

Magiszter

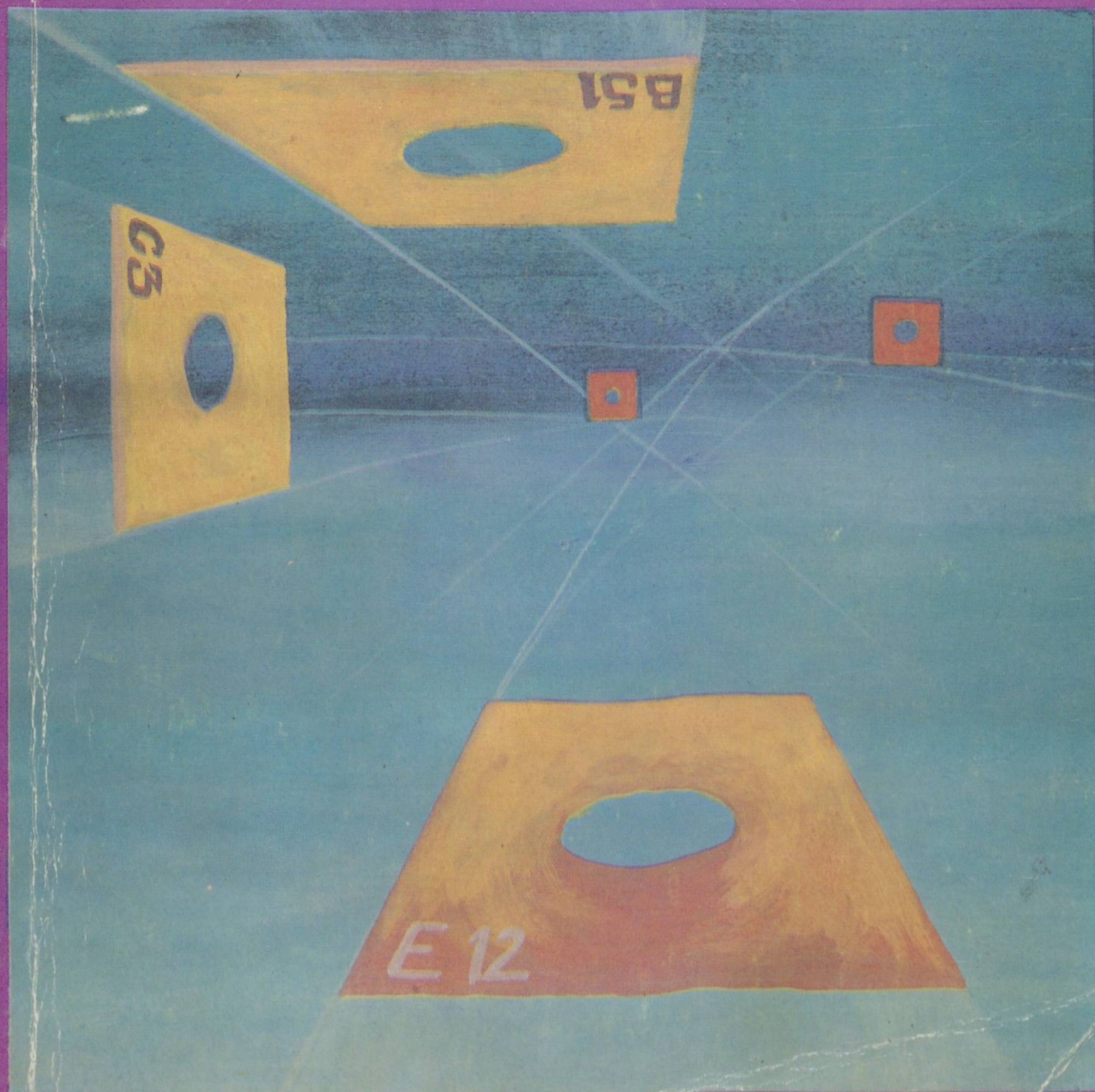
Számítástechnikai szerkesztőség

ADATBÁZISELEMEK OSZTÁLYOZÁSA

/algoritmusok és programjaik/

Programvédelem nélkül!

Szoftver
IBM PC, XT, AT
és
kompatibilis mikroszámítógépekre



I. rész

MEZEY — LÉVAI

Adatbáziselemek osztályozása

szoftver melléklettel

AKADÉMIAI KIADÓ ÉS NYOMDA VÁLLALAT

Felelős kiadó: Hazai György főigazgató

Felelős szerkesztő: Oláh Judit

A borítót Szabadi Zoltán tervezte

ISBN 963 05 48 704

ISSN 0238-3896

A kéziratot a TYPOTEX GT gondozta

A kézirat lezárva: 1988. július

Készült az OPTIMUM Képző- és Nyomdaipari Vállalat közreműködésével

a VEGYTERV Reprotechnikai üzemében

Felelős vezető: Balogh Mihály

VTV. 88.185

37205

MEZEY GYULA — LÉVAI GÁBOR

ADATBÁZISELEMEK OSZTÁLYOZÁSA

automatikus osztályozás alkalmazása adatbázisok tervezésében

szoftver melléklettel

MS/PC DOS 2.xx - től

MTAK



0 00003 32621 3



Akadémiai Kiadó, Budapest 1988

665578

Szoftver - dokumentáció
Sorozatszerkesztő: Pintér Tibor
Lektorálta: Kotsis Domokos

©Akadémiai Kiadó és Nyomda Vállalat
Mezey Gyula – Lévai Gábor, 1988
Minden jog fenntartva

MAGYAR
TUDOMÁNYOS AKADÉMIA
KÖNYVTÁRA

A szoftver - dokumentáció a MATEX programcsomaggal készült
©Akadémiai Kiadó és Nyomda Vállalat · Fadgyas – Miklós

M. TUD. AKADÉMIA KÖNYVTÁRA
Könyvtár 16332 / 1988. sz.

Tartalomjegyzék

1. Információs rendszer fejlesztését előkészítő teendők	9
1.1. A bizonylati album	9
1.2. A rendszerelemzés és -tervezés előkészítése	10
1.2.1. A módszer kiválasztása	11
1.2.2. A teendők részleges leírása	11
1.2.3. A rendszerelemzéshez és -tervezéshez szükséges interjúk megtervezése	12
1.2.4. A felvett adatok hitelesítése, ellenőrzése	13
2. Rendszertervezési módszertanok	15
2.1. A tervezési, modellezési módszerekről	15
2.1.1. Output-orientált tervezés	15
2.1.2. Adatorientált (azaz adatbázis) tervezés	16
2.1.3. Stukturált rendszertervezés	16
2.2. Gyors átvilágítás és az alrendszerek képzése	17
2.2.1. A CSF (Critical Success Factors) átvilágítási módszer	18
2.2.2. A SADT (Structured Analysis and Design Technique) rendszerelemzési módszere	19
2.2.3. Az ISAC (Information Systems Work and Analysis of Changes) helyzetfelmérési és rendszerelemzési módszertana	21
2.2.4. A BSP (Business Systems Planning) átvilágítási módszer	23
3. Adatmodellezés	27
3.1. Az adatmodellezés fogalma, sajátosságai	27
3.2. Az analitikus és szintetikus adatmodellezési eljárás	31
3.3. Az elvi adatmodell kialakítását megelőző jellemzés és helyzetfelmérés	34

4. Funkcionális elemzés	37
4.1. A funkcionális elemzést megelőző helyzetfelmérés	39
4.1.1. Aszinkron (reakció) jellegű tevékenységek felmérése	40
4.1.2. Szinkron jellegű tevékenységek felmérése	41
5. Logikai adatmodellezés	45
5.1. Adattétel (elemi adat, mező)	45
5.2. Szegmens (csoport, group, részrekord) adatstruktúra	46
5.3. Szegmensreláció (group relation, chain, set, adatgyűjtés)	47
5.4. Adatmondat (logikai rekord, entry, mondat)	48
5.5. Adatmondat-reláció	49
5.6. File (adatállomány)	49
6. Adatbázis-hozzáférés	51
6.1. Hozzáférési modell	51
6.2. Adatbázis-kezelés	55
6.2.1. Adatbázis-kezelő rendszerek	55
6.2.2. A DDL és a DML	56
6.2.3. Az alséma definíciói	57
7. Tárolási szerkezetek	61
7.1. Fastruktúra leképzése tárolóra	61
7.2. Hálóstruktúra leképzése, tárolása	63
7.3. Struktúraállományok	65
8. Elérési (elhelyezési) módok, file-kezelés	67
8.1. Gyors tulajdonosság-sorrendű szerkezetek	69
8.2. Gyors közvetlen elérést biztosító elérési módok	70
8.3. CODASYL — jelentés által tárgyalt elérési, elhelyezési módok	73
9. Az adatbázis változásai	77
9.1. Az adatbázisfelelős	77
9.2. Az adatbázis tartalmi épségének védelme és helyreállítása	79
9.2.1. Védelem	79
9.2.2. Helyreállítás	80
9.2.3. Az adatok időegyezősége	82
II. rész	85

1. Osztályozással segített tervezési szakaszok kiemelése	87
1.1. Bevezetés	87
1.2. A séma megtervezése	89
1.3. Teendők a hozzáférési igények meghatározásának szakaszában	92
1.4. A logikai adatbázis-tervezés menetén belül áttekintett tevékenységek	95
1.4.1. Egyed típusok leképezése szegmenstípusokba	96
1.4.2. Szegmenstípusok összevonása	98
1.5. A fizikai adatbázis-tervezés menetén belül áttekintett tevékenységek	100
1.5.1. Elhelyezési mód meghatározása	101
1.5.2. Az adatbázis felosztása area-kra	102
2. Az alkalmazott módszerek köre	105
2.1. Az osztályozás módszerei	105
2.2. Nem hierarchikus klaszteráló módszerek jellemzése	107
2.3. Hierarchikus klaszteráló módszerek csoportosítása	108
2.4. A taxonomikus mértékek alaptípusai	110
2.4.1. Mennyiségi ismérvek (tulajdonságok)	110
2.4.2. Minőségi (kategorizált, illetve rendezési) ismérvek (tulajdonságok)	111
2.4.3. Vegyesen előforduló minőségi és mennyiségi ismérvek	112
3. Egyed típusok leképezése szegmenstípusokká	113
3.1. Elvi megfontolások	113
3.1.1. Alkalmas klaszterálási módszer kiválasztása	113
3.1.2. Szegmensképző algoritmus ismertetése	114
3.1.3. Alkalmas taxonomikus mérték	117
3.2. A hatékonyság és a gyakorlati megvalósítás szempontjai	118
3.2.1. Hasonlósági együttható alkalmazása	119
3.3. Távolságmérték alkalmazása	124
4. Szegmenstípusok összevonása nagyobb egységekbe	125
4.1. Az „A” tervezési stratégia teendői	125
4.1.1. Szegmenstípusok osztályozása	125
4.1.2. Szegmenselési fák meghatározása	132

4.1.3. Mértékadó szegmensszintű adatmodell kiválasztása szegmens- osztályból	133
4.1.4. Mértékadó szegmensszintű adatalmodell összevonása erdővé ...	134
4.2. „B” stratégia lényeges elemei	134
5. Az osztályozást megvalósító program	139
5.1. A program segítségével elemezhető adatbázis-tulajdonságok	140
5.2. A program tevékenységei	140
5.3. A program használata	141
5.4. Az input adatállomány szerkezete	144
5.5. Elnevezések kezelése	148
5.6. Hibakezelés	149
5.7. Az output listák tartalma	156
1. melléklet	159
2. melléklet	161
Táblázatok, ábrák	167
Irodalomjegyzék	247

1. fejezet

Információs rendszer fejlesztését előkészítő teendők

Az információs rendszer elemzése, a majd megtervezéséhez szükséges adatgyűjtés (helyzetfelmérés és dokumentálás) a fejlesztést jóváhagyó döntés után kezdődhet el. A helyzetfelmérés előkészítésének kezdeti lépéseiként a rendszerszervezési módszert kell pontosan rögzíteni, majd a fejlesztési program hatékony mechanizmusának, belső szervezeti rendjének kialakítását kell egyezteteni, végül nagy vonalakban meg kell ismerni a felhasználó szervezet működésének jellegét és legfőbb problémáit. A tapasztalat szerint a rendszerelemzéshez és az új rendszer megtervezéséhez szükséges adatgyűjtés eredményeinek dokumentálása négy, egymástól jól elkülöníthető szakaszra tagolható:

- a szervezeti működés dokumentumainak összegyűjtése és áttekintése (ez a legszélesebb értelemben vett bizonylati album elkészítését jelenti),
- a további adatgyűjtési és dokumentálási folyamatra való felkészülés, ennek a folyamatnak a részletes megtervezése, az ahhoz szükséges segédleteknek, kérdőíveknek és a dokumentálás rendjének megtervezése,
- a különböző területek adatorientált felméréséhez szükséges interjúk elkészítése, az interjúk eredményeinek megfelelő dokumentálása,
- az így összegyűjtött adatok teljességének, valódi voltának és pontosságának az ellenőrzése.

A külföldi szakirodalmi adatok, de a hazai tapasztalatok is arra utalnak, hogy a helyzetfelismerés során végzett adatgyűjtés és annak dokumentálása viszonylag nagy idő- és energiaráfordítást igénylő munka. Példaképp említhetjük, hogy az USA-beli Wichita Falls-ban a városi információs rendszer felmérése, a teljes ráfordítást tekintve, hozzávetőlegesen kb. 15 ezer ember-óra volt, és ebből kb. 10 ezer ember-óra a tényleges felmérésre, a maradék 5 ezer ember-óra pedig a felméréshez való felkészülésre és a felmérés utáni dokumentálásra fordítódott. Általános tapasztalat, hogy e maradék időt mindenütt messze alábecsülték. Alábecsülték csaknem minden esetben

az interjúkészítésre fordítandó időt is, holott a teljes felmérésre szánt időnek ez csupán átlagosan kétharmadát jelentette.

1.1. A bizonylati album

A bizonylati album készítése a szervezők számára sok szempontból rendkívül hasznos.

Először is, segítségével nagyrészt azonosíthatók (megnevezhetők) azok a tevékenységek, amelyeket jelenleg a szervezeti egységek végeznek.

Lehetőséget ad arra, hogy a dokumentumokon használt fogalmakkal összeállítsunk egy olyan hierarchikus (hálós) szerkezetet, amely átfogja a szervezet egészében végzett feladatokat, tevékenységeket. Ez egyben módot ad az egyes interjúkat végző csoportok és a szervezeti egységek megbeszéléseinek megszervezésére.

Lehetőséget ad továbbá arra, hogy az interjúvégzés folyamatát ütemezni és strukturálni lehessen. Így a felmérés során nyert adatokat, a dokumentációt, el lehet rendezni valamiféle szerkezetben.

Lehetőséget nyújt ahhoz, hogy az ezután következő interjú szakaszának munkaterhelését hozzávetőleg meg tudjuk becsülni.

Lehetőséget ad arra, hogy a szervezetben használt fogalmaknak a dokumentumról történő leolvasása, rendezése útján megalapozzuk az adatszótár feltöltését. Ennek a későbbiekben jelentős szerepe lesz.

Összességében, ezeknek a dokumentumoknak az összegyűjtése és elemzése, illetve összegyűjtésükkel párhuzamosan a szervezeti egységek dolgozóival való kapcsolatteremtés lehetőséget ad arra, hogy azok a szervezők, akik más területről jöttek és még nem ismerték eddig a szóban forgó szervezet működését, képet nyerjenek a szervezet tevékenységéről, működéséről.

1.2. A rendszerelemzés és -tervezés előkészítése

Az adatgyűjtésre és a helyzetfelmérésre, dokumentálásra való felkészülés igen jelentős abból a szempontból, hogy a későbbiekben a feladat jó hatásokkal, gyorsan legyen elvégezhető. Három jelentős tényező van amit ebben a szakaszban meg kell oldani:

- ki kell választani, esetleg ki kell fejleszteni azt a módszert, amellyel a helyzetfelmérés során összegyűjtött adatokat majd fel kell dolgozni,
- ebből kiindulva el kell készíteni a teendők részletes leírását az adatgyűjtést végző és a megkapott adatokat dokumentáló szervezők részére,
- az interjúkészítés és a helyzetfelmérés meghatározott területre irányítása érdekében a folyamatok pontos azonosítására ki kell dolgozni egy szakfeladatok és szaktevékenységek szerint tagolt listát.

1.2.1. A módszer kiválasztása

Az első feladat tehát: kiválasztani a megfelelő tervezési és rendszerelemzési eljárást. Ennek azért nagy a jelentősége, mert eleve meghatározza, hogy a helyzetfelmérés során mely adatok azok, amelyeket feltétlenül gyűjteni kell, és melyek azok, amelyekre viszont — legalábbis ebben a szakaszban — még nem vagyunk kíváncsiak. A szóba jöhető elemzési és tervezési módszerekről a következőkben szólnunk majd.

1.2.2. A teendők részleges leírása

Tekintettel arra, hogy az adatgyűjtést végző személyek nem feltétlenül gyakorlott rendszerszervezők vagy rendszerelemzők, a második fontos feladat, hogy előkészítsük számukra az írott munkaprogramot és kérdőíveket. Attól függően, hogy kiket foglalkoztatunk, más és más részletességgel kidolgozott helyzetfelmérési munkautasításokat és a teendők részletes vagy kevésbé részletes leírását kell előre megadnunk. Nyilvánvaló, hogy egy gyakorlott rendszerszervező számára nem szükséges az interjú levezetésének részletes lépésekre bontott leírása, ugyanakkor olyan valakinek, aki ilyet nem csinált, szükséges és hasznos is, ha ezt valami módon vezérfonalként rögzítik.

Eldöntendő kérdés, hogy a felmérést szervezeti egységek szerint vagy folyamatok szerint haladva akarjuk-e, tudjuk-e végezni, az összegyűjtött dokumentáció rendszerezésére, tárolására. Meg kell jegyezni, hogy bár az egyes folyamatok követéséhez, a folyamat tevékenységelemein való végighaladás szükséges, de egyszerűbb és munkaszervezési szempontból hatékonyabbnak látszik a felmérést egy-egy osztályon úgy végeznünk el, hogy az lehetőleg az osztály munkáját legkevésbé zavarja meg. Márpedig, hogyha a folyamatokat követjük és egy osztályon 20-50 folyamat egy-egy részletét is megta-

láthatjuk ez azt jelenti, hogy ezt az osztályt annyi interjút készítő szervező fogja majd felkeresni, ahány követi ezeket a folyamatokat. Ha egy szakigazgatási szervhez egy szervezőt vagy interjút készítő csoportot jelölünk ki állandó kapcsolattartásra, akkor ez csökkenti a dolgozóknak feltett kérdés-mennyiséget, nem fognak többször felmérni azonos adatokat, nem fognak ismételten kérdezni ugyanazokról a dolgokról különböző felmérő csoportok, hanem elkerülhető mindez és így csökkenti a dolgozókkal szembeni igénybevételt is. Amellett lehetőséget ad arra, hogy azok a szervezők, akik ilyen helyzetfelmérő team-be tartoznak, mélyebben, jobban megismerjék a „saját” szervezeti egységük munkáját is.

1.2.3. A rendszerelemzéshez és -tervezéshez szükséges interjúk megtervezése

Az interjúk lefolytatása, azaz a tényleges helyzetfelmérés több szakaszra tagolható. Az első szakasz magának az interjúnak az elindítása, az emberi kapcsolat megteremtése. A második szakasz az adatgyűjtő kérdőívek kitöltése, a harmadik szakasz a folyamatábrák megrajzolása.

Az interjú lefolytatása során a teendők röviden a következők: Hivatalos megbeszélésen találkozni kell az illető egység vezetőjével, majd ezt követően a csoportvezetőkkel, végül az egyes csoportok dolgozóival is. Nem csupán az egységek vezetőitől és a csoportvezetőktől, hanem valamennyi dolgozótól tájékozódni kell az egység vagy saját munkájával kapcsolatban (mik a munkaköri feladatai vagy az alá tartozó egységnek mik a funkciói). A vezetőtől ezen felül meg kell tudni egysége szervezetét, felállítását. Jól ismert, hogy mindenkitől meg kell kérdezni, milyen információk azok, amelyekre munkájához szükség van, milyen döntéseket kell hozni még akkor is, ha ezek egészen mikroszintű rutindöntések, mik a fő problémák a jelenlegi információ-áramlatban (vagy az információáramlattól függetlenül is) és milyen lehetséges javítási, helyesbítési lehetőséget lát a jelenlegi állapothoz képest stb.

A helyzetfelmérés lényeges mozzanata az interjú készítésekor az adatgyűjtő kérdőív kitöltése. Ezek az adatgyűjtő kérdőívek általában arra vonatkoznak, hogy jelenleg milyen információt dolgoznak fel a vállalat egyes szervei, szervezetei, egységei, dolgozói. Az interjút ezután össze kell foglalni, kiemelve azokat az elemeket, amelyek az egész rendszer szervezése szem-

pontjából fontosak. Hangsúlyozni kell, hogy az interjú befejezése után az interjút végző csoportnak vagy szervezőnek el kell készítenie az interjú során elhangzottakat rögzítő dokumentumot.

1.2.4. A felvett adatok hitelesítése, ellenőrzése

Mielőtt a helyzetfelméréskor összegyűjtött adatokat elemeznénk és további tervezéshez felhasználnánk, szükség van a felvett adatok ellenőrzésére. Ezt legkönnyebben úgy végezhetjük el, ha a kérdőívet vagy interjújegyzőkönyvet kitöltő szervező (vagy team) a kitöltött lapokat hitelesítés céljából a dolgozónak visszaadja. Ezután — szükség esetén — a dolgozó a jegyzőkönyvet kiegészíti, végül aláírásával hitelesíti.

2. fejezet

Rendszertervezési módszertanok

2.1. A tervezési, modellezési módszerekről

2.1.1. Output-orientált tervezés

Jelenleg a tervezési módszerek egy része a teljes részletességgel megtervezett output igényekből indul ki, és azokat a feldolgozásokon át visszavezeti a bejövő input adatokig, tárolt adatokig. Ha ismertek és hosszú távon rögzítettek az információigények, változatlanok az előállításához szükséges algoritmusok, valamint az alapadatok nem túlzottan összetettek, és előre meghatározható a feldolgozási ciklus, akkor ezek a módszerek beválnak. Nagyon fontos és lényeges jellemzőjük az adatállományok valamint ebből következően az adatok és a kódok szevezeti egységenkénti tagoltsága és az így különállóként létrejött adatfeldolgozások közötti „kínai falak”. Baj akkor van, ha az elemzési, a tervezési és a gazdálkodási területen az információigény dinamikusan változik, de az információrendszer nem képes rugalmas outputot biztosítani. Itt ezért szükség lehet párhuzamos kézi nyilvántartásra is, illetve olyan tipikusan számítógépes feladatok kézi végrehajtására, mint például a gépi outputokon alapuló elemzések (mivel új igény esetén nincs lehetőség az input és a törzsadattárak gyors átszervezésére).

A rendszertervezés output-feldolgozás-input szemléletéből fakadóan, a rendszerek elkülönített volta miatt, a törzsadattárak tervezése a feldolgozásra van orientálva. Ez egy-egy részrendszer esetén jó hatékonysághoz vezethet, integrált rendszereknél azonban az összes alkalmazás hatékonysága előre így már nem vehető figyelembe. Ily módon természetes, hogy hagyományos módszerrel nem tudjuk megtervezni a későbbi integráláshoz legjobb hatékonyságú adatállományokat. A változtatás viszont a karbantartási igények miatt nehézkes.

Az alkalmazási programot akkor is újra kell írni, ha az információszerkezetek változatlanok ugyan, de az alkalmazó szervezetben az információszer-

vezési és -tárolási viszonyokban, illetve a hardware és software környezetben változás lép fel. A régi programok módosításának az idő múltával csak növekvő terhet egyre nehezebben tudja kezelni a fejlesztő vagy alkalmazó szervezet.

2.1.2. Adatorientált (azaz adatbázis) tervezés

Az alapadatok, szemben a belőlük létrehozott információkkal, egyetlen szervezet vonatkozásában viszonylag állandóak, és számuk jelentősen kevesebb, mint a belőlük nyerhető lehetséges outputok száma. Nagyon lényeges különbség, hogy míg a hagyományos szemléletű adatfeldolgozásban az adatok közötti kapcsolatok programokon keresztül valósulnak meg, addig az adatbázis maga tartalmazza ezeket a kapcsolatokat. Az adatbázisban navigáló funkcionális programoknak voltaképpen semmi közük egymáshoz, sokat közülük egymástól függetlenül, párhuzamosan is végre lehet hajtani. Mindezek miatt az alapadatok körének és kapcsolatainak megtervezése lépett előtérbe az outputoktól indító tervezés helyett. Az outputok egy része lekérdező rendszerrel szolgáltatható. Csak a gyakori és összetett jelentéseket célszerű előre programozni.

Adatbázison alapuló feldolgozásnál az aktuális input általában maga az adatbázis. Például a készletmozgásokat gyakran aktuális inputként kezelik. Azonban ezek törzsadatként való tárolása nemcsak a gépi könyvelést segíti elő, hanem a forgalom elemzését is. Ezért adatbázisoknál általában a mozgás is törzsadat. Az outputok tervezését nemcsak az adatbázis, hanem az inputok tervezése is megelőzi. Az adatbázisorientált módszerrel olyan rugalmas rendszert lehet tervezni, amely a változtatásokat dinamikusan követi, ugyanakkor csökkenti a tervezés átfutási idejét, és a programozási időt.

Az adatbázist akkor lehet hatékonyan felhasználni és alkalmazni, ha egy könnyen kezelhető, az adatbázist kezelő rendszer (röviden:ABKR) is rendelkezésre áll.

2.1.3. Stukturált rendszertervezés

A rendszertervezésnek ez a módszere az úgynevezett moduláris tervezés elveinek továbbfejlesztésével jött létre. Álláspontja szerint egy rendszer csak úgy ismerhető meg, ha részekre bontjuk, mégpedig valós funkciót ellátó ré-

szekre (modulokra). A rendszer részei hierarchikus struktúrába rendezhetők. A részeket a „fekete doboz” elv alapján határozhatjuk meg. A hierarchiát felülről lefelé haladva fejlesztjük ki, a lépésenkénti finomítás elvét alkalmazva. Akkor mondhatjuk, hogy megalkottunk egy modellt, ha leírjuk az elemeit, az elemek tulajdonságait, továbbá ismerjük, hogy milyen kapcsolatok állnak fenn az elemek, illetve a tulajdonságok között. Az információs rendszer elemeinek vizsgálata — és az adatkezelési módszer fejlődése — bebizonyította, hogy a rendszermodell két elkülöníthető, de egymással kapcsolatban lévő modellből áll: az adatmodellből és az eljárásmodellből.

A feladat tehát kettős: meg kell terveznünk az adott feladat adatmodelljét és eljárásmodelljét. Illeszteniük kell őket oly módon, hogy együttesen szerves egészként szolgáltassák a rendszermodellt.

A javasolható strukturált tervezési módszer végrehajtása tehát két fő úton halad: eljárásmodell és adatmodell is készül, az előbbi főként az utóbbi alapján tervezett adatbázis helyességének, teljesítőképességének ellenőrzését, javítását szolgálja. E tervezési módszer főbb lépéseinek összefüggéseit HI-PO-szerűen az 1-12. ábrák mutatják be. A gyakorlatban joggal felmerül a kérdés, mit tervezzünk először: a rendszer adatmodelljét vagy eljárásmodelljét? Mivel mindkettőnek van önálló struktúrája, meg kell vizsgálni, hogy van-e logikai kapcsolat a két struktúra között. Ha van, akkor következtethetünk-e az adatmodell struktúrájából az eljárásmodell struktúrájára vagy fordítva? Az adat- és eljárásmodell között az egyik jelentős kapcsolódási terület az ún. hozzáférési modell.

2.2. Gyors átvilágítás és az alrendszerek képzése

A következőkben olyan eljárásokat ismertetünk, amelyek az információrendszer fejlesztésének első szakaszában a problématerületek és az alrendszerekre bontás kérdéseit kezelik. Négy ilyen módszert tekintünk át röviden:

- a CSF (Critical Success Factors) elsősorban átvilágításra,
- az SA (Structured Analysis) szabályozási körök elhatárolására,
- az ISAC (Information Systems Work and Analysis) ugyancsak szabályozási körök elhatárolására,
- a BSP (Business Systems Planning) alrendszerek elhatárolására alkalmas a fejlesztés kezdeti szakaszában.

2.2.1. A CSF (Critical Success Factors) átvilágítási módszer

A CSF a vezetői információigényeket súlypontozza. A módszer abból indul ki, hogy egy adott munkakört betöltő vezetőnek ahhoz, hogy sikert mutasson fel (de legalábbis ahhoz, hogy a helyén huzamosabb ideig megmaradjon) néhány, ebből a szempontból kulcsfontosságú dologról, ügyről, eseményről, véleményről tudnia, rendszeresen informálódnia kell. Ezeket a „kritikus sikertényezők” (CSF) nevezhetjük. Általában nincs egyszerre tíznél több olyan tényező, amelyre egy vezetőnek figyelnie kell. Ezek azonban még időfüggőek is, így ezeket rendszeresen felül is kell vizsgálni, hogy megfelelően lépést lehessen tartani az aktuális „üzleti klímával”. Lehetséges, hogy a vezetőben még nem tudatosult elég mélyen, hogy egyik vagy másik tényező, amelyet rendszeresen figyel, elemez, végső soron sikeres egyéni pályafutása érdekében kulcsfontosságú számára. Ettől függetlenül ezt a szervezőnek fel kell ismernie, s fontos tanulság lehet, hogy az információrendszernek egy konkrét vezető szemszögéből vett sikere érdekében nem annyira a szervezetben betöltött munkaköre, mint inkább az egyén főnökének éppen aktuális elvárása döntő jelentőségű.

A CSF szerint a kritikus tényezőket négy fő körülmény határozza meg. Az első a szervezet fő profilja, illetve az ennek megfelelő népgazdasági ágazat. Az ágazat sajátos problémái valamennyi benne működő szervezetnél észlelhetők. A második a szervezetnek az ágazaton belül elfoglalt helyzete, súlya, szerepe. A harmadik az ország általános gazdasági, társadalmi problémái. A negyedik a működés azon területei, melyekre általában nem kell különösebben odafigyelni, de amelyek most éppen nem működnek megfelelően, így különös figyelmet érdemelnek. Ilyen kritikus tényező lehet pl. a minőségromlás. Az a tapasztalat, hogy a kritikus tényezők szervezettől, időszaktól és vezetőtől függően változnak.

A kritikus tényezők meghatározásának egy lehetséges módja, hogy többször rögzítik a szervezet hosszú-, közép- és rövidtávú céljait. Ezek felsorolása már kiindulópontja a kritikus tényezők meghatározásának. A szükséges információkat általában interjú során lehet az egyes vezetőktől megtudni. A meglévő információrendszer ismeretében a szervezők megtervezhetik a vezetőt tájékoztató legfontosabb jelentések formáját. A nehezen mérhető tényezők mértékére vonatkozóan pontossági skálákkal ellátott nyomtatványokat

célszerű tervezni. A jelentések végső formátumát természetesen részletesen megbeszéljük az érintett vezetőkkel. Ha bármi módosításra szorul, egy újabb finomító menetre van szükség. A tapasztalatok szerint a CSF- módszer, amennyiben a szervezet valamennyi vezetőjére alkalmazzák, jelentős mértékben megjavítja a vezetők közötti kommunikációt. Ha CSF-módszerrel történő átvilágítást a felsővezetőkkel kezdjük és fokozatosan haladunk lefelé a szervezetben, akkor — az eddigi tapasztalatok alapján — olyan folyamat bontakozhat ki, amely egységesíti a különféle vezetői információigényeket.

2.2.2. A SADT (Structured Analysis and Design Technique) rendszerelemzési módszere

A SADT a Soflech által kifejlesztett módszertan. Ennek első része a SA (Structured Analysis) módszer. A módszertan tartalmaz:

- egy grafikus nyelvet modellépítéshez,
- módszert a modellek kifejlesztéséhez,
- vezetői eljárásmódot a modellek kifejlesztésének irányításához.

Az az alapgondolata, hogy az elemző, miközben ezeket a grafikus modelleket kialakítja, jobban megértse a rendszert, mint a hagyományos helyzetfelmérési módszerekkel. Lényegében a modellezni kívánt terület felülről lefelé haladó lebontásáról van szó.

A SADT gondolatmenete szerint több, különböző modellt célszerű alkotni egy rendszer elemzése, áttervezése majd fejlesztése céljából:

- funkcionális modelleket, hogy megmondjuk mit kell a rendszernek tennie,
- megvalósítási (implementációs) modelleket, hogy megmondjuk hogyan kell működnie,
- konverzációs modelleket, hogy lehetőség legyen módosításra stb.

A funkcionális modellt tevékenységi diagramok (dobozok) sorozata írja le. A tevékenységi diagramok hierarchikus rendbe szervezettek. A legfelső szintű tevékenységi diagram, néhány alrendszerre bontva, a teljes vizsgált rendszert ábrázolja a környezetébe beágyazva. Nem elképzelhetetlen, hogy több ember-hónapnyi munkát is igényel egy új, komplex rendszer 3-6 főbb alkotó elemének (alrendszerének) a meghatározása. A lebontás során szervezeti változások merülnek fel, a felelősség megosztásának új módja rajzolódik ki, nem ritkán magának a rendszernek a határait is kiterjesztik

stb. Tulajdonképpen ez az igazi strukturált elemzés. A diagramokat nem regisztráló dokumentálásra, hanem a követelmények (igények) kibontakoztatásának szerves részeként alkalmazzák. Talán ez az egyik legfontosabb (és legértékesebb) vonása az SA módszernek.

A tevékenységi diagramokon a téglalapok (dobozok) tevékenységeket, a dobozokat összekötő nyilak pedig adatkapcsolatot ábrázolnak. A diagramok nem folyamatábrák, és a nyilak nem egyeznek meg az adatáramokkal. Minden egyes doboznak, összes nyilaival együtt, teljesen le kell írnia azt a tevékenységet, amire vonatkozik, és semmi olyat nem tartalmazhat, ami kívül áll az illető tevékenységen. A tevékenységi doboz bal oldalába befutó nyíl bemenő adatot, a jobb oldaláról kiinduló nyíl pedig kimenő adatot jelöl. A doboz tetejére befutó nyilak vezérlésekre, míg a doboz aljáról kiinduló nyilak ún. mechanizmusokra vonatkoznak.

A vezérlési nyilaknak mindig szerepelniük kell. Ezek azokat a feltételeket vagy korlátozásokat jelentik, amelyek befolyásolják a tevékenységet. A mechanizmus-nyilakat ritkábban használják. Ezek olyan erőforrásokhoz kapcsolódnak, melyeket akkor használnak fel, amikor a tevékenységet ember vagy számítógép végzi. Az egyik dobozból származó output egy másik doboz vezérlése vagy inputja lehet. Párhuzamosan lejátszódó folyamatokat is ábrázolhatunk az egymás után lefutó nyilak mellett. Minden dobozba beírják a tevékenység 3-6 szavas leírását. Ezen kívül felirattal látják el az összes nyilakat is. Ezután a legfelső szintű diagram minden alrendszerét (dobozát) kibontják egy-egy külön diagramon. Ezek a második szintű diagramok néhány részrendszert tartalmazhatnak csak. Ezen kívül a második szinten lévő diagram inputjai és outputjai a legfelső szintű diagram inputjaiból és outputjaiból származnak.

A kötött szöveg elvét következetesen be kell tartani mind a diagram, mind a doboz szintjén. Ez utóbbi esetben a kötött szöveg a dobozon belül végrehajtott funkciók összességét azonosítja, függetlenül attól, hogy ki és hogyan hajtja végre ezeket a funkciókat. A nyilak több dobozt is összekapcsolhatnak. A felirattal a szervező jelezheti, hogy az egyes dobozok a teljes adatkapcsolatot vagy annak csak egy részét használják fel. Egy tevékenységdoboz akkor „aktíválódik”, amikor felhasznál bizonyos inputokat és vezérlést ahhoz, hogy bizonyos outputot előállítson. Egy diagram teljes „története” (story) tartalmazza az egyes dobozok összes lehetséges aktiválódását. Ez a magyarázata annak, hogy egy SA diagram annyira informáci-

ógazdag. Egyetlen diagram általában annyi információt hordoz, mint 10-15 oldal szöveg. A tevékenységi diagramok lebontásának a folyamata addig folytatódik, amíg további dekompozíciókra már nincs szükség. A funkcionális modell kidolgozását követően kell hozzáfogni az adatmodell létrehozásához, az adatok felülről lefelé történő dokumentációjának munkájához. Ennek során felrajzolják az adatmodell egymást követő szintjeit. A legfelső diagram adatosztályai például tartalmazhatják az adatfile-okat, tranzakciókat és jelentéseket. Az adatmodell a rendszernek teljesen más képét tükrözi, mint a tevékenységi modell. Gyakran megesik, hogy a szervezőnek az adatmodell felállítása során új felvetései támadnak a követelményeket illetően, és vissza kell térnie a funkcionális modellre, hogy módosítsa azt.

2.2.3. Az ISAC (Information Systems Work and Analysis of Changes) helyzetfelmérési és rendszerelemzési módszertana

A módszertan három, egymás után alkalmazandó módszere alkotja a helyzetfelmérést és a rendszerelemzést:

- változtatáselemzés,
- tevékenységelemzés,
- információelemzés.

A változtatáselemzés az a célja, hogy felkutassuk a szervezet dolgozóinak problémáit, és találjuk is meg a problémák valódi okát, ne csak a tünetekkel törődjünk. Fontos ezenkívül az is, hogy a fejlesztési intézkedések megfelelő kombinációját válasszuk ki. Előfordul, hogy az aktuális folyamatokban, szabályozásban kell változtatásokat eszközölni, és nem (vagy nem csak) az információs szférában, mert a problémákat kiváltó okok csak így szüntethetők meg. A változtatáselemzés a problémák összegyűjtésével, majd az okok feltárásával, elemzésével kezdődik. Ennek eredménye többek között egy ún. problématáblázatban és az érdekelték listájában foglalható össze. Az aktuális tevékenységet tevékenységi gráfokkal és tulajdonságtáblákkal írjuk le. A tulajdonságtábla a probléma kvalifikált leírását adja. Minden változtatási alternatívát többek között egy új tevékenységi gráf és tulajdonságtábla sorozattal célszerű leírni. Ennek alapján lehet eldönteni, milyen változtatást érdemes eszközölni.

A tevékenységelemzés már a szervezet csak azon tevékenységeinek elemzésével és tervezésével foglalkozik, melyek a jövőbeli információrendszer-

rel kapcsolatosak. Az elemzés a tevékenységek részletes leírásával kezdődik. Ezt a részletezést addig kell bontani, amíg az információs alrendszerek külön tevékenységként nem jelentkeznek. Ezt a részletes tevékenységgráfokon és a tulajdonságtáblákon dokumentálják. Ezután minden egyes információ-alrendszerre költség-haszon elemzést kell végezni.

Az információelemzés célja, hogy leírja, mit tartalmazzon és mit csinálgjon a jövőbeli információrendszer. A leírást egyrészt arra használják, hogy közös specifikáció legyen a rendszerelemzésben résztvevők között az egyes részek információtartalmáról folyó eszmecsere megkönnyítésére, másrészt arra, hogy pontos alapja legyen az ezután következő alrendszer tervezésének. Az információelemzést minden egyes információ-alrendszerre külön kell elvégezni. Az információelemzés mélysége attól függ, hogy milyen mértékben lehet egzaktul megfogalmazni, algoritmizálni az egyes alrendszerek működését. Azokra vonatkozóan, amelyeket alig lehet formalizálni mivel kevésbé gépesíthetők durva információelemzés is elegendő. Azokra a részekre nézve viszont, amelyeket automatizálni lehet, mélyebb információelemzést kell végrehajtanunk, mielőtt hozzákezdénénk az adatrendszer tervezéséhez. Az információelemzés a tevékenység elemzéséből származó tevékenységgráfnak megfelelő vázlatos információ-precedencia gráf megszerkesztésével indul. Ezután komponenselemzéssel váltakozó precedenciaelemzést kell végrehajtani. A precedencia (azaz sorrendi) elemzés azokat a szükséges és elégséges információelőzményeket kutatja fel, amelyekkel el lehet jutni a kívánt információhalmazokhoz. Ennek során fel kell tártani a teljes input információhalmazt.

A komponenselemzés a különféle információhalmazok tartalmának és felépítésének vizsgálata. A precedenciaelemzés eredménye többek között az ún. I-gráfokon, a komponenselemzés eredménye pedig az ún. C-gráfokon (komponenskapcsolat-gráfok) ábrázolható.

Ezután a folyamatelemzés következik. Ennek során a folyamatokat differenciáltan meg kell fogalmazni. Ezek többek között a folyamatok listájához kötve és a folyamatáblákon dokumentálhatók.

Az adatrendszer tervezésével párhuzamosan további tulajdonságelemzés is végezhető. Az eredmények ugyanolyan tulajdonságtáblákba foglalhatók össze, mint a változtatáselemzés és a tevékenységelemzés esetén. Az ezután következő adatrendszer-tervezés célja az, hogy eszközfüggetlen megoldást alakítson ki a specifikált információ-alrendszerekre.

Az ISAC-módszer hangsúlyozza a felhasználók részvételét a változtatóelemzés folyamatában.

2.2.4. A BSP (Business Systems Planning) átvilágítási módszer

A BSP az információrendszer alrendszerekre tagolásához és az egész információrendszer fejlesztésének megtervezéséhez, irányításához fontos kiinduló adatokat szolgáltat. Nem csupán vállalati, hanem az intézményi, hatósági stb. jellegű szervezetek információrendszerének tervezésére is alkalmas. Közép-, esetleg hosszabb távon is olyan, viszonylag stabil információrendszervázlat és tagolást hoz létre, amely a szervezet valamennyi folyamatát átfogja. Várható, hogy mindaddig nem lesz szükség a szervezet információigényeinek hasonló meghatározására és új fejlesztési terv készítésére, amíg nagyméretű profilváltozás nem következik be a szervezet életében.

A módszer két előkészítő lépésből és tizenegy végrehajtó lépésből áll. Vegyük ezek közül röviden sorra néhányat:

A közvetlen vezetői részvétel biztosítása

Amennyiben a szervezet egészére vonatkozó információrendszer fejlesztéséről van szó, akkor a felsővezetők köre, ha kisebb egységről (egységek), akkor azok vezetői végzik aktívan (5-7 fő) a tervezési munkát. A vezetői munkacsoportot a legmagasabb beosztású vezető irányítja, a résztvevőket ő jelöli ki. Az egész átvilágítás hatékonysága a résztvevő vezetők áttekintőképességétől és vezetői tudásától, az általuk vezetett terület feladatairól és az azok elvégzéséhez szükséges információk rosszabb vagy jobb ismeretétől függ. A BSP-szakértő (rendszerszervező) által készített munkatervből és a szóbeli kiegészítésekből a vezetői munkacsoport minden tagja megtudja, mi lesz a feladata, miért és hogyan kell azt végrehajtania, ahhoz mely adatokra lesz szüksége. A munka egy későbbi lépésében időt kell szakítani egy sor vezetői interjú készítésére azokkal a vezetőkkel, akik a munkacsoportban nem vesznek részt. Miután a résztvevők a munkához szükséges dokumentált adatok nagy részét összegyűjtötték, és a BSP-szakértő is elkészítette a csoportmunka részletes, személyre, napra lebontott ütemtervét, megbeszélést tartanak. A csoportmunka ezután következő időtartamára (6-8 hét teljes munkaidő javasolt az IBM szerint) lehetőleg karanténszerű nyugodt környe-

zetet kell biztosítani. A résztvevők feladata: ismereteiket összegyűjteni és rendszerezni a területükön meglevő vagy tervezett számítógépes feldolgozásról, a szervezeti sémákról, a készítendő tervfajtákról, adatszolgáltatási és beszámoló jelentésekről, nyilvántartásról, valamint a tipikus problémákról; és emellett összeállítani a szervezetre vonatkozó feladatkörök, folyamatok nomenklatúráját.

A folyamat meghatározásának alapja az a gondolat, hogy nem csupán a munkaerővel, munkakörökkel, anyaggal és pénzzel, hanem az egyes termékekkel, szolgáltatásfajtákkal stb. is gazdálkodunk, hiszen az ezekből meglevő „készlet” szintén erőforrás.

Az adatosztályok meghatározása

Az adatosztályok meghatározásának két módszerét is alkalmazhatja a BSP. Az első azokból a fontosabb „dolgoz”-ból indul ki (pl. a tanács esetében: földterület, népesség, építmények, közművek, szervezetek, anyag, energia, szolgáltatások=pénz stb.), amelyeknek jórészt ez a szervezet a „gazdája”, amelyekkel ő sáfárogdik (használ), vagy amelyeket ő állít elő, hozzá kapcsolódó adatok meghatározásán alapszik. A másik módszer a munkafolyamatokat veszi tekintetbe, és ezen keresztül végzi el az osztályozást.

Az adatosztályok „dolgoz”-ból kiinduló azonosítása úgy történik, hogy egy táblázat fejrovatába összeírják a szervezet profilja szempontjából legfontosabb (10-20) „dolgot”, majd a táblázat oldalrovatában négy sort vesznek fel: a tervdokumentációk, az állománynyilvántartások, az ellátást és visszavonást tartalmazó, továbbá a statisztikák és beszámolók, mérlegek, adatszolgáltatások sorát. Ezután következik a táblázat rovatainak kitöltése azoknak az alapvetően fontos dokumentumoknak a nevével, amelyek a megfelelő helyre illenek. Ezt az eljárást megkönnyíti, ha már kész bizonylati albumunk van a szervezetre vonatkozóan. Ezzel szemben, a folyamatból kiinduló adatosztály-meghatározási módszer lényege az, hogy minden folyamathoz, melyet az előző lépésben határoztak meg, felsorolják mind a legalapvetőbb felhasznált (input), mind az előállított (output) adatokat.

Elvileg mindkét módszer ugyanoda kellene, hogy vezessen, de a gyakorlatban ajánlatos mindkét nyomon elindulni és ezek eredményének összevetésével kb. 30-60 adatosztályhoz eljutni. Ezért az IBM a BSP csoport két részre osztását és a két eljárás egyidejű követését, majd közös munkával a

két eredmény „összefésülését” és az így kapott számos adatosztály közül csak a legfontosabbak (kb. 50-60) megtartását javasolja.

3. fejezet

Adatmodellezés

3.1. Az adatmodellezés fogalma, sajátosságai

Az adatmodellezés az adatok jelentéséből, információtartalmából eredezteti csoportosításukat és a csoportok közti kapcsolatokat. Az előállítandó adatmodellt korántsem lehet csupán formai, szintaktikai jegyekkel jellemezni. Jelenleg nem ismeretes olyan módszer, amely a felhasználói információigények vagy akár a helyzetfelismerési adatok alapján közvetlenül tervezné meg és állítaná elő az adatmodellt. Az adatmodellező tehát nem is tervezi, csupán feltárja az információrendszer elvi adatmodelljét vagy elvi sémáját. A logikai adatmodellezés szakaszában már az elvi adatmodell alkotta információstruktúrából és a felhasználói információigényből kiinduló olyan tervezés folyik, ahol azt vizsgálják, hogyan lehet a fenti információstruktúrákat az adott korlátok és célfüggvény mellett adatstruktúrákká leképezni. Az adatstruktúra a ténylegesen tárolásra kerülő adatok adott szemantikai és szintaktikai szabályok szerinti leírása. Ebben a leírásban már a lényeges hardware és software környezet sajátosságait is figyelembe kell venni.

Az elvi séma mibenlétéről legalább két, jól elkülönülő — egy tágabb és egy szűkebb — álláspontot is megfogalmaztak. Az egyik szerint az elvi séma a való világot és így a valóság összes lehetséges állapotának egy részalmazát tükrözi. A másik szerint csupán egy adatbázis fogalmi tartalmát, és így a szóban forgó adatbázis összes megengedett állapotának halmazát írja le. Véleményünk szerint az elvi séma a való világ egy részének — idődimenziótól részben vagy teljesen független — képét tükrözi, mégpedig úgy, hogy a való világ ezen részében előforduló valamennyi fogalom és a közöttük fennálló és általunk az adatmodellbe felvételre érdemesnek talált kapcsolat definícióját tartalmazza. Az idő fogalmát maguk a definíciók veszik figyelembe. Nem szükséges, hogy a megvalósított adatbázisban is megjelenjen minden fogalom és kapcsolat, amely az elvi sémában szerepel, de az igen, hogy az

adatbázisban csakis olyan fogalmak és kapcsolatok lehessenek, amelyeket az elvi sémában már definiáltunk.

Fogalmi modellen az adatbázis olyan időfüggő fogalmi tartalmát értjük, amely semminemű, a megvalósítással és a hatékony működéssel összefüggő elemet nem tartalmaz. A fogalmi modellnek meghatározható belső szerkezete van. Ez a belső szerkezet az adatbázistervezés adatmodellezési szakaszában úgy alakul ki, hogy egyrészt magának a való világnak a tartalmát és szerkezetét, másrészt az adatbázis fogalmi sémájának meghatározásához szükséges modellezési szabályokat a tervező figyelembe veszi. Az elvi sémák és fogalmi modellek azonosíthatók például a belső szerkezetük szerint is. Ennek azért van jelentősége, mert így arra, hogy pontosan mely részekből, szakaszokból is áll egy modellezési folyamat, következtetni lehet abból, hogy mi lett vagy előreláthatólag mi lesz a fogalmi modell szerkezete. Annál nagyobb jelentősége van ennek, minél bonyolultabb, összetettebb fogalmak (pl. gazdasági mutatók, indexek) szerepelnek egy adatbázisban, hiszen az őket alkotó egyszerű fogalmakat és az azok között fennálló szemantikai kapcsolatokat is fel kell tárnunk ahhoz, hogy a korrekt fogalmi modell jól alapozza meg az adatbázis hatékony működését. Fontossága pedig akkor tárul elénk, ha olyan adatbáziskezelő rendszert (ABKR) kívánunk tervezni, amely éppen az általunk kezelhető fogalmi szerkezetek típusához igazodó mivolta miatt várhatóan eleve igen hatékonyan lesz képes a kezelt adatok manipulálására.

Az adatmodellezés többféle módszerrel történhet. Az alkalmazott ki-fejezések és fogalmi definíciók ugyan nem esnek egybe, az adatmodellezés alapvető fogalmai azonban (bár az egyes módszerek némelykor eltérő elnevezéseket használnak) az egyed, a tulajdonság és a kapcsolat, vagy a használt más fogalomváltozatok ezekre visszavezethetők. Egyed lehet mindaz, amiről információt szerzünk. Végül minden adat vonatkozik valamire. Ez a valami lehet dolog, esemény, személy stb. Az információrendszer nem közvetlenül magát ezt a valamit (nevezzük egyedeknek), hanem nevét és az egyedek saját nevén kívül még jellemző egyéb tulajdonságai neveit mint adatokat kezeli. Természetes nyelvi szövegben az egyedeket névszók jelölik, rendszerint vagy alanyként vagy tárgyként. A tulajdonságok általában névszók vagy határozószók, jelzőként, illetve állítmányként használva. A valóság összefüggései következtében fennáll, hogy egy meghatározott egyed saját nevén kívül a rá vonatkozó jellemző tulajdonságok, azaz nevek (adatok) között tulajdonság-

ként egy másik egyed neve is található. A természetes nyelv legtöbb mondata ilyen egyedek közötti kapcsolatot fejez ki.

Mind az egyedek, mind a megnevezésük halmazokat alkotnak. Minthogy e halmazok meghatározhatóak (egyik sem eleme önmagának), a logikából átörökítve, típusoknak nevezik őket. A típus fogalmát a logika a halmazelmélet ellentmondásainak kiküszöbölése végett dolgozta ki; a számítástechnikában modern programozási nyelvek (PASCAL, ALGOL 68, ADA) tervezése során alkalmazták. Itt a típusok olyan értékek véges készletét jelentik, amelyeket az egyes adatok potenciálisan felvehetnek. A típus fogalmával analóg a statisztikában használatos sokaság fogalma is. Ezt azért kell megemlíteni, mert az államigazgatási információrendszer jelentős részben statisztikákat, adatgyűjtéseket ölel fel. A továbbiakban csak az információrendszer által kezelt egyedek neveire gondolva, e neveket hívják egyedtípusoknak. Minden egyedtípuson belül létezik egy rendezési reláció (amely irreflexív, antiszimmetrikus, tranzitív és trichotóm), amelynek az a gyakorlati jelentősége, hogy a típusba tartozó egyedelőfordulások a számítógépben láncú alakíthatók. Az egyed fogalma a számítástechnikában gyakran a rekord, illetve a részrekord (szegmens) fogalmával azonosítható. Így az egyedtípus fogalma a gyakorlatban megfelel az adott rekordokból álló adatállomány leírásának.

Az egyedtípushoz hasonlóan lehet beszélni tulajdonságtípusról (a tulajdonságértékek véges készlete). Ennek a statisztikában használt szinonímája az ismerv (az ismerv értékek, illetve ismervváltozatok véges készlete). Az ismerv (tulajdonságtípus) értékeinek felsorolása diszkrét ismerv esetében lehetséges, de ha igen sok értéke fordul elő, akkor a folyamatos ismerv esetével megegyezően, már az osztályközökre bontást alkalmazzák. A statisztika az ismerv fajtája szerint megkülönböztet minőségi és mennyiségi, valamint területi és idő ismervlistákat. A számítástechnikai gyakorlat általában a mezőt tekinti tulajdonságértéknek. Adott rendszerben minden egyedtípus meghatározott tulajdonságtípusokkal jellemezhető, tehát minden egyedre vonatkozóan adott az őt leíró tulajdonságok sorozata. Egy típuson belül az egyedelőfordulások akkor különböztethetők meg, ha egy (és csakis egy) azonosítható tulajdonságtípusban eltérő értékekkel (vagy ismervváltozattal) jellemezhetők.

Valamely egyedelőforduláshoz egy tulajdonságtípusból minden időpillanatban egyetlen tulajdonságérték tartozik, és ezen értékek egyenként minden alkalommal részhalmazokra bontják az egyedtípust. Működő szervezeteknél

nagyrészt rendszeresen ismétlődő folyamatokkal találkozunk. Különböző folyamatokat leíró adatok összehasonlítása csak úgy lehetséges, ha az érintett egyedtípusok releváns tulajdonságai egybeesnek. Ennek biztosítására a releváns ismérvek értékeiről listát, azaz nomenklatúrát kell létrehozni.

Az egyedelőfordulások nevei (azonosítók) összehasonlíthatók. Ezen keresztül a nomenklatúra felhasználásával az egyedrészhalmazok között végre lehet hajtani az ismert halmazműveleteket. Különböző egyedtípusok között azonban csak a Descartes-szorzat értelmezhető. Mint már az előbb rámutattunk, a tulajdonság fogalma nem kötődik egyértelműen az általa jellemzett egyed fogalmához. Igen gyakran különböző típusokba tartozó egyedek tulajdonságai is összehasonlíthatók. Ez a számítógépes gyakorlatban azt jelenti, hogy a különböző mezők értékei összehasonlíthatók. Ezen túlmenően, létezik az ún. gyenge logikai redundancia jelensége, ami röviden azt jelenti, hogy az egyedtípust jellemző tulajdonságtípusok egyike-másika egyedtípust jelölhet. Ez az az eset, amikor a különböző egyedtípusok közötti kapcsolatokról lehet beszélni. Használható a Descartes-szorzat segítségével formalizálható kapcsolattípus fogalma, amely akkor meghatározott, ha az egyedtípusok egyenként meghatározottak, és adott az a szempont, amelynek alapján eldönthető, hogy a Descartes-szorzat egyes elemei (a kapcsolat-előfordulások) a kapcsolattípusokba tartoznak vagy sem.

A természetes nyelvek csaknem minden mondata egy kapcsolat kifejezésének tekinthető. Az egyedelőfordulásból képezhető a kapcsolatelőfordulás, az összetartozó kapcsolatelőfordulások összessége a kapcsolattípus. Bármely két egyedtípust kiválasztva, előbb-utóbb található (legalábbis valamiféle közvetett) kapcsolat közöttük. Nem mindegy azonban, hogy az adatmodell megalkotása során mely kapcsolatok lesznek hangsúlyozva és melyek elhanyagolva. Azokat a kapcsolatokat kell általában jelentősnek tartani, amelyek függvénykapcsolatot fejeznek ki. A függvények fontossága azért rendkívüli, mert gépi algoritmusok, automaták segítségével csak függvényeket tudunk megvalósítani; nagyon elvont értelemben mindenfajta működőképes rendszert függvényként lehet megtervezni. Így magának az információrendszernek az eljárásmodellje is egy ilyen összetett függvény leírása. A relációkon értelmezhető vetítés (projekció) és az összekapcsolás (join) csakis olyan rendszerben végezhető el korlátlanul (információvesztés nélkül), amelyben bármely két egyedtípus között vagy nem létezik kapcsolat, vagy ha van, akkor az függvénykapcsolat. A rendszer elemei között a hatást kifejező függvény-

kapcsolatok azok, amelyek a legfontosabbak. Ezeket a tulajdonságtípusok és egyed típusok között az adott esetben valóban létező kapcsolatokat nevezik funkcionális függéseknek. Megállapítható az, hogy az egyed típust jellemző többi tulajdonságtípusok is funkcionálisan függenek az egyed azonosító tulajdonságtípustól.

Az adatmodellezés szempontjából fontos függvények közötti művelet a függvénykompozíció (balösszetett függvény képzése), mivel asszociatív volta következtében segítségével funkcionális függésekből egyszerűen állíthatók elő közvetett összefüggések. Két egyed típus között valamiféle kapcsolat mindig felfedezhető. E kapcsolatok közül a közvetlen hatásokat fejezik ki és így igen lényegesek azok az — egyed típusok között létező — kapcsolatok, amelyek determináló, azaz függvényjellegűek.

A funkcionális függvények kiválasztásának egyik módja az olyan ún. redukálhatatlan kapcsolatok megfogalmazása, amelyekre a többi kapcsolat visszavezethető. A tapasztalat azt mutatja, hogy az események megtörténését kifejező kapcsolat általában redukálhatatlan. Továbbá az is fennáll, hogy az egyed típuson belül az egyed leíró tulajdonságok az egyed azonosító típusoktól függenek funkcionálisan, sőt ez igaz az illető tulajdonságtípusok tetszőleges Descartes-szorzataira is. A redukálhatatlan kapcsolatok kiválasztásának alapja az, hogy ha két egyed típus között létezik valamilyen kapcsolat, reláció, akkor maga ez a reláció már funkcionálisan függ mindkét típustól.

3.2. Az analitikus és szintetikus adatmodellezési eljárás

Adatmodellek készítésének két fő módszere ismeretes: az analitikus és a szintetikus eljárás; de bármelyik kétféle módon is értelmezhető. E két értelmezést a szakmai zsargon nagyvonalúan egybemossa, de itt meg kell különböztetni őket:

- a) Az első értelmezés szerint az adatmodell létrehozásának módszere analitikus, ha almodellekből épül fel a nagyobb részt átfogó modell, és szintetikus, ha külön kisebb almodellek megformálása nélkül a cél egyenesen az átfogó modell megkonstruálása.

- b) A második értelmezés szerint az eljárás analitikus, ha előbb egyedtypusokat határoz meg, majd ezekhez rendeli a tulajdonságtípusokat, illetve az egyedtypusok között határoz meg kapcsolatokat. Ezzel szemben az eljárás szintetikus, ha a tulajdonságtípusokból indul ki, és a közöttük feltárható funkcionális és kölcsönös függések, valamint a funkcionális függetlenség figyelembevételével rögzíti az egyedtypusokat és a kapcsolatokat.

Természetesen lehetséges, hogy a módszer pl. az első értelmezésnek megfelelően szintetikus, míg a második szerint analitikus stb. A fent felsorolt alapvető módszerek, illetve azok valamely kombinációjának kiválasztásában például a következő szempontokat érdemes figyelembe venni: A létrehozandó adatbázisok — legalábbis végső kiépítésükben — nagy méretűek lesznek, és igen sokrétű felhasználást kell kielégíteniük. Számítástechnikai előzményük alig van; általában eddig manuálisan vezetett nyilvántartásokat kell számítógéppel kezelni, némely esetben azonban ezek a nyilvántartások is hiányoznak. Figyelembe kell azt is venni, hogy az adatbázisok létrehozása, feltöltése hosszú időt és jelentős anyagi ráfordítást igényel, így nagy szerepe van a fokozatosságnak, a részeredmények lépésenkénti, tervszerű kifejtésének. Megalapozottabb elgondolás ezért az, hogy előbb néhány felhasználási területhez kapcsolódó almodellt érdemes kidolgozni, (majd később integrálhatók ezek az almodelleket nagyobb modellé). Így az almodellek önállóan is kifejezhetők, és alkalmat adnak adatbáziskezelési tapasztalatok gyűjtésére, ami az előzményekkel alig rendelkező számítógépkalmazás esetében különösen fontos. Ezt a megalapozottabb elgondolást célszerű követni, különösen akkor, ha a fejlesztés több, földrajzilag távol eső bázison zajlik, s koordinált munkájuk hatékonyabb, ha egy-egy bázis egy-egy alrendszer gazdája. Az alrendszer egészét tekintve tehát, egy adatbázisnak előbb pl. a népességyilvántartási részrendszere — esetleg annak is csak egy része — készül el, majd pl. az épület- és lakásnyilvántartás jön létre, és a kettő összekapcsolása csak egy további lépésben következik.

