben mutatják mintapéldáink. Világos, hogy a kocsiknak nemcsak típusa és férőhelye, hanem márkaneve, lóereje, köbcentije, súlya stb. is van. Ha mindezen ismereteket a KOCSI egyed tárolja, akkor a többszörözés problémája, az inkonzisztencia veszélye, a törlési és beviteli anomália sokkal súlyosabban mutatkozik.

14.3.2 A normalizálás harmadik lépése

A tranzitív függés bajokat okoz; azt ki kell szűrni. A kérdés az, hogy miként? Ismét a kivetítés műveletéhez folyamodunk. Az egyedtípusból el kell távolítani a tranzitívan függő tételeket. Ezt természetesen megint úgy tesszük, hogy új egyedtípusba visszük leíróként a helytelen függésű - kétszeresen meghatározott - tulajdonságot, azonosítóként pedig annak meghatározóját. A 14.9 ábra mutatja a 14.8 ábra példájának a megbontását.

VEVŐ

Vevőkód	Vevőnév
111	X
222	Y
333	Z

RENDELÉS

Rendelésszám	…	*Vevőkód*
123456		*111*
123458		*222*
123656		*333*
123656		*111*

14.9 ábra: A RENDELÉS egyedtípus megbontása

A teljesség kedvéért megmutatjuk a 14.7 ábra KOCSI egyedének a normalizálását is. Lásd a 14.10 ábrát.

KOCSITÍPUS

Típus	Férőhely
Lada	5 fô
BMW	5 fô
Polski	4 fô

KOCSI

Rendszám	*Típus*	Tulajdonosnév	Foglalkozás	Telephely	Főhatóság
ABC 134	*Lada*	Kovács Rózsa	Keramikus K	Budapest	-
BCD 265	*BMW*	AB Kft	-	Szeged	-
DEF 896	*Polski*	Kovács Rózsa	Mérnök M	Pécs	-
FGH 333	*Lada*	XY vállalat	-	Budapest	Z

14.10 ábra: A KOCSI egyedtípus megbontása

Az olvasó újra beláthatja, hogy a kivetítésekkel nyert egyedek esetében már nem lépnek fel a korábbi anomáliák. Mindez annak köszönhető, hogy az eredeti egyedeket megtisztítottuk a tranzitíven függő tulajdonságoktól és jobb normálformájú egyedeket alkottunk.

D 14/5 **Az egyed akkor és csak akkor van legalább 3NF alakban, ha minden nem-kulcs tulajdonsága függ a teljes azonosítótól és csakis attól függ.**

A harmadik normálformájú egyedben nincs ismétlődő tulajdonság (0NF), részleges függés (1NF) és tranzitív függés (2NF). Minden leíró tulajdonság függ a kulcstól (1NF), csak a teljes kulcstól (2NF) és semmilyen más tulajdonságtól, csak a kulcstól (3NF).

Most ismét illene elmondanunk azt, hogy a normalizálás nem végezhető mechanikusan, hanem szemantikus normalizálásra (ld. 14.2.3 alpont) van szükség. Felhívhatnánk a figyelmet a harmadik normálforma és a struktúra összefüggéseire (ld. 14.2.4 alpont). Az utóbbi témában a 2NF alakra való megbontással szemben csak egy eltérést tapasztalnánk. Mivel a tranzitív függést okozó tulajdonság (pl. Típus) az eredeti egyedben nem kulcsrész, az új (KOCSITÍPUS) és az eredeti (KOCSI) egyed közötti 1:N fokú kapcsolat alulról nem szükségszerűen kötelező. Például elvileg nyilvántarthatunk olyan kocsit is, amelynek ismerjük a rendszámát, a tulajdonosát stb., de egyelőre ismeretlen számunkra a kocsi típusa.

Persze jól tudjuk, hogy az olvasó egyáltalán nem elégedett a 14.10 ábra KOCSI egyedtípusának a tartalmával. Azonban az eddigi eljárással sokkal jobb szerkezetet nem tudtunk összecsiholni. Mielőtt a jó megoldás titkát elárulnánk, rá kell mutatnunk a normalizálás általános természetére.

14.4 A normalizálás természete

Eddig az ismertebb és egyszerűbb normálalakokkal foglalkoztunk. Csak a következő fejezetben térhetünk át az összetettebb normálformák ismertetésére. Mielőtt ezt megtennénk, fel kell hívnunk a figyelmet a normalizálás néhány általános - a konkrét normálformától független - elméleti törvényszerűségére és az azokon alapuló gyakorlati megfontolásokra. Ugyanis az alább leírtak még fokozottabban érvényesek a további normálformákra.

A mai adatbáziskezelő rendszerek egyáltalán nem, a fejlesztőrendszerek pedig csak korlátosan képesek ***automatikusan*** felfedezni azt, hogy egy egyedtípus nincs tökéletes normálformában. Ez nem is csoda. Honnan tudhatná egy automata, hogy a 12.3 ábra Főhatóság és Felügyelet tulajdonsága valójában azonos; a látszólag egyező Foglalkozás némileg eltérő; a Típus itt meg ott mást jelent? A homonimákkal és szinonimákkal egyetlen gépi eszköz sem tud megbirkózni, hacsak nem közöljük vele az ilyen jelleget. Mivel ezt nekünk kell megtennünk, nyugodtan leszögezhetjük, hogy a csodás támogatások ellenére is a normalizálás mindig ***emberi*** feladat marad.

Ennek legfőbb okát már említettük. A normalizálás sohasem lehet ***mechanikus***. Már az első lépés, az 1NF alakú egyedek kialakítása során is emberi beavatkozásra van szükség (ld. 13.4.3 alpont). Mint fentebb láthattuk, a normálforma dekompozíció sem csak annyit jelent, hogy kivetítjük az egyedből a helytelenül függő tulajdonságokat. Nekünk kell meghatároznunk, hogy a kiemelt tételek végül is a modell melyik részébe kerüljenek illetve egyáltalán normalizálási feladatról van-e szó (ld. 14.2.3 alpont). A megfelelő normálforma kialakítása közben a valós jelenségek összefüggéseit emberileg kell értelmezni. Vagyis ***szemantikus*** normalizálást kell végezni. Erre ma semmilyen automata sem képes.

A fentiek miatt sohasem várhatjuk azt, hogy valóban „egycsapásra” kialakul a modell optimális szerkezete. A mechanikus normalizálás sokszor ***iterációkat*** feltételez. Az ismételt vizsgá-

latok a humán normalizálás esetében sem kerülhetők ki, mert az ember képtelen egyszerre áttekinteni a valamennyire is összetett modell „apró” átalakításának valamennyi következményét. Ha az egyik egyedtípust normalizálja, a másik válik kedvezőtlen alakúvá. Mégpedig nem mindig olyan szerencsés módon, mint mintapéldánk esetében, amelynek teljes normalizálását majd a következő pontban vezetjük le.

Mielőtt erre sort kerítenénk, el kell mondanunk, hogy a normalizálás ***nem végcél***, hanem ***eszköz***. A nem-normalizált egyedek ismétlődést tartalmaznak. A tökéletes normálformánál alacsonyabb alakú egyedek pedig tárolási, karbantartási, hozzáadási és törlési anomáliákat okozó redundanciákkal járnak. Ezért akkor, ha ezeket a bajokat el akarjuk kerülni, mert az a végcél, a normalizálás eszközéhez folyamodunk. Ámde mi a helyzet abban az esetben, ha nincs szándékunkban a problémák megszüntetése?

A „nincs szándékunkban” kitétel világosan mutatja, hogy látjuk az elvi gondokat, de azokat valamilyen gyakorlati megfontolás miatt eltűrjük. Például tudjuk azt, hogy nem illik a KOCSI egyedtípusba felvenni a Férőhely tulajdonságot (csak 2NF), de mivel annak értéke úgysem változik, szándékosan nem alkotunk külön KOCSITÍPUS egyedet. Nem vesszük ki a KOCSI egyedből a tranzitíven függő tulajdonságot és nem hozzuk az egyedet jobb (3NF) alakra. Mármost ilyen ***„tudatosan rossz***” modell kialakítására csak az ember képes, ami igazolja az alpont első két bekezdésének a mondanivalóját.

Ezt a „tudatosan rossz” kifejezést meg kell magyaráznunk. A 12.3 ábra KOCSI egyedtípusának a tervezője két utat követhetett. Az első - a valószínűbb - az, hogy sohasem hallott a normálformákról. Ezért nem is gondolt arra, hogy az egyed Férőhely tulajdonsága fizikailag redundáns és a KOCSI nincs 3NF alakban. Nem számolt azzal, hogy a Típus később nemcsak a Férőhely, hanem a Márkanév, Üzemanyag, Kocsisúly stb. tulajdonságokat is meghatározza. Tehát szükség lenne külön KOCSITÍPUS egyedtípusra. A második út az, hogy a tervező már mindezt átlátja, de „csakazértis” a KOCSI egyed tulajdonságává teszi a Férőhelyet. Tehát tudatosan rossz normálformát választ. Azért, mert ...

Amikor a tervező már tudja, hogy valamelyik egyedtípus nincs a legjobb normálalakban és mégis amellett a forma mellett dönt, ***denormalizálásról*** beszélünk. Ha a tervező ismeri és megalkothatná a jó egyedszerkezetet, azt már az elméjében normalizálta és másodlagos megfontolások miatt mégis az alacsonyabb alakot választja, csak akkor lehet szó denormalizálásról. Visszaalakítani csak már egy kialakított képet lehet.

Bevalljuk, hogy csak az elméleti teljesség kedvéért említettük a denormalizálást. Nem igazán kedveljük ezt a fogalmat, mert elvileg nem tiszta. A normálalakok elvi alapjait ismerő - tudatos - tervező a ***fogalmi*** tervezés szintjén normalizál. Azért, mert tudja, hogy a kocsi és a kocsitípus szemantikailag két különböző lényeg. A kocsitípus leírható ismeretekkel anélkül is, hogy létezne a típusnak megfelelő konkrét kocsi. Például egy kocsigyártó már megtervezhette a típust, mintapéldányt is alkothatott, de még nem fut az utcákon a típusnak megfelelő kocsi. Ezzel szemben a denormalizálás mindig csak hatékonysági - tárolási és kezelési - megfontolásokon alapul. Vagyis elvileg a ***logikai*** - és nem a fogalmi - szerkezeti mérlegelések szintjére tartozik.

14.5 Normalizálási sorrend

A kezdő tervezőknek azt javasoljuk, hogy tartsák be az 1NF, 2NF, 3NF normalizálási sorrendet. Tehát hozzák minden egyedtípusukat először 1NF formára, majd 2NF alakra és végül 3NF formátumba. Ha már képesek áttekinteni egy fogósabb modellrészlet minden tényezőjét, csak akkor próbálkozzanak azzal, hogy egyszerre több egyedet normalizáljanak. Egy tapasztalt tervező már képes az ilyen „varázslatokra”. A 12.3 ábra ugyancsak zavaros modelljéből a 12.5 ábra tiszta

adatbázistervét kialakítani egy rutinos tervezőnek pár percnyi munka. Azonban a rutin csak a gyakorlatból szerezhető meg. Ezért most szépen sorra vesszük a 12.3 ábra feladatának a megoldási lépéseit. A kiinduló helyzetet és a részeredményeket nem táblázatos, hanem lista formátumban mutatjuk. Lásd 14.3 példa.

14.3 példa

TULAJDONOS	(***Törzsszám***, Típus, Tulajdonosnév, Foglalkozás, Telephely, Felügyelet)
KOCSI	(***Rendszám***, Típus, Tulajdonosnév, Foglalkozás, Telephely, Főhatóság, Férőhely)
KÁR	(Kárszám, Dátum, Típus, Tulajkód, Károsszeg)

Természetesen az első teendő az, hogy a kiinduló modellt egyértelművé tegyük. Megszüntessük a homonimákat és a szinonimákat illetve a valóságot hűen tükröző neveket adjunk a tételeknek. Enélkül a legzseniálisabb tervező sem juthat messzire. Példánkban a Típus homonimát és a Felügyelet-Főhatóság illetve Törzsszám-Tulajkód szinonimát kell felszámolni. A Telephely rejtett homonima, nem egyértelmű név a KOCSI egyedben. Így helyette új nevet választunk. A megtisztított normalizálási alapot a 14.4 példa mutatja.

14.4 példa

TULAJDONOS	(***Törzsszám***, Tulajdonostípus, Tulajdonosnév, Foglalkozás, Telephely, Felügyelet)
KOCSI	(***Rendszám***, Kocsitípus, Tulajdonosnév, Foglalkozás, Tárolóhely, Felügyelet, Férőhely)
KÁR	(Kárszám, Dátum, Kocsitípus, Törzsszám, Károsszeg)

Ez a tervváltozat már egyértelmű, de - mint tudjuk - rossz. Az első két egyed legalább első normálformájú, így azokkal most nem törődünk. A KÁR egyednek viszont nincs igazi azonosítója. Jelen esetben innen (is) észrevehetjük, hogy az egyed (sorokba rejtett) ismétlődést tartalmaz, vagyis 0NF alakú. Mivel az egyedből hiányzik az igazi kulcsrész, azt kiegészítjük a Rendszám tulajdonsággal. Majd leválasztjuk az ismétlődő részt egy új KÁR/KOCSI egyedtípusba (vö. 14.1 ábra). Az eredeti KÁR egyedben csak a kulcs és a Dátum marad. Lásd a 14.5 példát. Mellesleg ennek a Dátumnak is adhatnánk szebb, beszélőbb nevet (Káresemény-dátum).

14.5 példa

TULAJDONOS	(***Törzsszám***, Tulajdonostípus, Tulajdonosnév, Foglalkozás, Telephely, Felügyelet)
KOCSI	(***Rendszám***, Kocsitípus, Tulajdonosnév, Foglalkozás, Tárolóhely, Felügyelet, Férőhely)
KÁR	(***Kárszám***, Dátum)
KÁR/KOCSI	(***Kárszám+Rendszám***, Kocsitípus, Törzsszám, Károsszeg)

Mivel az első három egyed legalább 2NF alakú, csak a negyedik egyeddel kell törődni. A részleges függések miatt kiemeljük a Kocsitípus és a Törzsszám tulajdonságokat. Az eredeti egyedtípusban csak a kulcs és a Károsszeg marad. A kiemelt Kocsitípus és Törzsszám tulajdonságokra nem hozunk létre új egyedtípust, mivel a meghatározó (Rendszám) által azonosított egyed már létezik (KOCSI). Mivel a két Kocsitípus fogalom azonos, ezt a tulajdonságot nem adjuk a KOCSI sorához. Abba csak a Törzsszám-ot visszük át. Lásd a 14.6 példát.

Korábban már rámutattunk arra, hogy a kétoszlopos táblák majdnem mindig 5NF alakúak. Ezért a KÁR egyedtípussal többet nem is kell törődnünk. 5NF formában vannak azok az egyedtípusok is, amelyek egyetlen funkcionális függést tartalmaznak. Ilyen a KÁR/KOCSI egyedtípus. Így azt sem kell majd tovább vizsgálnunk.

14.6 példa

TULAJDONOS	(***Törzsszám***, Tulajdonostípus, Tulajdonosnév, Foglalkozás, Telephely, Felügyelet)
KOCSI	(***Rendszám***, Kocsitípus, Tulajdonosnév, Foglalkozás, Tárolóhely, Felügyelet, Férőhely, Törzsszám)
KÁR	(***Kárszám***, Dátum)
KÁR/KOCSI	(***Kárszám+Rendszám***, Károsszeg)

A TULAJDONOS legalább 3NF alakú. Ezért már csak a KOCSI-t kell tovább normalizálni. Az elsődleges kulcson kívül az egyedben két meghatározó van. A Kocsitípus → Férőhely függés miatt az utóbbit ki kell emelnünk a KOCSITÍPUS egyedbe. A Törzsszám meghatározza a Tulajdonosnév, Felügyelet és Foglalkozás adatokat. Mivel az első kettő a TULAJDONOS egyedet azonos értelemmel jellemzi, nincs szükség azok átvitelére. Ezeket a tulajdonságokat egyszerűen el kell hagynunk a KOCSI egyedtípusból. A Foglalkozással is ezt kell tennünk. Viszont ez a tulajdonság nem egyértelmű. Ezért nem mondhatjuk azt, hogy az már a TULAJDONOS jellemzője. Az eltérő kódértékek miatt meg kell keresnünk a felhasználót. Tisztáznunk kell, hogy valóban szükség van-e a kétféle kódolásra. Ha nem, akkor meg kell kérdeznünk, hogy melyiket alkalmazzuk. Ha nincs szükség a kétféle foglalkozási adatra (ezt feltételezzük), akkor valóban nincs szükség a tulajdonságnak a beillesztésére a TULAJDONOS egyedbe. Ekkor csak azt kell feljegyeznünk, hogy melyik kódolás szerinti a Foglalkozás. Ha szükség van a két változatra, akkor viszont új nevet kell adni a két tulajdonságnak, mert azonos nevű tételek nem szerepelhetnek egy egyedben. Lásd a 14.7 példa végleges megoldását, amely megfelel a 12.5 ábra modelljének.

14.7 példa

TULAJDONOS	(***Törzsszám***, Tulajdonostípus, Tulajdonosnév, Foglalkozás, Telephely, Felügyelet)
KOCSITÍPUS	(***Kocsitípus***, Férőhely)
KOCSI	(***Rendszám***, *Kocsitípus*, *Törzsszám*, Tárolóhely)
KÁR	(***Kárszám***, Dátum)
KÁR/KOCSI	(***Kárszám+Rendszám***, Károsszeg)

14.6 A dekompozíció sajátosságai

Eddig a normalizálást dekompozícióval végeztük. A kedvezőtlen normálformájú egyedeket lebontottuk több egyedre. Ezt a műveletet ***kivetítéssel*** végeztük. Az eredeti egyedből kiemeltük az azonosítótól nem függő (vö. 1NF) vagy kedvezőtlen függésű (vö. 2-3NF) tulajdonságokat. Az eljárás eredményeként mindig tökéletesebb szerkezethez jutottunk. Ez két dolgot jelent.

Egyrészt azt, hogy a kedvezőbb normálalak esetén nem lépnek fel az alacsonyabb normálformát jellemző visszásságok. Másrészt azt, hogy az új modellszerkezet alapján mindig visszaállítható a régi - a normalizált - struktúra. A dekompozíció tehát megfordítható, ***reverzibilis művelet***. A 14.8 ábra egyedéből a 14.9 ábra modellrészletét alakítottuk ki. Semmi akadálya sincs annak, hogy a műveletet visszafordítsuk. A 14.9 ábra két egyedéből visszakapjuk a 14.8 ábra egyetlen egyedének az ismereteit. A normalizálással nem vesztettünk ismeretet. Az ilyen megbontást ***veszteségmentes dekompozíciónak*** (angolul: non-loss decomposition) nevezzük. Az

1-3NF szerinti dekompozíció mindig veszteségmentes. Majd a következő fejezetben látunk példát a veszteséges megbontásra is.

Itt röviden ki kell térnünk arra, hogy a kivetítés fordított irányú művelete az ***összekapcsolás*** (angolul: join). Ennek feltétele az, hogy két egyedtípus rendelkezzen azonos tartalmú tulajdonságtípussal (pl. Vevőkód). A művelet szimmetrikus. Mindegy, hogy a 14.9 ábra VEVŐ egyedéhez kapcsoljuk-e a RENDELÉS egyedet, vagy megfordítva. Most feltételezzük az utóbbit. Az eljárás során az egyik egyedtípust (RENDELÉS) kiegészítjük a benne nem szereplő, a csak a másikat (VEVŐ) jellemző tulajdonságtípusokkal (Vevőnév). Ugyanakkor az első egyedtípus minden sorát kibővítjük a második egyedtípus minden olyan sorával, ahol a kétféle sorban a kapcsoló tulajdonság (Vevőkód) értéke azonos. Valahányszor a RENDELÉS táblában „111" a Vevőkód értéke, annyiszor bővül a sor az „X" vevőnév-értékkel. Az olvasó meggyőződhet róla, hogy ilyen módon valóban visszakapja a 14.8 ábra RENDELÉS egyedének a tábláját.

Felhívjuk a figyelmet arra, hogy a visszafordíthatóság egyáltalán nem jelent azonos terveket. A 14.9 ábra megoldását - egyéb megfontolások mellett - éppen azért választottuk, hogy olyan vevők ismereteit is nyilvántarthassuk, akiknek nincs aktuális megrendelésük. Ezért lesznek olyan VEVŐ egyed-előfordulásaink, amelyek azonosítójának (Vevőkód) az értéke még/már nem fordul elő a RENDELÉS egyed-előfordulásokban és amelyeken éppen ezért az összekapcsolás nem végezhető el. Az eredeti adatbázis visszaállítható, de ugyanakkor az új adatbázis a kiindulónál több ismeret tárolására képes.

A veszteségmentesség csak szükséges, de nem elégséges feltétele a dekompozíciónak. Az általános szakirodalom megkülönböztet ***jó*** és ***rossz*** lebontásokat (lásd például [22], 253. oldal). Vizsgáljuk csak meg közelebbről ezt a minősítést!

A 14.8 ábra RENDELÉS (***Rendelésszám***, Vevőkód, Vevőnév) egyedét elvileg kétféle módon alakíthatjuk át. A 14.8 példa által mutatott megoldások lehetségesek.

14.8 példa

VEVŐ (***Vevőkód***, Vevőnév)
RENDELÉS-1 (***Rendelésszám***, *Vevőkód*)

RENDELÉS-2 (***Rendelésszám***, Vevőkód)
RENDELÉS-3 (***Rendelésszám***, Vevőnév)

Mindkét lebontás veszteségmentes, vagyis azokból előállítható az eredeti RENDELÉS egyed. Ennek ellenére az első dekompozíciót jónak, a másodikat rossznak tartjuk. Azért, mert az első esetben a tranzitív függést maga a szerkezet oldja fel a két egyedtípus kapcsoló tulajdonsága (Vevőkód) által. A kapcsoló tulajdonság pedig automatikus korlát. A RENDELÉS-1 egyedbe nem vihető be olyan Vevőkód érték, amely ne szerepelne a VEVŐ egyedben. A Vevőnév tételt pedig nem lehet a Vevőkód-tól függetlenül aktualizálni.

Ezzel szemben a második megoldás karbantartási anomáliát mutat. Ha a Vevőnév értékét átírjuk egy RENDELÉS-3 előfordulásban, akkor ki kell keresni az azonos Rendelésszám értékű RENDELÉS-2 előfordulást. Meg kell állapítani a Vevőkód értékét. Majd ki kell gyűjteni az ugyanilyen Vevőkód értékű RENDELÉS-2 tételek Rendelésszámait. Ahol a RENDELÉS-3 egyedben ez a Rendelésszám fordul elő, ott mindenütt át kell írni a Vevőnév tartalmát az új értékre.

Ezt az gondolat-akrobatikát csak az elméleti teljesség kedvéért idéztük fel. Azért, ha netán valaki találkozik a jó és a rossz dekompozíció fogalmával, tudja, hogy miről is van szó. Valójában a neves szakírók gondolatmenete háromszorosan is hibás.

Először: A tiszta adatmodellezés eleve kizárja azt, hogy két egyednek azonos legyen az elsődleges kulcsa. Ezért eleve nem hozható létre a RENDELÉS-2 és RENDELÉS-3 egyedpáros úgy,

hogy megegyezzen az azonosítója. Codd az optimális normalizálás elveinek a kifejtése során [21] leszögezte, hogy ha két egyed azonosítója közös, akkor a két egyed valójában egy és azok tulajdonságtípusait egyetlen tulajdonságsorba kell összevonni. Ezzel pedig visszakapnánk az eredeti RENDELÉS (***Rendelésszám***, Vevőkód, Vevőnév) egyedet. Így a normalizálás végtelen ciklusba esne.

Másodszor: Sokan nem figyelnek a külső összefüggésekre. Holott evidens, hogy már a normalizálás előtt tudjuk, hogy mindenképpen lesz egy VEVŐ egyedtípusunk is, mert hiszen van egy ilyen ismeretekkel leírandó jelenségcsoportunk. Ennek az egyednek természetes jellemzője a Vevőnév. Így a RENDELÉS-3 egyed mindenképpen rossz megoldás.

Harmadszor: Eddigi normalizálási lépéseink során mindig ragaszkodtunk ahhoz, hogy az eredeti (RENDELÉS) egyedből le kell választani a helytelenül függő tulajdonságot (Vevőnév) annak meghatározójával (Vevőkód) együtt egy új egyedtípusba (VEVŐ). Ha ezt a rutint betartjuk, akkor egyszerűen nem alkalmazhatunk rossz dekompozíciót.

Akkor pedig miért vetettük fel egyáltalán a jó és a rossz dekompozíció kérdését? Azért, mert csak az 1-3NF alakok esetében adódik mindig egyértelmű jó megbontás. A későbbi normálalakok némelyikénél több megbontási változat is akad, amelyek nem teljesen egyenértékűek.

Az 1-3NF formák kialakítása során alkalmazott dekompozícióval, ha azt az általunk ismertetett szabályok szerint hajtják végre, ***független*** (angolul: independent) egyedtípusokat kapunk. A megbontásnak csak annyi a következménye, hogy az ***egyeden belüli*** (angolul: intra-entity) korlát ***egyedek közötti*** (angolul: inter-entity) feltétellé válik. Ha a 14.8 ábra RENDELÉS egyedtípusában mindig ki kell tölteni a Vevőkód értékét (belső korlát), akkor a 14.9 ábra minden RENDELÉS egyedéhez kell, hogy tartozzon egy VEVŐ egyed (külső korlát).

A 14.8 példa második megoldása esetében a belső korlát nem vált külsővé. A két egyed karbantartását nem lehet korlátokkal ellenőriztetni. Ilyenkor azt mondjuk [23], hogy a megbontás ***nem-független*** (angolul: non-independent). Nem olyan egyedeket eredményez, amelyek a korlátok betartása esetében egymástól függetlenül karbantarthatók. Mint tudjuk, a 14.8 példa második megoldása rossz. Viszont a továbbiak során találkozni fogunk olyan helyzetekkel, amelyekben egyszerűen nincs mód a független dekompozícióra.

14.7 Az alternáló kulcs

Eddig csak olyan példákat mutattunk, amelyekben az egyedtípusoknak csak egy azonosításra alkalmas tulajdonságtípusa volt. Felismertük, hogy a RENDELÉS egyed kulcsa a Rendelésszám, a KOCSI-é a Rendszám stb. Legfeljebb az volt a probléma, hogy az egyednek egyáltalán nem volt kulcsa (0NF) illetve abban felléptek egyéb meghatározó tulajdonságok is (1-2NF). Ha az egyedtípus azonosítóját megtaláljuk, akkor az alapvető normalizálás szinte gyerekjáték.

Azonban néha találkozunk olyan egyedtípusokkal is, amelyek egynél több olyan tulajdonságot tartalmaznak, amely azonosító lehetne. Ilyenkor ezeket a tételeket ***kulcsjelölteknek*** (angolul: candidate key) nevezzük. A kulcsjelöltek az összes többi nem-kulcs tulajdonságot funkcionálisan meghatározzák. Ugyanakkor közöttük nyilván kölcsönös függés áll fenn. Vegyük csak alapul a 14.9 példát.

14.9 példa

VEVŐ (Vevőkód, Vevőnév, Vevőcím ...)

Tételezzük fel, hogy a vevőnevek egyediek, a vevőcímek viszont nem azok. Ezért az egyedben két kulcsjelölt található: a Vevőkód és a Vevőnév. (Emiatt nem is jelöltük meg az egyed azonosítóját a szokásos módon.) A két lehetséges azonosító mindegyike funkcionálisan meghatározza a Vevőcím tételt. Emellett pedig fennáll a Vevőkód ↔ Vevőcím mindkét oldalról erős kölcsönös függés.

Ez a helyzet önmagában véve semmilyen komoly gondot nem okoz. Mivel a modellezés szabályai szerint minden egyedtípusnak csak egy azonosítója lehet, a kulcsjelöltek közül ki kell választani az ***elsődleges kulcsot*** (angolul: primary key). Legyen az példánk esetében a Vevőkód. A választás után a további kulcsjelölteket ***alternáló vagy helyettes kulcsnak*** (angolul: alternate key) hívjuk. Tehát példánk esetében a Vevőnév alternáló kulcs.

Az alternáló kulcs lényegét sokan félreértik. Tévedésüket egy további példával világítjuk meg.

14.10 példa

SZERZŐDÉS (Belső-kód, Szerződésszám, Szerződés-dátum ...)

A tervező a Belső-kód és a Szerződésszám tulajdonságot kulcsjelölteknek tekinti és kijelenti, hogy a Belső-kód elsődleges kulcsnak a Szerződésszám alternáló kulcsa. A két tételt a következő módon használja: amíg a konkrét szerződés nem kap Belső-kód azonosító értéket, addig azt a Szerződésszám tartalmával azonosítja.

A kétszeres elvi tévedés teljesen evidens. Egyrészt az azonosítók (definíció szerint) sohasem lehetnek üres vagy ismeretlen értékűek. Ezért a Belső-kód egyáltalán nem lehet kulcs. Másrészt a kulcsjelöltek között (definíció szerint) kétoldalúan erős kölcsönös függés áll fenn. Ha a Belső-kód (vagy a Szerződésszám) értéke üres is lehet, akkor nem áll fenn ez a kölcsönös függés, tehát nem beszélhetünk kulcsjelöltekről és a Szerződésszám nem nevezhető alternáló kulcsnak.

Az, hogy a fejlesztők nem tudnak időben Belső-kód értéket adni a szerződésnek nem fogalmi szintű modellezési gond. A gyakorlati problémát nem lehet az elmélet összezavarásával fedezni. Tudomásul kell venni, hogy az alternáló kulcs nem más, mint ***integritási korlát***, mint ahogyan minden függési viszony az. Ez a specifikus korlát azt jelenti ki, hogy mindkét tulajdonság egyedi értékű és az egyik tulajdonság minden egyes értékéhez minden időben a másiknak pontosan csak egy értéke tartozik illetve ez megfordítva is igaz.

A normálforma dekompozícióban az alternáló kulcsot sajátos módon kezeljük, amennyiben egyáltalán nem vizsgáljuk függési viszonyait. Minek is tennénk, hiszen minden ilyen összefüggése a többi tulajdonsággal az elsődleges kulcséval azonos jellegű.

Arra viszont fel kell hívnunk a figyelmet, hogy a helyes alternáló kulcsok megtalálása nem könnyű feladat. A „minden időben csak egy” nem azonos a „minden időpontban egy” kitétellel. Nézzük csak meg a 14.11 példát.

14.11 példa

KOCSI (***Rendszám***, ..., Casco-kötvényszám, Kötés-dátum, Önrész, ...)

Bár egy kocsinak minden időpontban csak egy Casco-biztosítása lehet, a Casco-kötvényszám nem a Rendszám alternáló kulcsa. Nem csak azért nem, mert nem minden kocsinak van Cascoja, tehát a kölcsönös függés nem létezik. Hanem azért sem, mert az idők során a kocsi Cascoja megszűnhet vagy keletkezhet. Ezért a Casco-kötvényszám nem stabil jellemzője a kocsinak és így sohasem lehetne annak valódi azonosítója.

Ha egy tulajdonság valódi alternáló kulcs, akkor nincs vele teendő. Ha viszont csak álhelyettesítő, amit a Rendszám → Casco-kötvényszám függés gyenge jellege mutat, akkor fel-

merülhet a normalizálás igénye. A Kötés-dátum, Önrész, ... tulajdonságoknak valóban a KOCSI egyedben van-e a helye, vagy érdemesebb egy külön CASCO egyedtípust létrehozni. Erre a kérdésre majd könyvünk későbbi részében válaszolunk (ld. 18.5 pont).

Ellenőrző kérdések - 14

14/01 Hányadik normálformában van a következő egyed? Számmal válaszoljon.

RENDELÉS
Rendelésszám, R-dátum, ..., Cikkszám, Rendelt-mennyiség

R1	D1	C1	M1
		C2	M2
R2	D2	C1	M3

14/02 Legalább hányadik NF alakban van az ismétlődést nem tartalmazó egyed?

14/03 Hányadik normálformájú a következő egyed:

SZÁMLATÉTEL (***Számlaszám+Cikkszám***, ..., Tételérték, Egységár)

14/04 Mi az előző példa helyes megoldása?

14/05 Adott a két alábbi egyedtípus. Mit tenne Ön a tranzitivitás megszüntetése során? Adja meg a helyes válasz sorszámát.

VEVŐ (***Vevőkód***, ..., Vevő-levelezési-cím)
RENDELÉS (***Rendelésszám***, ..., *Vevőkód*, Vevőcím)

- Mivel a Vevőcím duplikátum, egyszerűen elhagynám a RENDELÉS-ből.
- Felkeresném a felhasználót, hogy a két cím azonos tartalmú-e.
- A cím és a levelezési cím két dolog. Tehát a Vevőcím-et a VEVŐ-be tenném.

14/06 Hányadik normálformában van a következő egyed? KB = Kötelező biztosítás.

KOCSI (***Rendszám***, ..., KB-kötvényszám, KB-kötésdátum)

15. MAGASABB NORMÁLFORMÁK

15.1 Hány, melyik és milyen a kulcs?

Az előző fejezetben bemutattuk az 1-3NF alakokat. Az egyedtípusoknak a funkcionális függéseken (FD) alapuló normalizálása viszonylag egyszerű feladat. Az ismétlődéseket - a feltételezett elsődleges kulcstól független tulajdonságokat - le kell választani (1NF). Ki kell szűrni a részleges függéseket (2NF) és a tranzitív meghatározásokat (3NF).

Az első nehezebb gondok akkor támadnak, ha az egyedtípusban ***több*** kulcsjelölt van. Még ezt a helyzetet is fel tudjuk oldani, ha a kulcsjelöltek elemiek és tisztán alternáló jellegűek (ld. a 14.7 pontot). Már komolyabb problémák léphetnek fel akkor, ha a kulcsjelöltek nem egyszerűek, hanem ***összetettek***. Ilyenkor két eset lehetséges. Ha az összetett azonosítók kölcsönösen függenek egymástól és részeik között nincs funkcionális függés, tehát például az A+B és a C+D két kulcsjelölt négy tulajdonsága közül egyik sem határozza meg a másikat, akkor tisztán alternáló kulcsokról van szó.

Ezzel szemben előfordul az a helyzet is, hogy az egyik összetétel valamelyik tagja meghatározza a másik kulcsjelölt egyik tagját. Például fennáll a B $\rightarrow$ D függés. Ekkor karbantartási anomáliák lépnek fel. Ezért az egyedtípust meg kell bontani. Viszont azt nem tehetjük az eddigi elméleti alapokon, mivel az 1-3NF definíciói szerint csak a „nem-kulcs" tulajdonságok függéseit vizsgáltuk. A második pontban bemutatott Boyce-Codd normál-forma (BCNF) segít bizonyos helyzetekben e probléma megoldásában.

Máskor az egyedtípus ún. csupakulcs reláció. Emiatt a projektív függéseken - a kulcs meghatározza a saját részeit - kívül nem is léphet fel bennük funkcionális függés. Az ilyen egyedekben is előfordulhatnak karbantartási anomáliák. Ezek kiszűrésére új függésfajtákat fogunk bevezetni és ismertetjük a 4-5NF alakokat a harmadik és negyedik pontban.

Ennek a fejezetnek az a célja, hogy feltárja a többféle kulcsjelölt esetén alkalmazandó megoldásokat és ismertesse az olyan egyedek megbontásait, amelyek csak kulcsot tartalmaznak, de mégis karbantartási anomáliákat mutatnak.

Bár egy szerzőnek ilyesmit nem illene tennie, mégis kénytelenek vagyunk bevallani, hogy ezt a részt csak kötelességből írtuk meg. Azért, hogy az olvasót teljeskörűen tájékoztassuk. Ebben a fejezetben „nem-szeretem" normálformákról lesz szó. Hátulról kezdve, az 5NF problémája olyan ritka, mint a fehér holló. A 4NF alak magánvéleményünk szerint felesleges kreálmány. Rá fogunk mutatni arra, hogy az az egyed, amely nincs 4NF alakban valószínűleg egyáltalán nem is normalizált, vagyis titokban 0NF alakú. A BCNF forma ritka esetekben jól alkalmazható, de a legtöbb helyzetben nem normalizálással, hanem szemantikai átértelmezéssel kell megoldani a többféle kulcsjelölt által okozott problémát. Végeredményben mi az egyszerű és érthető megoldásokat keressük. Nem rajongunk azért, ha elméleti szakemberek briliánsan megoldanak egy gyakorlatilag nem is létező gondot.

15.2 A Boyce-Codd normálforma (BCNF)

Ha az egyedben előre felismerjük az elsődleges kulcsot és nincs alternáló kulcs vagy a kulcsjelöltek mindegyike egytagú (ld. 14.9 példa), akkor a normalizálás viszonylag egyszerű eljárás. Az 1-3NF kialakítása nem jelent nagy kihívást. Viszont egészen más a helyzet akkor, ha még nem döntöttünk az azonosító felől azért, mert több kulcsjelöltünk is van úgy, hogy azok többtagúak. A problémát a 15.1 ábra példájával szemléltetjük. Az esetet C. J. Date-től vettük át [22, 251. oldal].

TANÍTÁS

Diák	Tárgy	Tanár
Kovács	Matek	Fehér
Kovács	Fizika	Fekete
Szabó	Matek	Fehér
Szabó	Fizika	Barna

15.1 ábra: Kétes szerkezetű TANÍTÁS egyedtípus

A feltételezések a következők:

- Minden tárgyra nézve igaz, hogy az azt tanuló diákot csak egy tanár tanítja a kérdéses tárgyra. Ezért fennáll a Diák+Tárgy → Tanár függés.
- Minden tanár csak egy tárgyat tanít. Ezért létezik a Tanár → Tárgy függés.
- Egy tárgyat több tanár is taníthat. Ezért nem létezik a Tárgy → Tanár függés.

15.2.1 A több összetett kulcsjelöltből fakadó gondok

A TANÍTÁS egyed ***karbantartási anomáliákat*** mutat. Ha kitöröljük belőle azt a sort, amely szerint Szabó fizikát tanul, akkor elveszik az az ismeret is, hogy Barna tanár úr fizikát tanít, ha Szabó volt az utolsó tanítványa. (A gyakorlatban nem valószínű, hogy egy tanár csak egy diákot tanít egy tárgyra. Ezért ezzel a példával csak a probléma jellegét kívántuk szemléltetni.) Nem tudjuk az egyedbe illeszteni azt az ismeretet, hogy Zöld tanár úr földrajzot tud okítani mindaddig, ameddig nem akad ilyen tárgyat tanuló diák. Ha Fehér tanárnő férjhez megy és Pirosra változtatja a nevét, akkor a módosítást több előforduláson kell végrehajtani. A karbantartási visszásságok miatt az egyedtípust át kell alakítani. A kérdés az, hogy mi a normalizálás elvi alapja és azt miként hajtjuk végre?

Vajon hányadik normálformában van a TANÍTÁS egyed? Mivel kulcsát még nem ismerjük, e felvetésre nem akarunk egyelőre válaszolni. Először meg kell keresni az azonosítót. Az egyedben nincs olyan elemi tulajdonság, amely elsődleges kulcsként szolgálhatna, hiszen mindegyik tulajdonságtípus többször is felveheti ugyanazt az értéket. Ezért a tételeket párosával kell vizsgálnunk.

A Diák+Tárgy meghatározza a Tanár-t, tehát kulcsjelölt. A Diák+Tanár párostól függ a Tárgy, vagyis az is lehetséges kulcs. A Tanár+Tárgy viszont nem lehet azonosító, mivel a Tanár meghatározza a Tárgy-at. Végeredményben az egyed mindegyik tulajdonsága vagy kulcsjelölt (meghatározó), vagy annak része. Most szándékosan mondjuk úgy, hogy a TANÍTÁS egyed minden tulajdonságtípusa nem „nem-kulcs” jellegű (ld. 14.2 pont). Márpedig az 1-3NF meghatározásai (ld. D 14/1, D 14/3 és D 14/5) szerint az alapvető normálforma hibák csak akkor lépnek

fel, ha a „nem-kulcs” tulajdonságok függetlenek a kulcstól vagy attól részlegesen illetve tranzitíven függnek.

Végeredményben az eddigi meghatározások szerint a TANÍTÁS egyed 3NF formájú és az eddigi alapokon nem normalizálható. Az új megfontolásokhoz látni kell, hogy a probléma gyökere kettős. Egyrészt az egyedben van olyan tulajdonság (Tanár), amely funkcionális meghatározó (tőle függ a Tárgy), de ez ***a meghatározó nem kulcsjelölt***. Másrészt van két összetett kulcsjelöltünk (Diák+Tárgy és Diák+Tanár), amelyeknek van közös része.

Most kell eldöntenünk, hogy melyik párost választjuk kulcsnak. Először azt nézzük meg, hogy milyen eredményre jutnánk, ha a Diák+Tárgy lenne az azonosító. Ekkor a Tanár → Tárgy függés miatt ezt a párost ki kell emelnünk az egyedből. Az eredményt a 15.2 ábra mutatja.

TANÁR

Tanár	Tárgy
Fehér	Matek
Fekete	Fizika
Barna	Fizika

TANULJA

Diák	***Tárgy***
Kovács	***Matek***
Kovács	***Fizika***
Szabó	***Matek***
Szabó	***Fizika***

15.2 ábra: A TANÍTÁS egyedtípus rossz megbontása

A 14.6 pontban beszéltünk a jó és a rossz dekompozícióról (ld. 14.8 példa). A 15.2 ábra megoldása rossz. A két egyedből ugyan visszaállítható a TANÍTJA egyed, de karbantartási gondok lépnek fel. Például Szabó-hoz bevihetünk olyan tantárgyat, amelyet egyetlen tanár sem oktat. Ezt azért tehetjük meg, mert a két egyedtípus között nincs kapcsolat. Vagyis a 15.1 ábra TANÍTJA egyedén belüli korlátot (a Tárgy mellett mindig Tanár is van) nem őriztük meg a 15.2 ábra két egyedtípusa közötti korlátként (a TANULJA nem kapcsolódik a TANÁR-hoz).

Emiatt a lehetséges megbontás miatt a szakírók komplikáltnak tartják az olyan egyedtípusok normalizálását, amelyekben van nem kulcsjelöltként szolgáló meghatározó és a két (vagy több) összetett kulcsjelölt közös résszel rendelkezik.

Mi nem látjuk ilyen sötéten a helyzetet. A TANÍTÁS egyedtípus normalizálására két utat is kínálhatunk. Először az egyszerűbbet mutatjuk be.

Tegyük fel, hogy a TANÍTJA egyednek a Diák+Tanár páros a kulcsa és feledkezzünk el az 1-3NF meghatározásaiban lévő „nem-kulcs” kitételről! Hiszen akkor, ha ez az összetétel a kulcs, a Tárgy már valójában nem-kulcs, hiába része egy másik kulcsjelöltnek. Ha a szigorú definíciókat így fellazítjuk, akkor kiderül, hogy a TANÍTJA csak 1NF alakban van. Azért, mert részleges függést tartalmaz. A Diák+Tanár a kulcs, de a Tárgy-at ennek egy része - a Tanár - is meghatározza. Ezért az egyedből ki kell venni a részlegesen függő tulajdonságot (Tárgy) annak meghatározójával (Tanár) együtt úgy, hogy az eredeti egyedben is megmarad a meghatározó. Az eredményt a 15.3 ábra mutatja.

TANÁR

Tanár	Tárgy
Fehér	Matek
Fekete	Fizika
Barna	Fizika

TANÍTJA

Diák	***Tanár***
Kovács	***Fehér***
Kovács	***Fekete***
Szabó	***Fehér***
Szabó	***Barna***

15.3 ábra: A TANÍTÁS egyedtípus jó megbontása

Mindenki meggyőződhet róla, hogy ebben a megoldásban nem lépnek fel a karbantartási anomáliák. Azért nem, mert a két egyedtípus kapcsolható, tehát egyedek közötti integritási korlát garantálja, hogy egy diákhoz ne vihessünk be nemlétező tanárt.

Persze ennek az egyszerű megoldásnak az a titka, hogy megtaláljuk a kulcsjelöltek közül a célravezetőt. Azonban miként dönthetjük el, hogy melyik a jó azonosító? Mi történne akkor, ha a Diák+Tárgy párost választanánk kulcsnak? Szükségszerűen a 15.2 ábra rossz megoldását kapnánk eredményül?

15.2.2 A kulcstörő függés

A felvetett kérdések megválaszolásához egy új fogalommal kell megismerkednünk.

D 15/1 **Az E egyedtípus C tulajdonsága akkor és csak akkor okoz külső kulcstörő függést, ha függ az A+B összetett azonosítótól és meghatározza annak A vagy B részét.**

A meghatározásban a „külső” jelző arra utal, hogy a feltételezett azonosítón kívüli, ahhoz képest leíró tétel okozza a kulcstörést. A „belső” kulcstörést majd a 16.4 pontban tárgyaljuk. Példánkban a Tanár tulajdonság függ a Diák+Tárgy együttestől, de ugyanakkor meghatározza a Tárgy tulajdonságot, ami az összetett kulcs része. Az utóbbi függés a kulcsnak csak egy részét érinti és így azt „megtöri”. A karbantartási problémát ez a ciklus-jellegű jelenség okozza. A Diák+Tárgy meghatározza a Tanárt, viszont a Tárgy függ a Tanártól. Ezért az azonosító és a leíró tulajdonság nem változtatható egymástól függetlenül.

Eddigi ismereteink szerint a normalizálás során mindig a helytelenül meghatározott tulajdonságot kell kiemelnünk az egyedtípusból a meghatározójával együtt úgy, hogy az az eredeti egyedben is megmarad. Tehát példánk esetében a Tárgyat kell végleg kivennünk új egyedtípusba annak Tanár meghatározójával, miközben a Tanár az eredeti egyedet továbbra is jellemzi.

Amint látjuk, ezzel a megoldással is a 15.3 ábra modelljéhez jutunk és nem követhetjük el a 15.2 ábra rossz dekompozíciójának a hibáját. Csak azt a bocsánatos bűnt kellett vállalnunk, hogy elfeledkezve a „nem-kulcs” szigorú kitételről az egyedből éppen a kulcsnak az egyik részét (Tárgy) távolítottuk el.

Most megadjuk a Boyce-Codd normálforma [24] hivatalos meghatározását, amihez majd hozzáfűzzük saját egyéni megjegyzéseinket.

D 15/2 **Az egyedtípus akkor és csak akkor van BCNF alakban, ha minden meghatározó tulajdonsága egyben kulcsjelölt is.**

A normalizálással foglalkozó kiváló szakemberek a fentiek szerint túl nagy hangsúlyt helyeznek a kulcs/nem-kulcs kérdésre. Ha a részleges és tranzitív függést nem csak a nem-kulcs tulajdonságokra értelmeznék, akkor semmilyen különbség sem lenne a 3NF és a BCNF alakok között. Hiszen a kulcstörő függés nem más, mint egy speciális részleges vagy tranzitív függés. Sőt, valójában ez utóbbi két függésfajta is egy, amit majd a következő fejezetben kimutatunk.

A D 15/2 meghatározás szerint a 15.1 ábra TANÍTJA egyedtípusa 3NF alakban van, de nincs BCNF formában, mert nem minden meghatározó tulajdonsága kulcsjelölt. Ezzel szemben a 15.3 ábra két egyedtípusa már BCNF alakú, hiszen nincs is bennük több kulcsjelölt. A karbantartási anomáliák megszüntetése miatt a BCNF forma jobb, mint a 3NF. Azt pedig az egymásba skatulyázás elve alapján tudjuk, hogy minden BCNF alakú egyed egyben szükségszerűen 3NF formájú is.

Ezzel a BCNF tárgyalását le is zárhatnánk. Azonban a BCNF alakra történő normalizálás felvet néhány olyan problémát, amely nem lépett fel az alapvető normálformák esetében.

15.2.3 BCNF problémák

J. Rissanen [23] ***atominak*** nevezi az egyedtípust, ha az nem bontható le egymástól független egyedekre (ld. 14.6 pont). A szakírók szerint a 15.1 ábra TANÍTJA egyedtípusa atomi, mert a 15.3 ábra két egyede egymástól nem független. Azért nem az, mert az eredeti Diák+Tárgy → Tanár függést nem tükrözi, hiszen a Diák+Tárgy egyik egyednek sem a kulcsa. Ezért a TANÍTJA egyedtípus nem aktualizálható anélkül, hogy a TANÁR egyedet kezelnénk. Ha Szabó abbahagyja a fizika tanulását, akkor nem a természetes „Szabó-Fizika" ismeretpárost töröljük, hiszen az nem is szerepel az adatbázisban. Ki kell keresni azt a tanárt, aki Szabót fizikára tanítja és a „Szabó-Barna" adatpárost kell megszüntetni. Ha Szabóhoz új tanárt akarunk bevinni, akkor meg kell néznünk, hogy az nem tanít-e olyan tárgyat, amelyet Szabónak már más oktat. Pl. Fekete tanár úr nem kapcsolható Szabóhoz.

A fentiek alapján a szakírók arra a következtetésre jutottak, hogy a TANÍTJA egyedtípus tökéletesen normalizálható, azaz BCNF alakra hozható, de mivel nem atomi és a normalizálás nem-független egyedeket eredményez, lehet, hogy a normalizálás nem-kívánatos.

Mi egy picit másként látjuk ezt a kérdést. Azért, mert azt nem a matematikai, hanem a szemantikai oldaláról nézegetjük. Csak bonyolultan tudjuk megszüntetni a „Szabó-Fizika" ismeretpárost. Valóban akkora probléma ez?

Szemantikailag a Diák, Tanár és Tárgy összefüggésének két értelmezése adódhat:

- A diák az adott tárgyat tanulja. A tárgyat egy tanár tanítja. Tehát a diák a tanárral csak közvetett kapcsolatban áll a tárgyon keresztül.
- A diákot adott tanár tanítja. A tanár egy tárgyat tanít. Tehát a diák a tárggyal csak a tanáron át áll kapcsolatban.

Esetünkben világos, hogy az utóbbi értelmezés állja meg a helyét, hiszen a példa eredeti feltételei (15.2 pont) ezt sugallják. Tehát akkor, ha Szabó abbahagyja a fizika tanulását valójában nem ezt teszi. Hanem Szabó felhagy Barna tanár úr óráinak a látogatásával. Ezért a nem-független egyedtípusok kérdése ebben az esetben csak részben, a bevitelnél probléma. Persze ha Szabót több tanár is okítaná fizikára és úgy szüntetné be e tárgy tanulását, akkor más lenne a helyzet. Akkor viszont a modell is másként nézne ki. Az eredeti TANÍTJA egyed nem lenne megbontható a BCNF elvei szerint.

Most nézzük meg a BCNF formát a másik oldalról is. Lásd a 15.1 példát.

15.1 példa

VIZSGA (Diák, Tárgy, Helyezés)

A diákok adott tárgyakból vizsgáznak. A feltételezés az, hogy a diákokat a tárgyban úgy minősítik, hogy helyezési sorszámot adnak nekik és minden helyezés egyedi. Vagyis két diák nem kaphatja ugyanabból a tárgyból ugyanazt a helyezést. A kérdés az, hogy mi legyen a VIZSGA egyed kulcsa?

Két kulcsjelölt adódik: a Diák+Tárgy és a Diák+Helyezés. A kulcsjelöltek átfednek. A VIZSGA mégis BCNF alakban van, mert nincs benne olyan meghatározó, amely nem kulcsjelölt. Az egyetlen probléma az, hogy az elsődleges kulcsot ki kell választani és meg kell adni az alternáló kulcsot. A 15.2 példa mutatja a két lehetséges megoldást.

15.2 példa

VIZSGA-1 (***Diák+Tárgy***, Helyezés)
VIZSGA-2 (***Tárgy+Helyezés***, Diák)

Matematikailag a két megoldás tökéletesen egyenértékes. Viszont el kell dönteni, hogy melyik páros legyen az elsődleges kulcs. Szemantikailag mindkettő értelmes. Ha valaki a diákok felől nézi a vizsgákat, akkor az első változatot részesíti előnyben. Ha viszont a tárgy az elsődleges szempont, akkor a második egyed jelenti a jó választást. Mindkét esetben a másik páros jelenti az alternáló kulcsot. A megoldás tehát adott, csak éppen az elsődleges-alternáló döntés jelent tervezési problémát.

Ilyen választási gonddal többször kell szembenézni. Lássuk csak a 15.3 példa esetét.

15.3 példa

RENDELÉSTÉTEL (Rendelésszám, Tételsorszám, Cikkszám, Rendelt-mennyiség)

A Tételsorszám a cikknek a Rendelésszámon belüli megjelenési sorrendjét mutatja. Egy-egy rendelésben nem szerepelhet kétszer ugyanaz a Tételsorszám és ugyanaz a Cikkszám érték. Ezért a példában két összetett kulcsjelölt is van: a Rendelésszám+Tételsorszám és a Rendelésszám+Cikkszám páros. A két kulcsjelölt átfed. Viszont azokon kívül nincs más meghatározó az egyedben. Ezért az egyed BCNF alakban van és nem szükséges azt megbontani. Csak éppen ki kell választani az elsődleges kulcsot.

A probléma a 15.1 példa által felvetett gonddal analóg, de attól egy hajszálnyit mégis eltér. A Tételsorszám teljesen mesterséges adat: a rendeléstételek papíron való elrendezését jelenti. Az értelmes kérdés, hogy kik voltak az első helyezettek az egyes tárgyakban. Viszont az, hogy melyek a rendelések első tételei, senkinek semmit nem mond. Ezért a 15.3 példa esetében az egyetlen lehetséges értelmes elsődleges kulcs a Rendelésszám és a Cikk-szám párosa. A Rendelésszám+Tételsorszám csak alternáló kulcs lehet.

Ezen a ponton megint fel kell hívnunk a figyelmet egy apró titokra. Nem egészen igaz az, hogy a 15.2 példa két megoldása egyenértékes és a 15.3 példa esetében sem csak a megjelölt indokok miatt döntünk a Rendelésszám+Cikkszám elsődleges kulcs mellett.

Sohasem szabad az egyedeket önmagukban, elszigetelten szemlélni. Ha a diákok tárgybeli helyezéséről van szó, akkor feltételezhető, hogy a diákokról egyéb ismereteket is akarunk vezetni. Tehát lesz külön DIÁK egyedtípusunk. Egészen bizonyos, hogy a rendeléstételekben lévő Cikkszám mögött a cikkre vonatkozó egyéb adatok sora rejtőzik. Ezért lesz CIKK egyedtípusunk is. Ha a VIZSGA kulcsául a Diák+Tárgy, a RENDELÉSTÉTEL azonosítójaként a Rendelésszám+Cikkszám együttest választjuk, akkor ezekkel a megoldásokkal tökéletesen tükrözzük a diák-tárgy és rendelés-cikk viszonyokat. Ezzel szemben a diák helyezése és a rendeléstétel sorszáma nem kapcsol másik egyed felé.

A fentiek miatt leszögezhetjük, hogy az összetett kulcsjelöltek közül mindig azt célszerű elsődlegesnek választani, amelynek részei más egyedek felé is mutatnak. Ezt követeli meg az opcionalitás elve is. Hiszen lehet, hogy egy diák tanul egy tárgyat - van diák-tárgy viszony -, de abból nem vizsgázik, tehát helyezésről nem is lehet szó. Nincs értelme a tételsorszámnak akkor, ha egy rendelés csak egyetlen cikkre vonatkozik.

Még két érvet kell említenünk a „természetes” összetételű - más jelenségek felé mutató - azonosítók választása mellett. Sokan fejből ismernek ezernyi cikkszámot. Ezért számukra a rendeléstételek Cikkszám szerinti lekérdezése nem gond. De ki tudja fejből, hogy hányadik tétel vonatkozik a csavarra mint cikkre a különböző rendelésekben? Nem fogalmi, hanem gyakorlati kérdés az, hogy kevés kezelő támogatja az alternáló kulcsot. A kulcs és az alternáló kulcs (részeinek az) értéke a modellezés szabályai szerint sohasem változhat. Ha viszont a 15.2 példa második megoldásában a Diák tulajdonságot leíróként értelmezi a kezelő, akkor annak tartalmát minden további nélkül meg lehet változtatni. Marika ért el föciből első helyezést, de a Marikát tévedésből át lehet írni Pistikére.

Bár lehet, hogy már picit unalmas, még egy esetet mutatunk be. Lásd a 15.4 példát.

15.4 példa

AKCIÓ (Akciókód, Vevőkód, Vevőnév, Kedvezmény)

A példa magyarázata: Vásárlási akciókat indítunk a vevők részére. Egy akcióban több vevő vehet részt és egy vevő több akciónak a részese is lehet. Ezért a ténylegesen igénybevett kedvezményt az Akciókód önmagában nem azonosítja. Ezt a kulcsrészt ki kell egészíteni a vevőre vonatkozó ismerettel. Csakhogy itt leszünk gondban akkor, ha kijelentjük, hogy a Vevőnév egyedi tartalmú. Azaz két vevőnek nem lehet azonos a neve.

Ezért az AKCIÓ egyednek két összetett kulcsjelöltje van: az Akciókód+Vevőkód és az Akciókód+Vevőnév páros. Az AKCIÓ egyed 3NF alakban van, mert az egyetlen nem-kulcs tulajdonsága (Kedvezmény) teljes függéssel függ a lehetséges kulcsoktól és mástól nem. Ugyanakkor az egyed nem BCNF formájú, mert van benne olyan meghatározó (Vevőkód és Vevőnév), amely nem ennek az egyednek a kulcsa.

A Vevőnév nem teljes függéssel függ az Akciókód+Vevőkód együttesétől, mert a vevő nevét a Vevőkód meghatározza. A Vevőkódot nem teljesen határozza meg az Akciókód és Vevőnév összetétel, mivel a Vevőkód függ a Vevőnévtől. Jelen esetben két olyan átfedő kulcsjelöltről van szó, amelyeknek kizáró részei egymást kölcsönösen meghatározzák.

Ebből a helyzetből komoly karbantartási anomália következik. Ugyanaz a Vevőnév érték több egyedelőfordulásban is szerepelhet. Az „111” kódú vevő „X” nevét minden további nélkül átírhatjuk „Y”-ra az AKCIÓ egyik sorában, miközben az ugyancsak „111” Vevőkód értéket tartalmazó másik sorban ezt nem tesszük meg. Arról pedig ne is beszéljünk, hogy az AKCIÓ fizikailag redundáns, hiszen többszörösen tartalmazza az „111 - X” értéksort.

Ezt a tárolási és aktualizálási anomáliát csakis az AKCIÓ megbontásával kerülhetjük el. Az teljesen világos, hogy az egyedből ki kell emelnünk a Vevőkód és Vevőnév párost a kölcsönös függés miatt. Az új VEVŐ egyed esetében döntenünk kell, hogy a két kulcsjelölt közül melyik legyen az elsődleges. Kézenfekvőnek látszik, hogy a VEVŐ (***Vevőkód***, Vevőnév) egyed mellett döntünk úgy, hogy a Vevőnév alternáló kulcs.

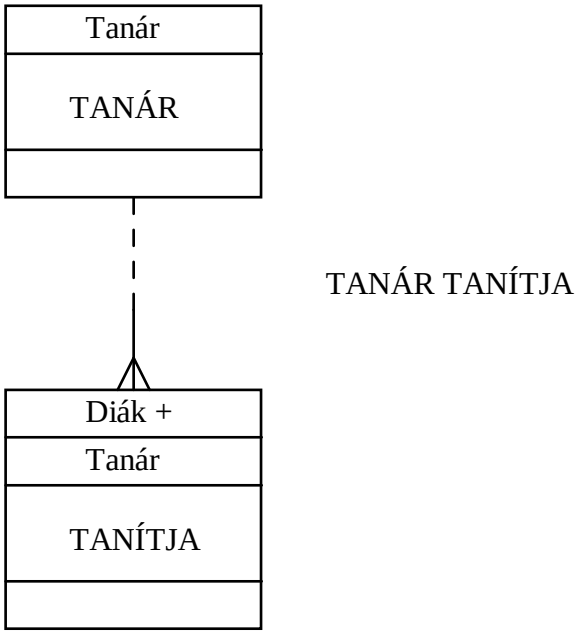
A kérdés az, hogy milyen tulajdonságok maradjanak az eredeti AKCIÓ egyedben? Két választásunk van: az AKCIÓ-1 (***Akciókód+Vevőkód***, Kedvezmény) és az AKCIÓ-2 (***Akciókód+Vevőnév***, Kedvezmény) egyedtípus. Mindkét megoldással formailag BCNF egyedeket nyerünk, mert már minden meghatározó egyben kulcsjelölt is.

Ezért a szakirodalom a két megbontást teljesen egyenértékesnek tartja. Bármennyire is sajnáljuk, ismét ellent kell mondanunk. Ha a VEVŐ egyedben a Vevőkód az elsődleges kulcs, akkor csakis az AKCIÓ-1 megbontás a logikus megoldás. Bár az alternáló kulcsok matematikai értelemben valóban egyenrangúak, a gyakorlatban sokszor nem azok. A Vevőkód értékét csak rendkívül kivételes esetben illik módosítani. A Vevőnév tartalmát viszont bármikor meg lehet változtatni. Arról pedig ne is beszéljünk, hogy a fogalmi modellezés mögött van némi praktika is. A Vevőkód X, a Vevőnév Y karakteres - sokkal hosszabb. Ezért minden józan tervező az AKCIÓ-1 egyedtípust fogja alkalmazni.

15.2.4 A normalizálás negyedik lépése

Ha egyedtípusaink már legalább 3NF alakban vannak, akkor meg kell nézni, hogy az egyed minden meghatározó tulajdonsága kulcsjelölt-e (ld. D 15/2). Ha nem az, akkor a kulcstörő függés mentén (ld. D 15/1) meg kell bontani az egyedtípust. Így a 15.1 ábra 3NF alakú, de mégis karbantartási anomáliákban szenvedő egyetlen egyedtípusából a 15.3 ábra két egyedtípusához jutunk.

Amint azt már a 2NF és 3NF kapcsán is tapasztaltuk, a dekompozíció hierarchikus szerkezetet eredményez (ld. 14.6 ábra). Az eredeti egyedből (TANÍTÁS) kialakított új egyedek (TANÁR és TANÍTJA) egymással fölé/alárendeltségi viszonyban állnak úgy, hogy a leválasztott tulajdonságok alkotják a fölérendelt egyedet. Lásd a 15.4 ábrát.



15.4 ábra: A TANÍTÁS modellrészlet diagramja

Eltekintve az előző alpontban említett problémáktól, a BCNF mechanikus normalizálás kiválóan működik. Csakhogy sohasem szabad mechanikusan normalizálni. A 15.5 példa kapcsán mutatunk rá az új megfontolásokra.

15.5 példa

UTCA (***Kerületkód+Utcaszám***, Irányítószám)

Nem valós, pusztán a problémákat bemutatni óhajtó feltételeink a következőek: Az Utcaszám a kerületen belüli relatív sorszám (mesterséges adat). Az utca nem nyúlik át több irányítókörzeten, különben nem állna fenn a Kerületkód+Utcaszám → Irányítószám függés és az Irányítószám nem lehetne az UTCA tulajdonságtípusa. Minden körzet csak adott kerületbe tartozik, ezért fennáll az Irányítószám → Kerületkód függés.

Az UTCA egyed 3NF alakú. Nincs benne részleges vagy tranzitív függés. Viszont nincs BCNF formában, mert van benne olyan meghatározó (Irányítószám), amely nem kulcsjelölt. A külső kulcstörés tipikus esetével állunk szemben. Az Irányítószám leíró tulajdonság meghatározza a Kerületkód+Utcaszám összetett kulcs egyik részét (Kerületkód). Ezért már annyi bizonyos, hogy létre kell hozni a KÖRZET (***Irányítószám***, Kerületkód) új egyedet. A kérdés csak az, hogy ezek után mi marad az eredeti egyedtípus (UTCA) tartalma.

Rossz megbontás lenne a KÖRZET (***Irányítószám***, Kerületkód) és az UTCA (***Kerület-kód+Utcaszám***) páros. Ezzel a dekompozícióval elveszne az az ismeretünk, hogy a konkrét utca melyik irányítókörzetbe tartozik. A normalizálás szabályai szerint a KÖRZET (***Irányítószám***, Kerületkód) és az UTCA (***Irányítószám+Utcaszám***) párost kellene létrehoznunk, mert az a veszteségmentes, a matematikai értelemben vett tökéletes lebontás.

Csakhogy... Csakhogy az Utcaszám eredetileg a kerületen belüli relatív sorszámot jelentette. Ha most ezt a tételt az Irányítószám-hoz kötjük, akkor tartalmilag „lyukas" lesz az adatmodellünk. Az I. kerületen belül az utcaszámok folytonosak (1, 2, 3 stb.) voltak. Az irányítókörzeteken belül pedig „véletlenszerűek" lesznek (3, 9, 18 stb.), hiszen az utcákat eredetileg nem körzetek szerint sorszámoztuk.

Amint látjuk, a mechanikus normalizálás matematikai értelemben mindig jó eredményre vezet, de egyáltalán nem biztos, hogy kielégíti ismerettükrözési igényeinket. Ezért a 15.5 példa normalizálása során nem mechanikusan fogunk eljárni. Lásd a 15.6 példát.

15.6 példa

KERÜLET	(***Kerületkód***, ...)
KÖRZET	(***Irányítószám***, ... *Kerületkód*)
UTCA	(***Irányítószám+Utcaszám***, ...)

KERÜLET	(***Kerületkód***, ...)
KÖRZET	(***Irányítószám***, ... *Kerületkód*)
UTCA	(***Utcaszám***, ..., *Irányítószám*)

Az első esetben az Utcaszám tételt átértelmezzük az Irányítószám tartalmán belüli relatív sorszámra. Így az a tétel az összetett kulcs része lehet. A második változatban az Utcaszám kerülettől és körzettől független egyedi azonosító. Ezért nem alkot párost az Irányítószámmal, hanem azt meghatározza. Mindkét esetben helyreáll a kerület-körzet-utca természetes hierarchia.

Ez a példa jól mutatja, hogy egyes adatmodellezési problémákat nem normalizálással küszöbölünk ki. A normálformahiba alapján (a 15.5 példa UTCA egyede nem BCNF alakú) fedezzük fel a problémát a matematikára támaszkodva. Viszont szemantikai alapon (az Utcaszám átértelmezésével) keressük a helyes megoldást.

15.3 A negyedik normálforma

A nem-normalizált egyedtípusokban ismétlődő tulajdonságok vagy csoportok találhatók. A 13.4 pontban próbáltuk elmondani, hogy a 0NF alakú egyedeket miképpen lehet 1NF formára hozni. Ott még nem említhettük, hogy az ismétlődéseknek van egy sajátos alternáló változata is, amelynek a kiküszöbölésére más technikát kell alkalmazni. A 15.5 ábra egyedtípusán szemléltetjük ezt a speciális esetet.

TANFOLYAM

Kurzus	Tanár	Téma
Fizika	Zöld	Mechanika
Fizika	Zöld	Optika
Fizika	Barna	Mechanika
Fizika	Barna	Optika
Fizika	Fekete	Mechanika
Fizika	Fekete	Optika
Matek	Fehér	Algebra
Matek	Fehér	Geometria

15.5 ábra: Furcsa ismétlődést tartalmazó TANFOLYAM egyed

Ezt a példát Date [22] kiadványból vettük (256. oldal). Sajnos a profi szakírók példái néha egy picit hibásak. Szándékosan mutatjuk be ezeket a féloldalas példákat. Egyrészt azért, hogy az olvasó a helyzetet értékelhesse, ami a lényeg. Másrészt azért, hogy megtanulja kritikusan szemlélni a modellezési kérdéseket. Támadjon, vitatkozzon, érveljen - akár az íróval szemben is - mert hiszen a valóság hű feltárása a jó modellező feladata.

Most nézzük először az érdemi mondanivalót.

Egy tanfolyamon több tanár több témát tanít. A lényeg az, hogy ha egy tanár a kurzuson részt vesz, akkor annak minden témáját tanítja. Megfordítva: Ha egy téma a kurzus része, akkor azt minden tanár tanítja, aki a tanfolyamon egyetlen témát is okít. Zöld tanár úr tanítja a mechanikát és az optikát, mert ez a két téma tartozik a fizika kurzushoz. Az optikát Barna tanár úr is előadja, ha a kurzuson már a mechanika oktatásában is részt vesz.

15.3.1 A többértékű függés és az általa okozott problémák

A TANFOLYAM egyedtípus BCNF alakban van. Ez a tábla ugyanis ún. ***csupakulcs*** (angolul: all-key) egyed. Mind a három tétel az egyetlen összetett kulcs része, ezért nincs a táblában olyan tulajdonság, amely meghatározó, de nem kulcsjelölt. Ennek ellenére a 15.5 ábra egyedtípusa karbantartási anomáliákat mutat. Például, ha Zöld tanár úr kilép, akkor nem egy, hanem több egyedelőfordulást kell törölni. Ha Piros tanárnő a fizika kurzuson oktatni kezd, akkor többszörös bevitelre (mechanika, optika) van szükség. Végül akkor, ha Fehér tanárnő férjhez megy és nevét Kékre változtatja, többszörös karbantartásra van szükség. Mindezeket a bajokat a Téma szempontjából is átgondolhatjuk.

Mivel a TANFOLYAM BCNF alakban van, az eddig ismertetett normalizálási eljárások nem alkalmazhatók a nyilvánvalóan látható karbantartási gondok kiküszöbölésére. Új koncepcióra van szükségünk.

Az egyedtípusban a Tanár és Téma párosok egy adott rendszer szerint jelennek meg. Általánosan ezt a mintát a következő módon fejezhetjük ki:

Ha megjelenik a táblában a <k1,t1,m1> és a <k1, t2, m2> sor,
akkor szerepelnie kell abban a <k1, t1, m2> és a <k1, t2, m1> soroknak is.

Ha az első kurzuson (k1) az első tanár (t1) az első témát (m1), a második tanár (t2) a második témát (m2) tanítja, akkor az első tanárnak is kell tanítania a második témát és a másodiknak is az elsőt.

A hasonló helyzetek feloldására vezették be a ***többértékű függés*** (angolul: multivalued dependency) fogalmát [25]. Ennek rövid neve ***MVD***, jele pedig „—>>”.

D 15/3 **Az E egyedtípus B tulajdonságtípusa akkor és csak akkor többértékű függéssel függ az egyedtípus A tulajdonságától, ha az adott A-értéknek megfelelő B-értékek készlete csak az A-tól függ és független az egyed C tulajdonságától.**

Példánkban a Tanár többértékű függéssel függ a Kurzus tulajdonságon, mert a Kurzus értékhez (Fizika) megfelelő Tanár értékhalmaz (Zöld, Barna, Fekete) tartozik és a Téma nem határozza meg a Tanár-t (minden témát több tanár okíthat).

Vegyük észre, hogy a többértékű függés mindig párosával jár. Ha fennáll az A —>> B függés, akkor léteznie kell az A —>> C függésnek is. Ezért az MVD-t általában csak olyan egyedtípuson szokták értelmezni, amely legalább három tulajdonságot tartalmaz.

A többértékű függés nem más, mint generalizált funkcionális függés. Vagyis az FD az MVD olyan speciális esete, amelyben a függő értékkészlet egyetlen tételre korlátozódik. Mivel az MVD mindig páros, annak jelölését így is szokták rövidíteni: A —>> B | C.

15.3.2 A normalizálás ötödik lépése

Mielőtt rátérnénk a lényegre, elmondjuk, hogy a 15.5 ábra példája miért hibás. Ha egy tanár nem oktathat többféle kurzuson és egy téma nem szerepelhet több tanfolyamon - a példa ezt sugallja -, akkor a Tanár és a Téma funkcionálisan meghatározza a tanfolyamot. Ebben az esetben pedig nincs szükség az MVD-re. Tehát a továbbiakhoz azt kell feltételeznünk, hogy vannak tárgyak, amelyeket több kurzuson is oktatnak.

Most nézzük a normalizálást. A TANFOLYAM egyedben két MVD-t fedezhetünk fel. A Kurzus többértékűen meghatározza a Tanárt és attól többértékűen függ a Téma is. Ha kiválasztunk egy Kurzus értéket, akkor meg tudjuk határozni a Tanárok ***készletét***, függetlenül a Téma értékétől. Ugyanez vonatkozik fordítva a Témák alhalmazára is.

Az MVD karbantartási anomáliát okoz. Ezért a TANFOLYAM egyedet meg kell bontani a két MVD mentén, vagyis a többértékű meghatározó és függő párost külön egyedtípusba kell kiemelni. Az eredményt a 15.6 ábra mutatja.

KURZUS TANÁRAI

Kurzus	*Tanár*
Fizika	*Zöld*
Fizika	*Barna*
Fizika	*Fekete*
Matek	*Fehér*

KURZUS TÉMÁI

Kurzus	*Téma*
Fizika	*Optika*
Fizika	*Mechanika*
Matek	*Algebra*
Matek	*Geometria*

15.6 ábra: Átalakított TANFOLYAM modellrész

Az eredeti TANFOLYAM egyedben az volt a probléma, hogy olyan MVD-t tartalmazott, amely nem volt egyben FD is. A RENDELÉS (***Rendelésszám,*** Rendelésdátum, Vevőkód) egyedben a Rendelésszám —>> Rendelésdátum | Vevőkód MVD-páros egyben funkcionális függés is, azaz fennállnak a Rendelésszám → Rendelésdátum és Rendelésszám → Vevőkód függések és ezért ezt az egyedet nem kell megbontani.

A 15.6 ábra két csupakulcs egyedében már nincs problémát okozó MVD. Ezért nem lépnek fel a karbantartási anomáliák. Így tehát a 15.6 ábra megoldása lényegesen jobb, mint a 15.5 ábra adatbázis-részlete.

D 15/4 **Az E egyedtípus 4NF alakban van akkor és csak akkor, ha az abban lévő bármilyen A —>> B MVD egyben A → B FD is.**

Mivel a 15.5 ábrában a Kurzus —>> Tanár MVD nem volt FD is, a TANFOLYAM egyed csak BCNF alakú volt. A 15.6 ábra két egyede viszont már 4NF alakú. (Az MVD-t gyakorlatilag csak legalább háromoszlopos táblában szoktuk vizsgálni, noha elméletileg a kétoszlopos tábla is megbontható két unáris egyedre az MVD mentén.)

Meg kell jegyeznünk, hogy egyesek a 15.5 ábra ismétlődését úgy akarják feloldani, hogy egy tanárhoz csak egy tárgyat adnak meg. Ezt azon az alapon teszik, hogy úgyis tudjuk, hogy a tanár mindegyik témát okítja. Gondolatmenetük kétszeresen is helytelen. Egyrészt így felvetődik a kérdés, hogy melyik témát vigyék a tanárhoz. A végén az egyik tárgyat egyik tanárnál sem adják meg. Másrészt az ismeretekre vonatkozó tudásunkat a modellnek mindig világosan, expliciten kell tükröznie. Nem szabad megengedni, hogy a tudás implicit legyen, csak a fejekben vagy a programokban létezzen.

Végül rá kell mutatnunk, hogy a TANFOLYAM egyednek többféle megbontása is létezhet. A 14.8 példában mutattuk be a jó és a rossz lebontást a funkcionális függés esetén. Akkor megállapítottuk, hogy az A → B, B → C és A → C függések fennállásakor az (A,B) és (B,C) páros adja a helyes dekompozíciót, míg az (A,B) és (A,C) páros szerinti megbontás karbantartási gondokat okoz. A szakírók szerint ugyanez a megállapítás vonatkozik az MVD-re is. Tehát a 15.6 ábra megoldásánál jobb lebontás a (Kurzus+Tanár) és (Tanár+Téma) vagy a (Kurzus+Téma) és (Téma+Tanár) páros.

Ezzel a megállapítással nem teljesen értünk egyet. Bármelyik megbontással két csupakulcs egyedet kapunk. Ha ez nem lenne igaz, akkor az eredeti relációban létezett volna funkcionális függés és megbontásához nem lett volna szükség az MVD alkalmazására. A csupakulcs relációk tartalmaznak ugyan közös tulajdonságot, de az a mi felfogásunk szerint nem kapcsoló tétel. Azért nem az, mert nem áll fenn az „egyik egyednek kulcsa, a másiknak nem” feltétel. Az M:N-es kapcsolódás mindig kétes szemantikai eredményekre vezet, amint azt előző kiadványunkban kimutattuk. Ezért tökéletesen közömbös, hogy az eredeti hármast a három lehetséges kombináció közül melyik két párosra bontjuk le.

15.3.3 Megjegyzések a többértékű függéshez

A 4NF alak kiküszöböl egy olyan karbantartási anomáliát, amit a BCNF még megenged. Ezért a 4NF alak tökéletesebb a BCNF-nél. Természetesen minden olyan egyed, amely 4NF alakú, egyben BCNF formájú is. Ezért nincs szükség a lépésenkénti normalizálásra: a rossz MVD-k kiküszöbölésével egyből a 4NF alakhoz juthatunk.

A 4NF mindig megtalálható, de nem biztos, hogy alkalmazni is akarjuk. Már a BCNF alaknál is tapasztaltuk, hogy az eredményezett egyedtípusok esetleg nem függetlenek egymástól, mert az eredeti egyed ***atomi*** (ld. 15.2.3 alpont) volt. Példánk esetében is felléphet karbantartási probléma. Valaki beilleszthet egy új sort a KURZUS TANÁRAI egyedbe anélkül, hogy megvizsgálná: az illető valóban tanítja-e a kurzus összes témáját.

Példánk mutatja, hogy csak meglehetősen összetett feltételek mellett lép fel az MVD-probléma. Ezért a gyakorlatban igen ritkán fordul elő, hogy az ötödik normalizálási lépést végre kell hajtani. Ezzel szemben egyes tervezők az MVD bűvöletébe esnek és úton-útfélen MVD-

megoldásokat keresnek. Általában háromféle tévedést szoktak elkövetni. Az egyik az, hogy elfeledkeznek arról a kitételről, miszerint az MVD mindig „párosával jár” és ott is többértékű függést keresnek, ahol az nem alternáló jellegű. A másik az, hogy az MVD-t nem csak a csupakulcs táblákon vizsgálják, holott csakis azok esetében léphet fel a 4NF igénye. A BCNF alakú, nem-csupakulcs egyed ugyanis eleve egyben 4NF formában is van.

A legsúlyosabb tévedés - amivel a szakirodalom példáiban is sokszor lehet találkozni - az, hogy az A –/–> B funkcionális függetlenséget B —>> A (tehát fordított irányú) MVD-nek értékelik. Mert hiszen akkor, ha az A nem határozza meg a B-t, akkor a B-hez több A érték, az A értékkészlete tartozik. A D 15/3 meghatározás azon a kitételéről, hogy „a C-től függetlenül” egyszerűen elfeledkeznek. Ha a TANFOLYAM példában nem lenne igaz az, hogy minden a kurzuson résztvevő személy a tanfolyam minden témáját tanítja, akkor is fennállna a Kurzus –/–> Tanár függetlenség, de szó sem lehetne a Kurzus —>> Tanár és a Kurzus —>> Téma MVD-párosról, hát még a Tanár —>> Kurzus függésről. Nem lépne fel a karbantartási anomália és ezért a TANFOLYAM egyedet nem lehetne - és nem is volna értelme - megbontani.

Most pedig elárulunk egy apró titkot. A magunk részéről a 4NF alakot szép, de kicsit felesleges elméleti zsonglőrködésnek tartjuk. A 15.5 ábra TANFOLYAM egyedtípusa - erre már utaltunk - rejtett ismétlődést tartalmaz. Tehát - ha így tekintjük - nem BCNF alakú, hanem nem is normalizált (0NF). A 15.7 ábra táblája ugyanazt az ismeretet hordozza, mint a 15.5 ábra egyede.

TANFOLYAM

Kurzus	Tanárok	Témák
Fizika	Zöld Barna Fekete	Mechanika Optika
Matek	Fehér	Algebra Geometria

15.7 ábra: Ismétlődést tartalmazó TANFOLYAM egyed

Mármost miként szoktuk az ismétlődéseket kiszűrni? Úgy, hogy az ismétlődő részt az eredeti egyed kulcsával együtt új egyedbe visszük. Példánkban két ismétlődés van, ezért a 15.6 ábra két egyedét fogjuk kapni az ismétlődések levágásával. Mivel az eredeti egyedben csak a Kurzus kulcs marad, megfontolás tárgya, hogy az egyoszlopos egyedtípust megőrizzük-e vagy sem.

Mi ilyen helyzetben meg szoktuk tartani az egyetlen tulajdonságot tartalmazó egyedtípust is. Ennek két oka is van. Egyrészt egészen bizonyos, hogy a tanfolyamokat nemcsak nevükkel, hanem egyéb ismeretekkel is le akarjuk írni. (Százszor elmondtuk már, hogy nem szabad az egyedeket önállóan, a többiektől függetlenül nézni.) Másrészt a KURZUS TANÁRAI (***Kurzus+Tanár***) és a KURZUS TÉMÁI (***Kurzus+Téma***) között nem definiálható kapcsolattípus. Ezzel szemben mindkét egyed valós kapcsolatban áll a TANFOLYAM (***Kurzus***, ...) egyedtípussal. Ekkor ugyanis fennáll az a feltétel, hogy a Kurzus az egyik egyedben kulcs, a másikban nem az (mi a kulcsrészt nem tekintjük kulcsnak).

15.4 Az ötödik normálforma

Eljutottunk az ötödik normálalakhoz (5NF). Ez a legvégső normálforma, amelynél - matematikai értelemben és a dekompozíciós eljárás alkalmazása esetén - nem lehet jobbat találni [26]. A forma szemléltetésére szolgáló meglehetősen kacifántos példát ismét Date úrtól vettük [22, 260. oldal], aki azt maga is „patologikus”-nak nevezte. Lásd a 15.8 ábrát. (Az olvasó készüljön fel a legrosszabbra.)

SZÁLLÍTÁS

SSZ	PSZ	CSZ
S1	P1	C2
S1	P2	C1
S2	P1	C1
S1	P1	C1

15.8 ábra: SZÁLLÍTÁSI modellrészlet

A példa magyarázata meglehetősen bonyolult. Vannak szállítók, projektek és cikkek. Ezek értékkészletei körkörösen egymáshoz kötöttek. Egy szállító meghatározott cikkeket adott projektekre szállít. Ha egy cikket szállít, akkor azt minden a cikket érintő projektre biztosítja és megfordítva: ha egy projekthez kapcsolódik, akkor annak minden cikkét szállítja. Egy cikket meghatározott szállító biztosít adott projektre. Ha a cikket egy szállító szállítja, akkor azt minden projektre teszi, amelyben a cikk szerepel és megfordítva. Ugyanígy a projekt szempontjából is egymáshoz kötődnek a szállítók és a cikkek.

Mindezt így roppant nehéz megérteni. A matematikai formula így hangzik:

ha megjelenik a táblában az <S1, P1, C2>, <S2, P1, C1> és a <S1, P2, C1> sor,
akkor szerepelnie kell abban a <S1, P1, C1> sornak is.

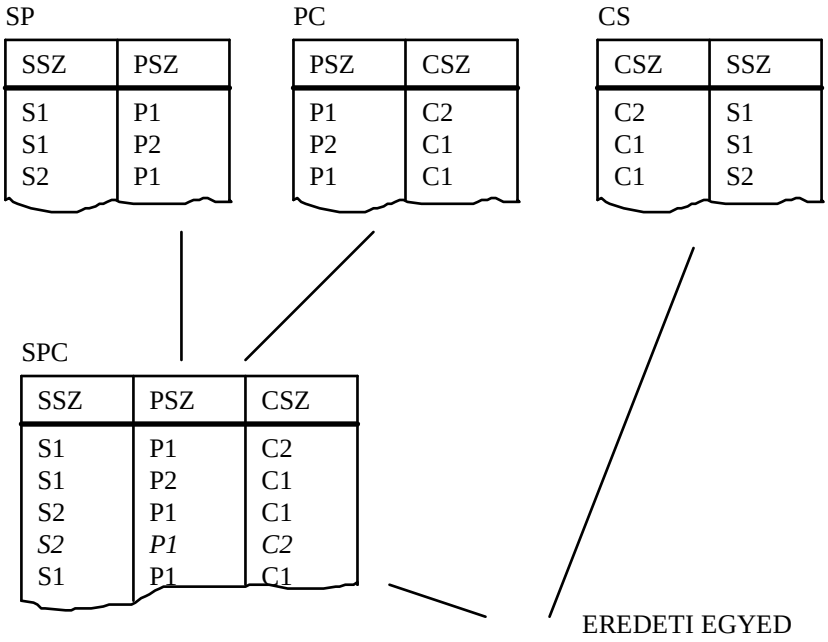
15.4.1 A kapcsolásfüggés és az általa okozott problémák

A SZÁLLÍTÁS csupakulcs egyed igen ravasz karbantartási problémákat rejteget. Tekintsük úgy, mintha a 15.8 ábra csak az első két sort tartalmazná. Most próbáljuk meg beilleszteni a harmadik sort. Ekkor azzal együtt a negyediket is be kell vinni. Az első szállító szállít a P1 projektre (első sor) és szállítja a C1 cikket (második sor). Azonban a beillesztésig a P1-C1 páros nem szerepelt az első két sorban. Vagyis nem kötöttük ki, hogy a P1 projektre a C1 cikket is szállítják. Az S2-P1-C1 sor bevitelével ezt megtesszük. Mivel az S1 szállító részt vesz a P1 projekt ellátásában, arra a C1 cikket is szállítania kell. Ezzel szemben a negyedik sor bevitele nem követeli meg a harmadik beillesztését.

A törlésekkel is bajunk van. A harmadik sort önmagában törölhetjük, mert nem veszik el az az ismeret, hogy a P1 projekthez tartozik a C1 cikk. Viszont a negyedik sor törlése együtt jár másik sor megszüntetésével is. (Melyikével?)

A karbantartási anomáliák miatt a SZÁLLÍTÁS egyedet elvileg meg kell bontani, bár az 4NF alakban van. Ezt három módon tehetnénk: Szállító-Cikk/Cikk-Projekt, Szállító-Cikk/Projekt-Szállító és Cikk-Projekt/ Projekt-Szállító egyedpárosokat alkotva. Azonban a 15.9 ábra baloldali

része mutatja, hogy a párosra bontás nem lehetséges. Azért nem, mert a párosok összekapcsolásával nem az eredeti egyedet nyerjük vissza. Téves ismeretekre vezetne a pusztán páros kapcsolás. Ezért mindhárom egyedre szükség van, amelyek kettős kapcsolása már megbízható eredményt ad. Ezt mutatja az ábra jobboldali része.



15.9 ábra: A három kivetített egyed összekapcsolása

Ha csak két táblázatot kapcsolunk össze (ld. baloldal), akkor olyan sort is kapunk, amely nem szerepelt az eredeti táblában (lásd a dőltbetűs előfordulást). Ha viszont a harmadik táblázattal összekapcsoljuk az előző művelet részeredményét, akkor visszakapjuk az eredeti táblát. Emiatt a körkörös viszony miatt nevezik az ilyen tulajdonságfüggést ***kapcsolásfüggésnek*** (angolul: join dependency).

D 15/5 **Az E egyedtípus akkor és csak akkor tesz eleget a kapcsolásfüggésnek, ha X, Y, ..., Z kivetítéseinek az összekapcsolásával helyreállítható úgy, hogy X, Y, ..., Z az E egyedtípus tulajdonságtípusainak a részhalmazai.**

A kapcsolásfüggés rövid neve ***JD***, jele pedig * (X, Y, ..., Z). Példánk esetében az SP, a PC illetve a CS a részhalmazok és ezért fennáll a * (SP, PC, CS) kapcsolásfüggés.

15.4.2 A normalizálás hatodik lépése

Amiként a többértékű függés (MVD) a funkcionális függés (FD) generalizációja, ugyanúgy a kapcsolásfüggés (JD) az előző kettő általánosítása. Tehát az FD és az MVD a JD speciális esete. A SZÁLLÍTÁS egyedben meglehetősen nehezen áthidalható karbantartási problémák léptek fel. Ennek az az oka, hogy az egyedben kimutatható JD nem MVD és nem FD. Ezért az egyedet meg kell bontani. Ez a dekompozíció - a korábbiakkal ellentétben - nem két, hanem több, egymáshoz körkörösen kapcsolható egyedet eredményezett.

D 15/6 **Az E egyed akkor és csak akkor van 5NF alakban, ha az egyedben lévő minden kapcsolásfüggés csak az egyed kulcsjelöltjei között létezik.**

Az ötödik normálformát (5NF) kivetítés-összekapcsolás (angolul: projection-join) alaknak is nevezik és ezért PJ/NF módon is jelölik. A 15.9 ábra három egyede ebben a normálformában van. Már nem mutatják a SZÁLLÍTÁS egyed karbantartási problémáit. Azonban nyilván fellép egy karbantartási nehézség. Bármelyik egyedet is aktualizáljuk, másikhoz is hozzá kell nyúlnunk. A származtatott egyedek egymástól nem függetlenek, mert az eredeti egyed atomi volt.

Most ismét el kell mondanunk, hogy minden 5NF alakú egyed természetesen 4NF formában is van, mivel a JD az MVD általánosabb esete. Az is világos, hogy a kapcsolásfüggést mutató egyed ismeretveszteség nélkül lebontható e függés mentén.

Az 5NF-módon való normalizálás elvi lehetőségét egy másik példán keresztül is bemutatjuk. Vegyük csak alapul a VEVŐ (Vevőkód, Vevőnév, Vevőcím, Vevőtípus) egyedet, amelyben feltételezzük a Vevőnév egyediségét. Ebben az egyedben több JD fedezhető fel a két kulcsjelölt (Vevőkód, Vevőnév) miatt. Lásd a 15.7 példát.

15.7 példa

* (Vevőkód, Vevőnév, Vevőtípus), (Vevőkód, Vevőcím)
* (Vevőkód, Vevőnév), (Vevőkód, Vevőtípus), (Vevőnév, Vevőcím)

Nem kétséges, hogy a két egyed illetve a három páros összekapcsolásával visszanyerjük az eredeti VEVŐ egyedet. Ez annak köszönhető, hogy mind a Vevőkód, mind a Vevőnév kulcsjelöltek és csakis ezeken alapulnak a kapcsolásfüggések.

Most nézzük meg a RENDELÉS (Rendelésszám, Vevőkód, Vevőnév) egyedet, amelyben azt feltételezzük, hogy a Vevőnév nem egyedi! Ennek a RENDELÉS-1 (Rendelésszám, Vevőkód) és RENDELÉS-2 (Rendelésszám, Vevőnév) módon való lebontása helytelen. Azért, mert az eredeti egyedben van olyan * (Vevőkód, Vevőnév) JD, amelyet nem az egyetlen kulcsjelölt implikál. A jelzett lebontással ez a JD elveszik. Ezért csak a RENDELÉS (Rendelésszám, Vevőkód) és VEVŐ (Vevőkód, Vevőnév) megbontás a jó.

15.4.3 Megjegyzések a végső normálformához

Minden normalizálási lépésnek az a lényege, hogy az eredeti egyed dekompozíciójával ***veszteségmentes*** szerkezetet kapjunk. Tehát az új egyedek összekapcsolásával visszanyerhessük az eredeti egyed ismereteit. Ha erre figyelve eleve a kapcsolásfüggéseket keressük, akkor nem kell törődnünk a részleges, tranzitív, kulcstörő és többszörös függésekkel. Biztosan és azonnal a létező legjobb - ötödik normálformájú - szerkezetet kapjuk, amint azt a legutóbbi RENDELÉS példánál is láthattuk.

Így szól az elmélet. Azonban a gyakorlat - sajnos - egészen más. Az előző alpont 15.7 példája mutatja, hogy a kapcsolásfüggések szerinti megbontás nem mindig egyértelmű. Ezen kívül roppant nehéz felfedezni a kapcsolásfüggéseket és átlátni, hogy azokat tényleg csak a kulcsjelöltek implikálják-e a D 15/5 definíció szerint. A gyakorlatban egyébként is nagyon ritkán fordulnak elő a 15.8 ábra példájához hasonló beteges helyzetetek. Azok esetében is kérdéses, hogy végre akarjuk-e tényleg hajtani a normalizálást, hiszen az eredmény-egyedek nem lesznek egymástól függetlenek.

Miután kötelességünknek eleget tettünk az 5NF bemutatásával, mindenkit lebeszélünk arról, hogy ezt a normálformát keresse és alkalmazza. Nem éri meg a fáradozást.

Most már csak azt kell megmagyaráznunk, hogy milyen értelemben tekintik „végsőnek” az 5NF alakot a szakértők. Az MVD a JD, az FD az MVD speciális esete. Az egyeden belüli tulajdonságfüggéseknek nincs olyan magasabb formája, amelynek a JD lenne a speciális válfaja. Ezért addig, ameddig a normalizálás lényegét a veszteségmentes, tehát összekapcsolással visszafordítható dekompozícióban keressük, az 5NF valóban a végső, a legtökéletesebb alak. Ezért ne várja az olvasó, hogy szó lesz ebben a könyvben 6NF, 7NF stb. normálformákról.

Léteznek azonban más alapú normálformák is, amelyek nem a dekompozíció elvére épülnek [[27] és [28]]. Ezek tárgyalásától kiforratlanságuk miatt eltekintünk. Viszont a következő fejezetben bemutatunk néhány saját, „titkos” normálformát. Ezeket nem kívánjuk előre minősíteni. Némelyikük az 5NF keretébe illeszkedik és csak az elmélet pontosítását szolgálják. Mások viszont nem a szigorú értelemben vett dekompozíciós eljárásba tartoznak és így másféle jellegűek, mint az 5NF alak. Másként szolgálják a modell javítását.

A fejezet zárásaként vissza kell térnünk a 15.1 pont bevezetőjének a végén elhangzott magánmorgolódásra. Egyes elméleti szakemberek briliánsan megoldják a gyakorlatilag nem is létező problémát. Ha visszatérünk a 15.7 példára, akkor az olvasó igazolva látja a magas elmélet iránti kételyeinket. A JD megoldja a nemlétező gondot (is). Megoldja? Egyetlen gyakorlati szakember sem fogja külön egyedbe helyezni a Vevőcímet úgy, ahogyan azt a 15.7 példa első változata mutatja. Ha a Vevőkód mellett a Vevőnév is kulcsjelölt, akkor ez egy gyakorlati tervezőt nem fog megzavarni. Semmiképpen sem fogja a 15.7 példa második változatát alkalmazni. Hanem elkönyveli, hogy a Vevőnév a Vevőkód alternáló kulcsa (ld. 14.7 pont), amely nyilván az egyetlen VEVŐ egyedet jellemzi és függéseinek a vizsgálatára nincs is szükség. Azért nincs, mert azok pontosan megegyeznek a Vevőkód függéseivel.

A fentiek miatt a tapasztalatlan gyakorlati tervező két hibát követhet el. Az egyik az, hogy egyáltalán nincs tekintettel az elméletre. Például nem alkalmazza az alapvető normálformákat. A másik az, hogy nem válogat az elméletekben és azokat minden kritika nélkül megpróbálja átvenni.

Ellenőrző kérdések - 15

15/01 Az alábbi kijelentések közül melyik igaz (I) és melyik hamis (H)?

- Minden kétoszlopos egyed eleve 3NF alakban van.
- Minden kétoszlopos egyed eleve BCNF alakban van.
- Minden kétoszlopos, egyetlen funkcionális függést mutató egyed 5NF alakú.
- Minden kétoszlopos egyed 5NF alakú.

15/02 Igaz-e (I) az az állítás, vagy hamis (H), amely szerint az E (A,B,C) egyed csak akkor egyenlő az E1 (A, B) és E2 (A, C) lebontásainak az összekapcsolásával, ha fennáll az A $\rightarrow$ B függés?

15/03 Igaz-e (I) vagy hamis (H) a következő négy kijelentés:

- Ha egy egyedben A $\rightarrow$ B és B $\rightarrow$ C, akkor A $\rightarrow$ C.
- Ha egy egyedben A $\rightarrow$ B és A $\rightarrow$ C, akkor A $\rightarrow$ {B, C}.
- Ha egy egyedben B $\rightarrow$ A és C $\rightarrow$ A, akkor {B, C} $\rightarrow$ A.
- Ha egy egyedben A $\rightarrow$ B és A $\rightarrow$ C, akkor {B, C} $\rightarrow$ A.

15/04 Igazak-e (I) vagy hamisak (H) a következő kitételek:

- A kulcstörő függést tartalmazó egyed mindig BCNF alakra hozható.
- A kulcstörő függést tartalmazó egyed mindig függetlenül karbantartható egyedekre bontható.
- A kulcstörő függést tartalmazó egyed mindig összekapcsolható egyedekre bontható.

15/05 Versenybridzset játszunk több asztalon. Vannak leosztások. Egy leosztásban több csapat vesz részt több asztalon. Minden leosztásban mindegyik csapat mindegyik asztalon részt vesz, amely asztalon játszik. Hányadik normálformában van a következő két egyed és mi velük a teendő:

BRIDZS (Leosztás, Csapat, Asztal)
BRIDZS (Leosztás, Csapat, Asztal, Eredmény)

16. A CSOPORTOK CSAPDÁI

16.1 Eltérő gondolkozásmódok

Vegyünk alapul három tulajdonságot: A, B és C. Tételezzük fel, hogy ezek között létezik az $A \rightarrow B$, $B \rightarrow C$ és $A \rightarrow C$ függéshármas. Most pedig tegyünk fel néhány kérdést.

Az első az, hogy melyik függés a felesleges, a rossz. Ha valaki csak ***mennyiségekben*** gondolkozik, akkor azonnal rávágja: „A harmadik, merthogy az a tranzitív." Viszont a ***minőségekre*** is figyelő tervező így szól: „A tranzitivitás ténye azt jelzi számomra, hogy itt valami baj van. Kétségtelen, hogy formailag az $A \rightarrow C$ függés felesleges, mert a másik kettőből következik. Ám az is előfordulhat, hogy az első két függés valamelyikét rosszul adták meg. Sőt, még az is, hogy az egyik A nem ugyanazt jelenti, mint a másik A."

„Miképpen lehet az, hogy az A nem mindig ugyanazt jelenti? Hiszen a normalizálás az egyedtípus dekompozíciója! Egy egyedben csak egy A lehet és így a felvetés értelmetlen." Szemben az ilyen elhamarkodott kitételekkel a tapasztalt modellező nem állít, hanem kérdez. Felteszi azt a második kérdést, hogy a három ominózus függést egy vagy több egyedtípuson belül kell-e értelmezni. A 13.3 pontban beszéltünk az egyeden belüli és az egyedek közötti függésekről. A részletekben elvesző tervező ***szeparált egyedtípusokat*** akar normalizálni. Az áttekintéssel bíró viszont a ***teljes optimális modellt*** keresi.

Az igen gyakorlott elemzők felvetik a harmadik kérdést is: „Tessék mondani, a három tulajdonság közül melyik ***egyszerű*** és melyik ***összetett***?". Mert ha az A összetett és annak része a B, akkor ugye nem közönséges tranzitivitással, hanem részleges függéssel állunk szemben. Még általánosabban szólva: a még nem eléggé jártas szakember megelégszik a függés tényének a felismerésével, míg a sokat látottak a függés sajátosságait is vizsgálják. Ugyanis az optimális adatmodell kialakítása során az összetett tulajdonságok - amiket ***csoportoknak*** fogunk nevezni - különleges elemzési problémákat okozhatnak.

Ennek a fejezetnek két célja van. Az első az, hogy átformálja a tervezők gondolkozásmódját. A másik az, hogy ismertesse a tulajdonságok közötti függések olyan speciális eseteit, amelyek a csoportokkal kapcsolatosak.

A köztudatban a normálforma és a normalizálás a relációs adatbázisok tervezéséhez kapcsolódó matematikai fogalmak. Ebből a felfogásból több probléma fakad. A reláció mint egység ***logikai szintű*** tényező. Ráadásul strukturálisan meglehetősen korlátos. Mint tudjuk, a relációs modell és az ilyen kezelők többsége nem támogatja a csoport ismereti egységét. Márpedig - mint ebben a fejezetben bizonyítani fogjuk - a csoport konstrukció nélkül nem lehet megalkotni az optimális adatmodellt. Ezért felfogásmódunkon változtatnunk kell. Egyrészt a normalizálást fel kell emelnünk a ***fogalmi szintű*** modellezés síkjára. Másrészt a tisztán matematikai szemléletet fel kell váltanunk a matematikával támogatott szemantikus látásmóddal. A csoportok által okozott modellezési problémák csak így oldhatók fel.

16.2 A függéstáblázat

Az eddigi normálformák kapcsán az olvasónak nyilván feltűnt két dolog. Az egyik az, hogy a 14. fejezetben tárgyalt részleges illetve tranzitív függések pontosan ugyanazokat a bajokat okozzák (ld. 14.2.1 és 14.3.1 alpont). A másik szembetűnő dolog az, hogy a kulcstörő függés igen nagy rokonságot mutatott a részlegessel, csak éppen kulcsrész tulajdonságra vonatkozott (ld. 15.2.1 alpont). Ezért az ember óhatatlanul elgondolkozik azon, hogy miért kellett a meghatározásokban - pl. D 14/1 - annyira hangsúlyozni a „nem-kulcs" kitételt. Van-e jelentősége annak, hogy az A $\rightarrow$ B, B $\rightarrow$ C és A $\rightarrow$ C függéssorozatban valamelyik tétel „nem-kulcs", azaz pontosabban nem kulcsrész?

Erre a kérdésre a szakírók implicit „igennel" válaszolnak. Ha a tényező kulcsrész/nem-kulcsrész szerepe közömbös lenne, akkor semmi értelme sem lenne megkülönböztetni a részleges és a tranzitív függést. Azért nem, mert tulajdonképpen a részleges függés is tranzitív, csak éppen a függéssor egyik tagja kulcsrész. Hasonlítsuk csak össze a 16.1 példa két függéssorát:

16.1 példa

RENDELÉS			RENDELÉSTÉTEL		
Rendelésszám	$\rightarrow$	Vevőkód	Rendelésszám+Cikkszám	$\rightarrow$	Cikkszám
Vevőkód	$\rightarrow$	Vevőnév	Cikkszám	$\rightarrow$	Cikknév
Rendelésszám	$\rightarrow$	Vevőnév	Rendelésszám+Cikkszám	$\rightarrow$	Cikknév

A RENDELÉS esetében a harmadik függést tranzitívnak, a RENDELÉSTÉTEL példájában viszont részlegesnek tekintjük, holott a „képlet" - A $\rightarrow$ B, B $\rightarrow$ C és A $\rightarrow$ C - tökéletesen ugyanaz. Ennek a különbségtételnek valószínűleg az az oka, hogy a szakirodalom elfeledkezik a RENDELÉSTÉTEL első függéséről, amely projektív jellegű és így triviális. Armstrong [20] ***projektivitási*** axiómája szerint egy összetett tulajdonság (Rendelésszám+Cikkszám) mindig meghatározza a saját összetevőit (Cikkszám).

Mármost ha a Cikkszám kulcsrész-jellege közömbös, akkor egyáltalán nincs különbség a két példa - és így a részleges illetve tranzitív függés, a 2NF illetve a 3NF - között. Ha viszont nem az, akkor minden függéskombinációban figyelnünk kellene a kulcs/nem-kulcs jellegre. Szerintünk nem mellékes a tulajdonság szerepe Ezért összeállítjuk a következő döntési táblázatot, illetve egyelőre annak csak a feltételtömbjét. Lásd 16.1 ábra.

			1	2	3	4	5	6	7	8
A	$\rightarrow$	B	K	K	K	K	N	N	N	N
B	$\rightarrow$	C	K	K	N	N	K	K	N	N
A	$\rightarrow$	C	K	N	K	N	K	N	K	N

16.1 ábra: A függések döntési táblázata (1)

A táblában három tulajdonság függéseit mutatjuk a feltételtörzsben (baloldalt). K bejegyzés jelzi a feltételjegyzékben (jobboldalt), hogy a meghatározó saját kulcsrészeként határozza-e meg a függőt (K bejegyzés), vagy sem (N bejegyzés). Az egyes oszlopokat szabályoknak nevezzük. Az 1. szabály első bejegyzése K. Ezek szerint a B-t az A összetett tulajdonság részének feltételezzük. Az 5. szabály első bejegyzése N. Ebben az esetben a B-ről nem feltételezzük, hogy az A része.

A döntési táblázatoknak van tevékenység vagy következmény tömbje is. Ez most hiányzik az ábránkból. A tevékenységtömb azt árulja el, hogy mi a helyzet akkor, ha az adott szabályban lévő feltételek mindegyike fennáll. Mi a teendő/következmény például akkor, ha mindhárom függés projektív (1. szabály), vagyis ha az A a saját maga részeként határozza meg a B-t (1. sor) és a C-t (3. sor) illetve a C a B-nek is része (2. sor)?

A következő pontban az egyszerűen feltárható döntési eseteket fogjuk vizsgálni.

16.3 Két egyszerű és három ismert függési helyzet

A táblázat ***első*** szabálya az abszolút ***triviális*** függést mutatja. Az A tulajdonság összetett úgy, hogy a B is és a C is a része. A B tulajdonság összetett úgy, hogy része a C. Ezt akkor tudjuk igazán megérteni, ha új jelöléseket vezetünk be. A továbbiakban a tulajdonságok összetételét illetve azonosságát a {} jelek között mutatjuk. Tehát a következő jelölésekkel fogunk élni:

A {X+Y+Z}
B {X+Y}
C {X}

Az első sor azt mondja, hogy az A az X, Y és Z tulajdonságokból összetett. A harmadik sor azt mutatja, hogy C azonos X-szel. Most már egészen világosan látható, hogy az A-nak része a B és a C illetve a C-t a B is tartalmazza. Három függésünk van:

A {X+Y+Z} $\rightarrow$ B {X+Y}
B {X+Y} $\rightarrow$ C {X}
A {X+Y+Z} $\rightarrow$ C {X}

Annak ellenére, hogy tranzitív függést látunk, nincs lehetőségünk a normalizálásra, vagyis a tranzitívnak tűnő harmadik függés megszüntetésére, mert az A {X+Y+Z} $\rightarrow$ C {X} függés axiomatikus. Valójában egyik függést sem tudjuk kiküszöbölni, mert mindegyik projektív. Ezért akkor, ha létezik például a TELEPÜLÉS (***Országkód+Megyekód+Településkód***) egyed, akkor azt nem lehet megbontani azon az alapon, hogy a teljes kulcstól az Országkód és Megyekód párosán át függ a végső tag, az Országkód is. A TELEPÜLÉS egyedben lévő függéskombináció ugyanis teljesen triviális.

A táblázat ***második*** szabálya egyszerűen ***ellentmondásos***. Elvileg nem fordulhat elő ez a helyzet. Ha az A {X+Y+Z}-nek része a B {X+Y}, annak pedig a C {X}, akkor kizárt, hogy az X ne legyen az X+Y+Z alkotóeleme is. Előbbi példánkkal élve: Ha egy település adott ország adott megyéjében van (A), a megyén belül egyértelműen meghatározott (B), akkor képtelenség, hogy a település ne kapcsolódjon magához az országhoz is (C).

A táblázat ***negyedik*** szabálya mutatja az általunk már ismert ***részleges függés*** képletét:

A {X+Y} $\rightarrow$ B {X}
B {X} $\rightarrow$ C {Z}
A {X+Y} $\rightarrow$ C {Z}

A 14.2 ábra RENDELÉSTÉTEL egyedében a Rendelésszám és Cikkszám párosa saját részeként határozza meg a Cikkszám-ot. Ettől és magától a teljes párostól nem-projektív módon függ a Cikknév tulajdonság. Ilyen esetekben a harmadik Rendelésszám+Cikkszám → Cikknév függést kell megszüntetni, amely formailag tranzitív. Vagyis a Cikknév tulajdonságot ki kell vennünk a RENDELÉSTÉTEL egyedből. Ezt megtehetjük, mert a harmadik függés nem projektív.

A táblázat ***nyolcadik*** szabálya a közönséges ***tranzitív függés*** képlete. Egyik függésben sem a saját részét határozza meg a tulajdonság, vagyis egyik függés sem projektív. Lásd az 14.8 ábra RENDELÉS egyedét. Ekkor is a harmadik Vevőkód → Vevőnév függés kiküszöbölése a cél. A Vevőnév tulajdonságot ki kell venni a RENDELÉS egyedből. Ennek nincs akadálya, mert ez a függés sem projektív.

Itt álljunk meg egy pillanatra. Vegyük észre, hogy nem egyszerűen a tulajdonságok összetett vagy elemi voltáról van szó. A táblázat nyolcadik oszlopának az esetében bármelyik (A, B, C) tulajdonság lehet akárhányszorosan is összetett. A lényeg az, hogy egyik összetétel sem lehet a másik teljes része. Ezért van három N bejegyzés az oszlopban.

A táblázat ***hetedik*** szabálya fedi le a ***kulcstörő függés*** esetét, amelynek képlete a következő:

A {X+Y} → B {Z}
B {Z} → C {X}
A {X+Y} → C {X}

Emlékezzünk csak az UTCA egyedre (ld. 15.5 példa). A Kerületkód+Utcaszám meghatározza az Irányítószámot, nem a saját maga részeként. Az Irányítószámtól függ a Kerületkód, ismét csak normális függéssel. Viszont a Kerületkód+Utcaszám a saját részeként határozza meg a Kerületkód-ot.

Most találkozunk az első dilemmával. Eddig az hihettük, hogy a formai tranzitívitás miatt az A → B, B → C és A → C függések közül a harmadikkal akad teendőnk. Táblázatunk első oszlopa esetében az volt triviális, tehát nem megszüntethető. A második szabálynál az volt ellentmondásos. A negyedik és a nyolcadik szabály esetében azt kellett és lehetett kiemelni.

A hetedik szabály esetében az A {X+Y} → C {X} függés projektív, ezért nem számolható fel. A másik két függés valamelyike pedig azért nem hagyható el, mert ismeretet veszítenénk. Hiszen csak az első két függésből következik automatikusan a harmadik, az elsőből és a harmadikból nem vezethető le a második, a másodikból és harmadikból pedig nem adódik az első.

Ezért a kulcstörő függés esetében nem a tranzitívitás szerinti megoldást alkalmaztuk. Vagy teljesen új azonosítót vezettünk be, vagy más azonosítópárt választottunk és ezzel alakítottuk át a függéseket például az alábbi módon:

A {Z+Q} → B {Z}
B {Z} → C {X}

A kulcstörő függés esetében a normalizálás már csak félig mechanikus. A táblázat 7. szabálya által jelzett függéssor nem eredményezi automatikusan a helyes megoldást (a harmadik függés megszüntetését), hanem csak felhívja a figyelmet a problémára. Ezzel kezdődik a ***szemantikus*** normalizálás, amelyben a tervező a jelenségek természetes összefüggéseit mérlegelve alakítja át az adatbázis-szerkezetet, amelynek a hibáját a mechanikus normalizálás tárta fel.

A következő pontokban olyan összetett normalizálási problémákra hívjuk fel a figyelmet, amelyek csak szemantikai alapon oldhatók meg. Mielőtt a nehezebb feladatokra rátérnénk, aktualizáljuk a 16.1 ábra táblázatát az eddig megismert következményekkel. Lásd a 16.2 ábrát.

			1	2	3	4	5	6	7	8
A	→	B	K	K	K	K	N	N	N	N
B	→	C	K	K	N	N	K	K	N	N
A	→	C	K	N	K	N	K	N	K	N
Triviális függés			X							
Lehetetlen eset				X						
Részleges függés						X				
Tranzitív függés										X
Külső kulcstörés									X	

16.2 ábra: A függések döntési táblázata (2)

Az 1. szabály esetében nincs teendő, mert a szerkezet jó. A 2. szabálynál azért nincs, mert ez a helyzet nem fordulhat elő. A 4. oszlopnál a részleges függést meg kell szüntetni és akkor legalább 2NF alakot nyerünk. A 8. szabály feltételkombinációjának a fellépésekor ismét normalizálnunk kell. A tranzitív függés eltávolításával legalább 3NF alakú lesz az egyedtípusunk. Végül a 7. oszlop szabálya által rejtett ellentmondás úgy szüntethető meg, hogy új kulcsot választunk és BCNF alakú egyedeket tervezünk.

A táblázat fennmaradó három szabályára nem találunk magyarázatot a normalizálással foglalkozó mindennapos szakirodalomban. A következő három pontban ezeket az eseteket fogjuk megvilágítani.

16.4 A belső kulcstörő függés

Igazából csak most kezdünk elérni a szemantikus normalizálás lényegéhez. Az első probléma szemléltetéséhez a 16.2 példát fogjuk használni:

16.2 példa

RENDELÉS-1 (***Rendelésszám***, Cikkszám, ...)
RENDELÉS-2 (***Rendelésszám+Cikkszám***, ...)

Egyelőre koncentráljunk a második egyedtípusra feltételezve azt, hogy fennáll abban is az első egyed által sejtetett Rendelésszám → Cikkszám függés. A 16.2 ábra táblázatának a ***harmadik*** szabálya mutatja ezt a szituációt. Ennek a képlete a következő:

A {X+Y} → B {X}
B {X} → C {Y}
A {X+Y} → C {Y}

Létezik egy A összetett kulcs, amely projektíven meghatározza a saját első X részét. Az X kulcsrész meghatározza az azonosító másik Y tagját, amely ismét csak projektíven függ az azonosító egészétől. Tehát a Rendelésszám+Cikkszám → Rendelésszám, Rendelésszám → Cikkszám és Rendelésszám+Cikkszám → Cikkszám függés-hármassal állunk szemben.

A projektív jelleg miatt nincs módunk arra, hogy megszüntessük az első vagy a harmadik függést. A tervező a kedvezőtlen konstelláció miatt úgy dönthet, hogy felszámolja a második függést. Bár döntését még elméletileg is meg tudja alapozni, lehet, hogy hibát követ el.

Az elméletileg jártas tervező tudja, hogy az összetett azonosító részei között nem létezhet függési viszony. Két tulajdonság csak akkor kapcsolható egy azonosítóba, ha értékeik viszonya hálós (M:N fokú). Ha a tulajdonságok lineáris (1:1 fokú) viszonyban vannak, akkor alternáló kulcsokról van szó. Ha összefüggésük hierarchikus (1:N fokú), akkor az egyik tag meghatározza a másikat és ez a másik nem lehet azonosítórész. Ha tehát a kulcs egyik tagja (Rendelésszám) funkcionálisan meghatározza a másikat (Cikkszám), akkor az első tag az utóbbi leírónak az azonosítója és így vele nem kapcsolható egy kulcsba.

Ezért a tervezőnek formailag igaza van akkor, amikor a RENDELÉS-2 egyed azonosítóját át akarja alakítani. Mielőtt ezt megtenné, olvassa el a következő történetet.

A 16.2 példát már 1984-ben (!) feladtuk hat tapasztalt adatmodellezőnek, akiket ráadásul normalizálást végző szoftver is támogatott. Mivel a tervezők és az automata csak a BCNF alakig készültek fel a normalizálásra, a példa dilemmáját nem tudták megoldani. Pedig a megoldás eléggé kézenfekvő.

Az adatmodellezés egyik alapvető titka, hogy a függéseket sohasem szabad egymástól elkülönülten, mindig csak egy-egy egyed vonatkozásában vizsgálni. Kétségtelen, hogy eddig mi a ***dekompozíciós*** normalizálási eljárást ismertük meg, amelynek tárgya mindig egyetlen egyed. Csakhogy éppen a 16.2 példa mutatja, hogy bizonyos helyzetekben nem a dekompozíció a helyes megoldás, és legszebb ideje áttérni a ***szemantikus*** normalizálásra.

A szemantikus normalizálás során a függéseket együtt, egyszerre több egyedben vizsgáljuk. Példánk esetében úgy találjuk, hogy a RENDELÉS-1 egyedben a Rendelésszám meghatározza a Cikkszám-ot és ezért viszonyuk hierarchikus. A RENDELÉS-2 egyedben viszont a két tétel az összetett kulcs része, azaz kapcsolatuk hálós. Tehát ellentmondásra bukkantunk. Következik-e ebből a visszásságból az, hogy tényleg valamelyik függési viszony rossz és ezért az egyik egyedet fel kell számolni? És ha igen, akkor melyiket? Mi az igazság: az, hogy a Rendelésszám meghatározza a Cikkszámot, vagy az, hogy a két tétel egymástól független?

Lám, éppen erre az utóbbi kérdésekre nem tudtak választ adni a fentebb említett tervezők. Mivel dekompozíciós eljárással dolgoztak, a két rendelés egyed pedig külön-külön (!) tökéletes normálalakban van, nem jöttek rá a turpisságra. Az automatájuk sem tudta azt felfedezni. Igaz, az automata nem is dönthetne az ember helyett arról, hogy fel kell-e számolni valamelyik egyedtípus és ha igen, akkor melyiket.

A példa titka többszörös. Az egyik titok az, hogy a függési viszonyokat nem egyedenként, hanem együttesen kell szemlélni. Így egyidejűleg megadhatjuk azt is, hogy a Rendelésszámtól függ is a Cikkszám, meg nem is. Ettől a kettős meghatározástól persze a legtöbb mai normalizáló automata „kiakad”. A másik titok az, hogy a jó normalizáló megengedi ezt a kettős függést, mert nem tudhatja egy automata, hogy a kettő közül melyik a rossz. Azt viszont, hogy a kettő együtt helytelen, a döntési táblázat harmadik oszlopa szerint ki tudja mutatni. A harmadik titok az, hogy nem a dekompozíció értelmében normalizálva, hanem szemantikus normalizálással kell megoldani a problémát.

A megfejtés pedig a következő: Vannak vevői rendeléseink, amelyek egytételesek és ezért a Rendelésszám → Cikkszám függés érvényesül bennük (RENDELÉS-1). Viszont léteznek szállítói rendeléseink is, amelyek többtételesek. A tételeket a Rendelésszám és a Cikkszám párosa azonosítja és így a két tétel egymástól független (RENDELÉS-2). Tehát az adott példa esetében nem normalizálásra, hanem a fogalmak szemantikai tisztázására lett volna szükség. A 16.2 példa letisztázott változata világosan mutatja a helyes megoldást:

16.3 példa

RENDELÉS-1 (***Vevő-rendelésszám***, Cikkszám, ...)
RENDELÉS-2 (***Szállító-rendelésszám+Cikkszám***, ...)

A kérdés most már csak az, hogy miért lett volna baj az, ha a 16.2 példát egyáltalán nem alakítjuk át? Nos, az adatbázis egyik ***integritási szabályáról*** van szó. A 16.2 példa azt sejteti, hogy a két egyedtípus a közös Rendelésszám tulajdonságon keresztül kapcsolatban áll. A tétel az egyik egyedben kulcs, a másikban kulcsrész - tehát kapcsoló szerepű. Ezért az integritási szabály alapján minden modellező és adatbáziskezelő rendszer kapcsolatot tételezne fel a két egyedtípus között. Abból pedig szép kalamajka származna, ha a vevői- és a szállítói rendeléseket összekevernénk egymással. Ettől ment meg bennünket a táblázat 3. szabálya.

Már csak annyi van hátra, hogy a gyakorlati problémát és annak feloldását elméletileg is megalapozzuk.

D 16/1 **Az E egyed belső kulcstörő függést tartalmaz akkor és csak akkor, ha az A+B azonosító összetett és annak egyik része meghatározza a másik összetevőt.**

Emlékezzünk vissza a külső kulcstörő függés esetére (ld. 15.2.2 alpont). Ott az azonosító egyik részét hozzá képest külső (leíró szerepű) tulajdonság határozta meg. Itt pedig arról van szó, hogy a törést az azonosítón belüli (kulcsrész szerepű) tulajdonság okozza. A két probléma feloldása teljesen eltérő jellegű. A külső kulcstörés esetében mechanikusan is eljárhattunk úgy, hogy a BCNF normálformát alkalmaztuk. Itt viszont csak a szemantikus megoldás segít. A normálforma-hiba rejtett homonimához (Rendelésszám) vezetett bennünket. Ilyenkor nem lehet és nem is szabad a dekompozíció módszeréhez nyúlni. Elegendő, ha a fogalmakat tisztázzuk (Vevő- és Szállító-rendelésszám) és azokat alkalmazzuk a szerkezetben.

Amint látjuk, a mechanikus normalizálással feltárt mennyiségi probléma (belső kulcstörő függés) irányította a figyelmünket a valós minőségi bajra (homonima), amit szemantikai módon küszöböltünk ki. Nagy tanulság ez az adatbázistervezőknek. Sohasem szabad csak a számokban bízni! Eláruljuk, hogy mi még a triviális függés (ld. 16.3 pont) esetében is körbejárjuk, hogy az A {X+Y+Z}, B {X+Y} és a C {X} tényezők esetében „ugyanarról” az X-ről és X+Y-ról van-e szó.

A 16.2 példa két egyede elvileg 5NF alakban volt, hacsak azt nem feltételezzük, hogy a második egyed egyáltalán nem alkalmazható a belső kulcstörő függés miatt. A két tétel együttese mégsem alkotott jó adatmodellt. A 16.3 adatmodell tökéletesebb. Mivel azonban nem dekompozícióval oldottuk fel a problémát, nem tudjuk megmondani, hogy a korábbihoz képest „hányadik” normálformában van a két egyed a normalizálás után.

16.5 A metszetfüggés

Egy meglehetősen összetett problémát kell megvilágítanunk ebben a pontban. A 16.4 példa szemlélteti újabb esetünket:

16.4 példa

RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, ..., Mennyiség)
DISZPOZICIÓTÉTEL (***Diszpozíciószám+Cikkszám***, *Rendelésszám*, ..., Mennyiség)

A példa magyarázata a következő: Rendelések érkeznek hozzánk. Ezek többtételesek. Egy-egy rendeléstételt több részletben is szállíthatunk. Mondjuk 80 valamit kértek tőlünk és mi három (például: 40, 25, 15) adagban teljesítünk. Ezért egy rendeléstételhez több diszpoziciótétel is tartozhat. Természetesen a diszpozíciónak mindig arra a rendelésre kell utalnia, amire maga a rendeléstétel is hivatkozik. Ezért szerepel a DISZPOZICIÓTÉTEL-ben a Rendelésszám. Ezzel szemben egy diszpozíció - nem tétel, hanem maga a teljes szállítás - több rendelés tételeit is tartalmazhatja. Így nem áll fenn a Diszpoziciószám → Rendelésszám függés és a fordítottja sem.

Ezek után a gyanútlan tervező azt hiszi, hogy a 16.4 példával nincsen semmi baj. Hiszen a két egyedben nincsenek kedvezőtlen függések; mindkettő 5NF alakú. A valóságban fellép egy eléggé kellemetlen probléma. A 16.4 példa terve hiányos. Mielőtt ezt kimutatnánk, el kell elmélkednünk a relációs elmélet és normalizálás féloldalasságáról.

A relációs modell elismeri az ***összetett azonosító*** tételt, de nem engedi meg az ***összetett leíró*** tulajdonságot. Ezért egy függés „baloldalára" írható A+B összetétel, de „jobboldalára" nem. Nem fejezhető ki ez a függés: A+B → C+D. Az elméletet ismerők most felzúdulnak kijelentve, hogy ez a megállapítás téves. Hiszen Armstrong additivitási axiómája mondja ki, hogy az A → B és A → C függések esetén mindig igaz az A → B, C függés. Ez így igaz. Ezért is nem használtuk az A+B → C, D formulát, hanem tudatosan a „+" jelet alkalmaztuk.

A modellezés egyik titka, hogy az „A meg B" nem azonos az „A és B"-vel. Egy személyi-szám „meg" egy személyi-szám az két személyi-szám. Viszont egy személyi-szám „és" egy másik személyi-szám az például - házasság. Vagyis a mennyiség mögött minőség húzódik meg.

Ez a helyzet példánk esetében is. A 16.4 példa egyáltalán nem tükrözi azt a valós tényt, hogy a diszpoziciótételeknek adott rendeléstételekhez kell kapcsolódniuk. A DISZPOZICIÓTÉTEL ebben a formájában köthető a DISZPOZICIÓ-hoz (Diszpoziciószám), a CIKK-hez (Cikkszám) és a RENDELÉS-hez (Rendelésszám). Viszont nem kapcsolható a RENDELÉSTÉTEL egyedhez. Azért nem, mert bárki bevihet olyan sort a DISZPOZICIÓ-TÉTEL-be, amelyben külön-külön érvényes Rendelésszám és Cikkszám van, csak éppen abban a rendelésben sohasem kérték a vonatkozó tételt, vagyis nem létezik a két tétel párosa. Magyarul és tömörebben: a két egyedtípus kapcsolata nincs megalapozva kapcsolati integritási korláttal.

A javított szerkezetet a 16.5 példa mutatja:

16.5 példa

RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, ..., Mennyiség)
DISZPOZICIÓTÉTEL (***Diszpoziciószám+Cikkszám***, *Rendelésszám+Cikkszám*, ..., Mennyiség)

Most már világosan látszik, hogy a 16.4 példa szerkezete nem jó. A második egyed azonosítója meghatározza az első kulcsát és mindkét azonosítótól függ a Mennyiség. Döntési táblázatunk 8. szabálya érvényesül: a Mennyiség a modell egészét tekintve tranzitív a második egyedben. Tehát meg kell szüntetni? Az roppant nagy hiba lenne, mert ismeretet veszítenénk, ugyanis a két mennyiség mást jelent. Nem az adatot, hanem a ***homonim*** nevet kell felszámolni. Például úgy, hogy a Rendelt-mennyiség és Diszponált-mennyiség beszélő neveket alkalmazzuk.