Összefoglalva leszögezhetjük, hogy általában az adatmodellezés a) pontban tekintett analitikus módszerét célszerű követni. Ezt megelőzően azonban az információs rendszer egészét alrendszerekre kell bontani és ezek tervszerű kifejezési sorrendjét, a fejlesztés bázisát, földrajzi helyét, a szükséges erőforrásütemezést kell meghatározni. Erre jelen esetben a BSP módszert választottuk ki.

Az adatmodell architektúrája egyed típusokat tulajdonságtípusokkal leíró és az egyedek közötti kapcsolatokat leíró rekordtípusokból áll. Sok szempontból célszerű arra törekednünk, hogy az adatmodell ne legyen redundáns, egy adatot ne tároljon majd többszörösen. Ennek eléréséhez célszerű alkalmaznunk az egyébként eredetileg a relációs modellek egyszerűsítéséhez, tömörítéséhez használatos ún. normalizálást. A szintetizáló eljárás a tervező számára egzaktabbnak tűnik, mert a modell kialakításának kezdetétől alkalmazza a funkcionális függés vizsgálatát. Ugyanakkor azonban, ha tulajdonságtípusok száma nagy, páronkénti összevetésük a kölcsönös és funkcionális függés vizsgálata céljából roppant időigényes, és aligha áttekinthető munka. Ha csupán — mondjuk — száz körüli a tulajdonságtípusok n száma, akkor az összehasonlítások $W = n(n - 1)$, száma csaknem tízezer. Amennyiben viszont a tulajdonságtípusok száma több száz, esetleg eléri az ezret, akkor hozzávetőleg egymillió összehasonlításra van szükség. A gyakorlatban tehát mindenképpen százezres nagyságrendű lenne az összehasonlítások száma, és az összehasonlítások eredményét természetesen dokumentálnunk is kell. Látható a feladat mérete. Ebből következően, az eljárás csak akkor ajánlható, ha a modell készítéséhez számítógép áll rendelkezésünkre. A funkcionális függés mérlegelése természetesen ebben az esetben is a tervezők feladata marad, erre a célra alkalmas programcsomag azonban biztosítja a vizsgálat logikai teljességét, hibamentességét és elkészíti a szükséges dokumentációt. Ha ilyen programcsomag alkalmazása nem jöhet számításba, akkor a szintetikus módszer nem javasolható. Gépi segédeszköz nélkül ugyanis sokkal egyszerűbben végezhető el az analitikus eljárás, ahol először az egyes egyed típusokat és a közöttük fennálló kapcsolatokat határozzák meg, majd az egyed típusokhoz hozzárendelik a tulajdonságtípusokat. Ha közben szükséges, módosításokat hajtanak végre, tehát pl. tulajdonságtípusból egyed típusokat csinálnak vagy fordítva. Ez természetesen maga után vonja a kapcsolatok módosulását is. Végül a kapott egyed típusokat leíró tulajdonságtípusokat magába foglaló rekordtípusokat még külön is ellenőrizni kell, hogy a normalizálás megtörtént-e, és ennek végeredményeként előállt-e az ún. harmadik normál alak. Lényeges feltétel az analitikus módszer alkalmazásakor, hogy az egyes adatokról eldönthető legyen, tulajdonságtípust vagy egyed típust reprezentálnak-e. A legegyszerűbben ez úgy tehető meg, hogy kezdetben mindazon adatok amelyek egynél több tulajdonságtípussal

jellemezhetők, egyedtípusnak, míg azok az adatok, amelyeket csupán jellemzésre használnak, tulajdonságtípusnak tekintendők.

3.3. Az elvi adatmodell kialakítását megelőző jellemzés és helyzetfelmérés

A helyzetfelmérés más módszerekre támaszkodik attól függően, hogy a felmérés szervezett információs rendszerének nagyvonalú architektúráját tárja-e fel az információs rendszer egésze fejlesztésének hatékony tervezése érdekében (átfogó helyzetfelmérés), vagy pedig egy már jól körülhatárolt alrendszer kifejlesztését célozza (részletes helyzetfelmérés). Most a részletes helyzetfelmérést tárgyaljuk.

A részletes helyzetfelmérésnek az adatmodell kialakításáig terjedő szakasza kifejezetten adatorientált lesz. A vizsgálódás közvetlenül annak megállapítása, hogy mely adatok azok, amelyeknek be kell kerülniük az adatbázisba. Itt rögtön felmerül az a kérdés, hogy a majdani adatbázisban legyen-e tárolva képzett adat is, vagy csak alapadat? Ezt a kérdést a felmérés szakaszában még nem kell eldönteni, sőt, igen nehéz is lenne eldönteni, mert a döntéshez sok olyan tényező mérlegelése lenne szükséges, ami ekkor még nem áll rendelkezésre. Ismerni kellene például a képzett adatok pontos képzési módját, az ehhez szükséges alapadatokat, a képzés gyakoriságát, a képzett adatok felhasználási gyakoriságát, tömegét stb. Mindez azonban csak a felmérés közben derül majd ki. Viszonylag kis szervezői létszám esetén célszerűbb, ha ebben a felmérési szakaszban kizárólag az alapadatokkal foglalkozunk. Alapadatoknak azokat az adatokat tekintjük, amelyek az adatbázisfelhasználó szervek működésének keretein belül keletkeznek, de nem más adatokból képezve, vagy pedig a szervezet környezetéből érkeznek, adatszolgáltatás formájában. Ami az egyedtípusok és tulajdonságtípusok egymáshoz rendelését illeti, a felmérés módszere bizonyos fokig különbözik, az analitikus vagy szintetikus modellezési módszer alkalmazásától függően. Ha majd lehetőség nyílik a normalizálást végző program használatára, akkor a szintetikus módszer alkalmazható, és tisztán tulajdonságtípusokra lehet összpontosítani. Ez más szóval annyit jelent, hogy elegendő az egyes elemi adattételeket definiálni. Erre a célra használható az ún. tulajdonságtípus lap (24. táblázat). Más a helyzet, ha az analitikus eljárást kell alkalmazni.

Ekkor az egyedtípusok meghatározása áll a középpontban. Az egyedtípusok kiválasztásához alkalmazhatók az előzőekben az egyedtípusok és tulajdonságtípusok megkülönböztetésére bevezetett irányelvek. Az egyedtípusokat jegyzékbe kell foglalni (19. táblázat).

A tulajdonságtípusok és egyedtípusok egymáshoz rendelése során nem kell teljességre törekedni mindjárt a felmérés és elemzés kezdetekor. A feltárt egyedtípusok ugyanis később is kiegészíthetők még hozzájuk tartozó tulajdonságtípusokkal. Ez elvileg mindaddig megtehető, amíg az adatbázis fizikai tervezése meg nem kezdődik (32. táblázat). Az egyedek közötti kapcsolattípusok meghatározásához segédeszközként használható az ún. egyedkapcsolati táblázat (20. táblázat). Ez egy négyzetes mátrix, amelynek sorai és oszlopai egyaránt a feltárt egyedtípusokat tartalmazzák, mégpedig azonos sorrendben. A sorok és oszlopok találkozásánál azt kell bejelölni, hogy a megfelelő egyedtípusok között van-e definiált kapcsolat, és hogy ez a kapcsolat milyen fokú. Mivel az egyes párokon belül az elemek sorrendje közömbös, a mátrix a főátlóra szimmetrikus.

A mátrix kitöltése során a következő kérdésekre kell választ kapni:

- van-e egyáltalán kapcsolat az éppen vizsgált két egyedtípus között?
- lényeges-e ez a kapcsolat az adott felhasználási terület szempontjából?
- milyen fokú a kapcsolat?

A mátrix jó segédeszköz ahhoz, hogy ne maradjon ki a vizsgálatból egyetlen párosítás sem; a megfelelő sor és oszlop találkozásánál a kapcsolat fokát írjuk be, ha nincs kapcsolat, akkor ezt pl. vízszintes vonal jelöli.

Fontos gyakorlati tapasztalat, hogy az egyedkapcsolati mátrix használata önmagában kevés az elemzés lefolytatásához. Világos ugyanis, hogy a valóságos világ egyedei elvileg szinte mindig kapcsolatba hozhatók egymással. A tapasztalat szerint ez a körülmény, valamint az, hogy a felhasználó minél nagyobb biztonságra törekszik, oda vezet, hogy igen sokféle kapcsolatot kívánnak definiálni. Ezzel — úgymond — a valóságos helyzet minél pontosabb visszatükrözésére törekszenek. A mátrixban ezt a tendenciát a sűrű kitöltöttség jelzi. Így a modell igen bonyolulttá válik, nehezen áttekinthetővé, később pedig az ennek megfelelő séma megvalósításakor igen sok set-et kell kezelni az ABKR-nek, ez pedig a működés hatékonyságát nagymértékben lerontja. Ezért ajánlatos, hogy a szükséges kapcsolatok meghatározása során dokumentációként ne csak és kizárólag kapcsolati mátrix, hanem egyidejűleg ún. Bachman-diagram is készüljön. A kapcsolatok ábrázolására W.

Bachman vezetett be grafikus ábrázolásmódot, amelyen a felhasználó világosan láthatja, hogy a már meglévő kapcsolatok hogyan foglalják rendszerbe az egyes egyedtípusokat, és hogy milyen elérési utakat tesznek lehetővé. Mindezt a Bachman-diagram vizuális jellege teszi lehetővé (21. táblázat). Az így feltárt kapcsolattípusokat jegyzékbe foglaljuk (22. táblázat), majd ezen túlmenően egy-egy nyomtatványtípusokon dokumentáljuk valamennyi egyedtypust (23., 24. és 25. táblázatok).

Az adatmodellnek az analitikus vagy szintetikus módszerek valamelyikével történő megtervezésével a helyzetfelmérés első szakasza lezárul. Az így kapott adatmodell semmiképpen sem tekinthető véglegesnek. Az ún. elvi adatmodell végleges formáját még két további lépés biztosítja: a normalizálás és a funkcionális elemzés.

4. fejezet

Funkcionális elemzés

Az adatmodell végleges elvi elemmé továbbfejlesztésének természetes útja a funkcionális elemzés. Szükséges, hogy rendelkezésre álljon az adatkezelési funkciók halmaza. Ehhez nem csupán a folyamatok (funkciók) összegyűjtését (14. táblázat), majd ezek tevékenységsorozatra bontását (17. táblázat) kell elvégezni, hanem az ezekben megjelenő adatkezelési funkciók (röviden funkciók), pl. feldolgozások, karbantartás vonatkozásában kell végrehajtani majd az adatmodell funkcionális elemzését is. Az elvi adatmodell ugyanis a tárolt adatok tényleges felhasználási módját még figyelmen kívül hagyja, és pusztán két dologra épít:

- az adatok szükségességére és
- a tulajdonságok között feltárt természetes szerkezetre.

A már említett harmadik normálalak biztosítja az egyértelműséget, alapot ad az integritás megőrzéséhez, egyszerű karbantartást tesz lehetővé. Figyelembe kell azonban venni a hatékonyság szempontjait is, továbbá bizonyosságot kell nyerni arról, hogy a kapott modellből a használnál szükséges információk előállíthatók-e. Ezeket a célokat szolgálja a funkcionális elemzés.

A hatékonyság figyelembe vétele azt jelenti, hogy a leglényegesebb felhasználói információk generálásának vizsgálata során egy sor statisztika készül a rekordtípusok, a set-ek használatának módjáról, gyakoriságáról, a rekordok kiválasztási szempontjairól, a kiválasztott rekordok hozzávetőleges mennyiségéről stb. Így elérhető, hogy egy későbbi lépésben, célszerűen legyen meghatározva az adatok tényleges tárolási módja, az, hogy az adott ABKR lehetőségeit figyelembe véve, hogyan valósíthatók meg a set-ek stb. Ezek a tényezők igen lényegesen befolyásolják az adatbázis működési hatékonyságát. Ez a hatékonyság megalapozottan csak úgy tervezhető, ha a funkcionális elemzést elvégezzük.

Valamennyi funkció elemzése igen hosszadalmas feladat. Az első lépés ezért az, hogy régebbi felmérések, illetve gyakorlott helyi dolgozók segítsé-

gével előfordulási gyakoriságuk szerint kell csoportosítani a funkciókat, és a továbbiakban csak a tömegszerűségük, fontosságuk miatt mértékadókkal kell foglalkozni. A kiválasztást a következő szempontok befolyásolják:

- a funkció végrehajtásakor fellépő problémák,
- a funkció jellege, súlya,
- az adatigények tisztázottsága.

A funkció jellegén azt értjük, hogy mennyiben tekinthető az adatbázis használata szempontjából fontosnak egy adott funkció. Arra ugyanis általában nincs sem elegendő energia, sem idő, hogy valamennyi funkció elemzése megtörténjen. A fontosságot a gyakoriság, a mozgatótt adattömeg, a választási idő együttesen határozzák meg. A legfontosabbak azok a funkciók, amelyek nélkül az adatbázis nem tartalmazhat helyes adatokat. Ezek az adatbázis-karbantartó funkciók, amelyeket tehát teljeskörűen elemeznünk kell. A további funkciók kiválasztására nézve már csak irányelveket lehet adni. Természetes, hogy a sűrűn gyakorlandó funkciók végrehajtására nézve az adatbázisnak lehetőleg hatékony működésűnek kell lennie; ezért célszerű elemzésre választani a gyakran ismétlődőket. Ha valamely funkció végrehajtása nagy adattömeg felhasználását igényli az adatbázisból, akkor a hatékonyság még a funkció viszonylag ritkább előfordulása esetén is lényeges. Ezért ezek a funkciók is elemzésre jelölendők. Az adatbázis párbeszédészemmmódú használata az egyik olyan helyzet, amikor a várakozási idő csökkenése érdekében lényeges a gyors válasz. Ha tehát a funkció gyors választ igényel, akkor különös gonddal kell elemezni. Ettől teljesen eltérő okból — a feldolgozás hatékonysága érdekében — de szintén lényeges a feldolgozás minél gyorsabb végrehajtása, ha olyan nagy tömegű gyűjtött anyag érkezik be, amivel az adatbázist karban kell tartani, majd szoros határidővel kell sok tablót készítenünk az így karbantartott adatbázis adataiból. Ilyenek például a negyedéves, féléves, éves statisztikai célú kötegelt feldolgozások. Emellett a navigálás legmegfelelőbb módjának megtalálása érdekében fontosak azok a kötegelt feldolgozások is, amelyek az adatmodell túlnyomó (vagy jelentős) részét érintik.

Tapasztalatok szerint egy átlagos szervezet esetében az összes folyamat mintegy negyedének elemzése elegendő az adatbázis tervezésekor. Természetesen az ügýtípusok ellátásához szükséges felhasználói (lekérdező) programok specifikálása a többi ügýtípus elemzését is szükségessé teszi, azonban ezt az adatbázis sémájának véglegesítését követően is el lehet végezni.

Szólni kell még az adatigények tisztázottságáról. Bármely információrendszer fejlesztésének, az adatfeldolgozás megtervezésének fontos része, hogy az eredményadatokat felhasználók egyértelműen közöljék, milyen tartalmú és rendezettségű tablókat milyen periodicitással igényelnek. Az adatigények specifikálása olyan munka, amely nyilvánvalóan a felhasználó (megrendelő) feladata is, hiszen feltehetően ő tudja legjobban megmondani, pontosan mire van szüksége. A gyakorlatban azonban ennek kiderítése javarészt a megbízott szervező dolga. Az adatigények meghatározása kulcsfontosságú, de egzakt módszerek hiányában igen nehezen elvégezhető szervezési feladat.

Már a helyzetfelmérés során fel szokott vetődni az igény olyan információk szolgáltatására, amelyeket a meglévő manuális vagy nem adatbázison alapuló rendszer eddig nem tudott nyújtani. Az ilyen igények kielégítésének két fontos adatfeldolgozási vetülete van:

- az érintett adattartalom megléte, azaz a kívánt új információ szolgáltatásához szükséges adatoknak szerepelniük kell az adatbázis adatai között. Ez ellenőrizhető adatmodellen (Bachman-diagram), majd szükség esetén a kiegészítés megtehető;
- teljes mértékben tisztázott kell, hogy legyen a kívánt információ előállítási algoritmus, illetve az ehhez szükséges felhasználói munkafolyamat. Ide kell érteni a szükséges adatok adatbázisba kerülésének folyamatát is.

4.1. A funkcionális elemzést megelőző helyzetfelmérés

A funkcionális elemzésre kiszemelt tevékenységek körében szét kell választani a szinkronizált, illetőleg az aszinkronizált jellegű tevékenységet. A szinkronizált jellegű tevékenység a kötegelt feldolgozásnak azt a változatát jelenti, amikor az adatbázis tartalmából különféle szempontok szerint kivonatokat készítenek statisztikák, beszámoló jelentések, kimutatások stb. céljából. Végresultátusban itt nem tranzakciók összegyűjtéséről és egy kötegben történő feldolgozásáról van szó, hanem az adatbázis tartalmának különféle szempontok szerinti lekérdezéséről és általában táblázatba foglalásáról. Az aszinkronizált jellegű tevékenység együtt jár az adatbázis tartalmának módosításával, illetve törlésével, vagy új adatok bevitelével. Ez történhet akár kötegelt, akár

tranzakció-orientált módon. Bár mindkét esetben az adatok használatának módjára vagyunk kíváncsiak, a felmérést mégis eltérően kell végrehajtani.

4.1.1. Aszinkron (reakció) jellegű tevékenységek felmérése

A felmérés rendje ebben az esetben a következő:

- meg kell keresni az indító eseményt,
- össze kell gyűjteni a reakciót jelentő adatkezelési funkciókat és fel kell tárni ezek egymás közötti összefüggését;
- meg kell állapítani, hogy a feltárt funkciók milyen elemi adatokat igényelnek, illetve milyen elemi adatokat állítanak elő,
- a felmérés eredményét írásban össze kell foglalni (ez kiegészülhet a vonatkozó jogszabályok, szabályzatok stb. szövegrészleteivel, illetve döntési táblákkal, továbbá az input-ouput bizonylatokkal, kitöltési utasításokkal; 30. táblázat).

Ezután meg kell vizsgálni, hogy előállítható-e a szükséges információ a már meglévő adatmodellből, illetőleg beépíthető-e abba a termelődő adatok, átvezethető-e a módosítások (26. és 27. táblázatok).

Ehhez szükség van mind az adatmodell Bachman-diagramjára, mind pedig az abban szereplő egyed, illetve tulajdonságtípusok leírására. Ezek segítségével kétféle dokumentum készíthető:

- navigációs táblázat (26. táblázat) és
- a funkció elérési útjának a leírása (27. táblázat).

A navigációs ábra azt szemlélteti, hogy a kívánt adatok elérése vagy új adatok helyének megkeresése érdekében hol kell majd belépni az adatbázisba, hogyan kell abban a kapcsolatokon áthaladni és hol kell abból kilépni. Az egész folyamatot az adatmodellben történő navigációnak nevezik. Ehhez kiindulásnak legcélszerűbben az adatmodell Bachman-diagramja használható. Annak érdekében, hogy a navigációs út jól megkülönböztethető legyen a Bachman-diagramban feltüntetett egyed típusok közötti kapcsolatoktól, szaggatott nyilakkal jelöljük azt, külön jelölve a belépés és a kilépés helyét. A belépési pontot az ún. vezérlőinformáció határozza meg. Megjegyezzük, hogy — elsősorban az adatbázis kötegetelt üzemmódú használatakor tapasztalhatóan — ugyanazon adatkezelési funkcióhoz a gyakorlatban többféle vezérlőinformáció is tartozhat, s ettől függően viszont többféle belépési pont, illetve navigációs út létezhet, melyek mindegyike ugyanazt az

adatvisszakeresési igényt elégíti ki. Ilyenkor dönteni kell, hogy melyik tulajdonságtípust tekintjük vezérlőinformációnak.

Sok esetben korántsem biztos, hogy a kapott egyedtípus valamennyi előfordulását el kell érni. Az éppen kívánt egyedelőfordulások kiválasztását eldöntendő feltételt másodlagos ismervnek nevezik. A navigáció leírásakor meg kell adni a kiválasztás másodlagos ismervét is (ha van ilyen). Az sem mellékes, hogy az egyedelőfordulások elérése milyen jellegű: egyedi, teljes vagy ciklikus (ismételt).

Végül érdekes még az, hogy mi az adatkezelés célja az egyes elemi adatok vonatkozásában: visszakeresés, módosítás vagy törlés. Mindezeket az egy adatkezelési funkció igényeit leíró információkat az egyedtípusok elérésének sorrendjében rögzítjük (27. táblázat). Az egyedtípusra irányuló hozzáférési igényeket összesítjük (28. táblázat). A funkcionális elemzés során általában kiderül, hogy egyes elemek kimaradtak a modellből, amelyet ezért ki kell egészíteni (esetleg új kapcsolattípussal vagy egyedtípussal és kapcsolataival is). Biztosítani kell, hogy az így kiegészített elvi modell normalizált legyen. E korrekciók maguk után vonhatják egyes navigációs utak változását, így a funkcionális elemzés dokumentumait is újra át kell nézni. Ez az a helyzet, amikor a logikai tervezéshez az összegyűjtött információk megfelelően kiértékelhetők.

4.1.2. Szinkron jellegű tevékenységek felmérése

Ezeknek a funkcióknak az esetében végső célunk azonos az előző pontban leírtakkal, azonban az elemzés kezdeti szakasza eltér. Nem arról van ugyanis szó, hogy egy esemény tevékenységsorozatot vált ki, hanem egy ismert — vagy annak feltételezett — táblaszerkezetet kell összehasonlítani az adatmodell szerkezetével, és megvizsgálunk az előállítás módját. Ennek megfelelően, az előző pontban említett dokumentációt megelőzően még egy további bizonylatot (30. táblázat) kell előállítani, amely a tábla szerkezetét bontja ki, alkalmassá téve azt a navigáció világos áttekintésére, illetve előkészíti a navigációs ábra megalkotását. Ezen a táblaszerkezetlapon kezdődik az outputok elemzése, ezért szükséges néhány fogalom meghatározása, illetve értelmezése.

A tulajdonságtípusokat az outputokon első közelítésben három csoportba soroljuk:

- fejadatok,
- tételadatok,
- záróadatok.

Minthogy az egyes outputok érdemi részét a tételadatok alkotják és elsősorban ezek származnak az adatbázisból, ebben a leírásban ezekre szorítkozunk. Adatbázisban tárolt adat azonban a fejadatok (a címbe) és a záróadatok között is megjelenhet. Ebben az esetben az adat mint az output által tartalmazott összes tétel legfelső csoportosító ismérve, az elemzésben is megjelenik.

Egy táblázat címe pl. általában megnevezi azt a sokaságot, amelyet a táblázat osztályoz, és lehetőleg felsorolja a csoportosító ismérveket. A fejes oldalrovatok a megfelelő sorok és oszlopok elnevezéseit tartalmazzák. Ha a táblázat osztályozást is foglal magában, a fej-, illetve oldalrovatokban az osztályozás alapjául szolgáló ismerv változatai állnak.

A tételadatok az output logikai felépítésében betöltött szerepük szerint két fő csoportba sorolhatók:

- mennyiségi jellemzők (ismérvek),
- minőségi (vagy más szóval csoportosító) jellemzők (minőségi ismérvek)

A mennyiségi ismerv szerinti megfigyelés eredményei — a csoportosító ismérvektől eltérően — kétféle jelleggel is feldolgozhatók:

- osztályozható a sokaság (ez a csoportosító ismérvnél is megtehető),
- a mennyiségi ismerv értékei összesíthetők a sokaság egészére vonatkozóan (ez a csoportosító ismerv változataira nem tehető meg).

Az első esetben gyakorisági sort kapunk, a második esetben egy összefoglaló számszerű értéket, a mennyiségi ismerv összes előforduló értékeinek összegét. A kétféle jellegű feldolgozás kombinálható is egymással, vagyis az értékösszegek nemcsak a sokaság részére, hanem az osztályozás eredményeképpen kapott csoportokra vonatkozóan is elkészíthető. Ez az eljárás a mennyiségi soroknak a gyakorisági soroktól különböző új típusához, az értékösszegsorhoz vezet. 30. táblázat négy változó esetén az összes lehetséges összegfokozatokat illusztrálja. Az értékösszegsor a vizsgált mennyiségi ismerv értékeinek az egyes osztályokon belüli összegeit tartalmazza (összegfokozatok).

Attól függően, hogy az outputon szereplő mennyiségi jellemzők egy vagy több rekordtípusból származnak, a táblázat tartalmazhat egy vagy több tételtípust. Az azonos tételtípushoz tartozó jellemzők tovább csoportosítha-

tók annak megfelelően, hogy ezek mint mennyiségi jellemzők, az adatbázis ugyanazon rekordtípusa által leírt egyed jellemzői, vagy ezzel a rekordtípussal közvetlenül vagy közvetve kapcsolatban lévő rekordtípus által leírt egyedek jellemzői. A továbbiakban az előbbieket elsődleges, míg az utóbbiakat másodlagos jellemzőknek nevezzük.

Az egyes outputok tételei összefüggésükben hierarchiát alkotnak, amely a tételek rendezettségében jut kifejezésre. Egy tétel a hierarchia legfelső szintjétől a legalsóig vezető valamely úton fekvő csomópontok sorozatából épül fel. A hierarchia egyes szintjei a tételtípus meghatározott jellemzőivel reprezentálhatók. Ezek a tételtípus hierarchiaképző jellemzői. Hierarchiaképző lesz a tételtípus valamennyi elsődleges jellemzője. Másodlagos jellemző akkor válik hierarchiaképzővé, ha az őt tartalmazó rekordtípus azonosítója mint elsődleges jellemző, nem szerepel a tételben. Az egy hierarchiaszinten lévő csomópontok a szintet jelölő jellemző lehetséges értékei. Egyetlen tétel tehát az outputon az értékadatok által meghatározott rekordtípus azon előfordulásainak összesített értékadatait tartalmazza, amelyekhez tartozó hierarchiaképző jellemzők mindegyike azonos értéket vesz fel.

5. fejezet

Logikai adatmodellezés

A funkcionális elemzés eredményei alapján módosulhat az eredeti adatmodell. Kiderülhet, hogy bizonyos adatokkal a modellt rövidíteni kell, esetleg bebizonyosodik, hogy olyan adatokat is felvettünk a modellbe, amelyeket egyetlen funkció sem használ. Az is lehetséges, hogy a kapcsolatok módosítása válik szükségessé. Az így elemzett elvi adatmodell funkcióként érintett részei, azaz almodelljei az ún. információszerkezetek. Ez az elnevezés is jelzi, hogy az információtartalom alapuló tervezés szakasza lezárult, s ezután a tartalmat figyelembe venni nem tudó módszereket használunk majd fel, egy — most már az információtartalom szempontjából megfelelő — modellből továbblépve. Mindaddig ugyanis az adatok kezelési módjától, a felhasználó ABKR-től teljesen függetlenül végeztük az elemzést. Most ezen túl kell lépnünk, ezeket a tényezőket is figyelembe kell vennünk, és elő kell állítani a fenti tényezőket tükröző ún. logikai adatmodellt.

A logikai adatmodell adatszerkezetekből áll; ezek összessége a séma.

Az információszerkezeti elemek és az adatszerkezeti elemek között az alábbi megfeleltetést célszerű létrehozni:

- 1 tulajdonságtípus — 1 mező (elemi adat, field adattétel)
- 1 egyed típus — szegmens(ek) (csoport, group, részrekord)
- 1 kapcsolat (előfordulások és típusok között is) — 1 szegmensreláció (set, adategyüttes)
- 1 adat-almodell — adatmondat-reláció (alséma, subschema)
- 1 adatmodell — adatbázis (séma)

5.1. Adattétel (elemi adat, mező)

Adattétel-szintű adatstruktúra fajtái: mezőérték, mező. Ez az adatstruktúrák legalsó szintű, legelemibb szerkezeti egysége, mely a tulajdonságértéket képviseli. Más adatszerkezeti egységtől az különbözteti meg, hogy csak

az adattételnek lehet önálló (független) értékattribútuma. A hierarchikusan egymás fölé rendelt adatszerkezeti egységek értékattribútumát ugyanis mindig meghatározzák a legalsó szintű adattételek értékei. Az adattétel konkrét értékét formailag egy adott jelkészlet meghatározott szintaktikai szabályok szerint összerendelt jeleinek halmaza (esetleg csupán egyetlen jel) adja meg.

Az értékattribútumon kívül az adattételek egy rendezett halmaza — mely egy tulajdonságtípus értékeit képviseli — további attribútumokkal is jellemezhető. Ezeket séma-attribútumoknak is nevezzük. Ilyenek a név, az értékosztály és az ismétlési attribútumok. Attól függően, hogy a képviselt tulajdonságtípus közvetlenül egyedelőfordulást azonosító funkciót lát-e el, vagy csak (potenciális) osztályozási (asszociatív csoportosítási), esetleg csupán értékhordozó (leíró) funkciót ellátó szerepkörben van, elsődlegesen vagy másodlagos kulcsmezőnek nevezzük el.

5.2. Szegmens (csoport, group, részrekord) adatstruktúra

Előre kell bocsátani, hogy szegmensen a CODASYL DBTG group fogalmát értjük. A CODASYL által használt group (csoport) elnevezés helyett azért alkalmazzuk itt az egyébként elterjedt szegmens kifejezést, mert a későbbiek során az adattételek és a szegmensok csoportosításáról, osztályozásáról lesz szó, és az ott használt matematikai statisztikai fogalomkörrel való interférienciát el kívánjuk kerülni.

Szegmens (group) a logikai szempontból összetartozó, mindig egyazon egyed típusra vonatkozó tulajdonságtípusokat képviselő mezők névvel (szegmens azonosítóval) ellátott csoportja. Egy szegmens előfordulásainak halmazát — melyet ugyanazon név azonosít — szegmenstípusnak hívjuk.

Szegmensszintű adatstruktúra fajtái:

- egyszerű szegmens,
- összetett szegmens,
- szegmensreláció.

Ha a szegmens csak adattételekből áll, akkor egyszerű, ha pedig adattételekből és szegmenskből (hierarchikusan egymás fölé és alárendelt csoportokból) is vegyesen áll össze, akkor összetett szegmensnek nevezzük. Egy egyszerű szegmensből úgy válik összetett szegmens, hogy az előbbiben több

olyan mező is előfordulhat, amelyek az egyedtípust jellemző tulajdonságtípusokon belül további logikai egységet képeznek. Ezek az alárendelt csoportokkal leképezett logikai egységek abban különböznek az egyszerű szegmenstől, hogy nem önállóak, és jelentőségük csupán egy összetett szegmens további mezőivel összefüggésben van. Annak is fennáll a lehetősége, hogy egy csoport további alárendelt csoporto(ka)t tartalmazzon. Az előbbi esetet egy egyszintű (14. ábra), míg az utóbbi esetet egy többszintű fagráf (15. ábra) illusztrálhatja. Az összetett szegmenstípusban szereplő minden alárendelt csoporttípust egy ún. ismétlési szám (attributum) is jellemez.

Egy összetett szegmens minden egyes előfordulásának legalább egy olyan tulajdonságtípust képviselő mezőt kell tartalmaznia, amely annak az egyedtípusnak az azonosítója, amelynek részét vagy esetleg egészét az összetett szegmenstípus kifejezi. Az elvi adatmodellből a logikai adatmodell kialakítása során egy adott egyedtípust legalább egy (de esetleg több) szegmenstípusba képezhetünk le. Egy szegmenstípusba sohasem kerülhet több egyedtípust jellemző tulajdonságtípus (az ún. gyenge logikai redundanciát hordozó kapcsoló tulajdonságtípusoktól eltekintve). A lekérés módja befolyásolja a logikai adatmodell kialakítását, végső fokon pedig az adatbázis működési hatékonyságát.

5.3. Szegmensreláció (group relation, chain, set, adatgyűjtés)

A szegmensreláció szegmenstípusok, illetve szegmenselőfordulások közötti kapcsolattípust képvisel. A szegmensek közötti reláción belül megkülönböztetünk egy ún. tulajdonos (owner, parent, master) és a tőle függő ún. tag (member, child, detail, dependent) -szegmenst. A kapcsolatokat Bachman-diagramon ábrázolhatjuk.

A szegmensrelációk lehetnek:

- egy szegmenstípuson belül a két eltérő szerepben (tag és tulajdonos) előforduló szegmenselőfordulások közötti szegmensreláció (16. ábra),
- egy tag- és egy tulajdonosszegmens-típus közötti kapcsolattípusnak megfelelő szegmensreláció (17. ábra),
- több tag- és egy tulajdonosszegmens-típus közötti kapcsolattípusoknak megfelelő szegmensreláció (18. ábra).

A tulajdonosszegmens-típus egy előfordulásához egyazon tagszegmens-típusnak legalább egy (de általában több) előfordulása tartozik. A szegmenselőfordulások között fennálló kapcsolat típusa lehet hierarchikus (ha egy tagszegmens-előfordulás nem tartozik több tulajdonosszegmens-előforduláshoz) vagy pedig lehet nem hierarchikus (általában a hierarchikusnál bonyolultabb). Az előbbi esetben egy tagszegmens előfordulás több tulajdonosszegmens-előfordulására fennáll az, hogy csak egyetlen úton érhető el, míg az utóbbi esetben néhány (vagy sok) tagelőfordulás több úton is elérhető.

5.4. Adatmondat (logikai rekord, entry, mondat)

Adatmondat-szintű adatszerkezet fajtái: adatmondat, adatmondat reláció, file. Az adatmondat adott felhasználói kör információigényének kielégítése céljából együvé csoportosított és relációkkal összekapcsolt szegmensek együttese. Adott felhasználói körön célszerű az adatbázist használó/kezelő programok azonos csoportját érteni, amelyeknek adatigényét zömmel ugyanazok a szegmenstípusok elégítik ki. Az adatigény tartalmában és a kívánt elérési sorrend tekintetében e szegmenstípus természetesen csoporton belül is némileg eltérő. Mégis, célszerű rögzíteni a fenti csoportokra vonatkozó ún. mértékadó adatmondatok tartalmát és szerkezetét, hiszen ezeknek megfelelően jelöljük ki az adatbázis fizikai megvalósításához nélkülözhetetlen adatállományok tartalmát és szerkezetét.

Az adatmondatokat elsősorban belső szerkezetük szerint célszerű megkülönböztetni. Ez a szerkezet lehet

- csak egyetlen szegmenstípusra vonatkozó:

szegmenscsoport, ami egyetlen, összetett szegmenstípust jelöl, azaz egy belső többszintű hierarchikus fászerkezet (19. ábra),

- több egyedtípusra is vonatkozó:

szegmenstípusok között fennálló, kizárólag hierarchikus relációkkal létrehozott fagráf szerkezet (20. ábra). A fagráf gyökerét alkotó szegmenscsoport (root, gyökérszegmens vezérlő információt tartalmazó entry-defining group, kopfsegment, fejszegmens) nem lehet más szegmenscsoport része vagy alárendelt szegmenscsoport; szegmenscsoportok között fennálló hierarchikus és

nem hierarchikus szegmensrelációkkal létrehozott hálós szerkezet (22. ábra). Ez kétféleképpen is keletkezhet:

- a szegmenstípusok szintjén: létezik olyan tagszegmens-csoport, amelynek több tulajdonosságsegmens-csoportja van (21. ábra),
- a szegmenselőfordulások szintjén: van olyan tagszegmens-típus, amelynek előfordulásait tekintve egy tagelőforduláshoz a tulajdonosságsegmens-típus több előfordulása tartozik (23. ábra).

A vezérlőinformációt tartalmazó szegmenscsoport lehet más szegmenscsoport része vagy alárendelt szegmenscsoport is. Meg kell jegyezni, hogy míg a fagrafszerkezet esetén a fennálló szegmensrelációk saját névattributummal való jellemzésére nincs szükség, addig a hálós szerkezet esetén a nem hierarchikus szegmensrelációkat saját névattributummal kell ellátni (24. ábra).

5.5. Adatmondat-reláció

Az adatmondat lehet (amint arra az előző pontban már utaltunk) vagy egy egyedtípusra vonatkozó szegmenscsoport szegmensreláció híján, vagy több egyedtípusra vonatkozó szegmenscsoportok szegmensrelációkkal stukturált összessége. Az utóbbi esetben a szegmenscsoportok közötti elérési kapcsolatok létrehozása érdekében még szükség van adatmondatok közötti relációkra is. Lényegében ugyanolyan hálós struktúra jön létre, mint ami a szegmensrelációkkal is előállítható lenne (25. ábra).

A szegmensrelációhoz képest azonban elvi különbség az, hogy az adatmondat-reláció két szegmenselőfordulás vagy két szegmenscsoport között nem közvetlenül, hanem — ha az adatmondat belső szerkezete hierarchikus vagy hálós — mindig a vezérlőinformációt tartalmazó szegmenscsoporton keresztül valósul meg. Ezek a relációk jellegzetesen nem hierarchikus kapcsolatok, hanem hozzáférési sorozatok (26. ábra).

Az, hogy a fizikai megvalósítás szempontjából a relációval összekapcsolt adatmondatok egy vagy több különböző file-ban lesznek-e tárolva, eleinte még közömbös. A fontos az, hogy a hozzáférési modell kialakításához az adatbázist kezelő egyes funkciók hozzáférési sorrendjeit követő útvonalaikat, amelyek az adatbázis egyes adatmondatát „pókhálószerűen” összekötik, rekordkapcsolatokkal kijelöljük.

5.6. File (adatállomány)

A file-okat elsősorban szerkezetük szerint célszerű megkülönböztetni:

- Nem kapcsolódó file (unrelated file),
- Kapcsolódó file (related file)

A nem kapcsolódó file-okban a hozzáférés szempontjából egy rekordtípus egyes rekord-előfordulásai állnak egymással kapcsolatban. (Ez mindig a file-on belül valamilyen rendezettséget, indexelést jelent). Amennyiben a rekordokra éppen belső hierarchikus vagy hálós szerkezet lenne jellemző, akkor ezen rekordtípus előfordulásai közötti kapcsolat mindig a vezérlőinformációt tartalmazó szegmenscsoporton kívül valósul meg. A kapcsolódó file-okban a hozzáférés szempontjából több, különböző rekordtípusnak egyes rekordelőfordulásai állanak egymással nem hierarchikus kapcsolatban, a vezérlőinformációt tartalmazó szegmenscsoportjaikon keresztül.

6. fejezet

Adatbázis-hozzáférés

Az adatbázis az ABKR által rendszeresen karbantartott file-ok összessége, amely megfelelő adatszerkezetek segítségével alkalmas az adatmodell információkapcsolatainak kellő mélységű leképezésére és így több felhasználói kör információigényeinek rugalmas kielégítésére. A felhasználó az ABKR segítségével a gépen tárolt adatszerkezethez több „lépésben” férhet hozzá. Ennek megfelelően, az adatszerkezeteket meg kell különböztetni a gépen ténylegesen megvalósított, fizikailag létező tárolt adatok ún. tárolási szerkezetétől. (A tárolási szerkezet: az adatok elhelyezésének rendje a tárolón. Erről az alkalmazói programozónak nem kell tudnia.)

Egy file adatszerkezetét az alkalmazói programozónak feltétlenül ismernie kell ahhoz, hogy a programnyelvben számára rendelkezésre álló eszközökkel fel tudja dolgozni a file-t. Például az alkalmazó egy egyedet egy logikai rekordnak feleltethet meg. Az elérési mód leképezi a tárolási szerkezetet az adatszerkezetre (és viszont), amikor a logikai rekordot elérni kívánó utasításra a szegmensekből összeállítja a két rekordot. Az alkalmazói programozó gyakran látja úgy, hogy az adatszerkezet egyszerűen szekvenciális vagy hierarchikus. Ez gyakran tévedés, és az adatszerkezet ennél lényegesen bonyolultabb, hálós szerkezet, ahol a hálóban egy adatszerkezeti elemnek (pl. egy rekordtípusnak) egynél több tulajdonosa lehet, míg a fatípusú hierarchiában csupán egy.

6.1. Hozzáférési modell

A hozzáférési modell egy olyan — általában gráf — modell, amely az adatbázis karbantartási és lekérdezési funkcióinak becsült gyakoriságára és tulajdonságtípus (részhalmazának) elérési igényeire támaszkodva lehetséges hozzáférési utakat jelöl ki. A tervezési adatok ehhez egyrészt a gráf-csomópontok nevei (amelyek tulajdonságtípusok halmazaként lehetnek vagy

az egyedtípusok vagy a szegmenstípusok nevei) alkotta halmaz, másrészt a lehetséges hozzáférési utak nevei alkotta halmaz, végül az egy-egy hozzáférési út mentén sorban érinteni szükséges csomópontok nevei, elérési sorba rendezve. A létező hozzáférési nyelvek általában vagy úgy érik el a keresett rekordokat mint attributumok halmazait, utána azonban az elérési műveletek útkövető információkat is magukban foglalnak, vagy úgy érik el a rekordokat, hogy pl. hierarchikus struktúrákat keresnek halmazorientált utasítások segítségével. A hozzáférési modell és a nyelv gráfelméleti fogalmakat használnak és paraméterei részhalmazok kompozíciójának, összekapcsolásának, rendezésének a specifikációi. A legjobb hozzáférési útstruktúra kiválasztása a lehetséges utak halmazából: az adatbázis majdani hatékony működésének az egyik legfontosabb tényezője.

Akár a szeriálisan vagy szekvenciálisan, akár a direkt vagy véletlen elérési móddal szemben felmerülhet, hogy egy rekordot tulajdonképpen nem is a cím alapján, hanem annak információtartalma szerint lenne jó elérni.

A jelenlegi háttértárolók szervezése azonban olyan, hogy erre közvetlenül nem ad módot. A probléma megoldását egy „illesztés” teszi lehetővé, amely az információstruktúrákat tárolóstruktúrákra képezi le. Ezt a többlépéses leképezési folyamatot file-szervezésnek is hívják.

A kívánt struktúra létrehozása érdekében az elvi adatmodellt alkotó információelemeket általában úgy kell (át)csoportosítani, hogy új csoportosítás esetén csökkenjen az adatbázis átlagos elérési ideje. A visszakeresési szempontok szerint csoportosított elemi információk, azok nevei és kapcsolati adatai gyakran hierarchikus elrendezésűek, ami eleve hatékony hozzáférést tesz lehetővé. A hozzáférést gyorsítja az is, ha a sorban egymás után elérni kívánt elemi információk (csoportjai) fizikailag is egymás mellett (vagy egymáshoz közel) helyezkednek el a háttértárolón.

Az adatbázistervezés egy fontos problémája: hogyan és hány szinten csoportosítják a tulajdonságtípusokat úgy, hogy az az adatbázis teljesítőképességét minél magasabb szinten tartsa.

Hosszabb távon az egyik legfontosabb szempont az, hogy az egyszer már létrehozott csoportosítás olyan legyen, hogy az adatmodell szerkezetének megfelelő (amennyire egyáltalán előrelátható) későbbi változásai ne rontsák el az adatbázisban már kialakított csoportosítást és a felépített kapcsolatokat, továbbá ne kerüljön sor az alkalmazói programok átírására.

A csoportosítás lehetőségeit legalább három szinten érdemes alaposabban megnézni:

- tulajdonságtípusok egyedtypussokká való csoportosítása az elvi adatmodellezéskor.

Erre jól kidolgozott módszer a normalizálás, amely ezt az ún. harmadik (illetve negyedik) normálformára alakítással éri el. Mivel ez a módszer általánosan ismert, ezzel e könyv nem is foglalkozik. Normalizált elvi modellt mind a CODASYL-típusú (hálós), mind a relációs ABKR-ek további átalakítás nélkül is kezelni tudnak. Ez azt jelenti, hogy az eddigi egyedtypusok közvetlenül rekordtypusokká, a kapcsolattypusok pedig set-ekké válnak. Mégis, a hatékony működés érdekében módosításokat célszerű elvégeznünk az egyedtypusok megosztása vagy összevonása révén;

- egyedtypusok szegmenstypusba való tördelése a logikai adatmodellezéskor.

Erre a feladatra klaszteráló algoritmusok jól alkalmazhatók. Mivel ezek adatbázisstervezésre való adaptálása még kevésbé ismert, ezért ezzel e könyv 2. részében részletesebben foglalkozunk. A megosztás hatékonysági szempontból akkor indokolt, ha vannak az egyedtypust alkotó tulajdonságtypusoknak olyan csoportjai, amelyekre a sűrű együttes kezelés jellemző. Az összevonást ugyanazon egyedtypusok gyakori együttes elérése iránti igény indokolhatja. Mindezek következtében egyes érintett kapcsolattypusok megbontása, illetve összevonása is szükségessé válhat, s a változások kihatnak a funkcionális elemzéseknél megállapított navigáció módjára is;

- szegmenstypusok areakba való csoportosítása a fizikai adatmodellezéskor.

A klaszteráló módszerek erre szintén alkalmasak. A hozzáférési modell szerepe a hatékony működést biztosító kedvező csoportosítás globális elemzése és meghatározása (akár az elvi adatmodellezéskor már kialakított egyedtypusok szegmenstypusokká széttagolása, akár összevonása árán is). Segítségével főleg az indexek, az elhelyezés, a nem közvetlen vagy közvetlen elérés kérdései dönthetők el. Általában, file-szervezésre vonatkozó döntéseket célszerűen az adatmodellből származtatott adatstruktúrák és az azok közötti egyirányú utakat (közvetlen elérést) leíró hozzáférési modell alapján lehet hozni.

Általános hozzáférési modellt legalább kétféle alapon lehet megszerkeszteni. Az egyik lehetőség az, hogy a modellben néhány (pl. háromféle) hozzáférési útfajtát engedünk meg, némileg eltérő paraméterekkel jellemezve azokat. A másik lehetőség az, hogy egyetlen útfajta szerepelhet csupán, az esetleg fellépő paraméterek széles választékával.

Igen fontos, hogy a hozzáférési modell adatstruktúrái mennyire érzékenyek a külső változások okozta hatásokra. Az adatbázist működtető és felhasználó intézmény belső szervezeti változásaitól és a külső környezet (és szabályozás) változásaitól eltekintve, csupán az adatbázist hordozó hardware és software összetevőinek változásai például hosszabb távon azt eredményezik, hogy még akkor is, ha az adatmodellben nincs változás, sok régi programot újra meg kell írni. Ez az improduktív ráfordítás igen tetemes lehet. A külső környezet változásainak hátrányait nagyrészt az adatbázist kezelő software célszerű kialakításával, illetve az adatbázis előrelátó megtervezésével is csökkenteni lehet.

Jelenleg bármely ABKR-től elvárhatók az alábbiak:

- az általa kezelt információ integrálásának és konzisztenciájának egyszerű karbantartása,
- útmutatás arra nézve, hogy az adott információ a rendszerben hol van tárolva,
- adatfüggetlenség, azaz a karbantartó és lekérdező programokat a változó környezet ellenére se kelljen gyakran újraírni.

Az adatfüggetlenség viszonylag jobb akkor, ha a programok olyan névstruktúrákkal címezik meg az adatbázisban tárolt információt, amelyek az időben alig vagy csak kevésbé változnak, és megfordítva, viszonylag rosszabb az adatfüggetlenség akkor, ha a fent említett névstruktúrák az időben meg lehetőségen gyakran változnak. Az ABKR típusától függetlenül, általában az utóbbi eset áll fenn.

A fenti ABKR elvárások a felhasználó oldaláról általában két követelményként fogalmazódnak meg:

- egy információt csak egyszer (egyetlen helyen) kell tárolni,
- az adatbázist használó programok egy viszonylag stabil névstruktúrát (azaz információstruktúrát) használnak.

6.2. Adatbázis-kezelés

Egy file-kezelő rendszer része lehet az ABKR-nek, hiszen az ABKR-en belül léteznie kell egy olyan funkciónak, amely a file-szerkezeteket a tárolóterületre leképezi. Az ABKR-re jellemző hogy kezdetben a file-kezelő rendszerek egyidejűleg csak egy file-t kezeltek batch módon. Később több file együttes és gyors válaszú kezelését valósították meg. Ez még nem különböztetné meg őket az adatbázis-kezelő rendszerektől, de valóban vannak lényeges eltérések, mégpedig:

- adatbázis-kezelés esetén adatdefiníciók és az alkalmazói programok egymástól független, elkülönített módon szerepelnek;
- számos rekordtípus létezik, és ugyanazon rekordtípushoz általában számos (kettőnél több) hozzáférési út vezethet;
- a rekordok elhelyezése a tárolókon is független attól, hogy a rekordok melyik set-hez tartoznak;
- mivel egy tranzakció sok rekordtípust építhet, és így sok elérést igényel, a hozzáférési hatékonyság igen fontos az adatbázisok esetében.

6.2.1. Adatbázis-kezelő rendszerek

A felhasználó feladata a hagyományos file-ok használatakor az, hogy az (általában bonyolult) adatszerkezet ismeretében a rendelkezésre álló tároló-szerkezetek egymáshoz kapcsolásával saját maga valósítsa meg a leképezést. Ezzel szemben adatbázisoknál a felhasználó csupán az adatszerkezetet (a sémát és az alsémákat) írja le egyfajta formális nyelven (Data Definition Language DDL), a leképezést a tároló-szerkezetre az ABKR végzi el automatikusan. Az adatok felhasználásához pedig olyan másik (Data Manipulation Language DML) formális nyelv szükséges, amely az információszerkezet felhasználó általi kezeléséhez szükséges alkalmazói programok hatékony fejlesztését segíti.

Megkülönböztethetünk hierarchikus, hálós és relációs ABKR-eket.

Hierarchikus ABKR (pl. IBM rendszerei)

Az elérés legkisebb egysége a szegmenselőfordulás. Egy rekordtípuson belül nem lehet sem egyedi sem ciklusos sem teljes kezelést megoldani. Hátránya,

hogy a módosítás során anomáliák léphetnek fel. Megoldható viszont több szegmenstípus alkotta faszerkezet kezelése, emellett a hierarchia legfelső szintjén lévő rekordelőfordulás egyedi, ciklusos, illetve teljes kezelése, amely azt jelenti, hogy a hierarchiában felette álló szegmens nélkül nem érhető el a hierarchiában alsó szinten álló szegmens.

Hálós ABKR (ún. CODASYL-alapú alrendszerek)

Az elérés legkisebb egysége a rekordelőfordulás. A rekordelőfordulásokat teljes, egyedi és ciklusos kereséssel érhetjük el. A módosítás során nem lépnek fel anomáliák. Hátrányai: tudni kell az adatbázis létrehozásakor, hogy mely kapcsolatra lesz szükség; az adatkezelő nyelv némileg bonyolult. E két hátrányt csökkenti a relációs ABKR.

Relációs ABKR

Az elérés legkisebb egysége a reláció (vagy domain). Az adatkezelő nyelvnek olyannak kell lennie, hogy lehetőséget adjon az éppen elérni kívánt reláció meghatározására. Erre a relációkalkulus vagy a relációalgebra igen alkalmas.

6.2.2. A DDL és a DML

A DDL utasítások vagy egy magasszintű nyelv (pl. COBOL) adatleírása részeként szerepelnek vagy attól független, önálló nyelvet alkotnak. Az így leírt adatok kezelésére a DML adatkezelő nyelv utasításai szolgálnak. Ez az elkülönítés az egyik feltétele annak, hogy a csupán DML utasítások alkotta vagy egy magasszintű programnyelv utasításai közé illesztett DML utasításokból álló programok az adatszerkezetektől függetlenek, azaz „adatfüggetlenek” legyenek. Célszerű azt biztosítani, hogy az adatok fizikai tárolásának módosítása ne legyen hatással az alkalmazói programokra (azokat pedig ne korlátozza az adatok tényleges fizikai elrendezése), de emellett azt is meg kell oldani, hogy az adatokat az egyes alkalmazói programok az „általuk kívánt” szerkezetben „láthassák”, továbbá azt, hogy az alkalmazói programok logikai adatigényének (adatszerkezet-igényének) változása ne vonja maga után az újraszervezést. Ezért a DDL elkülönülten definiálja egyrészt az adatmodell egészének fizikai szerkezetét, másrészt az egyes adat-almodellek szerkezetét.

A séma adatstruktúra elemeit — a CODASYL ajánlásnak megfelelően — DDL nyelven írják le, az alkalmazói programoktól teljesen függetlenül. Az alkalmazói programban éppen dolgozó adatelérési utasítás hatására a

könyvtárból kiválasztásra kerül a programhoz tartozó alséma. Az alséma képviseli a program adatigényét, s ezt az ABKR hozzárendeli (ezt binding-nek nevezik) a sémához, amely az adatok fizikai tárolását írja elő. Ezután az operációs rendszer megfelelő adatelérési szintje az alkalmazói programhoz kijelölt pufferbe hozza az igényelt adatokat.

Összefoglalva, a DDL, arra való, hogy segítségével írjuk le az adatbázis létrehozásához szükséges sémát és az alkalmazói programhoz való kapcsolatot biztosító alsémákat, tehát a DDL-lel írják le az adatszerkezetet és az azok védelmére szolgáló jelzéseket. A DDL sémaleírás egyrészt az adatmodell egészét írja le, másrészt figyelembe veszi az alkalmazói programok elérési igényeit, végül illeszkedik a konkrét hardware és software tárolási és elérési lehetőségekhez. A DDL alséma-leírás pedig amellet, hogy a megfelelő adatmodellt leírja, meghatározza, hogy az alkalmazói programok (egyes csoportjai) milyen adatszerkezetet kezelhetnek.

A fentiekből is kitűnik, hogy a DDL sémadefiniáló része olyan fizikai struktúrát jelöl, amely nagymértékben függ az éppen alkalmazott tároló-eszköz paramétereitől. A nem csupán adat-, hanem berendezésfüggetlenség érdekében alakult ki egy harmadik önálló nyelv a tárolóvezérlő nyelv (DMCL). Ennek segítségével a tároló fizikai szintű kezelésére vonatkozó információk egy elkülönített helyre csoportosíthatók. A séma tehát úgy adható meg, hogy a DDL nem írja le a konkrét külső tárolókat és azok jellemzőit, de mégis elkerülhetetlen a fizikai tárolási és elérési mód (együttesen: az elhelyezési mód) szimbolikus (pl. ISAM, CALC stb.) megadása. Egy adott ABKR esetén egyébként csupán néhány, ún. elérési vagy elhelyezési mód közül lehet választani.

Az adatbázist egymásközt „megosztva” használja számos program, és egy adott program által érintett részadatbázis jól körülhatárolható. Alséma jelöli ki az adatmodell azon részét, amelyet egy program használ. Maga az alséma sorra azokat a rekordokat, mezőket, és set-eket írja le DDL nyelven, amelyeket el kíván érni, de egy idevágó DDL-leírás nem nevezi meg azokat a rekordokat és kapcsolatokat, amelyeken keresztül a program a DDL-ben kíván hozzáférni a felsorolt adatokhoz.

6.2.3. Az alséma definíciói

Az IMS kezdeti verziói az alsémát azon adatok szegmensszinten és programfüggetlen módon definiált csoportjának tekintették, amelyekre egy programnak szüksége lesz. Azt, hogy mire lesz szüksége, a PSB (Program Sensitivity Block)-ban adták meg úgy, hogy megjelölték az érintett logikai és fizikai adatbázis(oka)t, az elérendő (azaz szenzitív) szegmenstípusokat. A program egy beolvasó utasítását követően, puffterületeken már behozott szenzitív szegmens bármely mezője — olyanok is, amelyre az adott programnak semmi szüksége sincs — elérhető volt. Az eléérés és a védelem (röviden: a szenzitivitás) legkisebb egysége itt tehát a szegmenstípus. A CODASYL-alapú ABKR-ek alséma fogalma viszont szigorúan már csak az adott program működéséhez szükséges rekordokat és mezőket tartalmazza (és védi).

Minthogy az adatbáziskezelési technikával szemben alapkövetelmény, hogy minden felhasználói kört annak kívánalmai szerint elégítsen ki, ezért rendszerint több alséma is van, amelyek közt nagymértékű az átfedés. Éppen ezért a séma és az alsémák megfeleltetése egy $1 : N$ típusú hozzárendelési probléma, azzal bonyolítva, hogy az alsémák logikai szerkezete jelentősen eltérhet a sémabeli szerkezettől. A hozzárendelés alapját képező adatok rendszerint az alsémában fordulnak elő. Kivétel ez alól a relációs adatbázis, ahol a sémának megfelelő leírás az összes leképezéshez szükséges adatot tartalmazza. A sémának az a jelentősége, hogy az operációs rendszer és az ABKR, valamint valamennyi alkalmazói program (tehát az összes program) számára egyaránt közös és programtól független definíciót ad az adatmodellre. Kétségtelen azonban, hogy míg az operációs rendszer és az ABKR programjai előbb-utóbb az adatbázis minden részét megmozgatják és így a teljes sémára szükségük lesz, addig az alkalmazói programok csupán az adatbázis általuk kezelendő részében érintettek, így arról nem is „kell tudniuk”.

Az egy-egy alkalmazói program által használt DDL-nyelvű adatmodell-leírást alsémának nevezte el a CODASYL-jelentés, amely szerint az alséma csakis egyetlen sémához tartozhat, sőt, egy program csakis egyetlen alsémát használhat. Az IMS-II-nél viszont a sémának megfelel valamennyi fizikai és logikai adatbázist leíró DBD-k összessége, míg az alsémának nagyjából a PSB felel meg. A dBASE III PLUS-nál az alsémák megfelelője az ún. view-file, amellyel az egyszer előállított, összetett adatszerkezeteket ismételt felhasználásra gyorsan elő lehet hívni, mégpedig a beállított ún. filter-feltételeknek megfelelő szelektivitással. A fentiekből már érzékelhető, hogy a

DDL igen sokféle lehet. Bármely DDL alapja egyrészt néhány szemantikailag elvont fogalom, másrészt a fogalmakat szerkezetbe rendező, meghatározott számú szabály.

Alapvető, általánosan használt fogalmak: a tulajdonság, az egyed és a kapcsolat, de a modellezési módszertől függően vannak egyéb fogalmak is. Mivel a DDL alapvető fogalmai szemantikailag elvontak, és a szerkezetbe rendezés szabályai csupán ezekre, és ezek egymással alkotott kapcsolatára vonatkoznak a DDL nem a valóság eseményeit tükröző, hanem pusztán a benne megengedett, saját szerkezeti előírásait leíró szemantikájú nyelv. Nyilvánvaló, hogy az adatmodell teljes szemantikai tartalmát általában nem is képes mindig pontosan visszaadni. Például a CODASYL által javasolt DDL-ek nem teszik lehetővé, hogy egy ugyanazon set-típusba tartozó rekordtípus tulajdonos-rekordtípus is, meg tag-rekordtípus is lehessen. Ha például egy többszintű alrendszer-hierarchiát kívánnánk leírni egy ilyen DDL-lel, akkor a hierarchia alsóbb és felsőbb szintje közötti kapcsolat kifejezésére, más lehetőség híján, egy új rekordtípust kell kialakítanunk.

7. fejezet

Tárolási szerkezetek

Amikor egy adatszerkezetet a számítógépben tárolunk, akkor egyrészt tárolnunk kell az adatszerkezet elemeit, másrészt előre ismernünk, vagy ugyanott tárolnunk kell az elemet az adatszerkezetbe összefogó, az azok közötti kapcsolat felépítését lehetővé tevő információkat is.

Adatbáziskezelés esetén a kapcsolatok tényleges kialakítása a szegmensek, illetve a rekordok szintjén futási idő alatt történhet (erről a későbbiekben részletesebben lesz szó).

Egy adatszerkezetet fizikailag többféleképpen is tárolhatunk, de lényegében végül is két különböző lehetőségünk van (és e két lehetőséget a mágnesszalag, illetve a mágneslemez adottságai biztosítják):

- szekvenciális leképezés; ebben az esetben egy keresett adatszerkezetet csak a teljes file-t végigkeresve érhetjük el,
- nem szekvenciális leképezés (distributed mapping); ebben az esetben az egyszerű szegmensnél bonyolultabb adatszerkezetet (mint pl. a rekordot) úgy (direct, CALC, indexed) lehet elérni, hogy segédeszközként mutatókat (pointereket) is alkalmazni kell. A pointerrel, amely valamilyen (szimbolikus, fizikai, direkt, indirekt stb). címet tartalmaz, adatszerkezeti elemek közötti kapcsolatokat fejezhetünk ki.

Ezek alapvetően az alábbiak lehetnek:

- a tulajdonostól a tag felé mutató kapcsolat: „lefelé mutató” pointer,
- a tagtól a tulajdonos felé mutató kapcsolat: „felfelé mutató” pointer,
- az egyik tagtól a másik tag felé mutató kapcsolat: „oldalra mutató” pointer.