A példából eddig két tanulságot szűrhettünk le. Az egyik az, hogy a mechanikus alapon feltárt tranzitivitást nem mindig dekompozícióval, hanem sokszor szemantikus módon kell megoldani. A másik az, hogy az összetett leírók - magyarul: ***csoportok*** - létét kizáró relációs elmélet és az azon alapuló normalizálási eljárások elvileg nem alkalmasak a tökéletes adatszerkezet megtalálására. Ugyanis az ortodox relációs modellhez ragaszkodva sohasem írhattuk volna a DISZPOZICIÓTÉTEL-be a *Rendelésszám+Cikkszám* összetett tulajdonságot. Következésképpen a tranzitivitáson át nem fedeztük volna fel az egyértelműséget zavaró homonimát.

Már a 12.4.4 alpontban rámutattunk arra, hogy a modellben egyértelmű neveket kell használni. Nem lehet a homonima-problémát azzal elkenni, hogy „úgyis tudjuk, hogy a két egyed-

típusban mást jelent a Mennyiség". Mi tudjuk. A felhasználó pedig a bemenetben a központ telephelyét adja meg a kocsi tényleges tárolási helye tartalmaként, mert hiszen az adat neve az volt, hogy Telephely. Azután meg rengeteg kimenetünk lesz, amelyen mindkét mennyiség megjelenik. Akkor úgyis meg kell különböztetnünk őket. Miért nem tudunk akkor mindjárt tisztességesen „beszélő" neveket használni?

Persze ez csak kitérő volt. Mert az igazi probléma az, hogy miként adható meg két Cikkszám nevű tulajdonság a DISZPOZICIÓTÉTEL egyedben? A válasz kézenfekvő: sehogy. A valódi megoldást a 16.6 példa mutatja:

16.6 példa

RENDELÉSTÉTEL (***Rtétel-azonosító***, ..., R-mennyiség)
DISZPOZICIÓTÉTEL (***Dtétel-azonosító,*** *Rtétel-azonosító,* ..., D-mennyiség)

Rtétel-azonosító {Rendelésszám+Cikkszám}
Dtétel-azonosító {Diszpozició+Cikkszám}

Ez a megoldás tökéletes. Felszámoltuk a tranzitivitás által jelzett homonimát. Gondoskodtunk a kapcsolati integritásról, mert most már mindenki láthatja, hogy a diszpoziciótételeknek a rendeléstételekhez kell kapcsolódniuk az Rtétel-azonosító kapcsoló tulajdonságon keresztül. Az pedig senkit se aggasszon, hogy egyes mai kezelőkben a példa alján lévő csoportokat nem lehet meghatározni. Egyrészt a relációs kezelők legközelebbi generációja már biztosan tartalmazni fogja ezt a képességet. Másrészt az adatmodellezés nem azonos az adatkezeléssel. Ha a kezelő nem is kezel csoportot, akkor is az elemzésben a fent leírt módon kell összeállítani az adatbázis tervét. Mert ha a kezelő nem biztosítja automatikusan (definitíven) a két egyed kapcsolódását, akkor azt nekünk kell megtennünk programmal (procedurálisan).

Már csak annyi van hátra, hogy megvizsgáljuk a példa mögötti elméletet.

D 16/2 **Az E egyed metszetfüggést tartalmaz akkor és csak akkor, ha az A+B összetett azonosító olyan csoportot határoz meg, amely tartalmazza az A vagy a B kulcsrészt.**

Mivel a meghatározó tulajdonságok átfednek, a metszet függés elnevezés találó. Ugyanis az ilyen függéskombináció képlete a következő:

A {X+Y} $\rightarrow$ B {Y+Z}
B {Y+Z} $\rightarrow$ C {Y}
A {X+Y} $\rightarrow$ C {Y}

Döntési táblázatunk ötödik szabályáról van szó. Az első függés nem-projektív, a másik kettő az. Nincs mód dekompozícióra, mert az első függés megszüntetésével ismeretet veszítenénk (nem kapcsolhatnánk a két egyedtípust egymáshoz). Legfeljebb azon érdemes elgondolkoznia a tervezőnek, hogy az Y+Z páros valóban ugyanazt jelenti-e a két egyedben. Mert ha nem, akkor homonimáról van szó és szemantikai megoldást kell keresni. Ezért annyira lényeges az, hogy a mechanikus normalizálás során felfedezzük a metszetfüggéseket.

16.6 A csoportfüggés

Ha az elmélet megold egy problémát, azonnal szembe kell néznie a következővel. Így jártunk mi is, amikor az előző pontban bevezettük a csoport tényezőjét. A 16.7 példa mutatja az ennek következtében fellépő egyik problémát:

16.7 példa

DISZPOZICIÓ (***Diszpoziciószám***, *Rtétel-azonosító*, ..., Cikkszám)

Rtétel-azonosító {Rendelésszám+Cikkszám}

Most azt feltételezzük, hogy minden rendeléstételt külön diszponálnak. Ezért a Diszpozició-szám folyamatos sorszám, vagyis nem csoport. A 16.7 példa megoldása mégsem jó, mert rejtett redundanciát tartalmaz. Ugyanis kétszer szerepel benne a Cikkszám. Egyszer expliciten, egyszer pedig impliciten az Rtétel-azonosító részeként. Mielőtt kifejtenénk azt, hogy ez miért jelent problémát, vissza kell térnünk a 16.6 példára.

Abban az esetben a Cikkszám két csoport implicit tagja volt és expliciten nem jelent meg az egyedben. Mármost azt tudni kell, hogy a ***csoport*** virtuális szerkezet. A 16.6 példa megoldása nem jelenti azt, hogy a Cikkszám kétszer kerül tárolásra. Mivel a Dtétel-azonosító és az Rtétel-azonosító csoportokban lévő Cikkszámnak ugyanaz a tartalma, világos, hogy a tétel egyszeresen kerül tárolásra és a két csoportszerkezet csak integritási korlátként szolgál. (Az egyik azonosít, a másik kapcsol a RENDELÉSTÉTEL felé.)

A 16.7 példában teljesen más helyzettel állunk szemben. Mivel a Cikkszám explicit leíró tulajdonság, az mindenképpen tárolásra kerül. Ha ez a Cikkszám azonos tartalmú, mint az Rtétel-azonosító csoportban lévő azonos nevű tétel, akkor az egyed redundáns. A ***logikai átfedés*** sajátos esetéről van szó: egy egyed - rejtetten - kétszeresen tartalmazza ugyanazt a tulajdonságot. Ilyenkor a megoldás is kézenfekvő: az explicit Cikkszámot el kell távolítani az egyedből.

Azonban a másik eset sem kizárt. Nevezetesen az, hogy az explicit Cikkszám nem ugyanazt jelenti, mint az Rtétel-azonosítóban lévő implicit Cikkszám. Például gyakran előfordul, hogy nem tudjuk a kért cikket szállítani, de a vevő tudtával valamilyen helyettesítő árut biztosítunk a megrendelő részére. Ebben az esetben világos, hogy a kétféle tétel ***homonima***. A kétértelműséget pedig mindig úgy oldjuk fel, hogy a valós nevükön nevezzük a dolgokat. Lásd a 16.8 példát.

16.8 példa

DISZPOZICIÓ (***Diszpoziciószám***, *Rtétel-azonosító*, ..., Helyettesített-cikkszám)

Rtétel-azonosító {Rendelésszám+Cikkszám}

Ha egy egyed expliciten és impliciten is tartalmazza ugyanazt a tulajdonságot, akkor vagy a valós, vagy a látszólagos logikai átfedés hibájában szenved. Ennek a problémának a megoldásában segít bennünket az elmélet.

D 16/3 **Az E egyed csoportfüggést tartalmaz akkor és csak akkor, ha az A+B csoport mellett expliciten tartalmazza annak A vagy B részét is.**

A csoportfüggés általános képlete a következő:

A {X} $\rightarrow$ B {Y+Z}
B {Y+Z} $\rightarrow$ C {Y}
A {X} $\rightarrow$ C {Y}

Döntési táblázatunk 6. szabálya világítja meg ezt az esetet. A második függés projektív, a harmadik nem az. Ezért az utóbbit minden további nélkül meg lehet szüntetni. Így nem is csoda, hogy ezt az esetet sokan összetévesztik az egyszerű tranzitivitással. Azonban, mint rámutattunk, általában ez a helyzet nem logikai átfedést, hanem homonimát takar. A helyettesítő tétel megjelölésének az elhagyásával ismeretet veszítenénk. Viszont két eltérő tartalmú tulajdonságnak nem lehet azonos a neve. Erre a problémára irányítja a figyelmet a csoportfüggés, amelynek alapján megtaláljuk a helyes megoldást. Az pedig nem más, mint a szemantikai normalizálás, vagyis a homonima felszámolása.

16.7 Tanulságok és a döntési táblázat kiegészítése

A csoportok nélkülözhetetlen adatmodellezési konstrukciók. Ha egy egyedtípus egy másiknak alárendeltje, akkor idegen kulcsként tartalmaznia kell a fölérendelt azonosítóját. Ha az utóbbi összetett, akkor csoportként kell feltüntetni a kapcsoló adatot. Nem elegendő az, hogy csak a csoport részeit adjuk meg, mert az „A meg a B” nem azonos az „A és a B”-vel.

Persze a csoportok nem-közönséges modellezési problémákat vetnek fel. Mert ameddig a csoport csak két tagú (pl. A {X+Y}), addig még talán át tudjuk tekinteni annak függéseit. Ha viszont többtagú, akkor bizony már nehéz az értékelés.

A gyakorlatban sokszor találkozunk implicit csoportokkal. Például egy papíron feltüntetik a Számlaszámot is és a Hivatkozási-számot is, de ezeket nem részletezik. A relációs rendszerekben az ilyen implicit csoportokat nem bontják meg, mert az kezelési nehézségekre vezetne. Viszont mi van akkor, ha a két tulajdonság közös tényezőt tartalmaz? Például mindkettőben szerepel a Vevőkód? Ellenőrzi-e valaki, hogy a két Vevőkód értéke azonos? Észreveszik-e a metszetfüggés esetét?

Az A {X+Y+Z+W} és B {X+Q+Z+V} jellegű csoportok átfedéseit és esetleges helytelen függéseit pusztán emberi úton nagyon nehéz felfedezni. Az ilyen jellegű helyzetek értékeléséhez már elengedhetetlenül szükségesek a normalizáló automaták.

A tanulságok folytatása előtt egészítsük ki döntési táblázatunkat. Lásd a 16.3 ábrát.

	1	2	3	4	5	6	7	8
A $\rightarrow$ B	K	K	K	K	N	N	N	N
B $\rightarrow$ C	K	K	N	N	K	K	N	N
A $\rightarrow$ C	K	N	K	N	K	N	K	N
Triviális függés	X							
Lehetetlen eset		X						
Részleges függés				X				
Tranzitív függés								X
Külső kulcstörés							X	
Belső kulcstörés			X					
Metszetfüggés					X			
Csoportfüggés						X		

16.3 ábra: A függések döntési táblázata (3)

Hosszú ideig a szakirodalom a normálformákat és a normalizálást a relációs adatbázisok ***logikai szintű*** tervezési segédletének tekintette. Mivel a relációs modell nem ismeri a csoport konstrukciót, a hagyományos normalizálás táblázatunknak csak azon oszlopaival foglalkozik, amelyek a B → C függés sorában „N" bejegyzést tartalmaznak. E négy szabály közül sem figyel a belső kulcstörés esetére, mert egy egyedtípuson belül az nem fordulhat elő.

Később kezdték felismerni, hogy a normalizálás valójában ***fogalmi szintű*** módszer. Az előző fejezetben láthattuk, hogy a BCNF alakot nem mindig dekompozícióval kapjuk meg. Vannak helyzetek, amikor az azonosító megváltoztatásával érünk el jobb modellt. Ebben a fejezetben rámutattunk arra, hogy a kedvezőtlen függések felszámolása igen sokszor csak szemantikai módon történhet. A fogalmak pontos jelentését kell feltárni és a korrekt megnevezések alkalmazásával egyértelművé tenni a modellt.

16.8 Pszeudo-tranzitivitás

A tulajdonságok helytelen függési viszonyainak a feltárása nem mindig olyan egyszerű, amint azt az előző két fejezet sejtette. A fenti pontok meggyőzhettek bennünket arról, hogy a csoportok sok galibát okoznak. A csoportok elrejtik az azokat alkotó tényezők valós függési viszonyait és így nem tárul fel előttünk a modell minden eldugott hibája. A 16.9 példa és a 16.4 ábra a csoportok egy újabb titkát rejtegeti.

16.9 példa

RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, ..., R-mennyiség)
DISZPOZICIÓTÉTEL (***Diszpozíciószám+Cikkszám***, D-mennyiség, ..., R-mennyiség)

RENDELÉSTÉTEL

Rendelésszám	***Cikkszám***	...	R-mennyiség
12345	***234568***		140
12345	***234567***		120

DISZPOZICIÓTÉTEL

Diszpozíciószám	***Cikkszám***	...	D-mennyiség	R-mennyiség
A11	***234567***		60	120
A12	***234567***		20	120
A13	***234567***		40	120
A21	***234568***		40	140

16.4 ábra: DISZPOZICIÓ adatbázis-részlet

Ebben a példában azt feltételezzük, hogy egy diszpozició mindig csak egy rendelésre vonatkozik. Ezért - szemben a 16.4 példával - a Rendelésszám nem lehet a második egyedtípus tulajdonsága, mert részlegesen függene az összetett kulcs első részétől. Ezért a diszpoziciótétel nem kapcsolódik közvetlenül a rendeléstételhez. Az olvasó mégis gyanút fog, hogy ezzel a tervvel nincs valami rendjén, mert kétszer szerepel benne az R-mennyiség leíró tulajdonság. A nyílt logikai átfedés (lásd 12.4.1 alpont) esetével állunk szemben, ha azt feltételezzük, hogy a tétel mindkét egyedben ugyanazt jelenti.

Éljünk ezzel a feltételezéssel. A táblákban az R-mennyiség a rendelt, a D-mennyiség a diszponált mennyiséget jelenti. A raktári lapokon gyakran találkozunk a 16.4 ábrának megfelelő megoldással. Az ügyintézők saját maguknak vezetik, hogy „ennyit rendeltek és abból ennyit adtam ki". A második egyed első sora szerint a „120" rendelt mennyiségből „60" egységet, a második sor szerint „20" egységet stb. adtak ki.

Ez természetesen redundáns ismeret. Ebben a konkrét esetben bárki könnyen felismerheti a kettős redundanciát. Egyrészt a két egyed nyílt logikai átfedésben van egymással. Másrészt a második egyedben feltűnő a fizikai redundancia. Vannak azonban olyan helyzetek is, amelyekben a helytelen struktúra nem látható át ilyen könnyen. Ezért a kérdés az, hogy általában milyen alapon ismerhetők fel az ilyen rossz szerkezetek.

Sajnos, az eddigi normalizálási eljárásokkal hiába próbálkozunk. Döntési táblázatunk egyik esete sem érvényes erre a szituációra. Ennek az az oka, hogy a példa két egyede egymástól független: tulajdonságaik között nincs közvetlen összefüggés. Ezért nem találunk olyan függéshármast, amit táblázatunk valamelyik szabályának meg tudnánk feleltetni. Mi tehát a megoldás? Mi a 16.4 példa normalizálásának az elvi alapja? Egyáltalán: normalizálható-e a modellrészlet?

Már többször utaltunk arra, hogy egy adatmodell egyedeinek a függéseit nem szabad csak önmagukban vizsgálni. A teljes függésrendszert együtt kell elemezni. A két táblázat által egyenként mutatott függési viszonyok (a kulcsok meghatározzák a leírókat) mellett tudjuk azt is, hogy létezik a Diszpoziciószám → Rendelésszám függés. Azért, mert kitételünk szerint minden diszpozició csak egy megrendelésre vonatkozhat.

Innen már csak egy lépés a probléma megoldása, ha ismerjük a ***pszeudo-tranzitivitás*** fogalmát:

D 16/4 **Az E egyed pszeudo-tranzitív függést tartalmaz akkor és csak akkor, ha a D tulajdonsága az A+B azonosító mellett függ az implicit A+C csoporttól is és fennáll a B → C vagy a C → B függés.**

Példánkban az R-mennyiség a Diszpoziciószám+Cikkszám együttese mellett függ az implicit Rendelésszám+Cikkszám csoporttól és ugyanakkor fennáll a Diszpoziciószám → Rendelésszám függés is. Tehát az R-mennyiség pszeudo-tranzitíven függ az azonosítótól.

Mivel az ilyen függés redundanciára vezet, azt ki kell küszöbölni. Ehhez a függés képletének az ismerete nyújthat segítséget:

X → Y
A {Y+Z} → C
B {X+Z} → C

Magyarul: Létezik két összetett tulajdonság (A és B), amelyek mindegyike meghatározza a harmadik (C) tételt. Eközben igaz az, hogy a két összetett tulajdonságnak van egy közös része (Z) úgy, hogy a nem-közös részek (X és Y) között funkcionális függés áll fenn.

A képlet szerint a harmadik függés felesleges, mert (pszeudo-)tranzitív. A dekompozíció szabályai szerint ilyenkor a függő tulajdonságot (R-mennyiség) a közvetlen meghatározóval (Rendelésszám+Cikkszám) együtt ki kell emelni a vonatkozó egyedből (DISZPOZICIÓTÉTEL) úgy, hogy a meghatározó az eredeti egyedben marad. A tranzitivitással szemben csak annyi az eltérés, hogy a meghatározó itt implicit és az is marad a kérdéses egyedtípusban. A 16.10 példa mutatja a 16.9 eset normalizálásának az eredményét.

16.10 példa

RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, ..., R-mennyiség)
DISZPOZICIÓTÉTEL (***Diszpozíciószám+Cikkszám***, D-mennyiség, ...)

Mivel a meghatározó és a függő párosa már létezett a RENDELÉSTÉTEL egyedben, a normalizálás nem jelent mást, mint a pszeudo-tranzitíven függő tulajdonság eltávolítását a másik egyedből. Ez mindig így törvényszerű. Ha a páros nem létezne, akkor nem lenne érvényes a képlet második tétele és fel sem fedezhetnénk a pszeudo-tranzitivitást.

A pszeudo-tranzitivitás ***elméleti*** háttere egyszerű. Azonban egy összetett modellben a helytelen függések ***gyakorlati*** megtalálása sokszor nem könnyű. Ismét gondoljunk a többszörösen összetett csoportokra. Például a következő összefüggésekre:

A {X+Y+Q+Z}
B {X+V+Q+W}
Y+Z $\rightarrow$ V+W

Két többszörösen összetett tételnek összetett a közös (X+Q) és a nem-közös része is úgy, hogy az utóbbiak között függés áll fenn. Ez a helyzet kiválóan alkalmas arra, hogy összegezzük fejezetünk mondanivalóját:

- A relációs normalizálás nem ismeri az összetett meghatározottat, ezért képtelen a legutóbbi eset megoldására. Általában még a legegyszerűbb tranzitivitást is csak akkor tudja kiküszöbölni, ha a meghatározottak mindegyike elemi.

- Az előzőek miatt a logikai szintű relációs normalizálásról át kell térnünk a fogalmi szintű szemantikus normalizálásra.

- A bonyolult összefüggések elemzése nem végezhető manuálisan. Ezért mindenképpen szükség van olyan automatára, amely képes az ebben a fejezetben ismertetett valamennyi normalizálási helyzet értékelésére is. Az automata arra szolgál, hogy rámutasson a problémákra. Viszont azok megoldása sokszor emberi feladat marad.

- A normalizálás nem korlátozódhat egy-egy egyedtípusra. A függési viszonyokat nemcsak egy egyedtípuson belül, hanem globálisan is értékelni kell.

Az utolsó kitétel már átvezet a következő fejezet mondanivalójához. Ott még az eddigieknél is világosabban kiderül, hogy nem az egyedtípusokat, hanem magát a teljes adatmodellt kell normalizálni.

Ellenőrző kérdések - 16

16/01 Melyik állítás igaz (I) és melyik hamis (H) a következő modellrészletre nézve:

SZÁMLA (***Számlaszám***, Cikkszám, Érték)
SZÁMLA (***Számlaszám+Cikkszám***, Érték)

- Feltételezhető, hogy a Számlaszám homonima.
- A két egyed külön-külön 5NF alakban van.
- Két egyednek nem illik azonos nevet adni.
- A modellrész rossz, de azt dekompozícióval nem lehet javítani.
- A belső kulcstörés technikai jelensége hívja fel a figyelmet a valós problémára.

16/02 Adott a következő két egyedtípus. A terméket egy bizonyos dolgozó meghatározott gépen állítja elő. Miért rossz a terv és miként nézne ki helyesen?

HASZNÁLJA (***Dolgozó-azonosító+Gépazonosító***, ..., Műszak)
TERMÉK (***Termékazonosító***, *Dolgozó-azonosító, Gépazonosító*)

16/03 Miért hibás a következő terv és mi a helyes megoldás? Kovács beteg lett és helyette - az előzetes tervvel szemben - Szabó kezelte a gépet.

HASZNÁLJA (***Használja-azonosító***, ..., Műszak)
TERMÉK (***Termékazonosító***, *Használja-azonosító, Dolgozó-azonosító*)
Használja-azonosító {Dolgozó-azonosító+Gépazonosító}

16/04 Milyen problémát rejt a következő terv? Egy számlán több rendelés tételei is megjelenhetnek, de egy rendelés tételei mindig egy számlán szerepelnek.

SZÁMLATÉTEL (***Számlaszám+Cikkszám***, ..., Érték)
RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, ..., Érték)

16/05 Adott a Szerződésszám tulajdonság. Ennek első pár jele nem más, mint a Vevőkód. További jelei az előzőn belül egyedi Kötésszám-ot jelentenek. Vagyis két vevőnek lehet azonos a Kötésszám értéke, de egy vevő minden szerződésének más a kötésszáma. A relációs rendszerek nem ismerik a csoportokat. Milyen gondok fakadnak ebből az Ön véleménye szerint?

17. NORMALIZÁLÁSI ELJÁRÁSOK

17.1 A normalizálás, mint feldolgozás

Ha egy egyedtípus nincs 2NF alakban, akkor ki kell szűrni a részleges függést. Ha nem 3NF alakú, akkor meg kell szüntetni a tranzitivitást. Mindenki könnyen megtanulhatja a miértet: a kedvezőtlen normálforma redundanciát, ezen keresztül pedig tárolótöbbletet, karbantartási anomáliát és inkonzisztenciát okoz. A kezdő tervezők hamar el tudják sajátítani az alapvető normalizálási megoldásokat. A helytelen függésű tulajdonságokat ki kell emelni új egyedtípusba úgy, hogy... Azonban akkor, amikor a gyakorlati életben elébük kerül egy valódi adatmodell, kazalnyi egyed-, tulajdonság- és kapcsolattípussal, akkor nem igazán tudják, hogy hol is fogjanak e kásahegynek.

Mindennek az az oka, hogy a normalizálást ***résztechnikaként*** fogják fel. Az egymástól szeparált minipéldákon keresztül ezt sugallják a szakkönyvek; így tanítják a modellezési kézikönyvek; ez a CASE-ekbe épített normalizálási részrutinok mögötti szemlélet is.

Pedig a normalizálás, mint elemzés maga sem más, mint ***ismeretfeldolgozás***. Mégpedig nem is az egyszerűbb fajtából. Mármost jól tudjuk, hogy minden feldolgozásnak négy alapvető tényezője van: bemenet, adatbázis, eljárás és kimenet.

A normalizálás ***bemenetét*** ebben a könyvben ***normalizálási alapnak*** fogjuk nevezni. Ez azon tényezők szervezett együttese, amelyeken végrehajtjuk az elemzést. Ide tartoznak maguk a modellelemek (egyedek, tulajdonságok) és a köztük lévő függési viszonyok.

Ez az alap nem más, mint egy előre elképzelt, kiinduló állapotú előzetes adatmodell. Ezt a normalizálási ***adatbázisban*** tároljuk. Az elemzések folyamán ezt átalakítjuk. Újabb bemenetek után ismételten feldolgozzuk addig, ameddig a kívánt eredményre jutunk.

A normalizálás terméke, végeredménye, ***kimenete*** a végső, az optimalizált adatmodell. Ez csak ritkán lesz teljesen tökéletes. Sohasem lesz teljesen lezárt, mert az élet változásaival maga az adatmodell is módosulni fog. Éppen ezért kell azt adatbázisban tárolni, mert különben minden változásnál újra bemenetként kellene megadni a normalizálás alapját.

Maga a normalizálás, mint feldolgozás a normalizálási alap és adatbázis függvényében különböző ***normalizálási eljárások*** szerint történhet, amelyek nyilván (részben) eltérő normalizálási kimeneteket eredményeznek.

Ennek a fejezetnek az a célja, hogy tájékoztatást adjon az általánosan ismert normalizálási eljárások bemeneteiről és eljárásairól. Kimutassa ezek tökéletlenségét. Vagyis azt, hogy ezekkel az eljárásokkal képtelenség jó kimenetet, optimális adatmodellt készíteni. Ezen túlmenően ismertetni fogunk egy teljesen új normalizálási megközelítést.

A „teljesen új” jelzőt nem szó szerint kell venni. Azon azt kell érteni, hogy a széles publikum számára még nem ismert eljárásról van szó. A könyv szerzője, barátai és ismerősei már csaknem tíz éve alkalmazzák ezt a folytonosan finomított, tökéletesített normalizálást.

17.2 Kapcsolathiány

Mielőtt a normalizálási eljárások ismertetésébe fognánk, rá kell mutatnunk arra, hogy mi indokolja a különböző normalizálási metódusok létjogosultságát. Ennek kapcsán pedig vissza kell térnünk az adatmodellel kapcsolatos négy alapvető kritériumra (ld. 13.1 pont).

Az optimális adatmodell valósághű, egyértelmű, teljes és minimális. Ez a felsorolás egyben fontossági rangsort is mutat. Ennek ellenére az eddigiek során a legnagyobb figyelmet a legkevésbé fontos kritériumnak, a ***minimalitásnak*** szenteltük. A hagyományos normálformák (ld. 14. és 15. fejezet) kivétel nélkül arra irányulnak, hogy akár a hatékonyság kárára is (vö. 4NF és 5NF) kiszűrjék a fizikai redundanciákat. A BCNF kapcsán még csak megpendítettük, az előző fejezetben pedig már hangsúlyoztuk az ***egyértelműség*** követelményét. A mechanikussal szemben a szemantikus normalizálás a modell magasabb fokú konzisztenciáját eredményezi. A csoportok bevezetésével egy lépést tettünk a ***valósághűség*** fokozásához is. Mert hiszen nem valósághű a modell, ha abban a Nyilvántartási számot vagy elemi adatként tüntetik fel és nem utalnak annak összetevőire, vagy az utóbbiakat felsorolják, de nem mondják meg, hogy azok egy egységet alkotnak. A csoport tényezője feloldja ezt az ellentmondást és így képes a valóság tökéletesebb tükrözésére.

Mindeddig meglehetősen hanyagoltuk a ***teljesség*** kérdését, bár a figyelmes olvasó már felfedezhetett néhány mozzanatot, amely ehhez a kritériumhoz kötődik. Az ismétlődések megszüntetésénél előfordul, hogy a kiemelt rész által alkotott egyed azonosítója nem teljes (ld. 13.4.3 alpont) és így azt ki kell egészíteni. A metszetfüggés kapcsán (ld. 16.5 pont) arra mutattunk rá, hogy a modellből hiányozhatnak kapcsolatok (kapcsolati integritási korlátok) és ezt a bajt csak a csoportos kapcsolótulajdonsággal számolhatjuk fel.

Ebben a pontban is az egyedek közötti viszonyokkal foglalkozunk, de most sokkal általánosabban. Az adatmodellek gyakori hibája a ***kapcsolathiány*** (angolul: inconnectivity). Erről már a 12.4.5 alpontban is szóltunk. A 12.3 ábra bevezető példájában a KOCSI és a TULAJDONOS egyede nem volt egymáshoz köthető, mivel egyik egyed sem tartalmazta a másik azonosító tulajdonságát. Bár a két egyednek voltak közös tulajdonságai (pl. Tulajdonosnév), jól tudjuk, hogy a két egyed nem kapcsolható össze a leíró szerepű tulajdonságon, hiszen létezhet például több „Kovács Rózsa” nevű tulajdonos is.

A 17.1 példa mutatja a kapcsolhatatlanság legegyszerűbb esetét:

17.1 példa

VEVŐ	(***Vevőkód***, Vevőcím)
RENDELÉS	(***Rendelésszám***, Vevőnév)

A példa két egyedében nincs közös tulajdonság, így sohasem juthatunk el a VEVŐ egyedből a RENDELÉS egyedbe vagy megfordítva. Nem tudjuk visszakeresni, hogy egy vevőnek melyek a rendelései és azt sem, hogy milyen címen található a rendelést feladó vevő. Ezért bár mindkét egyedtípus a legtökéletesebb (5NF) normálformában van, a tervrészlet mégis pocsék. Ráadásul a kapcsolathiány gondja általában nem jár önmagában; nagyon sokszor egyéb problémákkal is párosul. Nézzük csak meg a 17.2 példát.

17.2 példa

VEVŐ	(***Vevőkód***, Vevőcím, Vevőnév)
RENDELÉS	(***Rendelésszám***, Vevőnév)

Most már világosan látszik a Vevőnév nyílt logikai átfedése, miközben a két egyed továbbra is kapcsolhatatlan. A tervezők ezt a kettős problémát nyilván „ösztönösen” is fel tudják oldani. Azonban több tucat vagy száz egyedtípust és több száz vagy ezer tulajdonságtípust felölelő - és még ekkor is csak közepes nagyságú - adatmodell esetében az ösztönös tervező tévedhet. Ezért meg kell keresni a probléma általános elméleti megoldási módszerének a nyitját.

A 14.3 pontban utaltunk arra, hogy megkülönböztetünk ***egyeden belüli*** (intra-entity) és ***egyedek közötti*** (inter-entity) függéseket. Mindeddig csak az előzővel foglalkoztunk. Ez szükségszerűen következett abból, hogy eddig csak a ***dekompozíciós*** normalizálási eljárást ismertettük. Ennek a módszernek pedig az a lényege, hogy egyenként - egymástól függetlenül - vizsgálja az egyedek szerkezetét és csakis a belső szerkezet hibáival törődik. Ez a módszer nem alkalmas az egyedek külső szerkezetének - kapcsolataik együttesének - az elemzésére.

A kérdés tehát az, hogy milyen metódus segítségével fedezzük fel a 17.2 példa által mutatott hibákat általánosan is. Azaz konkréten miként találjuk meg a 17.3 példa hibamentes szerkezetét.

17.3 példa

VEVŐ	(***Vevőkód***, Vevőcím, Vevőnév)
RENDELÉS	(***Rendelésszám***, *Vevőkód*)

A 17.3 példa megoldása tökéletes. Nemcsak a Vevőnév redundanciáját szüntettük meg, hanem a Vevőkód tulajdonságon át megteremtettük a két egyed korrekt kapcsolatát is. A Vevőkód az egyik egyedben azonosító, a másikban leíró szerepű. Így tökéletesen alkalmas a két egyedtípus közötti ún. ***hivatkozás-integritás*** (angolul: referential integrity) kifejezésére. A RENDELÉS egyedben egyértelműen tudunk hivatkozni a rendelést feladó vevőre és a VEVŐ egyedből kiindulva megtaláljuk az egyes vevők rendeléseit.

Végeredményben a kapcsolathiány azért jelent gondot, mert nem tárjuk fel a hivatkozás-integritást, ami nem más, mint az egyedek külső szerkezetére vonatkozó ***korlát***. Példánk esetében ez a korlát így hangzik: „Minden vevőnek lehet 0, 1 vagy N rendelése. Ugyanakkor minden rendelésnek egy és csakis egy vevőhöz kell tartoznia.” Amennyiben két egyed között hiányzik ez a kapcsolat vagy - ami ugyanaz - hivatkozás, úgy helytelen összefüggésű adatok kerülhetnek az adatbázisba (vevőhöz nem kapcsolható rendelés, mint a 17.1 példában), tehát az adatbázis épsége, integritása - valósághűsége - megsérül.

A problémák gyökere az, hogy a relációs modell nemcsak a csoport, hanem a kapcsolat fogalmát sem ismeri. A kapcsolattípus az egyedtípusok olyan külső viszonya, amely a két egyed belső szerkezetén, a közös kapcsoló-tulajdonságtípuson alapul - és megfordítva. Nos, ez a „megfordítva” kitétel nem érvényesül a relációs rendszerekben. A hivatkozás-integritási korlát csupán azt mondja ki, hogy ha már a B egyedben van az A egyed kulcsa, akkor minden B előfordulás csak létező A előforduláshoz kötődhet. Mivel kapcsolatot nem lehet a relációs rendszerekben meghatározni, így nem érvényesülhet a kölcsönösség elve. Nem lehet kijelenteni, hogy amennyiben az A egyedtípus a B egyed fölérendeltje, úgy az utóbbiban fel kell tüntetni az előbbi elsődleges kulcsát.

Itt rá kell mutatnunk a ***kapcsolhatóság*** kétféle értelmezésére. A relációs tervezés ***logikai*** szintű módszer. Ha két reláción végre lehet hajtani az összekapcsolás (join) műveletét, akkor azokat kapcsolhatóknak tekintik. A ***fogalmi*** modellezés szintjén ez csak szükséges, de nem elégséges feltétel. Két egyedtípus akkor kapcsolható, ha köztük kapcsolattípust lehet meghatározni a közös kapcsolótulajdonságon át, amely legalább az egyik egyedtípusban azonosító. Mármost az MVD és a JD (ld. 15.3 és 15.4 pontok) szerinti dekompozíció mindig olyan relációkat/egyedtípusokat eredményez, amelyek logikailag kapcsolhatók, de fogalmilag nem azok. (Ezért nem szeretjük ezeket a megoldásokat.) A kétféle kapcsolhatóság között óriási a különbség. A fogalmilag kapcsolható modellrészben definitíven lehet megadni a szerkezetet és a korlátot. Viszont a csak

logikailag kapcsolható egyedek viszonyát procedurálisan kell korlátozni. (Ha az egyik egyedet aktualizáljuk, akkor a másikkal is tenni kell valamit. Azt, hogy mit, nekünk kell programoznunk.)

Az 1-3NF alakra történő dekompozíció fogalmilag is kapcsolható tényezőket eredményez. Ennek ellenére még ez az eljárás is tökéletlen. Ugyanis az alapvető normálformák jól tájékoztatnak arról, hogy milyen tulajdonság ***nem lehet*** az egyedben, de azt nem mondják el, hogy melyeket ***kell*** az egyedtípushoz kötni. Mert nézzük meg összefoglalóan, hogy mit is jelent a harmadik normálforma:

Az egyedtípus akkor és csak akkor van legalább harmadik normálformában, ha minden nem-kulcs tulajdonsága függ a kulcstól (1NF), csakis a teljes kulcstól (2NF) és semmi mástól, csak a kulcstól (3NF).

Ez a szöveg analóg az angolszász bíróságokon a tanú által tett esküével:

„Esküszöm, hogy az igazat fogom mondani, a teljes igazat és semmi mást, csak az igazat."

Vegyük észre, hogy az esküszöveg nem tartalmazza ezt a kitételt: „Meg fogom mondani az igazat.". A tanú nem köteles magára vallani. Ugyanez a hiány fedezhető fel a 3NF meghatározásában is. Mármost a „teljes igazság" adatmodellezési titka a következő módon fogalmazható meg:

Ha egy tulajdonságtípus közvetlenül függ egy másiktól, akkor a másik által azonosított egyedtípusban van a helye.

A 17.2 példa esetében a Vevőkód közvetlenül függ a Rendelésszámtól, noha ott ezt a függést egyelőre nem látjuk. Azért nem, mert a dekompozíció módszere nem foglalkozik az egyedek közötti függésekkel és ezért a fenti szabályt nem is alkalmazhatja. Így a 17.2 példa modellje rossz. Ha érvényesítenénk az elhangzott szabályt, akkor a Vevőkódot az őt közvetlenül meghatározó Rendelésszám által azonosított RENDELÉS egyedbe kellene tennünk, érvényesítve a kapcsolati teljesség kritériumát. Ezzel végül a 17.3 példa megoldásához jutnánk.

Ha a teljességi szabályt betartjuk, akkor első lépésként a 17.4 példa megoldását kapjuk:

17.4 példa

VEVŐ	(***Vevőkód***, Vevőcím, Vevőnév)
RENDELÉS	(***Rendelésszám***, *Vevőkód*,Vevőnév)

Amint látjuk, a RENDELÉS egyedben lévő tranzitivitás most már nyilvánvalóvá vált. A teljesség szabályát tehát két dolog miatt kell komolyan vennünk. Egyrészt a valósághűség miatt (a 17.2 példában a vevők és a rendelések nem kapcsolódnak úgy, mint a valóságban). Másrészt a minimalitás érdekében (a 17.2 példában nem fedezhettük fel a Vevőnév tranzitivitását).

A nem-minimális adatmodell lehet teljes és valósághű. A hiányos nem lehet valósághű és esetleg nem is minimális. Így látható, hogy

a modell optimumkritériumai szorosan összefonódnak.

Olyan normalizálási eljárásra van szükség, amely ezt a tételt felismeri és alkalmazza, amely nemcsak a minimalitás követelményét hangsúlyozza.

17.3 A dekompozíció összefoglalása

A normálforma dekompozíció módszerének alkalmazásakor a ***normalizálás alapja*** kettős. A tervező egyrészt előfeltételezi az egyed-tulajdonság-viszonyok halmazát. Vagyis meghatározza az általa elképzelt egyedtípusokat és az azokhoz kötött tulajdonságtípusokat, azaz attribútumokat. Másrészt egyedenként feltárja a belső funkcionális, többértékű és kapcsolási függéseket. Tehát a ***normalizálási adatbázist*** az egyedenként szeparált attribútumok és a közöttük meghatározott belső függések képezik. A 14. és 15. fejezetekben leírt szabályok szerint történik az attribútumok levágása új egyedtípusokba.

A ***normalizálási eljárás*** így megbontást és - közös kulcsú egyedek esetében - összevonást jelent. Ha a Vevőnév tulajdonságot kiemeljük a RENDELÉS egyedből és így kapunk egy VEVŐ (***Vevőkód***, Vevőnév) átmeneti egyedet, akkor ha ilyen egyedünk még nincs, azt véglegessé tesszük. Ha van már VEVŐ egyedünk, akkor oda vándoroltatjuk át a Vevőnév tételt. A helyesen végrehajtott normalizálás ***kimenete*** csupa 5NF alakú egyedtípus lesz, hacsak szándékosan nem denormalizálunk (ld. 14.4 pont) illetve a nem-független lebontás miatt (BCNF, 4NF, 5NF) nem mondunk le a normalizálásról. Az eredményezett egyedtípusok együttese nem feltétlenül alkot kapcsolható, teljes adatmodellt (ld. az előző pontot). Így ***a dekompozíció módszere elméletileg tökéletlen modellelemzési eljárás***.

17.4 Az „univerzális reláció”

A normálforma dekompozíció egymástól szeparált egyedtípusokat normalizál. Ezért nem képes feltárni a 17.2 példa Rendelésszám és Vevőkód tétele közötti függést. Hiszen a két tétel két különböző egyedtípust jellemez. Ezt a viszonyt akkor lehetne feltárni, ha a két tulajdonságtípust valamiképpen egyetlen egyedtípusba tennénk. Azonban ha már látjuk az összekötés szükségességét, akkor az már feleslegessé is válik, mert megtaláltuk a helyes megoldást. A gyakorlatban éppen az a probléma, hogy nem vesszük észre a tételek közötti függési viszonyokat. Azért nem, mert X db egyedünk és Y db tulajdonságunk van.

Ezt a gondot egy ideig az ún. ***univerzális reláció*** (angolul: universal relation) - ld. [29] - konstrukcióval próbálták feloldani. Az alkalmazás összes - egyértelműen meghatározott, tehát nem homonim és szinonim - tulajdonságtípusát egyetlen óriási általános „egyedtípus” tényezőiként képzelték el. Így természetesen egymás mellé - vagyis egy „egyedtípusba” - kerül minden tulajdonság. A tulajdonságok függései megvizsgálhatók. Ekkor már látható a Rendelésszám → Vevőkód funkcionális függés, a Vevőnév tranzitivitása és minden egyéb közvetett - tehát megszüntetendő - és közvetlen tulajdonság viszony.

Ebben a módszerben a ***normalizálás alapja*** az egyetlen univerzális egyedtípus annak attribútumaival, vagyis az alkalmazás összes tulajdonságtípusával és a közöttük meghatározott közvetlen - nem-részleges, nem-tranzitív, nem-többértékű stb. - függésekkel. Ez a bemenet képezi a ***normalizálási adatbázist*** is, amelyet minden változás esetén újra kell definiálni, hiszen módosul maga az univerzális reláció. ***A normalizálási eljárás*** nem más, mint az univerzális reláció megbontása. Annyiban különbözik a dekompozíciótól, hogy nem lépésenként történik, mert csakis a közvetlen függések mentén halad. Nem alkotunk először 2NF, majd 3NF stb. alakú, esetleg továbbnormalizálandó egyedeket, hanem egycsapásra a végső egyedeket alakítjuk ki. A ***normalizálási eredmény*** a dekompozícióénál elvileg teljesebb, hiszen az egyedek közötti függések az univerzális egyeden belüli függésekként jelentkeznek. Így elméletileg nem feledkezhetünk meg a Rendelésszám → Vevőkód függésről és teljes, kapcsolható adatmodellt kapunk.

Elméletileg. Azonban ez a módszer - más szempontból - elvileg is hibás. Gyakorlatilag pedig eleve kudarcra volt ítélve. Most csak a gyakorlati okot ismertetjük, amely a darabszámokban rejtőzik. Ha csak pár tucat tulajdonságtípusunk van az egyetemes relációban, akkor a közöttük lévő függéseket könnyen meghatározhatjuk. Viszont már 500 tulajdonságtípus ***elemi funkcionális*** függéseinek az elemzéséhez is elvileg 127245 vizsgálatot kellene végrehajtani. Minden tulajdonságot minden másikkal össze kellene vetni annak megállapítására, hogy létezik-e közöttük elemi funkcionális függés. Tehát N tulajdonság esetében N * (N-1)/2 vizsgálatra lenne szükség.

Ekkor még szót sem ejtettünk az A+B $\rightarrow$ C jellegű ***összetett*** függésekről, amelyekben maga az A, B és a C is lehet összetett. Vagyis az összetételek tetszőleges kombinációit is vizsgálni kellene. Ezeket is figyelembe véve már csillagászati számokhoz érkezünk. Pedig az embernek még arra sincs gyakorlati lehetősége - magyarul: ideje -, hogy pár tízezer tulajdonságfüggést átböngésszen. Ezért ***az egyetemes reláción alapuló normalizálási eljárás elméletileg és gyakorlatilag alkalmazhatatlan.*** (Az elméleti tökéletlenséget majd alább igazoljuk.)

17.5 Normálforma szintézis

Mielőtt a módszer ismertetésére áttérnénk, rá kell mutatnunk egy lényeges momentumra.

A hagyományos tervezési eljárások abból indulnak ki, hogy a tulajdonságtípusok egyedtípusokhoz tartoznak. Tehát az egyedtípusokat elsődleges modellezési tényezőknek tekintik, szemben a másodlagos tulajdonságtípusokkal. Teszik ezt annak ellenére, hogy az utóbbi tényezők önálló jogon is léteznek (vö. doméjn) és ezért nem másodlagosak. Sőt! A relációs elmélet szerint a reláció az értéktartományok Descartes-féle szorzata. Ezért a doméjn elvileg megelőzi a relációt, ahhoz képest elsődleges. A mi véleményünk szerint az adatmodellben az egyed- és tulajdonságtípusok egyenrangú tényezők. Ezért evidens, hogy az adatmodellezést éppen úgy lehet az utóbbiaknál is kezdeni, mint az előbbieknél.

Ezt a gondolatot ismerte fel Bernstein, a ***normálforma szintézis*** [30] eljárásának a kidolgozója. Ebben a módszerben nem az egyedtípusok, hanem az azoktól független tulajdonságtípusok (értéktartományok) halmaza és az azok közötti függések jelentik a ***normalizálás alapját***. Mivel ezek a függések nem egyeden belüliek, nem FD-MVD-JD jellegűek. Az ún. tartományfüggésekről (ld. 13.3 pont) van szó. A ***normalizálási adatbázist*** a doméjnek, a tartományfüggések és az ezek alapján meghatározott egyedtípusok jelentik. Az ezeken belüli tulajdonságfüggéseket ez a módszer tartományfüggésekként értelmezi.

Az egyedtípusokat tekintve a dekompozíció felülről-lefelé lebontó ***normalizálási eljárás***. Ezzel szemben a szintézis alulról-felfelé építkező módszer. Az egyedtípusokat tulajdonságtípusokból építjük föl a közöttük meghatározott tartományfüggések (DD) szerint. Ez a következő „recept” szerint történik:

- Vedd a tulajdonságtípusok (mint tartományok) teljes készletét.
- Határozd meg a közöttük lévő valamennyi tartományfüggést.
- Számold fel a nem-közvetlen (részleges, tranzitív stb. jellegű) függéseket.
- Hozz létre annyi egyedtípust, ahány különböző meghatározó van a „baloldalon”. Minden ilyen egyedtípusnak a baloldali tulajdonságtípus lesz az azonosítója.
- Az így létrehozott egyedtípusokhoz a baloldali meghatározók szerint kapcsold hozzá leíró tulajdonságként a „jobboldali” tulajdonságokat.
- Az egyedek között annyi kapcsolatod lesz, ahányszor a „baloldali” tényező más függésben megegyezik a „jobboldali” tétellel.

Most lássuk mindezt a 17.5 példán úgy, hogy a 17.3 példa tulajdonságtípusait és az azok közötti függéseket vesszük alapul:

17.5 példa

Vevőkód ==> Vevőnév
Vevőkód ==> Vevőcím
Rendelésszám ==> Vevőkód
Rendelésszám ==> Vevőnév
Rendelésszám ==> Vevőcím

A két utolsó függés tranzitív, tehát töröljük azokat a függéshalmazból. Marad két különböző baloldalunk. Ezért két egyedünk (VEVŐ, RENDELÉS) lesz a két baloldali tétellel (Vevőkód, Rendelésszám), mint azonosítóval. Az első egyedben a Vevőnév és Vevőcím, a másikban a Vevőkód lesz jobboldali tételként leíró tulajdonság. Egyetlen olyan párosunk van, amely bal- és jobboldalt is szerepel. Ezért a modellben egy kapcsolat van, mégpedig a VEVŐ és a RENDELÉS között. A kapcsolat a Vevőkód kapcsolótételen alapul.

A szintézissel „kapásból" a 17.3 példa tökéletesen jó megoldására jutottunk. Annyira nemesen egyszerű módszerrel, hogy az szinte nem is lehet igaz. Nem is az.

Az élesszemű olvasó nyilván felfedezte, hogy az univerzális reláció és a szintézis mennyire közel áll egymáshoz. Ha a tényleges egyedtípusoktól függetlenül, egy nagy egyetemes egyedtípusban vizsgáljuk a függéseket, akkor azok csak tartományfüggések lehetnek. Az egyetemes reláció és a szintézis függéshalmaza csak egy elvi dologban tér el egymástól: a kapcsolásfüggés (JD) csak egyedtípuson belül értelmezhető, értéktartományok között nem. A fentiekből következik, hogy a 17.4 pontban feltárt gyakorlati probléma a szintézisre is vonatkozik. A szintézis szép elmélet, csak ***gyakorlatilag*** végrehajthatatlan. Megfordítva: az alább ismertetett elméleti korlátok az univerzális relációra is vonatkoznak.

A gyakorlatban ritkán bár, de előfordul, hogy egy E (***A+B***) egyedtípust csak azért hozunk létre, hogy össze tudjuk kapcsolni az egymással M:N fokú viszonyban álló F (***A***, ...) és G (***B***, ...) egyedtípusokat. Teljesen világos, hogy az E egyedtípus a szintézis módszerével sohasem kreálható. Hiszen ebben az egyedben semmilyen függés sem fedezhető fel, kivéve a két projektívet, vagyis azt, hogy az összetett kulcs meghatározza a saját részeit. Így sohasem fogunk rájönni arra, hogy szükséges maga az A+B összetétel. Ezért - szemben a dekompozíció eljárásával - szintézissel sohasem tudunk meghatározni ***csupakulcs egyedtípusokat***.

A szintézis eljárása ***szemantikailag*** teljesen hibás modellre vezethet. Most idézzük fel a 13.5 példát a 17.6 példában:

17.6 példa

KÖZPONT	(***Központ-azonosító***, ...)
TELEPHELY	(***Telephely-azonosító***, ..., *Központ-azonosító*)
RENDELÉS	(***Rendelésszám,*** *Központ-azonosító, Telephely-azonosító,* ...)

Emlékezzünk rá, hogy a rendelést hol a központ, hol a telephely adja fel. Ezért a RENDELÉS egyedben mindkét tulajdonságnak (Központ-azonosító és Telephely-azonosító) szerepelnie kell. Ha a központ adja fel a rendelést, akkor a Telephely-azonosító értéke értelmetlen. Ezért a RENDELÉS egyedben nem áll fenn a Telephely-azonosító → Központ-azonosító funkcionális függés. Viszont ugyanez a viszony határozottan létezik a TELEPHELY egyedben.

A normálforma dekompozíció eljárásánál ez a kettősség nem okoz zavart. A 17.6 példa megoldása minden további nélkül elfogadható a dekompozíció szabályai szerint. Ezzel szemben

már világosan látható a normálforma szintézis igen komoly elméleti korlátja. Ennél az eljárásnál két értéktartomány között csak egy függési viszony határozható meg. Így a tervezőnek csak két - egyaránt rossz - lehetősége van.

Ha az elemző közvetlennek feltételezi a Telephely-azonosító ==> Központ-azonosító függést, akkor megkapja a TELEPHELY egyedet. Viszont nem teheti a RENDELÉS-be a Központ-azonosítót, mert az közvetett - tranzitív - a szintézis szabályai szerint. Így nem jön létre a KÖZPONT - RENDELÉS kapcsolat. Ha a tervező kijelenti, hogy a Telephely-azonosító és a Központ-azonosító független, akkor a RENDELÉS egyedben együtt szerepelhet a két tétel. Viszont nem kerül a TELEPHELY egyedbe a Központ-azonosító és így kimarad a modellből a KÖZPONT - TELEPHELY kapcsolat.

Tehát a szintézis valójában kizárja azt, hogy két tulajdonság több egyedben is együtt forduljon elő, ha van olyan egyed, amelyben az egyik meghatározza a másikat. Ezért ***a szintézis eljárása elméletileg tökéletlen.*** A függések teljes halmazának az igényéből indult ki ez az eljárás, és éppen ezt a teljességet nem teszi lehetővé.

Íme, a normalizálás ***22-es csapdája***. A dekompozíció nem figyel az egyedek külső viszonyaira és ezért nem képes jó belső szerkezeteket biztosítani. A szintézis elhanyagolja az egyedek belső szerkezetét és ezért képtelen jó külsőt produkálni. Az univerzális reláció pedig ötvözi a másik két eljárás hibáit amellett, hogy gyakorlatilag is alkalmazhatatlan.

Mielőtt a csapdából való kikecmergés módját megmutatnánk, fel kell hívnunk a figyelmet a szintézis egyik pozitív - nem mindenki által kihasznált - vonására.

17.6 Kérdezni tudni kell

A kezdő normalizálók gyakori kettős hibája az, hogy az általuk előre elképzelt egyedtípusokhoz és még inkább azok azonosító tulajdonságához ragaszkodnak. Azt a kérdést teszik fel, hogy függenek-e az egyed tulajdonságai az előre elképzelt azonosítótól és a feltárt függések helyesek-e (nem-részlegesek, nem-tranzitívak stb.). Ez a megközelítés elvileg picit helytelen, gyakorlatilag pedig nagyon nem célravezető.

Elvileg azért helytelen, mert a dekompozíció nem mindig abból indul ki, hogy az egyedtípusnak adott az elsődleges kulcsa. Az előző két fejezetben sokat beszéltünk az olyan egyedekről, amelyeknek több kulcsjelöltje is van. Gyakorlatilag azért nem célszerű a dekompozíció merev követése, mert lépésenkénti normalizáláshoz - többletmunkához - vezet (vö. 14.1 példa). Azt pedig a 17.2 pontban láthattuk, hogy a dekompozíció módszerével nem mindig alkotható ***konnektív***, vagyis kapcsolathiány-mentes adatmodell.

Mi lesz a következménye annak, ha valamilyen tényezőt kifelejtünk a modellből, tehát a tervünk nem teljes? Ha egy egyedtípusból ***leíró*** abszolút szerepű tulajdonságot - pl. Vevőnév - hagyunk ki, az nem jár súlyos hátrányokkal. Az első információigénykor fény derül a hibára; az adatmodellt és az adatbázist kibővítjük; pár egyszerű programot átírunk és kész. Persze ezt is jobb lenne elkerülni. Ámde akkor, ha ***azonosító*** abszolút szerepű tulajdonság marad ki kapcsolóként valamelyik egyedből, akkor az előzőek mellett komolyabb kellemetlenségekkel is szembe kell néznünk. Ha a 17.2 példa RENDELÉS egyedébe nem tettük be a Vevőkód tulajdonságot, akkor nem derül fény a Vevőnév tranzitivitására. A hiány pótlásakor meg akarnánk szüntetni a redundanciát is. Ehhez viszont az adatbázis átalakítása és a RENDELÉS egyedet kezelő programok módosítása is szükséges.

Persze az idézett példa túlzottan is áttekinthető ahhoz, hogy valaki elkövesse az inkonnektivitás, a nem-teljes belső egyedszerkezet kialakításának a kettős problémáját. Ámde egy összetett adatmodellnél számolni kell a figyelmetlenséggel és áttekinthetetlenséggel. Ezért most eláruljuk

az adatmodellezés egyik fontos titkát, amelyen alapul a mi normalizálási szemléletünk és eljárásunk. A lényeg az, hogy „kérdezni tudni kell”.

A kezdő tervező az egyedtípus ***belső szerkezetét*** illetően azt a kérdést teszi fel, hogy ***függ-e*** a RENDELÉS (***Rendelésszám***, ..., Vevőkód, Vevőnév) Vevőnév tulajdonsága a feltételezett kulcstól és ha igen, akkor megfelelő-e a függés? Majd rájön arra, hogy nem az, mert tranzitív. Ezzel szemben a tapasztalt elemző csak egyetlen kérdést vet fel: ***mitől függ*** közvetlenül a Vevőnév tulajdonság? Magyarul: megfordítja a kérdés irányát. Nem a feltételezett azonosítótól kérdez a leíró felé, hanem a leíróhoz keresi a megfelelő kulcsot. Evidens, hogy így a leíró valódi helyét - például többszörös tranzitivitás esetén - sokkal hamarabb találja meg. Ha az E1 (***A***, B, C, D) egyedben létezik az A → B → C → D függéssor, akkor az A-tól indulva csak több lépésben bukkan a D végső egyedére (vö. migráltatás). Ezzel szemben ha azt kérdezi, hogy mitől függ közvetlenül a D, akkor azonnal felfedezi az E3 (***C***, D) egyedet.

Nézzünk másik példát! Adott a DISZPOZICIÓTÉTEL (***Diszposzám***, ..., Rendelésszám, Cikkszám, Rendelt-mennyiség) egyed. A tervező talán sohasem jön rá a megfelelő megoldásra, ha azt kérdezi, hogy függ-e és helyesen függ-e a Rendelt-mennyiség a Diszposzám-tól. Ha viszont az iránt érdeklődik, hogy mi határozza meg a Rendelt-mennyiség-et, akkor rábukkan a Rendelésszám+Cikkszám → Rendelt-mennyiség függésre (vö. 16.5 példa).

A normálforma szintézis korszerűsített, általunk alkalmazott eljárásban merült fel először a fordított irányú kérdezés gondolata. Hiszen ennél a módszernél nincsenek előre elképzelt egyedtípusok és azonosítók. Ha van 500 tulajdonságtípusunk azt végigkérdezni, hogy az egyik meghatározza-e az összes többit, praktikus lehetetlenség. Éppen ezért azt a kérdést vetjük fel, hogy azt az egyet melyik határozza meg a többi 499 közül közvetlen módon. Tehát - szemben az eredeti eljárással - nem állapítjuk meg az összes függést.

Ilyenformán elvi lehetőség nyílik az egyedtípusok ***külső szerkezetének*** a kiteljesítésére is. Egyedtípustól függetlenül mi határozza meg közvetlenül a Vevőkódot? Nos, többek között a Rendelésszám. Ezért a Vevőkód-nak a RENDELÉS egyedben van a helye. Persze a gyakorlatban ez másképpen fest, mert 500 tulajdonság esetében nem fogjuk végignézni a többi 499-et, hogy mi határozza meg közvetlenül a szóbanforgó egyet. Azt a titkot, hogy miért nem, majd csak a 17.10 pontban fejtjük ki a maga teljességében. Itt és most egy résztitokra hívjuk fel a figyelmet.

A dekompozíció az egyed belső szerkezete felől igyekszik megtalálni a külsőt, nem mindig sikeresen. A szintézis a tulajdonságfüggések alapján próbálja ugyanezt megtenni. Ám mi lenne akkor, ha fordítva is kérdeznénk? Nemcsak függésekben gondolkodnánk, hanem explicit külső szerkezetekben, vagyis kapcsolatokban is? Példa: Tudjuk, hogy a vevők és a rendelések hierarchikus viszonyban állnak, mert modellünkben meghatároztuk a VEVŐ - RENDELÉS kapcsolatot. Akkor hogyan lehetséges az, hogy a RENDELÉS egyed mégsem tartalmazza a VEVŐ azonosítóját (ld. 17.1 példa)? Tehát a kapcsolat alapján ki kell egészítenünk a RENDELÉS belső szerkezetét a Vevőkód-dal. Az explicit külső szerkezetből indulva így is javíthatjuk a belsőt. Mindez pedig ismételten a függések és az egyedek belső-külső szerkezetének az összefüggéseire irányítja a figyelmünket.

17.7 Függések és szerkezetek

Nem véletlenül hangsúlyoztuk annyira eddig is a normálforma és a modellstruktúra viszonyát. A tulajdonságtípusok egyedeken belüli (intra-entity) és egyedek közötti (inter-entity) függése ugyanannak a dolognak a két oldala. Ha a 17.4 példában fennáll a RENDELÉS egyeden ***belül*** a Rendelésszám → Vevőkód függés, akkor ugyanez a viszony a RENDELÉS egyed Rendelésszám és a VEVŐ egyed Vevőkód tulajdonsága ***között*** is létezik.

A hagyományos relációs normalizálás alapvető gondja az, hogy az egyedek külső szerkezetét csak ***impliciten*** - korlátokkal illetve ekként szolgáló függésekkel - lehet megadni. Azokat nem lehet ***expliciten*** - kapcsolattípusokkal - kifejezni, mivel ez a fogalom nem ismeretes a relációs adatmodellben. Ezért nincs is mód az explicit külső szerkezet alapján áttervezni a belső szerkezetet. Pedig az implicit és explicit struktúra viszonya világos.

A tulajdonságok közötti FD nem más, mint a tulajdonságok ***hierarchikus*** (1:N fokú) viszonya. Minden Rendelésszám értékhez csak egy (1) Vevőkód érték tartozhat, viszont minden Vevőkód érték több (N) Rendelésszám tartalomhoz is kapcsolódhat. Ha most ezt a viszonyt nem a RENDELÉS egyeden belül, hanem a VEVŐ és a RENDELÉS egyedek között szemléljük, akkor eljutunk az inhomogén hierarchikus birtoklási kapcsolattípushoz.

A Rendelésszám és a Vevőkód abszolút szerepe azonosító. Ez azt jelenti, hogy a két tulajdonság értékei 1:1 viszonyban állnak magukkal az egyedelőfordulásokkal. Minden konkrét egyednek kell, hogy legyen azonosítója (az azonosító értéke nem lehet üres vagy ismeretlen) és minden egyednek csak egy elsődleges kulcsa lehet. Így a Rendelésszám → Vevőkód függés tulajdonságértékek közötti, elölről nézve N:1 fokú viszonya egyben azt is jelenti, hogy a RENDELÉS és a VEVŐ egyedtípusok között is N:1 fokú összefüggés áll fenn. Ez a kapcsolat inhomogén (kétféle egyedet érint), hierarchikus és birtoklási természetű (a vevőknek vannak rendeléseik). Összefoglalóan: A RENDELÉS egyeden ***belüli*** N:1 fokú funkcionális függés mint implicit viszony a VEVŐ és a RENDELÉS ***közötti*** fordított irányú 1:N fokú explicit összefüggésnek, kapcsolatnak felel meg.

Ez a tudás már egészen komoly kiinduló alapot jelent a számunkra. Ha két egyedtípus kulcsa egymástól funkcionálisan független, akkor azok értékei és maguk az egyedtípusok is M:N fokú, ***hálós*** viszonyban állnak egymással. A két egyedtípus összefüggése nem lehet közvetlen. Egyik sem tartalmazhatja a funkcionális függetlenség miatt a másik azonosítóját. Így nincs lehetőség arra, hogy a két egyedtípus között kapcsolatot határozzunk meg. Ha a két egyedet összefüggésbe akarjuk hozni, akkor egy harmadik egyedtípust kell felvennünk olyan módon, hogy az mindkét egyedtípushoz közvetlenül kapcsolódjon. A 17.7 példa mutat ilyen esetet:

17.7 példa

RENDELÉS	(***Rendelésszám***, ...)
CIKK	(***Cikkszám***, ...)
RENDELÉSTÉTEL	(***Rendelésszám+Cikkszám***, ...Rendelt-mennyiség)

Az első két egyed viszonya hálós. Összefüggésüket a harmadik egyed fejezi ki. Mint már tudjuk, az összetett kulcs projektív függéssel meghatározza a saját részeit. Ezért a harmadik egyed Rendelésszám+Cikkszám → Rendelésszám és Rendelésszám+Cikkszám → Cikkszám belső függése a két meghatározott azonosító abszolút szerepe miatt egyben külső függés is. Ezért a RENDELÉSTÉTEL alulról N:1 fokú kapcsolattal kötődik a másik két egyedhez és így megtestesíti azok M:N fokú viszonyát. Egy (1) rendeléshez több (N) rendeléstétel tartozhat és egy (1) cikkre több (M) rendelés vonatkozhat.

Most nézzük a harmadik esetet, amelyet a 17.8 példa szemléltet:

17.8 példa

KOCSI	(***Rendszám***, ...)
CASCO	(***Cascoszám***, ...)