Hierarchikus adatszerkezet esetén egy rekordnak csupán egyetlen felfelé mutató, de több lefelé mutató pointere lehet. Hálós szerkezet esetén egy rekordnak több felfelé mutató pointere is lehet. Mind hierarchikus, mind hálós adatszerkezetnél előfordulhat, hogy egy rekordtípus bizonyos feltételek fennállása esetén tagja, míg a feltételek hiánya esetén nem tagja ugyanazon adategyüttesnek (set-nek). Az oldalra mutató pointerekkel a set tagjainak

sorrendje is kifejezhető. Erre a célra főleg pointer-tömböket (indextáblákat) vagy láncokat és gyűrűket, de esetleg bit-térképet alkalmaznak.

7.1. Fastruktúra leképzése tárolóra

A fa leképezésével (tárolásával) kapcsolatban igen sokféle algoritmus létezik, ilyen többsége felhasználja a fa bejárását, szögpontjainak szisztematikus vizsgálatát úgy, hogy minden szögpont pontosan egyszer kerüljön sorra. A fa bejárása a szögpontok lineáris rendszerét adja, amelynek értelmében beszélhetünk egy adott szögpontot követő vagy megelőző szögpontról. Így a fák leképezhetők soros szerkezetű adathordozóra is, ha az egyes szögpontokra vonatkozó mondatokat a gyökérből kiindulva, a felülről lefelé haladás sorrendjében tárolják. Az ilyen tárolás azonban azon előnye mellett, hogy mágnesszalagon is megvalósítható, azzal a jelentős hátránnyal jár, hogy az adatstruktúra változása esetén az állományt teljes egészében újra kell írni és ekkor túlcsoordulás is előfordulhat. Ha a struktúra csak ritkán módosul és a file biztonságos újraírása megoldható, ez a megoldás mégis beválhat. Jóval gyakoribb eset, hogy a file aktivitása nagy. A fastruktúra gyakori módosítása miatt ekkor célszerűen úgy tárolják az állományt, hogy a rekordok egymás közti logikai kapcsolatait, illetve a soros tárolásból adódó fizikai sorrendje teljesen elkülönüljenek.

Olyan tárolóeszköz és olyan fizikai file-szerkezet alkalmazása szükséges továbbá, amely ún. alapelérést (közvetlen elérést) biztosít.

A hierarchikus vagy hálós adatstruktúrák mátrix (tömb) formában való tárolása sparse-mátrix (azaz ritka mátrix) megvalósítását jelenti. A magasszintű programnyelveken rendelkezésre álló tömbdeklaráció itt kevés haszonnal jár, mert a tárterület jobb kihasználása végett a tömbből úgyis ajánlatos listát készíteni. Az így képzett listák ráadásul terjedelmesebbek, mint a tömb közbejötté nélkül készítették lennének. Előtérbe kerül tehát a fák eleve „faszerű”, a háló eleve „hálószerű” tárolása. Amennyiben a magasszintű nyelv lehetővé teszi struktúra deklarációját, a programozó ezt használja, amennyiben ez nem lehetséges, az adatstruktúra listaszerű tárolását célszerű megoldani.

A faszerkezetet láncolással (chaining) vagy pointertömbök (bundling) segítségével, illetve struktúrafile létrehozásával képezhetjük le. Fastruktú-

rák tárolására alkalmasak például a multilisták is, az ilyen tárolási mód azonban meglehetősen helyigényes, és karbantartása bonyolult. Alkalmazható pointer-tömbös megoldás is (29. ábra), melynek előnye, hogy a struktúra módosítása igen könnyű. Hátrány marad azonban, hogy a fa bejárása időigényes. A fa nagy részének vagy egészének lebontása mindig csak a gyökérből (illetve a tulajdonosból) kiindulva tehető meg, továbbá listaelemeinek hossza különböző lesz és ez is hátrányos lehet (42. ábra). Célszerűnek tűnik ezért, másik módszert választani: a fát bináris fává alakítva tárolni. A fastruktúra egyszerűen láncolt listával reprezentálható (30. ábra). Oly módon kell eljárni, hogy a listaelemek két-két pointert, ún. testvér- és fiúcímet tartalmazzanak.

A módszer előnye, hogy a listaelemek hossza azonos lesz. Azt a hátrányt azonban, hogy a lebontás csak felülről lefelé végezhető, ez a megoldás sem küszöböli ki. Ennek elkerülését csak külön struktúrafile(ok) létrehozása, és abban (szimmetrikus visszacsatolt) lista(k) alkalmazása teszi majd lehetővé.

Igen jól használható még a biráris fák visszacsatolásos elhelyezése. Az eddigi példaként használt biráris fa visszacsatolásos reprezentációja a (31. ábra), ahol, a $\Rightarrow$ nyíl tényleges előremutató, a $\rightarrow$ nyíl pedig visszacsatoló pointert jelöl. A két eltérő típusú pointer megkülönböztetésére elegendő minden szögpont kapcsolatmezőjéhez egy egybites jelző mező. A visszacsatolásos reprezentáció esetén megoldatlan még az üres bináris fa reprezentációja. A listák cirkuláris elhelyezéséhez hasonlóan, itt is célszerű bevezetni a lista fejét (43. ábra).

Nem üres bináris fa szögpontoknak a listafejtől balra való elhelyezésével jön létre. Az ilyen visszacsatolásos reprezentáció legnagyobb előnye, hogy a bináris fa bejárásának algoritmus a egyszerűsödik, a bejáráshoz nem szükséges segédverem használata.

A szemlélet alapján úgy tűnik, hogy visszacsatolásos reprezentáció esetén egy új szögpont elhelyezése a bináris fába bonyolult feladat; valójában azonban e feladat megoldására is egyszerű algoritmus adható.

Természetesen megadható a fák más belső reprezentációja is; általában igaz, hogy a célszerű reprezentációt az elvégzendő operációk típusa határozza meg.

7.2. Hálóstruktúra leképzése, tárolása

Egy információstruktúra, pl. egy gyártmány, beépülését (szerelését) kifejező családfa célszerűen leírható aciklikus digráf adatstruktúrával is. A struktúrát így leképezve, nem kell tárolni a kezdeti (fa) ábrázolásánál még meglévő redundáns adatokat, így tárolóhely és idő takarítható meg. Amennyiben az eredményadatokat megjelenítve az eredeti fastruktúrát vissza kívánjuk nyerni, a fa és az aciklikus digráf közötti egyértelmű megfelelés következtében a visszaalakítás elvégezhető, ha az aciklikus digráfot kifejező részlisták diszjunktak. A digráfok leírására nem csupán a két-, hanem az azzal egyenértékű „egydimenziós, stringszerű” forma is rendelkezésünkre áll. A digráfok ilyen alakja (számítógépes szóhasználat) általános listának, jegyzetnek vagy röviden LISTÁ-nak nevezhető.

Nyilvánvaló, hogy a hálóknak többféle belső reprezentációja lehet. A hálók egyszerű belső reprezentációja, a multilistaszerkezet, azonban csak ritkán használatos, mivel bármely LISTA első elemével kapcsolatos operációk (pl. az első elem törlése vagy új elem behelyezése a LISTA első elemeként) annyi kapcsolatmező tartalmának megváltoztatását kívánják, amennyire a LISTÁ-ra hivatkozások száma. Ezért szokásosabb minden LISTA definiáló előfordulásában a lista fejét feltüntetni, és amikor szükséges, arra hivatkozni, azaz a fej címét tárolni.

Egyedítípust képviselő ún. alapfile mellett csupán kiegészítő szerepet játszó komplex file-szerkezetet (az ún. multilistát) célszerű létrehozni (33. ábra).

Egy tulajdonságtípus valamennyi tulajdonságértékéhez egy-egy részlistát kialakítva (a lista elemeit pointerok kapcsolják össze), a listák fejeit egy indextáblázatba tömörítjük. Annyi indextáblázat készül, ahány másodlagos kulcsra akarjuk majd a file-t elérni. Mindez lehetővé teszi, hogy logikai rekordok ne csupán a tárolás alapjául szolgáló elsődleges kulcs, az azonosító alapján legyenek elérhetőek, hanem másodlagos kulcs, illetve több másodlagos kulcsérték alapján asszociatív visszakeresés is végezhető legyen. A keresett rekord a listán végighaladva választható ki és érhető el. Elvileg egy file-ban akárhány lista kialakítható, de bizonyos határon túl ezek kezelése túl bonyolult.

Az asszociatív visszakeresést azonban nem egy vagy több listát követve, hanem a kívánt rekordot közvetlenül elérve valósítja meg az ún. invertált listaszerkezet. Ez egy teljes indexelésű segédfile, amelyet az alapfile (amely

a multilista esetéhez hasonlóan itt is lehet indexelt, soros indexelt, szekvenciális, direkt vagy véletlen) mellett, annak hatékony kezelése érdekében hoznak létre. Ez a komplex file-szerkezet az alapfile-on kívül tartalmazza a valamennyi visszakeresési kulcs (tulajdonságtípus) szerinti teljes indextáblázatot, melyek viszont — az alapfile rekordjaival ellentétben — mutatókat is foglalnak magukba.

A több kulcsra invertálás tehát a kulcsok szerinti indextáblázatok létrehozását jelenti. Az inverz file olyan korlátozott multilista-szerkezet is lehet, amelyben a lista „hossza” 1. Az előző esetben teljes (többszörös) indexelésről, a második esetben láncolásról van szó.

7.3. Struktúraállományok

A stuktúraállományra akkor lehet szükség, ha egy rekordnak egynél több tulajdonosa van. Általában itt egy rekordelőfordulás egyszerre két láncban is szerepel, és míg a lánc fej (és láb) eleme(i) a tulajdonosok, addig a lánc többi eleme tagrekord. Mindez nem zárja ki, hogy fagráf adatszerkezet is tárolható legyen a struktúrafile-ben. A CODASYL-alapú ABKR-ekben a set-ek megvalósításához lánc vagy pointer-tömb (index) vagy ún. dinamikus pointer-tömb tárolószerkezetet alkalmaznak. Ez utóbbi azt jelenti, hogy a pointertömböt csupán a program futási idejére hozzák létre.

Nem mindig lehet, és főképpen nem mindig célszerű egy adatstruktúrát fizikai file-szerkezetre leképezni. Jelen esetben az indexszekvenciális file is egy fizikai file-szerkezetnek tekinthető, mert van olyan magasszintű nyelv, ill. több olyan géptípus (pl. IBM, Siemens, ICL, CDC stb.), ahol kezelését software támogatja. A közvetlen hozzáférésű tárolókon alkalmazható direkt és indexszekvenciális file-szerkezeteket az ún. kereszthivatkozás segítségével „egyesítve”, olyan rekord-rekord kapcsolatokat lehet létrehozni, ami az adatstruktúrák kezelését rugalmassá teszi. Fastruktúrák, aciklikus gráfok, ciklikus gráfok stb. kezelésének szempontjai indokolják, hogy ezeket a bonyolultabb logikai adatstruktúrákat több file hordozza, mégpedig gyakran úgy, hogy a gráf csomópontjaira vonatkozó ún. törzsmondatokból az adatállomány zömét kitevő törzsállományt (master file), míg a gráf pontjai között fennálló kapcsolatokat ún. láncállományba (struktúra file) tárolva képezzük (34. ábra).

Ez a gyakorlat továbbfejlesztve és a tapasztalatokkal igazolva általánossá vált, és több komplex file-kezelő rendszert (DBOMP, ISI, BASTEI stb.) ennek nyomán alakítottak ki. Ahol a kialakítandó adatállományoknak két nagy csoportja van:

- törzsállományok,
- láncállományok.

Ha több kulcs szerinti visszakeresésre is van igény és a véletlen hozzáférésű feldolgozás is előtérbe kerül, a rekordok szegmensekre bontása és azok összeláncolása jelenti az egyik megoldási lehetőséget (a másik az invertálás). A törzsállományt lényegében ISI módon és pointer-tömbös alakban (keresztindex-táblázat) kell létrehozni, ahol a láncon végigmenve, az egy kulcshoz tartozó listaelemek sorra visszanyerhetők (40. ábra). Elvileg egy file-ben akárhány lánc kialakítható, de egy bizonyos határon (5-7 kulcs) túl ezek kezelése, karbantartása bonyolultabbá válik (41. ábra). Számos tapasztalat utal arra, hogy láncállományok nélkül, csupán törzsállományokra építve nem lehet jó hatásfokú adatbázis-kezelő rendszert kifejleszteni.

8. fejezet

Elérési (elhelyezési) módok, file-kezelés

Az adatbázis-kezelő rendszerek egy fejlődési folyamat során alakultak ki. A fejlődés első szintje az ún. az alapszintű elérési módok mint pl.

SAM (Sequential Access Method),

DAM (Direct Access Method),

BSAM (Basic Sequential Access Method),

ISAM (Indexed Sequential Access Method),

BISAM (Basic Indexed Sequential Access Method) ,

BDAM (Basic Direct Access Method),

QISAM (Queued Indexed Sequential Access Method),

OSAM (Overflow Sequential Access Method),

CSAM (Control Sequential Access Method),

SIAM (Separatad Index Access Method).

Ezek az elérési módok a file-okat kezelve logikai rekord szintű adatelérést biztosítanak. Ha egy file több alkalmazási programot szolgál ki, akkor a teljes adatrekord leírása minden egyes programban szerepel, holott általában mindig egy-egy részrekordra (szegmensre) volna éppen szükség a program számára.

Második szinten pl. a BOMP, DBOMP, VSAM stb. (komplex) file-kezelő rendszereket tekintjük. Ezen a szinten pl. a VSAM (Virtual Storage Access Method) egyesíti és továbbfejleszti a SAM, DAM, ISAM lehetőségeit, és lehetővé válik a VSAM állomány hardware-független szervezése. De a VSAM sem tud logikai adatstruktúrát kezelni, így egy alkalmazói program még mindig teljes rekorddal kénytelen dolgozni. Ennek megfelelően nem valósulhat meg sem a program és az adat egymástól való függetlensége, sem a redundancia csökkentésének lehetősége úgy, ahogy azt kívánjuk. Ezeket az elvárásokat viszont a következő szinten lévő elérési módok már sokkal jobban megközelítik. Megjegyezzük, hogy létrehozható e két szint „közötti” olyan megoldás, amely a valamelyik alkalmazási program által egyaránt

igényelt adatmezőket összevonja egy központi file-ba, és a csupán egyes alkalmazásokat érintő mezőket ún. tranzakciós file-okban helyezi el.

A fejlődés harmadik szintjének tekintjük az olyan „hierarchikus” elérési módokat, mint pl.

- az IMS-ben használt HSAM (Hierarchical Sequential Access Method), amely a SAM-ra és a BSAM-ra támaszkodik,

HISAM (Hierarchical Indexed Sequential Access Method), amely az ISAM, BISAM, QISAM és OSAM módszerekre támaszkodik.

- a HIDAM (Hierarchical Indexed Direct Access Method), amely az ISAM, OSAM és DAM, BDAM együttesre támaszkodik. Ezeket azért nevezik hierarchikusnak, mert több szegmenstípus által vegyesen alkotott faszerkezet egy előfordulását (a teljes mélységű hierarchiát) tudják egyszerre könnyen kezelni.

A fejlődés negyedik szintjén a hálós ABKR-ekben használatos, rekord-típusonként rögzített ún. elhelyezési módok tekinthetők. Az elérési és elhelyezési módokat a harmadik és negyedik szintre vonatkozóan részletesebben a következőkben tárgyaljuk.

A fejlődés ötödik szintjének a relációs ABKR-eket tekintjük, amelyeknél a felhasználónak nem kell kijelölnie sem az elérési, sem az elhelyezési módot.

A tervezéskor és betöltéskor minden elérési mód hatékonysága leromlik, ha a tárolt rekordok sűrűsége nő. Az indexelt szekvenciális elérés bármely változatában gond lehet az indextábla túlcsoportulásából adódó lassulás, közvetlen elérésnél viszont az adatrekordok túlcsoportulása (szinonímák) okozhat problémát. Az adatbázis bővülései miatt tehát igen fontos a tárgazdálkodás. Az ABKR tárgazdálkodása nem engedheti meg abszolút fizikai címek használatát alkalmazói programban. Ezen túlmenően az ABKR tárgazdálkodása lehet olyan, hogy a file tárterületét mint összefüggő EXTENT-et (vagy ilyenek egy csoportját) jelöli ki, majd fizikai címekkel ellátott kisebb fizikai rekordokra (például blokkokra vagy lapokra) tördeli és a blokkon belül a logikai rekordra való hivatkozásra még ún. adatbáziskulcsokat is (logikai hivatkozást) használ, amely természetesen önmaga is helyet foglal el. A felhasználó így nem férhet hozzá a fizikai címekhez, de nem is érinti azok változása (tehát a hardware és a tárolási szintű változás).

Az ABKR tárgazdálkodása olyan is lehet, hogy egyes fizikai rekordok valamelyes bővülést még önmagukban, nagyobb bővülést a független (cilinder) túlcsoportulási területen, ennél nagyobb pedig szükség szerint dinamikusan

odaosztott, további területen foglalnak le. Ezek adatbáziskulcs helyett többnyire egy relatív byte címmel hivatkozhatnak a blokkon belüli rekordnak a file elejétől számított elhelyezési pozíciójára.

8.1. Gyors tulajdonoszegmens-sorrendű szerkezetek

Ezt a szerkezetet a HSAM és HISAM példáján mutatjuk be. Jellemzőjük, hogy az egyazon rekordba tartozó szegmens-előfordulásokat fizikailag is egymás közvetlen közelébe tárolják. Az ABKR számára az adatszerkezet leírására szolgáló nyelven (IMS esetén ez a DBD) rögzítik az érintett szegmenstípusok között fennálló hierarchikus struktúrát, amely egymás után, a fa prefix bejárásának sorrendjében van a blokkban tárolva. Hierarchikus ABKR-nél mindig a teljes hierarchia (fizikai adatbázis) előfordulásaihoz való hozzáférésről van szó. Ezért — és ez a gyors, direkt feldolgozást biztosító tárolási szerkezetre is vonatkozik — az elérés hatékonyságát az is befolyásolja, hogy a hierarchia fenti bejárési útján a bal oldali szegmensek előbb érhetők el, mint az ugyanazon szinten lévő jobb oldaliak. A hierarchiaszinten belül tehát a gyorsabban szükséges szegmenseket balra, a ritkábban szükséges szegmenseket jobbra sorolva célszerű elhelyezni.

On-line alkalmazásoknál tipikusan megfigyelhető a válaszütem növekedése, amely akár az adat, akár az indexfile-ok méretének bővülése, azaz a rekordsűrűség növekedése (röviden a túlcsoportosítás) miatt áll elő.

Habár az alkalmazói programok, működésük logikáját tekintve, nem függenek a file-ok méretétől, működésük hatékonysága függ például az ABKR pufferkezelési módjától, az alkalmazói adatkezelési és tárolási módoktól, az egyetlen eléréssel hozzáférhető rekordok számától, valamint a tranzakciók fajtáitól és időbeni eloszlásától. Például az ABKR puffereklését ugyan meg nem változtathatjuk, de az ABKR kiválasztása segítheti a fa- vagy hálószerkezetek kezelését, ha lehetővé teszi, hogy a program több különböző rekordtípus előfordulásaihoz egy eléréssel férjen hozzá.

A HSAM (Hierarchical Sequential Access Method) esetén egyetlen tulajdonos-szegmens előfordulásához több, eltérő típusú és típusonként (esetleg) különböző számban előforduló alárendelt tagszegmens tartozhat. A pointer az elsődleges kulcs. A HSAM sem az egyedi, sem a ciklusos, de még a

rekordtípuskénti teljes keresést sem támogatja. Csak azt segíti elő, hogy a hierarchia, egyes előfordulásait előhozza, vagyis a HSAM csakis a hierarchikus adatszerkezetek elérésére alkalmas módszer.

A HSAM korlátai szükségessé tették, hogy legalább az egyes fa-előfordulások közvetlen elérését meg lehessen valósítani. E célból fejlesztették ki a tulajdonosszegmensek azonosítóit egy indexfile-ba összefogó és az egyes fa-előfordulások gyökerét így közvetlenül elérő módszert, a HISAM-ot. Ennél az ISAM és — a túlcsordulás kezelésre — az OSAM módszert is hasznosították. Az IMS újabb változatai a BSAM-ot használják az ISAM-OSAM helyett, és ez még a ciklusos kezelést is segíti. Az eredeti formájában létrehozott HISAM esetében az egy adatmondatba tartozó szegmenstípusokat csoportosítják több tárolási egységbe. Ez akkor lehet célszerű, ha a hierarchikus szerkezetű rekord nagy számú tagszegmens-típusból áll és az adatbázisfunkciók egyes csoportjai egyszerre döntően ezen tagszegmenseknek egy-egy jól körülhatárolt csoportját igénylik. Ilyenkor előnyös ez a tagolt tárolás, mert egy rekordnak csak egyetlen éppen érintett tagszegmensét kell elérni és kezelni.

Az ISAM file-ba ehelyett tulajdonosszegmens elsődleges kulcsát pointerként az OSAM file(ok)-ba helyezett csoportokba is be kell vinnünk. Új rekord (ez már egy OSAM file-ba kerül) később beszúrhatósága érdekében az ISAM file és OSAM file rekordjai még egy-egy további mutatót tartalmaznak. Az IMS azonban a fenti két elérési módon kívül, amelyek közül a HSAM csupán faszerkezetek elérését teszi lehetővé, ún. „logikai” hálószerkezetek kezelését is megvalósította, ehhez azonban a BDAM és a HDAM elérési módszerek is használhatók. Ezek közül az IMS esetén aszerint célszerű választani, hogy milyen arányban van szükség rekordtípus-elérésekre, illetve egy több rekordtípusból összetett faszerkezet előfordulásainak elérésére.

8.2. Gyors közvetlen elérést biztosító elérési módok

Az egyes szegmenseket, szegmenscsoportokat fizikailag össze nem függő területen tárolják. Példaként az ún. HDAM és HIDAM módszereket mutatjuk be. Mivel a rekordok szegmensei egyedileg közvetlenül címezhetők, illetve

mutatóval egymásból is elérhetők, lehetőség van arra, hogy egyszerű módon ne csupán a rekordok, hanem a rekordrelációk szintjén is létrehozzuk a kapcsolatokat. Ez azt jelenti, hogy a rekordtípusok közötti kapcsolatok (ez általában egy háló) kezelése az ilyen esetben célszerűen biztosíthatók. Az adatbázis működésének hatásfokát azonban nem csupán az adatállomány elérési módjának alkalmas kezdeti megvalósításával, hanem az adatbázis változó környezetéhez alkalmazkodóan, az elérési mód későbbi megváltoztatásával is növelni lehet. Elég gyakori például, hogy az adatbázis tervezői kezdetben, még a felhasználói igények felmérésekor, arra következtettek, hogy azokat az igényeket valamilyen szekvenciális elérési mód (pl. HSAM, HISAM stb.) ki tudja elégíteni és az adatbázis-állományok betöltését így is végezték el. (Ettől az elérési mód még megváltoztatható pl. HIDAM, CALC stb.-re.) Ez a változtatás rendszerint az egész adatbázis tárolását egyszerre változtatja meg, de mivel ez csaknem akkora feladat, mint az eredeti betöltés, célszerűbb ilyen esetben az adatbázis egy fokozatonkénti (ún. dinamikus) átszerkesztése. A jelenleg működő ABKR-ek közül alig néhány teszi lehetővé a dinamikus átszerkesztést. Ami pedig az alkalmazói programokat illeti, igen célszerű, ha úgy készülnek el, hogy a későbbi várható adatbázis átszerkesztése miatt minél kevésbé tartalmazzanak az elérési módtól függő, és ezért esetleg kicserélhető utasításokat. Az adatbázis átszerkesztését a konkrét ABKR-típus által nyújtott lehetőségek esetleg nagymértékben megközelíthetik, ezért célszerű ilyen szempontokat is figyelembe venni az ABKR kiválasztásakor.

A HDAM (Hierarchical Direct Access Method) esetében az egyes szegmenseket, szegmenscsoportokat fizikailag össze nem függő területen tárolják. Ilyenkor a legcélszerűbb alkalmazni, ha csak egyes adatmondatok egy-egy alkalommal való elérését kell biztosítani. Alapvetően ezt támogatja a rekordtulajdonos-szegmensének közvetlen címzése. Innen tovább az első tagszegmenst két irányba (ún. fizikai gyermek-fizikai szülő) mutatók láncolják, majd az ugyanezen a hierarchia szinten lévő többi tagszegmenst (ún. fizikai iker) ugyancsak mutatók kapcsolják össze. Új rekordok vagy új tulajdonos vagy tagszegmensek beszúrása, illetve törlése csupán a mutatók átállítását igényli. Ha néhány szegmens éppenséggel nem is csak egy, hanem két vagy több rekord részét képezi, akkor ezek ún. „logikai mutatóval” egymással egy csoportba összekapcsolhatóak.

A HISAM-nál már említettük, hogy ha adatbázisfunkciók egy csoportja alárendelt szegmensek egy-egy jól körülhatárolt csoportját igényli, akkor az együttkezelhetőség céljából hasznos az ilyenfajta kapcsolatok létrehozása. A HIDAM-nál ezt a hierarchia különböző szintjein lévő szegmensekre nézve megtehetjük, míg a HISAM-nál csupán mindig egyazon szinten alkalmazható. Minden szegmenscsoporthoz egy elsődleges és egy túlcsoordulási tárterület tartozik. Az előbbire a tulajdonosszegmensek és a gyakran használt tagszegmensek, az utóbbira a ritkább hozzáférési igény vagy helyhiány miatt behelyezett tulajdonos- és tagszegmensek kerülnek. A túlcsoordulási terület szerkezete a HSAM-éhoz hasonló, a túlcsoordult szegmens az OSAM-mal érhető el.

A HIDAM (Hierarchical Indexed Direct Access Method) a HISAM és a HDAM elérési módok előnyeit egyaránt hordozza. Itt a HDAM-hoz még egy indexszerkezet is társul, így az egyes rekordok a tulajdonosszegmens címen kívül a szegmens kulcsát is tartalmazzák. Az indexállomány két részre tagolt: az ISAM-területen az indexszegmensek, az OSAM-területen az újonnan beszúrt indexszegmensek találhatók.

A szekvenciálisan tárolt alapfile tulajdonos- és tagszegmenseit is külön OSAM-területen helyezik el.

A HDAM és HIDAM mellett, hogy az egyedi keresésre jók, alkalmasak szekvenciális keresésre is; nem csupán a tulajdonosszegmensek szekvenciális, hanem a hierarchia bármely szintjén lévő szegmensek (szegmenscsoportok) feldolgozására is alkalmasak. Kétirányú keresésre azonban egyik elérési mód sem ad lehetőséget. A ciklikus keresést azonban másodlagos kulcsok alapján biztosítja a VSAM, és így annak IMS OS-ban működő változatai is.

8.3. CODASYL — jelentés által tárgyalt elérési, elhelyezési módok

A CODASYL javasolta elhelyezési módok fizikai alapegysége a rekord. Az ABKR-ek számára ajánlott (elhelyezési) módok segítségével a felhasználó meghatározhatja a rekordok optimális eloszlását a háttértárolókon, ezzel a neki legjobban megfelelő elhelyezést biztosítva. Az alapvető elérési mód célszerűen DAM lehet egységesen. Ugyanakkor ez az optimalizálás nem veheti figyelembe a háttértár konkrét típusát és fizikai mivoltát, szerkezetét, mert akkor az adatok és a programok függéséből adódó problémák pl. egy háttértároló típusváltásakor kerülnének előtérbe.

A fenti ok miatt is az adatbázis háttértár-területét nagyobb egységekre, ún. area-kra (realm) tagolják, és a rekordokat ezekhez a területegységekhez rendelik úgy, hogy azok elhelyezését az area-n belül optimalizálhatjuk. Az area kisebb egységekre, lapokra (page) osztható; ezeknek hossza egy area-n belül azonos. Az adatbázist pl. mágneslemezen néha egy, de általában több file formájában tárolják.

Ezek a file-ok az alkalmazott elérési mód (DAM) szerint blokkokból állnak. A blokkok mérete megfelel az adatbázis page-méretének, azaz egy lemezfordulással egy lapnyi blokk hozható be.

Az ABKR tárgyzádkódását végző program azonosítja az adatbázison belüli area-kat, azaz minden egyes, két-két adatbáziskulccsal behatárolt összefüggő területet, és e határoló értékekhez egyértelműen hozzárendeli a háttértáron elhelyezett file relatív blokkcímeit úgy, hogy az ott kapott tárolóterületek egymást át nem fedő, de nem szükségképpen összefüggő area-kat jelöljenek.

A file-ok és az area-k többféleképpen is megfeleltethetők egymásnak, hiszen az egész adatbázis lehet egyetlen file vagy area-nként külön file-okból állhat, de egy file több area-n is elhelyezkedhet vagy egy area akár több file-ból tevődhet össze. A lapok és az area-k úgy feleltethetők meg egymásnak, hogy egy area-t több, folyamatos sorszámú lap alkothat, de egy lap csakis egyetlen area-hoz tartozhat, és azon az area-n belül a lapméret area-nként eltérhet. A rekordtípusok és az area-k úgy feleltethetők meg, hogy egy rekordtípus minden egyes előfordulása csak ugyanabban az area-ban lehet.

Deklarálni kell az adatbázis rekordtípusait, valamint rekordtípusként külön azok elhelyezési módját, végül a rekordtípust alkotó mezőtípusokat. Az elhelyezési móddal rekordtípusként egyrészt rögzítjük a rekordtípus tárolási módját az area-n belül, másrészt ezzel együtt a rekordtípus egyetlen kiválasztott elhelyezési módját is.

Egy adatbázisból lényegében négyféle módszerrel nyerhetők rekordok:

- közvetlen címmel, (ahol egy pointer mutat a rekord helyére a háttértárban),
- set-en keresztüli eléréssel (ahol a „következő” rekord címét magában az adatszerkezetben előre elhelyezett pointer határozza meg),
- a rekord egy, esetleg több mezőjének tartalmát tekintve elsődleges kulcsnak; (ennek két alapformája van: egyik az ún. hash címezés, ahol véletlen elérésről van szó, a másik a többszintű indexeléssel való címezés),
- teljes kereséssel a file-on.

A gyakorlatban az ABKR megengedett elérési módjai sokszor a fenti alapvető lehetőségek valamilyen összetételéből adódnak. Pl. a CODASYL-alapú IDS és az IDMS alkalmazza, az első három (IDMS-béli elnevezésük DIRECT, VIA SET, CALC) formát, míg a CODASYL-tól eltérő IMS négy olyan elérési módot használ (HSAM, HISAM, HDAM, HIDAM), amelyek jobbra a többszintű indexelés és a közvetlen címzés kombinációi stb.

CALC : itt a rekord valamelyik mezőjét kell kijelölni elsődleges kulccsal, és az ABKR eszerint randomizálva tárolja a rekordokat (ez nem befolyásolja a rekordelőfordulások logikai kapcsolatait).

A CALC elhelyezés lényegében a véletlen elérésű file-szerkezettel azonos, tehát a megadott kulcsra a rendszer egy hash-algoritmussal számítja ki a rekord címét. Ez az algoritmus lehet a rendszer sajátja vagy megadható más algoritmus. Ha a rekordtípus előfordulásához közvetlen hozzáférésre van szükség, akkor célszerű CALC elhelyezést alkalmazni, megadva azt a kulcsot, mely szerint a közvetlen hozzáférést kívánatos biztosítani. Erre az elhelyezési módra nemcsak akkor lehet szükség, ha az adott rekordtípus az adatbázisba való belépési pont, hanem akkor is, ha a navigáció során őt mint tulajdonost a tagrekord felől hasznos elérni. Kötelező a CALC elhelyezési mód akkor, ha olyan kapcsolat tulajdonosáról van szó, amely kapcsolat set-ként nem valósítható meg. Ekkor ugyanis az eredeti tagtól a tulajdonoshoz csak a benne kapcsolódó, adattételként szereplő tulajdonoskulcs alapján juthatunk el. Akár tulajdonos, akár tagrekord is lehet CALC elhelyezésű.

Ekkor a véletlen hozzáférést követően a rekordelőfordulás egy vagy több mezője tartalmából képzett szimbolikus kulcsból az ABKR egy közvetlen címet határoz meg. Ez a közvetlen cím azonban még csak egy adatbázisbeli relatív cím és lapszám. A rekordelőfordulást a lapon belül oda lehet elhelyezni, ahová a lapon belüli újabb relatív cím (ún. sorszám) éppen mutat. A rekordelőfordulás egyedi azonosítóját (az adatbáziskulcsot) az ABKR a lapazonosító és a lapon belüli sorszám együtteséből képezi és ennek alapján is el lehet azt érni.

Szükségünk lehet arra is, hogy másodlagos kulcs alapján keressük vissza a rekordokat. A CODASYL-alapú rendszerek egy része erre explicit lehetőséget ad (SEARCH-KEY), más részük (pl. IDMS) nem. Másodlagos kulcs szerinti visszakeresésre azonban mind a két esetben mód nyílik, ha ez a kulcs külön rekordtípusként van meghatározva. Lényegében így működnek azok a rendszerek is, ahol SEARCH-KEY megadására van lehetőség. Ekkor a másodlagos kulcsnak megfelelő rekordtípus (ez SEARCH-KEY esetén implicit) révén újabb set-ben lesz tag az a rekord, melyre a másodlagos kulcsot bevezetjük. Végeredményben tehát ezt a lehetőséget csak a hatékonyság rovására használhatjuk ki.

DIRECT: ugyancsak a közvetlen hozzáférést biztosítja, de első közelítésben nem valamilyen adattételre mint kulcsra, hanem az ún. adatbáziskulcsra nézve, melyet az ABKR ad minden egyes rekordelőfordulásnak, egyedi, belső azonosítóként. Egyes rendszerek (pl. IDMS) módot adnak arra, hogy megfelelő (adatbáziskulcs formai követelményei szerinti) kulcsot adva egy rekordnak, ezt fogadja el a rendszer adatbáziskulcsként. Az adatbáziskulcs megadásával a felhasználó határozza meg, hogy a rekordelőfordulás melyik lapra — vagy túlsordulás esetén (az area-n belül) ahhoz minél közelebbi lapra — legyen elhelyezve.

VIA SET: azokra az egyedtípusokra nézve célszerű elhelyezési mód, ahol közvetlen elérésre nincs szükség, mert mindig valamilyen set-en keresztül jutunk el hozzájuk. Ilyenkor el kell dönteni, hogy több set esetén, melyik legyen az, amelyiken keresztül hasznos lehetővé tenni a hozzáférést. A visszakeresési idő és a biztonság miatt célszerű, ha a set-ben résztvevő tulajdonos- és tagrekord-típusok egymáshoz lehetőleg minél közelebb (pl. egyazon blokkon tárolva) helyezkednek el. Ez azt jelenti, hogy nagy valószínűséggel a tulajdonos- és a tagrekord-típusok egy lemezeléréssel egyszerre hozhatók be a pufferbe.

Lapkapcsolaton keresztül történő (VIA SET) elérés csak tagrekordoknál lehetséges. Ilyenkor a tagrekord-előfordulásokat csak a tulajdonosrekord-előforduláson, azaz a set-en keresztül haladva valamely irányba lehet tárolni, illetve elérni, így a set célszerűen lehetőleg ugyanazon a lapon helyezkedik el. A set-en belüli sorrend meghatározása céljából előre látni kell azt is, hogy minden set egészének külön lapra helyezése azt jelenti, hogy a set előfordulását alkotó rekordelőfordulást (CALC) és a tag (VIA SET) rekordelőfordulások set-enként egy lemezeléréssel hozhatók be a tagba.

A szimbolikus kulcs ismeretében indextáblázatokon keresztül is hozzáférhetünk az adatbáziskulcs jelölte címhez, de az elérés, mivel ehhez több szükséges, tehát lassúbb, és a táblázatok tárigényesek (SPF). Ha az ABKR tárgyzárlat olyan, hogy a törölt rekordok helyét újra használhatjuk és a file mérete viszonylag kicsi, akkor a közvetlen elérési mód mellett a változások okozta hatékonyságromlás általában kisebb. Az elhelyezési mód kiválasztása (és az adatbázis ilyen szerkezetbe való kezdeti betöltése) azon az igényfelmérésen alapul, amelyet az adatbázis-tervező elvégez ugyan, de amely (adatbázisfunkcióra vonatkozó) „igénykeverék” idővel megváltozik. A felmérésnek ugyanis legalább két független, változó összetevője is van:

- maga a vállalat (ezt az adatbázis is tükrözi),
- a felhasználó által eddig megszokott szekvenciális elérési igény (az adatbázis lehetőségeit felismerve, egyre több random elérési igény lép fel).

Adott operációs rendszer és adott hardware-feltételek mellett is igen eltérő lehet a tárolási szerkezet megvalósítása, ha a felhasználási igények jelentősen megváltoznak. A hozzáférési utak optimális kialakítása érdekében újabb — az eredetileg létrehozott adatmodell szintjén nem is létező — kapcsolatok jöhetnek számításba.

Amennyiben a felhasználói programok körében olyan mértékű változás következik be, melyet a korábbi tárolási rendszer már nem képes hatékonyan kihasználni, akkor a tárolási struktúrát az új igények szerint át kell alakítani.

9. fejezet

Az adatbázis változásai

9.1. Az adatbázisfelelős

Az adatbázisfelelős feladata, hogy az ABKR nyújtotta szolgáltatások (pl. utility-k), és saját fejlesztésű megoldások segítségével rendszeresen feljavítsa az adatbázis környezetének változásai, illetve az ABKR működése által lecsökkentett adatbázis-teljesítményt. Méri (megbecsüli) az adatbázis változásait, és ő dönt arról, hogy mikor és milyen módon a legcélszerűbb a változások káros hatásait kiküszöbölni. Rá van bízva, hogy a logikai adatmodell szintjén megjelenő szegmenstervezéstől az area-k meghatározásának lépéseitől vagy az adatbázis fizikai szerkezetének megváltozásától kezdi el a beavatkozást. Amellett hogy így elvégezheti a fizikai adatbázis jelentős ráfordítással járó átszerkesztését, olyan, általában kisebb ráfordítást igénylő módszereket is alkalmazhat, mint például a feldolgozások módosítása, a job-ok futási sorrendjének vagy maguknak az alkalmazói rendszereknek a megváltoztatása.

Ahhoz, hogy az adatbázis tartalma és szerkezete követhesse a valóság és így az azt tükröző adatok dinamikus változásait, olyan fontos szempontokat kell figyelembe venni, mint például:

- az adatbázist több felhasználó „megosztva” egyidejűleg fogja kezelni,
- az adatok kezelését számukra minél egyszerűbb programokkal lehessen megoldani,
- az adatok megváltozása minél kevésbé vonja maga után a felhasználók programjainak kényszerű megváltoztatását.

Az adatbázis változása rendeltetésszerű használatának egyenes következménye. A változások lehetséges típusait, okait teljeskörűen nem soroljuk fel, csupán néhány fontosabbat emelünk majd ki a későbbiekben. Egy kiválasztott ABKR azzal, hogy benne adott a séma-alséma összekapcsolás (binding) módja, eleve meghatározza, hogy milyen mértékben lehet hatással a tárolási szerkezet megváltozása a felhasználói programra. Az összekapcsolás,

önálló nyelvű ABKR esetén a séma DDL-nyelvű inputja után közvetlenül fizikai címekre hivatkozási táblák előállítását jelentheti. Ezeket a program tényleges végrehajtásakor használják fel, abszolút fizikai címek előállításához. Lehetséges olyan összekapcsolás is, amely az alkalmazói program fordításakor hozza létre az abszolút fizikai címeket. Beépülő nyelvű ABKR esetén ez a gyakori. Az előbb említett két összekapcsolási mód hátránya, hogy a séma megváltozása a programok újrafordítását igényli. Ezt a hátrányt ki lehet küszöbölni akkor, ha az összekapcsolást az alkalmazói program betöltésekor, esetleg még később, a file-ok (area-k, set-ek) megnyitásakor vagy végső fokon az adatok elérésekor hajtják csak végre.

Ma nincs olyan ABKR, amely meg tudná valósítani az adatok teljes függetlenségét a programtól; az ABKR-ek séma és alséma processzorai sem tudják ezt elérni. Ha egy rekord tartalma pl. egy új mezővel bővül, akkor azt a DDL segítségével le kell írni, és ha az ABKR bővülő nyelvű, az alséma által érintett programok utasításai közé kell illeszteni, hogy ezt a mezőt elérhesse. Az így beillesztett utasítás mellett a program logikája is általában változik. A módosított programokat tárgykódba újra fordítani és könyvtárazni kell. A fentiek olyan esetre vonatkoznak, amikor a logikai rekord fizikai rekordból áll össze. Ahol ez éppen fordítva van, azaz egy fizikai rekord több logikai rekordból jön létre, ott a fizikai rekord az alkalmazói programok egy részében azok „tudta nélkül”, új mezőkkel bővíthető, ha az ABKR mező-mélységű eléréssel rendelkezik. Ha viszont az elérés mélysége szegmens (mint pl. IBM rendszerei) vagy rekordszintű, akkor a mező helyett ténylegesen csak új szegmenst vagy rekordot lehet beszúrni.

Egy file-on belül ha egyszerű mezőcseré lép fel, azt az alkalmazói programozó nem veheti észre. A szükséges csere végrehajtását az ABKR séma-alséma leképezési funkciója kezeli. Ha viszont, a mező (és így a rekord) szélességét érintő változás lép fel, azt az ABKR már nem tudja kezelni; ehelyett az érintett alkalmazói programokat kell módosítani. Séma módosítása CODASYL- alapú ABKR esetén a fizikai file módosításával jár. A séma módosítását követően, az alsémákkal való egyezőség biztosítása érdekében ez utóbbiakat át kell tekinteni, mert kiderül(het), hogy alsémát is kell változtatni, és a programokat újra kell fordítani.

A logikai adatmodell változtatása a set-tagsági viszony megváltozását eredményezheti. Ha a taghoz új tulajdonos kapcsolódik vagy új funkcionális függéseket kell odakapcsolni, akkor a megfelelő tagsági viszony fizikai kap-

csolatrendszerét képviselő elemek módosítása bonyolultá válik. Utility programot e célra nemigen találhatunk, így a felhasználó vagy az adatbázisfelelős feladata olyan programot írni, amely például új függések hozzáadásakor elvégzi ezen új kapcsolatok létrehozását.

Ha az adatbázisba új egyed típusok és új kapcsolatok (tehát új rekord- és set-típusok) egyszerre és nagy számban kerülnek, akkor az adatbázis olyan jelentős bővüléséről van szó, amit az ABKR és az előző állapotra tervezett adatbázis célszerűen csak úgy tud befogadni, ha az adatmodellt újratervezik, és általában az egész összegzett adatbázist újratöltik. Amennyiben ebből az adatmodellből az derül ki, hogy a régi és az új részek egymástól szinte függetlenek és csupán egy-két kapcsolat köti őket össze, akkor a régi rész újratöltése helyett elegendő a már meglévő érintett file-okat a kapcsolatokat képviselő mezőkkel (pointer) kibővíteni. Az adatszótár segítségével feltárható, hogy a fenti változások mely programokat érintik, s újra kell-e tervezni vagy elég csak módosítani és újrafordítani azokat.

9.2. Az adatbázis tartalmi épségének védelme és helyreállítása

9.2.1. Védelem

Az adatbázis értékét a hardware, software és felhasználói adathibák ellen egyaránt védeni kell. Az adatbázist eleve úgy kell tervezni és kivitelezni, hogy kicsiny legyen a teljes rendszer hibájának a veszélye. Ezt rendszerint az adatbázis fizikai megosztásával, külön berendezéseken való tárolásával érik el. Az adatbázis védelmét szolgálja az alsémán kívül a rendelkezésre álló terület fizikai felosztása, amely a rekordtípusoknak egymástól fizikailag jól elkülönített tárolását és részenkénti védelmét segíti elő. A felosztási egység itt nagyobb, mint egy fizikai rekord (blokk vagy lap) mérete, és az ABKR-ektől függően (kissé eltérő tartalommal) partició, area, realm, subfile, dataset, fizikai adatbázis stb. neve van.

Area (vagy realm) pl. a CODASYL sémában az adatbáziskulcsok egy névvel ellátott csoportját jelöli. Az egy realm-ba (vagy area-ba) egymás mellé rendelt több rekordtípus ugyancsak megvalósítható. Az egy realm-ba kerülő rekordtípusok (szegmenstípusok) csoportosítási elve általában a

gyakori együttes használat. Fentieken túl, ha a programok is külön partíciókban futnak, jó programvédelem valószínűsíthető meg. Szükséges, hogy a felhasználóknak saját jelszava legyen; elérési jogosultságukat a rendszer ellenőrzi, így minden terminál csak a jelszó ellenőrzése után kapcsolódhat az ABKR-hez.

A már elérhető adatbázis kezelése közben is léphetnek fel az adatbázist károsító hibák. Az ezek elleni védelmet is tervezni kell. Ezek főbb fajtái:

- az adatvesztés elleni védelem, a feldolgozás valamennyi fázisában. Ez rendszerint csak hardware- vagy rendszerhiba esetén fordul elő. Fontos, hogy a rendszer erről értesítse a felhasználói programokat;
- az adatbázis hibás aktualizálása elleni védelem, amely magába foglalja: az input-adatok érvényességének ellenőrzését, az adatok elérésére, aktualizálására, törlésére vonatkozó jogosultság vizsgálatát, a helyes adatbázisverzió, tranzakció, és archiv file használatát;
- a programok váratlan kölcsönhatása elleni védelem, amely az egyidejű update ellenőrzést és az ún. holtponthoz feloldását követeli meg az ABKR-től.

9.2.2. Helyreállítás

A hardware hibák elleni védelem leggyakrabban alkalmazott módszere a Check-point/restart. Lényege ennek, a file-kezelő programok futása során vagy szabályos időközönként vagy egy erre alkalmas állapot bekövetkezőkor az ABKR megőrzi a file-ok éppen aktuális állapotát. A futás során esetleg bekövetkezett hiba felismerése után ettől a ponttól lehet a feldolgozást újra indítani. Ez a módszer a megismerés miatt elvesztett gépidőt csökkenti, viszont nagy — bár viszonylag olcsó — tárolókapacitást igényel.

A program és a hardware, valamint egyéb hibák elleni védelem másik eszköze a file-ok több generációjának megőrzése. A file módosítás előtti állapotát a módosítás tranzakciófile-jaival őrzi meg a rendszer. Ha kiderül, hogy a módosítás során valami hiba történt, az adatok, illetve a program kijavítása után a feldolgozást az előző generáció (vagy generációk) alapján meg lehet ismételni.

A helyreállítás fő eszköze a visszagörgetési (helyreállítási) mechanizmus, amely egyrészt az adatbázis kiemelt tartalmának, másrészt a tranzakciónak a másolatán alapszik.

A másolatoknak (image) három típusa van:

- az „előző állapot” másolat az adatmódosítás előtti lapot rögzíti (egy szekvenciális archív file-on);
- a „gyors előző állapot” másolat az adatmódosítás előtti lapot egy közvetlen elérésű archív file-on őrzi meg;
- az „utóállapot” az adatmódosítás után létrejött lapot rögzíti egy szekvenciális archív file-on.

Az első két állapotot arra használhatjuk, hogy időben visszafelé haladva, eltüntesse az adatbázisból a hiba hatásait. A harmadik akkor alkalmazható, ha időben előre haladva, újból helyre kell állítani az adatbázist,

A „gyors előző állapot” típust visszagörgetéshez és az ún. gyors helyreállításhoz, az „előző és az utóállapot”-ot pedig az ún. lassú helyreállításhoz használja az ABKR.

A gyors helyreállítás olyan folyamat, amely helyreállítja az adatbázis hiba előtti állapotát. Alkalmazása az adatelérési rutinok belső hibája vagy rendszerhiba esetén válik szükségessé. A gyors helyreállítás, amely automatikusan vagy gyors operátori beavatkozás nyomán indul meg, a „gyors előző állapotot” tartalmazó archív állományból történik. A visszaállított adatbázis-állapotról ezután egy „utóállapot” típusú másolat készül.

A lassú helyreállítás hardware-, software-, üzemeltetési vagy felhasználói hiba esetén használható a sérült adatbázis (adatbázisrész) újra betöltésére. A helyreállításhoz a rendszer a korábbi adatbázis dumpot és egy archív file-t használ, amely vagy a módosítás utáni eseményeket és ellenőrző pontokat (check point), vagy a módosítás előtti és utáni helyreállítási pontokat (recovery point) tartalmazza. Az ellenőrző pontok készítésekor nincsen aktív program, a helyreállítási pontok készítésekor viszont a programok elérhetik az adatbázist. Ezért az adatbázis visszatöltésekor a helyreállítási pont készítésének ideje alatt lezajló eseményeket „vissza kell pörgetni”, hogy megkapjuk a korrekt adatbázis-állapotot.

Az egyidejű osztott hozzáférés elsősorban az adatok karbantartásakor jelenthet hibaokot, amit általában úgy próbálnak kiküszöbölni, hogy a karbantartás idejére a rekordtípusok egy adott körét kizárólagosan csak egy felhasználó kezelheti.

A zárolás a legtöbb rendszerben rekordszinten történik és csak a karbantartás időtartama alatt áll fenn. A rekordok zárolása általában háromféleképpen valósítható meg:

- zárolást jelző bit átállítása a zárolt rekordokban (hátránya, hogy rendszerhiba esetén a zárolt rekordokat először fel kell szabadítani),
- a zárolt rekordok azonosító listájának operatív tárba tárolása (amelyhez minden elérés előtt az egyes programoknak oda kell fordulni),
- a zárolt rekordoknak az operatív tárban tartása (hátránya a nagy tár-igény, ezért csak azokat a rekordokat tartják a tárban, amelyeket igen gyakran módosítanak).

Zárolás esetén felléphet az ún. holtpontról (dead-lock) állapota, amikor az előző program, más programokat kizárva, leköti az 1. rekordot, a második program pedig ugyancsak kizárólagosan a 2. rekordot, majd kölcsönösen hozzá akarnak férni a másik által lekötött rekordhoz. Ekkor a zárolások miatt mindkét program várakozó (letiltott) állapotba kerül, és egyik sem tudja a feldolgozást folytatni. Az ABKR-nek fel kell ismernie ezt a helyzetet és a rendszert ki kell mozdítani a holtpontról. A rendszerek nagy többsége ilyenkor vagy a legmagasabb prioritású vagy a legtöbb zárolt rekorddal rendelkező programot engedi tovább úgy, hogy a többi programot a zárolás előtti állapotra állítja vissza, azaz zárolt rekordjaikat felszabadítja. Ezt a folyamatot „visszagörgetés”-sel éri el.

A visszagörgetés (roll-back) tehát olyan folyamat, amely egy program vagy egy utasítás hatását fordítja vissza. Az események visszafordítása a következő okokból következhet be:

- az adatmódosítás után derül ki egy utasítás hibája,
- holtpontról esetén az adott programban kell felszabadítani a zárolt rekordokat,
- operátori utasításra,
- a programban lévő utasítás hatására.

9.2.3. Az adatok időegyezősége

On-line ABKR-ben, ahol a karbantartás folyamatos, a különböző időpontokban készült, de később együtt felhasznált kimutatások között jelentős tartalmi eltérés lehet. Az adatok időegyezőségét csak software megoldással lehet biztosítani, mert az inputok beérkezésének és feldolgozásának sorrendje nem mindig egyezik meg az események sorrendjével. A helyes időbeli sorrend előállítása érdekében minden input rekordban rögzíteni kell a sor-

rendiséget egyértelműen meghatározó időpontot is. ABKR-ek például a következő technikai megoldások alapján állíthatnak elő időegyező jelentéseket:

- a jelentéskészítés és elemzés ideje alatt felfüggesztik az adatbázis aktualizálását (interaktív környezetben nem alkalmazható),
- a jelentéskészítéshez és -elemzéshez szükséges adatokról másolatot készítenek (másolatkészítés többletidőt jelent, és ezen idő alatt nem végezhető karbantartás),
- az adatok korábbi verzióinak megőrzése, időpont megjelölésével. Így egy rekord a korábbi rekordváltozatokat magához láncolja, lehetővé téve a visszamenőleges-esemény jelentések elkészítését is (hátránya a nagy tárolóigény).

II. rész

1. fejezet

Osztályozással segített tervezési szakaszok kiemelése

1.1. Bevezetés

Az információs rendszer fejlesztésével szemben támasztott egyik alapvető követelmény, hogy ezt a munkát számítógép segítse. A legtöbb eredmény az elvi adatmodellnek a valóság elemzése alapján való megadásához kapcsolódik. Mivel a logikai adatmodell a tulajdonostípusok közötti funkcionális függéseken kívül a felhasználó hozzáférési követelményeit, valamint az adatbázis-tervezési módszer és a választott ABKR sajátosságait is tükrözi, a logikai tervezésre általános érvényű módszerek kevésbé adhatók. Alkotható azonban olyan számítógépes eszköz, amelynek rendeltetése a hozzáférés-elemzés általános támogatása. A fizikai tervezést segítő módszerek és eszközök általános érvényű voltáról még ennél is kevesebb mondható; ezek az adatbázisok tervezésének viszonylag kevésbé vizsgált területét jelentik.

Az elvi adatmodell leképezését a társzerkezetre általában úgy kell végrehajtani, hogy az adatbázis működési hatékonyságát leginkább kifejező tárkihasználás, válaszidő és helyreállíthatóság tekintetében optimumra kell törekedni. A mindhárom szempontból, együttes optimalizálás helyett általában úgy járunk el, hogy a logikai adatbázis tervezésekor a válaszidő minimalizálásra, a későbbi fizikai tervezés során a tárkihasználásra és a helyreállíthatóságra mint domináns szempontokra vagyunk tekintettel. A felhasználó számára a válaszidő az egyik legfontosabb szempont. A válaszidőt az alábbi tényezők alapvetően befolyásolják:

- a hozzáférési igények,
- a hozzáférés jellege: a minél gyorsabb visszakeresésre vagy pedig a gyors karbantartásra, a minél frissebb adatokra tesszük-e a hangsúlyt (ez be-

folyásolja azt, hogy pl. egy rekordtípus legfeljebb hány adategyüttesben legyen tag),

- a hivatkozás gyakorisága: egy rekordtípus elérése iránti becsült relatív gyakoriság,
- a hozzáférés módja: teljes keresés, egyedi keresés, ciklusos keresés,
- a végzendő művelet: hozzáadás, törlés, visszakeresés.

Az ABKR korlátait, az adottságait az, elhelyezési módok (az ABKR által biztosított elhelyezési módok) és a tárolási módok (pl. adatsűrítés lehetősége) szabják meg.

Jól ismert, hogy az adatbázisok hatékonysága erősen függ a tulajdonságtípusnak rekordtípusra (vagy részrekordtípusra) bontásától, illetve rekordtípusoknak (a továbbiakban: részrekordtípus = szegmenstípus) nagyobb egységekké csoportosításától. Szinte természetesen kínálkozik alkalmas, számítógéppel segített csoportképző eljárás alkalmazása, még a fizikai tervezést megelőzően. Mivel a tapasztalat szerint így jelentősen csökkenteni lehet az adatelérés idejét, az adatbázis-tervezésben ennek nagy gyakorlati jelentősége van.

A feladat két fő lépésre tagolt:

- egyedtípusok szegmenstípusokká bontása ,
- szegmenstípusok nagyobb egységekké csoportosítása.

A megoldás módszerét mindkét esetben az osztályozásra alkalmazható eljárások egy csoportjából, a klaszteranalízis módszereinek köréből választottunk ki.

A klaszterálás (más néven klaszteranalízis) a többváltozós matematikai-statisztikai módszerek egyike. Igen alkalmas bonyolult rendszerek belső struktúrájának feltárására és tervezésére. Egyre több területen, pl. a könyvtári visszakeresésben, az adatfelismerési és a statisztikai problémákban eredményesen alkalmazzák. Az információs rendszerek tervezésének egyes fázisaiban is alkalmazzák klaszteranalízist, pl. az egyes alrendszerek elhatárolására, egy szervezet információrendszere teljes architektúrájának feltárására, bonyolult programrendszerek tervezéséhez. Az adatbázis fizikai tervezése során — különösen az igen nagy adatbázisok esetén — egyre gyakrabban alkalmazzák az adatszerkezetek tárolószerkezetekre való leképezésének (a rekord-elrendezés optimalizálására, illetve az elérési idők minimalizálására) céljából.

Tudomásunk szerint eddig csupán elvétve és az itt leírttól eltérő módszereket használva alkalmaztak klaszteranalízist az adatbázis logikai tervezése során. Mivel az elvi adatmodell fizikai megvalósítása még nem veszi figyelembe a hatékonyság szempontjait, ezért a logikai adatbázistervezésen belül az elvi modell alkotta kiinduló klaszterstruktúrát úgy transzformáljuk át, hogy egy egyedítípust általában több szegmenstípusba képezünk le, majd e szegmenstípusok közül az adatigény szempontjából közelálló funkciókat, kiegészítőket egy nagyobb csoportba, area-ba vonjuk össze. A II. rész azzal foglalkozik, hogy e két utóbbi tevékenységre, mely klaszterálási módszerek és milyen taxonomikus mértékek (vagy más néven: proximitás-függvények) alkalmasak.

Másképpen fogalmazva, az adatbázis-tervezésen belül hozzáférés-elemzés és ehhez az eszközként alkalmazott klaszterelemzés kérdéskörét tárgyaljuk.

Olyan módszert ismertetünk, amely adott funkciók alapján lényegében egy újabb — logikai szintű — adatmodellt hoz létre. A módszer által kezelt adatmodellben a tulajdonságtípusok egyrészt az adattípusok, másrészt a funkciók csoportosításában szerepelnek. Az így létrehozott logikai szintű adatmodellből kiindulva, az area-tervezés előkészítő lépéseként, kiinduló csoportosítást kívánunk elérni a szegmenstípusok és funkciók figyelembevételével a fizikai tervezés során.

Ebből a fenti megfogalmazásból nyilvánvaló, hogy a javasolt módszer az adatbázis tervezésének logikai és fizikai szintjét érinti. A módszer segítségével előállított modellnek megfelelően, a kiinduló az elvi adatmodell pontosítható, és csupán ennek alapján megalapozhatjuk a logikai adatmodell első változatát. A módszer alkalmas arra is, hogy segítséget nyújtson az ABKR kiválasztásában, a fizikai tervezés szintjén az area-tervezés elvégzéséhez döntési szempontokat adjon és azt előkészítse. A módszernek nagy előnye, hogy számítógépen egyszerűen megvalósítható.

1.2. A séma megtervezése

Hálós adatmodellekből úgy lesz séma, ha az egyedítípusokból rekordtípusok, a tulajdonságtípusokból adattétel-típusok, a kapcsolattípusokból eset-típusok lesznek, és kialakulnak mindazok a tényezők, amelyeket a DDL

és DMCL segítségével le kell írni az adatbázis generálását megelőzően. Célszerű az egyedtípusokat eleve megbontani, azaz egy egyedtípust több szegmenstípussá alakítani. Ez úgy történik, hogy egyedtípusonként egy-egy ún. C mátrixot kell felépíteni, amely az egyedtípust jellemző elemi adatok iránti adatkezelési igényeket mint becsült gyakoriságot tartalmazza. Az adatkezelési igények súlyozhatók is, figyelembe véve a hozzáférés jellegét. Ezután, tekintettel több különböző egyedtípushoz tartozó szegmenstípus közötti relációra, meg kell állapítani az együtt használható szegmenstípusokat (szegmensosztály), majd a navigációkat figyelembe véve, és az elemi adatredundanciát a lehetőség szerint minimalizálva, egy-egy szegmensosztályra jellemző adatbázisrekordok adattartalmának és szerkezetének meghatározására kerül sor. Végül az adatbázis későbbi bővítésére is tekintettel, rögzíteni kell az adatbázis-rekordokba összevont szegmenstípusokat.

Az ABKR-ek megszabják a lehetőségeket az adatok tárolását illetően, ugyanakkor a tevékenységek meglehetősen széles skáláját elsősorban a hatékonyság szempontjai határozzák meg.

A hatékonyság érdekében, ismét értékelni kell a funkcionális elemzés eredményeit. Ennek az értékelemzésnek közvetlen célja, hogy a felhasználói követelményekhez a legjobban illeszkedő módon legyen meghatározva

- a rekordok elhelyezési módja,
- a set-ek megvalósítási módja,
- az adatbázis area-felosztása, és
- a másodlagos kulcsok kezelése.

Emellett az elemzés eredményei alapján kell olyan járulékos tényezőket is meghatározni, mint pl. az elsődleges vagy másodlagos kulcsok szerint felépülő indextáblázatok elhelyezése.

Az elemzés legfontosabb eszköze a különböző statisztikai kigyűjtések elkészítése. Az egyik legfontosabb ilyen statisztika a set-ek használatának módjára vonatkozik. Azt kell vizsgálni, hogy az egyes funkciók végrehajtása során szükséges navigációk az egyes set-eket döntően a tulajdonostól a tagrekordtípus irányába haladva veszik-e igénybe vagy pedig fordítva. A tagtól a tulajdonos irányába történő mozgás általában annyit jelent, hogy a tulajdonosban megtalálható kulcs az, amire szükség van. Ha egy set döntően ilyen módon használatos, akkor megfontolandó a set megszüntetése, és az eredetileg tagrekordba való elhelyezése. Így kettős eredményt érünk el:

- gyorsabbá válik az adott kulcs megtalálása, hiszen a már egyszer elért rekordban van, és
- egy rekord kevesebb set-ben lesz tag, ezáltal a karbantartási idő csökkenthető.

Ez a megoldás egyúttal továbbra is lehetővé teszi, hogy az eredetileg tagrekordból kiindulva — az ott elhelyezett kulcs alapján — továbbra is direkt módon legyen elérhető az eredetileg tulajdonosrekord, ha azt az adott kulcs szerint CALC elhelyezési módúra definiáljuk. Természetesen ezzel egyidejűleg az eredetileg tulajdonosrekord típustól a tag felé irányuló navigáció jelentősen megnehezül. A kérdés tehát valóban az, hogy a döntő felhasználói igény milyen jellegű. A fenti kérdések eldöntéséhez közvetlenül a 29., illetve közvetve a 26., 27., 28., táblázatok adnak segítséget.

Egy további statisztika az egy rekordtípushoz való hozzáférés jellegére vonatkozik. Itt azt kell megvizsgálni, hogy melyek azok a rekordtípusok, amelyekhez direkt hozzáférést kell biztosítani — ezek a navigációnál általában a belépési pontok — és melyek azok, ahol elegendő, ha set-en keresztül érjük el az okot, illetőleg ha szekvenciális hozzáférést kell biztosítani. Az elemzés eredményeképpen minden egyes rekordtípusra el kell dönteni, hogy mi kell a LOCATION MODE IS... utasításba (pl. CALC, DIRECT vagy VIA SET). Ehhez a 28. táblázat ad segítséget.

Végül a harmadik fontos statisztika a set-ek tulajdonos-tag irányú használatát elemzi tovább annak érdekében, hogy a set tényleges megvalósítási módját meghatározhassuk a (POINTER ARRAY, CHAIN, LIST) MODE IS... utasításban és eldönthessük a set-en belüli tagrekord-előfordulások sorrendjét, az ORDER IS... utasításban (pl. SORTED, FIRST, LAST, NEXT stb.)

Ki kell gyűjteni, hogy mely rekordtípusokat kell együtt használni, és ez lesz a realm- (vagy area) felosztás alapja; ez a statisztika készíti elő az alsémák kijelölését is.

Mindezek eredményeképp megalapozottan eldöntenthető

- a rekord elhelyezési módja,
- a set-ek megvalósítási módja,
- az adatbázis arae felosztása, és
- a másodlagos kulcsok kezelése.

Igen célszerű, ha a rekordtípusok és set-ek dokumentálására kialakított dokumentumokon elvégezhető a tervezés egyes eredményeinek dokumen-

tálása, méghozzá oly módon, hogy a bizonylat egyes rovatainak olvasása sorrendjében, és tartalmában, lehetővé tegye az adatleíró utasítások írását. Ilyen bizonylatot mutatunk be példaként a 15. és 16. táblázaton. Ezek a lapok a Simens USL adatbáziskizsgáló rendszerben használható Schema DDL és SSL (DMCL) nyelvekhez alkalmazkodnak. Kis módosítással alkalmassá tehetők más CODASYL- ajánlásokat követő ABKR-nél történő felhasználásra is.

Egy lényeges megjegyzés erejéig vissza kell térni a funkcionális elemzésre, ugyanis az elemzés eredményeinek kiértékelése az adatmodell megváltoztatásához vezethet. Egyes set-ek megszűnhetnek, ún. system set-ek hozhatók létre, ezzel együtt bizonyos adattartalombeli változások történnek stb. Mindez azt eredményezheti, hogy a funkcionális elemzések újabb ismételése válik szükségessé, a módosítások figyelembevétele céljából. Az elemzéseknek ez az iteratív jellegű végrehajtása, az új változatok elkészítése, felhívja a figyelmet a dokumentációra. Tovább fokozódik a dokumentáció jelentősége akkor, ha nagy a modell és sok az elemezhető funkció. Feltétlenül gondoskodni kell tehát arról, hogy mindig világosan megkülönböztethető legyen, hogy melyik dokumentum tükrözi a naprakész állapotot.

1.3. Teendők a hozzáférési igények meghatározásának szakaszában

A logikai tervezés kiindulópontja a már funkcionálisan elemzett elvi adatmodell. A funkcionális elemzés szakaszának legfőbb tevékenységei:

1. Az elemzés előkészítése:

- a funkciók összegyűjtése,
- adatnevek egyeztetése,
- a tulajdonságtípusok meglétének ellenőrzése,
- az adatmodell esetleges kiegészítése,
- mértékadó funkciók kiválasztása és elemzése.

2. Az elemzés végrehajtása mértékadó funkcióként:

- belépési pont meghatározása,
- a funkció ellátához szükséges egyed típusok sorrendjének megjelölése,

- egyed típusonként az előfordulások aktuális részhalmazának és az érintett tulajdonságtípusokon végzendő műveleteknek a rögzítése.
3. Az elvi adatmodell esetleges koordinációja:
- elvi adatmodell kiegészítése,
 - navigációs utak korrekciója,
 - a funkcióleírások ellenőrzése.
4. Az elemzés eredményeinek összesítése egyed típusonként:
- az elemzett funkciók ellátásához szükséges valamennyi egyed típus (ezen belül tulajdonságtípusaik) összegyűjtése,
 - egyed típusonként (ezen belül tulajdonságtípusokra lebontva) a hozzáférési igények összesítése.

A funkcionális elemzés céljai:

- a normalizált adatmodell pontosítása (1. és a 3. tevékenységcsoportok),
- az adatmodellhez kapcsolódó algoritmusmodellek megalkotása (2. tevékenységcsoport),
- a logikai adatbázis-tervezés előkészítése (1. utolsóként felsorolt tevékenysége és a 2., 4. tevékenységcsoportok).

Az elemzés előkészítését közvetve mértékadó funkciók kiválasztása, és közvetlenül a mértékadó funkcióként végrehajtásra kerülő elemzés szolgálja.

Az elvi adatmodell esetleges korrekciójának elvégzése után a funkcionális elemzés eredményeit egyed típusonként csoportosítva összegezni kell. Tekintettel arra, hogy a funkcionális elemzés céljára kiválasztott mértékadó funkciók általában nem érintik az adatmodell egészét, valamint arra, hogy egy adatbázis kialakításában egyébként is célszerű a fokozatosság, indokolt, hogy az első lépésből csak azokat az egyed típusokat vegyük figyelembe, amelyeket a funkcióelemzés is érintett. Ezeket így az adatmodell egészét képező egyed típus-halmazokból ki kell gyűjteni. Hasonló okokból célszerű döntés az, hogy az egyes kigyűjtött egyed típusokból a logikai tervezéshez csak a funkcióelemzések során érintett tulajdonságtípusokat hagyjuk meg. Ebben az utóbbi szelekciós kérdésben még tovább is mehetünk, pl. helyezkedhet a tervező olyan álláspontra, hogy csak azokat a tulajdonságtípusokat vonja majd be a logikai tervezésbe, amelyekkel szemben előreláthatóan nem változik meg a felhasználói igény, formátumuk pontosan definiálható, kezdeti betöltési értékeik ismertek, illetve folyamatos karbantartásuk biztosítható.

Az anyag további részében az adatmodell egyedtípusai halmazán, illetve tulajdonságtípusai halmazán és kapcsolattípusai halmazán a „leszabás” után megmaradó halmazt értjük. Az egyedtípusok fenti értelemben vett J -elemű halmazát $\{E\}^J \neq 0$ jelöli. A tulajdonságtípusok Q -elemű halmazát pedig $\{A\}^Q \neq 0$, míg a mértékadó funkciók a R -elemű halmazát $\{F\}^R \neq 0$ jelöli. A hozzáférési igények összesítése egyedtípusonként egy-egy táblázat kitöltését jelenti. A táblázat sorai az egyes egyedtípusokhoz az elvi adatmodell alapján rendelt tulajdonságtípusokat $\{A\}^M \subseteq \{A\}^Q$ jelölik. A táblázat oszlopai az egyedtípus tulajdonságtípusait igénylő funkciókat képviselik $\{F\}^N \subseteq \{F\}^R$. Az E_j egyedtípusra vonatkozó hozzáférési igények modelljét első közelítésben bináris mátrixként hozhatjuk létre (amelyet a későbbiekben $\overline{C}_j(M, N)$ mátrixnak nevezünk), $E_j \in \{E\}^J$. Ekkor a $\overline{C}_j$ mátrix az egyedtípusra irányuló igényeket meglehetősen durván tükröző modell. A mátrix egy eleme. $c_{mn} \in \overline{C}_j$, ekkor még csupán azt fejezi ki, hogy az F_n funkció ellátásához szükség van-e vagy sem az A_m tulajdonságtípus valahány előfordulására.

$$c_{mn} = \begin{cases} 1, & \text{ha } F_n \text{ ellátásához } A_m \text{ szükséges} \\ 0, & \text{egyébként,} \end{cases}$$

ahol $m = 1, 2, \dots, M$ és $n = 1, 2, \dots, N$, valamint fennáll $F_n \in \{F\}^N$ és $A_m \in \{A\}^M$.

Ez a modell akkor lesz részletesebb (második közelítés), ha figyelembe tudjuk venni a C_j mátrix egyes oszlopaire vonatkozó súlyozó tényezőket. Három súlyozó tényezőt veszünk figyelembe:

- az F_n funkció súlyát,
- az E_j -re irányuló keresések számát (F_n esetére)
- az E_j -re irányuló adatkezelés célját (F_n esetében).

Az $F_n \in \{F\}^N$ funkciókat (ahol $n = 1, 2, \dots, N$), rendre az S_{F_n} súllyal látjuk el; ekkor a $\overline{C}_j$ mátrix egy oszlopon belül egyazon súllyal ellátott bináris mátrixként kezelhető. Ha az is rögzített, hogy az adott F_n ellátásához a szóban forgó E_j egyedtípusra vonatkozó egyedi, teljes vagy ismételt keresésre van szükség, akkor ez az N -edik oszlopra vonatkozó súly értékét módosítja, mégpedig úgy, hogy az S_{F_n} súlytényezőt a keresések relatív becsült gyakoriságával (S_{gy}) megszorozzuk. Hasonló hatása lesz annak, hogy az E_j egyedtípusnál az F_n funkciót tekintve az adatkezelés célja visszakeresés, hozzáférés vagy törlés.

A fenti három tényezőből képezhető a $\overline{\overline{C}}_j$ mátrix n -edik oszlopára vonatkozó súly, tehát:

$$S_n = S_{F_n} \cdot S_{gy} \cdot S_a.$$

A gyakorlatban az elemzés végrehajtására fordítható időalap korlátai és az elemzésre kijelölt mértékadó funkciók nagy száma vagy bonyolultsága, esetleg az információk begyűjthetősége nem mindig teszi lehetővé, hogy a hozzáférési igényeket ennél részletesebben, pontosabban rögzítsék. Természetesen kívánatos, hogy a helyzetfelmérés során, az F_n esetében az E_j tulajdonságtípusain végzendő műveletek is rögzítettek legyenek. Ebből ugyancsak súlyozó tényező, S_m képezhető, amely azonban már nem a $\overline{\overline{C}}_j$ mátrix n -edik oszlopára, hanem csak az oszlop megfelelő elemeire vonatkozik. Ha a már előbb tárgyalt oszlopokra vonatkozó súlyok mellett az utóbbi elemekre vonatkozó súlytételeket is figyelembe vesszük, akkor a $\overline{\overline{C}}_j$ mátrix már nem tekinthető egyszerűen egy oszlopként súlyozott bináris mátrixnak, hanem — harmadik közelítésben — csak egy eleve nem bináris mátrixnak, amely már kellő pontossággal modellezi az egyedtípusra vonatkozó hozzáférési igényeket. Az első közelítésű modell általában nem alkalmas a logikai tervezés megalapozásához, de a második és harmadik közelítésű modellek erre már alkalmasak.

1.4. A logikai adatbázis-tervezés menetén belül áttekintett tevékenységek

A logikai adatbázis-tervezés menete során, a hatékony működés érdekében, az elvi adatmodellen rendszerint változtatásokat kell végrehajtani. Ezek leggyakrabban:

1. Az elvi adatmodell ABKR által nem kezelendő részeinek „leszabása”
2. ABKR kiválasztása.
3. A kiválasztott ABKR korlátainak és lehetőségeinek figyelembevétele.
4. A hatékonyság és biztonság érdekében alkalmazott kompromisszumok figyelembevétele.

Eltérő tervezési szituációk (és így sorrendek) is léteznek, melyek közül gyakori az 1,4,2,3, valamint 2,1,4,3 sorrend is. Az elvi adatmodell nagylép-tékű átalakítása az 1,2,3 szakaszokban végezhető el, az egyedtípuson belüli változtatások a 4. szakaszban hajthatók végre. A leszabás során az elvi adatmodellnek az ABKR által kezelni kívánt részét el kell választani a más módon kezelni kívánt részeiktől. Az értékelés további részében a rövidség kedvéért az „elvi adatmodell” elnevezés a már „leszabott” elvi adatmodell-részt fogja jelenteni.

A logikai adatbázisterv elkészítése során a már leszabott adatmodell is átalakítható. Erre egyrészt azért van szükség, mert az adott esetben alkalmazásra kerülő ABKR általában csak korlátozásokkal képes a normatívtól és funkcionálisan elemzett kész elvi modell kapcsolatait kezelni. Másrészt, ettől függetlenül, az átalakítás azért válhat szükségessé, mert akár az elvi, akár az átalakított modell kezelése általában nem lenne eléggé hatékony. Ha az egyedtípus tulajdonságtípusainak van olyan részhalmaza, amelyre azonos hozzáférési jogosultság vagy gyakori együttes használat jellemző, akkor ezt az egyedtípust nem egyetlen rekordtípusba, hanem e csoportnak megfelelő részrekordtípusokba (a továbbiakban : szegmenstípusokba) képezzük le.

Ha ugyanazon egyedtípusra (vagy azok meghatározott szegmenstípusa-ira) a gyakori együttes használat jellemző, akkor a szóban forgó szegmens-típusok (esetleg az egyedtípusok) összevonását célszerű elvégezni.

A fenti tevékenységeknek háromféle hatása van:

- a fizikai tevékenység jobban előkészített, így a file-előkészítés eredménye is csak jobb lehet,
- a hatékonyabban kezelhető adatbázis érdekében tudatos az eltérés az adatmodell normalizált állapotától,
- nem csupán az egyedtípusok, hanem a kapcsolattípusok megbontása vagy összevonása is szükségessé válhat.

1.4.1. Egyedtípusok leképezése szegmenstípusokba

A szegmensképzés lényeges jellemzője, hogy az adatbázistervező nem csu-pán a normalizált adatmodell kapcsolatait, hanem pl. az egyes funkciók adatigényeit és a lekérdezési gyakoriságokat is figyelembe veszi. Célszerű le-het szegmensek kialakítása pl. az adatbiztonság, illetve az adatkarbantartás

szempontjai miatt, hiszen a szegmens-szintű elérés egyúttal szegmensszintű védelmet is lehetővé tesz. Egy már működő adatbázis környezetében változik a funkciók köre, ezek adatigénye, fellépésük gyakorisága, feldolgozásuk prioritása is. A szegmenstervezés számára ez azt jelenti, hogy a későbbiekben várható bővülések lehetőleg új szegmenstípusok legyenek, és ne a meglévő szegmenstípusok belső tartalmát és szerkezetét kelljen változtatni. Szempont az is, hogy ha kevesebb szegmenstípus van, könnyebb lehet egy program kódolása és a kevesebb elérés miatt rövidebb a program futása, viszont a szegmensek egy konkrét program által nem is használt adatokat „cipelnek magukkal”, amelyek mozgatása, lebontása felesleges teher.

Számos gyakorlati szempont merül fel, mint láttuk — amelyet az adott helyzetnek megfelelően kisebb vagy nagyobb súllyal kell figyelembe venni, és amelyek abba az irányba hatnak, hogy még az egyes adatok elérését is lehetővé tevő ABKR alkalmazása esetén is képezzünk szegmenseket (esetleg itt egyes szegmenstípusokat csupán egy tulajdonságtípus alkot). A szegmensképzés eredményeképp egyed típusonként megkapjuk az egyed típust jellemző tulajdonságok diszjunkt (közös elem nélküli) részhalmazait. Az adatmodellt alkotó $\{A\}^Q$ tulajdonságtípus-halmazt ugyanis leképezzük egy, a szegmens-típusok alkotta T -elemű $\{Sz\}^T \neq \emptyset$ halmazra, mégpedig úgy, hogy $\{A\}^Q$ egyes elemei esetleg több szegmenstípushoz is tartozhatnak, azaz a leképezés nem egyértelmű. Másképp fogalmazva, fennáll a szegmenstípusokká leképzett adatmodellekre nézve az, hogy

$$\exists Sz_t, Sz_z \in \{Sz\}^T [(Sz_t \cap Sz_z \neq \emptyset) \wedge (Sz_t \not\subseteq Sz_z) \wedge (Sz_z \not\subseteq Sz_t)],$$

ahol $t = 1, 2, \dots, T$ és $z = 1, 2, \dots, T$, valamint $t \neq z$, emellett viszont az egyes egyed típushoz tartozó W -elemű $\{Sz\}^W$ halmazra nézve, ahol $\{Sz\}^W \subseteq \{Sz\}^T$ is fennáll az, hogy

$$\forall Sz_u, Sz_v \in \{Sz\}^W [(Sz_u \cap Sz_v = \emptyset) \vee (Sz_u \subseteq Sz_v) \vee (Sz_v \subseteq Sz_u)],$$

ahol $u = 1, 2, \dots, W$ és $v = 1, 2, \dots, W$ valamint $u \neq v$.

A tulajdonságtípusoknak a szegmensképzéssel nyert, ugyanahhoz az egyed típushoz tartozó diszjunkt részhalmazai egyikében helyezkedik el az egyed típus azonosítója. A kapott többi szegmenstípusba ugyancsak be kell

vinnünk — az egyedtípushoz tartozást kifejezendő — az egyedtípus azonosítóját. Ez történhez úgy, hogy az azonosítót megismételjük, de úgy is hogy azt a pointer képviseli. Ez a gyenge logikai redundancia egy külön esete, ahol a kapcsoló tulajdonságtípus mindkét szegmenstípus azonosítójával megegyezik. A szegmensekre bontott fölérendelt vagy alárendelt egyedtípust jellemző kapcsolattípus megbontása is szükségessé válhat. Az eredeti (szétbontás előtti) egyedtípust jellemző fölérendeltségi kapcsolattípusok azon a (szétbontás utáni) szegmenstípuson keresztül valósulnak meg, amely a kapcsoló tulajdonságtípust ténylegesen tartalmazza. Az eredeti egyedtípust jellemző alárendeltségi kapcsolattípusok a szétbontás után kapott valamennyi szegmenstípus esetében megmaradnak. A szegmensképzés eredményeképp tehát megkapjuk egyrészt a szegmenstípusok halmazát, másrészt a szegmenstípusok közötti kapcsolattípusok halmazát.

1.4.2. Szegmenstípusok összevonása

A szegmensképzés befejezése után elképzelhető, hogy összevonásokat kell majd végrehajtani. Ennek több célja is lehet.

Az első cél: a hatékonyság érdekében ezen a módon legyen korlátozva, hogy az adatmodell egészére nézve a kapcsolattípusok darabszáma mennyivel haladja meg a szegmenstípusok számát (egy szegmenstípus legfeljebb 3-4 adategyüttesben lehet tag).

Általában cél annak korlátozása, hogy egy szegmenstípus hány adat-együttesben legyen tag. Hogy ez fontos vagy mellékes cél-e, az az adott esetben megváltoztatott ABKR azon tulajdonságától függ, hogy gyorsan vagy lassan képes-e kezelni nagymennyiségű új tagelőfordulás beillesztését (törlését). Ha az ABKR ilyen tekintetben gyors, akkor ez a cél nem túl lényeges és a szegmenstípusok összevonásának fő célja inkább az area-képzés segítése lesz. A szegmenstípusok összevonásának következménye az, hogy a kapcsolattípusok tekintetében is változásoknak kell bekövetkezni.

- A) Ha két szegmenstípus ugyanazon egyedtípus szegmenstípusa volt, vagy ez nem áll ugyan fenn, de kölcsönös függésben vannak, akkor
- valamelyik szegmenstípusban lévő azonosító megszűnik,
 - így a kapcsolattípus is megszűnik,

- ha a két szegmenstípus fölérrendeltjei között van azonos szegmenstípus, akkor az egyik fölérrendeltségi kapcsolattípus szemantikai tartalma változik,
 - a két szegmenstípus alárendeltjei egyaránt az új, összevont szegmenstípus alárendeltjei lesznek.
- B) Ha a két szegmenstípus eltérő egyedtípushoz kötődik és közöttük funkcionális függés áll fenn, akkor
- a) az alárendelt szegmenstípusba bevonva a fölérrendeltet:
 - mindkét szegmenstípus fölérrendelt kapcsolatai továbbra is megmaradnak,
 - az összevont szegmenstípus azonosítója a két szegmenstípus azonosítójából kimetszett közös azonosító rész lesz,
 - és ezért megváltoznak mindkét szegmenstípus alárendelt kapcsolatai is,
 - b) a fölérrendelt szegmenstípusba bevonva az alárendeltet:
 - a fölérrendelt szegmenstípus fölötti kapcsolattípusok nem változnak,
 - a fölérrendelt szegmenstípusok egyéb alárendelt kapcsolatai változatlanok,
 - az alárendelt szegmenstípusok alárendelt kapcsolattípusai megváltoznak, mert a kapcsolatot most az új, összevont szegmenstípus azonosítója jelenti.

A második cél: az egyes adatkezelési funkciók által kezelt adatmodellek közötti átfedéseket felismerve, a nagymértékben átfedő adatmodellekből viszonylag homogén csoportokat képezve meg kell határozni, hogy mely funkciókhoz lehet majd közös alsémát definiálni. Ezzel a programozási munka is csökkenthető.

A harmadik cél: area (realm) -felosztást kell végezni. Ez azt jelenti, hogy szegmenstípusokat kell nagyobb egységekbe, area-kba csoportosítani.

A negyedik cél a harmadikkal szorosan összefonódik; az area-nak partíciókba, file-okba való leképezését jelenti azon ABKR -ek esetén, ahol partíciók, ill. file-ok definiálhatók.

Mind a négy fenti tevékenység a szegmenstípusok csoportosításának adata. Ezekben közös még az is, hogy alapvetően annak alapján végezhető el, hogy a szegmenstípusokat mely adatkezelési funkciók használják. Eltérő jellegzetesség, hogy míg az első és másodikként említett tevékenységek a

logikai tervezés, addig a harmadik és a negyedik a fizikai tervezés menetében kerülnek sorra. A szegmenstípusok összevonása tehát nem csupán az adatbázis logikai, hanem a fizikai tervezésének tevékenységeit is segíti.

1.5. A fizikai adatbázis-tervezés menetén belül áttekintett tevékenységek

A fizikai adatbázis-tervezés kiindulópontja a logikai adatmodellezés és funkcionális elemzés során feltárt adatok. Ezek közül a leglényegesebbek: a hozzáférési igények második (esetleg harmadik) közelítésű modelljét alkotó tényezők. A fizikai tervezés menete két szakaszra bontható:

1. Hozzáférési módok tervezése egy szegmenstípus esetében. Ez a szakasz az alábbi tevékenységekre tagolható:

- meg kell állapítani, hogy a kiválasztott ABKR esetében hányféleképpen és milyen módon oldható meg a teljes, az egyedi, és az ismétléses keresés; ezen belül azt is meg kell állapítani, hogy melyek lehetnek az ABKR standard rekordtípus-elhelyezési módjai,
- a hozzáférési igények modelljéből kiindulva meg kell határozni a szegmenstípus alkalmas elhelyezési módját,
- adategyüttesenként külön-külön el kell dönteni a használandó pointer-típust,
- meg kell határozni az adategyüttesen belüli tagrekord-előfordulások logikai rendezettségének fajtáját,
- meg kell határozni az adategyüttes tagsági viszonyai változásának kezelési módját.

2. A tárolási szerkezet meghatározása. Ez a szakasz az alábbi tevékenységekre tagolható:

- az adatbázis felosztása area-kra,
- az area-k leképezése particiókba, ill, file-okba (ez nem minden ABKR esetén kell),
- az area-kat alkotó page-ek méretének meghatározása,
- az adatbázis helyének kiszámítása.

Meg kell jegyezni, hogy a részletes fizikai tervezést jelentősen befolyásolja.

- a logikai adatbázis-terv,
- az adatbázis integritásának biztosítása,
- az adott ABKR lehetőségei,
- a rendelkezésre álló tárolók fajtája,
- az alkalmazandó programnyelv (host-language) adatkezelési lehetőségei.

Éppen ezért, a logikai adatmodellre építve csupán a tárolási szerkezet és a hozzáférési mód tervezésének indulásához tudunk algoritmizált segítséget adni.

1.5.1. Elhelyezési mód meghatározása

Példaként tekintsük kiválasztott ABKR-nek az IDMS-t. Itt három standard elhelyezési mód van (LOCATION MODE):

- CALC; célja: a szegmens-előfordulások egyenletes eloszlását véletlen szerűen lehet biztosítani az area-n, továbbá szimbolikus kulccsal közvetlen elérés biztosítható.
- VIA SET; célja: lehetőleg minél közelebb legyenek egymáshoz azok a szegmenstípusok, amelyeket együtt kívánnak elérni. Közvetlen elérés nem lehetséges.
- DIRECT; célja: adatbáziskulccsal közvetlen elérés biztosítható.

Az egyedi keresés megoldható:

- közvetlenül (DIRECT),
- véletlen hozzáféréssel (CALC).

A teljes keresés megoldható:

- egyedi közvetlen elérés ismétlésével,
- az area soros keresésével,
- singular set létrehozásával, majd elérésével. Ismételt keresés megoldható;
- főlérendelt egyed-előfordulásból az alárendelt egyed-előfordulás elérésével (adategyüttes létrehozása szükséges, a tagrekordot általában VIA, a tulajdonosrekordot CALC módon célszerű elhelyezni),
- egy egyedtípus másodlagos ismérv azonos értékével rendelkező előfordulásainak elérésével (indexeléssel, IDMS Sequential Processing Facility (SPF)-jével).

A tervezés kiindulópontja az, hogy adatmodellenként meghatározható egy szegmenstípusok alkotta sorrend, amelyek az adatmodellt tekintve ún. precedenciahálót alkotnak. A tervezőnek egyrészt a szegmenstípus egy precedenciahálóban elfoglalt helyét és azt figyelembevée, hogy a szegmenstípus elérése iránti igények között van-e közvetlen elérési igény, másrészt a tulajdonostag lánc hosszát és a tagrekord elérése iránti igény gyakoriságát figyelembe véve lehet dönteni. Megfogalmazható néhány általános irányelv az elhelyezési mód kiválasztására nézve:

- csak azokat a szegmenstípusokat szabad VIA SET módon elhelyezni, amelyek közvetlen elérésére soha nincsen szükség,
- egy adatmodell hierarchiába rendezhető elérési precedencia-sorrendet is képvisel. A hierarchia legfelső szintjén nem lehet VIA, annál inkább szükséges CALC vagy DIRECT elhelyezés. A legalsó szinteken viszont éppen fordított a helyzet.
- a közbenső szinteken csak akkor célszerű VIA alkalmazása, ha az alsó szinteken ez nem jelent majd hosszú elérési időt.

A fenti irányelvből még általában nem válhat egyértelmű szabály, mert az egy szegmenstípusra irányuló hozzáférési igények sokszor egymást kizáró elhelyezési módokat követelnek. Az eligazodás céljából először össze kell gyűjteni az egy szegmenstípusra irányuló hozzáférési igényeket. Ez szoros kapcsolatban van az area-leképezéssel, amelynek során össze kell gyűjteni azokat a szegmenstípusokat, amelyek a rájuk irányuló hozzáférési igények szempontjából egymáshoz közelieknek tekinthetők.

1.5.2. Az adatbázis felosztása area-kra

Szegmenstípusokat többnyire akkor rendelhetünk ugyanabba az area-ba, ha ezeket ugyanazon adatkezelési funkciók használják. Szempont lehet az is, hogy így pl. az IDMS SPF-je alkalmazása esetén az indexeknek és a szegmens-előfordulásoknak külön helyen kell lenniük. A válogatás alapja, hogy az egyes area-kba kerülő szegmenstípusokat adatkezelési funkciók egy csoportja szegmenséből nézve homogén részhalmazt alkossanak. Mielőtt ezt az összeválogatást elkezdenénk, végre kell hajtani a szegmenstípusok ún. véglegesítését. Annak, hogy a szegmenstípusok szerkezete nem végleges, az az oka, hogy létezik egy független, az adatok karbantartásáért felelős, valamint az adatok elérését korlátozó, adatbiztonsági szegmensrendszer,

amelyek helyzetfelmérés után közvetlenül megadják, hogy szempontjuktól mely adatípusokat lenne célszerű vagy éppenséggel kötelező együtt kezelni. Esetleg további szempontrendszer(eket) is figyelembe kell venni. Az adott feladattól és környezettől függően más és más e szempontok egymáshoz viszonyított súlya. Minden felvett szempontrendszer egy feltehetően eltérő szerkezetet (szegmensfelosztást) tükröz. A rendszer tervezőjének a feladata, hogy egyedítípusonként áttekintve — mondjuk három szempont szerint — az egymás melletti szerkezeteket, véglegesítse a szegmenstípusok szerkezetét. E véglegesítéshez támaszt adhat az, ha pl. egy képernyőn egymás mellett (több szempont szerinti) többféle szerkezet úgy jelenhet meg, hogy a tervező könnyen áttekintse, s viszonylag gyorsan lehessen a véglegesítésről dönteni. A több szempont szerint képezett szerkezetek figyelembevételével kialakított eredő végleges szerkezet meghatározása most nem feladat. Ha matematikailag korrektül kell meghatározni az eredő végleges klaszterszerkezetet, akkor általában azt mondhatjuk, hogy a modellezőnek hasonlósági vagy távolsági függvényértékeket kell kezelnie. Ennek három útja létezik:

- folytonos térbeli modell esetén MDS (többdimenziós skálázás) módszerek pl. INDSCAL, PARAFAC-2,
- diszkrét, nem térbeli modell esetén klaszteráló módszerek pl. INDCLUS,
- többszörös fák és hibrid módszerek (multiple-trees), pl. INDTREES.

A gyakorlatban a megoldást leginkább a többszörös fák modellje és hibrid módszerek köréből célszerű választani.

Ezek a módszerek új kutatási irányt jellemeznek; alkalmazási módjait itt nem tárgyaljuk, csupán röviden utalunk rá. A továbbiakban az egy csoportba vagy osztályba kerülő szegmenstípusok meghatározásához a kiindulópont az, hogy az előbb említett véglegesítés eredményeképpen az egy szegmenstípus tulajdonságtípusai rögzítetté válnak. Ezt követően, az egy funkció szempontjából vett tulajdonságtípusok egy részhalmazát jellemző adatmodell helyett a szegmenstípusok megfelelő részhalmazát is tekinthetjük adatmodellnek, de megkülönböztető jelzővel látjuk el és szegmensszintű adatmodellnek nevezzük. A szegmensszintű adatmodellek összessége táblázatba rendezhető. A táblázat egyes oszlopai az adatmodellt kezelő funkciókat képviselik, sorai pedig a szegmensképzés eredményeként meghatározott szegmenstípusokat. A táblázat neve $\overline{B}(T, R)$ mátrix és egy eleme $b_{tr} \in \overline{B}$

azt fejezi ki, hogy az F_r funkció ellátásához szükség van-e vagy sem az Sz_t szegmenstípusban lévő valamennyi tulajdonságtípus előfordulására.

$$b_{tr} = \begin{cases} 1, & \text{ha } F_r \text{ ellátásához } Sz_t \text{ szükséges} \\ 0 & \text{egyébként,} \end{cases}$$

ahol $t = 1, 2, \dots, T$ és $r = 1, 2, \dots, R$, valamint fennáll $F_r \in \{F\}^R$ és $Sz_t \in \{Sz\}^T$

A táblázat r . oszlopában $b_{tr} = 1$ elemei alkotják az F_r -hez tartozó szegmensszintű adatmodellt. A $\bar{B}$ mátrix sorainak és oszlopainak permutálásával kialakuló, közel diagonális alakú mátrixot, más néven ún. lépcsőház-mátrixot képezhetünk (1. táblázat), amelyben jól kivehetők az egymástól többé-kevésbé elhatárolódó blokkok, az ún. szegmensosztályok. Általában egy szegmensosztály alkot majd egy area-t. Ezen túlmenően, lehetőség lesz arra, hogy a tervező a szegmensosztályok közötti átfedéseket figyelembe véve hajtsa végre az area file-okba való rendezését abban az esetben, ha az alkalmazott ABKR-ben ez szükséges. A file-ok tartalmának és szerkezetének meghatározását az ún. mértékadó adatbázisrekordok kiválasztásával lehet elvégezni. A tulajdonságtípusnak szegmenstípusba, azoknak pedig area-kba (és file-okba) csoportosításakor figyelembe kell venni, hogy az adatbázis egészét tekintve minimalizáljuk a szegmenselőfordulások elérési idejét.

A hozzáférési modellre épülő optimalizálás esetében azonban tudomásul kell venni, hogy amíg az S_{F_r} és az S_{gv} súlytényezők értékeit a felhasználó adatigényéből kiindulva meg tudjuk határozni, addig az S_a és a S_m súlytényezők konkrét értékeit (ezek arányszámok) nagymértékben meghatározzák az implementáció (a géptípus, a háttértárolók, az ABKR, a választott file-szerkezetek) részben már adott, részben pedig a fizikai tervezés későbbi szakaszában még eldöntendő körülményei. Ezért a szegmensképzés az area-képzés olyan, a fizikai tervezés későbbi menetéből visszacsatoló, többfordulós folyamat, amelyben mindig jelentős tere marad a visszacsatolások miatt felmerülő korrekciók, s azok hatásai mérlegelésének, a tervezői segédeszközöknek.

2. fejezet

Az alkalmazott módszerek köre

2.1. Az osztályozás módszerei

Az adatmodellben szereplő tulajdonságtípusok szegmenstípusokba, majd a szegmenstípusoknak nagyobb egységekbe való nyalábolása osztályozási feladat. Az osztályozás feladatai általában három csoportba sorolhatók:

- adott osztályok és az azokba sorolt egyedek sajátosságaiból az egyes osztályok jellemzőinek meghatározása,
- adott egyedeknek előre megadott osztályokba (nomenklatúrába) történő beosztása (más néven, prekoordinált osztályozás),
- adott egyedek halmaza és az egyedek jellemzői ismeretében az egyedekből alkalmas osztályok (gyakoriak a class, cluster, clump, group, clique elnevezések) meghatározása (más néven, posztkoordinált vagy automatikus osztályozás).

Kizárólag ezzel az utolsó problémakörrel, pontosabban az ún. automatikus osztályozásnak, és ezen belül is a klaszteranalíziseknek (a továbbiakban: klaszterálás) az információs rendszerek logikai tervezésében való alkalmazásával foglalkozunk. A posztkoordinált osztályozás alapja egy csoportképző algoritmus, és nincsen előre megadott nomenklatúra. Az alkalmas csoportképző algoritmus általában valamelyik többváltozós matematikai-statisztikai módszer egyik eljárása. Rendszerint több különböző eljárást is ki szoktak próbálni. A terminológiát illetően, ott, ahol a matematika egyedet mond, az adatmodellezés megfelelő kifejezése éppen a tulajdonságtípus elnevezés, ahol pedig a tulajdonságról beszél, ott az adatmodellezés a funkció elnevezést használja. A pontos értelmezés érdekében ezért az utóbbi fogalomra vonatkozó két elnevezés közül a funkció elnevezést fogjuk használni, de az előbbi fogalomra az adattípus elnevezést használjuk majd, mert a két elnevezés bármelyike zavaró módon keveredne más tartalmú fogalmakkal (pl. egyedtípus).

Előzetes helyzetfelmérés, elemzés után egyedtypusonként van egy-egy táblázat, amelyet az E_j egyedtypus esetében $\overline{C}_j$ mátrixnak neveznek. Keresni kell azt a módszert, amely lehetővé teszi a csoportonkénti osztályozást. Ehhez először át kell tekinteni az osztályozási módszereket, majd elvi és hatékonysági jellemzőiket figyelembe véve, ki kell választani a szegmens-típusok és az area-k képzéséhez alkalmas módszert. A módszer meghatározásával összhangban kell kijelölni az alkalmazható taxonomikus mértéket is.

Tekintettel arra, hogy más módszerek K^3 -nal arányos lépésszámú eljárásra vezetnek, gyakorlati alkalmazásukra nagy adathalmazok esetén ritkán kerül sor; helyettük általában klaszteráló módszereket alkalmaznak. A megfelelő klaszterálási módszer kiválasztásakor igen fontos meghatározni, hogy milyen az adatmodell tulajdonságtípusai által alkotott klaszterek szerkezete. A klasztereket is lehet klaszterálni: kisebb csoportokat nagyobbá összefogni. Ekkor rétegezett klaszterekről, egyébként pedig egyszerű (vagy egyszintű) klaszterekről beszélünk. A gyakorlatban megkülönböztünk diszjunkt (elváló vagy disjoint) és átfedésses (overlapping) klasztereket. Hierarchikus a klaszterek szerkezete akkor, ha elváló és rétegezett. A fenti csoportosítást a 35. ábra illusztrálja. Jardine és Sibson a klaszterek szerkezetét, illetve a klaszterek alapvető lehetőségeit alábbiak szerint határozza meg:

$$C = \{C_1, C_2, \dots, C_k\}$$

a klaszterek halmaza és C_i klaszter, ahol $i = 1, \dots, k$. Egyszerű (simple) klaszterek:

$$\forall C_i, C_j \in C [(C_i \not\subseteq C_j) \wedge (C_j \not\subseteq C_i)],$$

ahol $i \neq j$ és $j = 1, 2, \dots, k$

Rétegezett (compound) klaszterek:

$$\exists C_i, C_j \in C [(C_i \subseteq C_j) \vee (C_j \subseteq C_i)], \text{ ahol } i \neq j$$

Ha bármely két klaszterre fennáll, hogy $C_i \cap C_j = \emptyset$ vagy az egyiket a másik teljes egészében tartalmazza, akkor a klaszterálás elváló, egyébként átfedésses.

Elváló (partitional) klaszterálás:

$$\forall C_i, C_j \in C[(C_i \cap C_j \neq \emptyset) \vee (C_i \subseteq C_j) \vee (C_j \subseteq C_i)],$$

ahol $i = j$;

Átfedésses (overlapping) klaszterálás:

$$\exists C_i, C_j \in C[(C_i \cap C_j \neq \emptyset) \wedge (C_i \not\subseteq C_j) \wedge (C_j \not\subseteq C_i)],$$

ahol $i = j$

A klaszterálás egy sorozatában C^g , ahol g egy orientális vagy intervallum-, illetve arányskálán mért paramétert (hatósági küszöbértéket) jelöl, a sorozat minden egyes lépésében nyert klasztert rétegezett klaszternek nevezzük. Ha a sorozat minden tagja elváló klaszter, akkor a sorozat egészét előállító eljárást hierarchikus klaszterálásnak, egyébként pedig átfedésses (átlapolt) rétegezett (itt egyszerűen csak hierarchikus) klaszterálásnak nevezzük. Az ordinális paraméter segítségével végzett hierarchikus klaszterálást taxonomikus osztályozásnak nevezik.

2.2. Nem hierarchikus klaszteráló módszerek jellemzése

A nem hierarchikus módszerekkel rendszerint diszjunkt klasztereket határozzunk meg. A klaszterek (szegmenstípusok) kívánt minimális száma általában előre is rögzíthető. Vannak módszerek, amelyekre minden egyes szegmenstípusba kerülő adattípusok számát tekintve megadható egy alsó, illetve egy felső korlát. Egymást átfedő klaszterek esetében az átfedés mértéke többnyire szabályozható. Gyakran adott egy optimalizálandó célfüggvény. Szinte valamennyi osztályozandó algoritmus interakív. Az eljárások egy része kiszűri a kivételeket (outliereket). Többnyire nincs szükség hasonlósági együtthatók meghatározására. Számos ilyen módszer azért gyors, mert csak egyszer kell elérnie egy adattípust. A nem hierarchikus módszerek végrehajtásának lépés száma, $K \cdot \log K$ -val vagy ahhoz közelítő értékkel arányos, ahol a K az adattípusok száma. Célszerű azonban figyelembe venni azt is, hogy a szegmensképzés eredménye általában nem független az algoritmus által kezelt adattípusok sorrendjétől és a kapott szegmensek karbantartása miatt gyakran szegmensátszervezésre kerülhet sor. Ez rögtön módosítja az

e módszerek időigényére vonatkozó meggondolásokat is. Ha ugyanis egy klaszterálást követő karbantartás úgy befolyásolhatja a klaszterek elhatárolását, hogy a karbantartás után újra kell képezni a klasztereket, akkor a helyes időbecslés (Van Rijsbergen):

$$\left[\sum_{i=1}^U (K_i \log K_i) \right],$$

ahol $i = 1, 2, \dots, U$ a karbantartások számát, és $K_i - K_{i-1}$ az egy alkalommal felvett, karbantartott adattípusok számát jelenti.

2.3. Hierarchikus klaszteráló módszerek csoportosítása

A hierarchikus klaszteráló módszerek — a típust tekintve — lehetnek

- felülről lefelé építkező (egyszerű divizív),
- alulról felfelé építkező (egyszerű agglomeratív),
- paraméterekkel vezérelt (akár divizív, akár agglomeratív) jellegűek (Fritsche).

Az egyszerű divizív módszer fő lépései (adatbázis-tervezés esetére konkretizált kifejezéseket használva):

1. Mind a K adattípus egyetlen kiinduló szegmenstípusba összefogott.
2. Valamennyi éppen létező (és legalább két adattípust tartalmazó) szegmenstípust kétfelé kell választani (vágás).
3. A 2. lépést ismételni kell addig, amíg valamennyi adattípus külön szegmenstípusba kerül.

Az adott módszerek között főleg az optimális vágás megítélése (mérése) tekintetében van eltérés. Az összes lehetséges felbontás ($2^{K-1} - 1$) áttekintése igen idő- és tárigényes folyamat.

Az egyszerű agglomeratív módszer fő lépései:

1. Mind a K adattípus önálló szegmenstípusnak (klaszternek) tekintendő.
2. Egyé kell olvasztani a két leginkább hasonló szegmenstípust (klasztert).
3. A 2. lépést addig kell ismételni, amíg valamennyi adattípus egy szegmenstípusba kerül.

Az alkalmazott módszerek főleg abban különböznek egymástól, hogy a klaszterek (szegmenstípusok) közötti hasonlóságot (különbséget) milyen függvénnyel adják meg.

A paraméterrel vezérelt módszer fő lépései:

1. Esetleg nem diszjunkt, de homogén klasztereket előállító eljárás alkalmazása.
2. Az 1. lépésben előállított egyes homogén klasztereken belül (pl. a legközelebbi szomszéd keresésével) minimális feszítő fák előállítása.
3. Dendrogram előállítása egy hasonlósági küszöbérték (cut-off paraméter) értékének lépcsőzetes változtatásával (ha a paraméter hasonlósági mérték, akkor monoton növelése agglomeratív eljárás).

A módszer ilyen összetett szervezését az alábbiak indokolják:

A minimális feszített fák elváló (diszjunkt) klasztereket állítanak elő (3. lépés). A minimális feszített fák létrehozásához szükséges a legközelebbi szomszéd típusú keresések (nearest-neighbour, illetve best matching) egyrészt igen időigényesek, másrészt tárigényük is nagy, ezért általában ezt nem célszerű a teljes sokaságra végrehajtani, de egy részsokaságra igen. E részsokaságokat egy előcsoportosítással (1. lépés) hozzák létre, választják el egymástól. Egy másik szempontból, az eljárás jellegét tekintve ez lehet valamilyen

- lánc-módszer
- centroid-módszer és
- hibanégyzet — vagy variancia-módszer.

A hierarchikus klaszterálást végző eljárásokat a nem hierarchikus módszerekhez képest sokan előnyösebbnek ítélik meg az alábbi okok miatt:

- a hierarchikus klaszterek esetén viszonylag hatékonyabb visszakereső eljárások léteznek,
- ha az adatok közötti természetes szerkezet olyan, hogy egymást részben átlapoló klaszterek vannak, akkor a hierarchikus klaszterálás végrehajtása gyorsabb folyamat,
- a hierarchikus klaszterstruktúra tárigénye lényegesen kisebb, mint az egy szinten (átlapolva) rendezetté,
- az ún. kivételeket az agglomeratív módszerek kiszűrjük.
- az agglomeratív klaszteráló eljárások időbeli bonyolultsága (K^2)-tel arányos, de létezik néhány, ennél valamelyest gyorsabb agglomeratív eljárás is.

A karbantartás következtében gyakori átszervezések időigényét is figyelembe véve, a nem hierarchikus módszerek hatékonysága már korántsem múlja felül jelentős mértékben a hierarchikus eljárásokét, mert az ún. egyszerű lánc-módszerekkel képzett klaszterekre fennáll, hogy

- a klaszterálás eredménye független az algoritmus által kezelt adattípusok sorrendjétől,
- viszonylag érzéketlenek a karbantartás során őket ért változásokra.

A hierarchikus klaszteráló eljárások fő hátrányai:

- az eljárások nem feltétlenül biztosítanak optimális megoldást, mert egy korai lépésben végrehajtott összevonás (vagy vágás) véglegesen rögzíti az adattípus klaszterhez tartozását, és ez később sem változtatható,
- az egyszerű lánc-módszerek ún. lánchatást is tartalmaznak,
- a divizív módszerek nem szűrik ki a kivételeket,
- a számítási eljárás vagy hasonlósági vagy távolsági mátrix elemeinek kiszámítását is igényli (így szegmensképzési algoritmus idő-, illetve tárigénye nagyobb),
- a klaszterek kívánt számát nem lehet előre megadni,
- a divizív módszerek idő- és tárigénye gráf-modell alkalmazása esetén nagy.

2.4. A taxonomikus mértékek alaptípusai

Az automatikus osztályozást egy — az adott feladathoz alkalmasan megválasztott — taxonomikus mértékkel operáló algoritmus hajtja végre. A taxonomikus mérték lehet valamely alkalmas távolságmérték vagy hasonló-sági együttható.

2.4.1. Mennyiségi ismérvek (tulajdonságok)

Az adatípusok valamennyi ismérve arány- vagy intervallumskálán mérhető.

Ekkor az adattípusokat jellemző tulajdonságokat (itt funkció) koordinátáknak fogva fel, az általuk alkotott sokdimenziós térben távolságmérték-ként alkalmazható az i -edik és a j -edik adattípus közötti távolságot kifejező euklideszi távolságmérték,

$$L_{ij} = \left[\sum_{n=1}^N (c_{in} - c_{jn})^2 \right]^{1/2}.$$

Ennek kiszámítása azonban időigényes, és ha nem a távolság pontos értékére van szükségünk, hanem annak ismerete is elegendő, hogy két távolság közül melyik a nagyobb (esetleg egyenlő-e), akkor használhatjuk az egyszerűbben számítható „háztömb- menti” (city-block) távolságmértéket:

$$L_{ij} = \sum_{n=1}^N |c_{in} - c_{jn}|.$$

Megjegyezzük, hogy egy távolságmérték csak akkor metrikus, ha fennállnak a következő feltételek:

1. $L_{ij} = 0$ akkor és csakis akkor, ha $i = j$;
2. $L_{ij} \geq 0$;
3. $L_{ij} = L_{ji}$;
4. $L_{ij} \leq L_{ik} + L_{jk}$.

A fentebb tárgyalt két mérték a Minkowski-metrika

$$L_{ij} = \left[\sum_{n=1}^N |c_{in} - c_{jn}|^P \right]^{1/P}$$

$P = 2$ és $P = 1$ eseteinek tekinthető. A P egyéb értékeinek alkalmazása a gyakorlatban ritkán fordul elő.

2.4.2. Minőségi (kategorizált, illetve rendezési) ismérvek (tulajdonságok)

Valamennyi ismérv ordinális vagy nominális skálán mérhető. Nominális skála esetében az egyes koordináták mentén mért változók sajátossága az, hogy értékük csak 0 vagy 1 lehet. Az euklideszi távolságmérték képlete a következő alakra módosul:

$$L_{ij} = \left[\sum_{n=1}^N \delta(c_{in}, c_{jn}) \right]^{1/2}$$

$$\text{ahol } \delta(c_{in}, c_{jn}) = \begin{cases} 1, & \text{ha } c_{in} = c_{jn}, \\ 0 & \text{egyébként.} \end{cases}$$

Gyakran használják a (négyzetes euklideszi távolságmértéknek megfelelő) Hamming-távolságot, mert egyszerűbben számítható:

$$L_{ij} = \sum_{n=1}^N \delta(c_{in}, c_{jn}).$$

Számos klaszteráló algoritmus csak akkor alkalmazható, ill. akkor hatékony, ha bináris változókat használunk.

2.4.3. Vegyesen előforduló minőségi és mennyiségi ismérvek

Ebben az esetben az a korrekt megoldás, ha távolságmértékünk

$$L_{ij} = \left[\sum_{n=1}^N L(c_{in}, c_{jn})^2 \right]^{1/2},$$

$$\text{ahol } L(c_{in}, c_{jn}) = \begin{cases} \delta(c_{in}, c_{jn}), & \text{ha az ismerv ordinális vagy} \\ & \text{nominális skálán mérhető,} \\ (c_{in}, c_{jn}), & \text{ha az ismerv arány- vagy} \\ & \text{intervallum skálán mérhető.} \end{cases}$$

Az egységes mérték érdekében azonban, amikor néhány ismerv kvalitatív, akkor a távolságmértékben hasonlósági együttható csak akkor alkalmazható, ha bináris változókat használunk.

3. fejezet

Egyedtypusok leképezése szegmenstípusokká

3.1. Elvi meggondolások

3.1.1. Alkalmas klaszterálási módszer kiválasztása

Szintetikus adatmodellezés esetén a modellt első közelítésben véges számú, egyenként véges értékészlettel rendelkező tulajdonságtípusok halmazának fogjuk fel, azaz ún. adatelem-katalógust állítunk elő. A következő lépésben a tulajdonságtípusok halmazából (amelyet a rendszer referenciahalmazának tekintünk) kijelöljük az egyedtypusok részhalmazait. Analitikus adatmodellek készítése során e két lépés sorrendje — durván szólva — felcserélhető. Ki kell emelni, hogy adott rendszerben minden egyedtypus meghatározott tulajdonságtípusok halmazával jellemezhető. Egy ilyen halmaz homogén klasztert alkot. Ugyanakkor létezik az ún. gyenge logikai redundancia jelenség, ami azt jelenti, hogy egy egyedtypust jellemző tulajdonságtípus(ok) egyben más egyedtypus(oka)t azonosíthat(nak); ezek az ún. kapcsoló tulajdonságtípusok.

A már előállított elvi adatmodell esetében ismert tehát a kiinduló részhalmazok száma, azok tartalma (a tulajdonságtípusok) külön-külön, és az átfedést jelentő halmazelemek egy része. Átfedés jelentkezik ugyanis az ún. erős logikai redundancia esetében is, ez azonban az elvi adatmodell szintjén nem jelent kapcsolatot. Addig tehát, amíg az átfedés (redundancia) egyes eseteit (a kapcsolattípusokat képező tulajdonságtípusokat) külön is számontarthatjuk, addig más eseteit (erős logikai redundancia) a tervezés során kapcsolatként nem kell figyelembe venni. Ez az egyik oka annak, amiért a nem hierarchikus klaszteráló eljárások alkalmatlanok. Ezek az eljárások, amelyek az átfedések kezelését is lehetővé tennék, nem tudnak különbséget tenni erős és gyenge redundancia között, mindkettőt egyformán

átfedésnek fogják fel. Márpedig a tervező kapcsolatként csak a gyenge logikai redundanciát tartja számon. A másik ok az a követelmény, hogy a szegmensképző algoritmus eredménye legyen független az algoritmus által kezelt tulajdonságtypusok sorrendjétől. Ez ugyanis kirekeszti a nem hierarchikus klaszteráló módszerek többségét, sőt, néhány hierarchikus módszert is.

A hierarchikus klaszterálás alkalmazása akkor előnyös, ha nem a homogén klaszterek elkülönítése a célunk, hanem a klaszterek optimális összekapcsolása. Hálós adatmodell esetében az egyedtypusok közötti átfedések miatt a hierarchikus módszerek nem alkalmazhatók az adatmodell referenciahalmazának egészére, csupán az egyes egyedtypusokra külön-külön, egymás után.

Ezzel megoldódik az is, hogy e módszercsalád időbeli bonyolultságára $O(m^2)$ körüli értékek jellemzők, (ahol $m = 10^3 - 10^4$ a referenciahalmaz elemszáma), és a tárigénye is jelentős. A fenti $O(m^2)$ érték az agglomeratív módszereket jellemzi, a divizív módszerek lépésszáma általában ennél lényegesen nagyobb; éppen ezért ezeket nem is vesszük most figyelembe,

A létrehozott szegmenstypusok érzéketlensége a környezetből jövő változásokra ugyancsak fontos követelmény, és befolyásolja, hogy mely klaszteráló módszereket lenne jó itt alkalmazni. Az adatbázis túl gyakori átszervezését célszerű lenne elkerülni. Lehetőleg olyan szegmenstypusokat kell létrehozni, amelyeket gyakran átrendezni még akkor sem kell, ha változik a folyamatok, adatkezelési funkciók vagy bizonylatok köre, adattartalma, illetve a külső vagy a szervezeti belső szabályozás. Ezen túlmenően is célszerű olyan klaszteráló módszert választani, hogy az egyes adatkezelési funkciók adatigényének nem teljesen pontos meghatározása, majd azt követő pontosítása miatt a szegmenstypusokat minél kevésbé kelljen átrendezni. Ez az előzőhöz hasonló, de más eredetű hatásra vonatkozó zavarérzékenységi-stabilitási követelmény.

Nem csupán a hierarchikus agglomeratív eljárások, hanem valamennyi klaszteráló eljárás közül az egyszerű lánc-módszerek tesznek eleget leginkább az előbb már említett követelményeknek. A lánc-módszer alkalmazásakor azonban az általában kedvezőtlen láncchatás is fellép. Ez kivédhető, ha a lánc-módszert egyidőben csupán a referenciahalmaz egy homogén részhalmazára (egy egyedtypus tulajdonságtypusaira) alkalmazzuk. Az eljárás

egyszerűsíthető, ha a klaszteráláshoz nincs szükség önkényes vagy tapasztalati alapon hasonlósági küszöbértékek felvételére.

3.1.2. Szegmensképző algoritmus ismertetése

Az itt csak röviden ismertetett algoritmus lényege az, hogy a hasonlóságot (illetve távolságot) figyelembe véve, először meghatározza mindegyik adattípus legközelebbi szomszédját (1. fázis). Így olyan, két elemből álló klasztereket képez, amelyek esetleg összefüggő láncszemeknek tekinthetők. Egyszerű lánczá akkor fűzik őket, azaz akkor olvasztják be a klaszterekbe, ha van közös elemük (3. fázis). Ebben az algoritmusban hasonlósági küszöbérték rögzítésére nincs szükség. Mivel az algoritmus az összefűzésekor már nem a kezdő klaszterekben lévő adattípusok hasonlósági mértéke, hanem egy közös adattípus megléte vagy hiánya alapján dönt arról, hogy két kezdő klasztert egybe olvasszon-e vagy sem, hasonlósági küszöbérték felvételére egyáltalán nincs szükség. Az eljárásban először egy legközelebbi szomszédsági relációt kell értelmezni, azaz minden adattípushoz hozzá kell rendelni a hasonlósági $\bar{D}$ mátrix szerint hozzá legközelebb álló másik adattípust, majd ezzel a relációval addig kell a műveleteket végrehajtani, amíg végül ekvivalenciarelációt kapunk.

A javasolt algoritmus alkalmazására bináris változók esetén a 2-5. táblázatok egy számpéldát is végigkövetnek. Az algoritmus egy már felállított $\bar{D}$ mátrixból indul ki, (2. táblázat), és az alábbi négy fő fázisból áll:

1. Fázis:

Célja azoknak az adattípus-pároknak a kijelölése, amelyek majd a 3. fázis elindításához szükséges kezdő klaszterek lesznek, vagyis létrehozni azokat a két-két elemet tartalmazó magokat, amelyeknek később az unióját kívánjuk képezni (ahol ez egyáltalán lehetséges). Ehhez első lépésként $\bar{D}$ mátrix minden egyes oszlopában kiválasztunk egy-egy maximális elemet, ($\max d_{ij} = \text{konst}$, ahol $i = 1, \dots, I$). Így minden egyes adattípus legalább egy magban előfordul, de többen is előfordulhat. a $\bar{D}$ szimmetriája miatt akár soronként is haladhatnánk, de itt nem ezt a konvenciót követjük. Az oszlop maximális elemének kiválasztásakor esetleg több, azonos maximális értéket is találhatunk, ezért képezzük a sor elemeinek összegét, és az osz-

lop maximális eleme az, amelyikhez tartozó sorösszeg a legnagyobb. Ennek a választásnak az az oka, hogy az oszlop indexe által jelölt adattípussal való szegmensképzésbe a szóba jövő néhány adattípus közül először azt kívánjuk bevonni, amelynek a többi adattípushoz való kapcsolata, hasonlósága kisebb. Ha az oszlop több, azonos maximális értékű eleme esetében még a hozzájuk tartozó sorösszegek is egyenlőek lennének, akkor a többi adattípushoz való hasonlósági együtthatói megoszlását figyelembe véve választunk ki egyet. Az 1. fázis végének állapotát mutatja be a 3. táblázat.

2. Fázis:

Célja kijelölni azt a kezdő klasztert (a szegmens két elemből álló „magját”), amely a 3. fázisban sorra kerülő folyamat jó kiindulópontja lesz. Ezért meg kell találni a $\overline{\overline{D}}$ mátrixban azt a két adattípust, amelyek egymáshoz a mátrixon belül a legnagyobb mértékben hasonlóak. Ez úgy érhető el, hogy az 1. fázisban az oszloponként kiválasztott maximális elemek közül az abszolút maximális elemet választjuk ki. Mivel a $\overline{\overline{D}}$ mátrix szimmetriája miatt (a mi példánkban) két ilyen azonos értéket találunk, ezért közülük ugyanúgy a megfelelő sorösszegek alapján választjuk ki a keresett mátrixelemet, mint az 1. fázis során. E mátrixelem sor- és oszlopindexe közvetlenül megmutatja a szegmens magját képező két adattípust. Ezek sorát és oszlopát töröljük a mátrixból. A 2. fázis végének állapotát a 4. táblázat mutatja.

3. Fázis:

Célja, hogy a 2. fázisban kijelölt klaszterekből kiindulva rendre egyé olvasszuk azokat a kezdő klasztereket, amelyekben van közös rész. Ennek végrehajtása érdekében, a szegmens magjának kijelölése után meg kell vizsgálni, hogy vannak-e még $\overline{\overline{D}}$ -ben további adattípusok, amelyeket e mag köré csoportosíthatunk. Miután megvizsgáltuk, hogy az első adattípus mely másikkal van a legszorosabb kapcsolatban (hasonlóság), azt vesszük szemügyre, hogy e másik mely továbbihoz kapcsolódik legszorosabban. Ehhez a második adattípusnak megfelelő sorban keressük az első adattípuson kívüli, az előzőek során az oszlopban már megjelölt maximális elem(ek)et. Ha ilyen nincs, akkor a 3. fázist, azaz a szegmens kialakítását befejezzük, és a 4. fázisra ugrunk. Ha viszont ilyen(ek)et találunk, akkor az ezek oszlopindexe által jelölt adattípusokat is ebbe a szegmensbe tartozónak tekintjük, sorukat és oszlopukat a $\overline{\overline{D}}$ mátrixból töröljük. Az így kiválasztott adattípusra vonatkozóan ugyancsak rendre meg kell vizsgálni, — a másik adattí-

pus kapcsolódásaira nézve előbb leírt eljárás megismétlésével — hogy mely megmaradó továbbiakhoz kötődik a legszorosabban, és ha a megfelelő sorban találunk még megjelölt (maximális) oszlopelemeket, akkor az ezeknek megfelelő adattípusokat is felvesszük a szegmensbe. Felvételüket az indokolja, hogy az így kijelölt adattípusok mutatják a legtöbb hasonlóságot a szegmensbe legutóbb felvett adattípussal, míg hasonlóságuk a szegmensen kívüli adattípusokhoz viszonylag kismértékű. Az eljárást ciklikusan folytatjuk mindaddig, amíg olyan adattípushoz (sorhoz) jutunk, amelyben (1. fázisban) oszloponkénti maximális elem kijelölése alkalmával megjelölt újabb elemeket már nem találunk. A 3. fázis második szegmenstípusa képzése végének állapotát az 5. táblázat mutatja be.