A „félpélda” nem véletlenül sikeredett ilyenre. Minden kocsinak csak egy CASCO-biztosítása lehet és minden ilyen biztosítás csak egy kocsira köthető. Ezért a Rendelésszám és a Cascoszám értéktartományok között kölcsönös, vagyis 1:1 fokú viszony áll fenn. A dilemma az, hogy miként fejezzük ki ezt az összefüggést, mert hiszen a 17.8 példa két egyede ebben a formában nem köthető egymáshoz.

Ha a normálforma szintézis elveihez ragaszkodnánk, akkor ugyancsak bajba kerülnénk. A KOCSI egyedbe kellene tennünk a Cascoszámot (meg a CASCO minden egyéb adatát), mivel az függ a Rendszámtól. A CASCO egyedbe kellene vinnünk a Rendszámot (meg a KOCSI összes többi tulajdonságát), mert az függ a Cascoszámtól. Így két teljesen azonos - csak más nevű és kulcsú - egyedtípust „nyernénk”.

Ilyen helyzetekben a normálforma dekompozíció sem segít. Gondoljuk csak át ezt az esetet:

17.9 példa

KOCSI	(***Rendszám***, ..., Kocsitípus)
CASCO	(***Cascoszám***, ..., Kocsitípus)

A redundancia nyilvánvaló, de nincs elméleti alapunk annak kiküszöbölésére, mert egyik egyedtípus sem tartalmazza a másik kulcsát. Íme, egy újabb csapda.

Az adatmodellezés a fentiek miatt „nem szereti” az egymással ***lineáris*** (1:1 fokú) viszonyban álló egyedtípusokat. Az ilyen esetekben az elmélet a viszony ***opcionalitásában*** találja meg a menedéket. Tegyük fel - bár tudjuk, hogy ez nem igaz -, hogy minden kocsira CASCO-t kell kötni. Vagyis az 1:1 fokú viszony kötelező. Ebben az esetben semmi értelme sincs a két egyedtípus szétválasztásának, mert a CASCO a kocsi mindenkori jellemzője. Most éljünk a valós feltételezéssel, amely szerint nem minden kocsira kötnek önbiztosítást. Ezért a viszony a kocsi oldaláról nézve opcionális. Minden CASCO kocsira vonatkozik, de nem minden kocsinak van CASCO-ja. Ekkor a célszerű megoldás a következő:

17.10 példa

KOCSI	(***Rendszám***, ..., Kocsitípus)
CASCO	(***Cascoszám***, ..., *Rendszám*)

A két egyedtípus egymáshoz kapcsolható. A Kocsitípus redundanciája megszűnt, mert az a CASCO egyedben tranzitív függést mutat a Rendszám betétele után. Nyertünk egy jó modellt - és két tanulságot.

Az egyik az, hogy 1:1 fokú viszonyban álló egyedtípusokat a tervező sohasem alkalmaz. (Kivételt képeznek az egyedaltípusok, amelyekről majd a 18.5 alpontban lesz szó.) Ha az egyedek kapcsolata kötelező, akkor azokat összevonja és az eredetileg feltételezett kulcsokat alternálóaknak jelöli meg az elérhetőség érdekében. Ha a viszony opcionális, akkor mesterséges hierarchiát kreál, vagyis az egyik egyed kulcsát a másikban leíró tulajdonságként ismétli meg.

A másik tapasztalat generikusabb. Már a 17.6 példa kapcsán is figyelnünk kellett a függések opcinalitására. A CASCO-példa korábbi esetében is a függés ereje (ld. 13.3 példa) volt a döntő. Nem lehet úgy jól normalizálni, ha nem vagyunk tekintettel a ***függéserő*** tényezőre. Ez is egy titok, amelyről később még bővebben is beszélni fogunk.

17.8 Két összetett szerkezeti probléma

A jó tervező titka, hogy okosan tud kérdezni. Amint a fentiekben láttuk, a döntő kérdések többsége az azonosítókra vonatkozik. Mielőtt ezt a témakört bővebben is megvilágítanánk, néhány további titkos összefüggésről kell fellebbentenünk a fátylat.

A ***dekompozíciós*** normalizálási eljárás során mindig csak egy feltételezett egyedtípussal foglalkoztunk. A rossz belső szerkezetű egyedtípust a vonatkozó normálforma szabályai szerint kettőre vagy többre bontottuk. Ezt mindig veszteségmentesen tettük. Vagyis úgy, hogy az eredmény-egyedtípusok összekapcsolásával visszanyerhettük az eredeti egyed ismereteit. A 4NF és 5NF általunk nem kedvelt eseteit kivéve mindig egymással kapcsolatban álló egyedeket kaptunk úgy, hogy az egyeden belüli függésből egyed közötti függést generáltunk. Ha a RENDELÉS (***Rendelésszám***, Vevőkód, Vevőnév) egyedet a VEVŐ (***Vevőkód***, Vevőnév) és RENDELÉS (***Rendelésszám***, *Vevőkód*) párosra daraboljuk, akkor az eredeti egyeden belüli Rendelésszám → Vevőkód függést külsővé (is) tesszük és ezzel biztosítjuk a két egyedtípus közötti explicit kapcsolatot.

A ***szintézises*** normalizálási módszer kapcsán gondoltunk csak arra, hogy a szembetűnő kapcsolati hiányok (ld. 17.1 példa) miatt egyszerre két egyedtípussal is törődnünk kell. Azért, hogy meghatározhassuk azok helyes külső függési viszonyait és ezeken keresztül az egyedek kapcsolatait.

Azonban itt nem állhatunk meg. Miért csak két egyedtípus összefüggésével foglalkozunk? Miért nem az összesével? A kérdés jogosságát a 17.11 példa szemlélteti:

17.11 példa

VEVŐ	(***Vevőkód***, ..., Vevőnév)
SZERZŐDÉS	(***Szerződésszám***, ..., *Vevőkód*)
RENDELÉS	(***Rendelésszám***, ..., *Szerződésszám*, Vevőnév)

A vevők csak szerződéseiken keresztül kapcsolódnak a rendelésekhez. A példa külső szerkezete tökéletes. A rendelésekből elérhetjük a szerződéseket, azokon keresztül pedig a vevőket - és a fordított út is simán járható. Csakhogy nagyon is feltűnő a Vevőnév redundanciája. Mintha az egyedek belső szerkezetével volna most gond.

Ezt a problémát nem is olyan könnyű feloldani. A RENDELÉS nem tartalmazza a Vevőnév tulajdonságot ***közvetlenül*** meghatározó Vevőkódot. Mármost a tervező vagy észreveszi, hogy a Vevőnév ***közvetve*** függ a Szerződésszámtól, vagy sem. Ha igen, akkor dekompozíciót alkalmazva a következő eredményre fog jutni:

17.12 példa

VEVŐ	(***Vevőkód***, ..., Vevőnév)
SZERZŐDÉS	(***Szerződésszám***, ..., *Vevőkód*, Vevőnév)
RENDELÉS	(***Rendelésszám***, ..., *Szerződésszám*)

Ekkor a tervező ismét szembesül a tranzitivitás bajával. Ezért a Vevőnév tételt kiveszi a SZERZŐDÉS egyedből. Miután a VEVŐ egyedbe nem teheti, mert már ott szerepel, egyszerűen elhagyja azt a 17.13 példának megfelelően.

17.13 példa

VEVŐ	(***Vevőkód***, ..., Vevőnév)
SZERZŐDÉS	(***Szerződésszám***, ..., *Vevőkód*)
RENDELÉS	(***Rendelésszám***, ..., *Szerződésszám*)

Most persze az olvasó jogosan kérdezheti, hogy minek gyüszmékel ennyit a tervező, hiszen azonnal elhagyhatta volna a Vevőnév tételt a 17.11 példa eredeti RENDELÉS egyedéből. Tehette volna, ha okosan kérdez (ld. 17.6 pont) és azt a kérdést veti fel, hogy mi határozza meg közvetlenül a Vevőnév tulajdonságot. Ugyanis a tisztán leíró tulajdonságok általában csak egy egyedtípushoz kötődnek. Ezzel szemben az azonosítók más - több - egyedben is lehetnek meghatározottak, azaz kapcsolók. A 17.11 példa három mélységi szintű modelljével szemben elképzelhetők - léteznek - sokkal összetettebb szerkezetek is. Ezek esetében már nem olyan könnyű feltárni, hogy egy kapcsoló tulajdonság szükséges-e az egyedben, vagy tranzitív. Honnan tudjuk például, hogy az E1 (***A***, *B*, *X*), E2 (***B***, *C*), E3 (***C***, *N*) ... En (***N***, *X*), Ex (***X***, ...) egyedsorozatban az X az E1 egyedben tranzitív? No meg azt, hogy ezzel szemben az E1 (***A***, ...), E2 (***B***, *A*), E3 (***C***, *A*) és E4 (***D***, *B*, *C*) szerkezettel - és most tessék a négy egyedtípus között számos másikat is elképzelni - semmi baj sincs a kétszeresen meghatározott A tulajdonságtípussal?

Azután fellép egy további probléma is. Az előbbiekben a Vevőnév tulajdonságot a tranzitivitás miatt az egyedtípusok között ***vándoroltattuk*** (angolul: attribute-migration), vagyis előbb a RENDELÉS egyedből a SZERZŐDÉS-be, majd onnan a VEVŐ-be, végső helyére tettük. Ám a tulajdonság megfelelő egyedének a megtalálása néha nem könnyű feladat. A 17.14 példa nem kis fejtörést okozhat a tervezőnek:

17.14 példa

MEGYE	(***Megyekód***, ..., *Városkód*)
JÁRÁS	(***Járáskód***, ..., *Megyekód*)
VÁROS	(***Városkód***, ..., *Járáskód*, *X*)

Két probléma lép fel. Az első: Tessék beilleszteni ebbe az adatbázisba egy új megyét, járást vagy várost! Eláruljuk, hogy ez így nem fog sikerülni. A megye beillesztéséhez kell a Városkód értéke; ahhoz a Járáskód tartalma; emehhez pedig a Megyekód értéke. Egészen csinos összetett adatciklusra bukkantunk. A három azonosító körkörös meghatározottsága miatt aligha lehet eldönteni, hogy az X adat végül is melyik egyedet fogja jellemezni. Akármelyik egyedbe is tesszük, a tranzitivitás miatt azonnal az „eggyel feljebb” lévő egyedbe kell vándoroltatnunk. Onnan a következőbe - és így tovább a végtelenségig.

Nem véletlen, hogy a kölcsönös meghatározottság miatt a modellezéselmélet azonnal kizárja az ***egyszerű ciklust***. Vagyis az olyan szerkezetet, amelyben két egyedtípus egymás azonosítóját tartalmazza. A 17.15 példa esetében nehezen dönthető el, hogy melyik egyedbe tartozzék a Kocsitípus adat, hiszen az a kocsit jellemzi, de a CASCO díja is a típuson múlik (vö. 17.9 példa).

17.15 példa

KOCSI	(***Rendszám***, ..., *Cascoszám*, Kocsitípus)
CASCO	(***Cascoszám***, ..., *Rendszám*, Kocsitípus)

Erről a példáról azonnal látjuk, hogy rossz, mert a Kocsitípus redundáns leíró ismeret és a két kulcs kölcsönös szerepeltetése is redundancia. Viszont a 17.14 példa ***összetett ciklust*** tartalmaz, amely - más esetekben - nemcsak három, hanem akárhány egyedtípust is érinthetne. Ott már alig látszik, hogy a kulcsok kölcsönösen mindegyik egyedet jellemzik. Nehezebben

dönthető el, hogy hová is tartozik az X tulajdonság. Az egyszerű ciklusokat a mai adatelemző automaták felfedezik, az összetetteket nem. Az előbbiek esetében a megoldás logikus: valamelyik egyedből el kell hagyni a másik kulcsát, a másikból pedig a redundáns tulajdonságot. Például a KOCSI-ból a Cascoszámot, a CASCO-ból a Kocsitípust (a megoldást a 17.10 példa mutatja).

Viszont a 17.14 példa problémája szemantikai megoldást feltételez. A MEGYE egyedben a Városkód nem akármelyik település jele, hanem a megyeközponté. Ettől függetlenül a megye és a város kétszeresen kapcsolódik egymáshoz. Egyszer közvetlenül, egyszer a járáson keresztül. Az összetett ciklusok karbantartási anomáliákat okoznak és ezért ki kell küszöbölni azokat. A megváltozott Városkód értéket két helyen kell karbantartani.

Mindenképpen baj származik az összetett ciklusból akkor, ha valaki technikai segédeszközt használ a normalizáláshoz. A 17.11 példa összetett tranzitivitását mindenképpen meg kell szüntetni, mert a Vevőnév redundáns. Csakhogy az automata nem képes segíteni akkor, ha a feladat összetett ciklust (17.14 példa) tartalmaz. Körbe-körbe fogja vándoroltatni az X tulajdonságot. Mivel pedig automata nélkül nem vagyunk képesek a jó adatmodell kialakítására (vö. az előző fejezet utolsó részének felsorolásával), az összetett ciklusokat ki kell irtanunk az adatmodellből.

Ezt már csak azért is meg kell tennünk, mert az automata által felfedezett összetett ciklusokról általában az derül ki, hogy szemantikai tisztázást igénylő homonimákat rejtenek. Vagyis az $A \rightarrow B$, $B \rightarrow C$ és $C \rightarrow A$ függéshármasban - ciklus - a két A, a két B vagy a két C valamelyike valójában nem ugyanazt jelenti.

17.9 Az adatmodell „fonalai”

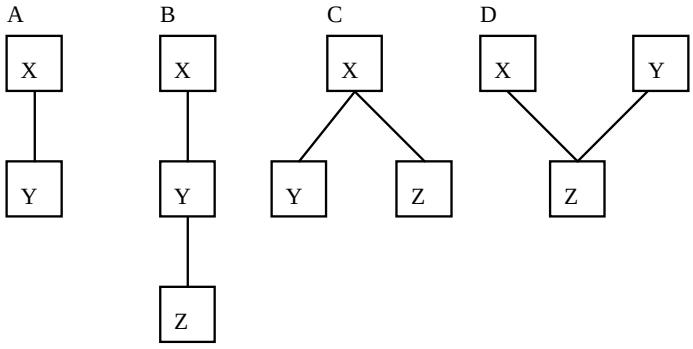
Mint már tudjuk, ha az Y egyedtípus tartalmazza a másik X egyedtípus azonosítóját, akkor az előbbi az utóbbi fölérendeltnek az alárendeltje, vagyis létezik az X-től az Y felé mutató kapcsolat. Megfordítva: Ha az X az Y fölérendeltje, akkor az utóbbit jellemeznie kell az előbbi azonosítójának. Ebből következően ha létezik a Q - W kapcsolat, akkor fenn kell állnia a $W \rightarrow Q$ azonosítók közötti függésnek. (N.B.: Az egyszerűség kedvéért az egyedtípust és a vele 1:1 viszonyban álló elsődleges kulcsot azonos módon jelöljük.)

Az adatmodell egyedtípusai összetett ciklusmentes (ld. előző pont) struktúrát alkotnak. Mivel a ciklus kizárt, az adatmodellnek van kezdőpontja és végpontja. ***Kezdőpontnak*** azt az egyedtípust tekintjük, amelynek azonosítóját más nem határozza meg, következésképpen nincs már alárendeltje. ***Végpontnak*** azt az egyedtípust nevezzük, amelynek kulcsától nem függ másik, tehát nincs már fölérendeltje. Természetesen egy összetett hálós szerkezetű adatmodellben több kezdő és végpont is szerepelhet.

Ha a kapcsolatokra figyelünk, akkor a kezdőpont a modell legalsó, a végpont a legfelső egyedtípusa. Mi azonban most a függéseket akarjuk vizsgálni, amelyek sorozata a kapcsolatinak pont fordítottja. Ha létezik az X - Y és Y - Z kapcsolatsor, akkor fennáll a $Z \rightarrow Y$ és az $Y \rightarrow X$ függéssorozat, amit röviden $Z \rightarrow Y \rightarrow X$ módon jelölünk.

Most nézzük meg, hogy milyen szerkezeti részegységek fordulhatnak elő egy általános adatmodellben. Eközben egyelőre feledkezzünk meg a leíró tulajdonságokról és csak az egyedeket helyettesítő azonosítókkal foglalkozzunk.

Mint tudjuk, az az ismeretegyüttes, amely csak egyetlen egyedtípust ölel fel, nem valódi adatbázis, mert az minimum két kapcsolt egyedtípust feltételez. Viszont a 17.3 példa - bár hiányzik belőle a kapcsolat megadása - már egy miniadatbázist mutat. Az adatmodell legkisebb részegysége az ***adategyüttest***, amely nem más, mint két egymással kapcsolt egyedtípus. Ebben a struktúrában az egyik egyedtípus (VEVŐ) a másiknak (RENDELÉS) fölérendeltje azáltal, hogy az utóbbi tartalmazza az előbbi azonosítóját (Vevőkód). Ld. a 17.1 sematikus ábra „A” részletét.



A - adategyüttes	függési képlete:	$Y \rightarrow X$		
B - lineáris szerkezet		$Z \rightarrow Y$	és	$Y \rightarrow X$
C - hierarchikus szerkezet		$Y \rightarrow X$	és	$Z \rightarrow X$
D - hálós szerkezet		$Z \rightarrow X$	és	$Z \rightarrow Y$

17.1 ábra: A modellstruktúra tipikus szerkezeti részegységei

Az ábrában dobozok mutatják az egyedtípusokat. Vonalak reprezentálják a kapcsolattípusokat. Mivel minden kapcsolat lefelé mutat és az 1:1 fokú kapcsolatokat kizártuk, nem rajzoltuk be az 1:N fokot mutató varjúlábakat. A tipikus szerkezeti részegységek tipikus függéssorozatoknak felelnek meg. Ezek képleteit is mutatja az ábra.

A VEVŐ és a RENDELÉS a 17.3 példa esetében egyszerű adategyüttest alkot. Ámde az ugyanilyen nevű két egyed viszonya a 17.11 példa esetében közvetett, mert egy harmadik egyeden (SZERZŐDÉS) keresztül valósul meg. Ezért a RENDELÉS már nem tartalmazza a Vevőkódot. Ettől függetlenül világos, hogy a VEVŐ a RENDELÉS (közvetett) fölérendeltje. A VEVŐ, a SZERZŐDÉS és a RENDELÉS ún. ***lineáris*** szerkezetet alkot. Ld. a 17.1 ábra „B" részletét. N.B.: Amikor itt szerkezetről beszélünk, az adatmodell felépítéséről, nem pedig az adatbázis struktúrájáról van szó. Az egyedelőfordulások (az alkalmazási adatbázis) szintjén a lineáris szerkezet valójában többszintes hierarchiát (fát) takar.

Most nézzük meg a 17.16 példát:

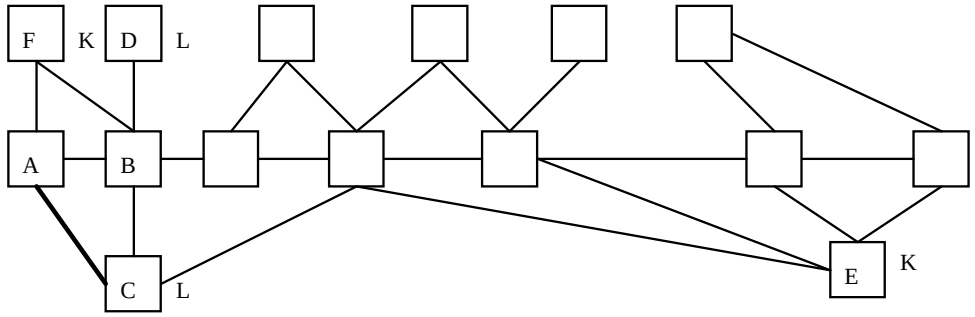
17.16 példa

SZEMÉLY	(***Törzsszám***, ...)
NYELVTUDÁS	(***Nyelvkód***, ..., *Törzsszám*)
VÉGZETTSÉG	(***Végzettségkód***, ..., *Törzsszám*)

A két utóbbi egyedtípus a SZEMÉLY alárendeltje. Fennáll a Nyelvkód $\rightarrow$ Törzsszám és a Végzettségkód $\rightarrow$ Törzsszám függés. Ám ez a kettő nem épül egymásra, mivel a nyelvtudás és a végzettség egymástól független dolgok. A két függés nem alkot egyetlen logikus láncot, mint az előző példában, hanem alulról felfelé egy csomópontba mutat. Az ilyen struktúrát nevezzük ***hierarchikus*** szerkezetnek. Lásd a 17.1 ábra „C" részletét.

Végül a 17.7 példa esetében a RENDELÉS és a CIKK ***hálós*** viszonyban állt. A RENDELÉS-TÉTEL a másik két egyed közös alárendeltje, amely az alkalmazási adatbázis szintjén megteremti a másik kettő M:N fokú viszonyát. Érvényes a Rendelésszám+Cikkszám $\rightarrow$ Rendelésszám és a Rendelésszám+Cikkszám $\rightarrow$ Cikkszám függés. Ezért ebben az esetben egy alárendelt mutat két fölérendelt felé. Ld. a 17.1 ábra „D" részletét.

A 17.2 ábra mutat egy sematikus adatmodellt. Amint látható, a modell a megismert egységekből épül fel, meglehetősen összetett módon. A vízszintes vonalak balról-jobbra mutató kapcsolatokat és ellentétes irányú függéseket jeleznek. Tehát például a B egyed az A tétel alárendeltje, vagyis tartalmazza annak kulcsát. Az A - B - C hármas valójában egy lineáris részszerkezet, csak az az elrendezés miatt nem úgy látszik.



17.2 ábra: Egy viszonylag egyszerű adatmodell váza

Most már könnyen belátható, hogy végeredményben az adatmodell teljes kapcsolatrendszere, vagyis az egyedek és azok viszonyai, leírható egy funkcionális függési hálóval. Így az adatmodell optimalizálása - ha a leíróktól eltekintünk - nem jelent mást, mint ennek a hálónak az átalakítását. A modellnek ciklusmentesnek, konnektívnek (megszakítás nélkülinek) és a tranzitív útvonalaktól megtisztítottnak kell lennie. Mivel még egy ilyen egyszerű modellben is számos függés létezik, a kérdés az, hogy miképpen lehet optimalizálni a 17.2 ábra által mutatott hálónak megfelelő jellegű modelleket?

Induljunk ki egy egyszerű részletből. A C egyedtípus tartalmazza az A kulcsát. Feltételezve azt, hogy a C $\rightarrow$ A függés erős, ez a viszony a B-n át tranzitív és azt meg kell szüntetni. Vagyis modellünkből törölni kell a vastag vonallal jelzett kapcsolatot. (N.B.: Ha a függés gyenge, akkor nem valódi a tranzitivitás és nincs teendő.)

Ezt a tranzitív viszonyt könnyű volt átlátni, mert a három egyed egymás mellett található. Azonban tegyük fel, hogy az E egyed tartalmazza az F kulcsát. Ez a két tétel egymástól már eléggé távol van. Miként fedezzük fel azt, hogy az E $\rightarrow$ F függés tranzitív?

A hagyományos normalizálás a tulajdonság vándoroltatásával történik. Például az L leíró a C egyedben a B-n keresztül tranzitív. Tehát migráltatjuk a B egyedbe. Ott még mindig tranzitív, tehát átvisszük a D egyedbe. Pontosabban: mivel már ott van egyszerűen elhagyjuk. Ugyanez a vándoroltatás a K leíró esetében az E egyedtől az F-ig már sokkal több lépést igényelne.

A hagyományos - lépésenkénti - normalizálás problémái többrétűek. Ezeket a következő tételekben lehet összefoglalni:

- A tulajdonságnak a közvetlen fölérendeltbe történő áthelyezése nem jó megoldás, mert adott esetben igen sok lépést - többszörös vándoroltatást - igényelhet.
- A C $\rightarrow$ L és D $\rightarrow$ L függések közvetlenül látszanak. Viszont a B $\rightarrow$ L függés nem explicit. Ahhoz, hogy a normalizálás sikeres legyen, a tervezőnek fel kell fedeznie azt, hogy ez a függés létezik a C egyeden belül. Ám mi van akkor, ha ezt nem ismeri fel?
- Egy egyednek számtalan fölérendeltje lehet. Azoknak ismét. Lásd az E egyedtípust. Ezért sokszor előfordul, hogy egy tulajdonságot az eredeti egyed több közvetlen, majd közvetett fölérendeltjébe is át kell vinni illetve később onnan továbbmozgatni.
- A hagyományos normalizálás bármelyik egyedtípusnál elkezdődhet. Például a B-nél. Ha később kerül sor a C normalizálására, akkor a B-t majd még egyszer elemezni kell akkor, amikor a C-ből oda vándoroltatjuk az L leíró tulajdonságot.

Amint látjuk, ez így nagyon terhes, nem-hatékony normalizálási eljárás. Főleg az ad-hoc kezdés okoz gondokat az egyedek többszörös normalizálási igénye miatt. E problémák elkerülésére sokan ***függési mátrixokat*** alkalmaznak. Így működött első normalizátorunk, a SZIAM is. A kiinduló függési mátrix csak a közvetlen függéseket tartalmazza. A függések egymásra vetítésével a mátrixot kiegészítjük a közvetett függésekkel is. Például a C → B és B → D függésekből adódik a C → D közvetett függés. Mivel így az már explicitté vált, az L leíró tranzitivitása egy pillanat alatt megszüntethető.

Így szólt az elmélet. A gyakorlat azonban mást mutatott. Egyrészt hatalmas méretű mátrixokat kellett felépíteni. Gondoljunk csak arra, hogy 500 tulajdonság esetén mekkora a táblázat mérete! Másrészt a mátrix nem működött ciklus esetén és az algoritmus nem tudott különbséget tenni a ciklus és a tranzitivitás között. Harmadrészt - erről még nem volt szó - a függéseket többszörösen is minősíteni kellene (ezt nem tették meg), ami még inkább növeli a méretet. Meg kellene adni a függés opcionalitását (erős vagy gyenge) és minőségét (részleges, tranzitív stb.) is. Azután van egy negyedik baj is: ilyen alapon igen nehézkesen kezelhetők az összetett függések. Például a meghatározott nem lehetett összetett.

A fenti problémák miatt új normalizálási eljárást - algoritmust - kellett találni. Ehhez el kellett vetni az ad-hoc kezdést involváló dekompozíció normalizálási alapját (az egyedtípusok halmaza) és a hatalmas méretekbe torkolló univerzális reláció illetve szintézis normalizálási inputját (a tulajdonságtípusok halmaza). Új normalizálási bemenetre volt szükség. Ezt szerencsénk volt megtalálni a ***fonál*** (angolul: thread) egységében [31].

D 17/1 **A fonál az adatmodell hálójának a kezdőpontjától a végpontjáig vezető függések sorozata.**

Természetesen az adatmodellnek több kezdő- és végpontja is lehet. Ezért a struktúrát több fonál alkothatja úgy, hogy azoknak vannak közös tényezői. Például a 17.2 ábrában fonál a C → A → F, a C → B → A → F, a C → B → D függéssorozat stb. Vegyük észre, hogy a fonál nem más, mint az adatmodell egy lineáris szerkezeti egysége. Ha tehát az adatmodellt fonalakra bontjuk, akkor valójában szétbogozzuk ezeket a korábban egymással összeszövődött lineáris struktúrákat.

A kérdés az, hogy mire jó ez a varázslat? Most eláruljuk a szerkezetek egyik titkát. Nincs olyan fonál a 17.2 ábrában, amely együtt tartalmazná a C és az E egyedtípust. A két egyed azonosítója között nincs sem közvetett, sem közvetlen függés. Ebből szükségszerűen következik, hogy a két egyed együtt nem okozhat tranzitivitást vagy hasonló problémát. Így teljesen felesleges azokat együtt vizsgálni. A párhuzamos szálakon nem fordulhatnak elő modellezési hiányosságok.

Innen következik a fonalak második, kettős titka. Az egy láncon lévő tulajdonságok a teljes fonalban kétszer szerepelhetnek, ha ***azonosítók*** és nem a fonal valamelyik végén találhatók. A C → B → D függéssor valójában a C → B és B → D páros rövidítése. A B nincs a fonal végén, ezért egyszer lehet jobboldalt (meghatározott), egyszer baloldalt (meghatározó). Viszont a ***leíró*** tulajdonságok a láncban csak egyszer szerepelhetnek. Ha tehát egy leíró a szálon többször is előfordul, akkor normalizálása egy pillanat műve: Abban az egyedben maradhat csak meg, amelyik a tartalmazók közül legközelebb áll a fonal végpontjához. Az L leíró a C-t és a D-t is jellemzi. Az utóbbi van a végponthoz közelebb - maga a végpont - és így az L leíró a C-ben tranzitív.

A harmadik titok a kulcsokra vonatkozik. Ha egy kulcspáros két fonálon is megtalálható úgy, hogy az egyik az egyiken közvetlenül, a másikon közvetve határozza meg a másikat, akkor az a másik egyed az első fonálon tranzitív. A C és az A tulajdonság együtt szerepel például a C → A → E és a C → B → A → E láncokon. Az előbbiben a C → A meghatározás közvetlen. Tehát ez a függés a rossz és az A kulcsát ki kell venni a C-ből, azaz meg kell szüntetni az A - C kapcsolatot. Mármost ha a C és az A között nincs sehol sem közvetlen függés, viszont létezne

több közvetett, akkor két eset lehetséges. Vagy a szerkezet úgy jó, ahogy van. Vagy - no de ez már olyan igazi titok, amit egyelőre nem fogunk elárulni. Ezt a csomót bogozza ki, aki tudja.

Most pedig foglaljuk össze a fonalak előnyeit:

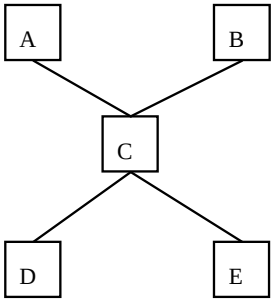
- A helytelen függések felfedezéséhez nincs szükség vándoroltatásra.
- A fonalak egycsapásra explicitté teszik az implicit függéseket. Ezért a leírók normalizálása egy pillanat alatt megtörténhet az adott láncon. Maguk a helytelen fonalrészek automatikusan kiszűrhetők.
- Mivel a normalizálás a fonal kezdőpontjánál indul, nem pedig ad-hoc módon, minden egyed elemzésére csak egyszer van szükség. (Ez így nem teljesen igaz. A lényeges az, hogy a hozzányúlások száma a minimálisra korlátozott.)
- Szemben a mátrixon alapuló elemzéssel, a mi algoritmusunkban lényegesen kisebb technikai apparátus szükségeltetik. A fonalak együttes mérete nagyságrendekkel kisebb, mint az ugyanazon adatmodellt tükröző függésmátrixé.

17.10 Apró szerkezeti titkok

A fonalak mindig feltételezik a felsőbb szintű egyed azonosító szerepű tulajdonságának a megismétlését a közvetlen alsóbb szintű egyedekben a kapcsolhatóság kedvéért (ez „jó” redundancia). Mindig kizárják a felsőbb szintű egyed leíró szerepű tulajdonságának a megismétlését bármelyik alsóbb szintű, a fonalon fekvő egyedtípus esetében, mert az felesleges ismeretismétléshez vezet (ez „rossz” redundancia). Ezért a fonalak mentén történő normalizálással kiszűrhető a legtöbb adatmodellhiba. Egy tanulmányban kimutattuk, hogy elméletileg a problémák 87-88 százaléka számolható így fel.

Azért csak a „legtöbb”, mert egy adatmodell nemcsak a redundanciák miatt, hanem a teljesség hiánya következtében is hibás lehet. Magyarul: a ***felesleges*** függések automatikusan megszüntethetők, ám a ***hiányzóak*** mechanikus feltárására nincsen mód. Ehhez kiegészítő emberi tevékenységre van szükség. (Az eddig ismertetett normalizálást mi természetesen automatával végezzük.) A kérdés az, hogy hol és miként avatkozzon be az ember?

Az adatmodell az ismeretek „szövete”, tehát abban párhuzamos fonalak is előfordulnak. Baj csak ezeken az ágakon léphet fel. Mégpedig ott is csak addig, ameddig a szálak össze nem futnak vagy szét nem válnak. A 17.3 ábrán lévő modellben négy fonal van. Rövidített írásmódban: D-C-A, D-C-B, E-C-A és E-C-B. A C egyed a csomópont. Ez lehet egy teljes, több egyedtípusból álló lineáris szerkezet - részfonal - is. Ez mondanivalónk lényegén nem változtat. Ezért most tételezzük fel, hogy a C csak egyetlen egyed.



17.3 ábra: Szerkezeti elágazások

Az ábra öt egyede között elvileg 5*4/2 = 10 összefüggést kellene vizsgálni, ha nem lennének fonalaink. Mivel azonban a láncok összefutnak, csak a csomópont alatti és fölötti helyzetre kell ügyelnünk. Vagyis csak az A és B illetve a D és E párosát kell elemeznünk.

A 17.3 ábra által mutatott elágazások esetében több kérdést kell feltennünk. Hiszen mindig figyelünk a tulajdonságok azonosító és leíró szerepére.

Tegyük fel, hogy a D és az E (vagy az A és a B - a többi párosítás érdektelen) egyedben szerepel egy közös X ***leíró*** tulajdonság. Ha annak abszolút szerepe azonosító - vagyis létezik X egyed is -, akkor a helyzettel máris nem kell foglalkoznunk. A leíró valójában kapcsoló, ezért a két egyed redundanciájával nem kell törődnünk. (Aktuálisan mindkét egyed tartalmazza a C kulcsát, ami teljesen normális szituáció.)

A D és az E egyed - vagy bármilyen párhuzamos ágon lévő egyedpáros - leíró tulajdonságának az átfedésére csak akkor kell figyelnünk, ha az abszolút szerep is leíró. Vagyis nincs X által azonosított egyed. Ilyenkor három eset lehetséges:

- Az X valójában azonosító, de ezt eddig nem fedeztük fel. Ezt a szituációt elméletileg nem lehet kizárni, de gyakorlati valószínűsége szinte nulla.
- Az X homonima, mást jelent a két egyedben. Nagyon gyakran találkozunk ezzel a helyzettel. Ilyenkor új nevek alkalmazásával egyértelművé kell tenni a modellt.
- Az X valamilyen tipikus, általánosan használt leíró tulajdonság. Ezért természetszerűen több egyedet is jellemez. Például valamilyen dátum, mennyiség stb.

Nincsen olyan szigorú modellezési szabály, amely tiltaná az ilyen többszörös leírókat. Azonban az okos tervező igyekszik elkerülni az efféle helyzeteket és minősített neveket alkalmaz. Már csak azért is megteszi ezt, mert előre nem tudhatja, hogy párhuzamos ágakon vagy egy fonalon szerepelnek-e majd az egyedek. Ha két egyedben is alkalmazza pl. a Név tulajdonságnevet és az egyedek egy fonalon vannak, akkor bizony az automata normalizátor kiabálni fog.

Itt ismét elárulunk egy apró titkot. Egy valamennyire is összetett adatmodellt kézzel nem lehet normalizálni: automatára van szükség. Az automata nevekkel dolgozik. Ha két leíró tulajdonság neve azonos, akkor azokat azonosnak fogja feltételezni és - talán feleslegesen - mindenféle normálforma hibákat fog jelezni. Ha viszont egy azonosító más néven szerepel kapcsolóként további egyedtípusokban, akkor az automata nem fedezi fel az azonosságot. Nem tudja a két egyedtípus összekapcsolni. Nem képes jó fonalakat alkotni. Így nem is fogja jelezni a valóban létező normálforma hibákat. Ezért a modellezés alapszabálya: a leírók nevét kell (abból baj sohasem lehet), az azonosítók nevét nem szabad ***minősíteni***.

17.11 A kulcsmátrix

Az adatbázistervezés egyik csapdája az volt, hogy a dekompozíció módszerével elméletileg nem lehet teljes adatmodellt készíteni, a szintézis eljárása pedig a vizsgálatok nagy száma miatt gyakorlatilag nem hajtható végre. Ezért új normalizálási metodikát fogunk javasolni, amelyet természete szerint ***heurisztikusnak*** lehet nevezni.

A módszer a fentebb már elhangzott feltételezésen alapul: Annak valószínűsége, hogy nem látjuk meg egy tulajdonságtípus abszolút azonosító szerepét, a nullával egyenlő. Ez ugyanis azt jelentené, hogy nem találtunk olyan másik tulajdonságot, amelyet ő funkcionálisan meghatároz. Ezért a normalizálás során a teljességre való törekvés közben nyugodtan elfeledkezhetünk a leíró tulajdonságokról. Viszont az azonosítók közötti függéseket alaposan meg kell vizsgálnunk.

A valódi probléma nem az, hogy a 17.3 ábra D és E egyedtípusában van-e egy X leíró, amiről nem fedeztük fel, hogy azonosító. A bajok akkor kezdődnek, ha a valóságban a D és az E egyed egymáshoz kapcsolódik, de ez nem tükröződik a két egyed tulajdonságaiban. Ha például a D az E fölérendeltje, akkor az E-ben szerepelnie kellene a D kulcsának. Ha ott nem található, akkor a modell kapcsolathiányos (ld. 17.2 pont).

	D	E	C	A	B
D	-	?	*	*	*
E		-	*	*	*
C			-	*	*
A				-	?
D					-

17.4 ábra: Kulcsmátrix

A kapcsolati hiányt az ún. ***kulcsmátrix*** segítségével tárhatjuk fel. A kulcsmátrix is egy függési mátrix (ld. 17.9 pont), de csakis az azonosító tulajdonságok függéseit tartalmazza. Így mérete (több) nagyságrenddel kisebb, mint a teljes függési mátrixé. Vagyis kezelhető. Ez a mátrix olyan trianguláris táblázat, amelybe nemcsak a közvetlen, hanem a közvetett kulcsfüggéseket is bejegyezzük. A 17.3 ábrára vonatkozó mátrixot a 17.4 ábra mutatja.

Ha a táblázatot okosan töltjük ki - elölre helyezzük a kezdőpontokat -, akkor világos képet kapunk. Amint látjuk, a tízféle kombinációból csak kettő maradt kérdéses. Hasonló nagyságrendekkel kell számolni egy „valódi" adatmodell esetében is. Vagyis átlagosan az egyedek kapcsolatainak az ötöde marad nyitott kérdés. 1000 egyednél 200? Ne tessék megijedni. Egyrészt ez nem egy nagy szám - pedig az 1000 egyed már egy meglehetősen összetett modellt jelent. Másrészt egy kérdőjel kitöltése - a közvetett függések miatt - számos más kérdést is megszüntet, hiszen új fonalak alakulnak ki.

A fentiekből már látszik, hogy a kulcsmátrix alkalmazásával párosuló ***fonalnormalizálás*** lényegesen hatékonyabb eljárás, mint a dekompozíció, az univerzális reláció vagy a szintézis által alkalmazott normalizálási algoritmus. Sőt, nemcsak hatékonyabb, hanem garantáltan jobb adatmodellre is vezet. Főleg akkor, ha elárulunk még néhány apróságot.

Tegyük fel, hogy az ugyanazt a leíró tulajdonságot tartalmazó két egyedtípus kulcsa között nincs funkcionális függés, noha azok elvileg kapcsolatban állnak. Magyarul: nem adták meg például a D egyedben az E egyed kulcsát. Ha ekkor belevágunk a normalizálásba, akkor nem derül ki a leíró tulajdonság tranzitív függése. Emiatt az okos tervező a normalizálás megkezdése előtt két dolgot csinál. Először is készít egy ***adatmodell-diagramot*** az egyedek és feltételezett kapcsolataik reprezentálására. Ezt összeveti az egyedek belső szerkezetével. Azaz megnézi, hogy a diagram szerint alárendelt egyedek tartalmazzák-e a fölérendeltek kulcsát. Másodszor elkészíti a kulcsmátrixot és elemzi azt. Csak az esetleges hiányok pótlása után fog a normalizálásba.

Az okos tervezőnek mindig két - persze egymásnak megfelelő - adatmodellje van. Egy tágabb és egy szűkebb. A kettőben a nevek, a struktúrák - szóval minden - egyezik. A ***tág*** adatmodell teljes: minden egyed-, tulajdonság- és kapcsolattípust tartalmaz. A ***szűk*** modell az előző alhalmaza. A „trükk" pedig a következő:

A tapasztalt tervező tud kérdezni (ld. 17.6 pont). Nem azt veti fel, hogy meghatározza-e a Vevőkód a Vevőnév tételt. Hanem azt, hogy mitől függ közvetlenül az utóbbi tulajdonság.

Miután elkönyvelte, hogy a Vevőkódtól és semmi mástól, a Vevőnévről már el is feledkezhet. Azt egészen egyszerűen nem kell normalizálni, mert nincs mit szegényen. Ezért a szűkebb - normalizálandó - modellváltozatban ez a tétel már nem jelenik meg.

Más. A modellben vannak „holtbiztos" egyedtípusok. Ilyenek az esetleg több másikhoz is kapcsolódó segédegyedek. Kód és név párosok, például: VÁLTOZÁS (***Változáskód***, Változás-

név). Standard listák a mértékekről, országokról, valutákról, típusokról stb. Ha ezek a fonalak végpontjai, vagyis nem mutatnak további egyedek felé, akkor egyszerűen nem okozhatnak normalizálási hibákat. Kivehetők a szűkebb modellből. Persze az ilyen „gyorsnormalizálást” csak a tapasztalt és fegyelmezett tervezőknek ajánlhatjuk. Akik nem követnek el szemantikai bakikat és figyeltek arra, hogy a „holtbiztos” egyedtípusok minden leíró tulajdonsága csakis abban az egy egyedben szerepel. Ha valaki a VÁLTOZÁS (***Változáskód***, Megnevezés) egyedtípust fedezné fel, azt óva intjük a gyorsnormalizálástól (no meg egyáltalán az adatbázistervezéstől). Ha egy cégnél többféle mértékegység-értékkészletet használnak - bizonyos összefüggésekben ilyet, másokban olyat -, akkor a MÉRTÉKEGYSÉG egyedet nem szabad kivenni a szűkített modellből, mert lesz meglepetés.

Ezzel fejezetünk végéhez érkeztünk. A tulajdonképpeni normalizálást most elhagyjuk és egy picit viharosabb vizekre evezünk.

18. SAJÁTOS SZERKEZETI TÉNYEZŐK

18.1 Minőségi adatmodellezés

A mai adatbáziskezelő rendszerek többsége COBOL-szinten szemléli az adatbázist. Sőt, még azon sem, hiszen például a csoportot még kevés kezelő ismeri el strukturális tényezőként. A kezelők számára a primitívségig egyszerű rekordképeket kell megadni adatbázisterv címén. Az adatmodellező rendszerek képességei sem sokkal ragyogóbbak, mert hiszen az x relációs rendszert támogató „CASE" valódi feladata az, hogy a tényleges kezelő kalitkájába zárja a fejlesztőt, kicsit kékre festve az amúgy igen szürke eget. Ezek után nem csoda, hogy a tervezők lapos, unalmas, a valóságra csak távolról emlékeztető, ellentmondásos, hiányos és redundáns adatbázisra vezető „modellt" alkotnak.

Az eszközök hiánya és a meglévő rendszerek korlátai miatt a fejlesztő számára minden rekordtípus, reláció, szegmens vagy más logikai adatkezelési egység azonosnak tűnik. Nem is foglalkozik a különbségekkel. Mert hiába látja, hogy itt és most - mondjuk - egy családfa szerkezetről van szó, ha ezt sem a modellező-, sem a kezelőrendszernek nem tudja megsúgni. A végén a családfa egyed is csak olyan reláció lesz, mint a többi. Akkor pedig minek fáradjon az elemzéssel, hiszen amúgy sincs elég ideje. Ezért a tervezők többsége a tervezőrendszerekkel (CASE) teljes összhangban megáll a legdurvább tervezési hibák kiszűrésénél és jó, ha egyáltalán törődik a 3NF normálalakkal.

Pedig az adatmodellezés legfontosabb feladata a valóság hű másának a megkeresése. Nem a lapos rekordképek leírása, hanem egy szemantikailag roppant gazdag modell összeállítása. Az adatmodellben kell, hogy tükröződjék minden az ismeretekkel kapcsolatos ismeret. A szoba a lakás része - de ezt nem lehet az adatkezelővel tudatni. Pedig ennek a viszonynak nyilván vannak adatkezelési következményei is. Az egyik út a másikhoz vezet, ami teljesen más jellegű („családfa") viszony, mint a rész-egész kapcsolat - de ezt sem lehet az adatkezelővel megértetni.

Remélhetőleg az olvasó már látja, hogy az adatmodellezésnek nem csak annyi a célja, hogy egy kezelhető adatbázisszerkezetet építsünk fel. Az csak a minimális cél. A valóság megértése, átlátása, korlátainak mérlegelése, sokszínűségének feltárása - ez az adatmodellezés igazi célja.

Ebben a fejezetben az adatmodell egyelőre még különlegesnek tartott sajátos szerkezeti és minősítési aspektusait fogjuk ismertetni.

Az első pontokban rá fogunk mutatni arra, hogy a speciális szemantikai tartalmat hordozó struktúrák is visszavezethetők a funkcionális függések mechanikus alapjára. Ezért támogatásuk hiánya a mai kezelőkben érthetetlen. Az utolsó pontokban az adatmodellnek a megszokottnál szélesebb értelmezésére és az adatmodellt alkotó tényezők osztályozásának a fontosságára kívánjuk irányítani a figyelmet.

18.2 Többszörös inhomogén kapcsolatok

A gyakorlatban néha előfordul, hogy két egyedtípus között nem egy, hanem több kapcsolattípust kell meghatározni. Tegyük fel, hogy valamilyen okból a magánszemélyek kétféle címét kell vezetnünk. Mondjuk a lakcímet és a levelezési címet, amely nem mindig azonos az előbbivel. Tételezzük fel továbbá, hogy egy általános címnyilvántartást tartunk, amelyben minden cím a Címkód által egyértelműen azonosított. A helyzetet első megközelítésben a 18.1 példa mutatja:

18.1 példa

CÍM (***Címkód***, ...)
SZEMÉLY (***Személy-azonosító***, *Címkód*, *Címkód***,** ...)

Szándékosan voltunk pongyolák a SZEMÉLY egyed esetében, hiszen evidens, hogy két azonos nevű tulajdonság nem szerepelhet egy egyedben. A példa kétféle megoldási módjának a megvilágítására ad alkalmat ez a pontatlanság.

Az egyik tervező a következő módon gondolkozik: A lakcím és a levelezési cím azonos is lehet adott esetben. Ezért a SZEMÉLY egyedben lévő Címkódot ***ismétlődő csoportként*** kell felfognunk. A Címkód független az egyed kulcsától, ezért azt külön egyedtípusba kell levágni az eredeti azonosító megismétlésével a 18.2 példának megfelelő módon:

18.2 példa

CÍM (***Címkód***, ...)
SZEMÉLY (***Személy-azonosító***, ...)
SZEMÉLY/CÍM (***Személy-azonosító+Címkód***)

A SZEMÉLY/CÍM egyed megteremti a másik kettő közötti valódi hálós viszonyt. Igaz, hogy ennek az egyednek nincs más feladata, mint a kapcsolás, de majd mindjárt találunk neki. Mert a 18.4 példa nem tükrözi, hogy a személynek melyik címéről van szó. Ezért a végső megoldás a következő a 18.3 példa szerinti:

18.3 példa

CÍM (***Címkód***, ...)
SZEMÉLY (***Személy-azonosító***, ...)
SZEMÉLY/CÍM (***Személy-azonosító+Címkód***, Címtípus)

A Címtípus utal a lakcím/levelezési cím jellegre. A megoldás jó, mert nem foglal külön helyet a levelezési cím, ha az azonos a lakcímmel és ráadásul tetszőleges számú címet lehet egy személyhez kötni. Ezért nem jelent gondot a bővítés sem. Igaz, a CÍM már nem érhető el közvetlenül a SZEMÉLY-ből és be kellett vezetni egy új adatot (Címtípus) is.

A másik tervező gondolatmenete a következő: Az egyedben nem lehet két azonos tulajdonság (Címkód). Ám erről szó sincs az adott esetben. A lakcím és a levelezési cím eltérő szemantikai lényegek. Az ugyanabból az értékhalmazból (Címkód) származó, de eltérő szemantikai jelentésű attribútumokat ***minősítő szerepnevekkel*** különböztetjük meg. Ezért rossz a 18.1 példa és azt valójában a 18.4 példa módján kell megfogni:

18.4 példa

CÍM (***Címkód***, ...)
SZEMÉLY (***Személy-azonosító***, *Lak-címkód*, *Levelezési-címkód***,** ...)

Ezzel a megoldással pedig nincs további teendő. Kapcsolattípus nemcsak akkor létezik két egyedtípus között, ha az egyik tartalmazza a másik azonosítóját, hanem akkor is, ha az egyikben a másik kulcsának a szerepneve található. Ilyenformán még az is előfordulhat, hogy egy egyedtípus a másikkal többszörös kapcsolatban áll.

D 18/1 **Két egyedtípus akkor és csak akkor áll többszörös kapcsolatban, ha az egyik a másik azonosítójának több szerepnevét tartalmazza.**

Akkor és csakis akkor, ha a többszörözés mértéke az idő során várhatóan nem változik, a szemantikai tartalom szerint minősített többszörös kapcsolat kedvezőbb megoldás, mint az úgymond ismétlődés megszüntetése. Ha egy személyhez több cím nem fog tartozni, akkor a 18.4 példa megoldása jobb a 18.3 példáénál. Azért, mert nem igényel új tulajdonságot (Címtípust) - a minősítés nem új tétellel, hanem szerepnévvel történik - és kevesebb elérést involvál.

Persze az adatmodellező nem látnok; a jövőt nehezen tudja megjósolni. Ezért a többszörös inhomogén - két különböző egyedtípus közötti - kapcsolat viszonylag ritkán használt megoldás. Mégis több okunk volt arra, hogy bemutassuk.

A 18.4 pontban lesz szó a homogén többszörös kapcsolatról, amely viszont igen gyakori szerkezet. Ezért nem ártott előre ismertetni a többszörösség lehetőségét.

A 17.10 pontot azzal zártuk, hogy az azonosító abszolút szerepű tulajdonságokat nem illik minősíteni. Félreértés ne essék, a 18.4 példa esetében a Lak-címkód és a Levelezési-címkód nem azonosító abszolút szerepű, mert nincsenek olyan egyedtípusok, amelyeknek kulcsai lennének. Viszont sok tervező alkalmazza a 18.5 példa szerinti megoldást:

18.5 példa

VEVŐ (***Vevőkód***, ...)
RENDELÉS (***Rendelésszám***, *Rendelés-vevőkód*, ...)

A RENDELÉS egyedben lévő Rendelés-vevőkód is kapcsolatot hordoz. Csakhogy azt sokkal nehézkesebb felfedezni, mintha a sima Vevőkódot használták volna. A szerepnév tudatosan meghatározott, a rendszer által ismert struktúra. A kezelő tudja, hogy a Lak-címkód a Címkód szerepneve. Ezzel szemben az egyszerű minősítésnél ez nem áll fenn. Ha a Rendelés-vevőkód nem szerepnév, akkor a minősítés zavaró. Ha pedig az, akkor felesleges. A minősített névnek nincs feladata és hiábavalóan terheli a rendszert.

Végül előfordulnak esetek, amikor a többszörös kapcsolat elkerülhetetlen. A jó modellezők gyakran alkalmaznak általános célú egyedtípusokat. Ilyen például a DÁTUM (***Dátum***) egyed. Evidens, hogy sok olyan egyéb egyedtípusunk lesz, amelyben több keltezés is szerepel. Tehát ezek többszörös kapcsolattal fűződnek a DÁTUM egyedtípushoz, mert aligha lesz olyan tervező, aki ezt az adatot ismétlődésesnek tekintve a 18.3 példa megoldását alkalmazná.

Arról majd a 18.6 pontban szólunk, hogy mi értelme van a DÁTUM egyed alkalmazásának. E pont lezárásaként arra hívjuk fel a figyelmet, hogy a többszörös kapcsolat sokszor „becsapós”.

Tegyük fel, hogy olyan szerződéseket kell ismeretekkel leírnunk, amelyekben pontosan két ügyfél szerepel és ismernünk kell e partnerek párosait is. Például házaspárokról van szó. A helyzetet a 18.6 példa szemlélteti, egyelőre tökéletlenül:

18.6 példa

PARTNER (***Partnerkód***, ...)
SZERZŐDÉS (***Szerződésszám***, Partnerkód-1, Partnerkód-2, ...)

A kétféle módon gondolkozó tervezők egyike a 18.2 példa megoldását választva levágná az ismétlődést úgy, hogy az új SZERZŐDÉS/PARTNER (***Szerződésszám***, *Partnerkód*) tételpárosa mellé semmilyen minősítő ismeretet sem tenne. (Az nem is lenne szükséges.) A másik elemző hozzá sem nyúlna a 18.6 példa megoldásához, hanem többszörös kapcsolatot tételezne fel.

Mindkét gondolatmenet hibás. A szerződés nem két külön ügyfélhez, hanem egy partnerpároshoz kapcsolódik. Itt belefutottunk az „A meg B" és az „A és B" 16.5 pontban ismertetett problémájába. Az előző két tervező egyáltalán nem tudta kifejezni azt a feltételt, hogy minden szerződésben csak egymáshoz tartozó partnerek szerepelhetnek. Ezt a korlátot nyilván egy további egyed fejezi ki. A SZERZŐDÉS ahhoz - és nem közvetlenül a PARTNER-hez kötődik. A jó megoldást a 18.7 példa mutatja:

18.7 példa

PARTNER (***Parnerkód***, ...)
PÁROK (***Párazonosító***, ...)
SZERZŐDÉS (***Szerződésszám***, *Párazonosító*, ...)

Párazonosító {Partnerkód-1+Partnerkód-2}

Nem a SZERZŐDÉS, hanem a PÁROK egyedtípus kapcsolódik kétszeresen a PARTNER egyedhez, amellyel a SZERZŐDÉS kapcsolata csak közvetett. Ez a példa már átvezet bennünket a többszörös homogén kapcsolatokhoz, amelyeket némi kitérő után tárgyalunk.

Ugyanis ezen kapcsolatok értelmezéséhez látnunk kell a szerepnevek függési viszonyait.

18.3 A szerepnevek függései

A többszörös kapcsolat a szintézises normalizálási eljárással nem fedezhető fel, hiszen a tartományok (pl. Személy-azonosító és Címkód) között csak egyféle viszony határozható meg (van függés - nincs függés) és a függés nem minősíthető. A dekompozíció sem tudna mit kezdeni a 18.8 példával, azaz nem tudná feltárni a SZEMÉLY egyedben a Helységnév tranzitivitását:

18.8 példa

CÍM (***Címkód***, ..., Helységnév)
SZEMÉLY (***Személy-azonosító***, *Lak-címkód*, *Levelezési-címkód*, ..., Helységnév)

Viszont az általunk alkalmazott heurisztikus normalizálás kulcsmátrixa figyelembe veszi a szerepneveket is. Módszerünk egyesíti a dekompozíció és a szintézis megoldásait. Ezért rá kell mutatnunk arra, hogy milyen módon kell felfogni a minősítő szerepnevek viszonyait más tulajdonságokhoz. Logikusnak tűnik, hogy négyféle összefüggést kell vizsgálnunk.

Az első esetben két olyan tétel viszonyáról van szó, amelyek nem ugyanabba a tartományba tartoznak, alapvetően nem ugyanazt jelentik. Az egyik és/vagy a másik is lehet szerepnév.

Ilyenkor a funkcionális függéseket úgy elemezzük, mintha egyik tényező sem lenne az. A RENDELÉS (***Rendelésszám***, Rendelés-vevőkód, Vevőnév) egyedben éppen úgy fennáll a tranzitivitás, mintha a vevő kulcsa nem szerepnévként, hanem egyszerűen (Vevőkód) jelenne meg. A RENDELÉSTÉTEL (***Rendelésszám+Cikkszám***, Tétel-cikknév) egyedben is látható a részleges függés, noha abban a Cikknév szerepnevét alkalmaztuk.

A második esetben az ugyanabba a tartományba tartozó szerepnevek viszonyairól van szó. Vegyük csak alapul a RENDELÉS (***Rendelészám***, Rendelésdátum, Szállításdátum, ...) egyedet. Bár a két keltezés a Dátum szerepneve, közöttük semmilyen függést nem fedezhetünk fel, hiszen több rendelést is feladhatnak illetve teljesíthetnek azonos napokon. Így a Rendelésdátum értékéből nem következtethetünk a Szállításdátuméra - és megfordítva. Ezért a kétféle keltezés tulajdonság „megfér" azonos egyedben.

Egészen más a helyzet a harmadik esetben, amely a szerepnévnek a saját elsődlegeséhez való viszonyát fedi le. Mielőtt erről a kapcsolódásról részletesebben is szólnánk, be kell vezetnünk egy új fogalmat. Az ún. ***triviális függésről*** van szó. Ez a kölcsönös függés egyik sajátos esete. Vegyük csak alapul a Címkód és a Lak-címkód kapcsolatát! Az összefüggés kölcsönös. „Alulról" - a szerepnév felől - nézve ez világos. Ha megadjuk a Lak-címkód értékét, akkor ismerjük a Címkódét is, mert a lakcím is egy cím. A triviális függés alulról mindig erős. „Felülről" - az elsődleges felől - vizsgálva a kérdést egy pillanatig habozunk. Ha kiválasztunk egy Címkód értéket, akkor nem feltétlenül bukkanunk rá egy Lak-címkód értékre. Ám ne feledjük el, hogy a függés nem azt jelenti, hogy mindig kell lenni értéknek a Címkód Z → Y Lak-címkód függés jobboldalán. A viszony csak azt fejezi ki, hogy ha egy Címkód értéket kiválasztunk, úgy legfeljebb (!) egy Lak-címkód értékre bukkanunk. Márpedig ez a feltétel teljesül. Ezért a függés kölcsönös, noha a Címkód oldaláról nézve lehet gyenge is.

A fentiek miatt a SZEMÉLY (***Személy-azonosító***, Címkód, Levelezési-címkód) egyedtípus hibásan tervezett. A két leíró tulajdonság - formálisan - kölcsönös függésben áll. Ez a felszíni jelenség - normálformahiba - ritkább esetben redundanciára vagy homonimára utal. Az első alesetben a Címkód pontosan ugyanazt jelenti, mint a Levelezési-címkód. A második alesetben két teljesen más tartalmú dologról van szó, vagyis a Levelezési-címkód valójában nem is szerepneve a Címkódnak. A leggyakoribb aleset a féloldalas minősítés. A Címkód valójában Lakcímkód-ot jelent, de a tervező nem alkalmazta a kellő behatárolást. Ha azt megtette volna, akkor visszajutottunk volna az előző - problémamentes - esethez.

Az elsődleges (Címkód) és a szerepnév (Levelezési-címkód) közötti egyeden belüli triviális függés mindig problémát jelez, ha egyik tétel sem az egyed azonosítója. (A következő két pontban rámutatunk arra, hogy amikor az egyik tulajdonság azonosító, akkor ezek a problémák nem léphetnek fel.) Ezért leszögezhetjük a szabályt, amely szerint:

Az egyedtípusban csak akkor szerepelhet egy elsődleges tulajdonság és annak szerepneve, ha a kettő közül az egyik az egyedtípus azonosítója.

A negyedik esetet a következő példa mutatja: BÍRÁLAT (***Bírálat-azonosító***, Bizottsági-tag, Bírálókód). A bíráló személy mindig bizottsági tag, akik szintén személy, viszont nem minden bizottsági tag bíráló. A BÍRÁLAT egyedben ugyanannak a tulajdonságnak két leíró feladatú szerepneve fordul elő és a redundancia már az első pillanatban szemet szúr. Ha megadjuk a Bírálókód értékét, akkor ismerjük a Bizottsági-tag tartalmát is. Tehát evidens, hogy az egyed a tranzitivitás egyik sajátos válfaját mutatja. Szemben a Rendelésszám → Vevőkód → Vevőnév tranzitivitással - ahol a Vevőkód és a Vevőnév nyilván nem ugyanabból az értéktartományból származik - itt egy trükkösebb helyzettel állunk szemben.

A formális megoldás adott. A Bírálat-azonosító → Bírálókód → Bizottsági-tag tranzitivitás miatt az utóbbi tételt el kell hagynunk az egyedtípusból. Viszont érdemes elgondolkoznunk azon, hogy minden bíráló bizottsági tag, de ez megfordítva nem igaz. Itt már a szerepnév (a Bizottsági-

tag a Személy-azonosító szerepneve) szerepnevéről (a Bírálókód a Bizottsági-tag szerepneve) van szó. Tehát alárendelt szerepnév-összefüggést kell vizsgálnunk, szemben a Lak-címkód és Számlázás-címkód mellérendelt szerepnév-viszonyával.

Miután leszögeztük, hogy a szerepnevek alárendelt-mellérendelt kapcsolódása egyáltalán nem közömbös, át kell térnünk a visszamutató viszonyok vizsgálatára.

18.4 Homogén viszonyok

A hagyományos normalizálási eljárások nem tudnak mit kezdeni a triviális függéssel. A szerepneveken alapuló tulajdonság-viszonyokat nem különböztetik meg a normális függésektől (ld. 18.3 pont). Így nem támogatják a többszörös kapcsolatokat sem (ld. 18.2 pont). A tervező vagy közönséges tranzitivitásként értékeli a Bírálat-azonosító → Bírálókód → Bizottsági-tag függéssorozatot, vagy a helytelen függést egyáltalán nem találja meg. Ez azért fordulhat elő, mert a dekompozíció és a szintézis módszere nem ad mechanikus megoldást a triviális függések értékelésére. Ennél fogva ezek az algoritmusok elméletileg hiányosak, hiszen nem garantálják az adatmodell valósághűségét.

Mindennek az az oka, hogy a hagyományos tervezési eljárások csak a „van”-ra koncentrálnak. Arra, hogy ***létezik-e függés*** az X és az Y tétel között. A minősítés elmarad. Nem kérdés az, hogy ***milyen függés van*** az X és az Y között: normális vagy triviális? A dekompozíció és a szintézis a viszonyt puszta mennyiségre redukálja (a funkcionális függés 1:N fokú tulajdonság-viszony), a minőség már kívül esik az e módszereket alkalmazó tervezők látókörén.

Persze ez a hiányosság érthető. Az egyedtípusokat (pl. SZEMÉLY) táblázatokként fogjuk fel. A táblák oszlopokból és sorokból állnak. Az előbbiek jelentik a tulajdonságtípusokat (Személy-azonosító, Lak-címkód stb.), az utóbbiak az egyedelőfordulásokat (1. személy, 2. személy stb.). A dekompozíció és a szintézis módszere csak arra készült fel, hogy a „vertikális” vetületet, a tulajdonságtípusok összefüggéseit, a tábla intenzionális aspektusát vizsgálja. Ez a két eljárás nem veszi figyelembe a „horizontális” vetületet, az egyedelőfordulások összefüggéseit, vagyis a táblázat extenzionális aspektusát.