4. Fázis:

Miután így kijelöltük a $\overline{\overline{D}}$ mátrix (illetve az ehhez tartozó $\overline{\overline{C}}$ mátrix) által reprezentált egyedtípus egyik szegmenstípusának tulajdonságtípusait, az ezeknek megfelelő adattípusok sorait és oszlopait a $\overline{\overline{D}}$ mátrixból elhagyva, annak redukált mátrixát képezzük. Amennyiben e redukált mátrix mérete legalább (3×3) , akkor újabb szegmenstípus képzéséhez a fent leírt eljárást kell ismét végrehajtani a 2. fázistól kezdve. Ha viszont a redukált mátrix mérete kisebb, mint (3×3) , akkor arra az egyedtípusra, amelyre az induló $\overline{\overline{D}}$ mátrix is vonatkozik, a szegmensképző algoritmust már nem működtetjük tovább, hanem áttérünk a következő egyedtípushoz (ha van még ilyen) tartozó $\overline{\overline{D}}$ mátrix feldolgozására, az 1. fázistól kezdve.

3.1.3. Alkalmas taxonomikus mérték

Tekintsük most a klaszterálandó $\overline{\overline{C}}$ mátrixba rendezett kiinduló adatainkat. Ha $C_{mn} \in \overline{\overline{C}}$ csakis F_n -nek A_m -re vonatkozó relatív elérési gyakoriságát fejezi ki, akkor esetünkben valamennyi ismérv értékeit egyaránt intervallumskálán mérhetjük.

A kiválasztás szempontjai:

- Milyen skálán mérhetők az adattípust jellemző ismérvek értékei?
- Szimmetrikus mértékre van-e szükség vagy sem?
- A klaszterálási algoritmus végrehajtása szempontjából eléggé hatékony-e a mérték?

Ezt figyelembe véve, esetleg alkalmazható lenne valamely metrikus (pl. az euklideszi) távolságmérték. Az előbbi feltétel azonban nem mindig telje-

sül. Tekintetbe kell még venni azt az erősebb megkötést is, hogy két adat-típus (A_i és A_j) lényeges jellemzői — azaz, hogy együtt ($P11_{ij}$), illetve egymástól külön-külön ($P01_{ij}$ és $P10_{ij}$) hány funkció igényli, illetve együtt hány funkció nem ($P00_{ij}$) igényli — szempontjából mikor jelenik meg a két adattípus egy funkcióban együtt $P11_{ij}$, és sokkal kevésbé érdekes, hogy mikor nem. A fentiek figyelembevételével csak olyan taxonomikus mértéket alkalmazhatunk, amely $P11_{ij}$ és $P00_{ij}$ súlyozása tekintetében nem szimmetrikus, és nem csupán intervallum, hanem pl. ordinális változó esetére is alkalmas mérték. A célszerű hasonlósági együtthatónak olyannak kell lennie, amely kifejezi, hogy egy véletlenszerűen kiválasztott funkció két adattípust egyaránt igényel $P11_{ij}$ míg az „együtt nem igényli” esetet $P00_{ij}$, figyelmen kívül hagyjuk. Ilyen normalizált aszimmetrikus hasonlósági együttható pl. az ún. Jaccard-együttható:

$$d_{ij} = \frac{P11_{ij}}{P11_{ij} + P01_{ij} + P10_{ij}}.$$

Előbbi feltételek mellett ez azt jelenti, hogy d_{ij} a $P11_{ij}$ monoton növekedő függvénye és fennáll, hogy itt $d_{ij} = d_{ji}$, továbbá $0 \leq d_{ij} \leq 1$. Az egyszerű lánc-módszer végrehajtása esetében a Jaccard- és a

$$d_{ij} = \frac{2 \cdot P11_{ij}}{2 \cdot P11_{ij} + P01_{ij} + P10_{ij}}$$

Dice-féle együttható alkalmazása teljesen azonos hatású.

Az egyszerű lánc-módszer és a minimális feszítőfa előállítás között szoros összefüggés mutatható ki. Tekintettel arra, hogy az előbbieken leírt algoritmus ugyancsak minimális feszítőfát állít elő, esetünkben a Jaccard- és a Dice-együttható egyaránt alkalmazható lenne, így e két hasonlósági együttható közül azt kell használni, amelynek választása az egész feldolgozási folyamat hatékonyságát jobban növeli.

3.2. A hatékonyság és a gyakorlati megvalósítás szempontjai

A $\overline{\overline{D}}$ mátrix előállításához a $\overline{\overline{C}}$ mátrixból kiindulva d_{ij} értékeket határozzunk meg, amelyek majd a $\overline{\overline{D}}$ mátrixot alkotják. Ez a $\overline{\overline{D}}$ mátrix a főátlóra

szimmetrikus ($d_{ij} = d_{ji}$), a főátló elemeinek ($i = j$) értéket ($d_{ij} = 1$ lenne) nem adunk, míg a többi mátrixelem (ahol $i \neq j$) értéke $0 \leq d_{ij} \leq 1$. Az algoritmus legidőigényesebb része éppen a $\overline{\overline{D}}$ mátrix előállítása, azaz a $d_{ij} \in \overline{\overline{D}}$ hasonlósági együtthatók kiszámítása a $C_{mn} \in \overline{\overline{D}}$ értékekből, mert ehhez például M számú sor között kell $M(M-1)/2$ számú — ha $\overline{\overline{D}}$ szimmetrikus, akkor csak $M(M-1)/4$ — páronkénti so-relemről so-relemre haladó összehasonlítást, majd az előállított $P11_{ij}$, $P01_{ij}$, $P10_{ij}$ hasonlósági paraméterekkel aritmetikai műveleteket végzünk. A $\overline{\overline{D}}$ mátrix előállítására azonban kidolgoztak már gyorsabb módszereket is. Ezek helyett egy még hatékonyabb algoritmust alkalmazunk, amellyel $\overline{\overline{D}}$ egy sorát egyszerre előállíthatjuk. A gyorsításra az ad módot, hogy ahol $d_{ij} = 0$, ott fennáll $P11_{ij} = 0$ is, így elegendő csupán a $P11_{ij}$ értékeket megállapítani. Habár valamennyi (M) adattípus esetén a hasonlósági együtthatók $d_{ij} = 0$ vagy $d_{ij} \neq 0$ értékének megállapításához ugyan változatlanul $M(M-1)/2$ mennyiségű összehasonlítást kell elvégezni, de az mégis gyorsabb, mint a hasonlósági együtthatók értékének teljes meghatározása. Ez utóbbi kiszámítására csak akkor kerül sor, ha $P11_{ij} = 0$. Az időmegtakarítás annál nagyobb, minél többször fordul elő $d_{ij} = 0$ érték.

Az általunk jelenleg implementált algoritmus öt lépésből áll. Az algoritmus használható mind hasonlósági együttható, mind távolságmérték alkalmazása esetében.

3.2.1. Hasonlósági együttható alkalmazása

1. lépés:

A már megismert $\overline{\overline{C}}(M, N)$ mátrixból kiindulva képezzük először az M elemű $\overline{P}$ vektort, amelynek egy elemét (p_m) úgy képezzük, hogy összeadjuk a $\overline{\overline{C}}$ mátrix egy sorában álló elemeket:

$$p_m = \sum_{n=1}^N c_{mn}, \text{ ahol } m = 1, 2, \dots, M.$$

A $\overline{P}$ vektor előállítása a $\overline{\overline{C}}$ mátrix ellenőrzését szolgálja. A $p_m = 0$ eset ugyanis azt jelenti, hogy a $\overline{\overline{C}}$ mátrixba bevont A_m adattípus ott tu-

lajdonképpen felesleges, hiszen az E_j egyedtypusnak egy olyan tulajdonságtípusáról van szó, amelyet az adatmodellre irányuló egyetlen adatkezelési funkció sem igényel. A $\bar{P}$ vektor bármelyik két p_i és p_j elemét ahol $i \neq j$ és $i = 1, \dots, M$ és $j = 1, \dots, M$ összeadva, a

$$p_i + p_j = 2 \cdot P11_{ij} + P01_{ij} + P10_{ij}$$

értéket kapjuk.

A $p_i + p_j$ előállításának két fő célja van: egyrészt segítségével a Dice-együttható viszonylag gyorsan előállítható, másrészt bizonyos mértékig ellenőrizhető a $\bar{C}$ mátrix kitöltése. Ha ugyanis $p_i + p_j = 0$, ez csak úgy állhat elő, hogy

$$P11_{ij} = P01_{ij} = P10_{ij} = 0.$$

Ez az eset azt jelenti, hogy a szóban forgó $\bar{C}$ mátrix helyesbítését kell elvégeznünk, hiszen a tekintett két adattípus (A_i és A_j) egyikét sem igényli a $\bar{C}$ mátrixban szereplő egyetlen funkció sem. A $\bar{C}$ mátrix létrehozása során tehát hiba történt, amelyet korrigálni kell. Vagy az volt a hiba, hogy a $\bar{C}$ egyszerűen felesleges adattípust (A_i -t vagy A_j -t) tartalmaz, vagy az, hogy a funkciók tényleges adatigényét pontatlanul tükrözi a $\bar{C}$ mátrix, elírás történthetett. Az is előfordulhat, hogy a két adattípus valamelyikét nem ehhez a $\bar{C}$ mátrixhoz, hanem egy másik egyedtypushoz tartozó $\bar{C}$ mátrixhoz kellett volna rendelni. A $p_i + p_j$ közvetlen előállíthatósága amellet szól, hogy a Jaccard- helyett a Dice-együtthatót alkalmazzuk.

2. lépés:

Ezután a $\bar{C}$ mátrixból előállíthatjuk annak l -edik (kezdetben $l = 1$) redukált mátrixát $\bar{J}_l$ -et, $J_l = (\bar{C}_{k_1}, \dots, \bar{C}_{k_r})$, ahol $r \leq N$ és $1 \leq k_1 < k_2 < \dots < k_r \leq N$ úgy, hogy ha $\bar{C}_j$ oszlopvektor l -edik eleme 1, akkor (fennáll $k_i = j$) bekerül $\bar{J}_l$ -be annak egy oszlopaként, egyébként pedig nem.

Tegyük fel hogy $\bar{J}_l(M, K)$ méretű mátrix (7. táblázat); ebbe a redukált mátrixba bevontuk mindazon változók (itt funkciók) körét, amelyek az A_i és a többi adattípus közös (együtt előforduló) jellemzőiként potenciálisan felléphetnek a $\bar{C}$ mátrixon belül.

3. lépés:

Képezzük először a $\overline{P11}_l$ vektort, amelynek elemei a $P11_{ml}$ skalárok, ahol

$$P11_{ml} = \sum_{k=1}^K j_{mk}$$

sorra megadják az A_l és az A_m (ahol $m \neq l$) közötti hasonlósági együtt-hatók számlálójában lévő értékeket, azaz a $P11_{ml}$ értékeket (kivéve $l = m$ esetében, ahol $d_{ml} = 0$) $j_{mk} \in \overline{J}_l$, ahol $k = 1, 2, \dots, K$ és $l = \text{konst.}$ (8. táblázat).

- Ahol $P11_{ml} = 0$ (de $P0_{ml} \neq 0$ és $P10_{ml} \neq 0$), ott $d_{ml} = 0$ és mivel A_l -et és A_m -et nem vonjuk össze egy szegmenstípusba, így vele e célból további műveleteket már nem fogunk végezni, tehát tárolására sem lesz szükségünk (ha a $\overline{D}$ mátrixot nem tömbként, hanem listaszervezetben tároljuk).
- Ahol azonban $P11_{ml} \neq 0$, ott fennáll $0 < d_{ij} \leq l$, és a már előbb előállított p_m felhasználásával kiszámítjuk a d_{ml} értékeket, azaz a $\overline{D}$ mátrix egy sorát (vagy oszlopát).
- Amennyiben $d_{ij} = l$, akkor az A_i és A_j megfelelő adattípusokat feltétel-nül egy szegmenstípusba vonjuk össze. Ez ugyanis azt jelenti, hogy két adattípust a C mátrixon belül kizárólag együtt igénylik, valamennyi funkció esetén. Még figyelembe kell venni a következőket: Természe-ten nincs kizárva, hogy itt nem két külön, hanem egy és ugyanazon — de valamilyen hiba miatt kétszer és eltérő indexxel azonosított — adattípusról van szó. Nyilván, egy adattípus fizikailag többszöri (redun-dáns) tárolását az információrendszer majdani adatbázisában általában célszerű elkerülni. Persze különbséget kell tenni az egy szegmenstípuson belüli, és a szegmenstípusok között fennálló redundancia esete között.

Egy szegmenstípuson belül mindenképpen el akarunk kerülni bármiféle redundanciát. Ezt egyrészt a szegmensképzés kiindulásához használt nor-malizált (harmadik normál alakú) adatmodell elkészítése, másrészt egy $\overline{D}$ mátrix előállítása közben elvégzendő logikai ellenőrzés biztosítja. Ez az el-lenőrzés azt jelenti, hogy ha két (A_i és A_j) adattípus, ahol $i \neq j$, valamennyi taxonomikus jellemzője egybeesik (azaz $d_{ij} = l$), akkor a két adattípust meg

kell vizsgálni és tényleges tartalmi egybeesés esetén az egyiket törölni kell a $\overline{\overline{C}}$ mátrixból. Kiköthetők olyan feltételek is, hogy már egymáshoz igen közelinek jellemzett ($0.95 < d_{ij} < 1$) adattípusokat is külön megvizsgálunk. Ha $0 < d_{ij} < 1$, akkor összevonásra sor kerülhet.

4. lépés:

Soronként kiválasztjuk a $\max(d_{mi})$ eleme(ke)t, egyben mindjárt képezzük a sorösszegeket is. Előre el kell határoznunk, hogy csupán ezeket tároljuk-e vagy valamennyi most kiszámított hasonlósági együttható értékét megőrizzük. Ezzel kapcsolatban az alábbi szempontokat kell figyelembe venni:

A 3.1.2. algoritmus gépi megvalósítása tekintetében alapvetően három alternatíva közül választhatunk:

- A teljes hasonlósági $\overline{\overline{D}}$ mátrixot tároljuk. Ekkor a tárigény viszonylag nagy és az előállítás a $\overline{\overline{C}}$ mátrixból $(2M^2 - 9M)/2$ számú összehasonlítást igényel. Így ez az alternatíva akkor kerül előtérbe, ha a $\overline{\overline{C}}$ mátrixok többségére fennáll, hogy a funkciók száma (N) meghaladja a klaszterálandó adattípusok számát (M), azaz $M < N$.
- A teljes hasonlósági $\overline{\overline{D}}$ mátrix tárolása helyett csupán egyes oszlopainak (vagy sorainak) maximális elemét tároljuk. Erre akkor nyílik lehetőség, ha a $\overline{\overline{C}}$ mátrix változóit arány- vagy intervallumskálán mérhetjük. Ha a $\overline{\overline{C}}$ mátrixok döntő többségére fennáll az, hogy $M > N$, akkor ennek az alternatívának a tárigénye az előbbihez képest lényegesen kisebb lesz, ugyanakkor azonban $(3M^2 - 11M)/2$ összehasonlítást, majd $M(M - 1)$ hasonlósági együttható kiszámítását igényli.
- Háttértárolón helyezük el a szekvenciális sorrendbe rendezett hasonlósági $\overline{\overline{D}}$ mátrixot. Ezt célszerűen akkor tehetjük meg, ha a klaszteráló algoritmus hatékonyságát ez nem korlátozza. A feszítőfák előállítására épülő algoritmusok hatékonyságát a fenti megoldás általában nem korlátozza, így ez számos esetben alkalmazható.

Hogy e három alternatíva közül melyiket alkalmazzuk most, azt az alábbi megfontolások befolyásolják: Tekintettel arra, hogy az adatbázis újraszerkesztésére annak életciklusa alatt sokszor szükség lesz, olyan klaszteráló módszert célszerű alkalmazni, amely mindig támaszkodhat az előző szerkesztést meghatározó taxonómikus mérték (vagy annak tényezői) eltárolt értékeire. Ez arra utal, hogy jelen esetben a tárolt hasonlósági bimát-

rixon alapuló implementációt helyesebb alkalmazni. Ezt az állítást még két további érv támasztja alá:

- Az adattípusok szegmenstípusokká csoportosításának alapját képező változók egy része intervallumskálán, más része ordinális skálán mért változó lehet. A változókon a funkciókat értjük, szubjektív súlyozásuk eredményezhet ordinális skálán mért változókat. Ez esetben pedig a tárolt hasonlósági mátrixon alapuló implementáció a korrekt megoldás.
- Az esetek többségében a funkciók száma kisebb, mint az adattípusoké. Ekkor a tárolt hasonlósági mátrixszal megvalósított agglomeratív klaszterálás hatékonyabb, mint az egyéb implementációs lehetőség.

Tekintettel arra, hogy mindjárt a szegmensképzés utáni tervezési szakaszban sok szempont miatt is szükség van több d_{ij} mátrixelem ($d_{ij} \in \bar{\bar{D}}$) gyors elérésére, helyes itt a hasonlósági együtthatók alkotta $\bar{\bar{D}}$ mátrix teljes tárolása. Ezt mind a $\bar{C}$ mind a szóban forgó $\bar{\bar{D}}$ mátrixok száma, valamint átlagos terjedelme, továbbá az általában rendelkezésre álló közvetlen elérési tárolók kapacitása is megengedi.

A mátrixok számára és terjedelmére vonatkozó becslésünk a következő:

Annyi $\bar{C}$, illetve $\bar{\bar{D}}$ mátrix van, ahány egyedtípus. Ezek száma egy átlagos információs rendszerben 20 és 100 között van. Egyedtypusonként, tapasztalat szerint, 5-30 adattípus fordul elő. Az adatképzési funkciók száma egyedtypusonként elég eltérő lehet, de a tervezéskor figyelembe vett kisebb hányaduk miatt ez a szám hozzávetőleg 15-20. Ebből adódik, hogy egy $\bar{C}$ mátrix mérete legfeljebb 20×30 , egy $\bar{\bar{D}}$ mátrixé ugyancsak legfeljebb 30×30 , mindkét fajta mátrixból legfeljebb 100-100 van egy átlagos információs rendszerben.

5. lépés:

Mivel $\bar{\bar{D}}$ szimmetrikus, elegendő az alsó háromszögmátrix előállítása, tehát l értékét eggyel megnövelve megvizsgáljuk, hogy $l < M$ fennáll-e. Ha igen, visszatérünk a második lépés elejére és új redukált mátrixot képezzünk. Amennyiben $l < M$ már nem áll fenn, akkor a $\bar{\bar{D}}$ mátrix előállítása befejeződik.

3.3. Távolságmérték alkalmazása

Az implementáció megoldását jelentősen befolyásolja az elvileg megfelelő taxonómikus mérték kiválasztása. Mivel a változók most többértékűek, ezért át kell térni a taxonómikus távolságmértékre, továbbá a hasonlósági mátrixról távolsági mátrixra.

A taxonómikus mértékekkel szembeni elvárásunk, hogy a mérték fejezze ki, hogy két tulajdonságtípusnak — egy szegmenstípusba összevonását mérlegelve, — fontosabb közös jellemzője, ha egy adott adatkezelési funkció esetén együtt előfordulnak, mint az, ha nem fordul elő egyikük sem. A hasonlósági együtthatók jórésze ilyen tulajdonságú, a távolságmértékek közül azonban ez igen kevésnek van meg. Tekintettel arra, hogy a hasonlósági együtthatóként legalkalmasabbnak a Dice (vagy Czekanowsky- féle) együttható látszik, az ennek kiterjesztése inverzeként meghatározható Lance-Williams-féle nem-metrikus távolságmérték alkalmazása a következetes választás.

Anderberg rámutat, hogy a Lance-Williams-féle nem-metrikus távolságmérték

$$L_{ij} = \frac{\sum_{n=1}^N |c_{in} - c_{jn}|}{\sum_{n=1}^N (c_{in} + c_{jn})} = 1 - \frac{2 \cdot P11_{ij}}{2 \cdot P11_{ij} + P01_{ij} + P10_{ij}},$$

éppen a Dice-együttható valós, többértékű változók esetére való általánosítása. Mivel azonban ez nem tesz eleget a metrika valamennyi követelményének (ugyanis ha $c_{mn} < 0$, akkor $c_{mn} \in \overline{C}$ esetében $L_{ij} < 0$ is előfordulhat), csak $c_{mn} \geq 0$ esetén alkalmazható.

4. fejezet

Szegmenstípusok összevonása nagyobb egységekbe

4.1. Az „A” tervezési stratégia teendői

Az „A” stratégia az alábbi tevékenységek végrehajtását jelenti:

- aa) Az előzetesen már létrehozott szegmenstípusok alkalmas csoportosításával ún. szegmensosztályok létrehozása.
- ab) A szegmensszintű adatmodellek belső hierarchikus szerkezetét jellemző ún. szegmenselési fák meghatározása.
- ac) Minden egyes szegmensosztályból az ún. mértékadó szegmensszintű adatmodell előzetes kiválasztása.
- ad) A szegmenselési fák egy erdővé összevonása.
- ae) A mértékadó szegmensszintű adatmodellek esetleges korrekciója és végleges rögzítése.
- af) Az erdő csomópontjaihoz a konkrét ABKR által megengedett standard elhelyezési módok hozzárendelése.
- ag) A mértékadó szegmensszintű adatmodellek alapján az egyes area-k tartalmának rögzítése.

Az aa) tevékenység végrehajtásához a tervezői döntés előkészítését számítógéppel segítjük, míg az ab), ac), ad), tevékenységek végrehajtását számítógép nem segíti.

Az „A” stratégia esetén a fizikai tervezési folyamat akkor hatékony, ha a szegmensosztályok száma viszonylag csekély és egy-egy szegmensosztályon belül a szegmensszintű elérési fák szerkezete is hasonló, valamint ezek összevonásakor az összeférhetőségi szabály betartása érdekében csak kevés esetben kell az előzetesen kiválasztottól eltérő más, mértékadó szegmensszintű adatmodellt ismételten választani.

4.1.1. Szegmenstípusok osztályozása

Az adatbázis-tervezésnek most következő szakaszában az az egyik célunk, hogy logikailag egymás mellé azok a szegmenstípusok kerüljenek, amelyeket ugyanazon adatkezelési funkciók használnak. Ezért a szegmenstípusok olyan posztkoordinált osztályozását fogjuk elvégezni, amely teljes, és biztosítja hogy a szegmenstípusok egy osztálya (röviden: egy szegmensosztály) alkalmas legyen egyszerre (minél) több funkció adatigényének kielégítésére (tehát ebből a szempontból homogenitást mutasson). A szegmensosztályok képzése közvetlenül az area- felosztás megoldását célozza, közvetve pedig előkészíti az elhelyezési módok meghatározását, szegmenstípusok (esetleg egyedtípusok) összevonását, kapcsolattípusok szétbontását, esetleges összevonását. A szegmensosztályok képzésére az alábbi módszer javasolható:

A funkcionális felmérés és a szegmensképző eljárás végeredményéből (szegmenstípusok) kiindulva hozzuk létre a $\overline{\overline{B}}$ bináris mátrixot. Sem az egy szegmenstípussal leírt szegmensek számát, sem az egy-egy adatmodell előfordulásának gyakoriságát jellemző adatokat itt most nem kell figyelembe venni, mivel ezt a szegmensképzés során már megtettük. Az így létrehozott $\overline{\overline{B}}$ mátrixot blokk-mátrix előállítására $\overline{\overline{B}}^*$ törekedve rendezzük át úgy, hogy $b_{tr} \neq 0$ értékű elemei minél nagyobb mértékben a főátló körül sűrűsödjének. Így az egyes szegmenstípusok osztályba tartozása a szemlélet alapján is könnyen eldönthető lesz. $\overline{\overline{B}} \sim \overline{\overline{B}}^*$, mert $\overline{\overline{B}}$ átrendezését elemi átalakításokkal (sor-, illetve oszlopcserék) végezzük el. Eleve feltehető azonban, hogy automatikus osztályozásunk nem lehet taxonomikus és a kapott osztályok átfedésben lesznek. Előre látható, hogy gyakorlatilag mindig van több olyan funkció, amely adatigényének kielégítését biztosító szegmensszintű adatmodell csak több szegmensosztályból gyűjthető össze. Ez egyben azt is jelenti, hogy emiatt szegmenskapcsolat nem csupán egy-egy szegmensosztályon belül, hanem szegmensosztályok között is fennáll.

A fenti körülményeket figyelembe véve, az elméletileg lehetséges legkedvezőbb osztályképzés, — amelyet a $\overline{\overline{B}}^*$ mátrix tükröz — a gyakorlatban igen ritkán valósítható meg. A remélhető legkedvezőbb eset az ún. lépcsőház-mátrix, hiszen egy szegmenstípus egyértelműen a funkciók egy jól körülhatárolható részhalmazához rendelhető, mivel ezek belőle egyaránt merítenek, Egy funkció gyakran nem rendelhető egyértelműen a szegmens-

típusok egy osztályához, mert elintézéséhez általában egyszerre több szegmensosztályból is kell adattípusokat meríteni.

Ha még ez sem jöhet létre (mert számos olyan szegmenstípus adódik, amely a fenti feltételnek nem tesz eleget), akkor ezt az fejezi ki, hogy az átrendezett $\overline{\overline{B}}$ mátrixban a fődiagonálison kívül találunk olyan homogén blokkokat, amelyek a szegmensosztályok közötti kapcsolatokat fejezik ki.

A szegmensosztályok képzéséhez olyan módszerre van szükség, amely a modell sokdimenziós terét a képernyő kétdimenziós síkjára úgy vetíti ki, hogy az ilyenkor szükségképpen fellépő torzulások (a leképzés nem távol-ságtartó) ellenére is gyorsan el lehessen dönteni: mely szegmenstípusok tartoznak egy szegmensosztályba (klaszterba) és melyek nem.

A $\overline{\overline{B}}$ mátrixból kiindulva pl. az alábbi módokon járhatunk el, ha a szegmenstípusok osztályozását kívánjuk elérni:

- sávzsélesség-csökkentő eljárást alkalmazunk,
- létrehozuk a szegmenstípusok taxonomikus mátrixát, és azt Robinson-tulajdonságú mutató mátrixxá rendezzük át,
- a Bolla M. és Tusnády G. által javasolt megoldást követjük,
- az Anderberg által javasolt stratégiát követve ún. harmonizált klaszterálást végzünk.

Sávzsélesség-csökkentő eljárás

Sávzsélesség-csökkentő módszer alkalmazására akkor nyílik lehetőség, ha a $\overline{\overline{B}}(T, R)$ mátrix négyzetes, azaz $T = R$. A sávzsélesség-csökkentés NP-teljes probléma, a vonatkozó irodalom kimerítően tárgyalja. (A ritka mátrixok sávzsélesség-csökkentésének új hatékony módszerét Arany Ilona dolgozta ki.) A fent hivatkozott feltételek fennállása esetén a szegmenstípusokból nagyobb csoportok, azaz szegmensosztályok képzésére éppen ennek, az — MTA SZTAKI-ban rendelkezésre álló —, elméletileg alaposan kidolgozott és igen korszerű megoldásnak az alkalmazása javasolható.

Taxonomikus mátrix létrehozása és átrendezése

Négyzetes mátrix esetében olyan mátrixátrendezés hajtható végre, amely a sávzsélesség minimumának elérését célozza. Azonban a $\overline{\overline{B}}(T, R)$ mátrixra általában fennáll, hogy $T \neq R$. A $\overline{\overline{B}}$ adatmátrix lényegében $\{S_z\}^T$ szegmenstípust jellemez az $\{F\}^R$ változó függvényben. Megtehetjük, hogy képezzük a fenti szegmenstípusok közötti hasonlósági távolsági ultrametrikus

függvényértékeket, és azokat egy $\overline{H}$ szimmetrikus mátrixba rendezzük. Ha hasonlósági függvényt alkalmazunk, akkor $h_{i,j} \in \overline{H}(T, T), 0 \leq h_{i,j} \leq 1$, ahol $i = 1, 2, \dots, T$ és $j = 1, 2, \dots, T$, ha pedig távolságfüggvényt, akkor $k = 1, 2, \dots, T$ esetén $h_{i,j} \leq \max(h_{ik}, h_{jk})$, minden i, j, k értékre.

A $\overline{H}$ mátrixból kiindulva valamely alkalmas hierarchikus klaszteráló algoritmussal már klasztereket készíthetünk, és ezáltal — úgy tűnik — meghatározhatjuk, hogy mely szegmenstípusokat kell majd egymáshoz közel elhelyezni. A klaszterálás végeredményeként előállítható egy dendrogram, azaz egy ultrametrikus fastruktúra szintjeibe rendezett gráf, amely alkalmasnak látszik arra, hogy pontosan tükrözze a szegmenstípusok alkotta tényleges klaszterstruktúrát (pl. a szinteket, a szintek közötti távolságokat stb.) Sajnos azonban ez nincs így. A hierarchikus klaszterálás valamennyi módszere mellett megmaradt az alapvető hiányosság, hogy egyik sem adja meg kétségtelenül és egyértelműen azt, hogy melyek is a valóságban ténylegesen létező klaszterek(s egyáltalán vannak-e), hanem agglomeratív módszerek esetén kötelezően egybeolvaszt (vagy divizív esetben kettévág) még akkor is, ha az az adathalmaz belső struktúrája egészét tekintve nem is indokolt. Tehát a dendrogram — vagy más formában kapott eredmény — nem annyira a ténylegesen meglévő klaszterstruktúrának, hanem egyértelműen a klaszterálás folyamatának, algoritmusának grafikus reprezentációja. Jellemző az is, hogy eltérő algoritmusokkal, ugyanazon adathalmazra alkalmazva azokat, eltérő eredményeket, struktúrát kapunk.

Történtek kísérletek arra, hogy ezt a nehézséget áthidalják. Általában a dendrogram olyan kiegészítésére törekedtek, hogy az tükrözze a taxonómikus függvény megfelelő értékeit (pl. a dendográf, a taxonómikus térkép és a threaded mátrix). Volt olyan törekvés is, hogy a H mátrixot a könnyű áttekinthetőség érdekében úgy kell átrendezni, hogy a mátrix eleget tegyen az ún. Robinson-tulajdonságnak. Ez a probléma NP-teljes és bonyolultabb, mint a valamelyest hasonló, de a sáv szélesség-csökkentésére törekvő ugyancsak NP-tejes feladat. A megoldásra irányuló kísérletek abból indulnak ki, hogy bármely hierarchikus klaszterálás végeredménye egy bináris fagráf, melynek terminális elemeit célszerűen átrendezve (rotáció) éppen azt a fát tudjuk megváltoztatni, amelynek megfelelően permutált taxonómikus-mátrix majd eleget tesz a Robinson-tulajdonságnak. Elsőként Hartigan javasolta, hogy a hierarchikus klaszterálást kezeljék úgy, mint egy gyökérrel rendelkező fagráf — melyen ultrametrika definált — kombinatorikus optimalizálási módszer-

rekkel való illesztését. Alkalmas algoritmust és a legkisebb négyzetek módszerével történő illesztést is alkalmazott az input adatmátrix egyedei közötti távolságok és az ultrametrikus fagrafon számított távolságok között. A gondolatmenetet továbbfejlesztve. Sattath és Tversky „four point metric” modelljében a fagraf helyett már egy gyökérpont és hierarchikus rendezettség nélküli gráf szerepel.

Általában azt mondjuk, hogy a gráf pontjainak célszerű átrendezéséhez vagy magából a taxonomikus mátrixból nyerhető információra, vagy egy megadott külső rendezési kritériumra van szükség.

Az előbbi esetet kezelő algoritmus célja, hogy a hierarchikus klaszterálással nyert klaszterek (bináris részfák) belső szerkezetét úgy rendezze át, hogy az egyik klaszternek (részfa) a szomszédos másik klaszterhez legközelebb eső egyedei egymáshoz a lehető legközelebb kerüljenek. Arra az esetre, ahol az egyedek között nincs összefüggés, ez az algoritmus optimális megoldást ad. Ezt az esetet DeGerman algoritmusá példázza, amelyben minden átrendezés után a külső rendezettségi kritérium és az átrendezett szögponatok közötti korrelációt a Kendall-féle sorrendmérő τ -val mérik. Erre az algoritmusra mutató megoldást építhető, amely azonban jelentős időráfordítással valószínűsíthető meg, és még mindig csak közelítő megoldást szolgáltat.

Dichotom változók és objektumok paralell klaszterálása

Bolla M. és Tusnády G. az alábbi iteratív megoldást javasolja:

1. lépés:

Először a $\overline{\overline{B}}(T, R)$ adatmátrixból kiindulva Mac Queen-módszerrel létre kell hozni L diszjunkt klasztert (itt: szegmensosztályt) $\{C^{(1)}, \dots, C^{(L)}\}$.

2. lépés:

Második lépésként meg kell határozni a

$$\sum_{S \in C^{(l)}} K(t, l) \longrightarrow \min, \text{ ahol } l = 1, \dots, L$$

szélsőértéket a $C^{(l)}$ -be tartozó szegmenstípusok és azok elérését igénylő adatbázisfunkciók alkotta minormátrix szinguláris értékeinek dekompozíciójával:

$$K(t, l) = \sum \|y_t - x_r^{(1)}\|^2$$

minden olyan esetre, ahol F_r értéke = 1 SZ_t -ben, és $\underline{x}_1, \dots, \underline{x}_R$ az adatbázisfunkciókhoz, $\underline{y}_1, \dots, \underline{y}_T$ pedig szegmenstípushoz tartozó pontok a k -dimenziós euklideszi térben ($k \leq \min\{T, R\}$).

Legyen a $Q^{(l)}$ az l -edik klaszterhez, C^l -hez tartozó célfüggvény:

$$Q^{(l)} = c \cdot 2^{h(l)} + \sum_{SZ_t \in C^{(l)}} K(t, l),$$

ahol C egy előre megválasztott konstans, $h(l)$ azoknak az adatbázisfunkcióknak a száma, amelyek a $C^{(l)}$ -hez tartozó szegmenstípusok valamelyikét elérik és $K(t, l)$ a t -edik szegmenstípusnak (SZ_t) az l -edik klaszterbe ($C^{(l)}$) való átmozgatásakor fellépő „költségfüggvény”.

Minden egyes klaszterre végezzünk diszkrét minimalizálást:

$$\min_{h(l)} Q^{(l)} \quad (\text{ahol } l = 1, \dots, L).$$

3. lépés:

Az SZ_t szegmenstípust át kell helyezni $C^{(l)}$ -ből $C^{(l^*)}$ -ba, ha l^* -ra diszkrét minimalizálással

$$\min_l \sum_r b_{tr} \|y_t - x_r^{(l)}\|^2.$$

4. lépés:

Ezután $Q = \sum_{l=1}^L Q^{(l)} \rightarrow \min$ érdekében a második lépéstől egy új iterációt kell végrehajtani.

Mivel Q értéke minden egyes iteráció után csökken és a szegmenstípusok száma véges, el fogjuk érni a Q célfüggvény egy lokális optimumát.

Harmonizált klaszterálás

Csoportosítandó egyedeknek a szegmenstípusokat, változóknak pedig a funkciókat tekintve, a cél voltaképpen a változók és az egyedek „harmonizált” osztályozásának végrehajtása. Ennek egy lehetséges megoldása az, hogy az egyedeket, majd a változókat felváltva, ciklikusan klaszteráljuk úgy, hogy végeredményben egy kölcsönösen, „harmonizált” eredményt kapunk. A feladat optimális táblázat átrendezés. Ilyen jellegű feladatok megoldására több módszer is ismeretes, de valamennyi heurisztikus.

Sok (több száz) tétel klaszterálására olyan iteratív heurisztikus algoritmust alkalmaznak, amely a kiinduló $\overline{\overline{B}}$ mátrix oszlopait, majd sorait permutálja. Ennek ismertetéséhez szükségünk van a feszítőút fogalmára. Feszítőútnak nevezzük azt az összefüggő részgráfot, amely eleget tesz az alábbi feltételeknek:

- minden szögpontja a gráfon belül van,
- ezek között van két olyan szögpont, amelyekhez csak egyetlen él csatlakozik,
- a részgráf összes többi szögpontja esetében egy-egy szögponthoz mindig csakis két-két él csatlakozik.

A feszítőút a feszítőfának egy speciális esete. Minimális feszítőútnak nevezzük azt a feszítőutat, amelynek teljes hossza a gráfban lehetséges valamennyi feszítőút közül a legrövidebb. Egy minimális feszítőút megtalálása az „utazó ügynök probléma” megoldásával egyenértékű.

A minimális feszítőút meghatározására gyakran alkalmazott algoritmusok, ha sok (legalább néhány száz) tétel klaszterálásáról van szó, túl lassúak. A gyorsabb eljárás érdekében megelégszenek azzal is, hogy ne a minimális, hanem egy ahhoz eléggé közelálló hosszúságú, azaz egy ún. rövid feszítőutat kapjanak meg.

Minden egyes permutáció alkalmával arra kell törekedni, hogy a $\overline{\overline{B}}$ mátrixra vonatkozó alábbi célfüggvény értékét maximalizáljuk.

$$f(\overline{\overline{B}}) = \frac{1}{2} \sum_{t=1}^T \sum_{r=1}^R b_{t,r} (b_{t,r+1} + b_{t,r-1} + b_{t-1,r} + b_{t+1,r}),$$

ahol $b_{tr} \neq 0$, továbbá $b_{0,r} = b_{t+1,r} = b_{t,r+1} = 0$

Az $f(\overline{\overline{B}})$ bármely méretű és formájú mátrixra számítható.

$$\begin{aligned} f(\overline{\overline{B}}) &= \frac{1}{2} \sum_{t=1}^T \sum_{r=1}^R b_{t,r} (b_{t,r+1} + b_{t,r-1}) + b_{t,r} (b_{t-1,r} + b_{t+1,r}) = \\ &= \frac{1}{2} \sum_{t=1}^T \sum_{r=1}^R f_t(\overline{\overline{B}}) + f_r(\overline{\overline{B}}). \end{aligned}$$

Minthogy $f(\overline{\overline{B}})$ kiszámításakor, ha a $\overline{\overline{B}}$ sorain haladunk végig, csak a belső összeg második tagja $f_r(\overline{\overline{B}})$ változhat, továbbá, ha $\overline{\overline{B}}$ oszlopait vizsgáljuk, akkor csak a belső összeg első tagja $f_t(\overline{\overline{B}})$ változhat, az algoritmus is két menetre, az oszlopok és sorok permutálására tagolható.

Az algoritmus két menetének lényege a következő:

I. menet: Megkeres az oszlopvektorok között egy rövid feszítőutat. Átrendezi az oszlopokat úgy, hogy azok sorrendjét a rövid feszítőút jelöli meg.

II. menet: Az oszlopcserék végrehajtása után megkeres a sorvektorok között egy rövid feszítőutat. Átrendezi a sorokat úgy, hogy azok sorrendjét a rövid feszítőút jelöli ki.

Az algoritmus első menetének lépései:

1. lépés: Válasszunk ki egy tetszőleges oszlopot a $\overline{\overline{B}}(T, R)$ mátrixból. $I = 1$.

2. lépés: Közvetlenül a kiválasztott oszlop mellé jobbra és balra is (ha lehetséges) egyenként rendre odapróbáljuk a megmaradt $R - I$ számú oszlopot és minden egyes ilyen belepróbálás alkalmával megnézzük (kiszámítjuk), hogy hogyan nőne vagy csökkenne $f(\overline{\overline{B}})$ értéke. Amelyik oszlopcserénél az $f(\overline{\overline{B}})$ értéke maximális lenne, azt az oszlopcserét ténylegesen végrehajtjuk. Az algoritmus módosítható úgy, hogy $f(\overline{\overline{B}})$ helyett bármely távolságmérték használható. Ekkor az oszlopcserét a minimális távolságmérték esetén kell végrehajtani.

3. lépés: $I = I + 1$, és ismételjük meg a 2. lépést, amíg csak $I + R$ be nem következik.

Az algoritmus időigénye $(T^2 \cdot R + T \cdot R^2)$ -nel arányos, azaz az időigény a $\overline{\overline{B}}$ mátrix méretétől és az alkalmazott távolságmértéktől függ, és csak kevésbé befolyásolja a kezdő oszlop, illetve sor megválasztása.

4.1.2. Szegmenselési fák meghatározása

Az egy szegmensosztályhoz tartozó valahány funkció közül néhány csakis ehhez a szegmensosztályhoz rendelt. Minden egyes funkció ellátásához végre kell hajtani egy navigációt. Az ehhez igényelt szegmenstípusok precedencia-sorrendjét még a funkcionális elemzés során fel kell tárni. Ebből funkcion-

ként egy szegmenselési fa konstruálható. A fa gyökere az a szegmenstípus, amely tartalmazza a Halassy által vezérlőinformációnak nevezett tulajdonságtípust. A szegmensosztályhoz egyértelműen rendelt funkciók navigációi által meghatározott szegmenselési fák gyökércsomópontként és levélként jobbra ugyanazon szegmenstípusokat tartalmaznak, de a szegmenstípusok fában elfoglalt szintje, helye általában eltérő.

Az a célunk, hogy az egy szegmensosztályhoz egyértelműen rendelt szegmenselési fák közül kiválasszunk egyet, — a szegmensosztály mértékadó szegmensszintű adatmodelljét — mert általában az ennek megfelelő tartalmú és szerkezetű area-t fogjuk majd a tárolón megvalósítani. A fenti cél eléréséhez a $\overline{\overline{B}}$ mátrixot oszloponként kiegészítve, $b_{t,r} \neq 0$ elemhez ($b_{t,r} \in \overline{\overline{B}}^*$) további jelölést téve bejelöljük, hogy egy-egy szegmenselési fában az érintett szegmenstípusok milyen relatív szintszámmal jellemezhetők. Az így kiegészített $\overline{\overline{B}}^*$ mátrixot $\overline{\overline{F}}$ mátrixnak nevezzük.

4.1.3. Mértékadó szegmensszintű adatmodell kiválasztása szegmensosztályból

Bár egzakt módon is eljárhatnánk, a szegmensosztály mértékadó szegmensszintű adatmodelljét a tervezői gyakorlat igényelte pontossággal szemlélet alapján is jól kiválaszthatjuk egy szegmensosztály szegmenselési fái közül. Ehhez elsőként az $\overline{\overline{F}}(T, R)$ mátrix egy-egy oszlopa által képviselt relációkat grafikusán vázoljuk. A szegmensosztályhoz egyértelműen tartozó funkciókra egyenként is megállapítva az érintett szegmenstípusok navigációs kapcsolatait, a kapott fastruktúrákat összevonással egygyé olvasztjuk, és az így összevont struktúrához a szegmensosztály egésze szempontjából érvényes hierarchiaszinteket állapítunk meg. Az összevonás alapja természetesen egyrészt az, hogy funkcióként a szegmensszintű adatmodellek mely elemei esnek egybe, másrészt az, hogy ez utóbbi egybeesők között fennálló relációk mennyiben esnek egybe.

Ezeket a lépéseket elvégezhetjük a számítógép segítségével is, hiszen kisebb feladatok esetén a gyakorlatban feltehetően csupán néhány tucat gráf közül kell kiválasztani a mértékadó szegmensszintű adatmodellnek megfelelő struktúrát. Ha ezt minden szegmensosztályra nézve már végrehajtottuk, akkor előállítottuk valamennyi mértékadó szegmensszintű adatmodellt.

4.1.4. Mértékadó szegmensszintű adatalmodell összevonása erdővé

Az egy szegmensosztályon belüli mértékadó szegmensszintű adatmodellek kiválasztása és ezzel az area belső szerkezetének kijelölése után kezdünk foglalkozni a több szegmensosztályhoz is kötődő funkciókkal. Ennek az a célja, hogy az előbb meghatározott mértékadó szegmensszintű adatalmodellek közötti kapcsolatokat úgy alakítsuk ki, hogy főleg a több szegmensosztályból is merítő funkció adatigényeire legyünk tekintettel. Itt két eset van:

- a) Az összeférhetőségi szabály esete: Több szegmensosztály közti kapcsolatot képviselő funkciók szóba jöhető szegmensszintű adatalmodelljei úgy kapcsolják össze a szegmensosztályok mértékadó adatalmodelljeit, hogy az egyszerűen azok egyesítését jelenti, külön-külön tekintett belső szerkezetük legcsekélyebb változtatása nélkül.
- b) Több szegmensosztály mértékadó szegmensszintű almodelljeinek összekötését a szóba jöhető szegmensszintű adatalmodellek úgy valósítják meg, hogy lesz olyan mértékadó szegmensszintű adatmodell, melynek belső szerkezete is megváltozik. Ezt lehetőleg kerüljük el. Amennyiben azonban erre mégis sor kerül, 4.1.3-tól indulva korrekciós meneteket kell végrehajtani mindaddig, míg az összeférhetőségi szabály alkalmazhatóvá válik.

4.2. „B” stratégia lényeges elemei

Ha hiányoznak az „A” stratégia alkalmazásához szükséges előfeltételek, akkor a „B” tervezési stratégia alkalmazása válhat célszerűvé. A „B” stratégia az alábbi tevékenységek végrehajtását jelenti:

- ba) Meghatározzuk minden egyes funkcióra a szegmensszintű adatmodelljének belső hierarchikus szerkezetét jellemző szegmens elérési fát (ezt ugyanúgy tesszük, mint az „A” stratégia ab) lépésénél).
- bb) A szegmenselérési fákat erdővé vonjuk össze az összeférhetőségi szabály figyelembevételével (ezt lényegében hasonlóan tesszük, mint az „A” stratégia ad) lépésénél).
- bc) Az összeférhetőségi szabállyal ellentétben álló esetekben a szegmenselérési fák gyökerét kijelölő vezérlőinformációt (ahol arra lehetőség van)

célszerűen megváltoztatva és más gyökérpontot kijelölve törekszünk az összeférhetőségi szabállyal való összhang megteremtésére és véglegesítjük az erdőt alkotó struktúrát.

- bd) Létrehozuk a szegmenselési fák alkotta erdő inverzét reprezentáló ún. Gozinto-gráfot (vagy más néven: beépülési gráfot).
- be) A Gozinto-gráfot alkalmas módon particionáljuk.
- bf) A Gozinto-gráf gyökér-, csomó-, és terminális pontjaihoz sorra elhelyezési módot rendelünk.
- bg) A Gozinto-gráf particióit figyelembe véve rögzítjük az egyes area-k tartalmát.

A szegmenstervezés végeztével kölcsönösen egyértelműen adott az, hogy egy szegmenstípus mely adattípusokat tartalmaz. Ismert az elvi adatmodell helyzetfelmérési adataiból az is, hogy egy adott funkció mely adattípusokat igényli, milyen elérési sorrendben (precedencia-sorrend). E két információból megkonstruálható egyrészt az, hogy egy adott adattípust mely funkciók és milyen sorrendben igényelnek, továbbá az is, hogy egy adott szegmens-típust mely funkciók és milyen sorrendben igényelnek.

Az „A” tervezési stratégiában leírt teendők sorozata oda vezet, hogy szegmenselési fából — igaz, ott csupán a mértékadó szegmensszintű adatmodellből — végül is egy erdőt kell összeolvasztani úgy, hogy a többszörösen előforduló szögpontokat feltárva ugyanazon szögpont az erdőben csak egyszer szerepeljen, mégpedig azon a hierarchiaszinten, ahol a szóban forgó szögpont a legalsó szinten fordul elő. Ezzel az erdő kezdeti létrehozásánál, de a későbbi naprakész állapotra hozásánál is, csaknem pontosan olyan műveletek kell végrehajtani, mint amelyek a determinisztikus bruttó szükséglettervezés adatbázisának kialakítása és karbantartása során feltétlenül szükségesek. Szemlélet alapján megállapítható, hogy analógia áll fenn a szükséglettervezésnél kezelt gyártmánycsaládfa-rendszer és az információfeldolgozás során kezelt szegmensszintű adatmodellek (elérési fák) összessége alkotta erdő szerepe, struktúrája, kezelési szempontjai és az arra alkalmazható módszerek tekintetében. A szegmensszintű adatmodellek alkotta erdő kezelésére igen alkalmasak azok a módszerek, amelyeket a gyártmánycsaládfa-rendszerek kezelésére fejlesztettek ki. Anélkül, hogy az előforduló fogalmakat definiálnánk és a gondolatmenetet részletesen ismertetnénk, csak igen tömör, a lényegre utaló összefoglalást adunk. A determinisztikus szükséglet kiszámítására alkalmas módszerek a következők:

1. Felépítő módszerek:

1.1. Gyártási fokozat módszer

1.2. Diszpozíciós módszer

2. Beépülési (Gozinto) módszerek:

2.1. Gozinto-lista módszerek

2.2. Gozinto-mátrix módszerek

Mindegyik csoportnál más és más az a rendezési ismerv, amely szerint a gyártmánycsaládfa-rendszer bejárása megtörténik. A módszerek közötti különbségek éppen ebből adódnak.

A beépülési (Gozinto)-módszerek esetében a gyártmánycsaládfák al-kotta gyártmánycsaládfa-rendszer (erdő) hatékony kezelése érdekében elő-ször létre kell hozni az annak inverzét alkotó beépülési gráfot, amely azután vagy ritka mátrix, vagy — még hatékonyabban — lista alakjában kezelhető. A Gozinto-lista módszerek közös jellegzetessége az, hogy egyaránt az egy szögpontról kiinduló irányított élek számát (azaz a beépülések számát) használják fel a későbbi bejárások sorrendjének vezérlésére. Végeredmény-ben az állapítható meg, hogy a beépülési módszerek csoportjából a Gozinto-lista módszerek azok, amelyek leghatékonyabban alkalmazhatók. Az eljárás vezérlése (sorrendje) kizárólag a gyártmánycsaládfa-rendszer belső struktú-rája által meghatározott ún. beépülési számtól függ. Ha a gyártmánystuk-túra elemeit olyan logikai sorrendben tároljuk, amely eleve a fenti sorrendet követi, akkor egyrészt feldolgozásuk (az erdő bejárása) viszonylag igen gyor-san elérhető, másrészt felismerhető az is, hogy az eredeti családfarendszer gyökérpontjaiból indított bejárás a feldolgozás folyamata egy-egy meneté-ben mindig csak egy jól körülhatárolt és vertikálisan kimetszett részgráfra terjed ki.

A Gozinto-lista módszernek az a sajátossága, hogy lehetővé teszi a több szintre tagolt irányított gráf egyértelmű particionálását, hogy az egyes fa-gráfok gyökerei ismertek legyenek, mert a particionálás folyamatát ezek-ből indítva lehet elvégezni. Az erdő particionálása itt azt jelenti, hogy egy gyökérből kiindulva a hozzá tartozó fát végig bejárjuk úgy, hogy egy cso-mópontból az odamutató éleken sorra végighaladunk, azaz a szóban forgó élek túlsó végén elhelyezkedő szögpontokhoz tartozó beépülési számokat sorra eggyel csökkentjük. Ha eközben valamelyik szögpont beépülési száma nulla lesz, akkor most az e szögpontról mutató éleken (mint előbb a gyökér esetében tettük) kezdünk el sorra továbbhaladni, s az előző csomópontból

(illetve gyökérbe) mutató további élek bejárását átmenetileg mindaddig felfüggesztjük, amíg ez az újonnan indított folyamat véget nem ér. Az egy gyökérből elindított és az adott fához tartozó ilyen bejárás eredménye egy, az erdőből vertikálisan kimetszett részgráf.

Tegyük fel, hogy ismertek azok a fák, amelyek egymással szorosan összefüggnek (azaz sok közös szögpontjuk van). Először ezek gyökereit helyezzük el egymás mellett ún. gyökér-listába, s csak később a velük laza kapcsolatban lévők (kevesebb közös szögpont) gyökereit. Amennyiben még nem lenne ismert, hogy mely fák állanak egymással szoros kapcsolatban, akkor ezt az erdő első bejárása során határozzuk meg úgy, hogy egy tetszőleges gyökérből kiindulva az odamutató élek végén elhelyezkedő valamennyi szögpontból rendre tovább megyünk az azokból máshová mutató élek alkotta utakon egészen a többi gyökérig. Amint egy gyökérhez így eljutottunk, a hozzárendelt kapcsolati mérőszámot eggyel megnöveljük. A bejárást tovább folytatjuk, amíg csak a levelekig is el nem jutunk. Az erdő bejárását ezután azzal a gyökérrel folytatjuk, amelynek a legnagyobb a kapcsolati mérőszáma. A beépülési gráfok szerinti lebontás előnyösebb, mint a fagrafok bejárása szerinti lebontás. Itt nincs szükség arra, hogy az egyes szegmens-elérési fákat kirajzolva, azokból erdőt szerkesszünk, meghatározva eközben az egyes szögpontok legalsó szintjét, az ún. diszpozíciós szintet. Ehelyett a szegmenstípusokból, vagyis az egyszintű hierarchialépcsőkből kell kiindulni. Ezekből három adatállományt építünk fel:

- Az elsőt szegmenstípusok azonosítói és a beépülési számok alkotják egy indexszekvenciális törzsszállományba szervezve.
- A másodikat a közvetlen egyszintű hierarchikus kapcsolatban álló szegmenstípusokból töltjük fel, egymás után betöltve egy hierarchialépcsőt egy, a fa- és inverz- (beépülési) struktúrát rögzítő ún. struktúraállományba, amely a törzsszállományhoz van láncolva. A betöltés során minden egyes szegmenstípushoz ún. beépülési láncot alakítunk ki, és rögzítjük, hogy az egyes láncok hány elemből állnak (beépülési szám). A beépülési lánc fejére mutató pointert és a beépülési számot a törzsszállományban tároljuk. Ezt elvégezve, az egyszintű hierarchialépcsőből felépítjük egyrészt a redundancia mentes erdőt, másrészt az annak inverz struktúráját hordozó Gozinto-gráfot is. A továbbiakban ezt a Gozinto-gráfot célszerű kezelni szegmensosztály-képzés helyett, ezzel oldva meg a gráf particionálását.

– A harmadik indexállomány az erdő bejárasi sorrendjét rögzíti.

A Gözinto-gráfok kezelésére, az adatállományok szervezésére más alternatívák is léteznek.

5. fejezet

Az osztályozást megvalósító program

A „DBC” program adatbázisok tervezéséhez nyújt hatékony számítógépes segítséget. A program IBM PC, XT, AT és ezekkel kompatibilis számítógépeken fut PC-DOS és MS-DOS operációs rendszerek alatt.

5.1. A program segítségével elemezhető adatbázis-tulajdonságok

A „DBC” programnak nem célja, hogy egy adott adatbázis-kezelő rendszer használata esetén a felhasználó által meghatározható jellemzők teljes választékát kezelje. Ehelyett olyan modellt használ, amelyen az adatbázisban tárolt adatok és az adatbázis használata során végrehajtandó feladatok kapcsolata elemezhető.

Az adatbázis több egyed típus elemi adatait tartalmazhatja. A „DBC” program számára ezeket az egyed típusokat sorszámmal kell ellátni. Szintén sorszámok azonosítják az elemi adattípusokat és az adatbázison végrehajtható tevékenységeket vagy funkciókat.

Minden egyed típusra vonatkozóan meg kell adni legalább egy mátrixot, amelynek annyi sora van, ahány adattípus szerepel az egyed típusban, valamint annyi oszlopa, ahány funkció az egyed típus adatait használja.

A mátrix egy sorának és egy oszlopának közös eleme kifejezi a megfelelő adat és funkció kapcsolatát: 0 érték esetén nincs kapcsolat, 0-nál nagyobb érték esetén pedig van.

Kétféle lehetőség van arra, hogy a 0-tól eltérő értékeket megadjuk. Ha csak annyit tudunk, hogy a funkcióhoz szükséges egy adat, de semmilyen további információ nem áll rendelkezésünkre vagy nem kívánjuk figyelembe venni, egyszerűen 1-gyel jelezzük ezt a helyzetet. Ekkor a mátrix csak 0-t és 1-et tartalmaz, ez a bináris eset.

Ha két adat-funkció-kapcsolatot ennél finomabban is meg tudunk különböztetni egymástól, a nem 0 értékeket pozitív egész számokkal adjuk

meg. Ebben az esetben úgy gondoljuk, hogy nagyobb szám valamilyen módon erősebb vagy gyakoribb kapcsolatot jelent.

Ennek egyik esete, amikor két funkció végrehajtási gyakorisága eltérő. Ilyenkor az adat-funkció-mátrix a funkcióhoz tartozó gyakoriságvektor (vagy súlyvektor) és egy bináris mátrix segítségével adható meg. Ekkor a mátrix egy oszlopában csak kétféle érték fordulhat elő: 0 és a funkció súlya.

A nem bináris mátrix megadásánál a legáltalánosabb esetben egyéb szempontokat is figyelembe véve, a mátrix minden elemét tetszőlegesen adhatjuk meg.

5.2. A program tevékenységei

A program a megadott adatokat két fázisban dolgozza fel. Az 1. fázisban minden egyedtípus feldolgozása önállóan történik meg, az eredmény nem függ más egyedtípusoktól. A 2. fázis egyesíti az 1. fázis egyedtípusokra vonatkozó eredményeit egyetlen, a teljes adatbázis-modellre vonatkozó mátrixszá.

Az 1. fázis során az egyedtípus adattípusait csoportosítjuk. Ennek a csoportosításnak az alapja egy olyan mérőszám, amely azt mutatja meg, hogy az adat-funkció-mátrix két sora mennyire hasonlít, illetve mennyire tér el egymástól. Ez a mérőszám bináris esetben a hasonlóságmérték, nem bináris esetben pedig a távolságmérték.

A csoportosítást végrehajtó algoritmus a legközelebbi szomszédokat tartalmazó párokból kiinduló agglomeratív eljárás, amelynek a következő tulajdonságai vannak:

- a) diszjunkt csoportokat eredményez,
- b) mindegyik adatot besorolja valamelyik csoportba,
- c) nem igényel semmilyen algoritmus-leállító paramétert,
- d) az eredményül kapott csoportok mindegyike legalább 2 elemet tartalmaz.

Az 1. fázis csoportosító eljárásának eredményei, a szegmensek, egyben a 2. fázis kiindulópontjai.

A felhasználónak lehetősége van arra is, hogy bármelyik egyedtípusra saját maga adjon meg szegmenseket, amelyeket más módon, ettől a prog-

ramtól függetlenül határozott meg, és ezt kívánja használni. Ezt a továbbiakban szegmensjavaslatnak nevezzük.

A program elkészít egy bináris szegmens-funkció-mátrixot, melynek sorai a szegmensekhez, oszlopai pedig a funkciókhoz tartoznak.

Ha egy mátrixelem 1, akkor az oszlophoz tartozó funkciónak szüksége van a sorhoz tartozó szegmens valamely adatára, ha pedig 0, akkor semelyiket sem használja.

A 2.fázis feladata ennek a mátrixnak a sor- és oszlopcserékkel történő átrendezése oly módon, hogy az 1 értékek lehetőleg tömör blokkokban helyezkedjenek el. Ezekhez a blokkokhoz tartozó szegmensek lesznek az adatbázis azon részei, amelyeknek tárolását célszerű úgy megoldani, hogy az egyik adat elérése esetén a másik már könnyen megszerezhető legyen.

5.3. A program használata

A program futása során a képernyőt ablakokra bontja, és az egyes ablakokat különböző funkciókra használja. Ilyen funkciók:

- a) a program által használt állományok meghatározása,
- b) választás a végrehajtható funkciók közül,
- c) az output tartalmának meghatározása,
- d) hibajelzések,
- e) tájékoztatás a program futásáról.

A program teljes futása 6 szakaszból áll:

- 1) adatellenőrzés;
- 2) hasonlóság és távolságmátrixok elkészítése,
- 3) szegmensek létrehozása,
- 4) a teljes adatbázisra vonatkozó szegmens-funkció-mátrix előállítása,
- 5) sor- és oszlopcserék a szegmens-funkció-mátrixon,
- 6) output készítése

E szakaszok befejezésekor a program a képernyő közepén lévő tájékoztató ablakban kiírja az éppen lefutott szakasz számát, időtartamát és a program futása során eddig a szakaszig felhasznált teljes számítási időt.

Egy teljes futás során a program 8 állományt használ, amelyek egyetlen periféria egyetlen katalógusába (directory) vannak bejegyezve. Az alapértelmezés szerinti egység a C:. A program csak a kiválasztott periféria alapértelmezés szerinti katalógusát használja!

A program indulásakor meg kell adni, ha a C:-től eltérő egységet kívánunk használni. A "Periféria:" kérdésre a RETURN-t megnyomva, az alapértelmezés lép életbe, az ettől eltérőt egy betűvel kell megadni, a: nélkül.