Pedig könnyen belátható, hogy az intenzionális és az extenzionális dimenzió nem választható el egymástól. A 18.1 példa dilemmáját az egyik tervező átértelmezéssel, intenzionálisan (ld. 18.4 példa), míg a másik új egyedsorok bevezetésével, extenzionálisan (ld. 18.3 példa) oldotta fel.

Az extenzionális dimenzió vizsgálata főleg a homogén viszonyoknál és az egyed-altípusoknál játszik szerepet. Az utóbbiakat majd a következő pontban tárgyaljuk. Itt az adott egyedtípus előfordulásai közötti kapcsolódásokat fogjuk elemezni.

D 18/2 **Ha egy egyedtípus előfordulásai összefüggenek egymással, akkor homogén vagy visszamutató viszonyról beszélünk.**

Ilyen összefüggésekkel nagyon gyakran találkozhatunk. Az egyik személy a másiknak a házastársa. Az egyik szervezet a másik alá tartozik. Az utcák, vezetékek, a termékekbe beszerelt alkatrészek másik utcákhoz, vezetékekhez, alkatrészekhez kapcsolódnak stb. A kérdés az, hogy az ilyen homogén viszony is kifejezhető-e a funkcionális függés kategóriáival?

Az egyedtípusok közötti viszonyokat - közvetlenül vagy közvetve - mindig tulajdonságtípusok hordozzák. A VEVŐ és a RENDELÉS egyed a közös Vevőkód tulajdonságon kapcsolódik. A CIKK és a RENDELÉS egyed viszonyát a RENDELÉSTÉTEL egyed tükrözi, de ennek az összetett Cikkszám+Rendelésszám azonosítója valósítja meg az összefüggést. Ezért azt kell várnunk, hogy a homogén viszonyok is tulajdonságokon alapulnak.

Csakhogy van itt egy bökkenő. A visszamutató összefüggések ugyanannak az egyedtípusnak az előfordulásait kötik egymáshoz. Ezért a homogén viszonyok egyazon egyedtípus két azonos értelmű tulajdonságán kell, hogy alapuljanak. A tételeknek azonos az értelme, de nyilván nem lehet azonos a neve. Nézzük csak meg a 18.9 példát.

Az első megoldás kétszeresen rossz. Egyrészt két tulajdonságnak is azonos a neve. Másrészt a modellrészlet nem mutatja a kapcsolódást. Ezzel szemben a második megoldás tökéletes. Hűen szemlélteti, hogy a személyeknek lehet egy házastársa, ezért a személyt jellemzi a házastársat azonosító tulajdonság, amely ekként összeköti a házas feleket.

18.9 példa

SZEMÉLY (***Személy-azonosító***, ..., Személy-azonosító)
SZEMÉLY (***Személy-azonosító***, ..., *Házastárs-azonosító*)

Mivel egyazon egyedtípus két előfordulását kapcsoljuk össze, a kapcsoló tulajdonságnak ugyanaz kell, hogy legyen a tartalma, mint az azonosítónak. Következésképpen a Házastárs-azonosító szükségszerűen a Személy-azonosító szerepneve.

A SZEMÉLY - HÁZASTÁRS kapcsolat mifelénk adott időpontra nézve 1:1 fokú és - sokak szerint szerencsére - opcionális. Tehát a Személy-azonosító és a Házastárs azonosító között ***kölcsönös függés*** áll fenn. Ugyanennek a szerkezetnek a mintájára tudjuk tükrözni az 1:N fokú homogén kapcsolatot is. Lásd a 18.10 példát:

18.10 példa

SZEMÉLY (***Személy-azonosító***, ..., *Főnök-azonosító*)

Amennyiben feltételezzük, hogy minden személynek minden adott időpontban csak egy közvetlen főnöke lehet, úgy a 18.10 példa egyede jól kifejezi a FŐNÖK - SZEMÉLY homogén hierarchikus kapcsolatot. Ezt a viszonyt a Személy-azonosító → Főnök-azonosító ***funkcionális függés*** hordozza, amely mindkét irányban gyenge, tehát a kapcsolat mindkét oldalon opcionális. Van olyan személy, aki csak beosztott és van olyan is, aki a legnagyobb főnök. Ha ez nem így lenne, akkor a kapcsolat végtelen ciklus formáját öltené.

Az előző pontban kijelentettük, hogy az elsődleges tulajdonság és annak szerepneve nem lehet ugyanannak az egyedtípusnak leíró szerepű tényezője. Most ezt a tételt kiegészíthetjük azzal, hogy amennyiben egy egyedtípus leíró tulajdonságtípusként tartalmazza saját azonosítójának szerepnevét, akkor visszamutató kapcsolatban áll önmagával.

D 18/3 **Az egyedtípus akkor és csak akkor áll visszamutató kapcsolatban önmagával, ha azonosítója funkcionálisan meghatározza saját szerepnevét.**

A meghatározás kizárja, hogy az összetett azonosító része legyen egy elsődleges tulajdonság és annak szerepneve. Az E (***A+As***) egyedtípus, ahol As az A szerepneve, rosszul tervezett. Az elsődleges és a szerepnév között törvényszerűen függés áll fenn, márpedig az azonosító részei között nem létezhet függés (ld. 16.4 pont).

Különböző hálózatok esetében gyakran találkozunk azzal a helyzettel, amikor az egyedtípus előfordulásai M:N-es viszonyban vannak egymással. Mint tudjuk, ezt a viszonyt nem lehet magán az egyeden belül kifejezni. Azért nem, mert a viszonyt tulajdonságtípus hordozza és ez a tulajdonságtípus többszörös értéket venne fel, azaz ismétlődő adat lenne. A 18.11 példa mutat erre egy esetet:

18.11 példa

TÉTEL (***Tétel-azonosító***, ..., *Beépülő-tétel-azonosító*)
TÉTEL (***Tétel-azonosító***, ..., *Befogadó-tétel-azonosító*)

Az ipari gyártásban (termelési) tételnek nevezik a termékek egységeit a végterméktől kezdve a főszerelvényeken, szerelvényeken, alkatrészeken át az anyagokig. A magasabb szintű tételbe több alacsonyabb szintű tétel épül be. Ezt a listát nevezik darabjegyzéknek (angolul: parts list). Ezért a 18.11 példa első egyedtípusa rossz, mert a leíró tulajdonság értéke ismétlődik. Ugyanígy helytelen a második egyedtípus is, mivel egy tétel több másiknak az alkotóeleme. Ezt a listát beépülési jegyzéknek (angolul: goes-into list) hívják. Végeredményben a valós szerkezetet ***családfának*** nevezik, mert a családok leszármazási ágainak a mintáját követi. (A megnevezés eléggé pontatlan, mivel valójában nem fáról, hanem hálóról van szó.)

A 18.11 példa problémái nagyon könnyen megoldhatók. Ez mindig a 18.12 példának megfelelő módon történik. Mivel ismétlődésről van szó, az ismétlődő tételt az eredeti kulccsal együtt külön egyedbe helyezzük. Azonban ha csak ennyit tennénk, akkor olyan összetett azonosítót kapnánk, amelynek egyik része meghatározza a másikat. Mivel nem minden tétel befogadó, a sima Tétel-azonosító egyébként sem fejezné ki a szemantikai lényeget. Ezért kell szerepnevet alkalmaznunk:

18.12 példa

TÉTEL (***Tétel-azonosító***, ...)
CSALÁDFA (***Családfa-azonosító***, ...)

Családfa-azonosító {Befogadó-tétel-azonosító+Beépülő-tétel-azonosító}

Az új egyedtípus mindig két homogén, hierarchikus és alulról opcionális kapcsolattal kötődik az eredeti egyedhez. Tehát itt találkozunk a szerepneveken alapuló többszörös kapcsolat tipikus és elkerülhetetlen esetével. Elvileg ugyan nem törvényszerű, hogy a családfa egyed kulcsának a része legyen a két szerepnév. Elképzelhető a 18.13 példának megfelelő megoldás is:

18.13 példa

TÉTEL (***Tétel-azonosító***, ...)
CSALÁDFA (***Családfa-kulcs***, *Befogadó-tétel-azonosító, Beépülő-tétel-azonosító*, ...)

Ez a változat két okból sem szerencsés és ezért kerülendő. Egyrészt mesterséges - és ekként semmitmondó - új tulajdonság bevezetését igényli. Másrészt ismeretet vesztünk, mert a két szerepnév közötti „és” (+) viszonyt „meg” összefüggésre degradáljuk.

18.5 A feltételes függés és az egyedaltípus

Sok adatkezelő egyik súlyos hibája, hogy nem engedi meg a valódi nulla, a nem-szignifikáns érték és a nem-értelmezhető érték közötti különbségtételt. A lényeget a SZEMÉLY egyedtípus Szülések-száma nevű tulajdonságával szemléltetjük. Négy eset lehetséges (ld. a 7.9 pontot):

- Az „A” személy hölgy, aki N-szer szült. Ekkor a tulajdonság valós értéke ***N***.
- A „B” személy hölgy, aki egyszer sem szült. Az érték ***valódi 0***.
- A „C” személy nő, akiről nem tudjuk, hogy hányszor szült. Az érték ***inszignifikáns 0***.
- A „D” személy férfi vagy pici lány. Ekkor az érték ***nem-értelmezhető***.

A megkülönböztetés szemantikai és gyakorlati (pl. statisztikai) értelmét nem akarjuk most kifejteni. Arra viszont rá kell mutatnunk, hogy a „nem-értelmezhető” érték is valódi tartalom. Jelöljük ezt az értéket az „NA” jellel, ami nem-alkalmazhatót (angolul: not applicable) jelent. Tehát a továbbiakban ***NA-értéknek*** fogjuk nevezni a nem-értelmezhető tartalmat. (Figyelem: az NA nem tévesztendő össze az n.a. „nincs adat” megjelöléssel, amely összekeveri a nem-ismert és a nem-alkalmazható lényegeket!)

Az NA-értékre épül saját találmányunk, az ún. ***feltételes függés***. Ennek meghatározása a következő:

D 18/4 **Az E egyedtípus A azonosítója akkor és csak akkor határozza meg feltételes függéssel az egyedtípus B tulajdonságát, ha létezik olyan (implicit) C tulajdonság, amelynek adott tartalma(i) esetében a B mindig NA-értéket vesz fel.**

A definíciót a 18.14 példa segítségével magyarázzuk:

18.14 példa

SZEMÉLY (***Személy-azonosító***, Foglalkozás, ..., Orvoskód, Rendelő-körzet, ...)

Az egyednek létezik egy olyan tulajdonsága (Személy-azonosító), amely meghatároz több másikat (Orvoskód, Rendelő-körzet) úgy, hogy ha a foglalkozás értéke nem „háziorvos”, akkor azok tartalma nem-értelmezhető.

A feltételes függés jele A C(i) $\rightarrow$ B, ahol C a feltételtényező és i maga a konkrét feltétel. Vegyük észre, hogy a feltételes függés a tranzitív függés speciális esete. Ha a foglalkozás „háziorvos”, akkor abból még nem következik az Orvoskód értéke. Ha a foglalkozás „mérnök”, akkor viszont az Orvoskód NA-értékű.

A feltételes függés nem tévesztendő össze az opcionalitással! A személy Szülések-száma tulajdonsága erősen függ a Személy-azonosítótól, ha minden személynél meg kell adni annak konkrét értékét (ez lehet a NA-érték is!). A függés gyenge, ha ez a kitétel nem áll fenn. Tehát alkalmazhatunk „nem-ismerem” értéket is.

A NA-érték különlegesen „szép” modellezési tényező, mivel magában hordozza az egyedtípus extenzionális és intenzionális vetületét. Maga a NA-érték az extenzióban - az előfordulásokban - lép fel, de ugyanakkor intenzionálisan - a tulajdonságtípusokra - is értelmezhető. Mindez pedig egy teljesen új normalizálási elvnek ad alapot.

Az eddigi normalizálási eljárások ***intenzionálisak*** voltak. A tulajdonságtípusok viszonyait elemeztük és a táblát „vertikálisan” bontottuk meg. A RENDELÉS (Rendelésszám, Vevőkód, Vevőnév) egyedből kivettük a Vevőkód/Vevőnév párost, amivel a tábla oszlopait szabdaltuk szét. Most rábukkantunk az ***extenzionális*** normalizálásra. A tisztán extenzionális megbontás a tábla sorai mentén, „horizontálisan” történik. Az egyedelőfordulásokat új egyedtípusokban szórjuk szét bizonyos feltételek szerint. Ezt a megbontást mutatja a 18.15 példa:

18.15 példa

ERŐFORRÁS (***Erőforrás-azonosító***, Személy, ..., Anyag, ..., Eszköz, ...)

SZEMÉLY (***Személyazonosító***, Személy, ...)
ANYAG (***Anyagazonosító***, Anyag, ...)
ESZKÖZ (***Eszközazonosító***, Eszköz, ...)

Az eredeti egyedtípus 5NF alakban van, mivel a tulajdonságok egymástól függetlenek. Mégis érezzük, hogy az ERŐFORRÁS végletesen rosszul tervezett egyed. Azért, mert ha az erőforrás személy-jellegű, akkor az anyag és az eszköz adatai NA-értékűek. Ha anyag, akkor a személy és az eszköz adatai nem értelmezhetőek. A tábla NA-értékei azt mutatják, hogy a valós jelenségeket nem helyesen tártuk fel. Ezért a táblát a sorai mentén extenzionálisan megbontjuk. A személyekre vonatkozó sorokat a SZEMÉLY, az anyagokét az ANYAG egyedbe tesszük stb.

D 18/5 **Azt a műveletet, amelynek során az eredeti egyedtípus előfordulás (rész)-sorait a feltételes függés szerint egyedtípus(ok)ba helyezzük, specializációnak nevezzük.**

A specializáció lehet ***teljes***, amikor az eredeti egyedtípus megszűnik (ld. 18.15 példa). Azonban a specializáció a legtöbb esetben ***részleges***. A 18.14 példa specializálásával a következő képet kapjuk:

18.16 példa

SZEMÉLY (***Személy-azonosító***, Foglalkozás, ...)
HÁZIORVOS (***Orvos-személy-azonosító***, Orvoskód, Rendelő-körzet, ...)

A specializációval nyert új egyedtípust az eredeti egyed ***altípusának*** nevezzük. Az orvos a személy főtípus (angolul: supertype) altípusa (angolul: subtype). Az egyedaltípust elvileg a főtípus kulcsa azonosítja. Viszont két egyedtípusnak nem lehet azonos a kulcsa. Ezért ilyenkor az eredeti azonosító korlátozó jellegű szerepnevét használjuk. Minden orvos személy, de nem minden személy orvos. Ez a szerepnév - feladata miatt - nem tévesztendő össze a pusztán minősítő feladatú szerepnévvel.

18.17 példa

SZEMÉLY (***Személy-azonosító***, Státus, ...K, T, D, E)

SZEMÉLY (***Személy-azonosító***, Státus, ...K)
TANÁR (***T-személy-azonosító***, T)
DIÁK (***D-személy-azonosító***, D)
EGYÉB (***E-személy-azonosító***, E)

A specializáció gyakran ***alternáló***. Ez azt jelenti, hogy az eredeti egyedtípus megbontásával több, egymást előfordulásaikban kizáró specializált egyedaltípust kapunk. Tegyük fel, hogy az iskola tanáraira, diákjaira és az egyéb személyzetre vonatkozó ismereteket akarunk vezetni. Vannak olyan ismeretek, amelyek e személyekre vonatkozóan közösek (K), és amelyek a státustól függenek (T, D, E). A SZEMÉLY egyed megbontását a 18.17 példa mutatja.

Az egyedaltípusoknak (TANÁR, DIÁK, EGYÉB) egymást kizáróaknak kell lenniük. Különben fizikai redundancia keletkezne, amely karbantartási anomáliákat okozna. Ha tehát netán van

olyan diák, aki egyben tanár is, akkor a T és a D ismeretsorok tulajdonságai közül azokat, amelyek egyszerre jellemezhetnek egy tanár-diák személyt, nem szabad lebontani.

A részleges specializációval nyert altípusok a főtípussal (SZEMÉLY) alulról kötelező, felülről opcionális 1:1 fokú „is-egy" típusú kapcsolatban állnak az azonosítójukon keresztül. A tanár is egy személy, a diák is egy személy. Az egyedaltípus a főtípus tulajdonságait „örökli". Ebből következik, hogy az altípusnak és a főtípusnak nem lehet közös tulajdonsága. (N.B.: Az altípusok között is kizárt az ilyen redundancia. Ha több altípusnak közös tulajdonsága van, akkor azt az eredeti egyedben - esetünkben a K ismerethalmazban - kell hagyni.)

A specializációnak elméleti és gyakorlati oka van. Ha a tanárokat és a diákokat a valóságban számunkra lényeges eltérő ismeretek jellemzik, akkor elvi alapunk van a specializációra. Mivel nem igaz, hogy a kezelőrendszerek az üres értékeknél nem igényelnek tárt és kezelési időt, a megbontásnak a gyakorlati indoka sem közömbös.

Azonban a világ sokszínű. A SZEMÉLY egyedtípust nem fogjuk nőkre, kislányokra és férfiakra specializálni azért, mert az egyetlen Szülések-száma tulajdonság erre alkalmat adna. Ilyen alapon külön egyedtípust kellene kreálni azokból a személyekből, akiknek van biciklije és azokból, akiknek nincs. Nem specializáljuk az ALKATRÉSZ egyedet azért, mert az Átmérő tulajdonságnak a szögletes alkatrészek esetében NA-érték a tartalma.

A specializációt vagy a szemantikai tisztaság, vagy a NA-értékek kiküszöbölése érdekében hajtjuk végre. Az utóbbi esetben ökölszabály igazít el bennünket. Ha az egyed tulajdonságainak a harmada NA-értéket vesz fel adott feltétel függvényében, akkor a specializációnak értelme van.

A specializáció fordított művelete az általánosítás, a ***generalizáció***. A 18.18 példa szemlélteti ezt a műveletet:

18.18 példa

TANÁR	(***T-személy-azonosító***, T, K)
DIÁK	(***D-személy-azonosító***, D, K)
EGYÉB	(***E-személy-azonosító***, E, K)

SZEMÉLY	(***Személy-azonosító***, Státus, ...K)
TANÁR	(***T-személy-azonosító***, T)
DIÁK	(***D-személy-azonosító***, D)
EGYÉB	(***E-személy-azonosító***, E)

A példa részleges generalizációt mutat. Az eredetileg tervezett három egyedtípus közös (K) ismerethalmazát kiemeljük a főtípusba, amelyben - kezelési okokból - felvesszük az altípusra utaló új tulajdonságtípust (Státus). A generalizáció feltételezi, hogy az általánosított egyedek azonosítói ugyanabból az értékhalmazból származó (implicit) szerepnevek. Ha ez nem lenne így, akkor kettős azonosítást kellene alkalmazni.

18.6 Unáris egyedek

Általában az egyedtípusokat kétdimenziós táblákként képzeljük el. Adott például a SZEMÉLY egyedtípus úgy, hogy a személyeket több tulajdonság jellemzi (horizontálisan) és természetesen több előfordulás található a táblában (vertikálisan). Ezzel a kettős képpel szemben hozzá kell szoknunk, hogy vannak unáris egyedtípusok is az adatmodellben.

Az ***unáris*** megjelölés a vertikális (intenzionális) dimenzióra vonatkozik. Azt mutatja, hogy az egyedtípusnak csak egy tulajdonságtípusa van. Tehát az egyedtípus olyan speciális ***csupakulcs*** reláció, amelyben az azonosító elemi. A 18.2 pont végén említett DÁTUM (***Dátum***) egyedtípus ebbe a kategóriába esik. Most válaszolunk arra az ott felvetett kérdésre, hogy egyáltalán mi értelme van az ilyen egyedtípusok kreálásának. A problémát több síkon fogjuk megközelíteni.

A mai tervezők igencsak hajlamosak a 18.19 példa által mutatott szerkezet kreálására:

18.19 példa

SZEMÉLY (***Személy-azonosító***, ...)
NYELV (***Nyelvkód***, Nyelv)
NYELVTUDÁS (***Személy-azonosító+Nyelvkód***, ...)

Az elméleti - és helyes - feltételezés az, hogy a NYELVTUDÁS egyed M:N-es viszonnyal kapcsolja össze a személyeket és a nyelveket. Egy személy több (N) nyelvnek is a birtokában van, miközben ugyanazt a nyelvet többen is (M) beszélik. Tehát a kapcsoló egyedben a kapcsolt egyedek azonosítóiból összetett kulcsot célszerű alkalmazni.

Valójában a 18.19 példa megoldása mögött gyakorlati okok rejtőznek. A Nyelv „hosszú" - mondjuk 14-karakteres - adat, a Nyelvkód pedig rövid - például 2-karakteres - tétel, hiszen éppen ezért kód. Bevezetésével, úgymond, tárt takarítunk meg.

Mennyit? Csak pár ezer karaktert! Ennek a néhány bájtnak a kedvéért bevezetünk egy új, élettelen, semmitmondó adatot, amelynek a kezelése és egyértelműségének a biztosítása külön gondokat okoz. Mert tegyük fel, hogy a „11=angol" kód-megnevezés páros jelöli az angol nyelvtudást. Mit gondol az olvasó: amikor az X személy angol nyelvtudását rögzíteni akarjuk, akkor hibázhatunk-e inkább, ha a „11" kódot, vagy ha az „angol" szót akarjuk beütni? A válasz egyértelmű: nem-humánus dolog a kódokat fejben tartani, ezért a kód alkalmazásakor követünk el több hibát. Ráadásul még az sem igaz, hogy a kódbevitellel jelentős időt takarítunk meg. Hiszen a bevitelt támogató listából az „angol" szöveg egy pillanat alatt kiválasztható.

Ezért a 18.19 példa helyes megoldása szerintünk a következő:

18.20 példa

SZEMÉLY (***Személy-azonosító***, ...)
NYELV (***Nyelv***)
NYELVTUDÁS (***Személy-azonosító+Nyelv***, ...)

Tervezőink még ma is a lyukkártyás/mágnesszalagos technikában gondolkoznak. A 70-es évek elején még volt jelentősége a Nyelvkód alkalmazásának. Ma ennek már semmi értelme sincs. Ennek dacára tervezőink teletűzködik adatbázisainkat vak-vezet-világtalant „segítőkész" kódokkal.

Persze ennek a rossz szokásnak a végső alapja az elméleti ismeretek hiánya. A tervezők egy része még mindig a régi COBOL szintjén gondolkozik. Még mindig azt hiszi, hogy a külső, a felhasználói azonosító a belső, tárolási kulcs egyetlen szilárd alapja. Ez súlyos tévedés. A mai adatkezelő rendszerek mindegyike saját generált belső adatbáziskulcsokkal dolgozik. Ezért az adatkezelés során két teljesen azonos ismeretsorra is mondhatjuk, hogy azok eltérő lényegek és két különbözőre is, hogy azonosak (amennyiben formális szempontból analóg felépítésűek).

Ezért a mesterséges kódokat csak mértékkel szabad alkalmazni. Viszont annál bátrabban kell nyúlni az unáris (ld. 18.20 példa: NYELV) egyedekhez a természetesség, az ellenőrizhetőség és a homogén lekérdezhetőség (ki milyen nyelvet tud - az adott nyelvet ki ismeri) érdekében. Gondoljunk arra, hogy majd a „beszélő számítógépek" korában - titkos adatbázisoktól eltekintve - nem azt fogjuk kérdezni a géptől, hogy kiknél „11" a Nyelvkód értéke. Hanem azt, hogy ki beszél „angol"-ul.

Az unáris egyedtípusok alkalmazásának van egy másik indoka is. A tapasztalt tervező tudja a „titkot”, amely szerint két mellérendelt egyedtípus sohasem kapcsolható össze a közös leíró tulajdonságon a szemantikai félreértelmezések veszélye nélkül. Lássuk csak a 18.21 példát (vö. 6.7 ábra):

18.21 példa

SZEMÉLY	(***Személyazonosító***, ..., Költséghelykód)
GÉP	(***Gépazonosító***, ..., Költséghelykód)

Ha ezt a két egyedtípust valaki a Költséghelykód alapján az összekapcsolás (join) műveletnek veti alá, akkor teljesen használhatatlan ismeretekhez jut. Olyan személyeket fog a gépekhez kötni, amelyeknek azokhoz a masinákhoz semmi közük sincsen. Csupán a személy illetve a gép ugyanazon a költséghelyen találtatik. Most tegyük fel - nem egészen reálisan -, hogy a költséghelyek ismereteire nem vagyunk kíváncsiak. Ennek dacára létre fogjuk hozni a KÖLTSÉGHELY (***Költséghelykód***) unáris egyedet. Ezzel két célt fogunk elérni.

Egyrészt ezzel a megoldással felvázoltuk a helyes szemantikai képet. Tehát ebből az egyedből kiindulva külön-külön le tudjuk kérdezni az adott költséghelyhez tartozó személyek és gépek ismereteit. (A tapasztalatlan tervező ne mondja most, hogy ezt indexekkel is megteheti. Az index nem fogalmi szintű tényező. Bármire alkalmazható. Nem fejezi ki a valós jelenségek tényleges viszonyait. Sokszor csak a tervező titkos és zavaros elgondolásait leplezi.)

Másrészt az új egyed (KÖLTSÉGHELY) validálási célokat is szolgál. Az indexekben gondolkozó tervező a 18.21 példa mindkét egyedében tetszőleges Költséghelykód értéket enged meg azzal, hogy majd „ráindexel”. Aki viszont nem fizikai (index), hanem fogalmi (új egyed) szintű megoldást alkalmaz, az a GÉP és a SZEMÉLY hivatkozási integritási korlátján keresztül ellenőrizni tudja, hogy a gépek és a személyek csak létező - előzőleg megadott - költséghelyekhez kapcsolódjanak.

A mai adatkezelők a tulajdonságtípusok (Költséghelykód) szintjén is megengedik a validálási kritériumok megadását. Ezek az ún. értéktartományra vonatkozó korlátok. Amikor a doméjn korlátját egyszerűen lehet kifejezni (pl. a Számlaszám 1 és 999999 közötti lehet), akkor ez a megoldás a célszerű. Ha viszont egy listás ellenőrzést kell alkalmazni, akkor az explicit unáris egyed felvétele lehet a kézenfekvő megoldás.

A DÁTUM (***Dátum***) unáris egyed alkalmazásának is a tiszta szemantikai elrendezés és az egyértelmű validálás a célja. Ezen kívül vegyük észre, hogy az időpontok sokszor hálósan kapcsolódnak egymáshoz. Lehetnek olyan egyedeink, amelyeknek a kulcsai két keltezésből tevődnek össze (-tól és -ig értelemben). Tehát a kulcsrészek szerepnevek, csak éppen ezt a jelleget nem tudjuk kifejezni, mert nincsen olyan egyedünk, amelyben a Dátum tisztán, nem-szerepnév tulajdonságként jelenik meg. A HALÁSZATI-TILALOM egyedben a Dátumtól és Dátumig két tulajdonságtípus szemantikailag értelmezhetetlen, ha még azt sem tudjuk, hogy mi a Dátum. A DÁTUM unáris egyed mindezt a problémát megoldja.

18.7 Szinguláris egyedek

Az előző pontban azokról az egyedtípusokról volt szó, amelyeket egyetlen tulajdonságtípus jellemzett. Nem csak a szimmetria kedvéért szólnunk kell az olyan egyedtípusokról is, amelyek viszont csak egyetlen egyedelőfordulással rendelkeznek és ilyen értelemben nem is valódi „típusok”.

Az egyetlen előfordulású egyedtípusokat ***szinguláris*** egyedtípusoknak nevezzük. Nem arról van szó, hogy gyakorlatilag ma csak egyetlen vevő ismereteit vezetjük. Ettől még a VEVŐ egyedtípus nem szinguláris. A szinguláris egyedtípusnak elvileg és minden időpontban egy és csakis egy előfordulása van. (N.B.: A relációs adatkezelők ezt a fogalmat nem ismerik. Holott a szinguláris minősítés éppen úgy integritási korlátként szolgál, mint az unáris megjelölés.)

Vajon mi lehet a háttere, oka, értelme annak, hogy egyetlen előfordulású egyedtípust hozzunk létre? A kérdésre adott válasz előtt az ún. származtatott tulajdonságok természetét és viszonyait kell áttekintenünk. ***Származtatott tulajdonságtípus*** az a tényező, amelynek értéke más tételek tartalmából matematikai és/vagy logikai művelettel, műveletekkel születik (ld. D 8/3). A rendeléstétel súlyát úgy határozzuk meg, hogy a cikk egységsúlyát megszorozzuk a rendelt mennyiséggel. Tehát a Rendeléstétel-súly származtatott tulajdonságtípus.

Az adatbázistervezés egyik régi-régi kérdése, hogy szabad-e az adatmodellben származtatott tulajdonságokat szerepeltetni (ld. 8.5 pont). Erre a felvetésre az első reakciónk határozott nem lenne, de a pont végére kiderül a teljes igazság. Az érvelést kísérve azt is el fogjuk árulni, hogy az adatmodellen belül hol van a származtatott tulajdonságtípus konkrét elméleti helye.

Először is azt kell látni, hogy a származtatási műveletek alapvetően kétféle jellegűek. (A kombinációikról nem fogunk megemlékezni ebben a könyvben.) Vannak intenzionális és vannak extenzionális származtatások.

Az ***intenzionális származtatás*** az adatmodell különböző tulajdonságtípusain alapul. Például a RENDELÉSTÉTEL Tételérték tulajdonságának a tartalmát a Tételérték = Mennyiség*Egységár képlet alapján számoljuk ki. Az intenzionális származtatás sajátossága az, hogy a származtatott tulajdonság a származtatási tényezőket szolgáltató egyedtípusok valamelyikét jellemzi. Egészen pontosan meg tudjuk mondani azt is, hogy melyiket.

Az intenzionálisan származtatott tulajdonság azt az egyedtípust jellemzi, amelynek kulcsa a származtatási tényezőket tartalmazó egyedtípusok kulcsainak a ***legkisebb közös többszöröse***. Ezt a fogalmat most nem a szoros matematikai értelemben használjuk. Hanem úgy, hogy a meghatározó kulcs a származtatási tényezők kulcsainak a minimális konkatenációja.

Példánk esetében a dolog így működik: A Mennyiség tényezőt a RENDELÉSTÉTEL Rendelésszám+Cikkszám kulcsa határozza meg. Az Egységár tényező a CIKK egyed Cikkszám tulajdonságától függ. Ezért a minimális konkatenáció a Rendelésszám+Cikkszám. Tehát a Tételérték az ezzel az összetett kulccsal azonosított RENDELÉSTÉTEL egyedet jellemzi.

Elvileg. A kérdés az, hogy szabad-e a Tételérték tulajdonságot a RENDELÉSTÉTEL valódi adataként is felvenni? Erre a felvetésre kétszeres nemmel kell válaszolnunk. Egyrészt a Tételérték tartalma a Mennyiség és az Egységár értékekből levezethető, így tartalma redundáns. Ez a redundancia karbantartási anomáliákhoz vezet. Ha például az Egységár módosul, akkor aktualizálni kell a Tételértéket is. Másrészt a Tételérték explicit felvétele ellentmondana eddigi normalizálási ismereteinknek is.

Az adatmodell-szintézis szabályai szerint ha egy tulajdonság (pl. Vevőkód) meghatároz egy másikat (Vevőnév), akkor azonosító szerepű és létre kell hozni a megfelelő egyedtípust. Úgy, hogy a meghatározó azonosító, a meghatározott pedig leíró. A VEVŐ (***Vevőkód***, Vevőnév) egyed törvényszerűen következik a Vevőkód $\rightarrow$ Vevőnév függésből.

Mármost vegyük észre, hogy az intenzionális származtatás nem más, mint egy sajátos függés. A Mennyiség+Egységár meghatározza a Tételérték tulajdonságot, mert a páros minden tartalmához csakis egy Tételérték tartozhat. No de kinek jutna eszébe létrehozni egy olyan egyedtípust, amelynek a Mennyiség+Egységár - változásnak kitett tartalmú - párosa lenne a kulcsa?

Most nézzük az ***extenzionális származtatás*** esetét. Az előző példára alapozva kérdezzük meg, hogy mennyi az adott rendelés tételeinek az összértéke, a Rendelésérték. Itt már nem arról van szó, hogy különböző tulajdonságtípusokat „dolgozunk össze". Hanem arról, hogy végigmegyünk a RENDELÉSTÉTEL egyedelőfordulásain és egyetlen tulajdonságtípus értékein alkalmazzuk a származtatást. Mert fennáll a Rendelésérték = Σ Tételérték összefüggés.

Ebben az esetben ***halmazműveletről*** van szó, amikor is a kifejezést nem a hagyományos értelemben használjuk. A „halmaz” szó az egy rendeléshez tartozó rendeléstételek előfordulás-halmazát jelenti. Végig kell ballagni a RENDELÉSTÉTEL előfordulásain és summázni kell a Tételérték tartalmát akkor, ha a Rendelésszám értéke ebben az egyedben és a RENDELÉS egyedben azonos.

A fenti analógiával élve a származtatott tulajdonságot ebben az esetben a származtató tényezővel kapcsolatos egyedek kulcsainak a ***legnagyobb közös osztója*** határozza meg. Mivel a kiinduló RENDELÉS egyed kulcsa a Rendelésszám, a kapcsolódó, a Tételérték tényezőt tartalmazó RENDELÉSTÉTEL egyed kulcsa pedig a Rendelésszám+Cikkszám együttese, a Rendelésérték a Rendelésszámmal azonosított egyedbe fog kerülni.

Ez a gondolatmenet mindaddig megállja a helyét, ameddig van kiinduló egyedtípus. Most azonban tegyük fel azt a kérdést, hogy mi a vevőink által feladott rendelések összértéke? A származtatási képlet világos: Összérték = Σ Rendelésérték. Tehát megint csak extenzionális - az egy egyedtípus (RENDELÉS) egyedelőfordulásaira vonatkozó tulajdonság (Rendelésérték) tartalmain végrehajtandó származtatási műveletről van szó.

Csakhogy most bajban vagyunk. A Tételérték summáját jogosan és ésszerűen a RENDELÉS már létező egyedtípushoz tudtuk kötni. Ámde milyen egyedet ír le az Összérték tulajdonság? Mi az a másik tulajdonság, ami őt meghatározza?

Az Összérték jellemző olyan tulajdonság, amelynek az egész adatmodellben csak egyetlen értéke van. Ugyanilyen jellemző lenne például az Átlagérték amely azt mutatja, hogy átlagosan mennyi a rendelések értéke. Az ilyen egyedi értékek a hagyományos modellezési gondolkozásban nem köthetők egyedtípushoz. Mert a „típus” eleve azt jelenti, hogy több előfordulást feltételezünk.

Ezen a korláton túl kell lépnünk. Hozzunk létre egy ÖSSZRENDELÉS egyedtípust. Ennek leíró tulajdonsága a rendelések Összérték tulajdonsága. A baj csak az, hogy nem találunk azonosítót. Ami felesleges is, ha egyetlen értékről van szó, tehát az azonosítás szükségtelen.

Módszerünkben bevezettük a szinguláris-egyed fogalmát. Az ÖSSZRENDELÉS egyed képe a következő: ÖSSZRENDELÉS (***Össz-rendelésszám***, Összérték). Az Össz-rendelésszám egy mesterséges kulcs. Alkalmazása elvileg azt feltételezi, hogy minden RENDELÉS egyedelőfordulásban megjelenjen az értéke. Hiszen teljesen világos, hogy az ÖSSZRENDELÉS és a RENDELÉS egyed 1:N fokú viszonyban áll. Ha az Összérték tartalmát valóban tárolni is akarjuk, akkor a fogalmi adatmodell szintjén el kell könyvelnünk a teljesen mesterséges kulcsot akkor is, ha eszünk ágában sincs azt ténylegesen alkalmazni. Az pusztán a tiszta szemantikai áttekintést segíti. Utal arra, hogy az Összérték tartalmát a Rendelésszámmal azonosított egyedeken végighaladva határozzuk meg.

Az elméleti hátteret tisztáztuk. Gyakorlatilag nem válaszoltunk a kérdésre, hogy szabad-e intenzionálisan és/vagy extenzionálisan származtatott tulajdonságokat felvenni az adatmodellbe?

Nos, a fogalmi adatmodellezés szintjén mindig célszerű meghatározni a származtatott tulajdonságok elvi helyét - befogadó egyedét - akkor is, ha eszünk ágában sincs tárolni a levezethető adatot. Azért érdemes, mert pontosan látjuk a származtatási algoritmus kezelési következményeit. Azt, hogy a Tételérték kiszámításához együtt kell kezelni a CIKK (Egységár) és a RENDELÉSTÉTEL (Mennyiség) egyedtípusok megfelelő tulajdonságait.

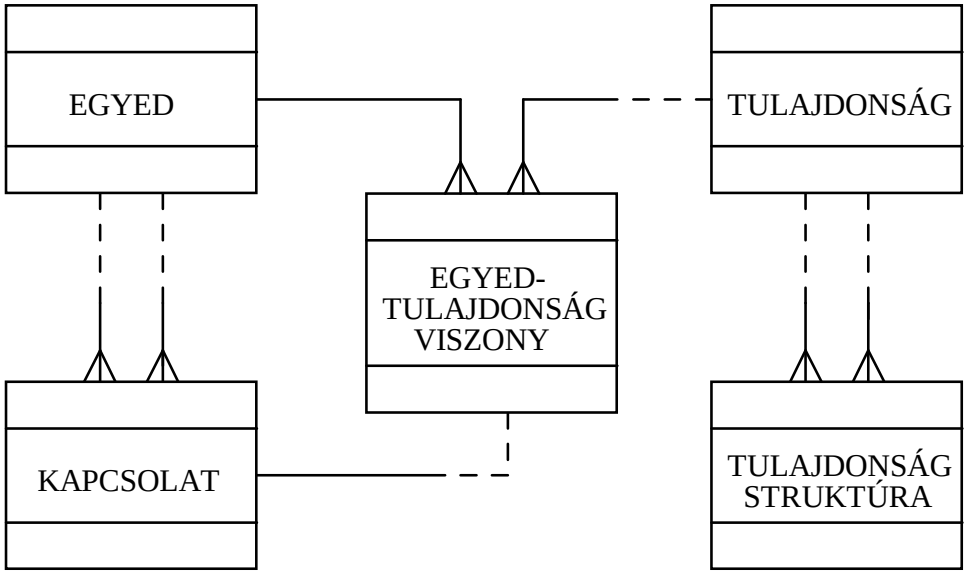
18.8 Konstansok és tulajdonságstruktúrák

Most megismételjük itt a 9.2 ábra metamodell-diagramját. Lásd a 18.1 ábrát. Az olvasóban felvetődhet a kérdés, hogy miként létezhetnek olyan tulajdonságok egy modellben, amelyek nem kapcsolódnak egyedekhez (lásd a TULAJDONSÁG-ból kiinduló, az opcionalitást mutató

szaggatott vonalat) és mit is jelent valójában a más modellezési koncepciókban nemigen alkalmazott „tulajdonságstruktúra”?

A magyarázatot kezdjük az utóbbival. Az adatmodellben a tulajdonságok egymáshoz kapcsolódnak. Ebben a könyvben három ilyen - családfa jellegű - viszonyról volt szó. A ***csoportos*** tulajdonságok tényezői a csoportokba épülnek úgy, hogy a csoportnak másik csoport is lehet a tagja. Ugyanilyen módon a ***szerepnév*** is felfogható az eredeti tulajdonsághoz kapcsolódó tényezőnek. Nincs kizárva, hogy a szerepnévnek is legyen szerepneve. Gyakorlati okokból a szerepnevek tiszta hierarchiát kell, hogy alkossanak (ld. 7.4 pont). Végül a ***származtatott*** tulajdonságoknak is vannak (származtatási) tényezői úgy, hogy maga a tényező is lehet származtatott.

A tulajdonságstruktúra igen fontos szerepet játszik a modellelemzésben. Hiszen ki kell zárni, hogy ezekben az összefüggésekben ciklusok legyenek; azt, hogy az egy struktúrát alkotó tényezők bizonyos kombinációkban ne kerülhessenek egy egyedtípusba; azt, hogy ugyanazon csoport tagja legyen egy tulajdonság és annak szerepneve stb.



18.1 ábra: A metamodell diagramja

A származtatott adatok között több olyan is akadhat, amelynek valamelyik tényezője nem egyedhez kötött ismeret. Nem jellemez egyedelőfordulást, mert értéke a konkrét egyedektől független. Az ilyen tényezőket nevezzük ***konstansoknak***. Nem a konstans-táblázatokról van szó. Ha egy konstansféle több értéket is felvehet úgy, hogy az értékek közvetve más egyedeket jellemeznek, akkor a konstansféle egyedtípusban tükrözhető. Ilyen tényező például az ÁFA-százalék. Itt kimondottan csak azokról a konstansokról van szó, amelyek egyértékűek és ezért nincs is nevük. Például: mindig 20% az x-féle adó mértéke.

Az okos fejlesztő nem tesz ilyen származtatási tényezőként szolgáló konstanst a programba. Magyarul: nem írja be expliciten a programba a „20” számot. Azért nem, mert változáskor át kell írni a programot (vö. adat-program függetlenség). Viszont a névtelen konstansnak nevet ad. Azért, hogy tulajdonságstruktúrában tudja kifejezni a konstans és az annak segítségével származtatott tulajdonság viszonyát. Ez pedig arra jó, hogy ha a konstans változik vagy megszűnik, akkor könnyen és gyorsan meg lehessen állapítani, hogy a változás milyen egyéb - abból származtatott - tulajdonságokat érint.

Nincs akadálya annak, hogy a tervező a konstansokat egy közös olyan technikai egyedbe helyezze, amely nem a tulajdonképpeni adatmodell része, amennyiben nem kapcsolódik más egyedtípusokhoz. Ha megtanuljuk a konstansokat (technikai) egyedelőfordulásokként kezelni, akkor könnyebben tudjuk változtatni programjainkat. Ezért - bár ez nem tartozik szorosan a mi témakörünkbe - a szövegkonstansokat is technikai egyedelőfordulásokként érdemes meghatározni. Ezzel növeljük a rendszer hordozhatóságát és elmozdulunk az objektumorientált adatbázisok felé.

18.9 Az osztályozás kérdésköre

Az egyszerű adatmodellezési módszerekben az adatmodellt alkotó egyed-, tulajdonság- és kapcsolattípusok közül általában csak a tulajdonságot szokták expliciten osztályozni. Általában meg kell adni a tétel ábrázolását; kulcs vagy nemkulcs jellegét; azt, hogy lehet-e inszignifikáns (ezt tévedésből nevezik „not null”-nak); azt, hogy az érték egyedi-e stb. A kapcsolatokat csak impliciten - a kapcsoló tulajdonságon keresztül - osztályozzák. Jobb módszerekben utalni lehet a kapcsolat fokára, opcionalitására és birtoklási/tipizáló természetére is. Az elfogadhatókban támogatják az egyedtípus besorolását is [pl. [16]].

Az említett irodalom alap, kapcsoló, leíró és hivatkozási egyedtípusokat különböztet meg. ***Alap*** egyed például a cikk és a rendelés. ***Kapcsoló*** a kettőt összekötő rendeléstétel. A személy adatait kiegészítő, az ismétlődés levágásával nyert előző-munkahely egyed tipikus ***leíró*** tényező. Végül minden olyan egyed, amely tartalmazza egy másik elsődleges kulcsát, ***hivatkozási*** egyednek tekintendő. Amint látjuk, ez az osztályozás féloldalas, nem-konzisztens. Hiszen a kapcsoló és a leíró egyed szükségszerűen hivatkozási is, de az alapegyed is lehet az.

Ennek ellenére üdvözölnünk kell ezeket az osztályozási kísérleteket. Ezek a próbálkozások a száz-százalékos elv (ld. 7.2 pont) jegyében arra irányulnak, hogy az ismeretre vonatkozó ismeret ne a programban legyen rejtve, sőt, ne is csak korlátként fogalmazódjék meg. Hanem a struktúrához kötött - tehát könnyen ellenőrizhető - feltétellé váljon. Vegyük csak alapul a kapcsolóként osztályozott egyedtípust! Erről azt feltételezzük, hogy (legalább) két másik egyedtípusnak az alárendeltje és ezért tartalmaznia kell idegen-kulcsként azok azonosítóját. Ha az egyedtípusban nincs két kapcsoló tulajdonság, akkor hibásan tervezett.

Sajnálatos módon a mai tervezési módszerek csak nagyon szűkös osztályozási lehetőségeket nyújtanak (ld. 9.5 pont). Feltehetően azért, mert a konkrét adatkezelők ilyen irányú képességei még korlátosabbak. Pedig milyen jó lenne, ha kijelenthetnénk a családfa egyedről (ld. 18.12 példa), hogy ő családfa típusú. Akkor máris tudhatnánk - és modellező rendszerünk ellenőrizhetné -, hogy ez olyan speciális kapcsoló egyed, amely egy hozzá képest alapegyedhez kötődik két, alulról kötelező és felülről opcionális kapcsolattal, amit az azonosító két része hordoz, amelyek viszont az alapegyed azonosítójának a szerepnevei.

A széleskörű osztályozás a jó adatmodellezés egyik titka. Mert egy dolog, hogy az adatkezelő nem képes az osztályozást érvényesíteni. Nem ismeri a családfa, a szinguláris, az altípus stb. egyedtípus féleségeket. Más dolog, hogy a modell valósághűségének, teljességének, kapcsolhatóságának, konzisztenciájának az elemzésekor hallatlan segítséget nyújt a jó osztályozás. És nemcsak az elemzésben, hanem a fejlesztésben is. Mert ha az adatkezelő képtelen az osztályozáson alapuló definitív kezelésre, akkor magunknak kell procedúrákat írnunk az osztályozásnak megfelelő validálásokra.

A szemantikában húsznál több kapcsolatfélét ismernek. A birtokolja és az altípusa viszony mellett a rész-egész, az ott van, a családfa, a házasság, a hasonlóság stb. jellegű összefüggéseket. Ezek mind valamilyen ismereti konzekvenciával járnak. Ezért legalább az adatmodellezés szintjén meg kellene kísérelni e minták alkalmazását és követését.

Ezzel a fogalmi adatmodellről szóló mondókánk lényegét lezárjuk. Azonban a következő igen vegyes tartalmú fejezet talán még tartalmaz apró tervezési titkokat.

19. TIPIKUS TERVEZÉSI HIBÁK

19.1 Fegyelmezett fantázia

A jó adatbázisterv kialakításához számos technikai trükköt, módszertani megoldást kell ismerni. Tisztában kell lenni az adatmodell elvi felépítésével, részleteinek az összefüggéseivel, a matematikai és szemantikai alapú normálformákkal, a normalizálási eljárás lépéseivel stb. Magyarul: tudni kell az adatmodell „képleteit" és azok levezetéseit.

Mármost az olvasó nyilván tisztában van azzal, hogy nem minden a szakmájában jártas, annak mesterségbeli csínjait-bínjait jól ismerő építészmérnök tervez küllemében kellemes, belsejében otthonos lakóépületeket. Olyan társasházakat, amelyekben a lakók egyenként is és közösen is jól érezhetnék magukat, ha erre hajlamosak volnának.

Az adatbázistervezés sem csak szakma. Ki merjük mondani: ***a modellezés művészet is***. Ne mondja senki, hogy a szocreál városi ház és egy Makovecz-épület között csak technikai különbség van. A modellezéshez fantáziára, alkotóerőre, stílusra is szükség van. A mai adatbázistervek legfőbb hibája, hogy fantáziátlanok; szocreál-jellegű szögletes sablonok szerint, igénytelenül készülnek és - elnézést a szóért - izzadságszagúak.

Az ismeretek feltárásához filozofikus fegyelmezett fantázia igényeltetik. Sajnálatos módon a fejlesztők nem tanulnak filozófiát - ***ismeretelméletet***. Nincsenek tisztában saját „alapanyagaik" természetével. Elsősorban ebből fakadnak a technikai jellegű, alapvető, szinte gyermekesen komolytalan tervezési hibák. A kísérletezés, a próbálkozás, a mi-lenne-ha-így-és-hogy-nézne-ki-úgy gondolkodásmód, a ***képzelőerő*** hiánya miatt modelljeink nem könnyedek, nem maguktól értetődőek, kívülről sem „csinosak". E könyv szerzőjének alkalma volt sok tucat (!) adatbázistervet áttanulmányoznia. Mindig feltette magának a kérdést: Melyik tetszik, melyikkel tudna azonosulni? Csak igen kevésben látta meg a fantázia, a szellemes és ízléses megoldások csíráit.

Kiemelt jelzőként használtuk a ***fegyelmezett*** szót. Ezen két dolgot értünk. Egyrészt a mérnöki pontosságot. Az egységes képet, a szabványok alkalmazását, a világosságot. Sokat ront egy amúgy fantáziadús, jól megalkotott terven az áttekinthetőség hiánya. Másrészt az önfegyelemről, a túlzott fantázia korlátozásáról van szó. Arról, hogy a tervező ne magának, hanem egy közösségnek alkossa az adatmodellt.

Ebben a fejezetben a tipikus tervezési hibákat fogjuk kimutatni. Külön kitérünk az idővel kapcsolatos ismeretek modellezésének a nehézségeire. A fejezet végén a sablonok titkára irányítjuk a figyelmet.

Az adatbázistervezés iránt érdeklődők mindig a normálformák, a matematikailag megfogható szerkezetek, a mechanikusan gyakorolható eljárások után faggatóznak. Holott az adatmodellezés titka egészen másban rejlik: az egészséges és fegyelmezett fantáziában. Szobrász akad ezernyi - viszont Michelangelo csak egy van...

19.2 Általános tervezési problémák

19.2.1 Név-variációk

A mindennapi életben sűrűn használjuk a „fogalmam sincs” fordulatot. A fogalmakat az adatmodellezésben (is) nevekkel jelöljük. Ezért nem véletlen, hogy pont a neveknél kezdjük a tervezési bajok ismertetését. Ugyanis a tervezőnek remélhetőleg az a szándéka, hogy aki a modellt áttekinti, annak legyen fogalma arról, hogy mit is tartalmaz az adatbázis.

Az adatmodellt alkotó egyed-, tulajdonság- és kapcsolattípusok nevei nem sérthetik a ***valósághűség*** követelményét. A névnek a tényleges tartalomra kell utalnia, különben az adatbázisba helytelen adatok kerülhetnek. Előfordult például, hogy a Gépkocsi-minősége nevű adatnak a „piros” értéket adták. Ez nem történt volna meg, ha a tartalmat hívebben kifejező Gépkocsi-állapota megnevezést alkalmazták volna.

Gyakori tervezési hiba, hogy a fogalmi és a logikai tervet összetévesztve - vagy csak lustaságból - a tervező rövidített neveket és/vagy ***névkódokat*** alkalmaz. Ezek nemcsak azért veszélyesek, mert nem közérthetőek és így nem tükrözik helyesen a tartalmat, hanem azért is, mert - akaratlanul - homonimákhoz vezethetnek. Írjuk csak ide például a „Rendszam” nevet és áruljuk el, hogy a tervező a rövidített nevek híve. Ezek után tessék eldönteni, hogy a gépkocsi adatáról vagy a rendelés kulcsának (Rendelesszam) a rövid nevéről van-e szó. Ha pedig kocsik is és rendelések is szerepelnek a nyilvántartásainkban, akkor máris felléphet a homonima problémája.

A tapasztalatlan tervező szerint mindez nem fontos, mert hiszen az egyedtípus neve alapján úgyis következtetni lehet a tartalomra. Ez tévedés. Sokszor nem jelenik meg az egyed neve a tulajdonság mellett (pl. összetett jelentéseken) és az adatnév gyakran csak a tervező részére tűnik annyira világosnak, partnereinek már nem. Ezért az okos tervező ragaszkodik a teljes és jól kifejező megnevezésekhez.

Apropó: a fentiekben nem véletlenül írtunk Rendszamot. Egy további modellezési hibára szeretnénk felhívni a figyelmet. A tervezők sokszor elhagyják vagy helytelenül használják az ékezeteket, holott azok alkalmazása nem tilos a fogalmi adatmodellezés szintjén.

Ezért modortalanság a Rendszam név alkalmazása a Rendszám megnevezés helyett. Azt pedig szinte szégyelljük kimondani, hogy a nevekben be kell tartani a magyar ***helyesírás*** szabályait. A szerző látott már adatbázist, amelyben a Kocsi-suja illetve a Tarolas-heje nevű adat szerepelt...

A neveknek ***egyértelműeknek*** kell lenniük. E szabály ellenére az adatbázisokban hemzsegnek a homonimák és a szinonimák. Ezeket a figyelmetlenség mellett a hamis beállítottság is okozza. A két dolog rokon, hasonló, tehát azonos nevet adnak nekik. Például ugyanazt az Eseménykód megnevezést alkalmazzák a valamilyen változást kifejező, az egyik egyedben egy-, a másikban kétkarakteres adatra. Elfeledkezve az alapszabályról, amely szerint két tulajdonság csak akkor azonos, ha nevük mellett jelentésük *és* értékhalmazuk is megegyezik. A szinonimákkal pedig nem is törődnek mondván, hogy úgyis mindenki tudja, hogy a Törzsszám és a Tulajkód ugyanazt jelenti.

Mármost a homonimák és szinonimák szinte lehetetlenné teszik az adatmodell elemzését. Azonban nem csak ezért kell kizárni őket. Eddig nem beszéltünk az adatmodell egyik fontos kommunikációs céljáról. A modell a vállalat információs térképe, amely kiválóan használható az új belépők - fejlesztők és felhasználók - tájékoztatására. Ám miképpen igazodik el az új munkatárs olyan modellen, amely tele van kétértelmű, nem-beszélő nevekkel?

Sok gondot okoz az emlékezet-kihagyás is. Találkoztunk már olyan esettel, amelyben a tervező előre és gondosan kiszűrte a homonimákat illetve szinonimákat, ám a végső modellben mégis előfordultak ezek a jelenségek. Miként történhet ez meg? A modell tényezői sokoldalúan kapcso-

lódnak egymáshoz. Például a Rendelés-dátum nevű adat több egyedet is jellemezhet. A tervező ezért előre elhatározza, hogy a minősített Vevő-rendelés-dátum nevet fogja alkalmazni adott összefüggésekben. Majd erről megfeledkezik és a RENDELÉS egyedhez mégiscsak a minősítetlen Rendelés-dátum nevű adatot köti, amely pedig a SZÁLLÍTÁS egyedben is szerepel.

Ez az utóbbi - megtörtént - eset ismételten felhívja a figyelmet a ***minősítés*** kérdésére. A tervezők általában nem tartják be a kettős szabályt: az azonosító abszolút szerepű tételeket nem szabad, a leírókat viszont általában illik minősíteni. A minősítésnek mindig egyensúlyosnak kell lennie. Ha kétféle rendelésünk van, akkor nem alkalmazhatjuk a Rendelés-dátum és a Vevő-rendelés-dátum párost, az előbbit a szállításokra értelmezve. A minősítést teljessé kell tenni a Szállító-rendelés-dátum név bevezetésével. Az „azonosító nem minősíthető" kitételt pedig helyesen kell értelmezni. Ez nem azt jelenti, hogy a kétféle rendelés esetében ne kellene szétválasztani a Vevő-rendelésszámot és a Szállító-rendelésszámot. Hanem azt, hogy a vevők számláin a Vevő-rendelésszám helyett nem szabad alkalmazni a Vevő-számla-rendelésszám túlminősítést.

Minősíteni kell, de szerényen. A Mennyiség név éppen úgy rossz, mint a Vevői-rendelt-mennyiség-a-cikkből megnevezés. Amint arra már rámutattunk, a minősítés nem történhet a jellemzett egyed kezdőbetűjével, mert az nem beszédes és nem egyértelmű. Ezzel szemben túl fecsegő neveket sem szabad alkalmazni. Nem helyes például, ha a név valójában mondat, ami ráadásul még az értékekre is utal így: „Van-e biztosítása (I/N)?". Ez nem megnevezés, hanem társalgás. Az adatnév beszélő kell, hogy legyen, de mindent úgysem mondhat el a tételről. A tartalom teljes kifejtésére nem a név, hanem a leírás alkalmas.

Még két problémára kell felhívnunk a figyelmet. A vállalatok nem alkalmaznak ***névszabványokat*** a megnevezések formai írásmódjára. Ebből következik például, hogy az egyik tervező A-B, a másik A_B módon írja ugyanazt a nevet. Az eltérő írásmódból technikai szinonimák és homonimák származnak. Ezek is sértik az egyértelműség elvét. Mert hiába tudom én e két adatról, hogy azonos. A számítógép és a modellező rendszer éppen úgy különbözőeknek tekinti őket, mint az A-B és A-C neveket.

Végül a legcsúnyább hiba, amit a tervező elkövethet - és ezt igen sűrűn meg is teszi - az, hogy önkényes, a felhasználók teljes körével ***nem egyeztetett*** neveket használ. Különösen a tapasztalt fejlesztő esik ebbe a bűnbe. Ő már találkozott egy bizonyos fogalommal az X környezetben és azt feltételezi, hogy az a mostani Y környezetben is ugyanazt jelenti. Tudása vélt birtokában elhanyagolja a felmérést és nem egyezteti a felhasználóval a nevet. Nem vizsgálja meg, hogy a név mögötti tartalom megegyezik-e a feltételezettel. Mindez pragmatikai hibákat eredményez: a felhasználó nem a kért ismeretet kapja. Ez a probléma már átvezet bennünket a következő kérdéskörhöz.

19.2.2 Absztrakciós gondok

Az adatmodellben alkalmazott nevek fogalmakat takarnak úgy, hogy azok a valós jelenségek osztályainak felelnek meg. Az egyed-, tulajdonság- és kapcsolattípusok a valós egyedek, sajátosságok és viszonyok absztrahált tükörképei. Ezért világos, hogy a tervező nemcsak azzal követhet el hibát, hogy nem-találó (nem-egyértelmű, nem-szabványos stb.) megnevezéseket használ, hanem azáltal is, hogy helytelenül ragadja meg magának a valóságnak a lényegét. Magyarul: ***rosszul tipizál***.

Az egyik legsúlyosabb gond az inkonzekvens egyedtipizálás. Ez a probléma az egyedelőfordulások halmazával kapcsolatos és azonosítási gondokra vezet. A szerző találkozott már olyan adatmodellel, amelyben magának a modellnek az ***érvényességi köre*** volt inkonzekvens. Mondjuk a terv néhány egyedtípusa azzal a feltételezéssel készült, hogy az adatbázis az egy beruházásra vonatkozó ismereteket rögzíti, a többi pedig abban a tudatban, hogy a vállalatnak több beruházása is lehet. Mármost a beruházásra szállítások történnek. Ha csak egy beruházásról van szó, akkor a

szállítást a Szállításszám azonosítja. Ha viszont több, akkor a Beruházás-azonosító+Szállításszám összetétel szükséges a hivatkozáshoz. Világos, hogy az ilyen modell teljesen ellentmondásos és használhatatlan.

Ezt a súlyos hibát nem nehéz elkövetni. A „számítógépesítés" mai hazai helyzetére még mindig a funkciónként, részlegenként, üzletáganként stb. szeparált fejlesztés a jellemző. A tervező - jóhiszeműen - vevőknek az őt a fejlesztéssel megbízó részleg megrendelőkörét tekinti. Természetesen a másik részlegen is születik egy VEVŐ egyedtípus. Rosszabb esetben a két VEVŐ egyedtípus azonosítója más értelmű, szerkezetű, hosszú stb. tulajdonság. Jobb esetben a jelentés és a felépítés azonos, de a hossz a helyi előfordulásszámra méretezett. Így evidens, hogy amikor elkövetkezik a valódi integrálás korszaka, akkor a két VEVŐ egyedtípus összevonása kisebb-nagyobb nehézségekbe fog ütközni. Ezért nagyon fontos szabály, hogy

egy cégen belül minden fogalmat a teljes szervezet szintjén kell értelmezni.

Ha ezt a szabályt megszegjük, akkor implicit - önkéntelen és sokszor redundanciára vezető - extenzionális particionálást használunk. Vállalatunk valamennyi vevőjének az együttese egyetlen előforduláshalmaz. Ha részlegenként külön VEVŐ egyedtípusunk van, akkor ezt az együttest alhalmazokra bontjuk. Mivel egyáltalán nem kizárt, hogy az X vevő az Y és a Z részlegnek is a kuncsaftja, ezek az alhalmazok átfedőek lesznek.

Az egyedtípus-absztrakcióval kapcsolatos gond az is, hogy a tervezők nem tudatosan alkalmazzák a specializáció és a generalizáció lehetőségeit. Adatmodelljeinkre nagyon jellemző a rejtett ***specializáció***, amely redundanciát és ellentmondást okoz. Tipikus jelenség például, hogy egy adatbázisban négy, öt vagy még több egyedtípus is tartalmazza a személyekre vonatkozó ismereteket. Ugyanaz a személy többféle minőségben is kapcsolatban állhat cégünkkel. Mindegyik viszonyban más-más ismeretek mellett közös lényegű adatok is jellemzik. Ezért a tudatos tervező készít egy közös SZEMÉLY egyedtípust és annyi altípust, amennyi speciális ismeretsorra van szükség (ld. 18.5 pont). A figyelmetlen fejlesztő viszont több egyedben szórja szét a személy ismereteit anélkül, hogy gondoskodna ezeknek az egyedtípusoknak a kapcsolódásáról.

Az ilyen álspecializáció óriási adatátfedést és inkonzisztenciát okoz. Többszörösen tároljuk a személy nevét, címét stb. Mivel esély sincs arra, hogy a többszörös példányokat egyidejűleg aktualizálják, a személy címe az egyik egyedtípusban már az új, megváltozott értéket fogja tartalmazni, a másikban pedig még a régit. Valahogy így fordulhatott elő az is, hogy a szerző címe a tavalyi telefonkönyvben még jó volt, az ideiben pedig már rosszul szerepel, holott nem is költözött el a lakásából. Világos, hogy ismereteit több egyedben tárolták és azok között nem tudták helyesen meghatározni az összefüggéseket.

Tervezőink hajlamosak a rossz ***generalizációra*** is. Tegyük fel, hogy vállalatunknak X természetes és Y jogi személy a partnere úgy, hogy az Y nagyságrendekkel kisebb, mint az X és ugyanakkor - ez lényeges momentum! - a kétféle partnerre vonatkozó ismeretek nagymértékben eltérnek egymástól. Az úgymond csak pár jogi személy miatt a tervező nem hajlandó az ekkor egyetlen helyes megoldásra, a specializációra, hanem a kétféle partnert egyetlen egyedtípusba vonja össze. Ebből azután félreértések, nem-alkalmazható (NA) adatok, elágazásokkal teletűzdelt programok stb. keletkeznek. Végeredményben a tervezők nem ismerik azt az alapszabályt, amely szerint

a specializációnak és generalizációnak sohasem az extenzió (a darabszám) az alapja.

A fentebb tárgyalt esetben a VEVŐ egyed rejtett extenzionális particionálásáról volt szó. Ha a tervező rosszul specializál vagy generalizál, akkor helytelenül rendezi el a tulajdonságtípusokat az egyedtípusok között, vagyis nem jó intenzionális particionálást alkalmaz. Például az egyetlen valós személy lényeg sajátosságait nem a célszerű módon osztogatja szét az egyedtípusok között.

Eddig az egyedek és a tulajdonságok absztrakciós problémáiról volt szó. Ilyen gondok a kapcsolattípusokkal összefüggésben is fellépnek. Ha helytelenül határozzuk meg magukat az egyedtípusokat, akkor azok viszonyai sem lesznek korrektek. Most azonban nem ezzel az általános bajjal, hanem csak a ***viszony fokának*** a helytelen bemérésével kívánunk foglalkozni. Az óvatlan tervező sok hibát követhet el ezen a téren. Hiszen már láthattuk, hogy az 1:1, 1:N és M:N fokú viszonyokat - illetve ezeken belül is a homogéneket és inhomogéneket - eltérő módon kell modellezni. Tehát nem mindig könnyű megtalálni a helyes megoldást.

Ezen a téren a tervező két hibát követhet el: vagy alul-, vagy túlbecsüli a viszony fokát. Az utóbbi esetben a „hátha többszörössé válik az összefüggés" alapon feleslegesen kezelendő, az elérési utat növelő kapcsolóegyedet határoz meg másik kettő között. Az előbbi esetben nem figyel arra, hogy az általános hierarchikus viszony alól akadnak ***kivételek*** is. „Minden" szerződés egy ügyfélhez kapcsolódik, „de" akadnak olyanok is, amelyek többhöz kötődnek. Ez a „de" általában éppen elég arra, hogy a tervezett 1:N fokú kapcsolat helyett egy harmadik egyedet is involváló M:N fokú viszonyt kelljen építeni. A kivételek kezdeti elhanyagolása oda vezet, hogy az adatmodellt hamarosan át kell alakítani.

Az egyik legsúlyosabb tervezési gond a fantázia hiánya és ami azzal párosul, a könnyű, a ***mechanikus megoldások*** keresése. Egy példasorozattal szemléltetjük mondanivalónkat. Vevőkről és az általuk feladott megrendelésekről van szó. Tehát annyi bizonyos, hogy lesz VEVŐ és RENDELÉS egyedünk. A továbbiakban a kettő viszonyát fogjuk firtatni, rámutatva a rossz és a jó absztrakciók lehetőségeire.

1.változat: Tegyük fel, hogy egy-egy rendelést mindig csak egyetlen vevő ad fel. A szokásos helyzetnek megfelelő modellrészletet a 19.1 példa mutatja. (Ugyanerre a megoldásra jut - tévesen - az a tervező is, aki elfeledkezik arról a pár kivételről, amelyben a rendelés mégiscsak több vevőhöz kötődik.)

19.1 példa

VEVŐ	(***Vevőkód***, ...)
RENDELÉS	(***Rendelésszám***, ..., *Vevőkód*)

2.változat: Számos olyan rendelésünk van, amelyben több megrendelő érdekelt. Ezért a 19.2 példának megfelelő modellrészletet tervezzük. (Ezt teszi - tévesen - az aggályos tervező is, aki tudja, hogy egy rendelésben mindig csak egy vevő érdekelt, de úgymond felkészül a több vevő lehetőségére is.)

19.2 példa

VEVŐ	(***Vevőkód***, ...)
RENDELÉS	(***Rendelésszám***, ...)
VEVŐ/RENDELÉS	(***Vevőkód+Rendelésszám***, ...)

A gondatlan és az aggályos tervező esetétől eltekintve mindkét megoldás kézenfekvő, egyszerű, mechanikus. Nem is ezek miatt fogtunk a példasorozatba.

3.változat: Most tegyük fel, hogy ezer megrendelésből mindig csak egy akad, amelyben pontosan két vevő érdekelt. Az összes többi egy megrendelőhöz kötődik. A modellezés mechanikus elvei szerint ekkor már a kapcsolat M:N fokú, tehát a 19.2 példa megoldása alkalmazandó. A tervező nem is tépelődik. Bár fáj a szíve a 999 felesleges VEVŐ/RENDELÉS egyedelőfordulás és a plusz hozzáférések miatt, nincs mit tennie.

A mechanikus tervező elfelejti, hogy nem szabad zárt modellrészletekben gondolkodni. Mindig meg kell vizsgálni a teljes szemantikai hátteret. Mert az adott helyzetben két alváltozat is lehetséges.

Tegyük fel, hogy a rendelésekben együtt fellépő partnerek önálló vevőkként a részünkre sohasem fontosak. A rendelésre a vevő neveként az van írva, hogy „A és B", de sem „A", sem „B" amúgy nem partnerünk. A két megnevezett címe, számlaszáma stb. azonos. Például a két partner házaspár. Ha holtbiztos, hogy „A" és „B" nem lép velünk külön-külön is kapcsolatba, akkor az „A és B" valójában „A meg B". Egyetlen lényeg, egyetlen partner. Tehát a 19.1 példa megoldása alkalmazható.

Más a helyzet akkor, ha a rendelésre „A és B" van írva és egyikük vagy másikuk már létező vagy lehetséges partnerünk. Ilyenkor a tapasztalatlan tervező többszörös összefüggést fedez fel és elhamarkodottan a 19.2 példa megoldásához fordul. Nem biztos, hogy helyesen teszi. Az a példa azt sugallja, hogy a rendelésben „A" és „B" külön-külön érdekelt. Pedig feltehetően az a valóság, hogy nekünk a párossal van dolgunk. Már többször felhívtuk a figyelmet a „meg" és az „és" különbségére. A magyarban a két szót sokszor pongyolán - felcserélhetően - használjuk. A modellezésben ezt nem tehetjük meg.

Ha egy rendelést „A és B" együtt ad fel, akkor az „A és B" külön partner. Mivel a rájuk külön vonatkozó ismereteket nem lenne célszerű megismételni, a családfa szerkezethez fordulunk. A helyes megoldást a 19.3 példa mutatja.

19.3 példa

VEVŐ	(***Vevőkód***, ...)
RENDELÉS	(***Rendelésszám***, ..., *Vevőkód*)
PÁROS	(***Vevőkód-1+Vevőkód-2***, ...)

A VEVŐ egyedben van „A", „B" no meg „C" kódú előfordulás. Amikor a RENDELÉS-t a páros adja fel, akkor abban a „C" érték jelenik meg. Ezzel egyidejűleg a PÁROS egyedben feltűnik a „C+A" és a „C+B" sor, amely mutatja, hogy kik alkotják a „C" rendelés vevőpárosát. Ez a megoldás működik. Valósághű és nem kényszeríti ki 999 esetben a kapcsoló egyedtípus (VEVŐ/RENDELÉS) nem is igazán kifejező egyedének az alkalmazását.

4.változat: A példán még csavarhatunk egyet. Tegyük fel, hogy a rendelésben az első helyen aláíró tartozik nagyobb felelősséggel. Mondjuk ő a fizető, a másik pedig a kezes. A mechanikusan gondolkodó tervező ekkor a 19.4 példa szerkezetét alkalmazná, amelyben a Felelősség adat utalna a fizető/kezes jellegre.

19.4 példa

VEVŐ	(***Vevőkód***, ...)
RENDELÉS	(***Rendelésszám***, ...)
VEVŐ/RENDELÉS	(***Vevőkód+Rendelésszám***, ..., Felelősség)

Ez a változat két problémával is jár. Egyrészt 999 rendelésnél még mindig feleslegesen alkalmazzuk a kapcsoló egyedtípust, mert a rendelésnek csak egy vevője van. Ráadásul a Felelősség adatot is szükségtelenül töltjük ki ezek esetében. Másrészt a többszörös vevőknél programmal kell kontrollálnunk, hogy pontosan csak egy fizető és egy kezes vevő kapcsolódjék a rendeléshez. Ezért a fentinél célszerűbb a 19.5 példa szerkezete.

19.5 példa

VEVŐ	(***Vevőkód***, ...)
RENDELÉS	(***Rendelésszám***, ..., *Fizető-vevőkód, Kezes-vevőkód*)

Ebben a megoldásban a RENDELÉS egyed többszörös kapcsolattal (ld. 18.2 pont) kötődik a VEVŐ egyedhez a két minősítő szerepneven keresztül, amelyek két dolgot mutatnak. Egyrészt mindkettő valós vevőre hivatkozik. Másrészt elárulják, hogy nem egyenrangú párosról, nem egy rejtett harmadik partnerről van szó.

Persze a 19.3 és 19.5 példa megoldása nem tökéletes. Az előbbi esetében utalni kellett volna arra a rendelésben, hogy vigyázat, itt párosról van szó. Az utóbbinál pedig hiányzik egy megjelölő adat, amely két dolgot mondana el. Egyrészt azt, hogy a Kezes-vevőkódot nem kell kitölteni, ha egyetlen vevő adta fel a megrendelést. Másrészt ezzel összefüggésben azt, hogy a fizető egyben kezes is. Tehát mindkét példa RENDELÉS egyedében fel kellett volna tüntetnünk a Vevőszám vagy hasonló nevű tulajdonságot.

Még egy utolsó, nem lényegtelen megjegyzés, amely az absztrakcióval kapcsolatos és csinosan összefoglalja az egyed, tulajdonság és kapcsolat együttes elgondolásának a szükségességét. Tegyük fel, hogy a tervező a Vevőnév tulajdonságban megengedi a „Kovács Lajos és Kovács Lajosné" tartalmat. Ez annyiban különbözik a „Szabó Ferenc és társa" értéktől, hogy az utóbbi egy céget, egy valós lényeget, míg az előbbi két külön egyedet jelöl. Tehát az „és" szócska használata teljesen mást jelent a két esetben.

Ha „Kovács Lajos" és „Kovács Lajosné" külön-külön is kuncsaftunk, akkor többszörös hibát követ el a tervező. Két különböző lényeget egyetlen tulajdonságba sűrít. Egyetlen azonosítót ad nekik anélkül, hogy utalna a valódi azonosító párosra (ld. 19.3 példa). Az összetett név miatt a rendelés nem lesz kapcsolható sem Kovács úrhoz, sem feleségéhez. Tehát vállalatunk sohasem tudja meg, hogy miképpen áll anyagilag Kovács úrral és miként a nejével. A karbantarthatóság kérdéséről pedig ne is beszéljünk. Ha az adatbázisban valahol szerepel a „Kovács Lajos és Asztalos Ica" megnevezés és Ica férjhez megy Lajoshoz, akkor az új „Kovács Lajosné" nevet sohasem fogjuk erre módosítani a konkatenált névben. Ezért a tervezőknek be kellene tartaniuk az alapvető szabályt, amely szerint

egy jelenség = egy fogalom = egy modellezési egység = egy név.

19.2.3 A körülírás hiánya

Az előző fejezet végén utaltunk arra, hogy mennyire fontos lenne az adatmodell tényezőinek a sokoldalú ***osztályozása***. Akkor is, ha az adatkezelőnk szemantikailag szegényes és nem támogatja az egyedek, tulajdonságok és kapcsolatok különböző besorolásait. Annak dacára, hogy az osztályozás a majdani validálások miatt a saját érdekeit szolgálná, a tervező nem fogékony erre a modellezési részműveletre. Nem a fogalmi modellben - ott volna a helye -, hanem majd csak a programspecifikációban és körülményesen magyarázza meg, ha nem felejti el, az egyedtípus jellegét. Pedig mennyivel egyszerűbb lenne egyetlen egyszer megegyezni abban, hogy például a családfa jellegű egyedtípus azt jelenti, hogy... és azt úgy kell validálni, hogy..., majd a modellben csak annyit mondani, hogy ez az egyedtípus családfa osztályú...

A tervezők nem hajlamosak az adatmodell tényezőinek a szöveges ***leírására*** sem, noha az effajta tájékoztatás igen hasznos lenne a felhasználókkal való egyeztetés során, az új belépők felvilágosítására stb.

Persze mindennek az a valódi oka, hogy a tervezők nem fogalmi szinten akarnak modellezni, hanem azonnal a kezelőhöz igazított adatbázisterveket kívánnak készíteni. Ezért vesznek el például az ***adattípus*** tényezőjét illetően is. Csakis a támogatott fizikai adattípusokban gondolkodnak. A kezelési időtöbblet miatt ritkán határoznak meg saját adattípusokat. A modern kezelők lehetővé teszik a saját típus kijelölését, de azok alkalmazásának hatékonysági ára van. Emiatt a tervező elfeledkezik a típus és a validálás kapcsolatáról.

Az ismeret szemantikai jelentését a jelek hordozzák. Nagyon sokszor előfordul, hogy csak ránézünk egy adatra és máris tudjuk, hogy azt hová tegyük. Mi jut eszébe az olvasónak erről a jelsorról: „(036) 1-1168-119”? Ugye, hogy a telefonszám? És mi akadálya van annak, hogy fogalmi szinten meghatározzunk egy ilyen adattípust? Semmi. Még ha fizikailag nem is így fogjuk tárolni az adatot, logikailag akkor is így kell majd kezelnünk. Ezért bár az adatábrázolás nem tisztán fogalmi szintű tényező, élni kell annak lehetőségeivel.

Mindez összefügg egy másik tényezővel. Az adatábrázolásnak része az ***adathossz*** is. A tervezők gyakori hibája, hogy a hosszt alul- vagy túlméretezik. Netán szokások rabjaivá válnak és a méreteket sablonosan határozzák meg. Például a Településnév hosszát standard módon 40 karakteresre veszik fel, holott a leghosszabb magyar településnév is csak 24 jelből áll. Máskor meg a sablonokkal szemben éppen hogy ad-hoc méreteket jelölnek ki.

Az olvasó most joggal kérdezi, hogy mi köze van az adathossznak a fogalmi adatmodellezéshez? Az első ránézésre nem sok. Valójában azonban a hossz nem csak fizikai adatszerkezeti tényező. A zavaros ábrázolási méretek a fogalmi terv tisztázatlanságáról árulkodnak. Nézzünk csak három példát.

Első példa: Az X rendszerben a lakcím ismeret része az Irányítószám. A tervező az adatot négy numerikus jelként határozta meg. Nem tudta, hogy a magyar irányítószám ma már öt jelből áll és az ötödik jel karakteres. Miért fogalmi szintű kérdés ez? Azért, mert a tervezőnek fogalma sem volt arról, hogy vannak az első négy karakterben azonos módon jelölt, de azokon belül az ötödik jellel megkülönböztetett irányító-részkörzetek.

Második példa: A tervező alkalmazza a Változás-módja és a Módosítás-oka adatokat. Az egyiket 27, a másikat 45 hosszúra definiálja. Majd kiderül, hogy egyes változási módok és módosítási okok egybeesnek. Tehát a két adatféle redundáns. Egyazon valós jelenség tükrözésére egyféle hosszat illenék alkalmazni. Ha a mód és az ok egybeesik, de az ábrázolási hossz eltér, akkor világos, hogy a tervezőnek nem volt tiszta fogalma e jelenségekről.

Harmadik példa: A tervezők spórolni kívánnak a méretekkel. Ezért több fogalmat is ugyanabba a tulajdonságba gyömöszölnek. Például a Cikkszám osztályozó kód csak 6 karakteres. Ha tudjuk, hogy a Cikkszámban tükrözött osztályozási ismérvek közül a végtermékek X, az alapanyag Y, a megmunkálás (technológia) Z változatot jelentenek és az X+Y+Z értéke milliónál nagyobb, akkor evidens, hogy a Cikkszám fogalmilag zavaros. A tervező az egyes kódpozíciók jelentését összekeverte. Különben a milliós tétel azonosítása lehetetlen lett volna 6 jelben.

A tervezők legnagyobb hibája, hogy a ***validálásokat*** - az ismereti korlátokat - nem a fogalmi modellben, hanem csak a programspecifikáció szintjén adják meg. Ebből szükségszerűen következik a redundancia és az ellentmondás. Csakis egyszer, egyetlen helyen - az adatmodellben - kellene megfogalmazni például azt, hogy a Személynév tulajdonság olyan tulajdonnév, amelynek minden tagját egyetlen üres választja el; elölről nem lehet üres; a név tagjai nagybetűvel kezdődnek; a titulus (dr., özv., ifj. stb.) a név végére kerüljön és így tovább...

Most megint felvethető a kérdés, hogy mi köze mindennek a fogalmi modellhez. Nagyon is sok. A személyek tulajdonneve egy bevett fogalom. Vezetéknévből, keresztnévből és opcionális titulusból áll. Ha az adatmodell mindezt nem tükrözi, nem egyetlen helyen jelöli ki a Személynév ábrázolását és validálását, akkor elkönyvelhetjük, hogy a tervezőnek fogalma sincs arról, hogy mit is jelent a Személynév.

19.3 A „CSAK” szindróma

A csoportokról (CS), azonosítókról (A) és a kódokról (K) lesz szó ebben a pontban. Ezek a tényezők annyira bonyolultan függenek össze egymással, hogy - bevalljuk - értelmes rend szerinti tárgyalásukra nem találtunk megfelelő módot. Hiszen vannak kódolt és csoportos azonosítók. Ha

ilyet lát az ember és megkérdezi a tervezőt, hogy miért is alakította úgy ki az elsődleges kulcsot vagy leíró tételt, akkor egyetlen választ kap - CSAK.

A problémák feltárását kezdjük a csoportoknál. Az implicit - rejtett - összetételekkel van baj. Vegyük például a gépkocsik Alvázszám tulajdonságát. Ennek első néhány jele állítólag a kocsi típusára utal. Ezek után mire véljük, hogy a KOCSI egyedtípusban szerepel egy Kocsitípus nevű tulajdonság is? Ez az ismeret nyilvánvalóan kétszeres, redundáns. Mi több, ellentmondások forrása. Mert mi garantálja, hogy az adat kezelését végző személy a Kocsitípus adatot és az Alvázszám első pár karakterét mindig egymásnak megfelelő módon fogja kitölteni? Miután, természetesen, az Alvázszám csoportba rejtett típus és a Kocsitípus kódolása teljesen eltérő... Tehát az első probléma az ***implicit csoportokkal*** kapcsolatos.

Ezzel a bajjal számtalanszor találkozhatunk a rosszul kialakított mesterséges azonosítók kapcsán, amelyek burkoltan csoportos szerkezetűek. Mivel a relációs rendszerekben nem lehet csoportot megadni, a tervező a modellben sehol sem árulja el például azt, hogy a Rendelésszám valójában a Vevőkód és egy sorszám együttese. Mi több, a Számlaszám is a Vevőkódból és egy sorszámból tevődik össze. Mivel a csoport nem bontható ki, a tervező - a hivatkozási integritás jegyében - külön is felveszi a Vevőkód-ot a RENDELÉS, a Rendelésszám tételt a SZÁMLA egyedben. Amivel a rejtett Vevőkód érték megtöbbszöröződik. Természetesen a tervező magát a Vevőkód-ot is úgy alakítja ki, hogy annak értéke - minden modellezési elv dacára - idővel változhat. Annak az esélye, hogy ilyen változás után a Vevőkódot a Rendelésszámban és a Számlaszám-ban is karbantartják, nulla.

A ***mesterséges azonosító csoportok*** egyébként is sok gondot okoznak. Az előszeretettel alkalmazott relatív sorszámokról van szó. Az összetett A+B azonosító úgy épül fel, hogy az A valamilyen másik egyedre utal, míg a B relatív sorszám. Tipikus példa a RENDELÉSTÉTEL (***Rendelésszám+Tételsorszám***, ...) kulcsa. Mi az ilyenféle azonosítókkal a baj? A relatív sorszám nem fogalmi tényező, hanem papírhoz kötött fizikai megjelölés. Ha az adott rendelés ötödik tételét felmondják (nem kérik), akkor az azonosítónak lőttek. Mert nem fogják az összes tételazonosítót „mínusz egy" értékkel átdolgozni, viszont ugyanakkor már az sem lesz igaz, hogy a hatodik tételsor hatodik, hiszen - a felmondás miatt - már valójában csak ötödik. Ezért ***relatív sorszámot*** - ami nem fogalmi tényező - nem igazán illik az azonosító részeként használni.

Az előbbi kitételt egy további példával igazoljuk. Tegyük fel, hogy szerződéseinket egy Területi-kód és egy Relatív-sorszám azonosítja. Az előbbi adat arra utal, hogy milyen körzetben kötötték a szerződést. Az utóbbi arra, hogy hányadik megkötött ügyletről van szó. Ez a csoportos, területre kódolt, sorszámot tartalmazó kulcs kétszeresen is rossz. Ha a szerződést átadják az X körzetből az Y területre, akkor nemcsak a Területi-kód változik, ha-nem a Relatív-sorszám is módosul. Mert a szerződés az X egységben az ötödik volt, az Y körzetben viszont már a huszonegyedik lesz.

Jó azonosítót kreálni nem kis művészet. A legjobb elsődleges kulcs a csoportoktól és kódolásoktól mentes ***abszolút sorszám***. A General Motors (GM) 1975-ös esete jól bizonyítja ezt a kitételt. Akkoriban a GM ötven millió dollárt fizetett a Brish Co. kódolási szakértő vállalatnak azért, hogy a mintegy 7-800 ezer alkatrésznek jól megfogható osztályozó azonosítókódot készítsen. A Brish csodákat művelt. Mindössze nyolc karakterben azonosította a GM valamennyi alkatrészét. A tíz centiméter vastag kódkönyvből mindenki pár másodperc alatt kikereshette, hogy mi az alumíniumból készített, x-dimenziós, hajlított rögzítőelem kódja, szemben az acélból készült, ugyanolyan méretű, hegesztett rögzítőelem kulcsával.

A GM ezt a kódot, amit több száz millió dollárért vezetett be, hamarosan kidobta, mint azonosítót. A technológia, a forma, az anyag változott. Ezért az instabil kulcsot állandóan és rengeteg helyen át kellett írni. A GM már akkor megtudta, hogy csoportos osztályozó kódot legfeljebb csak leíró tulajdonságként szabad alkalmazni. Az azonosítás terén pedig átváltott a legegyszerűbb megoldásra: a sima abszolút sorszámra.

A területileg osztott adatbázisok esetében a tervezők azért nem vállalják az ***abszolút sorszám*** azonosítót, mert úgymond annak kiadása és egyediségének az ellenőrzése nehézkes. Ha vállalatunk területenként köt szerződéseket a partnereivel, de nincs on-line adatbevitel egyetlen központi számítógépre, akkor valóban nehéznek látszik az azonosítás kérdése. Mi garantálja, hogy a Kecskeméten 10 óra 50 perckor rögzített szerződés 1234567 sorszáma után a Kőszegen 10 óra 51 perckor megkötött ügylet a 1234568 sorszámot kapja?

Semmi. És miért kellene azt a kulcsot kapnia? Az abszolút sorszám lényege az, hogy nem tükröz semmilyen ismeretet. Sem helyet, sem időt, sem mást. Csak az egyediség az egyetlen kritériuma. Ezért területileg osztott adatbevitel esetében nyugodtan ki lehet osztani az azonosító tartományait az X, Y és ... Z körzetek között. A lényeg az, hogy eszébe ne jusson valakinek ezt az előzetes kiosztást területi behatároló ismeretnek tekinteni. Az azonosító felszabdalása - most oda adtuk ki az xxx-yyy tartományt - csak technikai manőver. A Szegeden felhasznált 3456789 sorszám olyan szerződést azonosít, amely később Debreceni illetőségű lesz, ugyanezzel a számmal.

Ja, hogy a sorszámok sorozata ettől „lyukas" lesz? Mert Egerben még nem fogyott el a kvóta, miközben Pécsett már újabb tartományt kell kiosztani? Kit érdekel! Az abszolút sorszámnak nem az a lényege, hogy zárt sorrendet alkosson. Hanem az, hogy biztosan egyedi legyen. Ez pedig minimális szervezéssel biztosítható. A tényleges adatbázison belül minden tétel úgyis sajátos belső kulcsot kap, amelynek az égadta világon semmi köze sincs a külső, a felhasználók által alkalmazott azonosítóhoz. Csak a régi COBOL-világban épült a belső fizikai kulcs a külső logikaira. Így az abszolút külső sorszám a megfelelő megoldás.

A kódolt csoportos azonosítók egyéb veszélyeket is hordoznak. Hierarchikusra szűkítik le a valójában hálós viszonyokat. Itt van ez a szerelvény, amely az „xxx" azonosítójú termék része. Nosza, adjuk neki az „xxxa" kulcsot. Csakhogy a szerelvény az „yyy" azonosítójú termékbe is beépül. Ezért a mérnökök annak az „yyyb" rajzszámot jelölik ki. Mi a következmény? Az „xxx" termék gyártásához Q, az „yyy" termeléséhez W darab azonos szerelvény tartalékolására van szükség. Az együttes igény ennél nyilvánvalóan kisebb. Valójában a cégnek csak V < Q+W alkatrész szükséges. Mivel a szerelvénynek nincs egyetlen - sorszám jellegű - kulcsa, az azonosítója a végtermékekhez kötődik (xxxa és yyyb), a cikk kétféle jelenségként jelenik meg az adatbázisban - és felesleges készlete igen drága lesz.

A szerző 1965-ben a Csepel Autógyár raktárosaként élte meg ezt a helyzetet. A normália üzemből X cikkszámon Q darab, Y cikkszámon Z darab tömítőgyűrűt szerzett be. „Hát nem láttya, hoty utyanasz. Moszt miként sütjük el. Itt fok állni egy évig." - mondta a szerb-német Ali bácsi. Mondta? Lehordott a sárga földig. Igaza volt. Tőle tanultam meg, hogy a cikkszám - az azonosítás - nem játék. A zseni mérnökök, a tudós közgazdák semmit sem sejtettek az információ lényegéről. Ali bácsi, aki törte a magyart és hat elemit sem végzett, azonnal tudta a dörgést: Ugyanasz - bármit isz mondanak odafenn asz urak. Moszt nekemnek kell aztatat elssózniam. Éf minek hoszta itten a nyakamomra eszt megind, Béla?".

Miután Ali bácsi rendesen eligazított az „azonos jelenség - több azonosító" csacskaságaiban, már csak annyi marad hátra, hogy az összetett kulcsokról, a ***konkatenált azonosítókról*** emlékezzünk meg. A hálósan kapcsoló egyedtípusok kulcsáról van szó. Ha a Rendelésszám azonosította RENDELÉS és a Cikkszám kulcsú CIKK hálósan - M:N fokú viszonnyal - kapcsolódik, akkor evidens, hogy a kapcsoló RENDELÉSTÉTEL kulcsa a két viszonyban álló egyed azonosítójának a konkatenációja (Rendelésszám+Cikkszám).

Ez eddig elvileg rendben is lenne. Csakhogy a gyakorlatban előfordulnak olyan esetek, amelyekben az A+B összetett kulcsnak a B része ismeretlen. Például nem ismerhető fel, hogy a szerződésben (A) ki az ügyfél (B). A papír elmosódott, rosszul kitöltött. A tervező mégis ragaszkodik a homályos ismeret beviteléhez. Ezért az ilyen esetekben a B-nek mesterséges tartalmat ad. Valós ügyfelet jelöl az „xxx..." kezdetű kulcs, ismeretlent az „yyy..." karaktersorral kezdődő.

Az effajta megoldások aláássák az adatbázis hitelességét. Mert vagy kikötjük, hogy minden szerződésnek létező ügyfélhez kell kapcsolódnia, amely esetben a nemlétező ügyfelet is külön mesterséges egyedként kell bevinni. Tehát lesz egy „yyy...” azonosítójú, valójában nem is létező „ügyfelünk”. Vagy azt mondjuk, hogy a SZERZŐDÉS/ÜGYFÉL egyed viszonya az ÜGYFÉL felé opcionális. Nem kötelező. Ekkor viszont azt sem tudjuk ellenőriztetni automatikusan, hogy a jól megfogalmazott szerződések ügyfelei ténylegesen léteznek-e az adatbázisban.

19.4 Egyéb csacskaságok

Néha találkozhatunk olyan adatbázistervvel, amelyben egy egyedtípus egy adott tulajdonságtípusra mindig ugyanazt az értéket veszi fel. Például egy könyvvel kapcsolatosan a személy szerepe mindig „lektor”. Ez olyankor fordulhat elő, ha az adatbázisban létezik a szerepre kihegyezett egyedtípus, például LEKTORÁLÁS. Azt a tulajdonságot, amely az adott egyed lényegéből következik és ezért minden előfordulásra azonos értékű, ***generikus tulajdonság-típusnak*** nevezzük. Természetesen ilyenkor a generikus tulajdonság tartalma redundáns és azért ezt a tételt el kell hagyni az egyedtípusból.

Miképpen fordulhat elő ez a hiba? Általában véve az implicit - nemtudatos – specializációnál kell számolni ezzel a helyzettel. A tervező létrehozza a SZEMÉLY egyed implicit SZERZŐ és LEKTOR altípusait, majd azokban felejti a személy szerepét (szerző, lektor), ami generikus tulajdonsággá válik a specializáció után. Néha az is előfordul, hogy összetett, például A+B azonosítójú egyedtípushoz kívánnak kapcsolni egy alárendeltet, amelynek mondjuk A+B+C a kulcsa. Csak éppen úgy, hogy a C értéke mindig azonos. Jelentse az A+B a könyvet és a személyt, a C pedig a személynek a könyvvel kapcsolatos szerepét. Ha a KÖNYV/SZEMÉLY egyed alá kívánjuk helyezni a LEKTORÁLÁS egyedet, akkor a C értéke mindig „lektor” lesz. Mivel egy könyvnek több lektora is lehet és egy könyvvel kapcsolatosan ugyanaz a személy több funkciót is betölthet (író, rajzoló), az alsóbb szintű egyedben feltalált generikus tulajdonság sokszor a felsőbb szintű egyed rossz tervezettségére utal. A KÖNYV/SZEMÉLY és a LEKTORÁLÁS együttesen hibás. Az alsóbb szintű egyed generikus tulajdonságának (szerep) mindig a felsőbb szintű egyedben, nem ritkán éppen annak azonosítójában van a helye.

Felhívjuk a figyelmet arra, hogy a generikus tulajdonság néha opcionális is lehet. Például a lektor mindig az x részlegről kerül ki *vagy* nincs részlege, az nem értelmezhető (külsős és a külső szervezet bennünket nem érdekel). Ettől még maga a tulajdonság generikus. Mivel a nem-értelmezhető (NA) értéknek nincs haszna, az értelmezhető érték pedig mindig ugyanaz, az ilyen esetekben is el kell hagyni a generikus tulajdonságot.

Az explicit csoportokkal már az előző pontban foglalkoztunk. Most az ***implicit csoportokról*** kell néhány újabb szót ejtenünk. A csoport implicit, ha az eleminek mutatott tulajdonság értékei valójában két vagy több tagból állnak. Ilyenkor általában az első tag(ok) tipizálnak és az utolsó az, ami a típuson belül szétválasztja a jelenségeket. Például: a Szervezeti-egység-kód első pár jele az egység típusát mutatja, a többi pedig a típuson belül konkrétan behatárolja az egységet.

Az implicit csoport két problémát okozhat. Az egyik az, hogy nem ad módot az explicit kapcsolat meghatározására. A másik az, hogy az előző ok miatt a csoport első - tipizáló - részét külön tulajdonságtípusként is feltüntetik, amivel persze hatalmas redundanciát okoznak. Ez a fajta redundancia - a generikus tulajdonsághoz hasonlóan - nem fedezhető fel és nem szüntethető meg a szokásos normalizálási eljárásokkal.

Az implicit csoport egyik fajtája a ***vektor***. Olyan tulajdonság, amely előre meghatározott számú részből áll úgy, hogy mindegyik összetevő értelme azonos. Például a tervező a RENDE-

LÉS egyedben alkalmaz egy Feltételek nevű tulajdonságot, amely 5 különböző feltételkódot tartalmaz egymás mellett a rendelés mineműségének a függvényében.

Az ilyen vektor nem más, mint rejtett ismétlődő csoport annak minden problémájával együtt. Például a hatodik feltétel esetében át kell alakítani az adatbázist. Ha az ismétlődések száma biztosan nem fog változni, akkor is modellezési bajokkal kell számolni. Tegyük fel, hogy létezik a FELTÉTEL (***Feltételkód***, ..., Feltételnév) egyedünk. Az implicit vektor esetében nem tudjuk ehhez kapcsolni a RENDELÉS egyedet többszörös kapcsolattal, mert hiszen nem látszódnak maguk a kapcsoló tulajdonságok. Ezért a hivatkozási integritást procedurálisan kell ellenőrizni. Emiatt az azonosító abszolút szerepű tulajdonságok esetében a vektor nem megengedett fogalmi szintű tényező.

Sok adatbázisban jelent problémát a ***kettős azonosítás***. Itt több különböző bajról kell beszélnünk. Általában az adatbázisok konverziója - kezdeti feltöltése vagy máshonnan való áttöltése - során szoktak alkalmazni a konverzió menetét ellenőrző sajátos azonosítókat. Ilyen-és-ilyen számmal már bevittük azt a tételt, hogy ... Azután a tervező elfeledkezik arról, hogy ezt a ***technikai*** adatot csak átmenetinek szánta és a mesterséges adat „élni kezd”, vagyis a konverzió befejezése után is alkalmazzák. A kettős azonosításnak ez a módja nyilván redundancia, amely ellentmondásokra is vezethet. Ezért kerülendő.

A 14.7 pontban beszéltünk az alternáló kulcsokról, amelyek a kettős azonosítás egy speciális esetét jelentik. Nem akarjuk azt mondani, hogy az alternáló kulcsok feleslegesek, mert vannak olyan speciális helyzetek, amikor használhatók. (Persze csakis a valódi alternáló kulcsokról beszélünk. Ha valamelyik, az egyedelőfordulást illetően egyedi értékű tulajdonság tartalma lehet ismeretlen is, akkor az egyszeres leíró tulajdonság, de semmiképpen sem kulcs.) Alternáló azonosítók felléphetnek a specializáció esetében. Az orvos is személy, tehát van Személyiszám (vagy azt pótló) adata és ugyanakkor van Orvoskód nevű egyedi azonosítója is. Bár ez a helyzet nem szerencsés, elképzelhető, hogy a személy azonosítójánál jóval rövidebb Orvoskód használata praktikus.

Az alternáló kulcsok akkor okoznak igazi problémákat, ha alternáló kapcsolókként lépnek fel, vagyis nemcsak az alapegyedben (pl. ORVOS) szerepelnek együttesen, hanem annak valamilyen alárendeltjében is (pl. ORVOS/RENDELÉS). Ezt a helyzetet feltétlenül kerülni kell. Azért, mert többszörösen mutatja ugyanazt az egy kapcsolatot. Ha mindkét adatot kitöltik az alárendeltben, akkor ellenőrizni kellene, hogy a páros értéke megegyezik-e a fölérendeltben lévő párosával. Általában is megfontolandó, hogy az A és B alternáló páros esetében az egyik tulajdonság (B) csak az alapegyedben szerepeljen és minden más egyedben csakis a másik tulajdonságot (A) szerepeltessük kapcsolóként. Könnyen zavarokra vezethet, ha a modell néhány egyedében az egyik, a további egyedeiben pedig a másik tulajdonságot alkalmazzuk a kapcsolót hordozó tételként.

Még a feledékenységről és az abból fakadó inkonzisztenciáról kell szólnunk. A tervezők hajlamosak elfeledkezni arról, hogy az adatmodell nem rekordképek özöne és ezért a ***kapcsolattípusokat*** is tartalmaznia kell. Ha viszont készítenek kapcsolattípus-listát, az néha nem felel meg az egyedtípusokban rejtőző kapcsoló tulajdonságok listájának. Magyarul: explicit kapcsolatként megneveznek olyan egyedviszonyokat is, amelyeket az egyedek tulajdonságai nem támasztanak alá. Megfordítva: a listán nem tűnnek fel olyan kapcsolatok, amelyek pedig következnek az egyedek belső szerkezetéből. Az ellentmondásnak sokszor az az oka, hogy a tervező teljesen megfeledkezik a ***szerepnevek*** explicit kijelöléséről. Ha nem mondja meg, hogy a Lektor-azonosító is egy Személy-azonosító, akkor persze nem határozza meg helyesen a lektornak, mint sajátos szerepű személynek a kapcsolatait sem.

19.5 Statikus szemlélet

19.5.1 Az idő modellezése

Könyvünkben mindeddig ***statikus*** szemléletben foglalkoztunk az adatmodellel. Olyan tervezési példákat mutattunk be, amelyek a jelenségek pillanatnyi ***állapotát*** tükrözték. Pedig a jelenségek változnak. A múlt héten X.Y. még a Forint utcában lakott, viszont ma már a Pengő közben él. A statikus felfogású adatbázisban a megváltozott lakcímet aktualizáljuk, vagyis a régi tartalmat az újjal írjuk felül. Természetesen ezzel a korábbi állapotra vonatkozó ismeret elveszik. Ezért jogosan merül fel a kérdés, hogy miképpen kell és lehet megszerkeszteni az adatmodellt, ha szükségünk van a múltbeli adatokra is?

Nem véletlen, hogy ezt a kérdéskört éppen a tervezési hibákkal foglalkozó fejezetben vettük elő. Az adatbázistervek igen gyakran megsértik az időre és a kapcsolódó tényezőkre vonatkozó egyértelműségi, teljességi és minimalitási modellezési szabályokat. Mindez arra vezethető vissza, hogy a modellezők nem ismerik e tényezők elméleti összefüggéseit. Ezért ebben a pontban együtt tárgyaljuk a ***dinamikus*** adat-modell kialakításának az elveit és az azok mellőzéséből fakadó problémákat.

Maga az ***idő*** végtelen és folyamatos. A végtelenség és a folyamatosság modellezésére nincs lehetőség. Ezért el kell dönteni, hogy részünkre az idő milyen intervallumot, időközt jelent és mi számunkra az idő diszkrét mértékegysége. Az intervallumról még azt is meg kell határoznunk, hogy az zárt-e vagy nyílt-e, azaz felöleli-e a kezdő és a záró időpontot, vagy sem. Végeredményben meg kell alkotnunk az adatmodell minden tényezőjét behatároló általános IDŐ fogalmat.

Mint tudjuk, az adatmodell egyed-, tulajdonság- és kapcsolattípusok valamint az ezekre vonatkozó korlátok szervezett együttese. Ezért az idő modellezésére több, célszerűen kiválasztandó megoldás létezik. A 18.1 ábra metamodellje ötféle szerkezeti tényezőt tartalmaz. Ennek megfelelően az idő modellezésében is öt strukturális tételre kell tekintettel lennünk.

Legelső feladatunk az, hogy meghatározzuk az idő általános tulajdonságot, a feltehetően algoritmikus ellenőrzésű Idő ***értéktartományt***. Adatbázisunk az X időponttól az Y időpontig tartó zárt időintervallumba tartozó jelenségekre vonatkozóan fog adatokat tárolni úgy, hogy az idő mértékegysége a nap (vagy az óra, a perc stb.). Valahányszor egy egyedben idő jellegű tulajdonság, vagyis ***attribútum*** jelenik meg (ld. egyed-tulajdonság-viszony), az csak az általános tartományba tartozó értéket vehet fel. A tervezők nem élnek ezzel a lehetőséggel. Ezért találkozhatunk olyan adatbázissal, amelyben a SZERZŐDÉS egyed Fizetési-dátum adatában 1893-as keltezés szerepel.

Abban az esetben, ha egy attribútum az Idő által előírtnál alulról és/vagy felülről szűkebb intervallumot feltételez, a korlátozó ***szerepnév*** megoldásához folyamodhatunk (ld. tulajdonság-struktúra). Ha egy szerződésféle csak Z dátumtól köthető, akkor a Kötés-dátum az Idő alulról korlátozó szerepneve. Ugyancsak korlátként szolgáló, származtatási jellegű tulajdonság-struktúrában köthetjük ki, hogy a Szállítás-dátum értéke nagyobb/egyenlő viszonyban áll a Rendelés-dátum tartalmával. Ha az adatmodell a száz-százalékos elv dacára nem tartalmazza ezt a struktúrát, akkor persze a megvalósítás során elfeledkeznek a validálásról. Így fordulhat elő, hogy a Törlesztés-kezdete tétel tartalma korábbi keltezést mutat, mint a Hitel-megkötés-dátum értéke. Arról pedig már sokszor beszéltünk, hogy a dátumot explicit csoportos tulajdonság-struktúraként is meg kell adni.

Az adatmodellben az időt ***egyedtípusként*** is lehet tükrözni. Természetesen az IDŐ (vagy DÁTUM) szinguláris egyedtípus (ld. 18.7 pont) nem mindig kerül megvalósításra konkrét, az időpontokat egyed-előfordulásokként tartalmazó táblaként. Van előnye annak, ha mégis ilyenként alkalmazzuk. Az IDŐ (***Idő***) egyedtípus minden olyan másik egyednek a fölérendeltje, amelyben megtalálható az Idő doméjn minősítő vagy korlátozó szerepneve. Például az IDŐ a SZERZŐDÉS

kétszeres fölérendeltje, amennyiben az utóbbi egyedben található a Kötés-dátum és a Lejárat-dátum szerepnévpáros.

Amint látjuk, ebben az esetben az idő kapcsoló tulajdonságokra ***kapcsolattípusok*** épülnek, és így az idő valóban érinti az adatmodell mind az öt szerkezeti tényezőjét. A megoldás előnye pedig az, hogy automatikusan érvényesíthető a két egyed közötti hivatkozási integritás. Csak olyan tétel vihető a SZERZŐDÉS egyedbe, amelyben a dátumok tartalmai az IDŐ (***Idő***) értékeiként is léteznek. Ezt a modellezési megoldást akkor is érdemes megfontolni, ha eszünk ágában sincs megvalósítani az IDŐ egyedtípust és annak kapcsolatait.

A modell szerkezeti felépítése lesz világosabb, ami megkönnyíti az elemzést.

19.5.2 Esemény, változás, állapot

Az ***esemény*** fogalmat kétféle értelemben használjuk az adatmodellezésben. Vannak olyan jelenségek, amelyek eleve esemény jellegűek. A 12.5 ábra KÁR egyedtípusa ilyen jelenséget tükröz. Sajnálatos módon az adatkezelők nem támogatják az egyedtípusok esemény/nem-esemény típusának megfelelő minősítését sem. Pedig a kétféle egyed másképpen viselkedik.

Az adatmodellezésben nagy szerepe van az egyedek ún. ***életciklusának***. Az adatbázisok egyedei valamikor megszületnek, esetleg megváltoznak, majd idővel meghalnak. Kovács Rózsa kocsitulajdonos lett (hozzáadás); lakcíme módosult (aktualizálás); majd eladta a kocsiját (törlés). Az esemény jellegű egyedtípusokra (KÁR) nem vonatkozhat aktualizálás, mert az esemény egyszer úgy történt meg, ahogyan a dolog esett. (Ismét elmondjuk, hogy a hibajavítás csak technikai értelemben változtatás, fogalmi síkon nem az. Mert nem magával a jelenséggel történt valami.)

A fentiekből következik, hogy az ***esemény jellegű*** egyedtípusoknak mindig kell, hogy legyen egy, az esemény bekövetkeztének az időpontjára utaló, az Idő doméjn szerepneveként szolgáló illetve az IDŐ egyedtípushoz kapcsoló dátum tulajdonsága. Ennek - és az egyed összes többi tulajdonságának - a tartalma mindig egyszeres és sohasem módosulhat.

Ezzel szemben a ***nem-esemény jellegű*** egyedtípusokhoz kétféle tulajdonság kötődhet. Az egyik változhat, a másik nem. Például a reménytelen kísérletezéseket kizárva szerintünk a Személy-neme tulajdonság (férfi és nő) sohasem változhat, szemben a lakcímmel. Most ne beszéljünk arról, hogy a tervezők rengetegszer megsértik azt a szabályt, hogy egy egyed azonosító tulajdonságának - és így természetesen annak részeinek - mindig stabilnak, változtathatatlannak kellene lenniük. Azt is csak futólag említjük, hogy a szemantikával nem bőven ellátott mai adatkezelőkben nincs mód az attribútumok stabil/instabil - változtatható/nem az - minősítésére.

Nagyobb baj az, hogy a fejlesztők és a felhasználók nem gondolják át azt, hogy melyik ismeret változhat és melyik nem. Ennek legfőbb oka az, hogy a technikai javításokat egy kalap alá veszik - mivel egy rutinnal kezelik - a valós változásokkal. Ugyanaz a program képes az elírt „Frint utca” karaktersort a „Forint utca” értékre javítani, mint a „Pengő köz” új tartalomra módosítani. Mivel a fogalmi értelemben nem változtatható tulajdonság betűsorát is be lehet rosszul ütni és éppen azért azt ki kell tudni javítani, a technikai és az érdemi átírás különbsége eleve elhomályosul.

A statikus szemléletből következik, hogy a valódi események célszerű modellezésére sem kerül sor. Most találkozunk az esemény szó második értelmével. Esemény a jelenség születése, módosulása, megszűnése. Az első és az utolsó esemény nyilván csak ***egyszeres*** lehet, míg a módosulás ***többszörös***. Rengeteg hibát lehet elkövetni, ha nem figyelünk erre a különbségre.

A legcsacsibb megoldás az ***időtartam*** modellezése az ***időpontok*** helyett. Volt alkalmunk találkozni olyan adatbázissal, amelyben a személy Életkor nevű tulajdonsága szerepelt. Ez az adat minden pillanatban változik, tehát alkalmazása a nem-változó Születési-dátummal szemben „halálos bűn”. (Persze itt is mérlegelnünk kell. Például egy statisztikai felmérés esemény jellegű - soha nem változó - egyedtípusokat involvál. Kovács Rózsa az X adatgyűjtés idején Y éves volt. Ez nem-változó tény. Ezért a felmérésekben előfordulhat az Életkor nevű tulajdonság.)

Gyakori probléma az ismétlődések elhagyása. A tervező az egyedben a Változás-oka és a Változás-dátuma tulajdonság-párost alkalmazza. Kovács Rózsa ekkor (dátum) költözött el (ok) a Forint utcából. Ez a megoldás kétszeresen rossz. Egyrészt Rózsa többször is lakhelyet változtathat (az ok ugyanaz, a dátum más). Másrészt neve is módosulhat az elköltözés napján (a dátum ugyanaz, az ok más). A tervező nem veszi figyelembe, hogy a változás a jelenség ***ismétlődő csoportja***, mind a változás tárgyát (tulajdonság), mind annak keltét illetően. Így aztán az eseményt rögzítő adatok (változás oka és dátuma) maguk is módosulhatnak. A végeredmény: a két adat senkinek semmi hasznos ismeretet sem nyújt. A változási indok és keltezés csak a legutolsó értéket tartalmazza. Az adatmodell továbbra is statikus, mert csak az aktuális állapotra utal.

A fejlesztők egy része az idő dilemmáját a megőrzéssel, az ***archiválással*** próbálja feloldani. Nem-tudatos, ösztönös, megoldással állunk itt szemben. Félreértés ne essék: semmi bajunk sincs az archiválással. Csak azt akarjuk kifejteni, hogy az technikai - fizikai szintű - megoldás. Nem pótolja az idő fogalmi szintű modellezését. Ugyanezért nem foglalkozunk az ún. ***időbélyegzőkkel*** (angolul: time-stamp) sem. Ezek a tárolt fizikai tétel kezelésének az időpontját mutatják és nem a jelenség valós változásának az idejét rögzítik.

A ***változások*** helyes modellezéséhez azt kell megérteni, hogy a változások maguk is ismeretekkel tükrözendő jelenségek. Tipikusan jellemzi őket a változás által érintett egyed (SZEMÉLY/Kovács Rózsa); a módosult ismeretféle (Lakcím); az esemény dátuma; a régi (Forint utca) és az új (Pengő köz) tulajdonságérték. Ha az eseményeket valóban így modelleznénk, akkor a következő képet kapnánk:

19.6 példa

SZEMÉLY (***Személy-azonosító***, ..., Lakcím)
SZEMÉLY-LAKCÍMVÁLTOZÁS (***Személy-azonosító+Dátum***, Régi-lakcím, Új-lakcím)

Ez a megoldás többféle kétséget ébreszt. A két egyedtípus együtt is redundáns, a változási egyed önmagában is az. A Lakcím tartalma nyilván azonos az Új-lakcím-mel és a második Új-lakcím értéke megegyezik az első Régi-lakcím tartalmával. Ezen könnyű segíteni: mivel az Új-lakcím nyíltan redundáns, azt egyszerűen elhagyjuk a változási egyedből. Ezzel viszont egy újabb bajt generálunk. Mégpedig azt, hogy a lakcímek kezelése nem lesz homogén. A régi lakcímeket a második egyedből, az aktuálisat pedig az alapegyedből kell előkeresni.

Azután fellép egy még komolyabb probléma. Egy jelenségnek számos olyan ismerete lehet, ami változik. Ebből az következne, hogy a SZEMÉLY-hez több változási egyedet (pl. SZEMÉLY-CSALÁDI-ÁLLAPOTVÁLTOZÁS) kellene kapcsolni. Tegyük fel, hogy a személyeknek húszféle adata változhat. Így azt kikeresni, hogy egy adott napon mi is jellemezte Kovács Rózsát, meglehetősen kényelmetlen lenne, hiszen 21 egyedben kellene tallózni (húsz változásiban és a változatlan adatokért az alapegyedben).

Ezt a problémát kétféle módon lehet feloldani. Előrebocsátjuk, hogy mindkét megoldás tökéletlen és egyelőre a modellezés-elmélet nem ismer kielégítő módszert a változások tükrözésére.

Vannak, akik az ***állapot-modellezés*** hívei. A jelzett problémát úgy oldják fel, hogy a 19.7 példának megfelelő struktúrát alkotnak.

19.7 példa

SZEMÉLY (***Személy-azonosító+Dátum***, ..., Lakcím, Változás-ok)

Ez a változat elvileg többszörösen is hibás. Kovács Rózsa egyetlen valós jelenség, tehát egy tükörképének (egyed-előfordulásának) kellene lennie az adatbázisban. Viszont ez a SZEMÉLY egyed több - dátumonként egy - előfordulást tartalmaz ugyanarra a személyre. Az csak részben baj, hogy egy napon többféle változás is felléphet. Ezeket a sokszoros változásokat nem szabad a

Változás-ok összetett kódolásával (pl. 1=lakcím-, 2=név-, 9=lakcím- és névváltozás) feloldani. Ilyenkor a Változás-ok tételt az azonosító részévé kell tenni. Mindez persze nem segít azon, hogy a nem-változott, de változtatható adatokat többszörösen tároljuk. Ráadásul az egyed csak első normálformájú, hiszen a személy nem-változtatható tulajdonságai (pl. Nem, Születési-dátum) részlegesen függnek az összetett azonosító első részétől.

Az állapot-modellezés gyakorlati kényelmetlenségeket is okoz. Az nem nagy gond, hogy az aktuális - legutolsó - állapotot több közül kell kikeresni procedurálisan. Komolyabb baj, hogy az egyedtípus előfordulás-halmaza többszörösére duzzad és ezáltal jelentősen megnő a kezelési idő. Ha a múltbeli adatokra csak ritkán, a jelenlegire gyakran van szükségünk, akkor az állapot-modellezés nagyon rossz hatékonyságra vezet.

Az elmondottak miatt sokan az ***esemény-modellezés*** mellett voksolnak. Ebben a változatban a modellrészlet a 19.8 példa mintáját követi:

19.8 példa

SZEMÉLY (***Személy-azonosító***, ..., Lakcím)
SZEMÉLYVÁLTOZÁS (***Személy-azonosító+Dátum+Változás-ok,*** Régi-adat)

Ez a megoldás sem tökéletes. Elvileg azért nem az, mert a Régi-adat nevű tulajdonság értéke inhomogén. Hol lakcímet, hol családi állapotot, hol mást jelent. A Változás-ok utal az érték tartalmára. Így nem teljesül az egy ismeret = egy név elve. Feltételesen - vagyis procedurálisan - lehet csak kezelni a Régi-adat tartalmát. Gyakorlati nehézségek is fellépnek. Ha azt akarjuk megtudni, hogy melyek voltak Kovács Rózsa adatai egy múltbeli napon, akkor több változás egyed-előfordulást kell kikeresni. Ráadásul nemcsak a vonatkozó dátum szerintieket, hanem esetleg a korábbiakat is. Rózsa X napon költözött el és az X-Y napon változtatott állást. Tegnap is új munkahelye lett. Ahhoz, hogy megismerjük Rózsa X napi munkakörét, elő kell keresnünk az X-Y nappal azonosított egyed-előfordulást is.

Amint látjuk, a változások modellezésére nincsen kielégítő megoldás. Ezért nem tudunk minden helyzetben alkalmazható módszert ajánlani. Tekintettel arra, hogy a múltbeli adatok közötti keresgélés általában ritkább, mint az aktuális ismeretek iránti igény és nem jellemző, hogy egy jelenségnek minden régi ismeretére egyszerre vagyunk kíváncsiak, inkább a 19.8 példa megoldása felé hajlunk.

Bizonyos környezetekben ennek a változatnak a generalizálása is megfontolandó. Sokszor alkalmaznak úgynevezett ***történeti*** (angolul: history) egyedtípust, amely egy helyen rögzíti a különböző egyéb egyedtípusokra vonatkozó eseményeket a 19.9 példa módján:

19.9 példa

ESEMÉNY (***Egyedtípus+Egyedazonosító+Dátum+Változás-ok,*** Régi-adat)

Ha tucatnyi egyedtípusunk van, amelyek mindegyike változékony, akkor ugyanannyi X-VÁLTOZÁS egyedet is fel kellene venni a modellbe. A proceduralitás - egyébként nem megengedhető - növelésének a bekalkulálásával a modellt az egyetlen (vagy néhány) történeti egyedtípus bevezetésével kivételesen egyszerűsíteni lehet. Ez az egyed az ismeret négy dimenzióját az ötödikkel, az idővel ötvözi. Utal a megváltozott egyed típusára (pl. SZEMÉLY), a konkrét előfordulásra (Kovács Rózsa), a tulajdonság típusára (Változás-ok=„Lakcím”), a változás időpontjára (Dátum) és a volt értékre (Régi-adat).

Azt mindenki láthatja, hogy az ESEMÉNY egyedtípus kezelése kellemetlenül összetett. Viszont ennek a megoldásnak van egy hallatlan előnye. Az állapot- és esemény-modellezés feltételezi, hogy a jelenség megváltozott ismeretének a szemantikai lényege azonos legyen a

korábbival. Ha mást értünk ma Lakcím-en, mint tegnap - például a Lakcím csoport és megváltoztatjuk a felépítését -, akkor a 19.6 és 19.7 példa megoldásai nem alkalmazhatók. A generikus ESEMÉNY egyedtípusban viszont az is rögzíthető, hogy a SZEMÉLY-nek korábban volt egy ma már nem használt vagy értelmében megváltozott adata. Ezért a kiforratlan szerkezetű, strukturálisan is módosuló adatbázisok esetében a korábbi állapotok/események megőrzésére kiválóan alkalmas a történeti egyedtípus.

Vegyük észre, hogy ez a megoldás nem egyenrangú az archiválással. Az archív állományokat ugyancsak archivált programokkal kell feldolgozni. Ha maguk a programok módosulnak a megváltozott adatszerkezet szerint, akkor a régi programokat kell előkeresni a korábbi struktúrájú ismeretek kezelésére. Ezzel szemben a történeti egyed kezelésére írhatunk egy viszonylag egyszerű általánosított programot. Persze mindennek az a feltétele, hogy a régi és az új - immár más értelmű - ismereteket pontosan el tudjuk különíteni. Ha ma mást jelent a Lakcím, mint tegnap, akkor a történeti egyedben ne legyen azonos a hivatkozásuk.

Mielőtt az idő modellezésével kapcsolatos gondolatokat lezárnánk, ki kell térnünk a korlátok egy speciális fajtájára. Az ***átmenet korlátairól*** (angolul: traversing) van szó, amely az állapotokkal kapcsolatos és amelyekről a tervezők sokszor elfeledkeznek.

Tipikus példaként a családi állapotot szokták említeni. Senki sem lehet özvegy, ha előtte nem volt házas. Csak akkor lehet először házas, ha korábban nem volt az. Más: egy szerződés nem szüntethető meg, mielőtt megkötötték volna. Ha átmenetileg szüneteltetett állapotban van, akkor újraéleszthető, de ha végleg megszűnt, akkor nem. Az idő és az állapot visszafordíthatatlan párosairól van itt szó, amelyeket az adatmodellben kellene korlátként meghatározni. Ezt sokszor nem tesszük meg - és adatbázisaink inkonzisztenssé válnak.

19.6 Modell-sablonok

Az adatbázistervezés bizonyos értelemben hasonlít a mérnöki munkára. (Az angol „engineering” szó tervezést jelent.) Vannak jó és rossz mérnökök, figyelmes és figyelmetlen adatbázistervezők. Az előképek, a részminták, a sablonok használatában mutatkozik meg a különbség.

A rossz mérnök, a figyelmetlen tervező visszaél a sablonokkal. A korábban bevált megoldásokat erőlteti olyan helyzetben is, ahová azok nem illenek. Bevalljuk, hogy kicsit félünk az úgymond rutinos számítástechnikusoktól. X.Y. már a sokadik készletgazdálkodási adatbázist tervezi. Előveszi korábbi sablonjait. Anélkül, hogy a konkrét helyzetet elemezné, a régi minták alapján rögtönöz, mert azt a részletet úgy szokta megoldani, hogy... Ráadásul az „öreg harcik” az első pillanatban nagyon is meggyőzően tudják „eladni” a régi megoldásokat. A felhasználó még észbe sem kapott, máris van egy rossz adatbázisa. Olyan lova, amit nem is akart.

A rutin sohasem pótolhatja a körültekintést, a figyelmességet, a jelenségek lényegének mindenkori pontos feltárását. Ugyanakkor nagy csacsiság lenne az is, ha nem használnánk ki a „típustervek” lehetőségeit. Ha nem élnénk a jól bevált sablonokkal. Szemben a gyakorlott tervezővel, aki ***tartalmilag*** is a saját elképzeléseit erőlteti, a kezdő fejlesztő nem ismeri fel és nem alkalmazza az általános ***formai-szerkezeti*** mintákat. Minden egyes helyzetben, új feladatnál újra feltalálja a spanyolviaszt. Ezért néhány tippel szolgálunk a számára.

1. sablon: Keresni kell az adatmodellben az alapvető természetükben azonos vagy hasonló jelenségeket. Ha a modellben van a SZEMÉLY mellett ORVOS, ÜZLETKÖTŐ, TANÁR, ZENESZERZŐ vagy hasonló nevű egyedtípus, akkor lehet, hogy valami nem stimmel. Ilyenkor élni kell a ***specializáció*** sablonával. Ha van Személynév és Orvosnév nevű adatunk, akkor adatbázisunk redundáns és inkonzisztens lesz. A doktornő férjhez megy és nevét megváltoztatja, mint orvos, a változást pedig nem vezetjük át az általánosabb SZEMÉLY egyedben - vagy éppen megfordítva.

2. sablon: Kutassunk az adatmodellben az azonos minták után. A vevőnek is van neve, a szállítónak is. A vevőnek is van címe, számlaszáma stb., a szállítónak is. Ráadásul a vevő más helyzetben lehet szállítónk is. Ekkor elővesszük a ***generalizáció*** sablonát. Nem hagyjuk magunkat megtéveszteni a minősített megnevezésektől (pl.: Vevőnév/ Szállítónév, Vevőcím/ Szállítócím stb.).

3. sablon: Figyeljünk a ***homogenitásra***. Ha bármilyen homogén hálózat modellezéséről van szó, akkor biztosan a családfa-egyed megoldáshoz kell nyúlnunk. Helyettesítő cikkek, utak, csatornák, általában vezetékek, egymásba fonódó projektek, többszörös célra adott pénzek, természetes személyek rokoni kapcsolatai, alternáló szállítási útvonalak stb. esetben a családfa-egyed holtbiztos megoldás. Ezzel szemben figyelnünk kell a hierarchikus jellegre. A cég belső, fix szervezeti felépítése, a részletekben fizetendő számla, a ház lakásai illetve a lakások helyiségei stb. adott esetben a tipikusan alkalmazható visszamutató kapcsolat megoldását követelik meg.

4. sablon: A ***többszörös kapcsolat*** is azonnal „adja magát". Olyan többértékű tulajdonságokról van szó, amelyek nem valódi ismétlések. Nem ugyanazt jelentik és ezért nincs szükség külön egyedtípusra a modellezésükhöz. Lássunk csak pár példát. A személyeknek pontosan csak egy állandó és egy ideiglenes lakcímére vagyunk kíváncsiak. A két címben lévő Irányítószám kétszeresen kapcsol a TELEPÜLÉS egyed felé, mert a település nevét csak ebben az állományban fogjuk tartani. A ZENE egyedtípusban utalnunk kell a zeneszerzőre és a szövegíróra. Ha a ZENE/zeneszerző és a ZENE/szövegíró összefüggést nem akarjuk külön ismerettel is jellemezni (pl. jogdíj), akkor a SZEMÉLY - ZENE kétszeres kapcsolatot fogjuk alkalmazni. (Ha a viszonyhoz külön ismeret is kötődik, akkor máris ott van a másik sablon: az M:N-es összefüggést kifejező kapcsolóegyed.) Ha modellezzük az időt, akkor a RENDELÉS (***Rendelésszám***, ..., *Rendelésdátum, Szállításdátum*) kettős kapcsolata a DÁTUM egyed felé kézenfekvő.

5. sablon: Itt egy újabb titkot árulunk el. Adatbázisaink tele vannak úgynevezett ***kétoszlopos*** (angolul: two-column) táblákkal. Ezek a régi gyakorlatból átvett kód-megnevezés párosokat rögzítik. Fizetési-mód-kód/Fizetési-mód-megnevezés. Változás-oka-kód/Változás-oka-megnevezés. Terület-jellege-kód/Terület-jellege-megnevezés stb. Vegyük észre, hogy a bármilyen „kód" már maga sem igazándi fogalmi szintű tényező. Ezért a mostani mondanivalónk kicsit kilóg a modellezés témaköréből. De ha már itt tartunk, akkor nem mulaszthatjuk el a helyzetre vonatkozó tippünket.