A használt 8 állomány neve azonos, csak a kiterjesztésük tér el egymástól:

- .WK1 munkaállomány, a futás után megmarad; tartalma: belső sorszá-
mok, az átrendezett szegmens- és funkció-sorszámok
- .WK2 munkaállomány, a futás után megmarad; tartalma: az input mát-
rixok és az 1. fázis számításainak eredménye
- .WK3 munkaállomány, a futás után megmarad; tartalma: az egyedtípusok
változatai közül a kiválasztottak sorszáma
- .WK4 munkaállomány, a futás után megmarad; tartalma: az átrendezett
szegmens-funkció-mátrix
- .WK5 munkaállomány, a futás során törlődik; ez az állomány csak a 2. fá-
zis során szükséges, helyigénye kb. $5 \cdot L \cdot L$ byte, ahol L a szegmens-
funkció-mátrix sor- és oszlopszáma közül a nagyobb.
- .DBC az input adatok állománya
- .NEV opcionális állomány, amely az egyedtípusok, az adattípusok és a
funkciók elnevezéseit tartalmazza. Ha ez az állomány az 1. fázis
futtatásakor létezik, a program az egyedtípusok, adattípusok és
funkciók nevét is használja. Ebben az esetben a későbbi tablóké-
szítéskor ez az állomány is szükséges; a helyes működésnek az a
feltétele, hogy a tartalma ne változzon meg.
- .LSn a futás eredményét tartalmazó tabló, ahol az n értéke utal a tabló
tartalmára:
 - $n = 0$ teljes tabló, amely az input mátrix értékeit is tartalmazza
 - $n = 1$ teljes tabló, amely az input mátrix értékeit nem tartalmazza
 - $n = 2$ az egyedtípusoknak csak a kiválasztott verzióit tartalmazó tabló,
az input mátrix értékeivel
 - $n = 3$ az egyedtípusoknak csak a kiválasztott verzióit tartalmazó tabló,
az input mátrix értékei nélkül

- $n = 4$ csak egyetlen egyedtípus adatait tartalmazza az input mátrix értékeivel
- $n = 5$ csak egyetlen egyedtípus adatait tartalmazza az input mátrix értékei nélkül
- $n = 6$ csak egyetlen egyedtípus kiválasztott verziójának adatait tartalmazza az input mátrix értékeivel
- $n = 7$ csak egyetlen egyedtípus kiválasztott verziójának adatait tartalmazza az input mátrix értékei nélkül
- $n = 8$ csak a 2. fázis eredményét tartalmazza

A program második kérdése az adatbázis-modell neve. Ekkor kell megadni az előbbi állományok közös nevét, kiterjesztés nélkül. Ilyenkor a program megállapítja a lehetséges futtatási módokat. A bal oldali ablakban a következő funkciók jelenhetnek meg:

- 0: Kilépés
- 1: Hibaellenőrzés
- 2: Hibaellenőrzés, 1. fázis
- 3: Hibaellenőrzés, 1-2. fázis
- 4: Hibaellenőrzés, 1-2. fázis+tábla
- 5: 2. fázis
- 6: 2. fázis, tábla
- 7: Tábla az 1. fázisról
- 8: Tábla az 1-2 fázisról
- 9: Új adatbázis

Ha van .DBC állomány, lehetséges:

- hibaellenőrzés
- hibaellenőrzés és az 1. fázis futtatása
- hibaellenőrzés, az 1. és 2. fázis futtatása
- hibaellenőrzés, az 1. és 2. fázis futtatása tablóképzéssel

Ha van .WK1 és .WK2 állomány, lehetséges:

- 2. fázis futtatása
- 2. fázis futtatása tablóképzéssel

Ha van .WK1, .WK2 és .WK3 állomány, lehetséges:

- tablóképzés az 1. fázisról (LSn állományok $n=0$ -tól 7-ig)

Ha van .WK1, .WK2, .WK3 és .WK4 állomány, lehetséges:

- bármely tabló elkészítése

A felkínált menü mindig a lehetséges összes funkciót tartalmazza, ebből kell a megfelelő számút kiválasztani. Ha semmilyen funkció nem hajtható végre (ekkor az ablakban csak a 0. és a 9. lehetőség jelenik meg), a program a választól függően kilép, vagy új adatbázis-modell nevet kér.

Ha tablóképzéssel együttjáró funkciót választottunk, akkor a tabló tartalmát is megkérdezi, a következő formában: a jobb oldali ablakban megjelenik a lehetséges választék:

- 0: Teljes tábla
- 1: Teljes tábla input nélkül
- 2: Kiválasztott verziók
- 3: Kiválasztott verziók input nélkül
- 4: Egy egyedtípus
- 5: Egy egyedtípus input nélkül
- 6: Egy egyedtípus kiválasztott verzió
- 7: (6)-os, input nélkül
- 8: Csak a 2. fázisról
- 9: Új funkció

Ekkor a megfelelő számmal a kívánt output kiválasztható. Ha a 4-től 7-ig terjedők közül választottunk, a program még azt is megkérdezi, hogy melyik egyedtípusról kérjük a listát. Ha korábban a 7-es funkciót (Tábla az 1. fázisról) választottuk, az outputon nem fog szerepelni a 2. fázis eredménye, ellenkező esetben pedig azt is megkapjuk. A 8-as funkciót és a 8-as listát választva, csak a 2. fázis eredményét kapjuk meg.

Ha a Periféria: vagy az Adatbázis: vagy Egyedtípus száma: kérdésre CTRL/Z-t, vagy az Adatbázis kérdésre RETURN-t válaszolunk, a program kilép. Ha az Egyedtípus száma: kérdésre 0-t válaszolunk, új funkcióválasztást kér a program. Minden válasz után, még az egy karakteresek után is RETURN-t kell nyomni.

Mindig a választott listának megfelelő nevű (.LSn kiterjesztésű) állományba kerül az output.

Ha nincs .DBC kiterjesztésű állomány, akkor csak eredménytabló készítés lehetséges.

5.4. Az input adatállomány szerkezete

Az input adatállomány sorokból áll, szövegszerkesztővel készíthető el. A sorok tartalma eltérő, az állománybeli szerepüknek megfelelően. A következő sortípusok fordulnak elő:

- a. feladatleíró sor,
- b. egyedtypus-leíró sor,
- c. adattípus-azonosítókat tartalmazó sor,
- d. funkcióazonosítókat tartalmazó sor,
- e. mátrixleíró sor,
- f. funkció súlyokat tartalmazó sor,
- g. bináris mátrix adatsora,
- h. numerikus mátrix adatsora,
- i. szegmensjavaslat adatsora.

Az egyes sortípusok részletes szerkezete, szerepe és előfordulása az input adatállományban:

1. Feladatleíró sor:

Az állomány első sora, csak egy van belőle, 3 számot tartalmaz:

DB1 Az egyedtypusok száma, az input állománynak pontosan ennyi egyedtypus adatait kell tartalmaznia $1 \leq DB1 \leq 256$

MXA A maximális adattípus azonosító sorszáma, ennél nagyobb számú adattípus nem fordulhat elő $2 \leq MXA \leq 999$

MXF A maximális funkcióazonosító sorszáma, ennél nagyobb számú funkció nem fordulhat elő $1 \leq MXF \leq 512$

2. Egyedtypus-leíró sor:

Az egyedtypushoz tartozó adatok közül az első sor, egyedtypusonként egy van belőle, 4 számot tartalmaz:

SSZ1 Az egyedtypus sorszáma, azonosítja az egyedtypust. Egy egyedtypus sorszáma csak egyszer fordulhat elő. $1 \leq SSZ1 \leq 512$

M Az egyedtypusba tartozó adattípusok száma, az egyedtypushoz tartozó mátrix(ok) sorainak száma $2 \leq M \leq 64$, $2 \leq M \leq MXA$

N Az egyedtypus adatait használó funkciók száma, az egyedtypushoz tartozó mátrix(ok) oszlopainak száma. $1 \leq N \leq 128$, $1 \leq N \leq MXF$

DB2 Az egyedtípus mátrixa DB2 féle változatban szerepel az állományban, ezek mindegyike alapján elkészülnek a szegmensek, de a 2. fázis ezek közül csak egyet használhat, a futás során ezt a program jelzi.

$$1 \leq DB2 \leq 32$$

3. Adattípus-azonosítókat tartalmazó sor:

Közvetlenül az egyedtípus-leíró sor után áll, az egyedtípusba tartozó M darab adattípus-azonosító sorszámaait tartalmazza.

Minden sorban 10 szám van, az utolsó sorban természetesen 10-nél kevesebb is állhat. Annyi ilyen sor van, hogy a soronkénti 10 szám az utolsó, esetleg csonka sorban lévőkkel együtt pontosan M darabot adjon.

A sor minden x elemére: $1 \leq x \leq M X A$

4. Funkcióazonosítókat tartalmazó sor:

Az adattípus-azonosítókkal azonos formátumban tartalmazza az egyedtípussal kapcsolatban álló N darab funkció-azonosító sorszámaát.

A sor minden x elemére: $1 \leq x \leq M X F$

5. Mátrixleíró sor:

Az egyedtípus különböző változatait azonosítja, minden egyedtípusban DB2 darab van, mindegyik egy mátrixot vagy egy szegmensjavaslatot ír le; 3 számot tartalmaz:

SSZ2 Az egyedtípuson belüli változat sorszáma, a különböző változatok sorszámanak 1-től egyesével kell növekednie DB2-ig. $1 \leq SSZ2 \leq DB2$

TIP A-változat típuskódja, lehetséges értékei:

- 0: szegmensjavaslat
- 1: bináris mátrix
- 2: nemnegatív egész értékű mátrix
- 3: súlyozott bináris mátrix

CN A szegmensjavaslatban szereplő szegmensek száma, csak $TIP=0$ esetén van szerepe, ekkor $1 \leq CN \leq M$

6. Funkciósúlyokat tartalmazó sor:

Ha a mátrixleíró sorban TIP értéke 3, akkor ezen sor(ok)ban adjuk meg az egyes funkciók súlyát, amelyek nemnegatív egész értékek lehetnek, formátuma és a sorok száma a funkcióazonosítók sorával azonos, mert minden funkcióhoz egy súly tartozik.

7. Bináris mátrix adatsora:

Ha TIP értéke 1, akkor a mátrix adatait ez a sor (vagy sorok) tartalmazza; a sorban csak 0 vagy 1 vagy szóköz fordulhat elő, a szóköz 0-t jelent. Egy sorban 80 értékes karakter lehet, ha 80-nál több funkció van. Egy mátrixsorhoz 2 ilyen sor tartozik az adatállományban, a másodikban legfeljebb 48 értékes karakter lehet, mert egy egyedítípushoz tartozó funkciók maximális száma 128.

8. Numerikus mátrix adatsora:

Ha TIP=2, akkor a mátrix nemnegatív egész számokból áll, ezeket tartalmazza ez a sor, formátuma és az egy mátrixsorhoz tartozó sorok száma a funkcióazonosító sorokéval azonos, mert a mátrix minden sora annyi számot tartalmaz, ahány funkcióval az egyedítípus kapcsolatban áll.

9. Szegmensjavaslat adatsora:

Akkor szerepel, ha a mátrixleíró sorban TIP értéke 0 volt, ez a változat azonban nem lehet az egyedítípus változatai közül az első. A szegmensjavaslatot úgy kell elkészíteni, hogy az egyes szegmenseket meg kell számozni 1-től CN-ig és elő kell állítani egy M elemű vektort, amelynek i . eleme azt mondja meg, hogy az i . adattípus hányadik szegmensbe tartozik. Ennek a vektornak az értékeit tartalmazza ez a fajta sor, formája és a sorok száma az adattípus-azonosítók sorával egyezik meg, mert minden adattípushoz egyetlen szegmens száma tartozik. Fontos, hogy az elkészülő tablók nem tartják meg ezt a számozást, mert a több egyedítípus eredményének összevonása után is meg kell tudni különböztetni egymástól az egyes szegmenseket. A tablók azonban elegendő információt tartalmaznak az azonosításhoz, mert az új szegmensszámok úgy keletkeznek, hogy minden egyedítípus saját számaihoz hozzáadódik az előző egyedítípusok szegmenseinek a darabszáma.

Az input-állomány felépítése ezekből a sorokból:

- a. Feladatleíró
- b. Egyedítípus-leíró
- c. Adattípus-azonosító
- d. Funkcióazonosító
- e. Mátrixleíró
- f. Bináris adatok sorai
- g. Numerikus adatok sorai
- h. Szegmensjavaslat
- i. Súly és bináris adatok

A b.-i. DB1-szer ismétlődő blokk, az e.-i. DB2-szor ismétlődő blokk, és a f.-i. TIP-től függően valamelyik.

Az egyes adatsorok és a bennük szereplő adatelemek írási szabályai:

1. Az egész számok előtt a szóköz megengedett.
2. Az egész számok között a szóköz a nullával azonos.
3. Az egész számok után egyetlen elválasztó karakternek kell állnia. Elválasztó karakternek számot a számjegyeken és a szóközön kívül minden. Két, közvetlenül egymás után álló elválasztó karakter azonos azzal, mintha egy zérus lenne közöttük. Ha a sorban nincs megjegyzés, az utolsó szám után nem kell semminek lennie, de itt is ajánlott egy elválasztó karakter, mert egy, az utolsó szám után álló szóköz képernyőn vagy kinyomtatva nem látható, de a szám értékét megváltoztatja.
4. Az utolsó egész szám után megjegyzés állhat a sorban.
5. Bináris adatokat tartalmazó sorban minden karakter egy értéket jelent, a 0 és a szóköz bináris 0-t, az 1 pedig 1-et, minden más karakter hibás, a sor ki nem használt részén megjegyzés állhat.
6. A program számábrázolási tartománya 32767-ig terjed.
7. Maximális sorhossz 127 karakter.

5.5. Elnevezések kezelése

Ha az egyedtypusokat, az adattipusokat és a funkciókat az azonosító sorszámokon kívül nevekkkel is el akarjuk látni, akkor az adatbázis-modell állományok között szerepelnie kell a „NÉV” kiterjesztésűnek is. Ennek az állománynak minden sora egy nevet tartalmaz. A sorok formája: rövidítés,sorszám,név

rövidítés:	E	egyedtypus,
	A	adattipus,
	F	funkció,

sorszám:	pontosan 3 jegyű azonosító sorszám, ha ez 3-nál kevesebb jegyű, akkor a szám elé 0-t vagy szóközt kell írni,
----------	--

név:	egy legfeljebb 20 karakteres megnevezés.
------	--

A sor mezőit vesszővel (,) kell elválasztani egymástól!

Az állományban a nevek sorrendje tetszőleges lehet. Ha egy elemet több névvel is ellátunk, az utolsót fogja a program használni, ha pedig valaminek nem adunk nevet, az eredményen név nélkül fog szerepelni.

5.6. Hibakezelés

A program a számításokat csak formai hibákat nem tartalmazó input állomány esetén végzi el. A hibák kiszűrésére szolgál az adatellenőrző fokozat. Ha ez a programrész hibát talál az állományban, a képernyőn jelzi, hogy mi a hiba, behatárolja annak helyét és ha lehetséges, közli a hibás adatot is.

A hibák egy része olyan természetű, hogy a hibakeresés nem folytatható, mert az állomány további szerkezetét meghatározó adat volt hibás, vagy egyáltalán nem is létezik. Ekkor a program befejezi az állomány feldolgozását.

Más hibák előfordulása esetén a megjelenő kérdésre válaszolva lehet eldönteni, hogy folytatódjon-e a hibakeresés, vagy lépjen ki a program az adatellenőrző fokozatból.

Ha volt hiba az állományban és minden esetben a folytatást választottuk, az ellenőrző fokozat lefutása után közli a program, hogy a hibák miatt a számítások nem lesznek elvégezve.

Két olyan helyzet van, amely csak a program 2. fázisának futását teszi lehetetlenné, az 1. fázis eredményei ekkor készen vannak és hozzáférhetőek. Az egyik akkor következik be, ha az 1. fázis futása során 512-nél több szegmens keletkezik. Ekkor a szegmens-funkció-mátrix előállítása nem lehetséges, mert ennek a program által kezelhető legnagyobb mérete 512×512 . A másik eset az, amikor olyan funkció fordul elő az adatbázismodellben, amely semelyik egyedtípus semelyik adattípusára nem hivatkozik. Ezt a tényt a program a képernyőn jelzi; felsorolja mindazokat a funkciókat, amelyek nem hivatkoznak egyetlen adattípusra sem, majd megáll.

Az adatellenőrző fokozat szöveges hibaüzeneteket jelenít meg a képernyőn, majd ezután olyan adatsorokat közöl, amelyek segítenek a hibás adat megtalálásában. Az előforduló hibaüzeneteket, kiváltó okukat és a javításuk módját a következő felsorolás tartalmazza:

1: Input megnyitási hiba (1).

Oka: a megadott adatbázis modell .DBC állományát a program nem tudja elérni.

Javítása: hozzuk létre az állományt,
állítsuk be a DOS alapértelmezés szerinti katalógusát a CD paranccsal,
adjuk meg helyesen a periféria nevét,
adjuk meg helyesen az adatbázis-állományok nevét.

2: Hiba a feladatleíró sorban.

Oka: az állomány legelső sorában levő három szám (amely a képernyőn is megjelenik) nem tesz eleget a rájuk vonatkozó kikötéseknek:

$$1 \leq DB1 \leq 256, \quad 2 \leq MXA \leq 999, \quad 1 \leq MXF \leq 512$$

Javítása: töltsük ki a sort hibátlanul.

3: Hiba az egyed típus azonosító sorban.

A feladatleíró sor:

Az egyed típus-azonosító sor:

Oka: egy egyed típus-azonosító sor valamelyik adata nem teljesíti a feltételeket:

$$2 \leq M \leq MXA, \quad 1 \leq N \leq MXF, \quad 1 \leq DB2 \leq 16.$$

Javítása: töltsük ki a sort hibátlanul; ehhez tájékoztatásul a feladatleíró sor és a hibás egyed típus-azonosító sor is a képernyőn látható.

4: Adattípus negatív vagy túl nagy.

A feladatleíró sor:

Az egyed típus-azonosító sor:

Az adattípus:

Oka: egy adattípus-azonosító (x) sorszáma nem teljesült, hogy

$$1 \leq x \leq MXA.$$

Javítása: írjuk be a helyes sorszámot, vagy ha az helyes, módosítsuk a feladatleíró sorban MXA értékét.

5: Adattípus ismétlődik.

A feladatleíró sor:

Az egyed típus-azonosító sor:

Az adattípus:

Oka: az adattípus sorszáma többször szerepel egy egyed típusban.

Javítása: számozzuk úgy az adattípusokat, hogy az egyed típuson belül egyediek legyenek az azonosító sorszáмок.

6: Funkciótípus negatív vagy túl nagy.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A funkciótípus:

Oka: egy funkció-azonosító (y) sorszáma nem teljesült, hogy

$$1 \leq y \leq MXF.$$

Javítása: írjuk be a helyes sorszámot, vagy ha az helyes, módosítsuk a feladatleíró sorban MXF értékét.

7: Funkciótípus ismétlődik.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A funkciótípus:

Oka: a funkció sorszáma többször szerepel egy egyedtípusban.

Javítása: számozzuk úgy a funkciókat, hogy az egyedtípuson belül egyedi legyenek az azonosító sorszámmok.

8: Hiba a tömbleíró sorban.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

Oka: a mátrixleíró sor tartalma hibás: SSZ2 értéke nem egyesével növekszik az egyedtípuson belül, vagy TIP értéke negatív vagy 3-nál nagyobb, vagy TIP értéke 0 és CN értéke 1-nél kisebb vagy M-nél nagyobb.

Javítása: írjuk le helyesen a sort.

9: Negatív vagy túl nagy szegmensszám.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

A szegmensszám:

Oka: a szegmensjavaslatban szereplő szegmensszám 1-nél kisebb, vagy nagyobb a mátrixleíró sorban megadott CN-nál.

Javítása: adjuk meg helyesen a szegmensszámokat, vagy ha azok jók, módosítsuk a mátrixleíró sorban CN értékét a helyesre.

10: Hiányosan megadott szegmensek.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

* A tömbleíró sor:

Oka: a szegmensjavaslatban minden szegmensben legalább 1 adattípusnak kell lenni.

Javítása: ha elírás történt, javítsuk ki, ha szükséges, számozzuk át a szegmenseket 1-től egyesével növekvőre, ha CN értéke túl nagy a mátrixleíró sorban, írjunk helyére megfelelő értéket.

11: Negatív súly.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

A súly:

Oka: egy súlyvektorban negatív érték fordult elő.

Javítása: ki kell javítani a funkció súlyát.

12: Hibás karakter bináris adatban.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

A karakter:

Oka: a bináris mátrix adatsorában csak 0 vagy 1 vagy szóköz lehet

Javítása: ki kell javítani a hibás karaktert a megfelelőre.

13: Csupa zérus sor bináris adatban.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

Oka: bináris mátrixban olyan sor fordult elő, amely nem tartalmazott 1-et, ami azt jelenti, hogy az adattípusra nincs szüksége semelyik funkciónak; ilyen adattípus nem fordulhat elő.

Javítása: ha hibás az adatsor, javítsuk ki, ha pedig nincs rá szükség, ki kell törölni, de ekkor a mátrixleíró sort is és az adattípusok azonosítóit tartalmazó sort is módosítani kell.

14: Negatív adat numerikus mátrixban.

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

Az érték:

Oka: nem bináris mátrixban negatív érték fordult elő.

Javítása: írjunk be 0-t vagy pozitív egész számot.

- 15: Csupa zérus sor numerikus mátrixban.

A feladatléíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

Oka: nem bináris mátrixban olyan sor fordult elő, amely nem tartalmazott pozitív számot, ami azt jelenti, hogy az adattípusra nincs szüksége semelyik funkciónak; ilyen adattípus nem lehet.

Javítása: ha hibás az adatsor, javítsuk ki, ha pedig nincs rá szükség, ki kell törölni, de ekkor a mátrixleíró sort is és az adattípusok azonosítóit tartalmazó sort is módosítani kell.

- 16: Felesleges adatok vannak a végén.

Oka: az utolsó egyedtípus adatai után is van(nak) még sor(ok) az állományban.

Javítása: ha tényleg az utolsó egyedtípus utolsó változatának adatai után áll(nak) a sor(ok), ki kell törölni, ha a feladatléíró sor DB1 adata nem megfelelő, akkor azt kell módosítani, ha pedig az utolsó egyedtípus leíró sorában a DB2 kisebb, mint a helyes érték, akkor azt kell módosítani.

- 17: Input lezárási hiba.

Oka: állománykezelési rendszerhiba.

Javítása: futtassuk újra a programot.

- 18: Nincs feladatléíró sor (EOF).

Oka: üres az input állomány.

Javítása: a program futtatása előtt töltsük fel az input állományt, vagy ha ez már megtörtént, adjunk meg helyes állománynevet.

- 19: Hibás feladatléíró sor (ERROR).

Oka: a program nem tudja elolvasni az input állomány 1. sorát.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

- 20: Nincs egyedtípus-azonosító sor (EOF).

Oka: a feladatléíró sor szerint még lenni kellene egyedtípus-azonosító sornak az input állományban, de annak már vége van.

Javítása: ha tényleg hiányzik adat az állományból, pótoljuk, ha pedig a feladatleíró sor DB1 adata túl nagy, írjunk oda jó számot.

21: Hibás egyed típus-azonosító sor (ERROR).

A feladatleíró sor:

Oka: a program nem tudja elolvasni az input állomány egyik egyed típus leíró sorát.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

22: Nincs adattípus-sor (EOF).

Oka: az input állomány véget ért, de a program még adattípus-azonosító sor(oka)t olvasna.

Javítása: tegyük kompletté az input állományt és futtassuk újra a programot.

23: Hibás adattípus-sor (ERROR).

A feladatleíró sor:

Az egyed típus-azonosító sor:

Oka: a program nem tudja elolvasni az input állomány egyik sorát, amelynek adattípus-azonosítókat kell tartalmaznia.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

24: Nincs funkció típus-sor (EOF).

Oka: az input állomány véget ért, de a program még funkció-azonosító sor(oka)t olvasna.

Javítása: tegyük kompletté az input állományt és futtassuk újra a programot.

25: Hibás funkció típus-sor (ERROR).

A feladatleíró sor:

Az egyed típus-azonosító sor:

Oka: a program nem tudja elolvasni az input állomány egyik sorát, amelynek funkció-azonosítókat kell tartalmaznia.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

26: Nincs tömbleíró sor (EOF).

Oka: az input állomány véget ért, de a program még mátrixleíró sort olvasna.

Javítása: tegyük kompletté az input állományt és futtassuk újra a programot.

27: Hibás tömbleíró sor (ERROR).

A feladatleíró sor:

Az egyedtípus-azonosító sor:

Oka: a program nem tudja elolvasni az input állomány egyik mátrixleíró sorát.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

28: Nincs szegmensjavaslat-sor (EOF).

Oka: az input állomány véget ért, de a program még szegmensjavaslat-sor(oka)t olvasna.

Javítása: tegyük kompletté az input állományt és futtassuk újra a programot.

29: Hibás szegmensjavaslat-sor (ERROR).

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

Oka: a program nem tudja elolvasni az input állomány egyik szegmensjavaslat adatsorát.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

30: Nincs bináris adatsor (EOF).

Oka: az input állomány véget ért, de a program még bináris mátrixhoz tartozó adatsor(oka)t olvasna.

Javítása: tegyük kompletté az input állományt és futtassuk újra a programot.

31: Hibás bináris adatsor (ERROR).

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

Oka: a program nem tudja elolvasni az input állomány egyik bináris mátrix adatsorát.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

32: Nincs súly-adatsor (EOF).

Oka: az input állomány véget ért, de a program még funkció súlyokat tartalmazó adatsor(oka)t olvasna.

Javítása: tegyük kompletté az input állományt és futtassuk újra a programot.

33: Hibás súly-adatsor (ERROR).

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

Oka: a program nem tudja elolvasni az input állomány egyik súlyokat tartalmazó sorát.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

34: Nincs numerikus adatsor (EOF).

Oka: az input állomány véget ért, de a program még nem bináris mátrixhoz tartozó adatsor(oka)t olvasna.

Javítása: tegyük kompletté az input állományt és futtassuk újra a programot.

35: Hibás numerikus adatsor (ERROR).

A feladatleíró sor:

Az egyedtípus-azonosító sor:

A tömbleíró sor:

Oka: a program nem tudja elolvasni az input állomány egyik nem bináris mátrixnak adatsorát.

Javítása: győződjünk meg az állomány olvashatóságáról és futtassuk újra a programot.

5.7. Az output listák tartalma

Az 5.3. pontban már szerepelt, hogy a program a különböző tartalmú eredménylistákat az adatbázismodellel megegyező nevű, de különböző kiterjesztésű állományokban helyezi el. Ebben a részben szerepel az egyes listák adattartalma és szerkezete. Részletesen csak a teljes „LS0” állomány tartalmát írjuk le, mert a többi ennek valamely részét tartalmazza, ezért azoknál csak az szerepel, hogy teljes állomány mely része található meg bennük.

A teljes lista (.LS0) követi az input állomány szerkezetét:

- a) az output lista tartalmazza az adatbázisodell nevét, az egyedtípusok, az adattípusok és a funkciók számát és a feladateleíró sorban szereplő maximális azonosítókat (MXA, MXF).
 - b) ezután minden egyedtípusról egy azonos szerkezetű blokk következik:
 - ba) az egyedtípus-azonosítója, adattípusainak és funkcióinak a száma, az input állományban lévő mátrixváltozatok száma, valamint annak a változatnak a sorszáma, amelyet a 2. fázis során figyelembe vett a program,
 - bb) az adattípusok azonosítóinak felsorolása,
 - bc) a funkciók azonosítóinak felsorolása,
 - bd) ezután minden mátrixváltozatról egy azonos szerkezetű rész következik:
 - bda) a változat sorszáma, a mátrix típusa, a keletkezett szegmensek száma,
 - bdb) egy lista, amely megadja, hogy egy adattípus melyik szegmensbe tartozik. Ha ennek a listának az i . eleme j , akkor az egyedtípus adattípusainak fenti „bb” listáján az i . helyen szereplő adattípus a j . szegmensbe került.
 - bdc) az input mátrix értékei. Ha az input mátrixot súlyozott binárisoként adtuk meg, akkor ezen a listán nem így, hanem a súlyokkal beszorzott oszlopokkal jelenik meg.
 - c) a második fázis számításainak eredménye szerepel ebben a részben:
 - ca) a keletkezett szegmensek száma (minden egyedtípusnak a kiválasztott változatát figyelembe véve), a funkciók száma,
 - cb) az átrendezett bináris szegmens-funkció-mátrix sorindexei az átrendezés utáni sorrendnek megfelelően,
 - cc) ugyanaz, mint az előző, de az oszlopindexekre vonatkozóan,
 - cd) az átrendezett mátrix. Ha a mátrixban annyi oszlop van, hogy egy sorba nem nyomtatható ki a mátrix egy teljes sora, akkor a lista oszlop csoportonként szétvágvá tartalmazza a mátrixot. A csoportokban lévő oszlopok számát a papír és a betűk szélessége együtt határozza meg, ezeket a lista készítése előtt kell megadni.
- A .LS1 lista szerkezete a .LS0-val azonos, de nem tartalmazza a „bdc” jelű részt (az input mátrixok értékeit).

A .LS2 annyiban különbözik .LS0-tól, hogy a „bd” rész minden egyed-típus esetében csak a 2. fázisnál figyelembe vett változatra készül el.

A .LS3 hasonló a .LS2-höz, de nem tartalmazza a „bdc” részt.

A .LS4 lista a „b” jelű részt csak egyetlen egyed típusra tartalmazza, amelynek sorszámát a listakészítés előtt a billentyűzetről kell megadni.

A .LS5 lista a .LS4-hez hasonló, de nem tartalmazza a „bdc” részt.

A .LS6-os lista a .LS4-től abban tér el, hogy a megadott számú egyed-típusnak csak a 2. fázisba bevont változatáról tartalmaz adatokat.

A .LS7 szűkebb a .LS6-nál, mert nem tartalmaz „bdc” részt.

A .LS8 lista csak az „a” és „c” részt tartalmazza.

1. melléklet

Adatbáziskezelő-rendszer kiválasztása

Attól függetlenül, hogy ezúttal választhatunk-e ABKR-t vagy a kiválasztást már nem vagy alig befolyásolhatjuk, szükséges az ABKR működésének hatékonyságát megvizsgálni. A funkcióelemzés során készült anyag átvizsgálásával megállapítható, hogy maradt-e olyan egyedtípus az eddig már szűkített adatmodellben, amelyet a többitől elszigetelten használnak, vagyis ahol navigáció gyakorlatilag nincs. Ha ilyen van, akkor célszerű ezt is leszabni az ABKR által kezelni kívánt szűkített adatmodellből. Ezután a használati igényekből kiindulva megvizsgálandó, hogy valóban indokolt-e ABKR használata, s ha igen, akkor milyen típusú ABKR választása volna előnyös. Át kell tekinteni azokat a kapcsolattípusokat, amelyeket döntően vagy esetleg kizárólag a kapcsolat irányával ellenkező irányban (tehát alárendelttől a fölérendelt egyedtípus felé) bejárva használnak. Az ilyen kapcsolattípust nem célszerű adategyüttesé leképezve megvalósítani. Ehelyett az alárendelt egyedtípusban kell meghagyni a fölérendelt egyedtípus azonosítóját, de adategyüttesként a kapcsolattípust nem kell deklarálni. Ennek az az oka, hogy ahol lehet, ott célszerű csökkenteni a megvalósítandó adategyüttesek számát, mivel ha azok száma nagy, a működés hatékonyságát lerontja. Ez különösen akkor indokolt, ha az alárendelt egyedtípus sok adategyüttesben lenne tag. Ahhoz ugyanis, hogy az adategyütteseknek megfelelő táblázatok új tagelőfordulás hozzáadása vagy törlése esetén az ABKR karbantartsa, viszonylag sok időre van szükség.

Annak, hogy egy egyedtípus sok adategyüttesben jöhetne tagként szóba, különböző okai lehetnek. Az egyik gyakori ok az, hogy a felhasználó még az előzetes adatmodell helyzetfelmérése során igen sokféle kapcsolatot sorol fel az egyedtípusok között. Mivel ezek egy része nem fontos kapcsolattípus (és a funkcionális elemzés helyzetfelmérése után ez többé-kevésbé ki is derül),

e kapcsolatokat kifejező séma megvalósítása eleve rendkívül gazdaságtalan lenne. A másik gyakori ok az, hogy a tervezők az esetek többségében a tárterülettel való takarékoskodás és a logikai redundancia csökkentése érdekében inkább hajlanak a komplex adatstruktúrák, adategyüttesek (multiple-set, multiple member set) megvalósítására, mint a szándékos „helypazarlásra” a gyakrabban karbantartási idők és egyszerűbb programozás érdekében.

Kétségtelen, hogy sok adategyüttes megvalósítása meggyorsítja a tárolt adatok visszakeresését. Lényegében tehát a visszakeresés és a karbantartás szempontjai egymás ellen hatnak: ha gyors visszakeresést tűzünk ki célul, úgy viszonylag lassú karbantartásra kell számítanunk és fordítva. Ennek megfelelően akkor járunk el logikusan, ha már jó előre mérlegeljük, hogy az adott esetben milyen a felhasználói igény: a minél gyorsabb visszakeresésre vagy pedig a karbantartásra, azaz a minél frissebb adatokra teszi-e a hangsúlyt? Így eldönthetjük, hogy megengedjük-e a sok adategyüttest vagy inkább az adategyüttesek számának csökkentésére törekszünk. Igaz ugyan, hogy elég tipikusnak tekinthető az olyan felhasználói igény, ahol a hangsúly a felhasználó gyors információellátásán van, azonban túlságosan rossz hatékonyságú karbantartás sem engedhető meg. Erre az esetre „gyakorlati ökol szabályként” ajánlhatók a következők:

- az adatmodell egészére nézve a kapcsolattípusok darabszáma lehetőleg ne haladja meg az adattípusok darabszámának kétszeresét;
- ha a kapcsolattípusok darabszáma azonos vagy kevesebb, mint az egyed-típusok darabszáma, akkor célszerű alaposan felülvizsgálni, hogy való-ban indokolt-e ABKR használata (lehetséges, hogy ez igen gazdaságtalan lenne);
- egy rekordtípus legfejebb 3-4 adategyüttesben legyen tag. Ha ennél több adategyüttesbeli tagság feltétlenül szükségesnek látszik, akkor megfontolandó, hogy helyes-e CODASYL típusú ABKR használata. Ilyenkor általában indokoltabb olyan adatbázis- vagy file-kezelő rendszer igény-bevétele, amely elsősorban invertált szerkezetekkel dolgozik. Egy rekord ugyanis általában azért szokott sok adategyüttesben tag lenni, mert sok-féle szempont (többnyire másodlagos kulcs) szerint kell visszakeresni. Erre az esetre pedig az invertált szerkezetek a legmegfelelőbbek.

2. melléklet

Hierarchikus klaszteráló módszerek jellemzése

A hierarchikus klaszteráló módszer célja az, hogy klaszterekbe sorolt adattípusok olyan reprezentációját adja, amelynél bármely két klaszter vagy diszjunkt vagy egyik a másikat tartalmazza. A legalsó szinten K osztály található, mindegyik csak egyetlen elemet tartalmaz. A legfelső szinten egyetlen osztály van, de ez már valamennyi elemet magába foglalja. Ha feltételezzük, hogy az egyes adatelemek egymástól függetlenek (azaz nincs közöttük kapcsolat), akkor ez éppen $K - 1$ egymást követő particionálás alkotta sorozat, ahol az előző particionálás eredményéből, azaz „szintből” úgy kapjuk a következőt, hogy — agglomerációs technika esetén — az előző szint két osztályát egyesítjük vagy — divizív technika esetén — az egyik osztályt kétfelé vágjuk. A fenti lényeges tennivalót a még technikailag szükséges kiegészítő lépésekkel együtt meglehetősen sokféle, egymástól eltérő algoritmussal lehet végrehajtani.

A klaszterált egyedeket gyakran fastruktúra leveleiként ábrázolják, a közöttük fennálló távolságot pedig helyesen egy ultrametrika fejezi ki; így e fát ultrametrikus fának is nevezik. Az ultrametrika feltétele tulajdonképpen azt fejezi ki, hogy két, egymást át nem fedő klasztert tekintve, az egyazon klaszterbe eső egyedek közötti távolságok mind kisebbek, mint a két különböző klaszterben elhelyezkedő egyedek közötti távolságok, és hogy az utóbbi, a két klaszter közötti távolságok értékei, egymással egyenlők. Fennáll az, hogy ha adott a távolságértékek egy halmaza, és ezek kielégítik az ultrametrika feltételét, akkor abból egy fastruktúra egyértelműen megszerkeszthető. Ha pedig egy fastruktúra adott, akkor abból az ultrametrikák egy végtelen családja konstruálható, de ha a gráf valamennyi szögpontjának magasságértéke ismert, akkor már az aktuális ultrametrika egyértelműen meghatározott.

Egy halmaz elemei között értelmezett ekvivalencia (tehát egy reflexív, szimmetrikus és tranzitív) reláció egyértelműen meghatározza az adathalmaz taxonomikus felosztását. Más szóval, az ekvivalenciareláció gráfjának az a sajátossága, hogy összefüggő részgráfokra, egymástól jól elváló komponensekre osztható. Tekintsük a C_j halmazt, amely itt az E_j egyedtípushoz tartozó adattípusok halmaza

$$C_j = \{A_1, A_2, \dots, A_M\},$$

majd a

$$C_j \times C_j = \{(A_1, A_1), (A_1, A_2), \dots, (A_M, A_M)\}$$

szorzathalmazt. E szorzathalmazt a célszerűen megválasztott taxonomikus mérték segítségével a valós számok egy részhalmazára egy alkalmas klaszteráló algoritmus képezi le. A taxonomikus mérték a $C_j \times C_j$ szorzathalmaz minden egyes elempárja esetén a két elem közötti egyneműséget (homogenitást) méri. Az algoritmus e mértéket figyelembe véve tudja biztosítani, hogy a kapott szegmenstípusok (klaszterek) valóban homogén elemekből állnak.

A $C_j \times C_j$ szorzathalmaz elemeire páronként vonatkozó taxonomikus mértéket egy D mátrixba rendezhetjük (2. táblázat), amely egy összetartozási relációt (távolságot vagy hasonlóságot) fejez ki a megfelelő C_j halmazban lévő adattípusok között. Ha fennáll $d_{ij} = d_{ji}$, ahol $d_{ij} \in D$, akkor ez a reláció reflexív és szimmetrikus is. Lehetséges azonban, hogy a tranzitivitás hiányzik, így az összetartozási reláció még esetleg nem ekvivalencia. Az ekvivalenciareláció fennállásának kimutatásában segíthet az, hogy e relációhoz rendelt mátrixnak az a sajátossága, hogy a sorok és oszlopok permutálása révén blokk-diagonális alakúra hozható úgy, hogy a főátlóra illeszkedő blokkon kívüli mátrixelemek értéke nulla. Ugyancsak segíthet a felmérésben az, hogy egy reláció akkor és csak akkor tranzitív, ha önmagával vett kompozíciója nem bővíti a reláció terjedelmét. Ezen utóbbi összefüggésre épül az ún. tranzitív lezárás módszere, amely azonban meglehetősen számításigényes, lassú eljárás. Kimondható, hogy az egyszerű lánc-módszer a tranzitív lezárás módszerének természetes általánosítása. Segíthet az ekvivalenciareláció fennállásának felismerésében az is, hogy a relációhoz rendelt gráf egymástól elváló részgráfokra bontható. Mindaz az információ, amely egy sokdimenziós térben elhelyezhető szögpontok csoportosításához az egyszerű

lánc-módszer végrehajtásakor szükséges, e szögpontokat kifeszítő minimális fa (a továbbiakban röviden: MFF) előállításával nyerhető.

Adott egy összefüggő, irányítatlan gráf, P . Tekintsük külön szögpontjainak $V \neq \emptyset$ és éleinek halmazát $E \neq \emptyset$, ahol $E \subseteq V \times V$.

Definíció

Feszítőfának nevezik E azon részhalmazát, amelyre fennáll, hogy V bármely két eleme között csupán egyetlen út van.

A feszítőfa alkotta részgráfban tehát nincsen hurok.

Minimális feszítőfa (MFF) akkor képezhető, ha feltesszük, hogy E minden eleméhez egy súlyt (távolság vagy költség stb.) lehet rendelni.

Definíció

P -nek egy minimális feszítőfája olyan feszítőfa, amelyre az élek súlyait összegezve, a kapott összeg minimális.

Az MFF előállításának idő- és tárigény szempontjából jelentősen eltérő módszerei vannak. A MFF előállítása során eddig a leghatékonyabbnak az ún. Prim-algoritmus bizonyult.

A Prim-algoritmus a következő:

Az M számú adattípus X_M alkotta

$$P = \{X_1, X_2, \dots, X_m\}$$

halmaz két halmazra (A és B) bontható. A -ba tartoznak az MFF-be már bevont, B -e pedig az MFF-be még be nem vont adattípusok.

1. lépés: Kezdetben $A = \emptyset$ és $B = P$.
2. lépés: Rendeljük A -hoz bármely $X \in B$ -t.
3. lépés: Keressük meg A -ban és B -ben azt az adattípuspárt, amelynek távolsága minimális (legközelebbi szomszédok).
4. lépés: A B -ben így kiválasztott adattípust a B -ből elvéve A -hoz rendeljük.
5. lépés: Ha $B = \emptyset$, az algoritmust befejezzük, ha viszont a $B \neq \emptyset$ akkor a 3. lépéstől újra lefolytatjuk.

A Prim-algoritmus más módszerekhez képest azért előnyösebb, mert az eljárás során a szögpontok közötti távolságokra csupán egyszer van szükség, és agglomeratív módszerrel könnyen lehet hierarchikus klasztereket is előállítani. Hátránya viszont, hogy a 3. lépés végrehajtása sok összehasonlítást

igényel, és vagy arra van szükség, hogy igen gyorsan elérhető tárban tart-sunk $M(M - 1)/2$ előre kiszámított távolságértéket (amire gyakran nincs elegendő tárterület), vagy ezeket ismételten elő kell állítani, ami viszont az algoritmust lelassítja. A Prim-algoritmus hatékony számítógépes megvalósítását Dijkstra és Yao adták meg. Van azonban a Prim-algoritmusnál tár-és időigény tekintetében egyaránt hatékonyabb eljárás, az ún. E_* algoritmus. Ebben eleve kevesebb összehasonlításra van szükség, mert a 3. lépésben a legközelebbi szomszédok megtalálásához elegendő csupán a legutoljára A-hoz csatolt egyetlen adattípus és a B-ben lévő többi adattípus közötti távolságok kiszámítása.

Az MFF előállításának ismertetéséhez szükségünk van a következő fogalmakra:

- különálló szögpont: a gráf azon pontja, amely nincs más ponttal összekötve;
 - fragment : részfa a gráf egészén belül;
 - szögpont és részfa távolsága : páronként a szögpont és a részfát alkotó szögpontok közötti távolságok közül a minimális távolság;
 - szögpont legközelebbi szomszédja: az a másik szögpont, amelynek a tekintett első szögponttól való távolsága nem nagyobb, mint egyéb szög-ponttól való távolságai;
 - részfa legközelebbi szomszédja: az a szögpont, amelynek a tekintett rész-fától való távolsága nem nagyobb, mint más részfáktól való távolságai;
- Ezek segítségével megfogalmazhatjuk az MFF előállításának két lépését:
- Bármely különálló szögpontot legközelebbi szomszédjához kötjük;
 - Bármely részfát legközelebbi szomszédjához kötjük.

Prim kimutatta, hogy M szögpont között $M - 1$ összekötést elvégezve megkapjuk az MFF-t.

A fenti két lépést kombinálva, különféle algoritmusok kreálhatók, amelyek megvalósításukat tekintve alapvetően két csoportba oszthatók. Ugyanis a részfa legközelebbi szomszédjának megkereséséhez tárolni kell vagy

- a) a részfa minden egyes szögpontjához a hozzá legközelebbi különálló szögpont azonosítóját és kettejük távolságát vagy
- b) minden egyes különálló szögponthoz a részfában lévő legközelebbi szomszédjának azonosítóját és távolságát.

Az így tárolt kapcsolatok az MFF létrehozásához szóba jövő gráfeket adják meg. Ha egy részfához új szögpontot adnak (vagy régit elvesznek), a

kapcsolatokat is karban kell tartani. Ezt figyelembe véve kimutatható, hogy általában hatékonyabb, ha a b) változatot alkalmazzuk.

Vannak azonban esetek, ahol az a) változat jobb. Ilyen eset az, ha a részfa legközelebbi szomszédjának keresésekor — egy előfeldolgozás eredményeképp — a keresést már nem kell az egész ponthalmazra kiterjeszteni, hanem csak egy kisebb részhalmazon belül folytatjuk le azt. E kisebb részhalmazon belül is magának a legközelebbi szomszéd keresésnek az időigényét csökkenti, ha részfaként olyan, ún. prioritási sort alkotunk, amelyben a sor elején találjuk a részfa legközelebbi szomszédját, a részfában lévő párjával együtt.

Ez az algoritmus a Prim-algoritmusnak egy olyan változata, amely bármely — még a háromszög-egyenlőtlenségnek eleget nem tevő — taxonomikus távolságmértékkel alkalmazható, és kihasználja a koordinátatér geometriájának sajátosságait. Hatékonysága közel $O(M \log M)$, így lényegesen gyorsabb, mint az addigi $O(M^2)$ körüli eljárások, viszont a tárigény nagyjából 10-20 %-kal (ez a dimenziók számától függ) meghaladja azokat.

Ennek az algoritmusnak a hátránya: előfordulhat, hogy többször is kell legközelebbi szomszédot keresni olyan szögponthoz, amelyhez ezt egyszer már megtettük, s nyilvánvaló, hogy szükségtelenül állítunk elő egyszer már kiszámított (de meg nem őrzött) távolságértékeket. Ezért a tárolóterület rovására az eljárás feltétlenül tovább gyorsítható akkor, ha az egyszer már kiszámított távolságokat megőrizzük.

Erre természetesen nincs mindig lehetőség a gépi korlátok miatt,

A probléma megoldása az lehet, hogy „jó” kezdőpontból indítják el az MFF felépítését. Jó kezdőpont ott van, ahol a szögpontok egymástól viszonylag legtávolabb állnak. Ennek kijelöléséhez viszont szükség van a teljes ponthalmazra vonatkozó előzetes, durva elemzésre, amellyel fel kell deríteni a sűrűn egymás mellett álló homogén pontcsoportokat, azaz homogén klasztereket.

Az előállított MFF-ből elhagyva azokat az éleket, amelyek súlya (hossza) egy paramétertér (hasonlósági küszöbérték) felett van, megkapjuk a gráf komponenseit. A paraméter értékének monoton lépcsőzetes változtatásával végül egyszerű láncsal összekötött hierarchikus elrendezésű klasztereket kapunk. Meg kell jegyeznünk, hogy MFF-nek egyszerű láncsal összekötött klaszterek hierarchiájába történő transzformálásakor információvesztés is fellép. E transzformációra mégis abból a praktikus okból van szükség, mert

az egyszerű lánc hierarchia adatstruktúrájának karbantartása lényegesen egyszerűbb, jobb hatásfokkal oldható meg mint az MFF-é.

Prim tétele kimondja, hogy a hasonlósági küszöb egy diszkrét értékének beállításakor az MFF ezen értékkel egyenlő vagy hosszabb éleit elvágva kapott részgráfok alkotják az eredeti ponthalmaz alkotta gráfból képezhető klasztereket és ezeken kívül az eredeti gráfból más klaszterek már nem képezhetők.

Táblázatok, ábrák

	F_1	F_3	F_4	F_5	F_7	F_8	F_6	F_{12}	F_9	F_{10}	F_{11}	F_2
Sz_1	1	1	1	0	0	0	0	0	0	0	0	0
Sz_9	1	1	1	0	0	0	0	0	0	0	0	0
Sz_4	0	0	1	1	1	1	0	0	0	0	0	0
Sz_5	0	0	1	1	1	1	0	0	0	0	0	0
Sz_7	0	0	1	1	1	1	0	0	0	0	0	0
Sz_6	0	0	0	0	0	0	1	1	0	0	0	0
Sz_2	0	0	0	0	0	0	1	1	0	0	0	0
Sz_3	0	0	0	0	0	0	0	1	1	1	1	1
Sz_8	0	0	0	0	0	0	0	0	1	1	1	1

1. táblázat $\overline{B}^*$ mátrixxá átrendezett táblázat

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
A_1		0.513	0.221	0.221	0.914	0.971	0.193	0.474
A_2	0.513		0.107	0	0.391	0.495	0.111	0.237
A_3	0.221	0.107		0.981	0.391	0.333	0.666	0.513
A_4	0.221	0	0.981		0.333	0.221	0.666	0.510
A_5	0.914	0.391	0.391	0.333		0.970	0.301	0.555
A_6	0.971	0.495	0.333	0.221	0.970		0.221	0.474
A_7	0.193	0.111	0.666	0.666	0.301	0.221		0.693
A_8	0.474	0.237	0.513	0.510	0.555	0.474	0.693	

2. táblázat $\overline{D}$ mátrix, felvett adatokkal

	A ₁	A ₂	A ₃	A ₄	A ₅	A ₆	A ₇	A ₈	$\sum_{j=1}^M d_{ij}$
A ₁		<u>0.513</u>	0.221	0.221	0.914	<u>0.971</u>	0.193	0.474	3.507
A ₂	0.513		0.107	0	0.391	0.495	0.111	0.237	1.854
A ₃	0.221	0.107		<u>0.981</u>	0.391	0.333	0.666	0.513	3.212
A ₄	0.221	0	<u>0.981</u>		0.333	0.221	0.666	0.510	2.932
A ₅	0.914	0.391	0.391	0.333		0.970	0.301	0.555	3.855
A ₆	<u>0.971</u>	0.495	0.333	0.221	<u>0.970</u>		0.221	0.474	3.685
A ₇	0.193	0.111	0.666	0.666	0.301	0.221		<u>0.693</u>	2.851
A ₈	0.474	0.237	0.513	0.510	0.555	0.474	<u>0.693</u>		3.456

3. táblázat Sorösszegek és maximális oszlopelemek

	A ₁	A ₂	A ₃	A ₄	A ₅	A ₆	A ₇	A ₈	d_{ij}
A ₁		<u>0.513</u>	0.221	0.221	0.914	<u>0.971</u>	0.193	0.474	3.507
A ₂	0.513		0.107	0	0.391	0.495	0.111	0.237	1.854
A ₃	0.221	0.107		<u>0.981</u>	0.391	0.333	0.666	0.513	3.212 ←
A ₄	0.221	0	<u>0.981</u>		0.333	0.221	0.666	0.510	2.932
A ₅	0.914	0.391	0.391	0.333		0.970	0.301	0.555	3.855
A ₆	<u>0.971</u>	0.495	0.333	0.221	<u>0.970</u>		0.221	0.474	3.685
A ₇	0.193	0.111	0.666	0.666	0.301	0.221		<u>0.693</u>	2.851
A ₈	0.474	0.237	0.513	0.510	0.555	0.474	<u>0.693</u>		3.456

4. táblázat A $\overline{\overline{D}}$ mátrix abszolút maximális eleme

	A ₁	A ₂	A ₃	A ₄	A ₅	A ₆	A ₇	A ₈
A ₁	(2) lépés	(8) lépés 0.513			(4) lépés 0.914	(6) lépés 0.971	0.193	0.474
A ₂	0.513				(7) lépés 0.391	0.495	0.111	0.237
A ₃								
A ₄								
A ₅	0.914	0.391				0.970	0.301	0.555
A ₆	(0.971)	0.495			0.970		0.221	0.474
A ₇	0.193	0.111			0.301	0.221		0.693
A ₈	0.474	0.237			0.555	0.474	0.693	

5. táblázat Szegmenstípusok képzése

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	$\overline{P}$
A_1	1	0	0	0	1	1	1	1	1	0	1	1	1	0	0	9
A_2	0	1	1	0	0	1	0	0	0	1	1	1	0	1	0	7
A_3	1	1	0	1	1	0	1	1	0	1	1	0	0	0	1	9
A_4	1	1	0	1	0	1	0	1	0	1	0	1	1	1	1	10
A_5	0	1	1	1	1	0	0	1	1	0	1	1	0	1	1	10
A_6	0	0	0	0	1	1	0	0	1	1	0	0	0	1	1	6
A_7	1	0	1	1	0	1	0	1	0	0	0	0	1	0	0	6
A_8	1	0	0	0	1	0	0	0	1	1	1	1	0	0	1	7

6. táblázat $A \overline{\overline{C}}$ mátrix sorösszegei

	F_1	F_5	F_6	F_7	F_8	F_9	F_{11}	F_{12}	F_{13}
A_1	1	1	1	1	1	1	1	1	1
A_2	0	0	1	0	0	0	1	1	0
A_3	1	1	0	1	1	0	1	0	0
A_4	1	0	1	0	1	0	0	1	1
A_5	0	1	0	0	1	1	1	1	0
A_6	0	1	1	0	0	1	0	0	0
A_7	1	0	1	0	1	0	0	0	1
A_8	1	1	0	0	0	1	1	1	0

7. táblázat $A \overline{\overline{J}}_l$ részmátrix ($l = 1, m = 1, 2, \dots, 8, k = 1, 2, \dots, 13$)

	F_1	F_5	F_6	F_7	F_8	F_9	F_{11}	F_{12}	F_{13}	$\overline{P11}$		
A_1	1	1	1	1	1	1	1	1	1	9		
A_2	0	0	1	0	0	0	1	1	0	3	P11	A_1-A_2
A_3	1	1	0	1	1	0	1	0	0	5	P11	A_1-A_3
A_4	1	0	1	0	1	0	0	1	1	5	P11	A_1-A_4
A_5	0	1	0	0	1	1	1	1	0	5	P11	A_1-A_5
A_6	0	1	1	0	0	1	0	0	0	3	P11	A_1-A_6
A_7	1	0	1	0	1	0	0	0	1	4	P11	A_1-A_7
A_8	1	1	0	0	0	1	1	1	0	5	P11	A_1-A_8

8. táblázat $A \overline{\overline{J}}_l$ részmátrix sorösszegei
($l = 1, m = 1, 2, \dots, 8, k = 1, 2, \dots, 13$)

Egy szegmenstípushoz
tartozó szegmenstípusok

Sz₁

Sz₂

Sz₂

Kiinduló
mátrix

$\overline{\overline{D}}$

$\overline{\overline{D}}_{125678}^{125678}$

$\overline{\overline{D}}_{78}^{78}$

Adattípusok
csoportja

A₃-A₄

A₁-A₆-A₅-A₂

A₇-A₈

9. táblázat Szegmensképzés végeredménye

	F ₁	F ₃	F ₄	F ₅	F ₇	F ₈	F ₆	F ₁₂	F ₉	F ₁₀	F ₁₁	F ₂
Sz ₁	-	-	Sz ₄									
	1	1	2									
Sz ₉	Sz ₁	Sz ₁	Sz ₁									
	2	2	3									
Sz ₄				-	-	-	-					
				1	1	1	1					
Sz ₅			Sz ₄	Sz ₄	Sz ₄	Sz ₄	Sz ₄					
			2	2	2	2	2					
Sz ₇			Sz ₄	Sz ₄	Sz ₄	Sz ₄	Sz ₄					
			2	2	2	2	2					
Sz ₆							Sz ₂	Sz ₂				
							1	1				
Sz ₂							-	-				
							1	1				
Sz ₃								Sz ₆	-	-	-	-
								1	1	1	1	1
Sz ₈									Sz ₃	Sz ₃	Sz ₃	Sz ₃
									1	1	1	1

¹ Egy mátrixelem felső részében a meghívó szegmenstípus, alatta a hierarchiaszint jele áll.

10. táblázat $\overline{\overline{F}}$ mátrixá átjelölt táblátat¹

1. A nyilvántartás megnevezése										2. Kódja									
3. Nyilvántartás vezetéséért felelős tanácsai szerv KSH területi kódja és szervezeti egysége																			
4. Az adatok biztosításában közreműködő szevek:																			
5. A nyilvántartásból tájékoztatást, jelentést, adatszolgáltatást kapó szervek																			
6. Elrendelő jogszabály										Jogsz.nincs									
7. Nyomtatványminta raktári, ill. szabvány száma										Nincs nyomtatv. Nincs mellékelve									
8. Vezetésére előírás utasítás száma										Nincs előírás Nincs csatolva									
9. Bejegyzések gyakorisága										óra nap hét dek. h. th. n.év f.év év 2 év 5 év t.év									
10. A nyilvántart.-ból szóbeli írásbeli jel. adatszolg. gyak.																			
11. Tanácsai tevékenységet akadályozná a nyilvánt. megsz.										Igen									
12. A nyilvántartás példányszáma:										Nem									
13. Készül-e belőle géppel olvasható adathordozóra is másolat, kivonat?										Igen Nem									
14. Történik-e belőle tanácsai célú helyi, gépi feldolgozás?										Igen Nem									

Válaszadó neve:										Aláírás:									
Beosztása :																			
Szervező neve:										Aláírás:									
Beosztása:										Dátum:									

11. táblázat Nyilvántartás-lap

1. Az adatszolgáltatás címe (tartalma)		2. Kódja	
3. Az adatszolgáltatásra kötelezett tanácsai szerv KSH területi kódja és szervezeti egysége			
4. Az adatok biztosításában közvetlenül közreműködő szervek			
5. Az adatszolgáltatást megkapják az alábbi szervek: Az alábbi határidőre:			
6. Elrendelő jogszabály		Jogsz.nincs Nincs eng. sz.	
7. Nyomtatványminta raktári, ill. szabvány száma		Nincs nyomtatv. Nincs mellékelve	
8. Kitöltésére előírás utasítás száma:		Nincs előírás Nincs csatolva	
9. Elkészítésének gyakorisága	óra nap	hét dek.	h. t.h.
			n.év f.év
			év 2 év
			5 év t.év
			e.eset
10. Az adatszolgáltatás elkészítendő példányszáma:			
11. Készül-e belőle géppel olvasható adathordozóra is másolat, kivonat?		Igen Nem	Anyaga papír Mágneses
12. Történik-e belőle tanácsai célú helyi, gépi feldolgozás?		Igen Nem	
Válaszadó neve: Beosztása :		Aláírás:	
Szervező neve: Beosztása:		Aláírás: Dátum:	

12. táblázat Adatszolgáltatás-lap

1. A tervdokumentáció megnevezése				2. Kódja				
3. A tervdokumentáció elkészítéséért felelős tanácsi szerv (KSH ter; kód) és szervezeti egysége								
4. Az adatok biztosításában közreműködő szervek:								
5. A tervdokumentációból tájékoztatást, jelentést adatszolgáltatást kapó szervek:								
6. Elrendelő jogszabály utasítás száma:				Nincs j.sz.				
7. Nyomtatvány raktári v. szabvány száma:				Nincs nyomtatv. Nincs mellékelve				
8. Kitöltéshez előírás módszertani útmutató				Nincs előírás Nincs mellékelve				
9. Elkészítésének gyakorisága:	óra	nap	hét	hó	ne.év	félév	év	több év
10. Szóbeli tájékoztatás gyakorisága								
írásbeli jelentés gyakorisága								
11. Készül-e belőle géppel olvasható adathordozóra is másolat, kivonat?				Igen		Anyaga papír		
				Nem		Mágneses		
12. Történik-e belőle tanácsi célú helyi, gépi feldolgozás?				Igen				
				Nem				
13. A tervdokumentáció hány példányban készül:								

Válaszadó neve:	Aláírás:
Beosztása :	
Szervező neve:	Aláírás:
Beosztása:	Dátum:

13. táblázat Tervdokumentáció-lap

1. Szakfeladat megnevezése					2. Kódja	
3. A szakfeladat elvégzéséért felelős tanácsai szerv KSH területi kódja és szervezeti egysége						
A szakfeladat ellátását az alábbi funkciók végrehajtása képezi						
Sor- szám	4. Funkciók megnevezése	5. Ügy- kör	6. Mun- kakör	7. P.	8. Bels.	9. Küls.
01.						
02.						
03.						
04.						
05.						
06.						
07.						
08.						
09.						
10.						
11.						
12.						
13.						
14.						
15.						
16.						
17.						
18.						
19.						
20.						
Szervező neve:		Aláírás:				
Beosztása:		Dátum:				

14. táblázat Szakfeladat-funkció lap

1. Szakfeladat megnevezése	2. Kódja
Közlekedés igazgatás	
3. A szakfeladat elvégzéséért felelős tanácsi szerv	
KSH területi kódja és szervezeti egysége	
08 2558 4 EKO és	
VB. Hiv. Műszaki O.	