A bemenet gyorsaságát és ellenőrizhetőségét javítani akaró kódok és a kimenet emberi élvezhetőségét szolgáló megnevezések párosai teszik ki az aprólékosan megtervezett adatmodell egyedtípusainak a felét. A tervező készít FIZETÉSI-MÓD, VÁLTOZÁS-OKA és TERÜLET-JELLEGE egyedtípusokat. Ezt kétszeresen is feleslegesen teszi. Egyrészt azért, mert a modellt teljesen hiábavalóan elbonyolítja, hiszen - mint említettük - a kódok nem valódi fogalmi tényezők. Ezért az ilyen megoldás elvileg hibás. Másrészt azért, mert az efféle „álegyedtípusok" rettentően lelassítják az adatkezelést. Az érdemi adatok kezelési idejét hatalmasan megnöveli a kód-megnevezés táblákban való keresgélés. Nem azért, mert ezek a táblák nagyok, hanem azért, mert az alapállományoktól el kell mozogni hozzájuk.

Ezért a szakértők azt javasolják, hogy a kétoszlopos táblákból egy (vagy kevés) háromoszloposat kell készíteni. Például ilyen módon: KÓDOLT-ISMERET (***Kódtípus+Kódérték***, Megnevezés). Ez a tábla az adatkezelés kezdetén a memóriába tölthető és így a sokkal nagyobb lemezhozzáférési-idő helyett a tárkezelési-idő alatt elérhető.

Nincs lehetőségünk arra, hogy az adatmodellezés sablonjait - tipikus megoldásait - tovább részletezzük. Ez nincs is szándékunkban. Minden tervezőnek magának kell kialakítania a szokásait a 19.1 pontban említett fegyelmezett fantázia szerint. Ezt a pontot azzal a kitétellel zárjuk, hogy a félig-lusta tervező a jó tervező. Az, aki keresi a fontosat, az újat, a speciális jelentést, de nem minden helyzetben találja ki a már rég ismert megoldásokat. Aki ismeri a sablonok alkalmazásának az arany középútját.

19.7 Szinttévesztés

Nem búcsúzhatunk el ettől a fejezettől a legsúlyosabb modellezési hiba említése nélkül. Az ismereti szintek összekeveréséről, végeredményben az adatmodell lényegének a meg nem értéséről van szó.

Az adatbázis az egyféle elrendezés szerint, egyidejűleg és együttesen, különleges technikai beavatkozások nélkül kezelhető ismeretek együttese. Az „egyféle elrendezés szerint" kitételnek semmi köze sincs a szoftverhez. Ugyanazon az adatbázison többféle kezelővel is dolgozhatunk. A lényeg az, hogy adott az egyetlen és egyféle módon szervezett adatmodellünk. Ellenpélda: A társadalmi biztosításokat pár éve teljesen más ismeretek jellemezték, mint ma. Más volt az alapok elosztása, felhasználása, elszámolása. Ezért szó sem lehet arról, hogy létezik egy folyamatos társadalombiztosítási adatbázis. Volt valamikor egy ilyen célú adatrendszer, ma pedig van egy másik, amelynek az előbbihez az égadta világon semmi köze sincs. Az adatbázisok és modelljeik is születnek, majd meghalnak...

Nem tekinthető „egy" adatbázisnak az a hasonló vagy azonos ismereteket tartalmazó két adattároló rendszer sem, amelynek az elrendezése - az adatmodellje - különbözik. Ha tavaly az X, idén az Y módon szerveztük meg biztosítási ismereteinket, úgy a két adatrendszer akkor sem képez egy adatbázist, ha ma ugyanazon a számítógépen, ugyanazzal a kezelővel manipuláljuk a két adathalmazt. Az adatbázis és az adatmodell kölcsönös és egyértelmű viszonyban áll egymással. Ha más a szerkezet, akkor más az adatbázis is.

Nem lényegtelen az „egyidejűleg és együttesen" kitétel sem. A technikai fogyatékosságok miatt sokszor kényszerülünk arra, hogy az elvileg egységes adatbázis részeit úgy ismételjük meg (idegen szóval: replikáljuk) több számítógépen, hogy a részek egyidejű és együttes kezelésére nincs mód. Van egy központunk és vannak leányvállalataink, fiókjaink, kihelyezett részlegeink. Nincs viszont on-line kapcsolatunk. Ezért nem tudjuk a részlegben megismételt, a központban is tárolt ismereteket egyidejűleg és együttesen kezelni. Például ugyanabban a minutában karbantartani. Mindennek semmi köze sincs a szoftverhez. Nem az a baj, hogy a központban és a fiókban más adatkezelő dolgozza fel az azonos felépítésű, azonos modellű (!) adathalmazt. Hanem az, hogy ezt nem egyidejűleg és együtt teszik. Az „adatbázis - egy" elvét sértve valójában több adatbázisunk van. Nem kell részleteznünk, hogy ebből milyen ismereti ellentmondások fakadhatnak.

A technika vaskemény korlátai ellen a modellező semmit sem tehet. Viszont elkerülhetné a nem is olyan szigorú kelepcéket. Azokat, amelyekben saját maga fogja meg önmagát. Most térünk át a szinttévesztéssel kapcsolatos valódi mondanivalónkra.

Láttunk számos olyan úgymond adatbázist, amelyben az ismeretek osztályozása - írd és mondd - adathordozó szerint történt. Az első floppy az X megye, a második az Y megye stb. adatait hordozza. Az adatbázis szerkezetnek - sőt: modellnek - titulált valamijében, annak egyedtípusaiban (?) utalás sem történt a megyére. Ha az X floppy-t teszik be, akkor annak az adatait, ha az Y-t, akkor az azon rögzített ismereteket dolgozzák fel. Magyarul: a lemezke fizikai címe jelenti az azonosítást. Az adatbázis akkor működik, ha „különleges technikai beavatkozásra" kerül sor, azaz bedugják, átmásolják a floppy-t. No ez kuruzslás, berhelés, ámítástechnika. Minden, csak nem adatbázis.

Ezzel érdemi mondanivalónkat lezártuk. Következzék a puding próbája.

20. ELEMZÉS ÉS DOKUMENTÁLÁS

20.1 Egy mintapélda

Eddigi szokásunkkal ellentétben most nem írunk bevezető pontot. Azonnal a dolgok közepébe vágunk. Az olvasó szíveskedjék áttanulmányozni az aprócska mintapéldát és próbálja megállapítani az abban rejlő hibákat. Majd a következő pontban elmondjuk saját elemzési eljárásunkat is. A tervet a 20.1 példa mutatja. Az utolsó sorban a „|" jelölés azt fejezi ki, hogy a jel utáni tételek az azt megelőző tényező szerepnevei.

20.1 példa

CIKK	(***Cikkszám***, Megnevezés, Egységár, Súly, Árucsoport-jel, *Termék-kód*, *Raktárkód*)
NORMÁLIA	(***Normáliakód***, Név, Egységár, Súly, Árucsoport-jel, *Raktárkód*)
ÁRUCSOPORT	(***Árucsoport-kód***, Megnevezés)
TERMÉK	(***Termékkód***, *Befogadó-cikkszám, Beépülő-cikkszám*, Darab)
RAKTÁR	(***Raktárkód***, Raktárcím, Raktárnév, *Főraktárkód*)
FŐRAKTÁR	(***Főraktárkód***, Raktárcím, Raktárnév)
ÜZLET	(***Üzletkód***, Boltcím, Boltnév, X...)
KÉSZLET-1	(***Cikk-kód+Raktárkód+Dátum***, Mennyiségi-egység, Mennyiség, Érték)
KÉSZLET-2	(***Normáliakód+Raktárkód+Dátum***, Mennyiségi-egység, Mennyiség, Érték)
BEVÉTEL	(***Diszpoziciószám+Cikkszám***, Bevétel-dátum, Egységár, Mennyiség, Érték, Raktárkód, Készlet)
KIADÁS	(***Diszpoziciószám+Cikkszám***, Kiadás-dátum, Egységár, Mennyiség, Érték, Raktárkód, Készlet)
Cikkszám	| Normáliakód, Befogadó-cikkszám, Beépülő-cikkszám

Cégünk számítógépes berendezéseket és alkatrészeket forgalmaz. A cikkeket árucsoportokba soroljuk. A csoportokat csoportkóddal azonosítjuk és megnevezésükkel írjuk le. A cikkeket cikkszám, megnevezés, egységár és súly jellemzi. A cikkszám első két karaktere utal az árucsoportra. Az alkatrészekről tudnunk kell, hogy melyik berendezéshez tartoznak és azokból hány szükséges egy termékhez. Ezeket az ismeretet vezetni kívánjuk az ún. normáliákról - csavarok, alátétek, madzagok stb. - is.

Áruinkat célspecifikus raktárakban tároljuk. Vagyis egy-egy árufélét csak egy helyen tárolunk. Kivételt jelentenek a normáliák, amelyek minden raktárunkban megtalálhatók. A raktárt kód azonosítja, név és cím írja le. Lerakataink csillagrendszerűek, vagyis a területi raktárak egy-egy központi raktárhoz tartoznak.

Speciális „raktárnak" számítanak az üzletetek, ahol cikkeinket forgalmazzuk. Az üzletet számos egyéb adat is jellemzi.

Termékeinkről napi készletnyilvántartást vezetünk. A berendezésekről és alkatrészekről darab, a normáliákról csomag szerinti nyilvántartást tartunk. A bevételeket és kiadásokat diszpozició- és cikkszám azonosítja. Ár, mennyiség és érték írja le. Néhány esetben az eladási és a beszerzési ár kedvezményes.

20.2 A mintapélda tartalmi hibái

A tartalmi problémák elemzését mindig a ***valósághűség*** kritériumánál kezdjük. Vagyis azt vizsgáljuk, hogy a tételek megfelelnek-e a tényleges jelenségeknek illetve absztrakciójuk - típusba sorolásuk - helyesen történt-e. Mivel néhány esetben az ár kedvezményes, a tulajdonságok közül a két utolsó egyed Egységár tétele hamisan tükrözi a valóságot. A két egyedbe a Beszerzési-ár és az Eladási-ár illik. (Ezeket természetesen csak akkor töltjük ki, ha az ár eltér az Egységártól.) Ezt a problémát formai úton, normalizálással is felfedezhetjük. A BEVÉTEL és a KIADÁS formailag részleges függést (Cikkszám → Egységár). tartalmaz. Mivel azonban a CIKK egyeddel szemben ebben a két egyedtípusban az Egységár valójában nem redundancia, hanem homonima, ezt a tulajdonságot nem távolítjuk el az egyedekből, hanem új megnevezéseket alkalmazunk.

A terv téves, amennyiben a NORMÁLIA egyedben szerepel a Raktárkód. A normáliákat a leírás szerint több raktár is tárolhatja. Ezért a példa vagy helytelenül tükrözi a valóságot, vagy pedig a NORMÁLIA egyedtípus nem-normalizált, mert abban a Raktárkód ismétlődő adat. Ezt a problémát is feltárhatjuk formai módon. A KÉSZLET-2 egyedtípusban a Normáliakód és a Raktárkód hálós viszonyt mutat, a NORMÁLIA egyedben viszont - a látszat szerint - hierarchikusat. Ez a belső kulcstörés tipikus esete.

A Cikkszám/Normáliakód és a Megnevezés/Név párostól eltekintve a CIKK és a NORMÁLIA adatai megegyeznek. Mi több, a Normáliakód a Cikkszám szerepneve, vagyis az is egy cikkmegjelölés. A tervező nem élt a generalizáció lehetőségével, vagyis rossz absztrakciót alkalmazott. A két egyedtípus összevonható éppen úgy, mint a két KÉSZLET egyed is.

A következő vizsgálati szempont az ***egyértelműség***. A Cikkszám/Cikk-kód illetve az Árucsoport-kód/Árucsoport-jel párosok szinonimák, amit a formális kapcsolati hiányon is láthatunk. A KÉSZLET-1 és az ÁRUCSOPORT formailag nem kapcsolható a CIKK-hez, az ÁRUCSOPORT a NORMÁLIÁ-hoz sem. Ha nem fedeztük volna fel a szinonimát, akkor a kapcsolatelemzésnél derült volna ki a turpisság. (Amint látjuk, a kritériumok összefüggenek. Adott esetben az egyértelműség és a teljesség hiánya együtt mutatkozott.)

A tulajdonságok között - a vizsgálatból már kiesett Egységáron kívül - számos homonimára bukkanhatunk. Ilyen a Megnevezés és a Mennyiség. Ezek leíró tulajdonságok, ezért közös nevük csak „bocsánatos bűn”, ami megfelelő minősítéssel azonnal korrigálható. Viszont a Diszpozíciószám is homonima. Ez a tétel kulcsrész, és így a Cikkszám-mal párosban sérti az azonosítók egyediségére vonatkozó szabályt, amely szerint két egyedtípusnak (BEVÉTEL és KIADÁS) nem lehet ugyanaz az elsődleges kulcsa.

Homonimaként mutatkozik az Érték is. Ezzel azért nem foglalkozunk, mert származtatott adatról van szó, amely sérti a „plusz egy” szabályt. Az érték a mennyiségből és az árból - amely egységár esetén egy eléréssel megkapható a CIKK illetve a NORMÁLIA egyedből - kiszámítható, tehát tárolására nem kerülhet sor sem a készletekben, sem a mozgásokban. Más lenne a helyzet, ha a tervező jelölte volna, hogy az Érték az adott egyedeket jellemzi, de azok tárolására nem kerül sor. Ekkor a homonima-probléma valós és átnevezésekkel kell pontosítani a modellt.

A legcsúnyább tétel a Készlet. Ez a mozgás utáni készletet mutatja, tehát valójában olyan szinonima, amelynek tartalma a két KÉSZLET egyed Mennyiség rovatával egyezik.

Most áttérünk a következő kritériumra, a tartalmi ***teljességre***. Elmaradt a tervből a Raktárkód | Főraktár-kód szerepnév megjelölés és emiatt formailag nem tudhatjuk, hogy a lerakat melyik területi központi raktárhoz tartozik. Persze ezt a hiányosságot a másik kritérium - a minimalitás - vizsgálata során is feltárhattuk volna. Kétszeresen szerepel a modellben a Raktárnév és Raktárcím adat. Mivel a főraktár is egy raktár, a FŐRAKTÁR egyedre egyáltalán nincs szükség; a főraktárak adatait is tárolhatjuk a RAKTÁR egyedben.

Még komolyabb gond, hogy a modell egyszerűen nem tükrözi azt a tényt, hogy a boltokban is van készlet, vagyis az üzlet is bizonyos értelemben raktár. Ha máshonnan nem, onnan mindenképpen észrevehettük ezt a bajt, hogy az ÜZLET egyed semmilyen más egyedhez nem kapcsolódik. Itt pedig megtorpanunk. Mert az mégsem lenne helyes, valósághű, ha az üzleteket raktárként modelleznénk. Hiszen az üzletnek számos egyéb adata (X) van.

Itt a generalizáció és specializáció párosához kell folyamodni. A RAKTÁR, FŐRAKTÁR és ÜZLET egyedünk mindegyike névvel és címmel ellátva. Egészen príma megoldás lenne, ha alkalmaznánk egy generikus EGYSÉG egyedtípust és annak az altípusa, specializáltja lenne a sok sajátos adatot felvonultató ÜZLET. Bizonyos egységekhez kapcsolódna KÉSZLET. Amivel megoldanánk azt a hiányosságot is, hogy a modell nem tükrözi az üzletekbe, mint raktárakba történő bevételeket és kiadásokat...

Most elérkeztünk a ***minimalitás*** aspektusához. A rossz logikai redundanciákat normalizálással fedezhetjük fel. A példa szerint a Mennyiségi-egység (darab, csomag) közvetlenül a cikkeket jellemzi. Ehhez képest a két KÉSZLET egyed részleges függést mutat, mert a Cikkszám egyedül is meghatározza a Mennyiségi-egységet. Ezért a KÉSZLET egyedekből ki kell emelni ezt a tulajdonságot és azt át kell tenni a CIKK egyedbe. Ha eddigre már nem jöttünk volna rá, akkor a normalizálás során azt is észrevennénk, hogy a BEVÉTEL és a KIADÁS egyedben a Készlet függ a Cikkszám+Dátum+Raktárkód együttestől. A két egyed nem volt 3NF alakban, mert nem minden meghatározó az egyed kulcsjelöltje (vö. a BCNF alakkal).

Utolsó lépésként még a ***rejtett fizikai redundanciákra*** vadászunk. A TERMÉK szemmel láthatóan családfa egyedtípus akar lenni. Általában úgy illik, hogy a családfa azonosítója az alapegyed kulcsának a szerepneveiből álljon. Mivel most nem ez a helyzet, gyanút fogunk. Az olvasó nem fedezhette fel, hogy a kétszeres kettős azonosítás gyönyörű példányával áll szemben. Mivel a Befogadó-cikkszám+Beépülő-cikkszám páros is alkalmas az egyed azonosítására, a Termékkód felesleges. Nagyon gyakran előfordul, hogy a végterméknek van egy cikkszáma is és egy termékazonosítja is (Termékkód). Ez önmagában véve nem baj. A CIKK egyedben a Termékkód nem alternáló kulcs, mert nem minden cikk végtermék. A TERMÉK-ben viszont a Termékkód és a Befogadó-cikkszám ugyanazt a dolgot jelöli, sértve az egyszeres azonosítás elvét.

Egy további probléma: az élesszeműek nyilván észrevették, hogy az Árucsoport-jel a CIKK egyedben ***implicit redundanciát*** okoz, hiszen ez az ismeret a Cikkszám első két jelével megegyezik. Ezért ezt az adatot el kell távolítani, miközben a Cikkszám implicit csoportot explicitté kell tenni.

Ezzel a példa elemzését befejeztük. Ezen a miniterven is láthattuk, hogy mennyi csacsiságot képes egy tervező elkövetni. Az eset jól mutatta, hogy a ***mechanikus normalizálás*** rengeteget segíthet a hibák feltárásában, de csak akkor, ha a modellt előbb egyértelművé tesszük. Vannak viszont olyan - és többnyire súlyosabb - problémák, amelyek csakis átértelmezéssel, ***szemantikus normalizálással*** küszöbölhetők ki. Gondoljunk csak a TERMÉK helytelen kulcsára, vagy az EGYSÉG fogalmának a bevezetésére.

A 20.2 példa mutatja a feladat első - helyes, de igen hevenyészett - megoldását. A {} páros a csoportokat, a | jel a szerepneveket, a $\subset$ jelölés az altípusokat, a sima vastag szedet a származtatott, de nem tárolt adatokat jelzi.

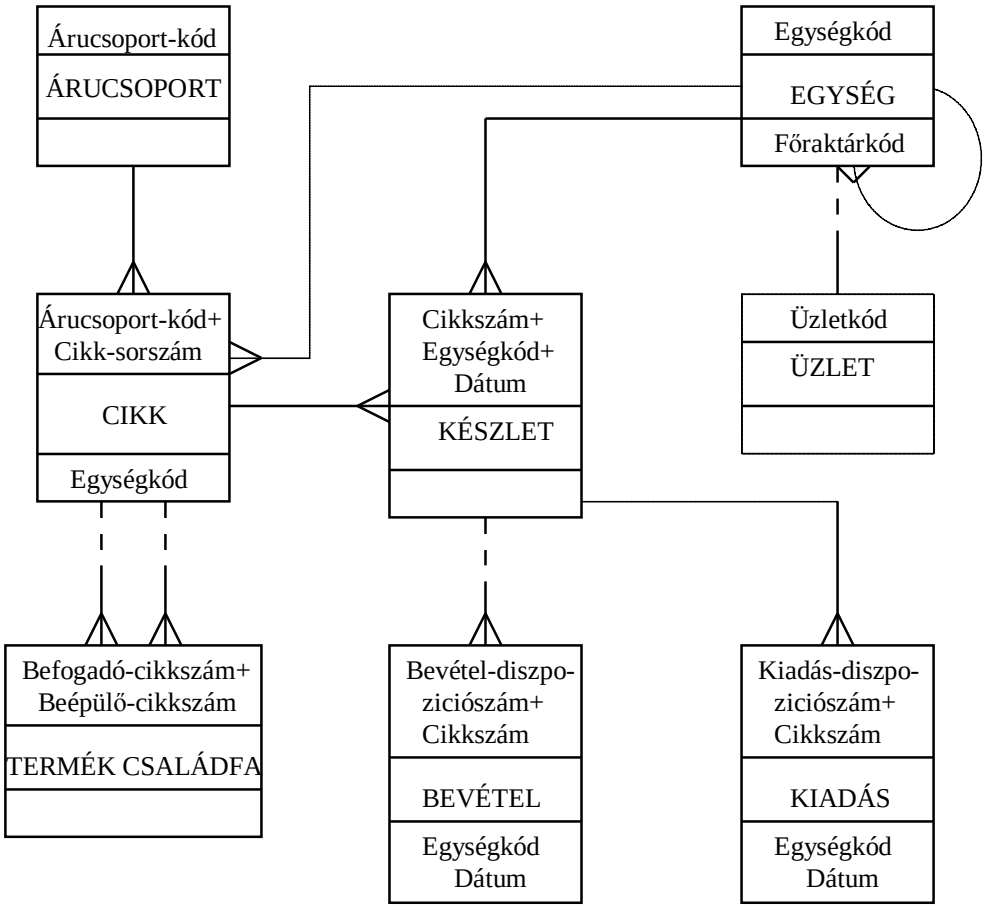
20.2 példa

CIKK	(***Cikkszám***, Cikk-megnevezés, Egységár, Mennyiségi-egység, Termékkód, Súly, *Egységkód*)
Cikkszám	{Árucsoport-kód, Cikk-sorszám}
ÁRUCSOPORT	(***Árucsoport-kód***, Árucsoport-megnevezés)
TERMÉKCSALÁDFA	(***Családfa-azonosító***, Beépülő-darab)
Családfa-azonosító	{Befogadó-cikkszám+Beépülő-cikkszám}
EGYSÉG	(***Egységkód***, Egységnév, Egységcím, *Főraktárkód*)
Egységkód	| Főraktárkód, Üzletkód
ÜZLET	(***Üzletkód***, X...) ⊂ EGYSÉG
KÉSZLET	(***Készletazonosító***, Készlet-mennyiség, **Készletérték**)
Készletazonosító	{Cikkszám+Egységkód+Dátum}
Készletérték	= Készlet-mennyiség*CIKK(Egységár)
BEVÉTEL	(***Bevételazonosító***, *Készletazonosító*, Bevételi-ár, Ártípus, Bevétel-mennyiség, **Bevételérték**)
Bevételazonosító	{Bevétel-diszpoziciószám+Cikkszám}
Bevételérték	= Ha Ártípus=„standard”, CIKK(Egységár)*Bevétel-mennyiség, egyébként Bevételi-ár*Bevétel-mennyiség
KIADÁS	(***Kiadásazonosító***, *Készletazonosító*, Kiadási-ár, Ártípus, Kiadás-mennyiség, **Kiadásérték**)
Kiadásazonosító	{Kiadás-diszpoziciószám+Cikkszám}
Kiadásérték	= Ha Ártípus=„standard”, CIKK(Egységár)*Bevétel-mennyiség, egyébként Kiadási-ár*Bevétel-mennyiség

20.3 Az adatmodell dokumentálása

Most pedig elmondjuk, hogy a fenti megoldás miért „hevenyészett”. Az adatmodell dokumentálásának nagyon szigorú szabályai vannak és a 20.2 példa nem tesz eleget ezeknek a követelményeknek. Nézzük csak meg, hogy miképpen épül fel egy jól meghatározott adatmodell-dokumentáció!

A terv első fejezete megadja az ***adatbázis nevét*** (például: ÁRUFORGALOM), majd röviden leírja annak célját illetve környezetét. Mondjuk így: „Az ÁRUFORGALOM adatbázis a cégünk összes részlege által forgalmazott termékeknek, az azokkal kapcsolatos készleteknek és készlet-mozgásoknak, a termékek felépítésének és tárolási helyeinek az ismereteit rögzíti. Célja az, hogy...”



20.1 ábra: Az ÁRUFORGALOM adatbázis-részletének a diagramja

A bevezető fejezetnek feltétlenül tartalmaznia kell a modell Bachman-diagramját. A mi példánkét a 20.1 ábra mutatja. A diagramhoz rövid magyarázatot illik fűzni főleg akkor, ha a modellben vannak kétségeket ébresztő tényezők. Nagyon kevesen értik meg például azt, hogy a szervezeti egységek összevonhatók egyetlen egyedtípusba és a raktárakat, főraktárakat, üzleteket már csak azért is egy EGYSÉG-ben jó tükrözni, mert különben több KÉSZLET egyedtípusra lenne szükség. A CIKK és az EGYSÉG közvetlen kapcsolata is kételyeket okozhat, mivel van közvetett kapcsolatuk is a KÉSZLET-en át. El kell ezért mondani, hogy a berendezéseket specifikus raktárakban tárolják és arra utal a kapcsolat, miközben a normáliák több raktárban is fellelhetők.

Az első fejezetben utalni kell arra a Függelékre, amely a leírásokban, listákban és diagramokon alkalmazott konvencionális jelöléseket ismerteti. Az pedig természetes, hogy a jól megszerkesztett tartalomjegyzéken kívül a bevezetőben is illik felsorolni a többi fejezet várható tartalmát.

A dokumentáció következő fejezetei nem kötöttek, tehát a most elmondottakat nem kell szentírásnak venni. Mi általában az ***egyedtípusok listáját*** adjuk meg a második részben. Ez nem tévesztendő össze a rekordképekkel. A listaszerű felsorolás az egyedtípusok nevét, rövid magyarázó leírását, darabszámát, minősítését stb. tartalmazza. Mindez a kezdetben fontosabb, mint az, hogy milyen adatok is tartoznak az egyedhez. Lásd a 20.3 példa nem teljes felsorolását.

20.3 példa

CIKK	A CIKK az általunk forgalmazott árukat jelenti. Ezek között vannak számítógépes berendezések, alkatrészek és segédanyagok (normáliák).... A CIKK alapegyed. Minimum X, átlagosan Y, maximum Z tétel nyilvántartására van szükség.
...	
TERMÉK CSALÁDFA	Az általunk forgalmazott áruk felépítését, szerkezetét mutatja ...
...	
ÜZLET	Cégünk szervezeti részlegeinek (ld. EGYSÉG) az altípusa, amely...

Az egyedlistát általában nevük ABC sorrendjében állítjuk össze. Ez a többi tényezőre is vonatkozik. Az adatmodellezés ebből a szempontból nem ismer olyan minősítéseket, hogy „fontosabb", vagy „előbb" stb.

A harmadik fejezetet a ***kapcsolattípusok listájának*** szentelnénk. Az ugyanis részletkérdés, hogy például a cikknek van-e neve. Bizonyára van; no és aztán? Sokkal lényegesebb, hogy a cikk, a készlete, a bevétele és a kiadása helyes összefüggésben áll-e egymással. A kapcsolatlistában, amely szintén ABC-szerinti, a fölé- és alárendelt egyedtípus neve után megadjuk a kapcsoló tulajdonság nevét, a kapcsolat jellemzőit (fok, opcionalitás, minősítés). Szemben az itteni 20.4 példával, a kapcsolattípusokhoz is rövid leírásokat fűzünk, amennyiben az kívánatosnak látszik. Természetesen a korábban említett Függelékben elmondjuk, hogy mit is jelent az „1:N" fok, a „K-K" kétirányúan kötelező kapcsolat stb.

20.4 példa

ÁRUCSOPORT - CIKK	Árucsoport-kód	1:N	K-K	birtoklási
CIKK – KÉSZLET	Cikkszám	1:N	O-K	birtoklási
CIKK - TERMÉKCSALÁDFA-1	Befogadó-cikkszám	1:N	O-K	családfa
CIKK - TERMÉKCSALÁDFA-2	Beépülő-cikkszám	1:N	O-K	családfa
EGYSÉG - CIKK	Egységazonosító	1:N	O-O	birtoklási
EGYSÉG - EGYSÉG	Főraktárkód	1:N	O-O	visszamutató
EGYSÉG - KÉSZLET	Egységkód	1:N	O-K	birtoklási
EGYSÉG - ÜZLET	Üzletkód	1:1	O-K	altípus
KÉSZLET - BEVÉTEL	Készletazonosító	1:N	O-K	birtoklási
KÉSZLET - KIADÁS	Készletazonosító	1:N	O-K	birtoklási

Mi a negyedik fejezetbe tennénk a ***táblaképeket***, vagyis az egyed-tulajdonság-viszony listákat. Természetesen nem a 20.2 példában mutatott módon. Az ilyen tömör ábrázolásra csak az elemzésnél van lehetőség és a szakkönyv terjedelme miatt szükség. Egy valódi dokumentációban a tulajdonságok szerepére, opcionalitására, egyediségére, fogalmi szintű ábrázolására, az egyedhez kötött speciális validálására stb. is utalni kell. Egy dokumentálási részletet mutat a 20.5 példa.

20.5 példa

CIKK

Tulajdonságnév	**Adattípus**	**Hossz**	**Szerep**	**Opcionalitás**	**Egyediség**	**Megjegyzés**
Cikkszám	Csoport		Kulcs	Kötelező	Egyedi	
Árucsoport-kód	Karakteres	2	Kulcsrész	Kötelező	Egyedi	
Cikk-sorszám	Numerikus	5	Kulcsrész	Kötelező	Egyedi	
Termékkód	Karakteres	8	Leíró	Opcionális	Egyedi	1)

Tervezői rossz szokás, hogy az alkotó a táblaképben, netán éppen a tulajdonság nevében sorolja fel a megengedett értékeket és az egyéb korlátokat. Ez persze megtöri az áttekintést. A helyes megoldás az, hogy a lista után írjuk le a megjegyzéseinket. Például a validálásra vonatkozó kitételeinket. Ezt se tegyük meg akkor, ha a megjegyzés a tulajdonságra mindig - egyedtől függetlenül, vagyis minden egyedben - vonatkozik. Ezért nem igazán „profi" megoldás, ha most az 1) megjegyzés alatt elmondjuk, hogy a Termékkódot akkor kell kitölteni, ha az Árucsoport-kód xy-xz értéket vesz fel...

A dokumentáció ötödik fejezetében kellene ezt megtennünk, ahol egyszeresen - egyedektől függetlenül - adjuk meg a ***tulajdonságok listáját***. Ez nevet, rövid leírást, általános ábrázolást és validálást, abszolút szerepet tartalmaz. Bár a ***tulajdonságstruktúra*** a 18.1 ábra szerint külön modellezési tényező, annak nem feltétlenül szánnánk külön hatodik fejezetet. Valamilyen ábrázolási konvenció szerint a tulajdonságlistában tüntetnénk fel a csoportok és a származtatott adatok tényezőit illetve a szerepneveket. Persze ha nagyon precízek akarunk lenni, akkor ezt külön fejezetben tesszük, mert...

Az adatmodell dokumentációjának nagyon pontosnak kell lennie. Nemcsak azt kell tudni visszakeresni belőle, hogy a Családfa-azonosítónak része a Beépülő-cikkszám, hanem azt is, hogy a Beépülő-cikkszám a Családfa-azonosító csoport tagja. Ebből következik, hogy az adatmodell-dokumentáció szükségszerűen redundáns. Az adatmodell tényezői közötti összefüggéseket oda-vissza - többszörösen - tartalmazza. Ez pedig szükséges baj.

Gondot fog jelenteni például, hogy az egyedtípusok nevét azonosan írjuk-e az egyedlistában, a kapcsolatlistában és a „táblaképekben". Ha az X tulajdonság csoportos tényezőjének említjük az Y tételt, akkor ezt fordítva is megtesszük-e és ugyanezt a két nevet alkalmazzuk-e?

Amint látjuk, eljutottunk a döntő kérdéshez. Minden szempontból konzekvens adatmodell-dokumentációt csak számítógéppel tudunk készíteni. Olyan szótárprogrammal, amely magát a modellt is adatbázisszerűen - tehát tényezőit egyszeresen - tárolja, viszont tetszőleges „nézetben" jeleníti meg. (Ld. a metaadatbázisokról szóló 9. fejezetet.)

Már csak egy vitapont maradt hátra, bár egyáltalán nem könnyű témáról van szó. Pusztán „csak" arról, hogy mit is nevezünk adatbázisnak? A kezelőrendszerek adatbázisnak az egy fizikai egységen - ez lehet egy kapcsolt hálózat is! - egyidejűleg és együtt kezelhető adatok halmazát tekintik egy adatbázisnak. Ez a megközelítés két problémát szül.

A kezelő számára nem egy adatbázis része az a két fogalmilag összefüggő adat, amelynek egyikét az X gépen, a másikát az Y eszközön kezelik úgy, hogy a gépek között nincs a kezelő-rendszer által menedzselhető on-line kapcsolat. Ezzel szemben a kezelő részére az ő általa manipulált bemeneti, kimeneti, átmeneti, biztonsági stb. tételek is az adatbázis részét jelentik, noha a bemenet, a kimenet, a technikai adat stb. nem tartozik az adatmodellbe.

A fejlesztők, felhasználók és vezetők közös óriási tévedése, hogy ***az adatmodell-dokumentációt összetévesztik az adatbázisterv dokumentációjával.*** Pontosabban szólva az előbbit el sem készítik. A technikai, fizikai, berhelt megoldásokkal terhelt, a modell egésze szempontjából nem-teljes, eszközökhöz kötött rekordképek rendetlen halmazára ráfogják, hogy adatmodell. Így persze nem lehet modellezni, modellt elemezni, információs rendszert fejleszteni.

A relációs kezelők használata során a valós jelenséget tükröző tábla (pl. CIKK), az annak karbantartására szolgáló bemenet, a hibaállomány, a lekérdezett ismerethalmaz is - reláció. Ezért megértjük a tervezők azon vágyát, hogy ezeknek a tábláknak a leírásait egy dokumentációban szeretnék látni. Engedve szigorunkból most azt mondjuk, hogy ám készítsenek egyetlen dokumentációt. Ámde ebben az esetben adják fel a korábbi ABC-sorrend elvét és a dokumentációban külön alfejezetekben különítsék el az adatbázis valódi modelljének a leírását az adatbázis manipulálásával kapcsolatos tényezők leírásaitól.

***

Most már szinte mindent elmondtunk, amit az adatmodellről tudni kell. Néhány titkot persze megőriztünk magunknak. Például azt, hogy vannak olyan titkok is, amelyeknek magunk sem ismerjük a nyitját. Mindenesetre könyvünk majdnem a végéhez ért. Már csak annyi maradt hátra, hogy a 21. fejezetben, az utolsóban, az eddigieknél egy kicsit komolyabb példával tegyük próbára az olvasó türelmét.

21. TERVEZÉSI ESETTANULMÁNY

21.1 A példa kerete

Elérkeztünk könyvünk végéhez. Most eljutottunk arra a pontra, hogy összegeznünk illik az elmondottakat. Az áttekintés során nem óhajtjuk megismételni a korábbi kitételeket. Sokkal célszerűbbnek látszik az, hogy egy összetettebb példához kapcsolódóan, egymással összefüggésben villantsuk fel az adatbázistervezés momentumait.

Természetesen a valóság nemigen produkál olyan „esettanulmányt", amely az összes lehetséges szerkezetet, megoldási módot és - számunkra ez is fontos - valamennyi modellezési hibát felvonultatná. Illetve a szerző már találkozott ilyen modellekkel, de azokat nem lehet itt ismertetni. Egyrészt azért nem, mert az nem lenne ildomos; az informatikust is köti a titoktartás. Másrészt a méretek sem teszik lehetővé egy valós eset bemutatását.

Így arra kényszerültünk, hogy összeállítsunk egy mesterséges „állatorvosi ló" esetet, aminek segítségével összegezhetjük a legfontosabb tervezési tudnivalókat.

Előre is figyelmeztetjük a gyengébb idegzetű olvasókat, hogy egy meglehetősen komplikált példa bemutatására vállalkoztunk. Ezért csak azok vágjanak a feladvány megoldásába, akik ahhoz elég erőt és türelmet éreznek magukban. A többieknek azt ajánljuk, hogy mellőzzék a kísérletezést. Viszont olvassák el a példától független összekötő szöveget, amelyben az adatbázistervezés egyéb titkaira hívjuk fel a figyelmet.

Ennek a fejezetnek két célja van. Az első részben egy mintapéldát ismertetünk azzal a szándékkal, hogy felkérjük az olvasót a példában rejlő hibák önálló felfedezésére. A második részben az általunk alkalmazott módszer(ek) lépéseinek a bemutatása közben feltárjuk a mintapélda hibáit és összeállítjuk a hibamentes adatbázistervet.

Ennek a fejezetnek az okos használatához legyen szabad valamit tanácsolnunk. Az olvasó tanulmányozza át a 21.2 pontban leírt példát. Fogalmi szintű modellről van szó. Ezért a tárolási és az ábrázolási aspektusokkal nem kell törődni. Az egyszerűség érdekében csak kevés (nemstrukturális) korlátot fogunk alkalmazni. Így a példa lényegében az egyed-, a tulajdonság- és a kapcsolattípusoknak és ezek összefüggéseinek a kialakítására koncentrál.

Az olvasó próbálja meg feltárni a kiinduló modell hibáit. Majd ezek alapján kísérelje meg saját maga az optimális modell összeállítását. Csak ezután lépjen át a következő pontokra, amelyek tanulmányozása során összevetheti a saját módszerét és eredményét a miénkkel.

Egy fontos megjegyzés: A 21.2 pont minden lényeges tudnivalót tartalmaz. Ámbátor lehet, hogy ezek között vannak alapvetően ellentmondóak. A feladatok közé tartozik az ellentmondások kiszűrése is. Viszont a megoldás nem lesz sikeres, ha az olvasó saját feltételezésekkel él és olyan új tényezőket vezet be, amelyeket a feladat nem ölel fel. A feladat lényege a példában lévő ismeretek mérlegelése és átstrukturálása, ehhez pedig nincs szükség teljesen új fogalmakra.

21.2 A kiinduló modell

Ez a példa ***rendezvényről*** illetve rendezvényekről szól. A rendezvény jellege közömbös. Elképzelhetnek táncházat, kongresszust, kiállítást, tanfolyamot stb. A lényeg az, hogy van egy rendezvényszervező cég, a Rendezvény Iroda (a továbbiakban: RI), amelyet ismeretekkel kell ellátni. Adatbázisba kell szervezni a rendezvénnyel kapcsolatos adatokat. A tervezők az alábbiakban leírt modellt állították össze. Először az egyed-, majd a kapcsolattípusokat ismertetjük.

Minden rendezvénynek van egy beceneve és egy teljes neve. Minden rendezvényért valamelyik város az elsődleges felelős. Ld. 21.1 tábla.

21.1 tábla

RENDEZVÉNY

Becenév	Rendezvény neve	Rend.város	Kezdődátum	Záródátum
DFF	Debreceni Folklór Fesztivál	DEB	xx.yy.zz	xx.mm.nn
BVK96	Budapesti Világkiállítás 1996	BUD	qq.vv.ww	qq,kk.ll

A rendezvényekben érdekelt városok listáját tartalmazza a 21.2 tábla, ami nem más, mint egy kód-megnevezés páros.

21.2 tábla

VÁROS

Városkód	Város
BUD	Budapest
DEB	Debrecen

Az időpontok elrendezése érdekében definiáljuk a rendszerben fontos dátumok listáját a 21.3 tábla szerint. Erről nincs több mondanivaló, ezért nem is adunk meg példaértékeket.

21.3 tábla

DÁTUM (***Dátum***)

Természetesen a rendezvényeken emberek működnek közre, nagyon különböző minőségekben. A 21.4 tábla mutatja a rájuk vonatkozó lényeges ismeretek egy részét.

21.4 tábla

SZEMÉLY

Egyedi az	Név	Cím	Város	Lakcím	Telefon
AB	XXX	Dr	Budapest	C1	1-...
CD	YYY	Id	Szeged	C2	62-...
EF	ZZZ	Prof	Pécs	C3	-
GH	QQQ	Dr	Szeged	C4	62-...

A Telefon opcionális adat. Ha kitöltik, akkor meg kell adni a város körzeti hívószámát is az adat elején.

A személyek nagyon sokféle módon működnek közre egy rendezvényen. Először is ott van az RI saját stábja. A Rendező Iroda bizottsági rendszerben működik. Vannak a rendezvények technikai, pénzügyi, személyi stb. feltételeinek a biztosításával foglalkozó bizottságok. Ezt mutatja a 21.5 tábla.

21.5 tábla

RENDEZŐ EGYSÉG TÍPUS

Rend Egy Típus Kód	RET név
EL	Elnökség
TB	Technikai Bizottság
PB	Pénzügyi Bizottság

Az egységtípusokon belül vannak konkrét, adott részfeladatokkal megbízott egységek. A részfeladatok egy része standard, más része lehet ad-hoc is. Például nem minden eseménynél van szükség új beruházásra. A 21.6 tábla mutatja a részegységeket. A Létrehozás adat az egység életre hívásának az évét jelöli. Az egységek a fentebbi típusokba soroltak.

21.6 tábla

RENDEZŐ EGYSÉG

Rend egység az	Rend egység típus	Rendező egység megnevezés	Létrehozás
TB1	TB	Belső építészet	19xx
TB2	TB	Szállítás	19yy
PB1	PB	Beruházás	19xx
PB2	PB	Bér és díjazás	19zz

A RENDEZŐ EGYSÉG-ek feladatait személyek látják el. Mivel bizottságokról van szó, azoknak van elnöke illetve titkára, több tagja, esetleg adminisztrátora stb. Ezeket a tipikus szerepeket mutatja a 21.7 tábla.

21.7 tábla

RI FELADAT

RI Feladat kód	RI Feladat megnevezés
TBT	TB titkár
TBG	TB tag
PBE	PB elnök

A konkrét személy által ellátott funkciót a 21.8 tábla mutatja. A táblában akkor történik utalás a városra, ha a szerep betöltése adott helyhez kötődik. Egyébként ennek az adatnak nincs tartalma.

21.8 tábla

SZEMÉLY FELADATA

Személy azonosító	***RI feladat***	***RI szerv egység***	Rend város kód
AB	***TBT***	***TB1***	BUD
CD	***TBG***	***TB1***	-
EF	***PBT***	***PB2***	PEC
GH	***TBG***	***TB2***	-

A rendezvény pénzügyi fedezetéhez támogatókat keresnek. Támogató lehet maga a rendező egység, a rendezvényt tartó város vagy bármilyen más szervezet. Ezt mutatja a 21.9 tábla.

21.9 tábla

RENDEZVÉNY TÁMOGATÓ

Rendezvény becenév	***Támogató***	RI Szerv Egység	Rend Város	Egyéb
DFF	***T1***	TB1	-	-
DFF	***T2***	-	DEB	-
BVK96	***T1***	PB1	-	-
BVK96	***T2***	-	-	ABCDE

A RENDEZŐ EGYSÉG az IR kvázi fix stábja. Amikor egy konkrét rendezvényre kerül sor, akkor kétféle ad-hoc bizottságot hoznak létre. Az egyik a rendezvény tartalmáért, a másik annak körülményeiért felel. Lásd a 21.10 táblát.

21.10 tábla

RENDEZVÉNY EGYSÉG

Rendezvény becenév	***Rendezvény egys kód***	Megnevezés
DFF	***PB***	Programbizottság
DFF	***RB***	Rendezőbizottság
BVK96	***PB***	Programbizottság
BVK96	***RB***	Rendezőbizottság

Mármost ezek a bizottságok is emberekből állnak, akik meghatározott feladatokat látnak el a rendezvény aktuális szervezeti egységeiben. A feladatok típusait mutatja a 21.11 tábla.

21.11 tábla

RENDEZVÉNY FELADAT

Rendezvény feladat típus kód	Feladat megnevezés
PBE	Programbizottság elnök
BIR	Program bíráló
RBE	Rendezőbizottság elnök
ZST	Zsűritag

A 21.12 tábla ismerteti, hogy melyik személy milyen feladatokat tölt be a konkrét rendezvény két bizottságának valamelyikében. Vannak olyan speciális feladatok is, amelyek nem kötődnek a szervezeti egységek egyikéhez sem. Ugyanaz a személy több feladatot is elláthat egy rendezvényen. A Kezdődátum mutatja a tisztség betöltésének a kezdetét.

21.12 tábla

SZEMÉLY RENDEZVÉNYI FELADATA

Rendezvény becenév	***Feladattípus***	***Személy az***	Szerv Egys	Kezdődátum
DFF	***PBE***	***AB***	PB	xx.yy.dd
DFF	***PBT***	***CD***	PB	xx.yy.dd
DFF	***PBT***	***EF***	PB	xx.zz.dd
DFF	***XXX***	***CD***	-	xx.qq.dd

A rendezvények programcsoportokból állnak, amelyeket itt műsoroknak fogunk nevezni. Minden műsornak saját neve van. Két rendezvényen nem szerepel ugyanaz a műsor. (Ugyanolyan nevű műsor előfordulhat, de ugyanolyan kulcsú nem.) A Kezdés és a Zárás a műsor kezdetének és befejezésének az időpontja. Egy rendezvényen lehetnek párhuzamosan futó műsorok is. Lásd a 21.13 táblát.

21.13 tábla

MŰSOR

Rendezvény becenév	***Műsorsz***	***Dátum***	***Kezdés***	***Zárás***	Műsornév
DFF	***1***	***D1***	***K1***	***Z1***	Megnyitó
DFF	***2***	***D1***	***K2***	***Z2***	Felvonulás
DFF	***3***	***D2***	***K1***	***Z3***	Táncház
DFF	***4***	***D2***	***K3***	***Z4***	Szalagavató

Az egyes műsorok programokból állnak. Ugyanaz a program nem szerepelhet több rendezvényen ill. egy rendezvény több műsorában. Azonos nevű program előfordulhat, de azonos kulcsú nem. A programok egyedi azonosítóját és címét tartalmazza a 21.14 tábla.

21.14 tábla

PROGRAM

Programsz	Cím
1	XY előadás
2	ZQ tánccsoport
3	YV szavalat

A programokat a műsorokhoz kell rendelni. Ezt mutatja a 21.15 tábla.

21.15 tábla

MÛSOR/PROGRAM

Rendezvény becenév	***Műsor szám***	***Program sz***
DFF	***1***	***28***
DFF	***1***	***68***
DFF	***2***	***22***
DFF	***2***	***55***

Vannak programok, amelyeket nem egy, hanem több személy készít és ad elő. A programok tényleges közreműködőit mutatja a 21.16 tábla. Egy programban több személy vehet részt - és megfordítva. A részvétel minősítésére nincs szükség.

21.16 tábla

KÖZREMŰKÖDŐK

Programsz	***Személy az***
1	***AB***
2	***CD***
3	***EF***

A rendezvény rendezői nyomon akarják követni az egyes programokat. Azt, hogy kit - pontosabban milyen programot - hívtak meg és mikor. Ki fogadta/utasította el a meghívást és mikor. Ki jelentkezett önként stb. A programokkal kapcsolatos tipikus eseményeket a 21.17 tábla sorolja fel.

21.17 tábla

PROGRAM-ESEMÉNYTÍPUS

Program esemény típus kód	Esemény megnevezés
MEG	Meghívás
JEL	Saját jelentkezés
ELB	Elbírálás

Természetesen az eseménytípusokat a konkrét programokra kell vonatkoztatni. Ezt teszi meg a 21.18 tábla.

21.18 tábla

PROGRAM ESEMÉNYEI

Programsz	***Program eseménykód***	***Dátum***	Rendezvény becenév
22	***JEL***	***D1***	DFF
78	***MEG***	***D1***	DFF
22	***ELB***	***D2***	DFF
68	***ELB***	***D3***	BVK96

Az események egyike az elbírálás. Ez persze nem mindegyik programra vonatkozik. Ám amelyikre igen, annak az esetében az RI mindent tudni akar a bírálatról. Ennek egyes ismereteit mutatja a 21.19 tábla.

21.19 tábla

BÍRÁLAT

Rendezvény becenév	***Programsz***	***Dátum***	***Személy az***	Szerep	Esemény
DFF	***22***	***D1***	***KL***	BIR	ELB
DFF	***22***	***D1***	***LJ***	BIR	ELB
DFF	***22***	***D2***	***FH***	BIR	ELB
DFF	***68***	***D2***	***KL***	BIR	ELB

A rendezvénnyel kapcsolatos személyeket tájékoztatni kell az eseményekről. A részvétel minőségének megfelelően vagy díjat kell kapni tőlük, vagy éppen ellenkezőleg, fizetni kell a számukra. A személyekre vonatkozó események típusait mutatja a 21.20 tábla.

21.20 tábla

SZEMÉLY-ESEMÉNYTÍPUS

Személy esemény típus kód	Személy esemény megnevezés
MEG	Meghívás
VIS	Visszajelzés
SZA	Számlázás

Persze mindezeket az eseményféléket konkrét személyekhez kell kapcsolni. Erre szolgál a 21.21 tábla.

21.21 tábla

SZEMÉLY ESEMÉNYEI

Becenév	***Személy eseménykód***	***Dátum***	***Egyedi azonosító***
DFF	***MEG***	***D1***	***AB***
DFF	***VIS***	***D1***	***AB***
DFF	***SZA***	***D2***	***AB***
BVK96	***MEG***	***D3***	***CD***

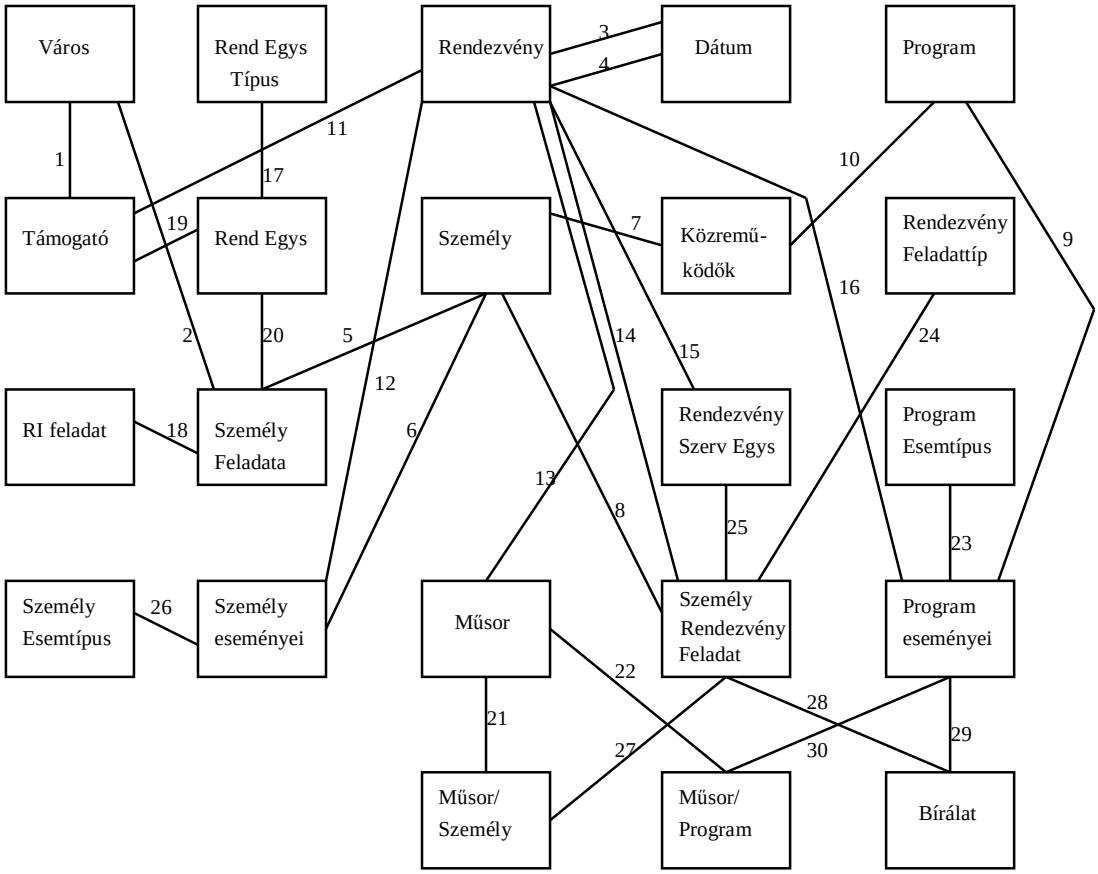
Végre elérkeztünk utolsó táblánkhoz. Az RI szeretné ismerni, hogy a rendezvény egyes műsorain kik és milyen minőségben vesznek részt. Vannak, akiknek a szerepe általános és akadnak olyanok is, akiknek a feladata konkrét programhoz kötődik. Ezért a program megadása opcionális. A PRO feladattípus programbeli előadót jelent. Lásd a 21.22 táblát.

21.22 tábla

MŰSOR/SZEMÉLY

Rendezvény becenév	***Feladattípus***	***Személy az***	***Műsor***	Programsz
DFF	***PRO***	***AB***	***1***	1
DFF	***YYY***	***CD***	***1***	-
DFF	***ZST***	***EF***	***2***	22
DFF	***XXX***	***CD***	***2***	-

Az egyedtípusokat és összefüggéseiket a 21.1 ábra mutatja.



21.1 ábra: Az esettanulmány diagramja

Az ábrán minden lefelé mutató kapcsolat 1:N fokú. Ezért nincsenek feltüntetve a nyilak. A kapcsolattípusok listáját a számoknak megfelelően a 21.23 tábla tartalmazza.

21.23 tábla

	FÖLÉRENDELT	ALÁRENDELT
1	VÁROS	RENDEZVÉNY TÁMOGATÓ
2	VÁROS	SZEMÉLY FELADATA
3	DÁTUM	RENDEZVÉNY
4	DÁTUM	RENDEZVÉNY
5	SZEMÉLY	SZEMÉLY FELADATA
6	SZEMÉLY	SZEMÉLY ESEMÉNYEI
7	SZEMÉLY	KÖZREMŰKÖDŐK
8	SZEMÉLY	SZEMÉLY RENDEZVÉNYI FELADATA
9	PROGRAM	KÖZREMŰKÖDŐK
10	PROGRAM	PROGRAM ESEMÉNY
11	RENDEZVÉNY	RENDEZVÉNY TÁMOGATÓ
12	RENDEZVÉNY	SZEMÉLY ESEMÉNYEI
13	RENDEZVÉNY	MŰSOR
14	RENDEZVÉNY	SZEMÉLY RENDEZVÉNYI FELADATA
15	RENDEZVÉNY	RENDEZVÉNY EGYSÉG
16	RENDEZVÉNY	PROGRAM ESEMÉNYEI
17	RENDEZŐ EGYSÉG TÍPUS	RENDEZŐ EGYSÉG
18	RI FELADAT	SZEMÉLY FELADATA
19	RENDEZŐ EGYSÉG	RENDEZVÉNY TÁMOGATÓ
21	RENDEZŐ EGYSÉG	SZEMÉLY FELADATA
21	MŰSOR	MŰSOR/SZEMÉLY
22	MŰSOR	MŰSOR/PROGRAM
23	PROGRAM ESEMÉNYTÍPUS	PROGRAM ESEMÉNYEI
24	RENDEZVÉNY FELADAT	SZEMÉLY RENDEZVÉNYI FELADATA
25	RENDEZVÉNY EGYSÉG	SZEMÉLY RENDEZVÉNYI FELADATA
26	SZEMÉLY ESEMÉNYTÍPUS	SZEMÉLY/ESEMÉNY
27	SZEMÉLY RENDEZVÉNYI FELADATA	MŰSOR/SZEMÉLY
28	SZEMÉLY RENDEZVÉNYI FELADATA	BÍRÁLAT
29	PROGRAM ESEMÉNY	BÍRÁLAT

Az olvasó most értékelheti az esettanulmány adatbázistervét. Próbálja meg feltárni annak hibáit a saját módszere szerint. Majd próbálja meg felvázolni a helyes modellt. Fel kell hívnunk a figyelmet arra, hogy a példa egy alapvető ***ellentmondást*** tartalmaz. Amíg erre rá nem jön az olvasó, addig nem fog jó megoldást találni. Segítségként annyit mondunk, hogy figyelje a kulcsokat és a kérdés az, hogy valami egy-e vagy több-e. Megoldásunkban mi a „többet” feltételezzük.

21.3 Az esettanulmány megoldása elé

A további pontokban számos titkot fogunk elárulni arról, hogy miképpen végezzük az adatbázistervek értékelését. Persze erre sokan így szólnak magukban: hja, ha már van mit elemezni, akkor könnyű. De miért nem a nulláról indul az esettanulmány?

Válaszunk többszörös. Az adott példa kiinduló helyzetének a leírása igen terjedelmes lett volna. Valószínűleg a jelenleginél is sokkal több kétséget ébresztett volna az olvasóban. Könyvön keresztül pedig nem lehet kérdezni, interjúkat csinálni - vagyis mímelni a valóságot. A terjedelmes szöveges leírásból mindenki egy saját tervet készített volna, amelyet a szerző nem ismerhet és így nem tudná szemléltetni a tipikus modellezési hibákat és azok kiküszöbölésének a módját.

Azután gondoljuk meg, hogy manapság az adatbázisok nem a semmiből születnek. Többnyire már vannak számítógépen kezelt adataink. Az esettanulmányt úgy is fel lehet fogni, mintha az egyes táblái már valóban léteznének és az lenne a feladat, hogy a meglévő - a táblaképeknek megfelelő - állományainkat integráljuk.

Végül nem az a gondunk, hogy a tervezők elsőre tökéletlen adatbázisterveket készítenek. Hanem az, hogy a szinte ösztönösen megálmodott modellt nem veszi kritika alá. Nem értékelik a saját terveiket. Pedig némi figyelemmel sokat tudnának javítani az első változaton. Ennek a tökéletesítésnek a technikáját kívánjuk feltárni az alábbiakban.

A megoldás ismertetése közben rá fogunk mutatni arra, hogy az elemzésnek melyik pontján és milyen módon használható egy jó ***számítógépes elemzési segédeszköz***.

21.4 Ismerkedés az elemzendő tervvel

Mivel a továbbiakban a saját módszeremet ismertetem, át fogok térni az egyes szám első személyre. Remélem megértik, hogy úgy könnyebb magyaráznom.

Amikor kezemhez kapom egy adatbázis tervének a dokumentációját, legelőször ***formai*** szempontokból vizsgálom azt. Úgy, mint bármilyen dokumentációt, sőt könyvet. Az első kérdésem az, hogy van-e tartalomjegyzéke a leírásnak és az megfelel-e az előző fejezetben az adatbázisterv dokumentálásával kapcsolatosan megadott követelményeknek. Talán nevetségesen hangzik, de az oldalszámozás hiánya, a nem egységes kép, a kereszthivatkozások elmaradása azonnal gyanút kelt bennem. Esettanulmányunk leírása aligha felel meg egy adatmodell formai dokumentációs követelményeinek. A formáról - például a diagramokban alkalmazott konvenciókról, a listákban használt kódokról - még sokat lehetne beszélni, de a formai aspektusok értékelését itt lezárom.

Második lépésem az, hogy megpróbálom bemérni az adatmodell ***bonyolultságát***. Ehhez bizonyos számokat és - ha van - a diagramot használom fel. Mintapéldánk csak 22 egyed- és 29 kapcsolattípust tartalmaz, tehát abszolút értelemben „kicsi” modellnek számít. (Dolgoztam már több száz egyed- és kapcsolattípust felölelő terven is.) Nem akad olyan tábla, amelyben az attribútumok száma meghaladná a hatot. Mármost az olyan modell, amely kevés egyedtípust tartalmaz úgy, hogy azokhoz sok tulajdonság kapcsolódik, általában egyszerűnek tekinthető. Ezzel szemben az olyan terv, amelyben a tulajdonságokhoz képest magas az egyedtípusok száma, többnyire bonyolultnak számít.

A komplexitás behatárolásában az ***azonosítók első áttekintése*** is segíthet. A modell egyszerűsége az elemi - egytagú - kulcsok hányadával nagyjából arányos. A mi példánkban magas az összetett kulcsok aránya. Ez a tény, valamint egy pillantás a Bachman-diagramra azt sejteti, hogy kis méretén belül mintapéldánk meglehetősen komplikált. Túl magas benne a kulcs/kulcsrész tulajdonságok aránya a leírókhoz képest. Ez nem baj, ez a helyzet.

Az összetettségre utalnak a látható és a rejtett ***altípusok*** is. Az esettanulmányban nincsenek expliciten megfogalmazott altípusok. Csakis a személyek szerteágazó szerepe mutatja a kötődések bonyolultságát. Ez nem jelenti feltétlenül azt, hogy majd altípusokat kell meghatározni. Persze ha más ismeretek is fontosak lennének, akkor felvetődhetne a személy specializálásának a kérdése. (Példánkban erre nem lesz szükség.)

Az összetettség megállapítása azért fontos, mert a bonyolult modellnél mindig él bennem a gyanú, hogy a kapcsolattípusokat helytelenül határozták meg. Ezt várom az esettanulmány példájától is.

Az értékelési feladat nagyon megnehezül, ha a terv hiányos - mint a miénk - és nem egyértelmű. Ezért az első átnézéskor megpróbálom behatárolni az ***egyértelműség*** szintjét. Az esettanulmány tervéről azonnal megállapítható, hogy hemzsegnek benne a homonimák és a szinonimák. A homonimákra a nem-minősített nevekből (pl. Cím, Megnevezés) lehet következtetni. Persze a minősített név is lehet homonima (Kezdődátum). Mivel tervezőnk nem alkalmazott ***névszabványokat***, egészen biztos, hogy a technikai szinonimák garmadájával kell számolnunk. Végül a tervező láthatólag nem ismeri a ***szerepnév*** lényegét. Ezt az össze-vissza minősítésből érzékelem. Emiatt és a ***szerepek*** hiánya miatt feltehető, hogy a terv kapcsolati rendszere nem lesz egyértelmű.

21.5 Az egyértelműség elemzése

Mivel olyan modellt nem lehet normalizálni, amely nem egyértelmű, legelső feladat az egyértelműség megteremtése. Bár mintapéldánk mérete nem nagy, a szerkezet mégis eléggé bonyolult. Emiatt feltétlenül számítógépes ***adatszótárt*** használok az elemzéshez. Sorra veszem a táblaképeket és szótárba viszem az egyed-tulajdonság-viszonyokat. Eközben fokozatosan bővítem a külön egyed- és tulajdonságlistát. Tehát amikor elkönyvelem, hogy a SZEMÉLY egyedhez kapcsolódik a Név tulajdonság, akkor e viszony mellett rögzítem az egyedlistán a SZEMÉLY, a tulajdonságlistán a Név tételt.

Persze ezt nem egészen így teszem. Egyrészt - szemben a mintapélda tervezőjével - saját egységes ***konvenciómnak*** megfelelően írom át a neveket. Másrészt általában kerülöm a ***rövidítéseket***. Pl. a „Rend Egy Típus Kód” nevet én így írom: „Rendező-egység-típuskód”. A modellek egyik hibája a helytelen ***minősítés***. Ezért a túlminősített neveket egyszerűsítem. Pl. a „Rend város kód”-ban a jelző nem korlátoz, ezért feleslegesen minősít. Bármelyik város lehet rendező. Ezért itt az egyszerű Városkód nevet használom. Ezzel szemben az alulminősített neveket kibővítem. Így lesz a program Cím-éből Programcím. A rosszul beszélő neveket átírom. Tehát a személy zavaró Cím adatából Titulus lesz.