A szakfeladat ellátását az alábbi funkciók végrehajtása képezi

Sor-szám	4. Funkciók megnevezése	5. Ügy-kör	6. Munkakör	7. P.	8. Bels.	9. Küls.
01.	rendeletalkotás (tanácsrendelet VB. határozat)	02(F)/4				
02.	nyomvonalakijelölési eljárás, közigazgatási bejárás	29(F)0				
03.	nyomvonalas jellegű területfelhasználási engedélyezés	29(F)0				
04.	tanácsi kösútépítés engedélyezés	29(F)0		1		1
05.	járda- és kocsibehajtó építés engedélyezés	26(F)8		1		1
06.	sajáthasználatú utak építés engedélyezés	81(F)0				
07.	műszaki átadás-átvétel	29(F)0		1		1
08.	használatbavételi (forgalombahelyezési) engedélyezés	21(F)0		1		1
09.	kösterületfelbontási engedélyezés	29(F)0		2		2
10.	kösút és járda igénybevételeének engedélyezése	29(F)0		1		1
11.	kösút megszüntetés	29(F)0				
12.	útcsatlakozás létesítésének, áthelyezésének, megszüntetésének engedélyezése	29(F)0				
13.	tanácsi és saájáthasználatú utak egyéb igazgatási ügyei	29(F)0				
14.	KPM utaknál a tárgyi szakhatósági feladatok ellátása	80(F)0				
15.	túlméretes és túlsúlyos járművek közlekedési engedélyezése	29(F)0				
16.	iparvágány létesítés engedélyezés	36(F)0				
17.	útátjárók létesítésének és megszüntetésének engedélyezése	29(F)0				
18.	sajáthasználatú vasutak ügyei	81(F)0				
19.						
20.						
Szervező neve: dr.Kovács Lajos		Aláírás:				
Beosztása: osztályvezető		Dátum: 1981.XII.17.				

14/1. táblázat Kitöltött szakfeladat-funkció lap

1. Felvétel helye (KSH ter.; kód): Szervezeti egység:		2. Dátuma
3. Szakfeladat:		4. Funkció- igény lap:
5. Sor- szám	6. Sorszámmal jelölt funkcióban fellépő problémák	7. Kiváltó tényezők, okok
Válaszadó neve: Beosztása:		Szervező neve: Beosztása:

15. táblázat Funkcióigény-lap

1. Funkció (folyamat) megnevezése és	2. Kódja	3. Eljáró szervek neve
4. Tevékenység sorszáma és meghatározása	5. Lapszám	
Bemenetek: Feldolgozás: Kimenetek: Záró műveletek:		
Név	Dátum	Aláírás
Kitöltötte		
Ellenőrizte		

17. táblázat Funkciótevékenység-lap

AB:		Oldal:
Elemző:		Dátum:
Jóváhagyta:		Állapot/változat:

18. táblázat Szabványos fejléc

AB:		Egyed típus – jegyzék	Oldal:		
Elemző:			Dátum:		
Jóvá hagyta:			Állapot/változat:		
Sorsz.	Egyed típus (szinonímák)	Tulajdonságtípusok	Előfordulások		
			Min.	Átl.	Max.

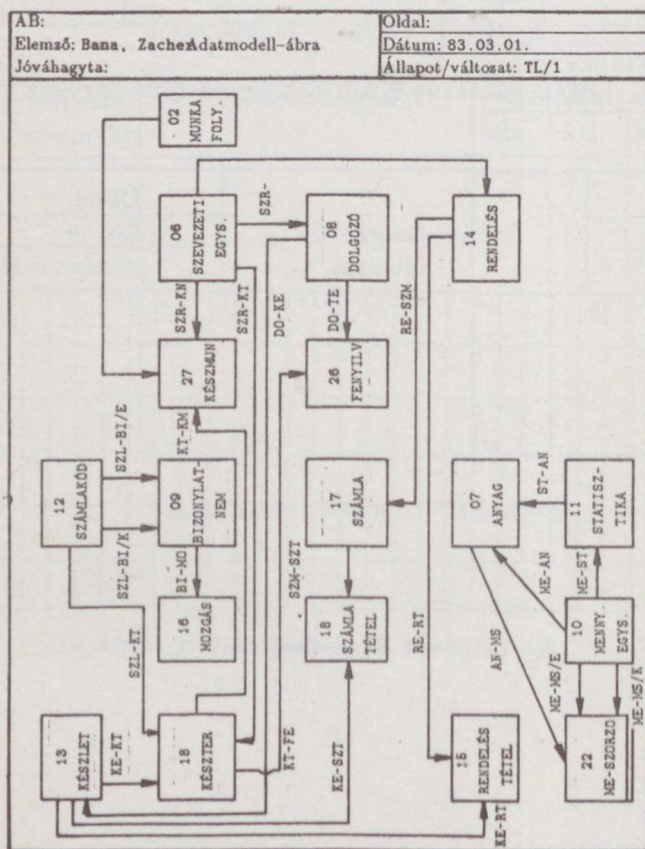
19. táblázat Egyedtypus-jegyzék

AB: AGIR-1		Egyedtypus - jegyzék	Oldal:2/2		
Elemző: ZACHER			Dátum: 83.02.05.		
Jóváhagyta:			Állapot/változat: TF/1		
Sorsz.	Egyedtypus (szinonímák)	Tulajdonságtípusok	Előfordulások		
			Min.	Átl.	Max.
25.	TERV		1000		100000
26.	FENYILV		20000		50000
27.	KESZMUN		30000		80000

[illegible]

AB:		Kapcsolattípus-jegyzék		Oldal:			
Elemző:				Dátum:			
Jóváhagyta:				Állapot/változat:			
Sorsz.	Kapcsolat-típus neve	Szinoním név	Főlérendelt egyedtípus	Alárendelt egyedtípus	Előfordulások		
					Min.	Átl.	Max.

22. táblázat Kapcsolattípus-jegyzék



21. táblázat Bachman-diagram

AB:AGIR-1		Kapcsolattípus-jegyzék		Oldal:1/2	
Elemző:ZACHER				Dátum:83.02.04.	
Jóváhagyta:				Állapot/változatTF/1:	
Sorsz.	Kapcsolat-típus neve	Szinoním név	Főlérendelt egyedttípus	Alárendelt, egyedttípus	
01.	SZE-MU	-	SZERZŐDÉS	MUNKA	
02.	MU-SZK	-	MUNKA	SZAKASZ	
03.	AU-SZE	SZERZŐDÖTT	AGIR-ÜGYFÉL	SZERZŐDÉS	
04.	AU-SZR	RÉSZLEGE	AGIR-ÜGYFÉL	SZERVEZETI EGYS.	
05.	AU-SZL	-	AGIR-ÜGYFÉL	SZÁMLAKÓD	
06.	AU-KE	-	AGIR-ÜGYFÉL	KÉSZLET	
07.	SZR-DO	DOLGOZÓJA	SZERVEZETI EGYS.	DOLGOZÓ	
08.	SZR-KT	-	SZERVEZETI EGYS.	KÉSZTER	
09.	SZR-BI	-	SZERVEZETI EGYS.	BIZONYLATNEM	
11.	AN-KE	TÁROLJA	ANYAG	KÉSZLET	
13.	DO-RE	ÜGYINTÉZŐ	DOLGOZÓ	RENDELÉS	
15.	DO-FE	KIADVA	DOLGOZÓ	FENYILV	
16.	DO-KE	FELELŐS	DOLGOZÓ	KÉSZLET	
17.	KT-FE	KIVETTE	KESZTER	FENYILV	
18.	SZL-BI/K	-	SZÁMLAKÓD	BIZONYLATNEM	
19.	MU-KM	-	MUNKA	KESZMUN	
20.	KT-KM	-	KÉSZLET	KESZMUN	
21.	KE-KT	-	KÉSZLET	KÉSZTER	
22.	RE-RT	R-TETEELE	RENDELÉS	RENDELÉSTÉTEL	
24.	SZM-SZT	SZ-TETEELE	SZÁMLA	SZÁMLATÉTEL	
25.	ST-AN	-	STATISZTIKA	ANTAG	
26.	ME-ST	-	MENNYISÉGI EGYS.	STATISZTIKA	
27.	ME-MS/E	EREDETI	MENNYISÉGI EGYS.	ME-SZORZÓ	
28.	ME-MS/K	KIVÁNT	MENNYISÉGI EGYS.	ME-SZORZÓ	
29.	FK-SZL	HASZNÁLJA	FŐKÖNYV	SZÁMLAKÓD	
30.	KE-TE	FOGLAL	KÉSZLET	TERV	
31.	AN-FO	BESZEREZHETŐ	ANYAG	FORRÁS	
32.	SZO-FO	SZÁLLÍT	SZÁLLÍTÓ	FORRÁS	
33.	SZK-TE	IGÉNYEL	SZAKASZ	TERV	

22/1 táblázat Kitöltött kapcsolattípus-jegyzék

AB:		Egyed típus				Oldal:			
Elemző:						Dátum:			
Jóváhagyta:						Állapot/változat:			
Sorszám:	Rövid név:	Név:	Szinonímák:						
Típus:	Altípus:	Forma:	Stabil:						
Meghatározás:									
Tulajdonságok:									
Az azonosító jellege									
Az egyed típus kapcsolatai/szerepe a kapcsolatban:									
Előfordulások száma									
Hozzáférés		min.:		átl.:		max.:		növekedés:	
jellege		Létrehozás		Módosítás		Törlés		Felhasználás	
joga									
Megjegyzések:									

23. táblázat Egyed típus-lap

Sorszám: 18.		Rövid név: SZT	Név: SZÁMLA TÉTEL	Szinonímák: SZ. TÉTEL
Típus: KONKRÉT		Altípus: ---	Forma: N	Stabil: IGEN
Meghatározás: Egy teljesített szolgáltatás vagy leszállított áru ellenértékének leírása				
Tulajdonságok: ld. EGYEDT<PUS LARTALOM-lap				
Az azonosító jellege: Részlegesen külső				
Az egyedítípus kapcsolatai/szerepe a kapcsolatban: SZM-SZT; KE-SZT				
Előfordulások száma		min.: 8000	átl.:	max.: 11000
Hozzáférés	jellege	Létrehozás	Módosítás	Törlés
	joga			
Megjegyzések:				

AB: AGIR-1	Egyedítípus	Oldal: 1
Elemző: BANA		Dátum: 83.02.05.
Jóváhagyta:		Állapot/változat: TF/1

23/1. táblázat Kitöltött egyedítípus-lap

Sorszám:		Rövid név:	Név:	Színólmák:	
Meghatározás:					
Jellemzett egyed típusok / szerep:					
Szármozás:					
Értékek:		Csoport:			
Hozzáférés	jellege	Létrehozás	Módosítás	Törlés	Felhasználás
joga					
Megjegyzések:					

AB:	Tulajdonságtípus	Oldal:
Elemző:		Dátum:
Jóváhagyta:		Állapot/változat:

24. táblázat Tulajdonságtípus-lap

Sorszám: 004		Rövid név: VALAR		Név: VÁLLALÁSI ÁR		Színvonal: ---	
Meghatározás: A szerződésben vállaltak szerződésben foglalt ellenértéke							
AB: AGIR-1		Tulajdonságtípus				Oldal: 1	
Elemző: ZACHER						Dátum: 83.02.05.	
Jóváhagyta:						Állapot/változat: TF/1	
Jellemzett egyed típusok / szerep: SZERZŐDÉS / LE							
Származás:							
Értékek:				Csoport:			
Hozzáférés		jellege		Létrehozás		Módosítás	
		joga				Törlés	
Megjegyzések: F (11/2), EZER ft-ban						Felhasználás	

24/1. táblázat Kitöltött tulajdonságtípus-lap

Sorszám:		Rövid név:		Név:		Színónímák:	
Meghatározás:							
Jelleg:		Viszony:					
Kapcsolt egyedtípusok		Főlérendelt		Alárendelt		Másodlagos ismérv	
Transzitivitás:							
Kapcsolt előfordulások száma				min.: átl.: max.: növekedés:			
Hozzáférés		jellege		Létrehozás		Módosítás	
		joga				Törlés	
Megjegyzések:							

AB:	Kapcsolattípus	Oldal:
Elemző:		Dátum:
Jóváhagyta:		Állapot/változat:

25. táblázat Kapcsolattípus-lap

Sorszám: 45		Rövid név: KE-SZT		Név: KÉSZLET-SZ. TÉTEL		Színvonal	
Meghatározás: Az azonos anyagra vonatkozó számlatételeket kapcsolja a készlethez							
Jelleg: O/K		Viszony: Bennfoglaló SZM-SZT					
Kapcsolt egyed típusok		Fölérendelt		Alárendelt		Másodlagos ismerv	
		KÉSZLET		SZÁMLATÉTEL		---	
Tranzitivitás:		---					
Kapcsolt előfordulások száma		min.:		átl.:		max.: növekedés:	
Hozzáférés		jellege		Létrehozás		Módosítás	
		joga				Törlés	
Felhasználás							
Megjegyzések:							

AB: AGIR-1	Kapcsolattípus	Oldal: 1
Elemző: BANA		Dátum: 83.02.04
Jóváhagyta:		Állapot/változat: TF/1

25/1. táblázat Kitöltött kapcsolattípus-lap

AB:FÜTI-AGIR		Oldal:
Elemző:Bana-Mezei	Funkció	Dátum:1983. XII.31.
Jóváhagyta:	űrlap	Állapot/változat:
Azonosító:	Név: Raktári készletek és forgalomellenőrzés	
<u>Gyakoriság:</u>	havi/n.é.	
<u>Tételszám:</u>	5-20000/AGIR-ügyfél/hónap, de valamennyi a cég által használt anyagon végig kell menni	
<u>Input:</u>	-- minden mozgásbizonylat feldolgozásánál erre a célra készülő output -- adatmodell	
<u>Feldolgozás:</u>	1. A többi bizonylat feldolgozásánál készített outputok átmásolása 2. Az előző pontban nem érintett anyagok adatainak feldolgozása -- Raktáron belüli mennyiség meghatározása -- Érték meghatározása -- 6 hón túl nem nagyobb készlet meghatározása	
<u>Output:</u>		
Raktár:	TERMSZ	
Cikkszám	CISZBO	
Cikk neve	AGNV	
Termelő szervezet		
Munkaszám		
Bizonylatszám		
Tételsor	TSSZ	
Mozgásszám		
Főkönyv	FK	
Egységár	ELAR	
M.e.	MEJEL	
Mennyiség		
Új készlet		
Különbség		
Tételérték	számított	
6 órán túli	számított	
Utolsó felhasználás	VTMOZ	

26. táblázat Navigációs táblázat

AB:		Funkcióleírás				Oldal:			
Elemző:						Dátum:			
Jóváhagyta:						Állapot/változat:			
Azonosító:				Név:					
Egyed	Műveletek	Elérési alap	Elérés típusa	Kapcsolat	Műveletek	Előfordulások száma	Kiválasztási ismérv	Előfordulások száma	Tulajdonságok és Műveletek

27. táblázat Funkcióleírás-lap

AB:		Egyed típus-használat		Oldal:						
Elemző:				Dátum:						
Jóváhagyta:				Állapot/változat:						
Egyed típus sorsz.:		Rövid név:		Egyed típus neve						
Funkcióazonosító:	Funkciónév:	Gyakoriság:	Elérés alapja:	Elérés jellege:	Kézelt előf. db.	Tulajd. típus	Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
Funkcióazonosító:	Funkciónév:	Gyakoriság:	Elérés alapja:	Elérés jellege:	Kézelt előf. db.	Tulajd. típus	Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
Funkcióazonosító:	Funkciónév:	Gyakoriság:	Elérés alapja:	Elérés jellege:	Kézelt előf. db.	Tulajd. típus	Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
Funkcióazonosító:	Funkciónév:	Gyakoriság:	Elérés alapja:	Elérés jellege:	Kézelt előf. db.	Tulajd. típus	Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	
							Elér.	Azon.	Műv.	

28. táblázat Egyed típus-használat

AB:		Kapcsolattípus- használat	Oldal:	
Elemző:			Dátum:	
Jóváhagyta:			Állapot/változat:	
Kapcsolat száma	Kapcsolat rövid neve	Használat		
		Oda	Vissza	

29. táblázat Kapcsolattípus-használati lap

AB:	Kapcsolattípus- használat		Oldal:
Elemző:			Dátum:
Jóváhagyta:			Állapot/változat:
Kapcsolat száma	Kapcsolat rövid neve	Használat	
		Oda	Vissza
01	SZK-DO		
02	SZK-KT	145, 152,	153, 167,
15	DO-FE	22, 146, 150, 151/7	
16	DO-KE		33,
17	KT-FE	29, 150,	22, 146, 157/9
18	SZL-BI/K	144, 145, 146, 147, 148, 150, 156/2, 152, 157, 158, 169,	
19	MK-KI		145, 147, 152, 158,
20	KT-KM	28, 115, 158,	26, 147, 148, 157, 160,
21	KE-KT	29, 33, 144, 155,	19, 22, 26, 145, 146, 147, 148, 150, 152, 155,
22	RE-RT	144, 155,	
24	SZM-SZT		155,
25	ST-AM		
26			
27	ME-MS/E		
28	ME-MS/K		
35	KE-RT		144, 155,
36	SZL-EI/E		
37	AN-MS		
39	SZL-KI		13, 25, 144, 145, 146, 154, 155, 157, 158,
42	BI-HO	144, 145, 146, 147, 148, 150, 151, 152, 155, 158, 160,	
43	ME-AN		12, 20, 22, 26, 28, 144, 145, 147, 148, 150, 151, 155, 156, 158, 160, 155,
44	RE-SZM		
45	KE-SZT	155,	
46	SZE-RE		122,
47	SZF-KH	26, 144, 148, 157, 160,	145, 147, 148, 149, 155,

29/1. táblázat Kitöltött kapcsolattípus-használati lap

Elemzők:		Adatmodell:	Elemzés dátuma
1	Funkció:		
<u>Táblaszerkezet:</u>			

30. táblázat Táblázatszerkezet-lap

AB:	Tulajdonságtípus jegyzék	Oldal:7/3
Elemző:		Dátum:
Jóváhagyta:		Állapot/változat:
074	HC	HIÁNYCIKK JELZÉS
075	UTN	UTÁNPÓTLÁSI NAPOK SZÁMA
077	EKOT	ELSZÁMOLTATÁSRA KÖTELEZETTSÉG KÓDJA
083	FKNEV	FÜKÖNYVI KÓD NÉV
087	TERT	TELJES ANYAGÉRTÉK
088	MJ	MINŐSÍTŐ JEL
090	SZAK	SZAKASZ AZONOSÍTÓ
099	AR/1	ÁRINDEX 1.
100	AR/2	ÁRINDEX 2.
101	AR/3	ÁRINDEX 3.
102	AR/4	ÁRINDEX 4.
104	BEAR	BESZERZÉSI ÁR
105	MVALAR	MUNKAVÁLLALÁSI ÁR
106	MBEFAR	MUNKA BEFOLYT ÁRBEVÉTELE
107	MTRAF	MUNKA TELJES RÁFORDÍTÁSA
108	MKIDO	MUNKA KEZDÉSI IDŐ
109	MBEFID	MUNKA BEFEJEZÉSI IDŐ
110	VACI	VÁLLALATI KÉSZLET
111	KETER	KÉSZTERMÉK AZONOSÍTÓ
112	VANE	VÁLLALAT MEGNEVEZÉSE
113	ÜMNESZ	AGIR-ÜGYFÉL MNB SZÁMÁRA
114	ÜT	AGIR-ÜGYFÉL TIPUSA

31. táblázat Tulajdonságtípus-jegyzék

AB: AGIR-1	Egyed típus- tartalom	Oldal:1
Elemző:		Dátum:
Jóváhagyta:		Állapot/változat:
<u>24.-SZETMUN</u>		
175 - SZEMU	(AZ)	
137 - SZE	(AZ1)	
008 - MKSZ	(AZ2)	
138 - ADOLSZ	(AZ3)	
216 - SZMKID	(LE)	
217 - SZMBID	(LE)	
021 - VKAK	(LE)	
022 - VKTAK	(LE)	
023 - VIBEK	(LE)	
024 - VFUK	(LE)	
025 - VBK	(LE)	
026 - VEK	(LE)	
027 - SZAKA	(LE)	
028 - SZKFUK	(LE)	
029 - SZDIJ	(LE)	
030 - SZALVAK	(LE)	
031 - SZALDIJ	(LE)	
032 - SZEK	(LE)	
033 - TAK	(LE)	
034 - TFUK	(LE)	
035 - TBK	(LE)	
036 - TGK	(LE)	
037 - TDIJ	(LE)	
038 - TEK	(LE)	
011 - STAT	(LE)	

32. táblázat Egyed típus-tartalom

AB: AGIR-1		Egyedtartalom típus		Oldal:1	
Elemző:Bana-Zacher				Dátum: 1983.03.12.	
Jóváhagyta:				Állapot/változat:TL/1	
Sémanév:AGIR nn				Verziószám:01	
AUTHOR:					
DATE:					
INSTALLATION: FÜTI					
REMARKS:					
FILE név	ASSIGN	DEV.TYPE	FILE név	ASSIGN	DEV.TYPE
AERA név	Laptartomány -tól -ig		FILE név	Blokktartomány -tól -ig	
	Séma név		Verzió sz.	COPY	

33. táblázat IDMS séma DL

Rekordtípus			Tervező:		1
Rekordnév			Dátum:		
			Változat:		
Szint	Adatnév	Picture	Type	Előford.	
Megjegyzés					

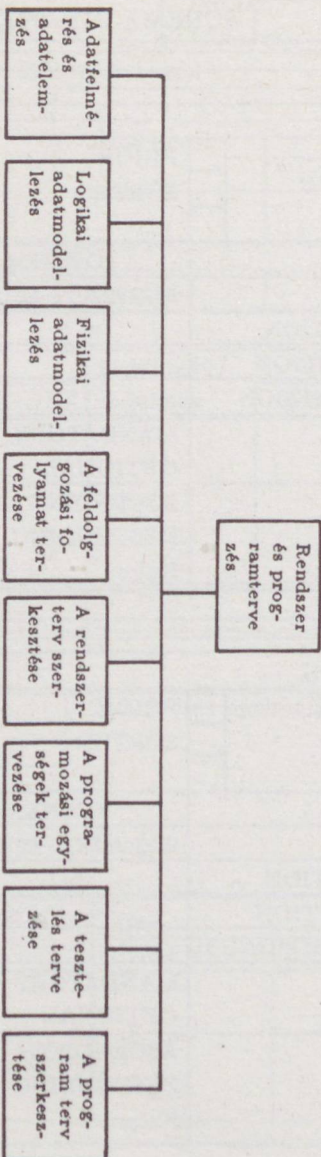
34. táblázat Rekord-lap 1

Rekordtípus				Tervező:		1	
Rekordnév				Dátum:			
				Változat:			
Elhelyezési mód							
DIRECT				CALC			
Adatnév		Jelölőmező		Adatnevek		Duplikáció	
						igen nem	
Realm(ek)							
Kereső kulcsok	Neve	típusa		eljárásnév / indexnév	duplikáció		
		CALC	INDEX		igen	nem	
DBTT - bejegyzések száma:				REALM:			
Populáció				Elhelyezés optimalás (SET-re)			
Db.		REALM					
Indexnév		Elhelyezés (REALM)		Típus		Din. ujrász.	
				DB-KEYLIST REP-KEY		(Db. PAGE)	
Kompresszió							

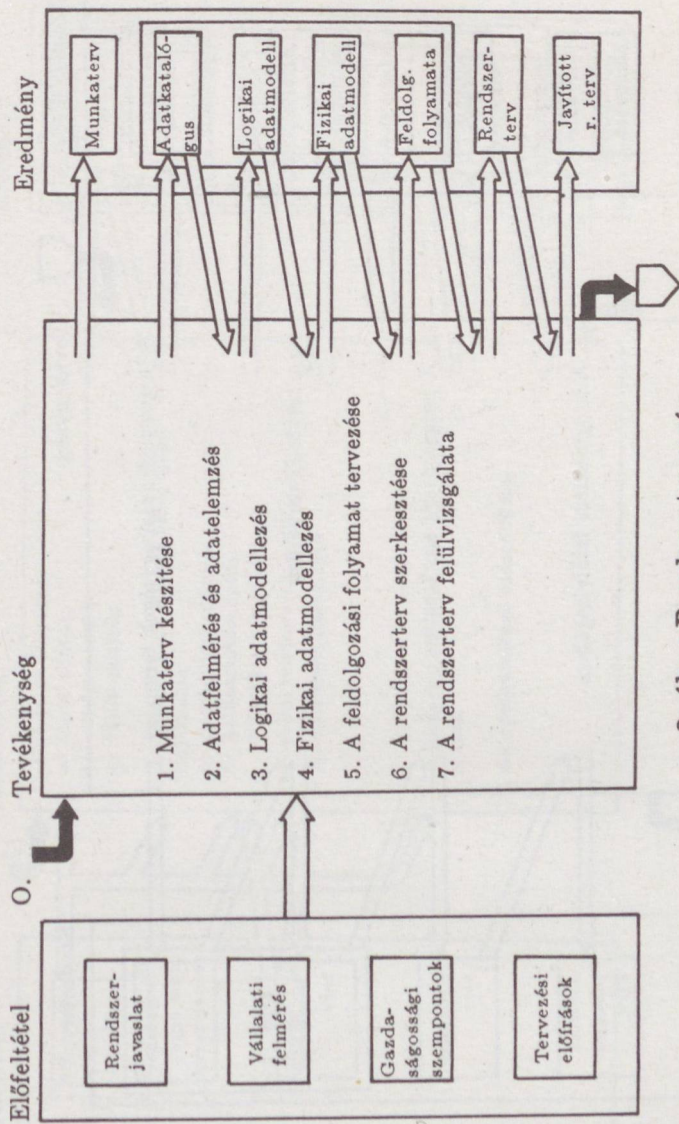
35. táblázat Rekord-lap 2

AB: AGIR-1		IDMS SCHEMA-DDL SZET		Oldal: 1	
Elemző: Bana-Zacher				Dátum: 1983.03.12.	
Jóváhagyta:				Állapot/változat: TL/1	
Szetnév: SZR-RE					
COPY:					
ORDER			MODE		
FIRST		PRIOR		CHAIN	☒
LAST		SORTED		LINKED PRIOR	
NEXT					
		OWNER		MEMBER	
Rekordnév		SZERVEZETI-EGYS.		RENDELES	
NEXT DBKEY POSITION		X		X	
PRIOR DBKEY POSITION					
OWNER DBKEY POSITION		--		X	
JELLEG		MANDATORY		AUTOMATIC	
		OPTIONAL		MANUAL	
SORTED		ASCENDING		FIRST	
		DESCENDING		LAST	
		Kulcs:		NOT	
Szetnév: SZR-KM					
COPY:					
ORDER			MODE		
FIRST		PRIOR		CHAIN	☒
LAST		SORTED		LINKED PRIOR	
NEXT					
		OWNER		MEMBER	
Rekórdnév		SZERVEZETI-EGYS.		KESZMUN	
NEXT DBKEY POSITION		X		X	
PRIOR DBKEY POSITION					
OWNER DBKEY POSITION		--		X	
JELLEG		MANDATORY		AUTOMATIC	
		OPTIONAL		MANUAL	
SORTED		ASCENDING		FIRST	
		DESCENDING		LAST	
		Kulcs:		NOT	

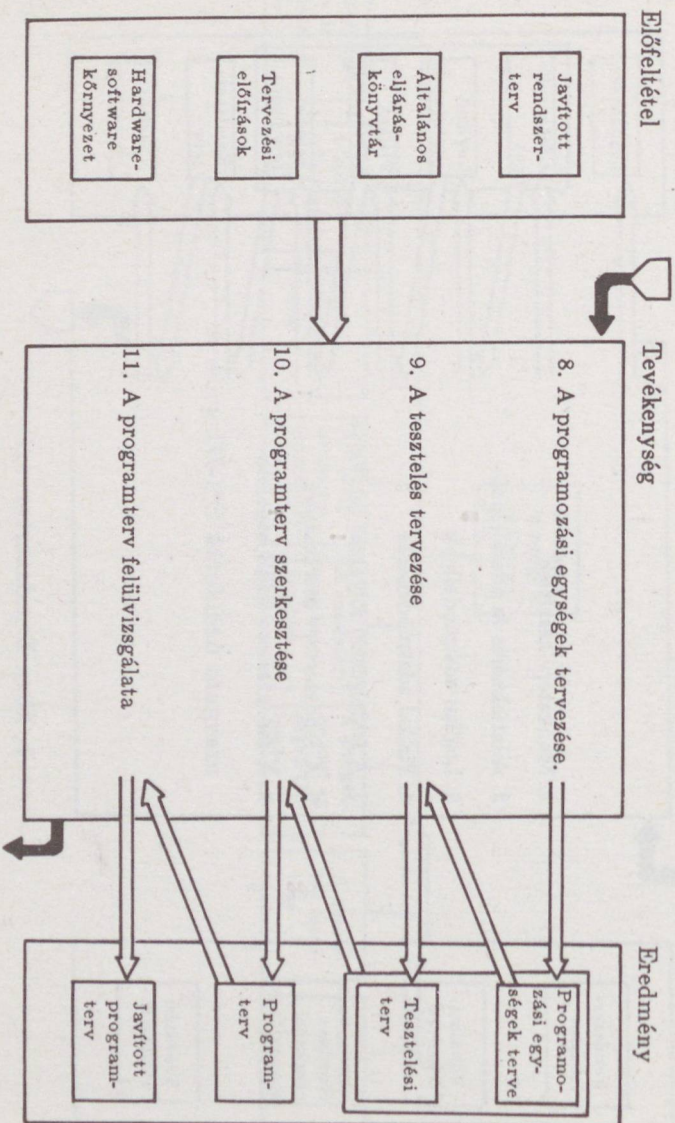
36. táblázat IDMS séma DDL szet



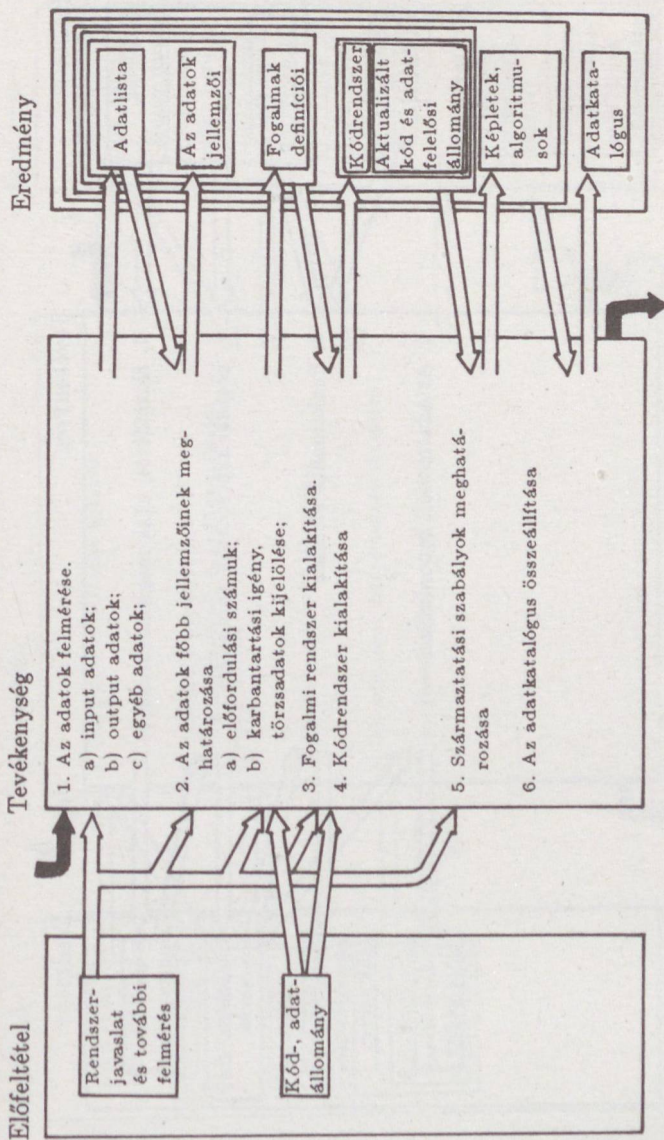
1. ábra HI-PO áttekinthető diagram



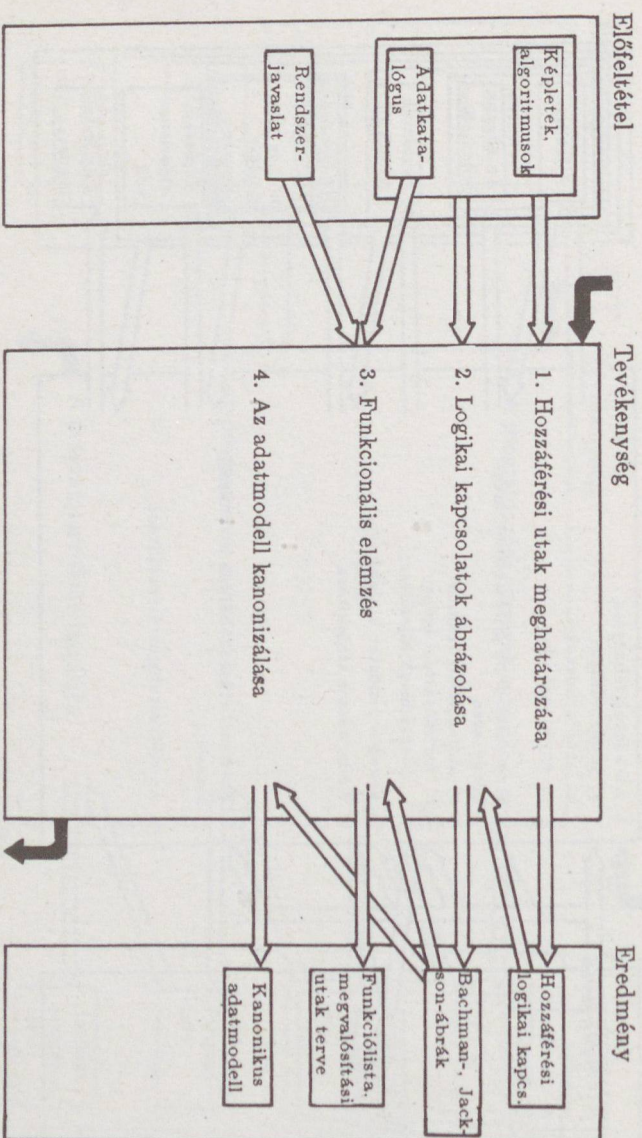
2. ábra Rendszertervezés



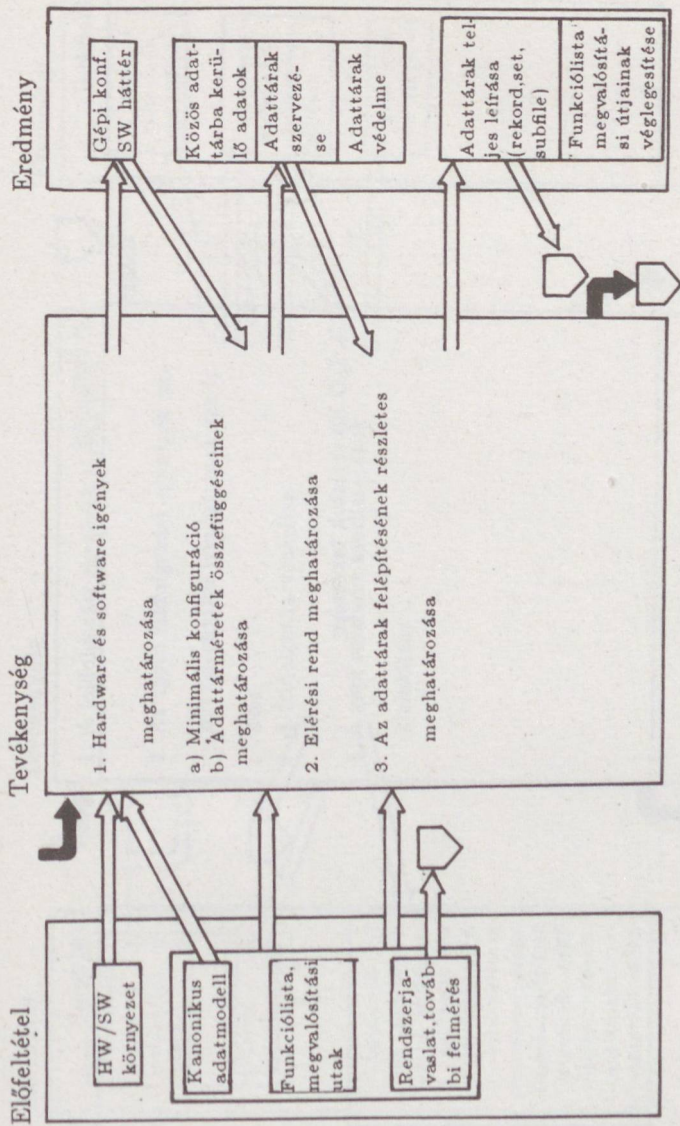
3. ábra Programtervezés



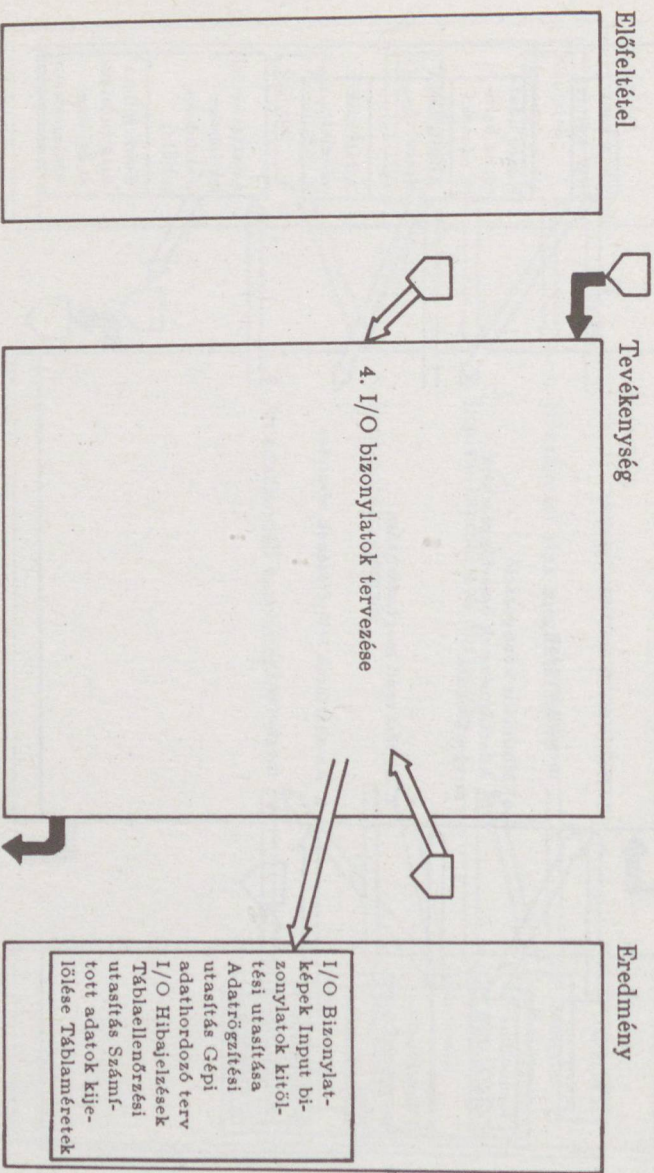
4. ábra Elvi adatmodellezés



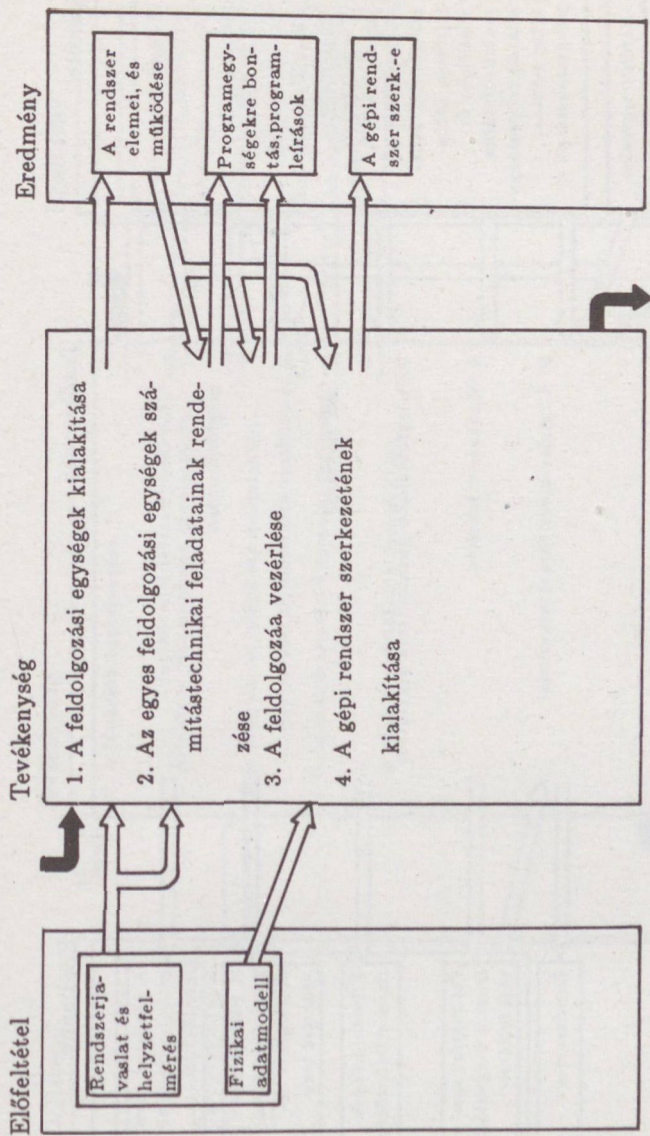
5. ábra Logikai adatmodellezés



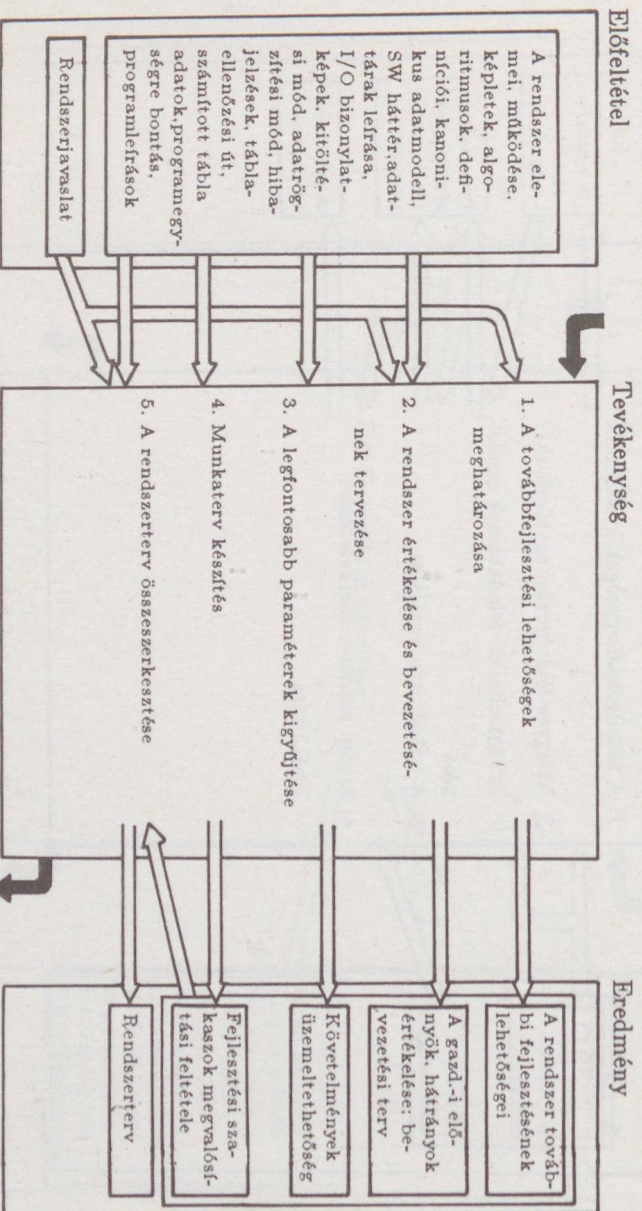
6. ábra Fizikai adatmodellezés



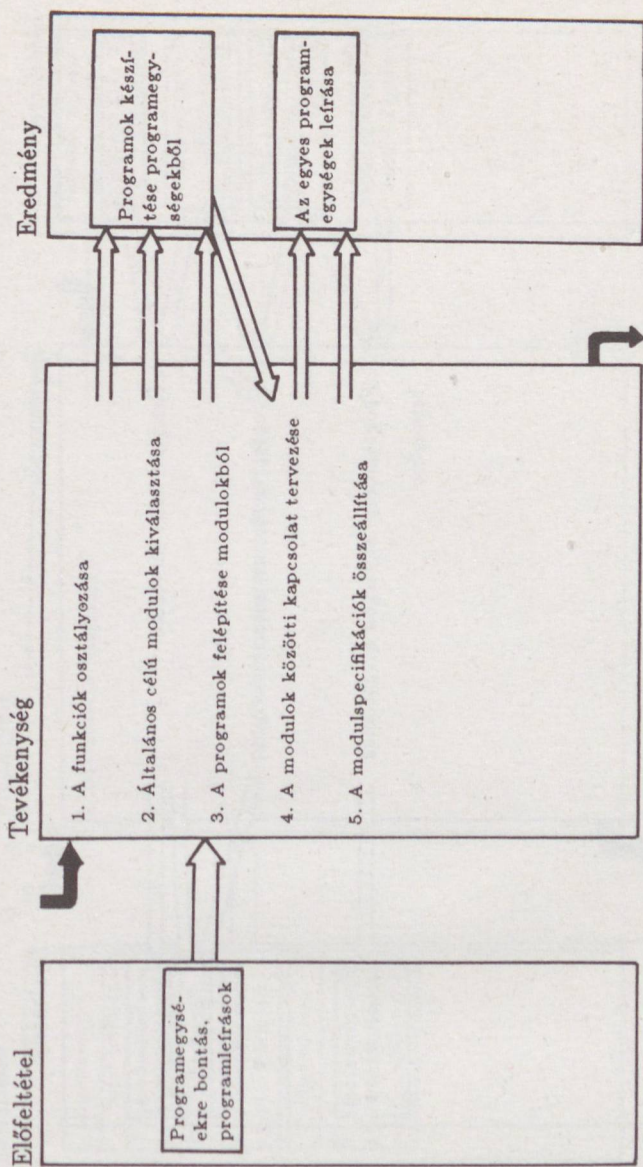
7. ábra Bizonylatok tervezése



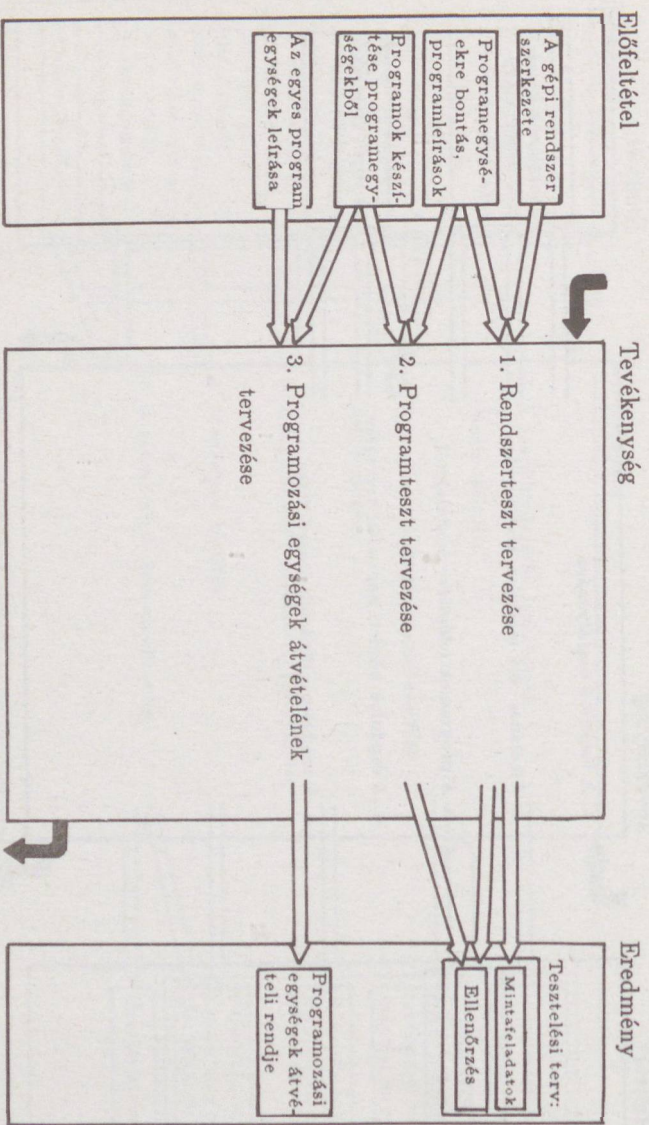
8. ábra Feldolgozási folyamatok tervezése



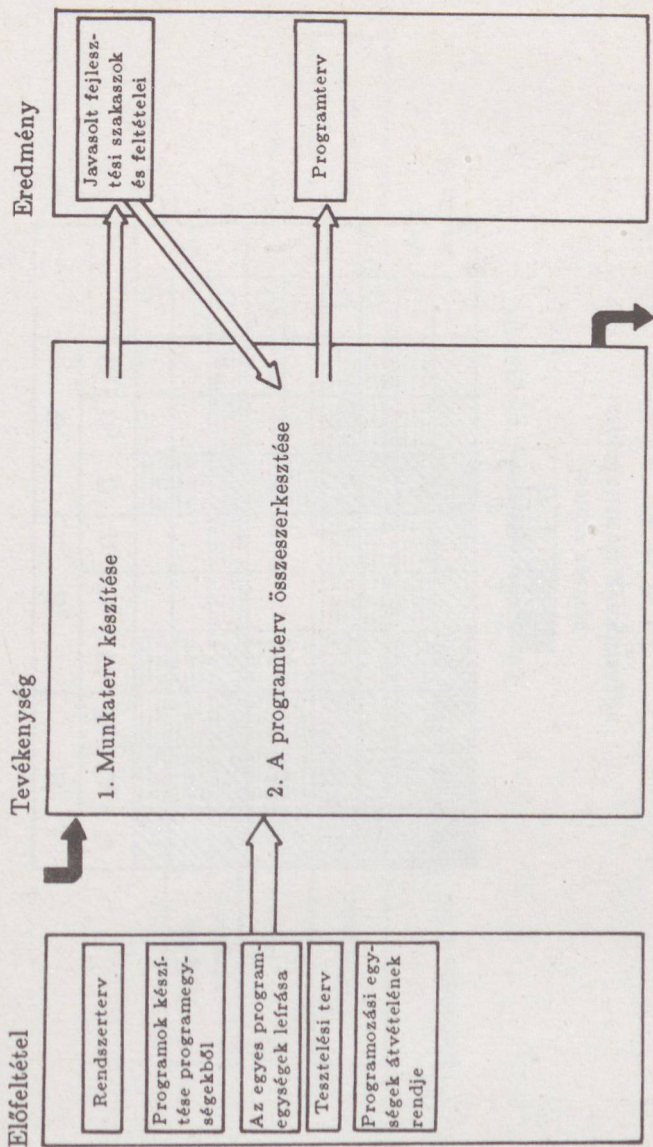
9. ábra Rendszerterv szerkesztése



10. ábra Programmodulok tervezése



11. ábra Teszttervezés



12. ábra Programterv szerkesztése

		B ₁			B ₂			B		
		D ₁	D ₂	D	D ₁	D ₂	D	D ₁	D ₂	D
	C ₁									
	C ₂									
	C									
A ₁	C ₁									
	C ₂									
	C									
A ₂	C ₁									
	C ₂									
	C									
A	C ₁									
	C ₂									
	C									

4 3 2 1 0

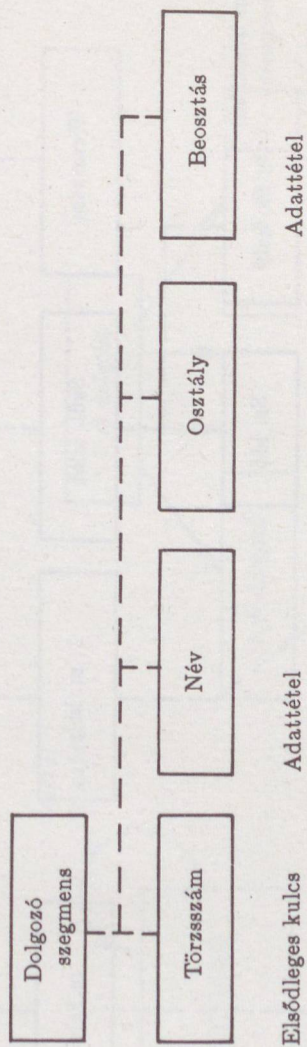


Ismérv szerint

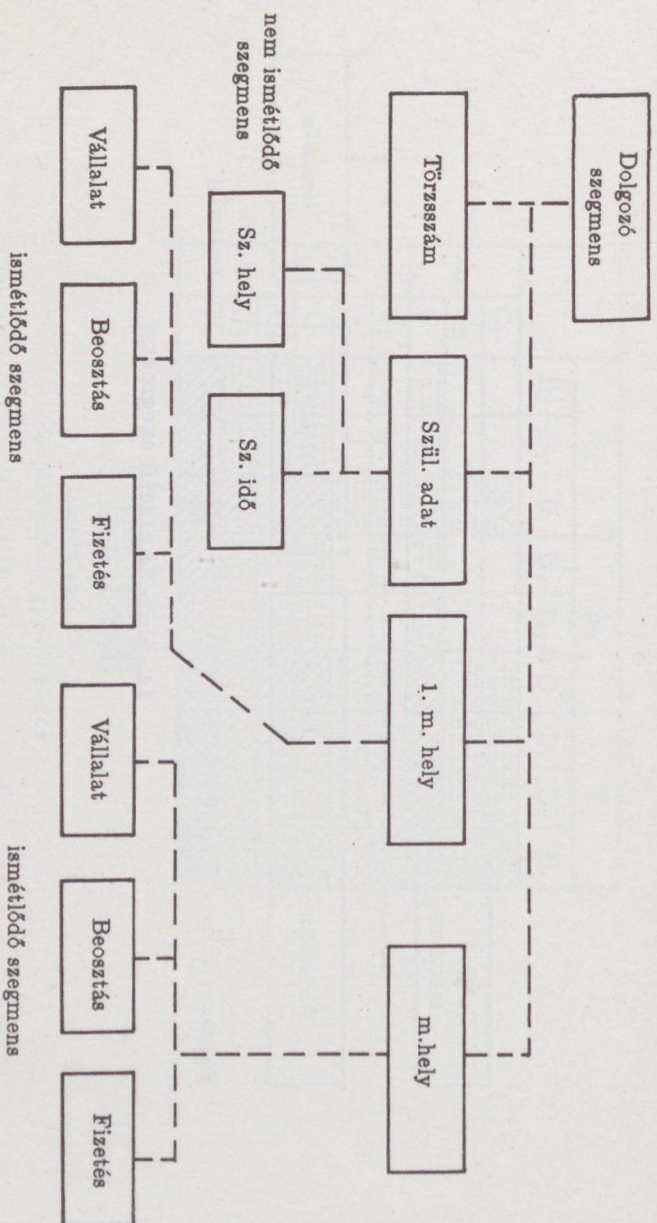
csoportosított gyakoriságok

Négy alternatív ismérv (A, B, C, D) szerint csoportosított.
A betűjelek melletti számok a csoportok sorszámát jelentik

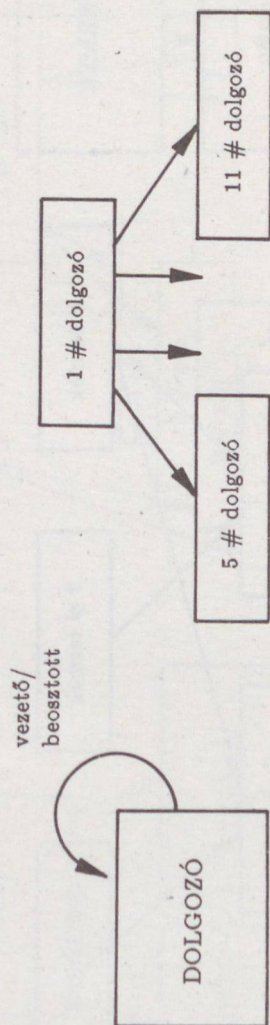
13. ábra A kombinációs tábla szerkezete



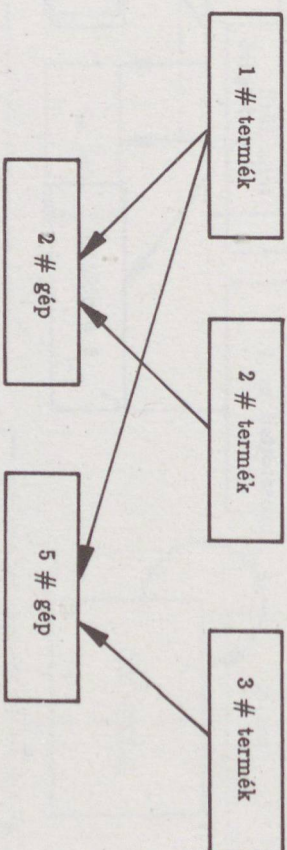
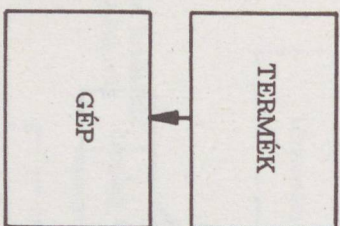
14. ábra Egyszerű szegmens



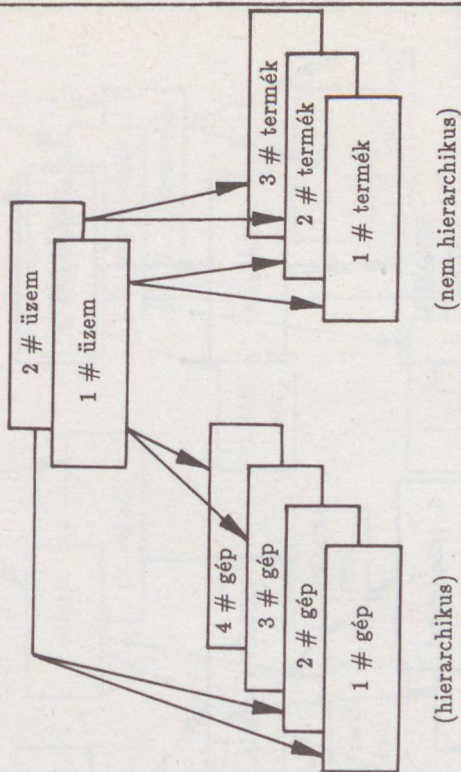
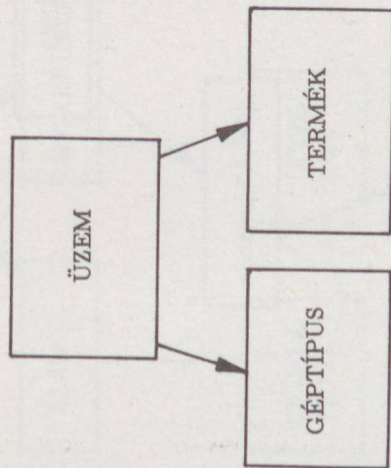
15. ábra. Összetett szegmens



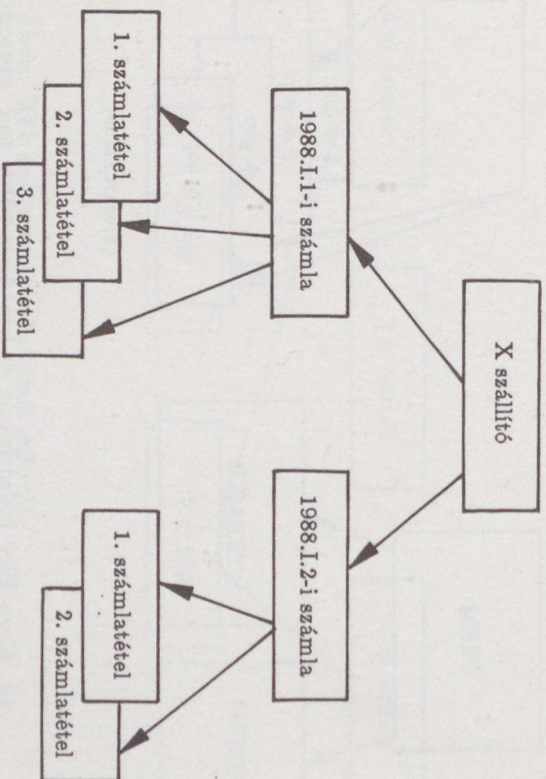
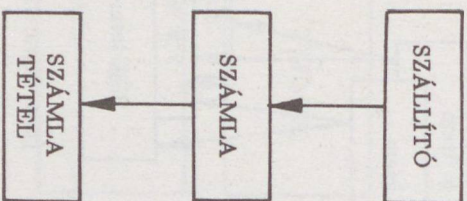
16. ábra. Egy szegmenstípuson belüli reláció