Vegye észre az olvasó, hogy ezzel a „keresztelővel” eleve kiszűröm a ***homonimák*** és ***szinonimák*** egy részét anélkül, hogy kutatnom kellene utánuk. A kiinduló modellben a Cím homonima volt - már nem az. A „Rend város kód” és a „Városkód” technikai szinonima volt - amit máris felszámoltam.

Amennyiben a szótár feltöltésénél mégis kétszeres tételre bukkanok, úgy több eset lehetséges. Két tényezőnek nem lehet azonos a neve. Ezért helytelen, hogy a kapcsolatlistán két DÁTUM - RENDEZVÉNY nevű kapcsolat található. Persze a kapcsolat kettős, de nevük ennek ellenére nem lehet azonos. A tervezők ritkán követik el azt a hibát, hogy két egyedtípusnak azonos nevet adnak. Ha ilyen mégis fellépne, a homonimát azonnal meg kell szüntetni. A legtöbb baj a tulajdonságtípusokkal van éppen azért, mert a tervek nem tartalmaznak tulajdonságlistát. Így a tulajdonságokat a tervező csak egyedekhez kötötten látja. Ebből szükségszerűen következik a homonimák és szinonimák tömege.

Amikor több egyedben is szerepel ugyanolyan nevű tulajdonság, három helyzetet kell vizsgálni. A legrosszabb esetben a tulajdonság ***homonima***. Ekkor jön az átnevezés, amit általában mindkét egyedben célszerű végrehajtani. Nem jó az, ha a Cím-ből az egyik egyedben Programcím lesz, a másikban Cím marad. A legjobb esetben a tulajdonság olyan tétel, amely az egyik egyedben kulcs, a másikban kapcsoló. Ekkor nincs teendő. (Persze ennek feltárásához a szótárban vezetni

kell a szerepeket.) Ha a tulajdonság leíró, akkor néha nincs teendő (pl. Dátum), néha az ember inkább minősít (ilyen eset nincs a példában).

Az embernek persze azonnal szemet szúr néhány ***szinonima***. Az eltérő írásmódból fakadó technikaiakat (pl. „RI szerv egység" és „RI Szerv Egység") könnyű megtalálni. Azonban a teljesen eltérőeket (pl. a személy kulcsa az „Egyedi az" és a „Személy az") nehéz felfedezni. A feltárt szinonimákat természetesen azonnal megszüntetjük. Majd a szótár feltöltése után két kis trükköt alkalmazunk. Először készítünk egy ABC-sorrendű listát a tulajdonságokról. Ebben azonnal felfedezzük - ha addig nem tettük volna -, hogy a „Személy az" és a „Személy azonosító" tételek slamposságból fakadó technikai szinonimák. Egy kis programmal azt is könnyen megállapíthatjuk, hogy az egyik név része-e a másiknak. Így felfedezhetjük, hogy a „Rendezvény becenév" és a „Becenév" szinonimák.

Az esetleg a modellben maradó szinonimák és homonimák kiszűrésének van egy dörzsölt módja is, amelyre majd a kapcsolatelemzésnél utalunk. Lásd a 21.9 pontot. Most még a ***szerepnevekkel*** kell foglalkoznunk. Én a modellben az egyszerűség kedvéért általában csak korlátozó szerepneveket használok. Minősítőket csak akkor, ha egy egyedben több azonos tartományból származó tétel van. Lásd a RENDEZVÉNY Kezdődátum és Záródátum tételét. A modell nem egyértelmű, mert nem jelenti ki e két tényezőről, hogy a Dátum szerepnevei. Miért hiba ez? A tervező két kapcsolatot feltételezett - helyesen - a DÁTUM és a RENDEZVÉNY között arra alapozva, hogy a kezdődátum és a záródátum is dátum. Igen ám, de elfeledkezett a DÁTUM - SZEMÉLY RENDEZVÉNYI FELADATA kapcsolatról, noha az utóbbi egyedben is van Kezdődátum nevű adat. Ez amúgy homonima. A lényeg az, hogy a szerepnevek explicit megadásának az elmaradása ilyen hiányosságokhoz vezethet. Ezért a szótár feltöltésénél én mindig expliciten utalok az elsődleges tulajdonság (Dátum) és a szerepnév (Kezdődátum) közötti összefüggésre.

Mi a teendő a Kezdődátum homonimával? És mi lenne a teendő ezzel az adattal, ha az csak a SZEMÉLY RENDEZVÉNYI FELADATA egyedben szerepelne? Nos, a felesleges minősítés mindig plusz modelltényezőre (szerepnév-viszony) vezet. Ezért kerülöm én a nem-korlátozó szerepneveket. A vonatkozó egyedben a sima Dátum nevet alkalmaznám, ezzel megszüntetve a homonimát és megteremtve a DÁTUM felé a kapcsolatot. Ha valaki a minősítést kedveli, ám tegye. Ekkor viszont a Kezdődátum homonima kiirtandó. Például a Feladat-kezdődátum nevet kell adni neki. Erről pedig expliciten ki kell jelenteni, hogy a Dátum szerepneve.

21.6 Azonosító- és azonosságelemzés

A tervezett táblákban számos probléma léphet fel. Nagyon sokszor megsértik az ***azonosítókra vonatkozó korlátokat***. Több egyednek ugyanaz a tulajdonság a kulcsa, vagy megfordítva, egy egyednek több kulcsot jelölnek ki nem ismerve az alternáló kulcs intézményét. Mintapéldánkban nem fordul elő ilyen hiba.

Trükkös probléma a ***szerkezeti azonosság***. Egy adatmodellben a következő részszerkezetek fordulnak elő: egyed-tulajdonság-viszony (táblakép), kapcsolat és tulajdonságstruktúra. Az utóbbin belül származtatási, csoport és szerepnév szerkezeteket különböztetünk meg (ld. 18.8 pont). E struktúrarészek furcsa problémákat okozhatnak. Csak három példát említünk.

Tegyük fel, hogy a tervező modellezi a ***csoportokat***. Mintapéldánk alkotója ezt nem tette meg. Néha előfordul, hogy a tervben X {A+B+C} és Y {A+B+C} csoportok találhatók. Mivel ezek felépítése azonos, de nevük eltérő, az ilyen esetekben ***strukturális szinonimákról*** beszélünk. Ezeket ki kell szűrni. Én az összetett kulcsokat mindig külön névvel látom el és szótáramba viszem az ehhez tapadó csoportstruktúrát. Ennek számos oka van. Itt csak a jelzett szinonima kiszűrhetőségére utalok. Mert azt könnyű felfedezni, hogy két egyednek megegyezik az egyszerű

A azonosítója. Ámde sok egyedet tartalmazó modellben azt átlátni, hogy itt is, meg ott is szerepel az A+B+C+D összetett kulcs, már nem olyan egyszerű. Az explicit csoportmeghatározást követő azonosság-elemzés segít e hiba feltárásában. (N.B.: Ugyanez a megállapítás a származtatott adatokra is vonatkozik. Ha az X és az Y adatnak ugyanazok a tényezői, akkor miért szükséges a két név - szinonima -, ha a származtatási algoritmus is azonos?)

Szerkezeti szinonima a ***kapcsolatoknál*** is felléphet. Az X kapcsolat is az A és a B egyedet köti össze, meg az Y viszony is. Akkor, ha a két egyednek csak egy közös kapcsolótulajdonsága van, a kapcsolatmeghatározás redundáns; megszüntetendő tévedésből született. Ám e hiba feltárásához az szükséges, hogy a kapcsolati struktúrákat az adatszótárba vigyük és elemezzük a fölé/alárendelt párosok azonosságát.

Végül az is megtörténik, hogy a tervező két X (***A***, C, D) és Y (***B***, C, D) egyedtípust kreál. A két egyed minden leíró tulajdonságában megegyezik. Ekkor a tervező vagy nem alkalmazta a generalizáció műveletét, vagy - tévedésből - két azonos egyedtípust alkotott. Mindkét helyzet megoldandó és mindkettőre az azonosságelemzés irányítja a figyelmet.

Mintapéldánkban nem szerepelnek a fenti gondok. A tapasztalatlan szakember akár azt is mondhatná, hogy ő ekkora - elnézést - ökörségeket az életben nem követne el. Csakhogy nem szabad arról elfeledkezni, hogy egy valóban összetett adatmodellt nem egyetlen tervező állít össze. Az adatbázis - egy, ám a tervezője sok. Ezért bizony előfordulhat, hogy két tervező ugyanazt vagy nagyjából ugyanazt az egyedet álmodja meg eltérő neveket adva a két tényezőnek. (N.B.: Saját elemzőnk ***átfedési százalékot*** is számít. Kimutatja, hogy két egyedtípusnak N százalékban azonosak a tulajdonságai. Ha az N magas, akkor a tervező elgondolkozhat az általánosításon.)

Most pedig térjünk a mindennaposabb, gyakoribb problémákra. Nevezetesen az azonosító helytelen meghatározására. Az ***azonosító kijelölése*** terén számos rossz megoldással találkozunk. Elsősorban az összetett azonosítók okoznak gondokat. Nevezetesen vagy azt, hogy az ***összetétel*** a valóságban képtelen azonosítani az egyedeket, mert az összetétel értéke nem egyedi. Vagy azt, hogy az összetétel felesleges tényezőket tartalmaz. Ezért én, mielőtt bármiféle mélyelemzésbe vagy pláne normalizálásba kezdenék, megvizsgálom a kulcsok megfelelőségét. A normalizálás nem lehet sikeres, ha a kulcsot rosszul adtuk meg.

Mintapéldánkban a MŰSOR egyed kulcsa katasztrofális. Egy rendezvényen több műsor szerepelhet, de ugyanaz a műsor nem fordulhat elő több rendezvényen. Az pedig evidens, hogy egy műsor nem kezdődhet és végződhet különböző időpontokban. Ezért az egyed kulcsa túlhatározott. Az azonosításra elegendő a Konferencia-becenév és a Műsor-sorszám együttese. Teljesen felesleges a behatároláshoz a Dátum, a Kezdés és a Zárás. Ezek valójában csak leíró tulajdonságok.

Most térünk ki a példa ***alapvető ellentmondására***, mert az az azonosítással kapcsolatos. A PROGRAM egyed kapcsán kijelentettük, hogy minden program kulcsa egyedi. Ha ez így igaz, akkor a Programszám funkcionálisan meghatározza a Műsor-sorszámot, ill. magát a Rendezvény-becenév tételt is. Hiszen ugyanaz a program nem szerepelhet több műsorban, sőt rendezvényen. Ha ez igaz, akkor, rossz a MŰSOR/PROGRAM egyedtípus azonosítója. Hiszen azt sejteti, hogy a program és a műsor funkcionálisan független.

Ehhez képest a Rendezvény-becenév és a Programszám tulajdonság a mintapélda számos egyedében mindenféle variációban együtt szerepel. Ha a Programszám egyedi, akkor meghatározza a Rendezvény-becenév tételt, tehát a két adat csak egyetlen egyedben tűnhetne fel együttesen: magában a PROGRAM-ban. (Emlékezzünk rá, hogy ha egy tulajdonság másikat meghatároz, akkor az utóbbinak az előbbi által azonosított egyedben van a helye. Ezt mondja ki a teljességi szabály. Ld. a 17.2 pontot.)

Vajon miképpen jöttem rá én erre az alapvető turpisságra? Nos, egyrészt figyelmesen olvasok. Ez sokszor kevés lenne. Viszont kis normalizáló segédeszközöm azonnal felfedezte a MŰSOR/PROGRAM egyedben lévő belső kulcstörő függést (ld. 16.4 pont). A Programszám meghatározza a Rendezvény-becenév adatot a leírás szerint, viszont ugyanakkor egy kulcsban is szerepel vele. Nono, ez nem stimmel.

Itt dönteni kell. Kétszeresen is az „egy vagy több” kérdése merül fel. Nézzék csak meg a 21.2 pont első mondatát! Rendezvény vagy rendezvények... A tervező nem tudott dönteni a modell hatóköréről (ld. 19.2.2 alpont). Eredetileg egyetlen rendezvény tervezéséből indult ki, amely esetben a program egyedisége nem kérdéses. Majd hirtelen átváltott a több rendezvény gondolatára. Ekkor viszont felmerül az a kérdés, hogy ugyanaz a program szerepelhet-e több rendezvényen. Ha nem, akkor a PROGRAM kulcsa meghatározza a RENDEZVÉNY-ét. Ekkor a Programszám és a „Rendezvény becenév” sohasem szerepelhetne együtt a PROGRAM-on kívül egyetlen más egyedtípusban sem. Viszont ezt mintapéldánk esetében számtalanszor megteszi. Ezért javaslom azt, hogy maradjunk meg a „több”-nél, vagyis annál, hogy egy program több rendezvényen is megjelenhet. Lásd a 21.2 pont utolsó mondatát.

Ezzel az azonosító- és az azonosságelemzés témakörét egyelőre befejeztük. Vegye észre az olvasó, hogy ez a pont még mindig az ***egyértelműség*** körül forgott. Az egyedek egyértelmű azonosításáról és a struktúrák egyértelműségéről volt szó.

21.7 A példa átmenetileg javított változata

Azonosítókat kellett rendbe tennünk. Homonimákat és szinonimákat megszüntetnünk. Implicit szerepnév és csoport szerkezeteket explicitté tennünk. A kiinduló modellen annyi mindent kellett változtatnunk, hogy nem haladhatunk tovább az eddigi változat rögzítése nélkül. Magukat az eredeti táblázatokat nem kívánnám megismételni, ezért az alábbiakban csak a javított listaképeket mutatom. A csoportokat {}, a szerepneveket [ ] jelek között részletezem. Az összetett kulcsokat az egyed nevéhez tett „cs” - csoport - betűvel jelölöm. Az ABC-sorrend elvét meghágva, az áttekinthetőség kedvéért a korábbi táblák sorrendjében ismertetem az egyedtípusokat.

RENDEZVÉNY (***Rendezvény-becenév***, Rendezvénynév, Városkód, Kezdődátum, Záródátum)
Dátum [Kezdődátum, Záródátum]
VÁROS (***Városkód***, Városnév)
DÁTUM (***Dátum***)
SZEMÉLY (***Egyedi-az***, Személynév, Titulus, Városnév, Lakcím, Telefon)
RENDEZŐ EGYSÉG TÍPUS (***Rendező-egység-típuskód***, Rendező-egység-típusnév)
RENDEZŐ EGYSÉG (***Rendező-egység-azonosító***, Rendező-egység-típuskód, Rendező-egység-megnevezés, Létrehozás-éve)
RENDEZŐ EGYSÉG FELADAT (***Rendező-egység-feladatkód***, Rendező-egység-feladat-név)
SZEMÉLY RENDEZŐ FELADAT (***Személy-rendező-feladat-cs***, Városkód)
Személy-rendező-feladat-cs {Személy-azonosító+Rendező-egység-feladatkód+Rendező-egység-azonosító}
RENDEZVÉNY TÁMOGATÓ (***Rendezvény-támogató-cs***, Rendező-egység-azonosító, Városkód, Egyéb-szervezetkód)
Rendezvény-támogató-cs {Rendezvény-becenév+Támogató-sorszám}
RENDEZVÉNY EGYSÉG TÍPUS (***Rendezvény-egységkód***, Rendezvény-egység-megnevezés)
RENDEZVÉNY EGYSÉG (***Rendezvény-egység-cs,*** Rendezvény-egység-megnevezés)
Rendezvény-egység-cs {Rendezvény-becenév+Rendezvény-egységkód}
RENDEZVÉNY FELADAT (***Rendezvény-feladat-típuskód***, Rendezvény-feladat-megnevezés)
SZEMÉLY RENDEZVÉNY FELADAT (***Személy-rendezvény-feladat-cs***, Rendezvény-egységkód, Dátum)
Személy-rendezvény-feladat-cs {Rendezvény-becenév+Rendezvény-feladat-típus-kód+Személyi-azonosító}

MŰSOR (***Műsor-cs***, Dátum, Kezdő-időpont, Záró-időpont, Műsornév)
Műsor-cs {Rendezvény-becenév+Műsor-sorszám}
PROGRAM (***Programszám***, Programcím)
MŰSOR/PROGRAM (***Műsor/program-cs***)
Műsor/program-cs{Rendezvény-becenév+Műsor-sorszám+Programszám}
KÖZREMŰKÖDŐ (***Közreműködő-cs***)
Közreműködő-cs{Programszám+Személy-azonosító}
PROGRAM ESEMÉNYTÍPUS (***Program-esemény-típuskód***, Program-esemény-típus-megnevezés)
PROGRAM ESEMÉNY (***Program-esemény-cs***, Rendezvény-becenév)
Program-esemény-cs {Programszám+Program-esemény-típuskód+Dátum}
BÍRÁLAT (***Bírálat-cs***, Rendezvény-feladat-típuskód, Program-esemény-típuskód)
Bírálat-cs {Rendezvény-becenév+Programsszám+Dátum+Személy-azonosító}
SZEMÉLY ESEMÉNYTÍPUS (***Személy-esemény-típuskód***, Személy-esemény-típus-megnevezés)
SZEMÉLY ESEMÉNY (***Személy-esemény-cs***)
Személy-esemény-cs {Rendezvény-becenév+Személy-esemény-típuskód+Dátum+Egyedi-azonosító}
MŰSOR/SZEMÉLY (***Műsor/személy-cs***, Programszám)
Műsor/személy-cs {Rendezvény-becenév+Rendezvény-feladat-típuskód+Személy-azonosító+Műsor-sorszám}

Tudom, hogy a fenti lista összeállítása mennyire nehéz feladat. Ehhez több megjegyzésem van. Egyrészt nem kínlódtam volna ennyit, ha az eredeti terv egyértelmű lett volna. Másrészt a nevek hosszúak, de magyarázóak. Ezeken lehet még faragni: eddig szándékosan nem tettem. Mert például a REFETIKOD névből az olvasó nem tudná, hogy a Rendezvény-feladat-típuskódra gondolok. Az adatmodellnek beszélőnek kell lennie, ami sok nyűggel jár. Harmadrészt ez a modell már majdnem egyértelmű - és ennyit megért az eddigi kínlódás.

21.8 A rejtett redundanciák elemzése

A fenti „majdnem” szó kicsit aggaszt. Nem állhatunk meg egy féligkész elemzésnél. Az adatmodellezéssel úgy van, mint a könyveléssel. Pár forintos eltérés utáni kutatás tárja fel, hogy milliókról van szó. Egyetlen adatmodellezési hiba is számos mást szülhet. Ezért folytatnom kell a nyomozást. Először a szemmel látható „apró” hibák felé fordulok.

A mintasorokkal szemléltetett táblaképek azért hasznosak, mert segítségükkel feltárhatjuk a modell rejtett redundanciáit. Ebben a pontban háromféle titkos fizikai adatátfedésre irányítjuk a figyelmet.

A 21.6 táblában a „Rend egység típus” tétel értéke mindig megegyezik a „Rend egység az” tulajdonság értékének az első két karakterével. Itt az ***implicit csoportból*** következő fizikai redundancia tipikus esetével állunk szemben. Lásd a 19.4 pontot. Az utóbbi tényező olyan csoport, amely impliciten tartalmazza az előbbit. Jogos tehát a kérdés, hogy miért kell egy egyed-előfordulás sorának kétszer tartalmaznia ugyanazt az ismeretet? A kézenfekvő megoldás az, hogy az azonosítót explicit csoportként adjuk meg. Ettől maga a modell persze bonyolultabb lesz. Viszont elkerüljük a karbantartási anomáliákat. Ezért én bizony minden egyedben explicitté fogom tenni a csoportot.

A másik probléma a ***generikus tulajdonságokkal*** kapcsolatos. Lásd a 19.4 pontot. A 21.19 táblában a Szerep és az Esemény olyan tulajdonságok, amelyeknek az értéke mindig ugyanaz.

Akkor pedig nyilvánvalóan feleslegesen terhelik az egyedet. Hiszen annak lényege a bírálat, amelyben a szerep mindig bíráló, az esemény mindig elbírálás. Az ilyen tételeket ki kell irtani. Ennek során nagy óvatossággal kell eljárni. Meg kell nézni, hogy valóban mindig csak egyféle-e az érték. Ugyanis a minták félrevezetőek is lehetnek. Vessünk csak egy pillantást a 21.12 táblára. Abban a „Szerv Egys" tétel mindig csak „PB" vagy üres értéket vesz fel. Ám ez megtévesztő: a tétel nem generikus tulajdonság. Hiszen a tábla előtti szöveg azt sejteti, hogy a kétféle bizottságban betöltött feladatokról van szó.

Most jutottunk el az alapvető ***normálformák*** kérdéséhez. Most elárulunk egy újabb titkot. A helytelen normálalakok csak akkor fedezhetők fel automatával, ha a helytelenül függő tulajdonság nyílt logikai redundanciát okoz. Ha a Vevőnév a VEVŐ és a RENDELÉS egyedben is szerepel a meghatározó Vevőkód tétellel együtt, akkor a tranzitív függést bármilyen automata könnyen feltárja. Ám ha a Vevőnév csak a RENDELÉS egyedben található a Vevőkóddal együtt, úgy a tranzitivitás automatikus feltárására nincs mód. Ekkor „kézzel" kell normalizálni. Mármost a normálformahiba mindig fizikai redundanciával jár. Ezért feltárásukban ismét csak a mintasorok segítenek. A 21.10 táblában feltűnő a Megnevezés redundanciája. Ez a tétel részlegesen függ a kulcstól. A SZERVEZÉSI EGYSÉG nincs 2NF alakban. Ezért abból ki kell emelni ezt a tulajdonságot egy új egyedbe.

21.9 Kapcsolatelemzés

Az egyedek belső szerkezetének az elemzése után ideje a külső szerkezettel is foglalkoznunk. Vizsgáljuk meg a mintapélda által sugallt kapcsolatokat (ld. 21.23 tábla). Ilyenkor lefuttatom kis elemző programomat és döbbenten tapasztalom a következőket: A beígért 29 kapcsolattípus több tétele nem megalapozott az egyedtípusok belső szerkezete - kapcsoló tulajdonságai - alapján, viszont ugyanakkor sok létező kapcsolatot nem definiált a mintapélda szerzője. Most ne részletezzük a többleteket és a hiányokat: az adatmodell nem konnektív (ld. 17.2 pont). A SZEMÉLY egyed máshoz nem kapcsolható.

Persze a problémák egy részét támogató automata nélkül is fel lehet fedezni. Arra egy pillanat alatt rájövök, hogy a Bachman-diagramon harminc, a kapcsolatlistán viszont csak 29 viszony van feltüntetve. Nem ez a nüansznyi hiba aggaszt. A kézzel készített modellben mindig lesznek ilyen következetlenségek. Azokat csak akkor kerülhetjük el, ha az adatmodellt eleve számítógéppel készítjük. Akkor is előfordul, hogy a tervező által expliciten megadott és az egyedtípusok kapcsoló tulajdonságai által igazolt kapcsolatok eltérnek egymástól. A fejlesztő olyan kapcsolatot is megad, amelyet nem támogat kapcsoló tényező, viszont ugyanakkor nem utal expliciten olyan kapcsolatokra, amelyeket pedig valódi kapcsoló tulajdonság alapoz meg.

Két egyedtípus között akkor definiálhatunk nem-valós kapcsolatot a modellezés elvei szerint, ha a közös nevű tulajdonság az egyiknek kulcsa és a másikban is szerepel, de nem ugyanazon tartalommal. Magyarul: a kapcsoló tulajdonság neve ***homonima***. Ezen kívül az is gyakran előfordul, hogy a tervező közvetlenül is meghatároz egy valójában csak közvetett kapcsolatot. Ha létezik az A, az A+B és az A+B+C által azonosított egyed, akkor az első és a harmadik kapcsolata csak közvetett - magyarul: tranzitív. Végül megtörténik, hogy a tervező M:N-es viszonyt ad meg kapcsolatként, mivel két egyedtípusnak közös tulajdonsága vagy tulajdonságai vannak. Elfeledkezik arról a szabályról, hogy kapcsolat csak akkor definiálható, ha a közös tulajdonság vagy csoport az egyik egyed azonosítója.

Ha viszont a feltételezett kapcsolat hiányzik, akkor két eset lehetséges. A viszony azért nem mutatkozik, mert a kapcsoló tulajdonság ***szinonima***. Hát persze, hogy inkonnektív a mintapélda modellje, amikor a SZEMÉLY kulcsát egyetlen kapcsolódó egyedben sem nevezzük az „Egyedi

az” néven. A másik eset trükkösebb és az összetett azonosítókkal függ össze. A tervező nem határoz meg csoportot. Ezért nem is veszi észre, hogy az A+B által azonosított egyed kapcsolatban áll a másikkal, amelyben szintén szerepel az A és a B. Az automata ezt a hibát azonnal kiszűri.

A harmadik helyzetre már utaltunk. A tervező csak implicit ***szerepnevet*** alkalmaz. Nem mondja meg, hogy a SZEMÉLY RENDEZVÉNY FELADAT-ban a Kezdődátum a Dátum szerepneve. Így természetesen hiányozni fog ez a kapcsolat is.

Most elárulunk egy további titkot. Már a szótár első feltöltésénél feltűnhetett számunkra a Városnév redundanciája. Ez a VÁROS és a SZEMÉLY egyedben szerepel. Mivel a tétel mindkét egyedben ***leíró*** tulajdonság, azon kapcsolat sohasem létesülhet. Viszont a redundancia kapcsán elgondolkodhatunk azon, hogy nem kellene-e a személyeket a városokhoz kapcsolni? Ha igen, akkor arra nyilván a Városkód tétel alkalmas.

Az általunk feltételezett, de a modell által alá nem támasztott kapcsolatokat a megfelelő kapcsolótulajdonság alkalmazásával hozhatjuk rendbe. A számunkra feleslegeseket a kapcsolótétel kiemelésével illetve a homonimák átnevezésével szüntethetjük meg. Ezeken a műveleteken túl a kapcsolatelemzés még egy momentumot tartalmaz: ennek a lépésnek a során szűrjük ki az egyedek közötti esetleges ***ciklusokat***.

De térjünk át a gyakorlatra, mintapéldánk valós problémáira. A tervező a DÁTUM egyedet csak kétszer határozta meg fölérendeltként. Holott mindenki láthatja, hogy a Dátum adat más egyedtípusokban is kapcsoló szerepet tölt be. A kapcsolatok közül hiányzanak a DÁTUM következő alárendeltjei: MŰSOR, SZEMÉLY RENDEZVÉNY FELADAT, PROGRAM ESEMÉNY, BÍRÁLAT, SZEMÉLY ESEMÉNY. Mindezekben szerepel a Dátum tulajdonság, a kapcsolat definiálása mégis elmaradt. A VÁROS is mostoha sorsra jutott: nem definiálták a SZEMÉLY és a RENDEZVÉNY felé való kapcsolatát. A többi hiány a többlettel, a rosszul megadott viszonyokkal függ össze.

A SZEMÉLY RENDEZVÉNY FELADAT és a PROGRAM ESEMÉNY egyednek a BÍRÁLAT felé való kapcsolatát, a PROGRAM ESEMÉNY és a MŰSOR/PROGRAM viszonyát a tulajdonságok nem indokolják. Például a PROGRAM ESEMÉNY nem tartalmazza a Műsor sorszámát. E tévesen meghatározott viszonyok miatt a BÍRÁLAT kicsit a levegőben lóg: annak a RENDEZVÉNY-nyel, SZEMÉLY-lyel és PROGRAM-mal való viszonya hiányzik. Mivel a PROGRAM ESEMÉNY nem kapcsolódik közvetlenül a MŰSOR/PROGRAM egyedhez, az utóbbinak a PROGRAM-mal való viszonya lenne szükséges.

Persze a többszörös összetételű kulcsokon rosszul meghatározott kapcsolatokat és azt, hogy azokat egyszerűbbekkel kell kiváltani, már csak automata segítségével fedezhetjük fel. Meglepődnék, ha a példa megoldására vállalkozók közül valaki is el tudná helyesen rendezni e picinyke esettanulmány egyedeinek az összefüggéseit.

21.10 A példa időleges megoldása

Az alábbiakban közreadom a példa időleges megoldási változatát. Majd ezen ismertetés után adok magyarázatot az „időleges” jelzőre. A listában immár a régebben megszokott dőltbetűs szedettel mutatom a kapcsoló tulajdonságokat.

RENDEZVÉNY (***Rendezvény-becenév***, Rendezvénynév, *Városkód, Kezdődátum, Záródátum*)
  Dátum [Kezdődátum, Záródátum]
VÁROS (***Városkód***, Városnév)
DÁTUM (***Dátum***)
SZEMÉLY (***Személy-azonosító***, Személynév, Titulus, *Városkód*, Lakcím, Telefon)

RENDEZŐ EGYSÉG TÍPUS (***Rendező-egység-típuskód***, Rendező-egység-típusnév)
RENDEZŐ EGYSÉG (***Rendező-egység-cs***, Rendező-egység-megnevezés, Létrehozás-éve)
Rendező-egység-cs {*Rendező-egység-típuskód*, Rendező-egység-jel}
RENDEZŐ EGYSÉG FELADAT (***Rendező-egység-feladatkód***, Rendező-egység-feladat-név)
SZEMÉLY RENDEZŐ FELADAT (***Személy-rendező-feladat-cs***, *Városkód*)
Személy-rendező-feladat-cs {*Személy-azonosító+Rendező-egység-feladatkód+Rendező-egység-cs*}
RENDEZVÉNY TÁMOGATÓ (***Rendezvény-támogató-cs***, *Rendező-egység-cs*, *Városkód*, Egyéb-szervezetkód)
Rendezvény-támogató-cs {*Rendezvény-becenév*+Támogató-sorszám}
RENDEZVÉNY EGYSÉG TÍPUS (***Rendezvény-egységkód***, Rendezvény-egység-megnevezés)
RENDEZVÉNY EGYSÉG (***Rendezvény-egység-cs***)
Rendezvény-egység-cs {*Rendezvény-becenév+Rendezvény-egységkód*}
RENDEZVÉNY FELADAT (***Rendezvény-feladat-típuskód***, Rendezvény-feladat-megnevezés)
SZEMÉLY RENDEZVÉNY FELADAT (***Személy-rendezvény-feladat-cs***, *Rendezvény-egységkód*, *Dátum*)
Személy-rendezvény-feladat-cs {*Rendezvény-becenév+Rendezvény-feladat-típus-kód+ Személyi-azonosító*}
MŰSOR (***Műsor-cs***, *Dátum*, Kezdő-időpont, Záró-időpont, Műsornév)
Műsor-cs {*Rendezvény-becenév*+Műsor-sorszám}
PROGRAM (***Programszám***, Programcím)
MŰSOR/PROGRAM (***Műsor/program-cs***)
Műsor/program-cs {*Rendezvény-becenév*+Műsor-sorszám+*Programszám*}
KÖZREMŰKÖDŐ (***Közreműködő-cs***)
Közreműködő-cs {*Programszám+Személy-azonosító*}
PROGRAM ESEMÉNYTÍPUS (***Program-esemény-típuskód***, Program-esemény-típus-megnevezés)
PROGRAM ESEMÉNY (***Program-esemény-cs***, *Rendezvény-becenév*)
Program-esemény-cs {*Programszám+Program-esemény-típuskód+Dátum*}
BÍRÁLAT (***Bírálat-cs***)
Bírálat-cs {*Rendezvény-becenév+Programsszám+Dátum+Személy-azonosító*}
SZEMÉLY ESEMÉNYTÍPUS (***Személy-esemény-típuskód***, Személy-esemény-típus-megnevezés)
SZEMÉLY ESEMÉNY (***Személy-esemény-cs***)
Személy-esemény-cs {*Rendezvény-becenév+Személy-esemény-típuskód+Dátum+ Egyedi-azonosító*}
MŰSOR/SZEMÉLY (***Műsor/személy-cs***, *Programszám*)
Műsor/személy-cs {*Rendezvény-becenév+Rendezvény-feladat-típuskód+Személy-azonosító+ Műsor-sorszám*}

21.24 tábla

	FÖLÉRENDELT	**ALÁRENDELT**
1	VÁROS	RENDEZVÉNY TÁMOGATÓ
2	VÁROS	SZEMÉLY FELADATA
3	DÁTUM	RENDEZVÉNY
4	DÁTUM	RENDEZVÉNY
5	SZEMÉLY	SZEMÉLY FELADATA
6	SZEMÉLY	SZEMÉLY ESEMÉNYEI
7	SZEMÉLY	KÖZREMŰKÖDŐK
8	SZEMÉLY	SZEMÉLY RENDEZVÉNYI FELADATA

9	PROGRAM	KÖZREMŰKÖDŐK
10	PROGRAM	PROGRAM ESEMÉNY
11	RENDEZVÉNY	RENDEZVÉNY TÁMOGATÓ
12	RENDEZVÉNY	SZEMÉLY ESEMÉNYEI
13	RENDEZVÉNY	MŰSOR
14	RENDEZVÉNY	SZEMÉLY RENDEZVÉNYI FELADATA
15	RENDEZVÉNY	RENDEZVÉNY EGYSÉG
16	RENDEZVÉNY	PROGRAM ESEMÉNYEI
17	RENDEZŐ EGYSÉG TÍPUS	RENDEZŐ EGYSÉG
18	RI FELADAT	SZEMÉLY FELADATA
19	RENDEZŐ EGYSÉG	RENDEZVÉNY TÁMOGATÓ
21	RENDEZŐ EGYSÉG	SZEMÉLY FELADATA
21	MŰSOR	MŰSOR/SZEMÉLY
22	MŰSOR	MŰSOR/PROGRAM
23	PROGRAM ESEMÉNYTÍPUS	PROGRAM ESEMÉNYEI
24	RENDEZVÉNY FELADAT	SZEMÉLY RENDEZVÉNYI FELADATA
25	RENDEZVÉNY EGYSÉG	SZEMÉLY RENDEZVÉNYI FELADATA
26	SZEMÉLY ESEMÉNYTÍPUS	SZEMÉLY/ESEMÉNY
27	SZEMÉLY RENDEZVÉNYI FELADATA	MŰSOR/SZEMÉLY
*	SZEMÉLY RENDEZVÉNYI FELADATA	BÍRÁLAT
*	PROGRAM ESEMÉNY	BÍRÁLAT
30	DÁTUM	SZEMÉLY RENDEZVÉNY FELADAT
31	DÁTUM	MŰSOR
32	DÁTUM	PROGRAM ESEMÉNY
33	DÁTUM	BÍRÁLAT
34	DÁTUM	SZEMÉLY ESEMÉNY
35	VÁROS	RENDEZVÉNY
36	VÁROS	SZEMÉLY
37	PROGRAM	MŰSOR/PROGRAM
38	PROGRAM	BÍRÁLAT
39	RENDEZVÉNY	BÍRÁLAT
40	SZEMÉLY	BÍRÁLAT

Az „időlegesen végső” kapcsolattípusok listáját a 21.24 tábla mutatja. A csillaggal jelölt tételek nem-valósak, megszűntek. Az azok utániak a tulajdonságok által ténylegesen alátámasztott új kapcsolatokat mutatják. Az új adatmodellt nem rajzolom fel: azt már bárki megteheti a korábbi ábra alapján. Már csak annyit kell elárulnom, hogy miért „időleges” ez a megoldás is.

Icipici redundanciák még mindig maradtak a modellben. A SZEMÉLY egyed Telefon tételének első pár karaktere a településre utal. Nyilvánvaló fizikai redundanciát okoz. Az egyes feladatok kódjai a rendező illetve rendezvény egységek kódjainak a kiegészítései. Ez is redundancia és a módosításoknál bajokat eredményezhet. Talán elviselhetőeket. Csak arra akarom felhívni a figyelmet, hogy a kódba épített kód problémákat okozhat. Ha valakinek nem tetszik a Rendezőbizottság megjelölés és ahelyett Szervezőbizottság nevet akar - ez bizony gyakran megesik a mindennapi életben -, akkor a korábbi „RB” kódokat át kell írni minden feladatot jelölő „rokon” tulajdonságban is „SB”-re. Az ***egymásra épített „beszélőkódok”*** nem jól tűrik a változtatásokat...

A normálformák szempontjából pedig modellünk - rossz. Remélhetőleg valakinek feltűnt, hogy a MŰSOR/PROGRAM Műsor/program-cs {*Rendezvény-becenév*+Műsor-sorszám+*Programszám*} azonosítója ellentmondásos. Ugyanis a Rendezvény-becenév és a Programszám

együttese még mindig belső kulcstörő függést okoz. Miután ugyanaz a program egy rendezvény több műsorában nem fordulhat elő, a Rendezvény-becenév+Programszám meghatározza a Műsorsorszámot. Vagy élünk ezzel a helyzettel, ami esetünkben nem okoz karbantartási gondot, vagy újra kell gondolnunk a rendezvény, a műsor és a program viszonyát.

21.11 Zárszó

Lányom szerint egyszerűbb példák sorozatával kellett volna próbára tennem az olvasó türelmét, mielőtt egy ilyen úgymond komplikált esetet vezetek fel. Mi tagadás, részben igazat adtam neki, de végül is nem hallgattam rá. A valósághűség megsértésére, az egyértelműség hiányára (a homonimákra, szinonimákra, implicit szerepnevekre, rossz azonosításokra) mindig adódnak egyszerű példák. Azt is könnyen be lehet mutatni, hogy egy modell hiányos vagy éppen redundáns. Kis mintapéldákon mindez kiválóan működik. Az ember nagy sikert arat azzal, hogy rábök: az X egyed nincs az n-dik normálformában.

Az adatmodell optimumkritériumai egymással összefüggenek. Ha a modell redundáns vagy hiányos, akkor feltehetőleg nem is egyértelmű. Ameddig nem egyértelmű, addig nem tárhatók fel a hiányosságok és a redundanciák. Mindez nem mutatható ki a szakirodalomban megszokott minipéldákon.

Saját feladatommal még önmagamat is jól megdolgoztattam. Pedig engem egy meglehetősen pompás modellelemző szoftver támogatott. Így nem csodálnám, ha az olvasók jelentős része felhagyott volna a kísérletezéssel. A példáért mégsem kérek elnézést. Ha egy picurka kis rendezvényszervezési adatbázis megtervezése ennyi gondot okoz, akkor miképpen tudjuk elrendezni egy valós cég sokkal összetettebb, sokkal komolyabb ismereteit?

Remélhetőleg három tanulsággal mindenképpen szolgált ez az esettanulmány. Az egyik az, hogy egy már rosszul megtervezett „épületet” sokkal nehezebb átalakítani, mint eleve jó épületet tervezni. Egyes környezetekben a „majd integrálunk” jelszó jegyében fejlesztenek. Úgy vélem, hogy sikerült kimutatni az utólagos integrálás nehézségeit. ***Előre kell integrálni***, nem utólag. Az „adatbázis - egy” törvényére figyelve szabványokkal és az azok betartását ellenőrző eszközökkel és módszerekkel elkerülhető lenne az egyértelműségi, azonosítási, kódolási, csoportosítási káosz kialakulása.

A második következtetés is kézenfekvő. Ha az olvasó megbeszélhette volna velem, hogy valójában mi mit is jelent, akkor egyszerűbb lett volna a megoldás. Ha lett volna mód az értelmes kommunikációra, a modellezés nélkülözhetetlen kellékére...! Ha érthetőbben írtam volna le a példát...! Nos, a mindennapok során a tervezőnek lenne alkalma felkeresni a magyarázatokban illetékest, mégsem teszi elég intenzíven. Volna módja arra, hogy érthetőbben fogalmazza meg az adatbázistervet, mégsem él ezzel a lehetőséggel. Pedig az esettanulmány jól mutatja a ***szóbeli és írásbeli kommunikáció*** eléggé nem hangsúlyozható fontosságát.

A harmadik tanulság általánosabb. Az adatmodellezéshez nemcsak szakmai tudásra, nemcsak egy jó adag tapasztalatra, hanem bizonyos további képességekre - ha szabad így mondanom: erényekre - is szükség van. ***Tervezési erény*** a türelem, a részletek iránti fogékonyság, a rendérzék, a gondolati fegyelmezettség, az odafigyelés, az igényesség, a kísérletezési hajlam - és még sorolhatnám. A szakmai-technikai ismereteket le lehet írni. Azokat lehet oktatni, gyakoroltatni, magyarázgatni. A tapasztalatot nem lehet átadni: azt mindenkinek magának kell megszereznie fáradságos munkával. És az erények? Az adatmodellezésre ***nemcsak tanítani, hanem nevelni is kellene***.

Nemcsak magamnak rossz, ha tökéletlen az adatbázistervem, hanem másoknak is. Nemcsak nekem jó, ha elegáns - optimális - adatmodellt alkotok, hanem partnereimnek is az. Könyvünket

az ismeret mint szellemi kenyér fontosságának a megvilágításával kezdtük és a másokra való figyelés fontosságát hangsúlyoztuk. Fejezzük be ugyanitt. Én nem azért rágódok egy adatmodellen napokat-heteket, mert azért pénzt kapok. Jó, az is kell. Nem is azért, hogy megdicsérjenek: ez jól sikerült. Persze a sikerélmény is szükséges. Én azért törekszem egy jó terv összeállítására, mert - az a dolgom. Nem a munkám, hanem a feladatom. Ehhez értek, ez a hivatásom. Aminek lehetőleg maradéktalan betöltésére - neveltek.

Ebben a könyvben nemcsak tudásomat, hanem szemléletemet is szerettem volna átadni az olvasónak. Nemcsak oktatni, hanem nevelni is akartam volna. Azokat, akik erre még talán kaphatóak. Ezért szólt ez a kiadvány elsősorban a fiataloknak.

FELADATMEGOLDÁSOK

1/01 **A**. A közlés nem tartalmaz új ismeretet.

1/02 **A**. A leírt ismeret mindig „csak” adat. Ha valaki nem tudja azt, hogy mi a rendszám vagy a kocsitípus, abban nem születhet információ.

1/03 **S**. Tudja Ön, hogy ki az a Sári? Személy vagy ló?

1/04 Aki olvasni tud, az észleli és érzékeli a jelsort. Ha nem ismeri a szót, akkor itt meg is reked (3). Aki tud eszperantóul, az ismeri a magyar megfelelőt: „navigálás”, tehát a közlést felfogja (6). Aki tudja, hogy ez a szó mit jelent, csak az képes értelmezni is a közlést (10).

2/01 **2, T, 1, 3**

2/02 **E, V, V, E**. A férőhely és a hídhossz nem vezet sehová. Viszont a rendeléstétel cikkszám ismerete azt sejteti, hogy a cikkről is van mondanivalónk. Ha Gábor szervezői tanfolyamra jár, akkor a tanfolyam maga is fontos, Gábortól függetlenül leírható jelenség.

2/03 **I, H, H, I**

2/04 **H, I, I, I**

3/01 **H, T, I**

3/02 **1**. Csakis egyetlen egyedtípusról lehet szó, mivel az orvos, a beteg és a lehetséges beteg egyaránt SZEMÉLY. Az egyedaltípusokról majd később lesz szó. Több egyedtípus kreálása az adott esetben ismeretredundanciával járna. Hiszen az orvos is biztosított és ő is lehet beteg.

3/03 **M**. A pótlék a személy bérét írja le, tehát tulajdonságtípusként tükrözendő. Viszont több személy is ugyanazt a pótlékot kapja feladatától függően. Így a pótlék általánosan is leírható a Pótléknév, Pótlékösszeg, Pótlékfeltétel ismeretekkel. Ezért egyedtípusként is megfogalmazandó.

3/04 A Rendelésdátum nem vezet sehová. Ezzel szemben a Vevőkód egy másik egyedre, a konkrét vevőre utal a VEVŐ - RENDELÉS általános kapcsolat szerint. Ezért a két tulajdonság között eléggé lényeges az eltérés.

3/05 **H, H, I, H**

4/01 **T**. A duplapontosságú adat kimondottan tárolási „trükk”.

4/02 **L**.

4/03 **I, I, I, I**

4/04 **V, E, N, V, N**

5/01 **3**. Külön egyed a kocsi, a káresemény és a kocsit ért kár. Egy kocsi akárhány kárt szenvedhet, ezért a kár adata nem kapcsolható oda. Egy balesetben több kocsi vehet részt, így az előbbi állítás megfordítva is igaz. Vegyük észre, hogy más jelenség a kocsi, a valahol történt baleset és a kocsit ért kár.

5/02 **R, R, H**

5/03 **2**

5/04 **I, I, I**

5/05 **H, H, H, I**

6/01 **H, H, I**. Ha nem így gondolkodnánk, Cliff Richard dalait összekevernénk Little Richard nótáival.

6/02 **L, S, H**

6/03 **AA, LL, AK, AA, LL**

6/04 **3**

6/05 **1**. Ismétlődő csoportot nem szabad alkalmazni. A „legmagasabb" képzettség pedig azért csacsiság, mert valakinek lehet több ilyen képzettsége is.

6/06 **3**

6/07 Számlaszám: AA. Vevőkód: AK. Sorszám: RR.

6/08 **4**. Sokan a 3. megoldást választják. Nem jutnak vele messzire, mert kiderül, hogy téves az egész ismeretsor. Minden jelenségnek ténylegesen azonosíthatónak kell lennie. Ameddig nem az, addig nem lehet vele számolni.

7/01 **N, H, N**

7/02 **H, I, I, I**

7/03 **I, H, H**

7/04 **R, J, J**

7/05 **R, R, J**

7/06 **F, P**

7/07 **2, 4**

7/08 **3, 4**

8/01 **I, I, I**

8/02 **K, K, F, K**

8/03 **I, H, I, I, H, I**. A Számlaérték a Számlatétel-értékből egy hozzáféréssel kiszámítható, mert a SZÁMLA a SZÁMLATÉTEL közvetlen fölérendeltje.

8/04 **3**

8/05 A könyvelés minden ismerete másutt már kezelt alapadatokra épül. Ha azokat nem tároljuk a számítógépen, akkor manuálisan kell bevinni őket. Ez pedig igen komoly hibák forrása. Nem beszélve a csalási szándékról, a számítógépes könyvelés sokszor azért nem sikeres, mert frissen bevitt - és nem ab ovo tárolt - adatokra épül.

8/06 **2, 3, 1, 0, 4**

12/01 **1** - A Vevőcím nyílt logikai átfedést okoz.

12/02 **3, 5** - A Vevőnév és Vevő-megnevezés szinonima. A két egyed nem kapcsolható.

12/03 **7** - A részlet egyensúlytalan. Könnyen kikereshető az, hogy egy személy milyen nyelveket beszél. Viszont nehezen keressük ki, hogy ki tud angolul.

12/04 **K** - Mivel bármikor szükség lehet újabb keltezésre (pl. szállítás dátuma), az általános Dátum név alkalmazása nem célszerű. A leírókat minősíteni illik. Ezért a Rendelésdátum név használata helyesebb lenne.

12/05 **K** - A terv technikai szinonimát tartalmaz. Gondolkozni kell azon, hogy a két adószám azonos lényeg-e. Az azonosítókat kapcsoló szerepük miatt nem illik minősíteni. Itt helyesebb lenne az egyszerű Adószám név használata.

12/06 **B** - Tipikus bizonylatszemlélet. A személy törzslapján szerepel a foglalkoztató szervezeti egység kódja és neve. Ezért a tervező mindkettőt felvette a SZEMÉLY állományba, holott külön SZERVEZET állományban lenne csak szabad a nevet rögzíteni. Ha a Szervezetnév változik, akkor azt így minden foglalkoztatottjánál karban kell tartani, tehát a feldolgozás többszörös.

12/07 **K** - Tipikus kimenetorientált terv. Nem magára a lényegre (cikk) utal annak azonosítójával (Cikkszám), hanem osztályozási ismérveket sorol fel.

12/08 Első ránézésre az Ár nyílt logikai átfedés, redundancia. Azonban könnyen meglehet, hogy homonimáról van szó. Létezik a cikkre jellemző Egységár és egyes rendelésekben kialkudott, kedvezményes vagy szerződéses árakat alkalmaznak. Ha egy cikk Szerződéses-ára mindig azonos, de nem egyezik meg az Egységárral, akkor a CIKK-ben kell szerepeltetni a két árat és a RENDELÉSTÉTEL-ben nincs ár. Ha a kedvezményes árak alkalomszerűek, akkor az Árnak a RENDELÉSTÉTEL-ben is van helye, de mindenképpen más - minősített - névvel.

13/01 **1,4**

13/02 **E** - Ez becsapás volt. A TELEPÜLÉS egyeden belül minden településnek van neve és irányítószáma.

13/03 **A** - Gyerekkód → Apakód
B - Gyerekkód <–/–> Apakód

13/04 Raktárkód <–/–> Cikk-kód
Raktárkód ← Cikk-kód
Rendelésszám → Cikk-kód
Rendelésszám <–/–> Cikk-kód
Irányítószám → Kerületkód
Férjkód ↔ Feleségkód
Férjkód <–/–> Feleségkód

13/05 **J** - Minden településnek van irányítószáma, de nem minden ilyen szám azonosít települést.

13/06 **3**

13/07 **3** - Két tulajdonság viszonya nem lehet ilyen is, meg olyan is. Ha millió rendelés között csak egy is van, amelyben több cikket kérnek, akkor már a RENDELÉS és a CIKK egyed M:N-es viszonyú, tehát kulcsaik egymástól függetlenek.

13/08 **S** - A terv rejtett ismétlődést tartalmaz, amit sohasem szabad álnevekkel leplezni.

13/09 SZEMÉLY (***Törzsszám***, ...)
SZÁMLA (***Törzsszám+Hónap***, ..., Számla-összeg)

14/01 **0** - Az egyed ismétlődést tartalmaz.

14/02 **1**

14/03 **1**

14/04 CIKK (***Cikkszám***, Egységár)
SZÁMLATÉTEL (***Számlaszám+Cikkszám***, ..., Tételérték)

14/05 **2**

14/06 **3** - Minden kocsinak van kötelező biztosítása. Ezért a Rendszám és a KB-kötvényszám függés kölcsönös. Az utóbbi tétel valódi „kulcs”, amelynek függéseit nem vizsgáljuk a 3NF alakig bezáróan. Ellenesetként szolgál a 14.11 példa Casco-kötvényszám tulajdonsága, amely nem volt valódi kulcsváltozat.

15/01 **I, I, I, H** - Elméletileg egy kétoszlopos csupakulcs táblázat lebontható két unáris táblára. Gyakorlatilag ennek nincs jelentősége, ezért nem is részletezzük a témát.

15/02 **H** - Egy egyed nemcsak a funkcionális, hanem a többértékű függés esetén is megbontható összekapcsolható módon. Tehát a kitételt így kell módosítani: az E (A, B, C) egyed csak akkor egyenlő az E1 (A, B) és E2 (A, C) lebontásainak az összekapcsolásával, ha fennáll az A ==> B többértékű függés.

15/03 **I, I, I, H**

15/04 **I, H, I**

15/05 Az első egyedben körkörös JD fedezhető fel, tehát az egyed csak 4NF alakú és hármas megbontást lehetne alkalmazni. Viszont a második egyedben az első három tétel funkcionálisan meghatározza a negyediket (melyik csapat, melyik asztalnál, milyen leosztásban milyen eredményt ért el). Ezért ez az egyed nem bontható meg ismeretveszteség nélkül. A példa jól mutatja, hogy az 5NF csak elméleti trükk. A gyakorlatban igen ritka a megbontható, csupakulcs, leíró tulajdonság nélküli egyedtípus.

16/01 **I, I, I, I, I**

16/02 A TERMÉK egyedben tetszőleges - létező - dolgozóra és gépre lehet utalni úgy, hogy a dolgozó nem is kezeli azt a gépet. Hiányzik a két egyed közti hivatkozási integritási korlát. A helyes megoldás:

HASZNÁLJA (***Használja-azonosító***, ..., Műszak)
TERMÉK (***Termékazonosító***, *Használja-azonosító*)
Használja-azonosító {Dolgozó-azonosító+Gépazonosító}

16/03 A Dolgozó-azonosító impliciten redundáns. Azt a Helyettesített-dolgozó-azonosító tétellel kell kiváltani a homonima elkerülésére. Az egyik azonosító Szabóra, a másik Kovácsra fog utalni.

16/04 A Rendelésszám → Számlaszám függés miatt pszeudo-tranzitivitás áll fenn. Az Érték vagy pszeudo-tranzitív függésű, vagy homonima. Feltehetőleg az előbbi.

16/05 A probléma a következő: a Szerződésszámot megadhatjuk elemi tulajdonságként, de akkor nem tudunk a Vevőkódon a VEVŐ felé kapcsolni és nem érvényesíthetjük a hivatkozási integritást. Vagy részeire bontjuk, de akkor procedurálisan kell ellenőrizni a Kötésszám Szerződésszámon belüli egyediségét. Nem deklarálhatjuk egyetlen helyen az egyediséget. Természetesen a célszerű megoldás a csoport lenne. A fogalmi tervezési szinten feltétlenül az, hogy mindegyik integritási korlátot ki tudjuk fejezni.

IRODALOMJEGYZÉK

[1] IBM: Vocabulary for Data Processing, Telecommunications and Office Systems. 7. kiadás, 1981. július.

[2] Idegen szavak kéziszótára. Szerk.: Bakos Ferenc. Terra, Budapest, 1967.

[3] Bachman, C.W.: Data Space Mapped into Three Dimensions. Infotech State of the Art Report 15., 1973.

[4] Falkenberg, E.: Significations: The Key to Unify Data Base Management. Information Systems, Vol. 2, No. 1, 1976.

[5] Nijssen, G.M.: Architecture and Models in Data Base Management. North-Holland, Amsterdam, 1977.

[6] Hall P. et al.: Relations and Entities. In: Modelling in Data Base Management Systems (ed.: Nijssen, G.M.), North-Holland, Amsterdam, 1976.

[7] Quittner Pál: Adatbázis-kezelés a gyakorlatban. Akadémia, Budapest, 1993.

[8] Proc. of IFIP TC2 Working Conference on Data Base Architecture. Velence, 1979. North-Holland, Amsterdam, 1979.

[9] Object-Oriented Databases. Proc. of 1st Internat. Workshop, IEEE Computer Society Press, 1986.

[10] ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report. FDT (ACM SIGMOD bulletin) 7, No. 2, 1975.

[11] ISO TC97/SC5/WG5: A Database Management System REFERENCE MODEL for STANDARDISATION. 1984 szeptember.

[12] Chen, P.P.S.: The Entity-Relationship Model - Toward a Unified View of Data. ACM TODS Vol. 1, No. 1, 1976.

[13] Rustin, R. (ed.): Data Models: Data Structure Set versus Relational. Proc. 1974 ACM SIGMOD Workshop on Data Description, Access and Control, Vol. 2, 1974.

[14] Bachman, C.W.: Data Structure Diagrams. Data Base (ACM SIGBDP) Vol. 1, No. 2, 1969.

[15] Griethuysen (ed.) ISO TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and Information Base, ANSI, 1982.

[16] Codd, E.F.: Extending the Database Relational Model to Capture More Meaning. ACM TODS, Vol. 4, No. 4, 1979.

[17] Appleton, D.S.: SYSTEM 2000 DBMS. Proc. of GUIDE 37. 1975.

[18] Malcolm, E.: SSADM Version 4. A User's Guide. McGraw-Hill, London, 1992.

[19] Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. CACM, Vol. 13, No. 6, 1970.

[20] Armstrong, W.W.: Dependency Structures of Data Base Relationships. Proc. IFIP Congress 1974.

[21] Codd, E.F.: Further Normalization of the Data Base Relational Model. Data Base Systems, Courant Computer Science Symposia Series, Vol. 6, Englewood Cliffs, N.J., Prentice-Hall, 1972.

[22] Date, C.J.: An Introduction to Database Systems. 3.kiadás, Addison-Wesley, Reading, Massachusetts, 1981.

[23] Rissanen, J.: Independent Components of Relations. ACM TODS, Vol. 2, No. 4, 1977.

[24] Codd, E.F.: Recent Investigations into Relational Data Base Systems. Proc. IFIP Congress, 1974.

[25] Fagin, R.: Multivalued Dependencies and a New Normal Form for Relational Databases. ACM TODS, Vol. 2, No. 3, 1977.

[26] Aho, A.V. és mások: The Theory of Joins in Relational Databases. ACM TODS, Vol. 4, No. 3, 1979.
[27] Smith, J.M.: A Normal Form for Abstract Syntax. Proc. 4th Intertnational Conf. on Very Large Data Bases, 1978.
[28] Fagin, R.: A Normal Form for Relational Databases That is Based on Domains and Keys. IBM Research Report RJ2520, 1980.
[29] Fagin, R.: The Decomposition Versus the Synthetic Approach to Relational Database Design. Proc. 3rd International Conference on Very Large Data Bases, 1977.
[30] Bernstein, P.A.: Synthesizing Third Normal Form Relations from Functional Dependencies. ACM TODS, Vol. 1, No. 4, 1976.
[31] Halassy, B.: Normal forms and normalization: practical designer's view. Information and Software Technology, Vol. 33, No. 6, 1991.

FOGALOMJEGYZÉK

1NF 196
2NF 202
3NF 206
4NF 225
5NF 229
Ábraorientált tervezés 129
Abszolút szerep 75
Adat 14, 28
Adatábrázolás 53
Adatadminisztrátor 140
Adatállomány 27
Adatalmodell 66
Adatbank 29
Adatbázis 43
Adatbázisgép 110
Adatbáziskezelés 114
Adatbevitel 149
Adatcsoport 48
Adatdimenziók 23
Adatfeldolgozási művelet 111
Adatkezelési művelet 110
Adatkimenet 151
Adatkonverzió 138, 148
Adatmodell 44
Adatmodell-diagram 68
Adatrögzítés 149
Adatszótár 124
Adattétel 28
Additivitási szabály 177
Alapadat 111
Alárendelt egyed 77
Alkalmazási adatbázis 121
Állománykezelés 113
Állománykezelő 32
Alternáló kulcs 212
Asszociáció 37
Atomi egyedtípus 218
Attribútum 180
Azonosítás 39
Azonosító 40, 74
Azonosító szerep 73
BCNF 218
Belső kulcstörő függés 236
Belső szint 64
Bemenetorientáltság 172
Bináris reláció 37
Birtoklási viszony 77
Bizonylat 148
Családfa egyedtípus 96, 275
Csoport 232
Csoportfüggés 241
Csupakulcs egyed 84, 223
Deklaratív adatkezelés 90
Denormalizálás 207
Deszkriptor 29
Doméjn 181
Egyed 33
Egyed belső szerkezete 72
Egyed külső szerkezete 72
Egyed-előfordulás 33
Egyed-előforduláshalmaz 33
Egyedaltípus 101, 277
Egyedciklus 259
Egyedtípus 33
Elemi azonosító 83
Elemi függés 177
Elhelyezési mód 54
Elsődleges kulcs 74
Erős függés 179
Értékhalmaz 35
Értéktartomány 90, 180
Érvényesítés 52
Eszközfüggőség 46
Fejlesztési adatbázis 121
Feltételes függés 276
Fizikai adatfüggetlenség 55
Fizikai adatszerkezet 53
Fizikai átfedés 166
Fizikai particionálás 109
Fogalmi adatszerkezet 51
Fogalmi séma 66
Fogalmi szint 64
Fogalomalkotás 144
Fonál 263
Fölérendelt egyed 77
Funkcionális függés 177
Függéserő 179
Generalizáció 102, 278
Globális nézet 61
Gyenge függés 179
Hálós viszony 79, 256
Házastárs viszony 98
Hierarchikus viszony 78, 256
Hivatkozás-integritás 249
Homonima 162
Hozzáférési mód 54

Idegen kulcs 75
Információ 14
Információs erőforrás szótár 125
Inkonnektivitás 165
Integritási korlát 180
Interjú 145
Is-egy kapcsolat 102
Ismétlődő adat 48, 183
Ismétlődő csoport 183
Ismétlődő tulajdonság 79
Kapcsolásfüggés 228
Kapcsolat 76, 104
Kapcsolat foka 77
Kapcsolat opcionalitása 77
Kapcsolat-előfordulás 41
Kapcsolat-előfordulás-alhalmaz 41
Kapcsolat-előforduláshalmaz 41
Kapcsolathiány 248
Kapcsolattípus 41
Kapcsoló szerep 75
Kiegyensúlyozatlanság 167
Kimenetorientáltság 173
Kiterjedés (extenzió) 108
Kivetítés 199
Konstans 283
Korlát 63
Kölcsönös függés 178
Kölcsönös viszony 84
Könyvtár 29
Kulcsjelölt 211
Kulcsmátrix 266
Kulcsrész 84
Kulcstörő függés 217
Külső szint 65
Látszólagos logikai átfedés 162
Lebonthatatlan reláció 38
Leíró szerep 74
Lineáris viszony 257
Logikai adatfüggetlenség 65
Logikai adatszerkezet 52
Logikai particionálás 108
Metaadat 122
Metaadatbázis 123
Metaadatmodell 123
Metaismeret 127
Metaszabvány 136
Metszetfüggés 238
Mező 27
Mondat 28
Mondattípus 25, 38
Navigálás 114
Nem-értelmezhető tulajdonság 100
Nem-független lebontás 211
Nem-normalizált egyedtípus 184
Nézet 65
Nézetorientáltság 58
Normálforma 184, 195
Normálforma dekompozíció 199
Normálforma szintézis 252
Normalizálási alap 247
Normalizálási eljárás 247
Nyílt logikai átfedés 161
Objektumtípus 37
Öröklődés 102
Összekapcsolás 210
Összetett azonosító 83
Összetett függés 178
Parciális nézet 62
Pragmatika 19
Procedurális adatkezelés 89
Projektivitási szabály 197
Pszeudo-tranzitív függés 244
Reflexivitási szabály 177
Rejtett logikai átfedés 163
Rekord 27
Relatív szerep 75
Részleges függés 180, 197
Specializáció 101, 277
Származtatott adat 111
Száz-százalékos elv 90
Szemantika 19
Szemantikai adatbázis 24
Szerepnév 91
Szinguláris egyed 281
Szinonima 163
Szintaktika 19
Szó 28
Szövegállomány 32
Szövegkezelő 32
Szövegszerkesztő 23
Tartalom (intenzió) 108
Tartományfüggés 181
Technikai adatok 117
Technikai homonima 164
Technikai szinonima 164
Teljes függés 180
Tervezési folyamat 137
Tervtermék 136
Tezaurusz 29
Többértékű függés 224

Többszörös kapcsolat 270
Tranzitív függés 204
Tranzitivitási szabály 204
Triviális függés 234
Tulajdonság 35
Tulajdonság szerep 74
Tulajdonság-struktúra 282
Tulajdonság-vándorlás 259
Tulajdonságérték 35
Tulajdonságtípus 35
Unáris egyed 279
Univerzális reláció 251
Validálás 149
Változásmenedzselés 139
Vertikális leképezés 56
Veszteségmentes dekompozíció 209
Virtuális egyed 65
Visszamutató kapcsolat 95, 274
Visszamutató viszony 95, 273