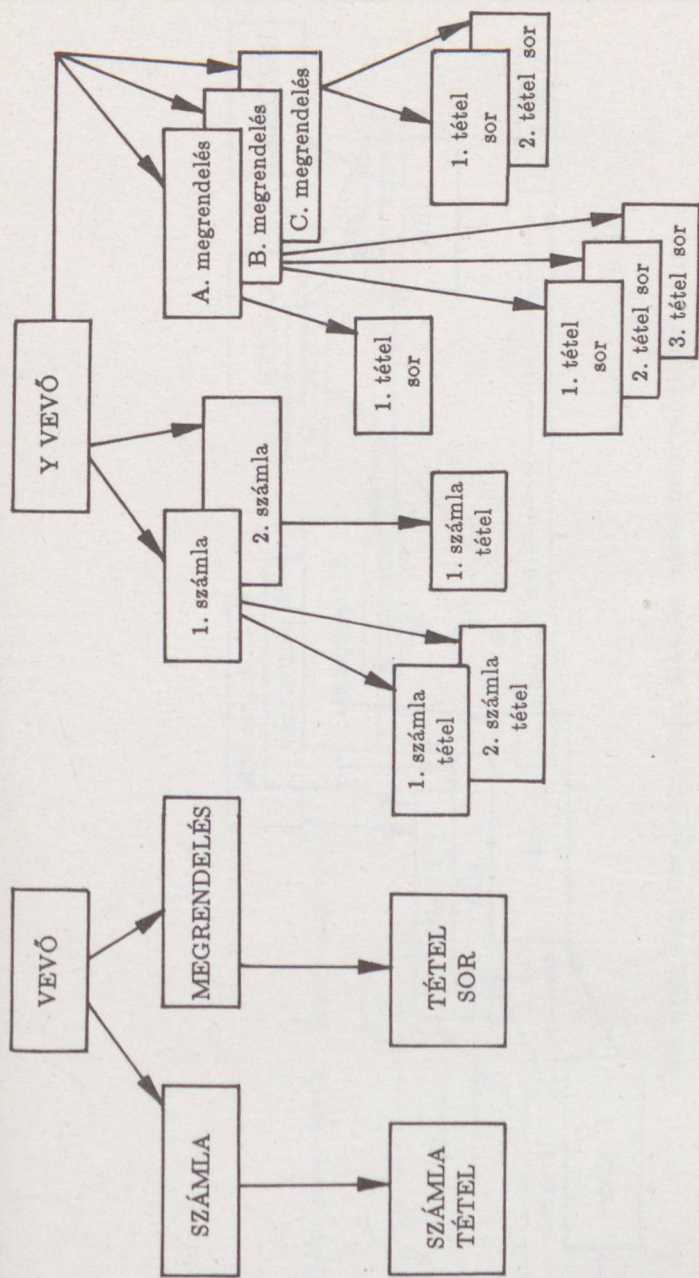
17. ábra. Két szegmenstípus közötti reláció



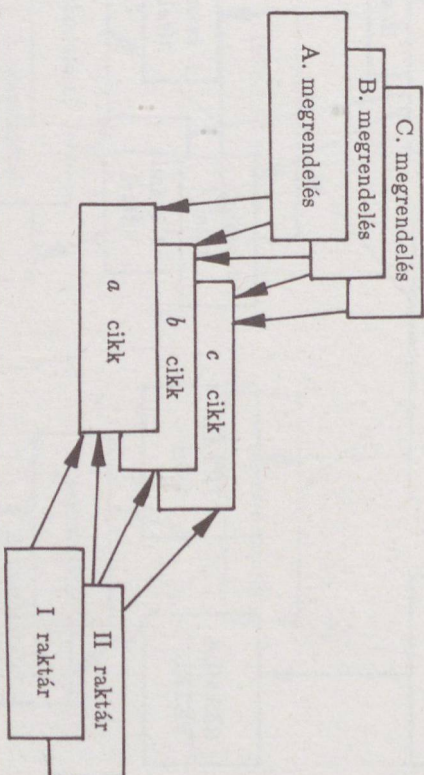
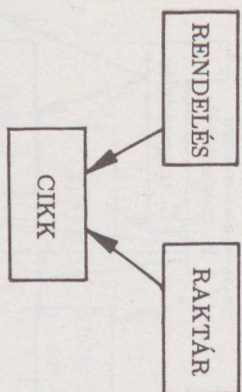
18. ábra Egy tulajdonos és kettőnél több tag szegmenstípus közti kapcsolat



19. ábra Egyszerű többszintű hierarchia

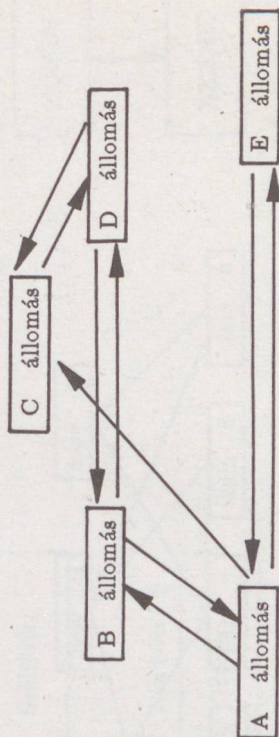
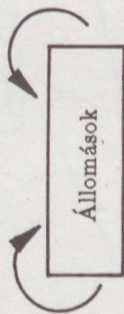


20. ábra Összetett többszintű hierarchia

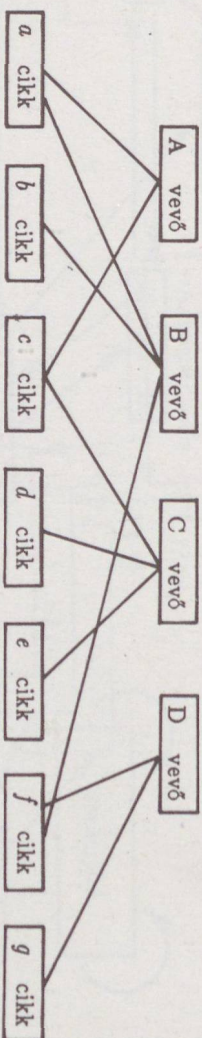
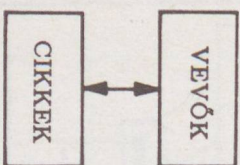


21. ábra Egyszerű hálós szerkezet

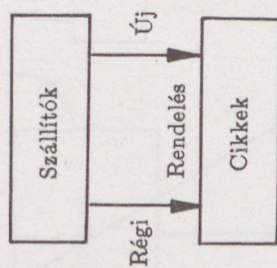
Induló szerelvények Érkező szerelvények



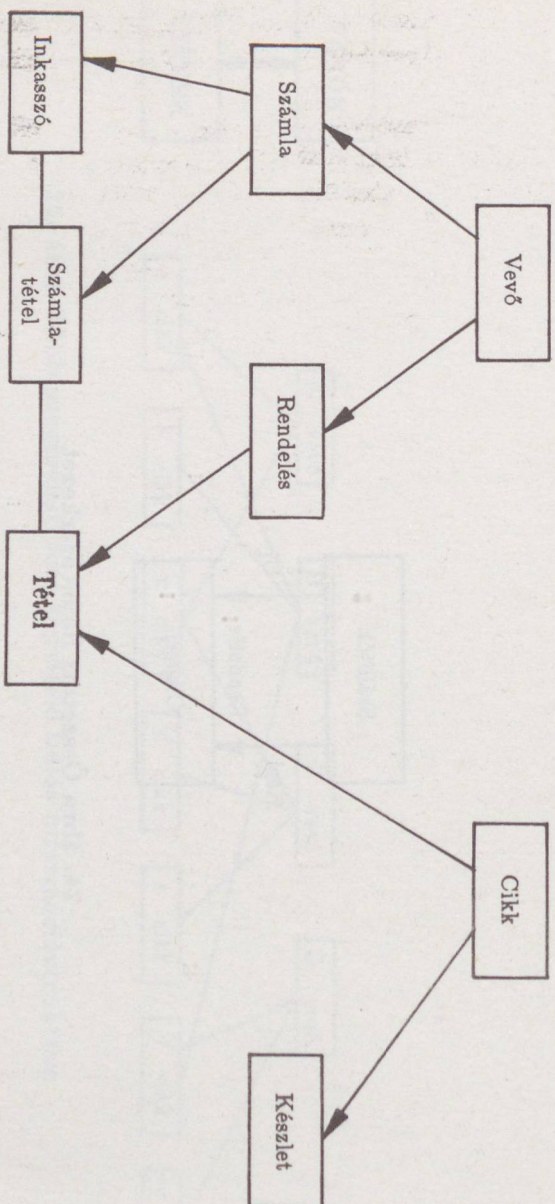
22. ábra Egy szegmenstípus alkotta hálós információstruktúra



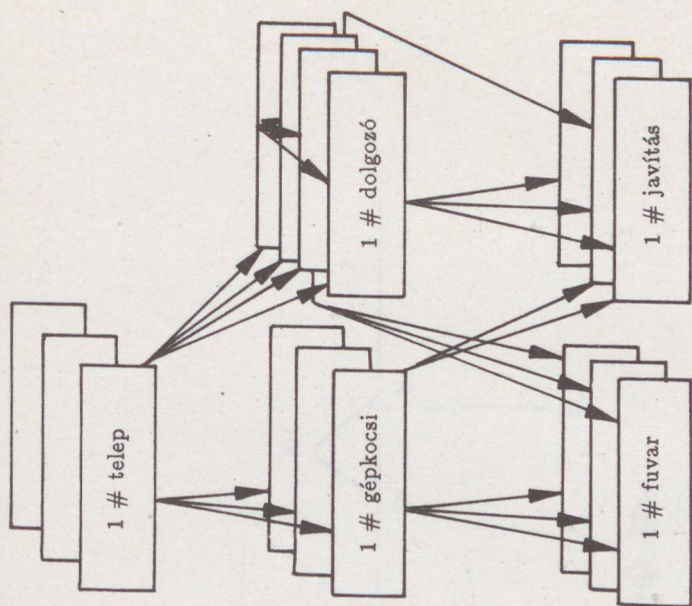
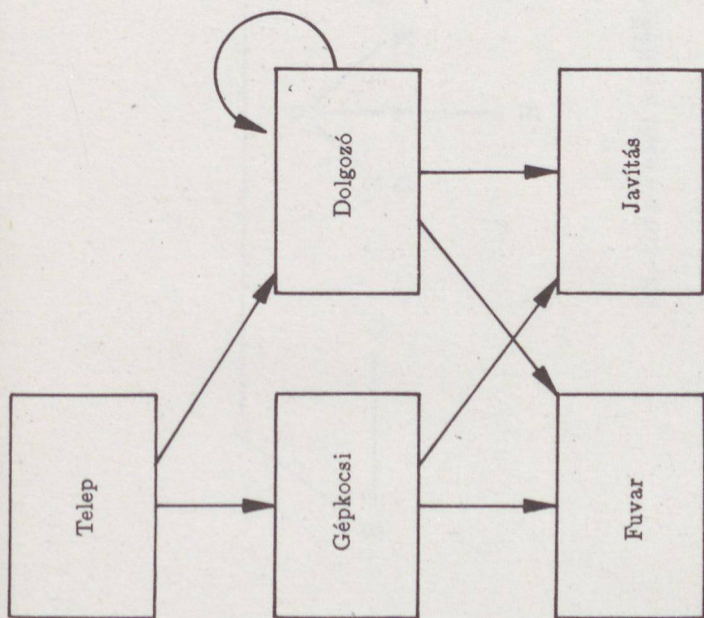
23. ábra Több szegmenstípust képviselő hálós információstruktúra



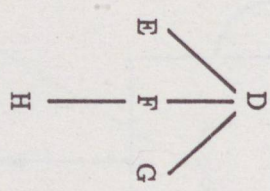
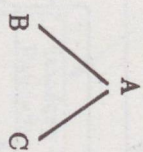
24. ábra Összetett hálós szerkezet



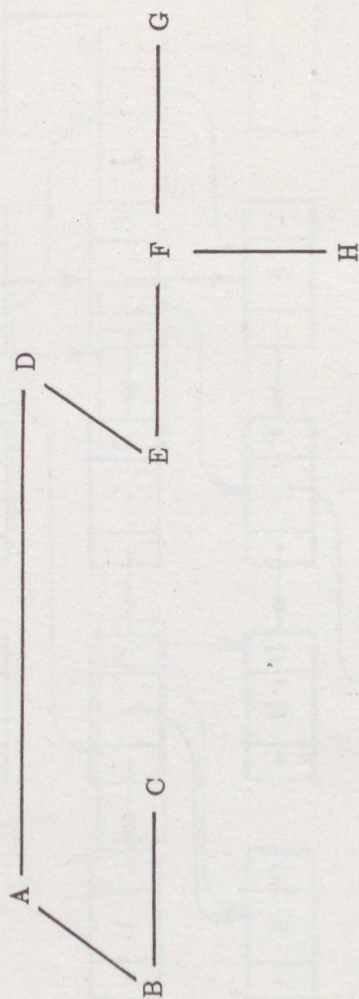
25. ábra Összetett hálós szerkezet (plex)



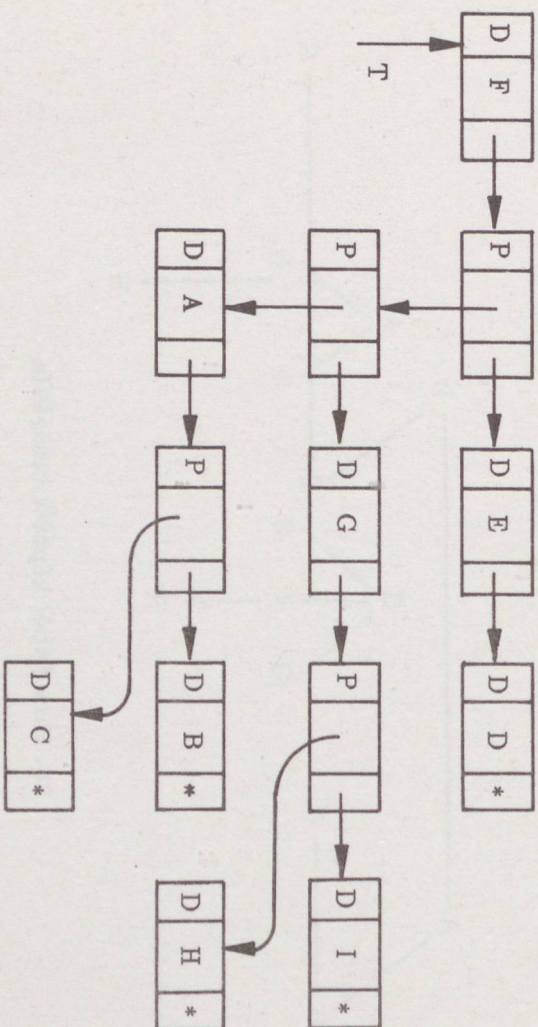
26. ábra Plex-típusú logikai rekord



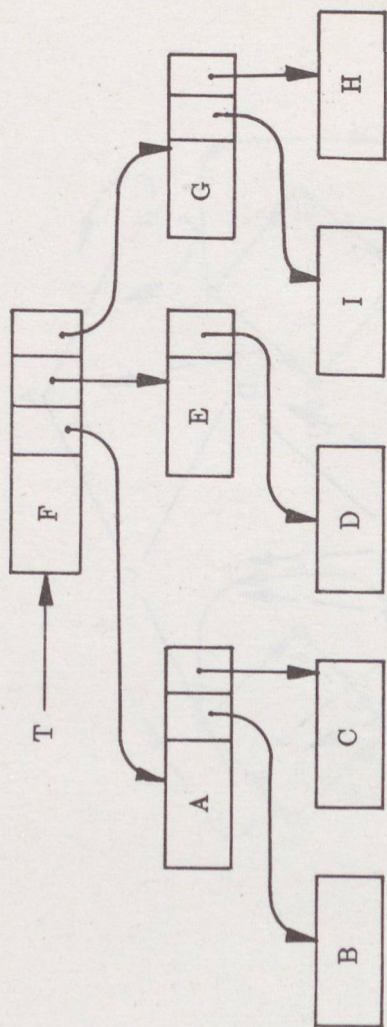
27. ábra Fák



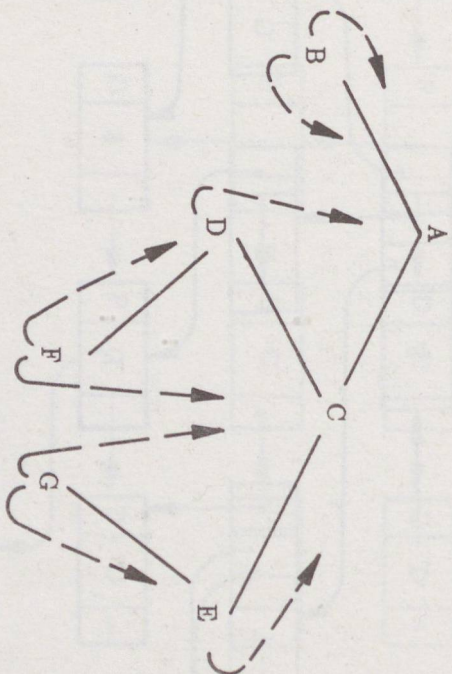
28. ábra Fából képzett bináris fa



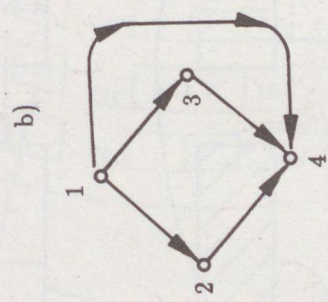
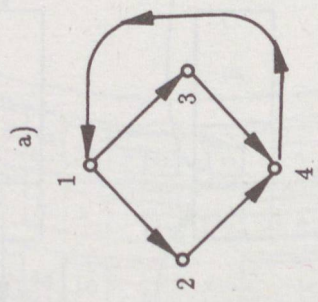
29. ábra Pointer-tömbös tárolás



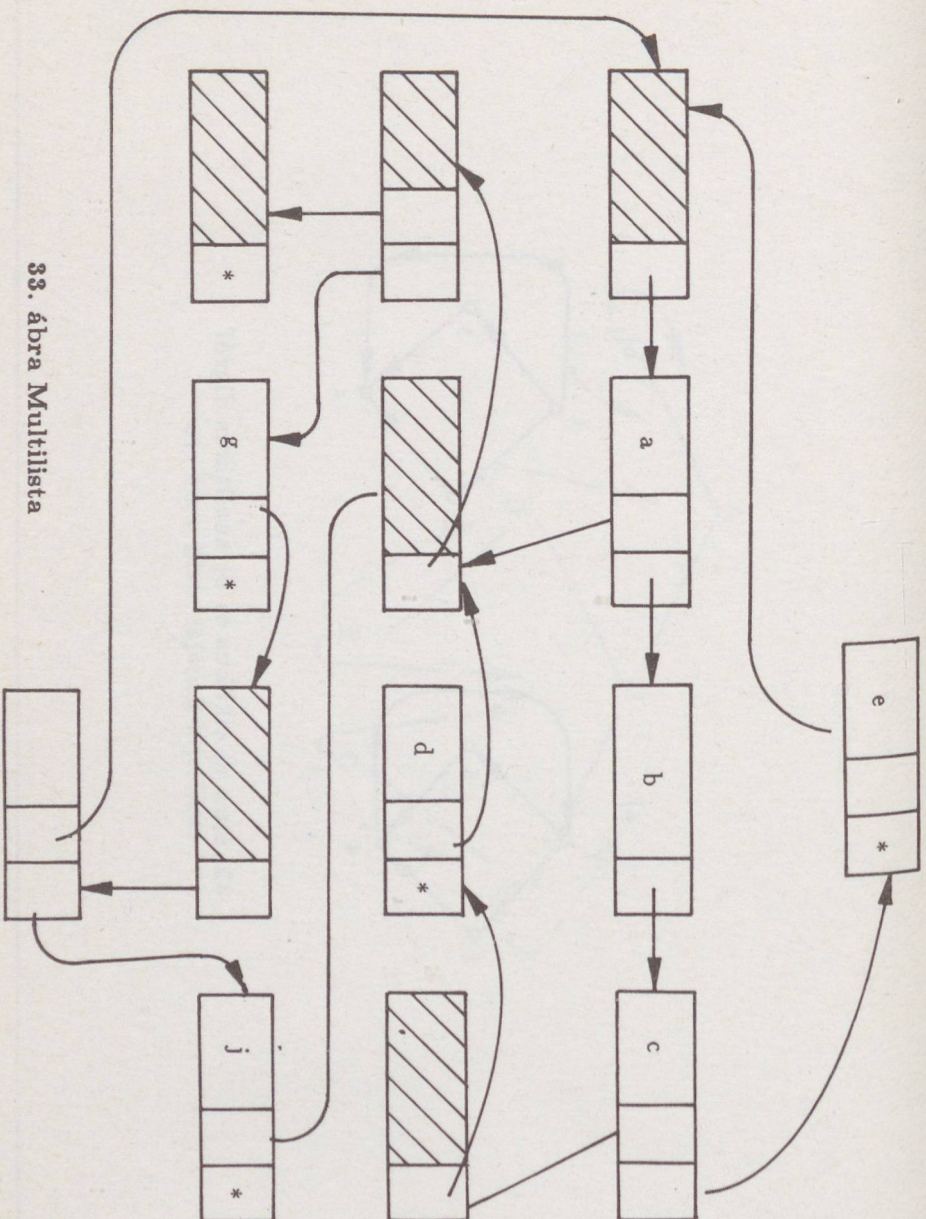
30. ábra Bináris fává alakítva tárolás



31. ábra Lista fejének kijelölése



32. ábra a) ciklikus és b) analitikus digráf



33. ábra Multilista

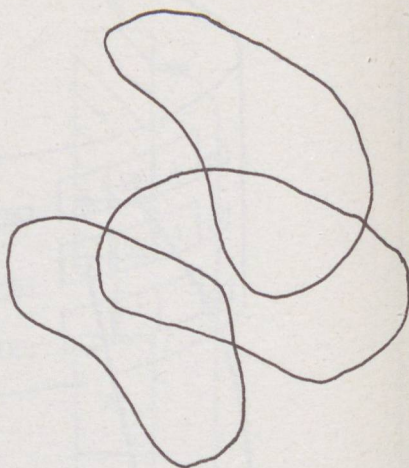


34. ábra Master és struktúra file

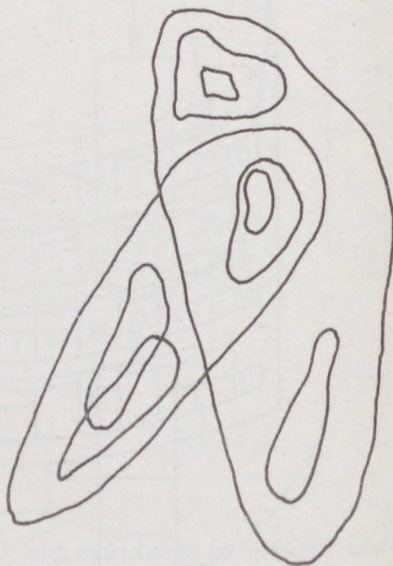
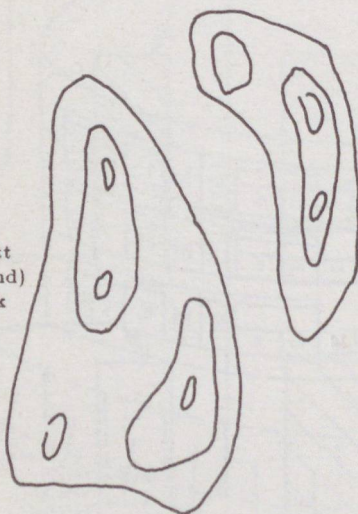
Elváló (partitional)

Átfedésses (overlapping)

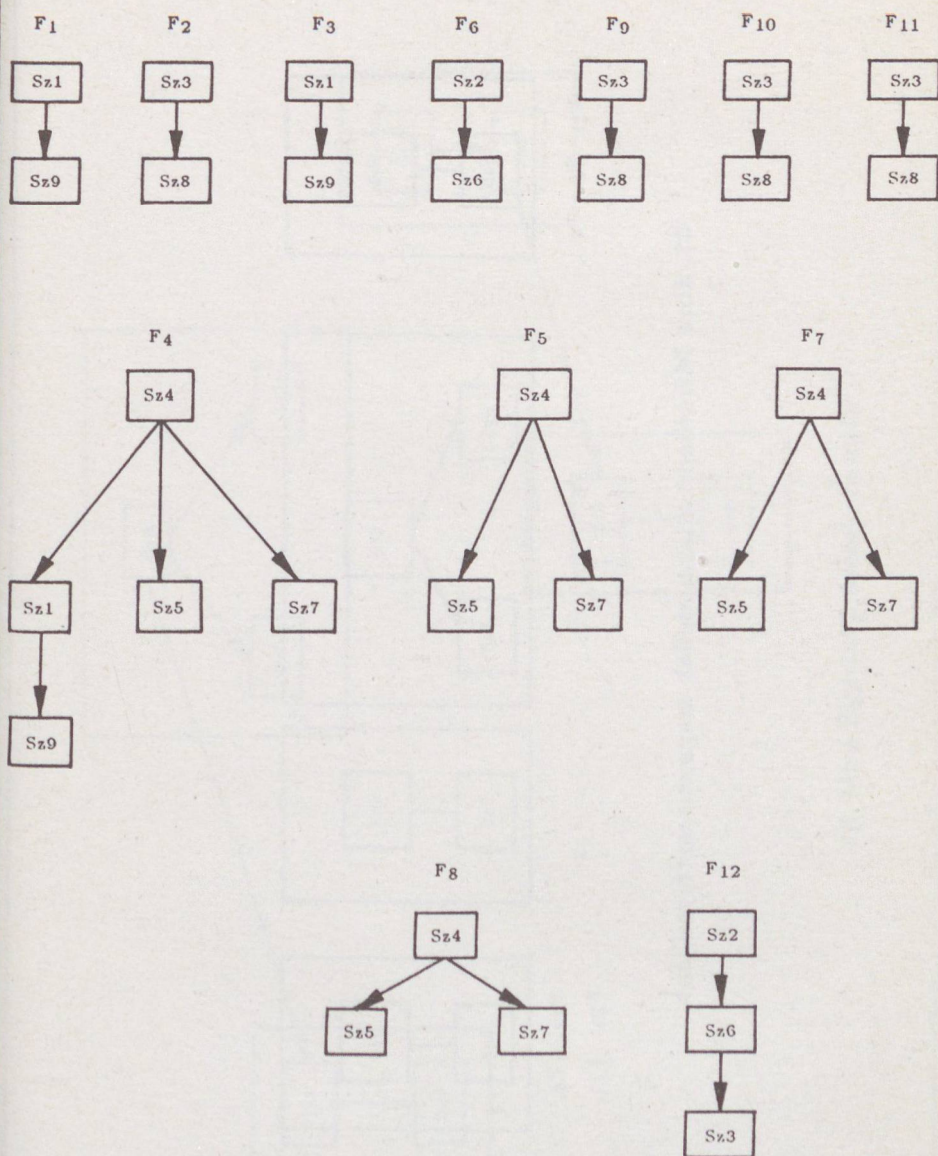
Egyszerű
(simple)
klaszterek



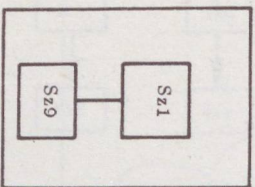
Rétegezett
(compound)
klaszterek



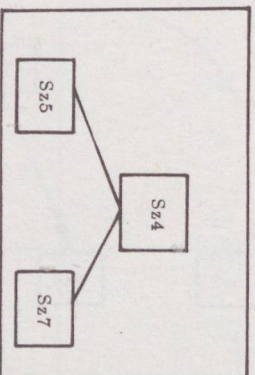
35. ábra A klaszerek egymáshoz való viszonya



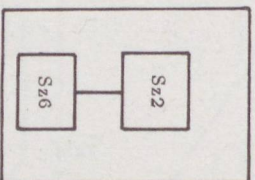
36. ábra Adatmodellek funkcióként



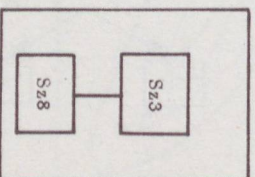
F₁, F₃, F₄.



F₄, F₅, F₇, F₈.

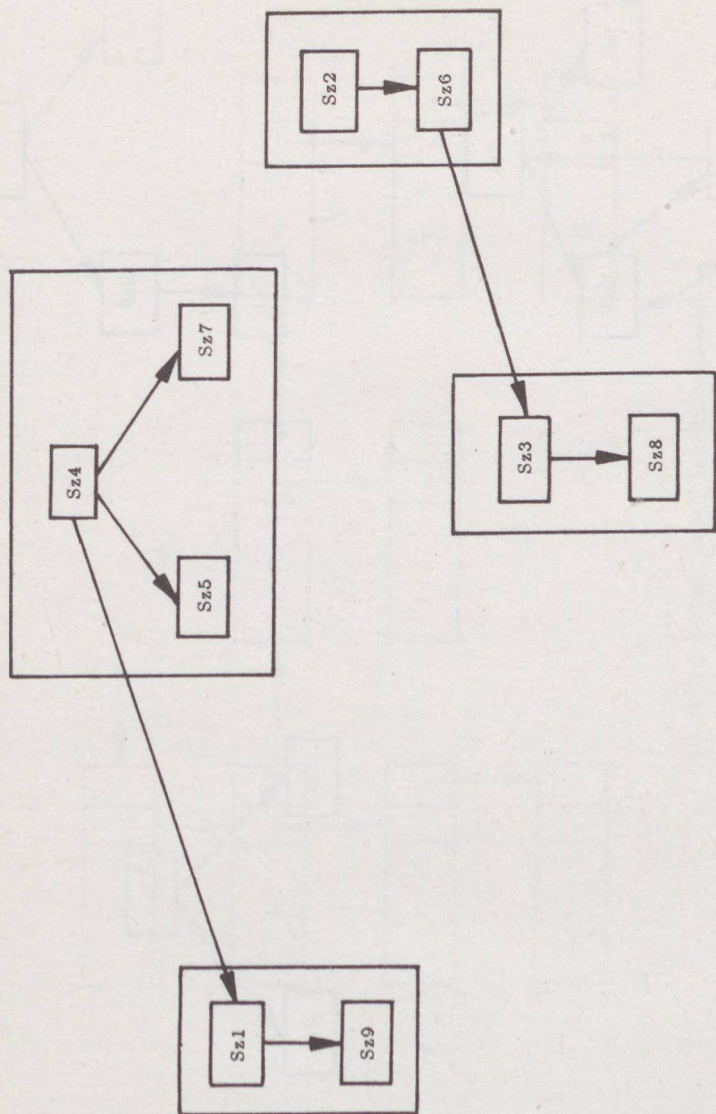


F₆, F₁₂.

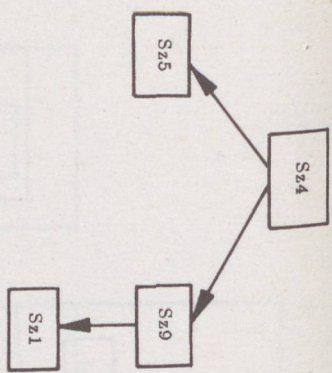


F₁₂, F₉,
F₁₀, F₁₁.

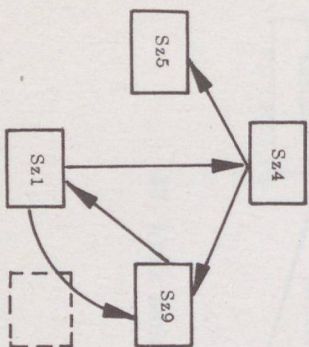
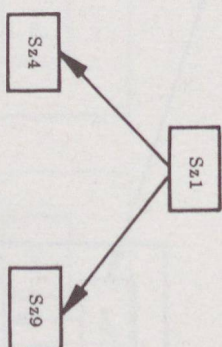
37. ábra Mértékdő adatmodellek szegmensosztályonként



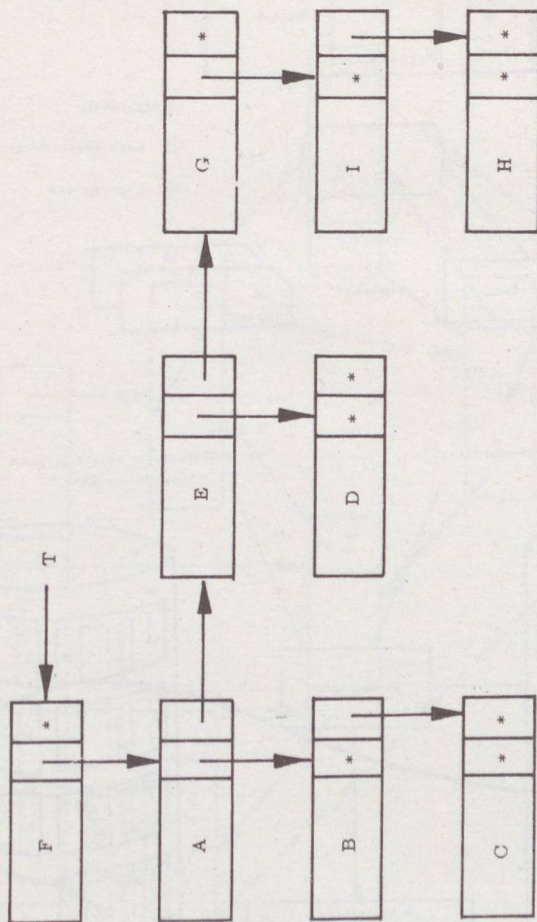
38. ábra Egyszerű összekapcsolás



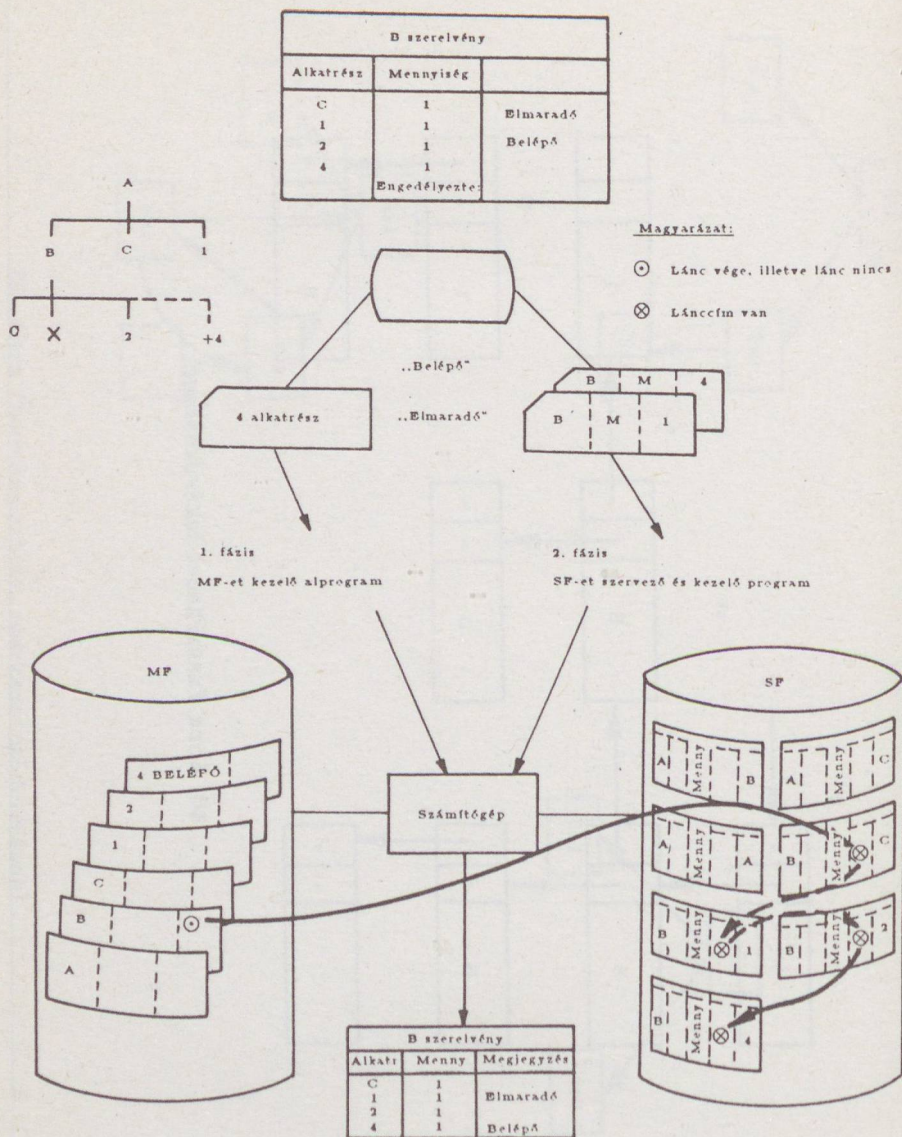
és



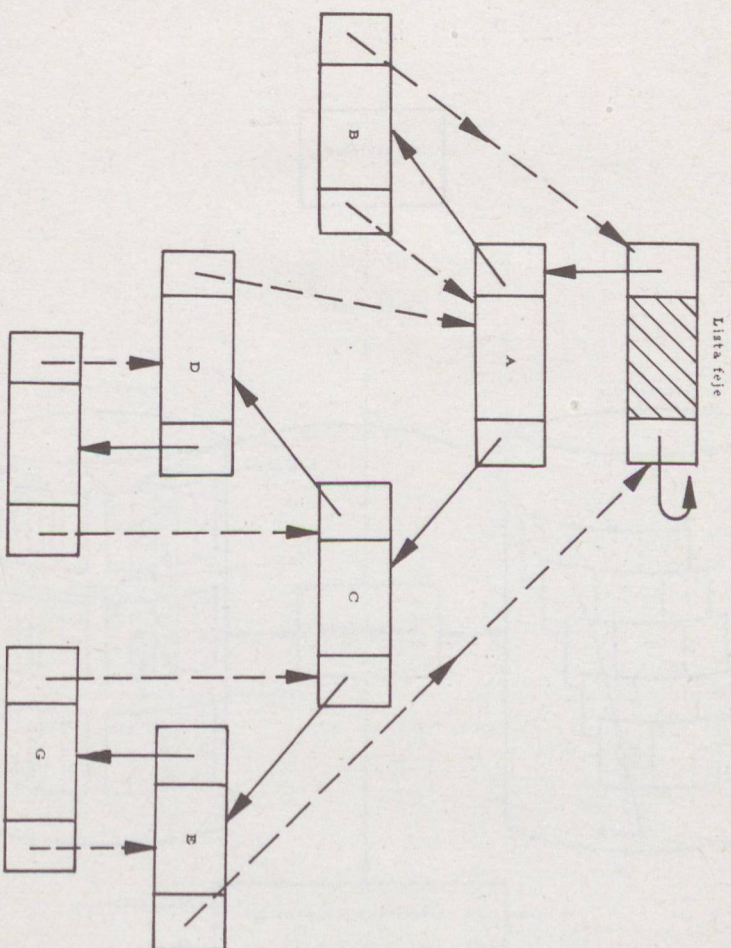
39. ábra Összekapcsolás, szerkezetmódosításai



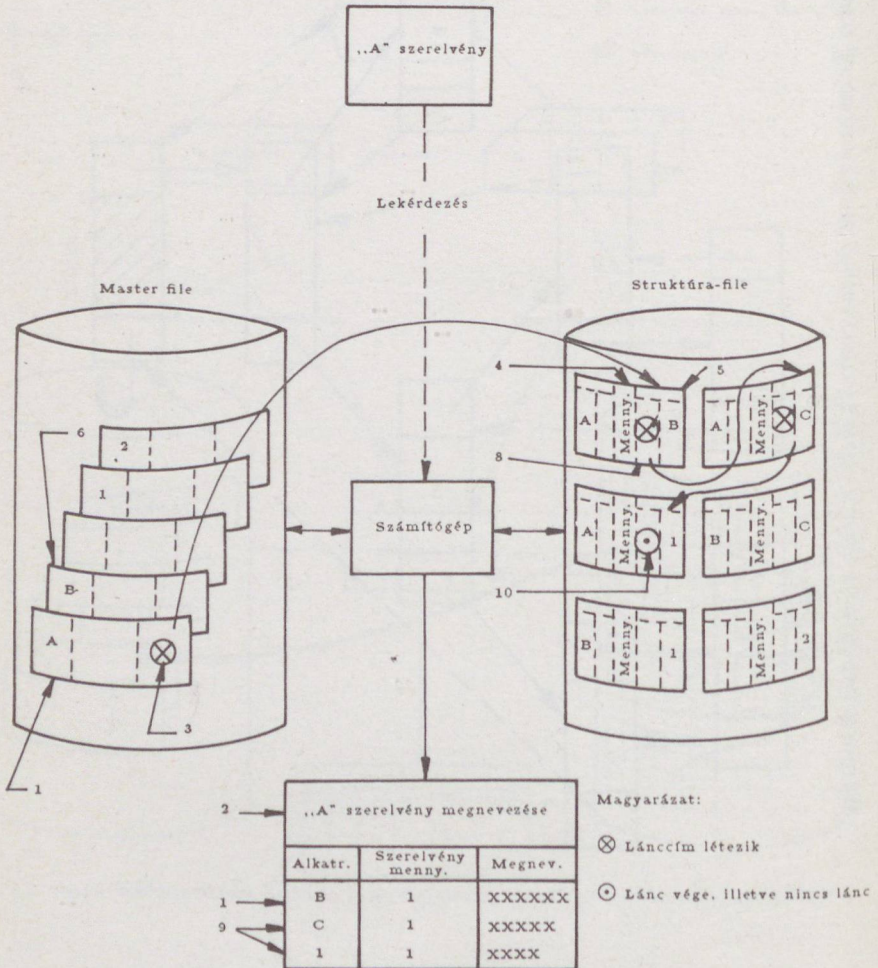
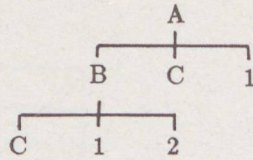
40. ábra Faszervezet lekérdezéssel



41. ábra Változtatás hatása a gyártmány szerkezeti láncállományára



42. ábrá Szimmetrikus visszacsatolt lista külön struktúra ále-ban



43. ábra Gyártmány szerkezet lekérdezése

Irodalomjegyzék

1. Anderberg: *Cluster Analysis for Applications*, Academic Press Inc. New York-London (1973)
2. Arany I.: *Another Method for Finding Pseudo-peripheral Nodes*, in *Annales, Sectio Computatorica*, tomus 4, ed. by Eötvös Lóránd University, Budapest (1983)
3. Arany I.: *How to Find Rooted Level Structure of Near-Minimum, Width*, *Bulletins for Applied Mathematics*, XXIX. Technical University, Budapest (1983 XXIX p. 75-95.)
4. Arany I.: *A Necessary Condition for Getting Acceptable Small Bandwidth for Sparse Matrices*, *PAMM Bulletins for Applied Mathematics* 192/82 (XXX) Technical University, Budapest (1983)
5. Arany I.: *Nagymértű ritka szimmetrikus mátrixok hatékony számítógépes kezelése*, Kandidátusi értekezés (1984)
6. Bolla, M.: *An Iterative Method for Clustering Simultaneously Objects and Variables of Dichotomous Characters DIANA-II*, Conference on Classification and their Application, Liblice (Czechoslovakia) (1986)
7. Bolla M.-Tusnády G.: *Parallel Clustering of Objects and Variables of Dichotomous Character*, International Working Conference, Computer Aided Medical Decision Making 1-4 October Prague (poster session) (1985)
8. Bana I.: *Osztott adatbázisok tervezése*, Budapest SZÁMOK (1985)
9. Békéssy A.-Demeterovics J.: *Contribution to the Theory of Database Relations*, *Discrete Mathematics* 27. (1979) p. 1-10.
10. Bentley-Frideman: *Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Space*, *IEEE Trans. Comput.* C-27/2 (1978) p. 97-105.
11. Berge C.: *The Theory of Graphs and its Applications*, John Wiley and Sons Inc., New York (1962)

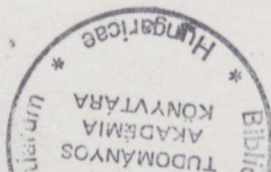
12. Bock, H.H.: *Automatische Klassifikation*, (Statistische Methoden II) (Ed. Walter) Lect. Notes on Oper Res. and Math. Syst. Vol. 39. (1970) p. 36-80.
13. Boorman, S.A.-Oliver, D.C.: *Metrics on Spaces of Finite Trees*, Journal of Mathematical Psychology (1973) 10. p. 26-59.
14. Boorman, S.A.-Arabie, P.: *Structural Measures and the Method of Sorting*, in: Multidimensional Scaling Volume I Theory eds. R.N. Sheparad, A.K. Romney, S.B. Nerlove 19 Seminar Press New York (1972) p. 225-249.
15. Bunke, H.: *What is the Distance Between Graphs?*, Bulletins of the European Association for Theoretical Computer Science (AETCS) 29. (1983) p. 35-39.
16. Cagan: *Term-term Correlation for Large Matrices*, Journal of the ASIS Vol. 21 (1970) p. 163.
17. Carroll, J. D.-Arabie, P.: *INDCLUS an Individual Differences Generalization of the ANCLUS Model and the MAPCLUS Algorithm* Psychometrika (1983) 48. p. 157-169.
18. Carroll, J.D.-Chang, J.J.: *Analysis of Individual Differences in Multidimensional Scaling via an N-way Generalization of Eckart-Young Decomposition*, Psychometrika, 35. (1970) p. 283-319.
19. Carroll, J.D.: *Spatial Non-spatial and Hybrid Models for Scaling*, Psychometrika 41. (1976) p. 439-463.
20. Carroll, J.D.-Pruzansky, S.: *Fitting of Hierarchical Tree Structure (HTS) Models, Mixtures of HTS Models via Mathematical Programming and Alternating Least Squares*, (USA-Japan Seminar of MDS, 20-24 August Univ. of California at San Diego, la Jolle, Calif. (1975)
21. Carroll, J.D.-Pruzansky, S.: *Discrete and Hybrid Scaling Models in Similarity and Choice*, ed. Lantermann and Feger Bern Hans Huber (1980)
22. Carroll, J.D.-Clark-DeSarbo, W.S.: *The Representation of Tree-Way Proximity Data by Single and Multiple Tree Structure Models*, Journal of Classification 1. (1984) p. 25-74.
23. Chen-Lee: *Some Algorithms Employing Nearest Neighbour Searching*, Institute of Comp. Dec. Sci. National Tsing Hua Univ. Hsinchu, Taiwan (1979)
24. Carmichael, J.W.-Sneath, P.H.A.: *Taxometric Maps*, Systematic Zoology 18. (1969) p. 402-415.

25. Codasyl Systems Committee: *A Survey of Generalized DBMS*, May. Report New York (1969)
26. Codasyl DBTG.: *Report*, April, New York (1971)
27. Codasyl Systems Committee: *Feature Analysis of Generalized DBMS*, Technical Report. May New York-London-Amsterdam (1971)
28. Demetrovics J.: *Reklamációs adatbázis modell*, MTA SZTAKI Közlemények Budapest (1978)
29. Cuthill, E.-McKee, J.: *Reducing the Bandwidth of Sparse Symmetric Matrices*, proc. 24-th Nat. Conf. Assoc. Comput. Mach. ACM Publ. (1969) p. 157-172.
30. Degerman, R.: *Ordered Binary Trees Constructed Through an Application of Kendall's Tan*, Psychometrika, 47. (1982) p. 523-527.
31. Day, W.H.E.: *The Complexity of Computing Metric Distances between Partitions*, Mathematical Social Sciences 1. (1981) p. 269-287.
32. Day, W.H.E.-Ebelsbrunner, H.: *Efficient Algorithms for Agglomerativ Hierarchical Clustering Methods*, Journal of Classification 1. (1984) p. 7-24.
33. Dijkstra: *A Note on Two Problems in Connection with Graphs*, Numer. Math. Vol. 1. (1959) p. 269-271.
34. Deutsh-Martin: *An Ordering Algorithm for Analysis of Data Array*, Oper. Res. 19/6 (1971) p. 1350-1362.
35. Dubin-Champoux: *Typology of Empirical Attributes: Dissimilarity Linkage Analysis (DLA)*, Technical Report 3. University of California, Irvine (1970)
36. Fischer: *Clustering and Aggregation in Economics*, John Hopkins Press, Baltimore, Maryland (1968)
37. Flament, C.: *Application of Graph Theory To Group Structure*, Englewood Cliffs, New Jersey, Prentice-Hall (1963)
38. Fritsche: *Automatic Clustering Techniques in Information Retrieval*, EURATOM (1974)
39. Futó P.: *Hipergráf modellen alapuló klaszterelemzés* Kandidátusi értekezés (1978)
40. Füstös L.: *A klaszteranalízis módszerei*, Módszertani füzetek MTA Szoc. Kut. Int. Budapest (1977)
41. Füstös L.-Meszéna Gy.-Simonné Mosolygó N.: *A sorváltozós adatelemzés statisztikai módszerei*, Akadémiai Kiadó Budapest (1986)

42. George, J.A.-Liu, J.W.H.: *An Implementation of a Pseudo-peripheral Node Finder*, ACM Trans on Math, Software, Vol. 5. No. 3. (1979) p. 284-295.
43. George, J.A.-Liu, J.W.H.: *Computer Solution of a Large Sparse Positive Definite Systems*, Prentice-Hall Inc. Englewood Cliffs, New Jersey (1981)
44. Gibbs, N.E.-Poole, W.G.-Stockmeyer, P.K.: *An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix*, SIAM J. Numer. Anal. Vol. 13. No. 2. (1976) p. 236-250.
45. Gulyás O.: *Osztályozási módszerek*, Többváltozós statisztikai analízis (szerk.: Móri F.T., Székely J.G.) Műszaki Könyvkiadó, Budapest (1986) p. 233-250.
46. Goodman, N.: *The Structure of Appearance*, Cambridge, Massachusetts. Harvard University Press (1951)
47. Gower-Ross: *Minimum Spanning Trees and Single Linkage Cluster Analysis*, Journal of Royal Stat. Ass. C: Appl. Stat. Vol. 18. (1969) p. 54-64.
48. Gower: *A Comparison of Some Methods of Cluster Analysis*, Biometrics 23. (1969) p. 623-637.
49. Gale, N.-Halperin, W.C.-Costanzo, C.M.: *Unclassed Matrix Shading and Optimal Ordering in Hierarchical Cluster Analysis*, Journal of Classification 1. (1984) p. 75-92.
50. Halassy B.: *Szintetikus adatmodellező*, SZIAM Budapest SZÁMALK (1981)
51. Halassy B.: *Adatmodellezés, adatbázis-tervezés*, Budapest SZÁMALK (1980)
52. Halassy B.: *Adatmodellezés a rendszerfejlesztésben* Budapest SZÁMALK (1983)
53. Hall: *Avoiding Informational Distortions in Automatic Grouping Programs*, System Zöll. 18. No. 3. (1969) p. 318-329.
54. Hall-Ball-Wolf-Eusebio: *PROMENADE: An Improved Interactive Graphics Man/Machine System for Pattern, Recognition Rep. No. RADCTR-68-572* Stanford Res. Inst. Melno Perk California (1969)
55. Harshman, R.A.: *PARAFAC-2*, Mathematical and Technical Notes Working Paper in Phonetics, UCLA, (1972) 22. p. 30-47.
56. Hartigan, J.A.: *Representation of Similarity Matrices by Trees*, Journal of the American Statistical Association 62. (1967) p. 1140-1158.

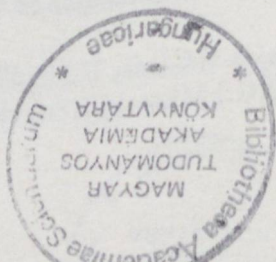
57. Hartigan, J.A.: *Direct Clustering of Data Matrices*, Journal. Amer. Stat. Ass. 67. New York John Wiley (1970)
58. Hartigan, J.A.: *Clustering Algorithms* New York, John Wiley (1975)
59. Horváth T.-Leitner L.-Mezey Gy.-Pongrácz T.: *A tanácsi információrendszer fejlesztése*, SZÁMALK (1983)
60. Horváth T.: *Automatikus osztályozás*, Könyvtári Figyelő 5. szám. (1978)
61. Jardine, N.-Sibson, R.: *Mathematical Taxonomy*, John Wiley, New York (1971)
62. Knuth, E.-Radó P.-Tóth A.: *Preliminary Description of SDLA*, Tanulmányok 105. MTA SZTAKI (1980)
63. Knuth, E.: *SMP, Computer Aid for a Simple Class of Semantic Transformations*, Report WP 11/19 MTA SZTAKI (1981)
64. Kendall, D.G.: *Seriation from Abundance Matrices*, Mathematics in the hods D:g. Kendall P. Tautu. Edinburgh University Press (1971) p. 215-252.
65. Lance-Williams: *Computer Programms for Hierarchical Polythetic Classification*, (Similarity Analysis) Computer J. 9. No. 1. (1966) p. 60-64.
66. Lance-Williams: *A General Theory of Classificatory Sorting Strategies*, (Clustering Systems) Computer J. 10. (1967) p. 271-277.
67. Lee-tseng: *Multikey Sorting*, International Journal of Policy Analysis and Information Systems 3/2/ (1979) p. 1-20.
68. Lee: *Clustering Analysis and its Applications Advances*, Information System Science (ed. Tou) vol 8. No. 16. (1981) p. 153-158.
69. Lenstra: *Clustering a Data Array and the Traveling Salesman Problem*, Oper. Res. 20//5/ sept. (1972) p. 993-1009.
70. Levenshtein, V.I.: *Binary Codes Capable of Correcting Deletions, Insertions and Reveals* Szovjet Fizika-Doklady, 10. (1966) p. 707-710.
71. Litofsky: *Utility of Automatic Classification for Information Storage and Retrieval*, Ph. D. Dissertation, Univ. of Pennsylvania, Philadelphia (1969)
72. Masuda: *Optimalisation of Program Organization by Cluster Analysis*, Proc. IFIP 74. (1979) p. 261-165.
73. Mezey Gy.: *Funkcia analizado Helpe de komputilo*, Interkomputo Budapest, NJSZT (1982) p. 136-146.

74. Mezey Gy.: *Cluster-Analysis and Logical Data Base Design*, DATE-BASE '83, Budapest június 6-8- OMIKK (1983) p. 124-127.
75. Mezey Gy.: *Szegmentsképzés adatbázis-tervezés során*, Automatizálás, Budapest, 9. szám (1983) p. 25-32.
76. Mezey Gy.: *Determinisztikus szükséglettervezés számítógépes adatbázisának kialakítási lehetőségei*, Szakmérnöki diplomaterv és doktori disszertáció, Budapest (1975)
77. Mezey Gy.: *Gyártmánycsaládfa lebontás számítógépes megoldásai és azok alkalmazási lehetőségei*, (Ügyvitelszervezési Megoldások sorozat) PM SZÜI, Budapest 2. szám (1975) p. 1-150.
78. Mezey Gy.: *A klaszteranalízis alkalmazási lehetőségei adatbázis szegmentsképzésével*, Mérés és automatika, Budapest 6. szám (1984) p. 222-225.
79. Mezey Gy.: *Mapping to Storage of a Network*, Structure MTA SZTAKI Közlemények, Budapest 30. (1984) p. 91-99.
80. Mezey Gy.: *Clustering in Logical Data Base Design*, International Working Conference, Computer Aided Medical Decision Making 1-4 Oktober Prague (1985)
81. Mezey Gy.: *Mapping of a Network Structure*, COMPSTAT '84 (6th Symposium on Computational Statistics) 27-31. August, Prague (1984)
82. Mezey Gy.: *Mapping to Storage of Data Model Network Structure Computation*, Statistics Quarterly. Vol. 2. Issue 1. (1985) p. 119-124.
83. Mezey Gy.: *Klaszterálás alkalmazása hálós adatbázis logikai tervezése során*, (Separatum) Alkalmazott Matematikai Lapok Budapest, Akadémia Kiadó (1985) p. 239-296.
84. Mc Cormick-Schweitzer-White: *Problem Decomposition and Data Reorganisation by a Clustering Technique*, Open. Res. (1974) 22/2 p. 413-414.
85. McQuitty, L.L.: *Single and Multiple Hierarchical Classification by Reciprocal Pair and Rang Order Types*, Educational and Psychological Measurement 24. (1966) p. 253-265.
86. McQuitty, L.L.: *Capabilities and Improvement of Linkage Analysis as a Clustering Method*, Educational and Psychological Measurement 24. (1964) p. 441-456.
87. Nishigaki-Nogi-Miyamoto: *Segmens Organization by Cluster Analysis*, Information Processing in Japan No. 16. (1976) p. 153-158.



88. McCammon, R.B.-Wenninger, G.: *The Dendrograph Computer Cont-rib.* No. 48. Univ. of Kansas (1970)
89. Papadimitriou, Ch.H.: *The NP-completeness of the Bandwidth Mi-nimalisation Problem*, Computing 16. (1976) p. 263-270.
90. Párniczky L.: *A statisztikai informatika alapjai*, SKV, Budapest (1976)
91. Párniczky L.: *Doktori disszertáció* MTA TMB. (1980)
92. Prim: *Shortest Connection Matrix Network and Some Generalisati-ons*, Bell System Techn. J. 36. p. 3189-1401.
93. Restle, F.: *A Metric and an Ordering on Sets*, Psychometrika 24. (1959) p. 207-220.
94. Robinson, D.F.-Foulds, L.R.: *Comparison of Phylogenetic Trees*, Mathematical Biosciences 53. (1981) p. 131-147.
95. Robinson, D.F.: *Comparison of Labeled Trees With Valency Tree*, Journal of Combinatorial Theory 11. (1971) p. 105-119.
96. Robinson, W.S.: *A Method for Chronologically Ordering*, Archeolo-gical Deposits American Antiquity 16. (1951) p. 293-301.
97. Salton, G.: *Automatic Information Storage and Retrieval*, McGraw-Hill, New York (1968)
98. Sammon: *On-line Pattern Analysis and Recognition System*, (OL-PARS) Rep. No. RADGTR-68-263 Rome Air Develop Center, Griffiss Air-Force Base, New York (1968)
99. Slagle-Chang-Heller: *Experiments With some Clustering Analysis Algorithms*, Pattern Recognition 6. (1974) p. 181-187.
100. Slagle-Chang-Heller: *A Clustering and Data- reorganisation Algo-rithm*, IEEE Trans, Systems Man Cybernet SMC-15/1 (1975) p. 121-128.
101. Sokal, R.K.-Snecath, P.H.A.: *Princioles of Numerical Taxonomy* San Francisco W. H. Freeman (1963)
102. Sörgel: *Mathematical Analysis of Documentation System*, Informa-tion Storage Retrieval Vol. 3. (1967) p. 129-173.
103. Teichroew, D.-Knuth, E.-Radó P.-Kang, K.C.: *Concept Refinement Approach CRA*, University of Michigan. Ann Arbor. Mich. (1981)
104. Telegdi L.: *Bináris változók struktúrájának vizsgálata*, Kandidátusi értekezés, Budapest (1985)
105. Telegdi L.: *Többdimenziós skálázás, Többváltozós statisztikai analí-zis* (szerk.: Móri F. T., Székely J.G.) Műszaki Könyvkiadó Budapest, (1986)

106. Tversky, A.-Sattath, S.: *Additive Similarity Trees*, Psychometrika, 42. (1977) p. 319-345.
107. Torgerson: *Theory and Methods of Scaling*, John Wiley and Sons, Inc. New York (1960)
108. Van Rijsbergen, C.J.: *Information Retrieval*, Butterworths (2 Nd. ed.) (1979)
109. Zahn: *Graph-theoretical Methods for Detecting and Describing gestalt clusters*, IEEE Trans. Comput. C-20/1 January (1971) p. 68-86.



Mezey Gyula — Lévai Gábor **Adatbáziselemek osztályozása**

Napjainkban szinte közhelynek számít, hogy a számítástechnika alkalmazásának súlypontja az információfeldolgozásra került, és így egyre nő az adatbázis kezelésének jelentősége. A nagy adatbázisok igen bonyolultak, tervezésük az utóbbi évek során külön tudománnyá terebélyesedett, mely világszerte a szakemberek érdeklődésének középpontjában áll.

Magyarországon különösen fontos e kérdéssel foglalkozni, mivel nálunk sok felhasználó megelégszik azzal, hogy az üzembe állított adatbázis-kezelő rendszer működjék, jóllehet egyre inkább indokolt lenne a hatékonysággal is törődni. A relatív közömbösségnek okát nemcsak a gazdasági ösztönzők hiányában kell keresnünk, hanem abban is, hogy ezeddig nem állt rendelkezésre elég közismert, a tervezést jól segítő eszköz.

Most tehát azt a könyvet és szoftvert tartja Ön a kezében, amely tudományos alapossággal foglalkozik a tervezés egyes kérdéseivel és egyben gyakorlati eszköz is az információs rendszer tervezéshez.

A könyv első része a nagy adatbázis rendszerek kialakításának módjait, fogalmait tárgyalja, a második rész a szűkebb szakma számára is újdonságot tartalmazó klaszterálási módszerrel foglalkozik, míg a szoftverrel az adatbázisban tárolt adatok és az adatbázis használata során végrehajtandó feladatok kapcsolata elemezhető.

A szerzők már jónéhány szervezésük során kipróbált és alkalmazott módszerüket adják most tovább.

