tanulmányok

156/1984

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

RELÁCIÓS ADATBÁZISKEZELŐ RENDSZEREK ÖSSZEHASONLITÓ VIZSGÁLATA

Irta:
Radó Péter

Tanulmányok 156/1984

A kiadásért felelős:

DR. VÁMOS TIBOR

Főosztályvezető:

DEMETROVICS JÁNOS

ISBN 963 311 173 O
ISSN 0324-2951

Coopinvest 84—150

A TANULMÁNYSOROZATBAN 1983-BAN MEGJELENTEK

140/1983 Operation Research Software Descriptions (Vol.1.) Szerkesztette: Prékopa András és Kéri Gerzson

141/1983 Ngo The Khanh: Prefix-mentes nyelvek és egyszerü determinisztikus gépek

142/1983 Pikler Gyula: Dialógussal vezérelt interaktiv gépészeti CAD rendszerek elméleti és gyakorlati megfogalmazása

143/1983 Márkusz Zsuzsanna: Modellelméleti és univerzális algebrai eszközök a természetes és formális nyelvek szemantikaelméletében

144/1983 Publikációk '81 /Szerkesztette: Petróczy Judit/

145/1983 Telcs András: Belső állapotu bolyongások

146/1983: Varga Gyula: Numerical Methods for Computation of the Generalized Inverse of Rectangular Matrices

147/1983 Proceedings of the joint Bulgarian-Hungarian workshop on "Mathematical Cybernetics and data Processing" /Szerkesztette: Uhrin Béla/

148/1983 Sebestyén Béla: Fejezetek a részecskefizikai elektronikus kisérleteinek adatgyüjtő, -feldolgozó rendszerei köréből

149/1983 L. Keviczky, J. Hethéssy: A general approach for deterministic adaptive regulators based on explicit identification

150/1983 IFIP TC.2 WORKING CONFERENCE "System Description Methodologies" May 22-27. 1983. Kecskemét. /Szerkesztette: Knuth Előd/

151/1983 Márkusz Zsuzsanna: On First Order Many-Sorted LOGIC

152/1983 Operations Research Software Descriptions /Vol.2./ Edited by A. Prékopa and G. Kéri

153/1983 T.M.R. Ellis: The automatic generation of user--adaptable application-oriented language processors based on quasi-parallel modules

154/1983 Publikációk'82 /Szerkesztette: Petróczy Judit/

1984-BEN EDDIG MEGJELENTEK

155/1984 Deák István, Hoffer János, Mayer János, Németh Ágoston, Potecz Béla, Prékopa András, Straziczky Beáta: Termikus erőmüveken alapuló villamosenergia-rendszerek rövidtávu, optimális, erőmüvi menetrendjének meghatározása hálózati feltételek figyelembevételével

Néhány technikai megjegyzés

1. A tanulmány szövegében egyes szavak, kifejezések alá vannak huzva folyamatos vonallal. Szándékunk szerint ezek azok a fogalmak, megnevezések, néha rész vagy egész mondatok, melyek szövegkörnyezetükből kiemelve is jellemzik az aktuális témát, és igy a rész, fejezet és paragrafus-cimekkel támpontul szolgálhatnak a tanulmányt gyorsan átfutni kivánó, csak az őt érdeklő részekben elmélyedő Olvasó számára. Az aláhuzott szövegrészekkel jellemzett egységek részegységeire vonatkoznak a szaggatott vonallal aláhuzott fogalmak, megnevezések.

2. Figyelembe véve, hogy az adatbáziskezelő rendszerek készitése nem csak tudományos kutatás tárgya, hanem virágzó iparág is, a rendszerek müködésének, felépitésének leirása a szakirodalomban sok esetben nem található meg egészen pontosan, néha félmondatokból, célzásokból kell kihámozni az /esetleg téves, ellentmondásos vagy kétértelmü/ információt. Szükségesnek véltük ezért, a nem köztudott és sokszor megerősitett tények közlésénél és felhasználásánál a forrás pontos megnevezését. Annyiban tértünk el az általános gyakorlattól, hogy néhány esetben mondatokon kivül is szerepelnek hivatkozások. Ilyenkor a hivatkozás az egész megelőző logikai egység /a tartalom alapján remélhetően jól elkülönülő egy vagy több bekezdés, esetleg egész paragrafus/ tartalmára vonatkozik.

TARTALOMJEGYZÉK

0. BEVEZETÉS

A relációs adatmodell sikerének okát a szakemberek általában két tényezőben látják:

1/ az adatszerkezet egyszerüsége;

2/ a modell matematikai megalapozottsága [LACR 83] . Ami a 2/ okot illeti, tény, hogy ez az alapozás az adatmodell 1970-es bevezetése óta napjainkig nagy intenzitással folyik. Az is tény azonban, hogy ez a munka még ma sem látszik befejezettnek, és ez az 1/ tényező igazságát is megkérdőjelezi. Valóban olyan egyszerü a relációs adatmodell?

Az adatkezelés "relációs "korszakának" kezdetét 1970-től E.F. Codd alapvető cikkének [CODD 70] megjelenésétől szokás számitani. Ebben a cikkben a szerző definiálja az elképzelés alapjául szolgáló "reláció" fogalmát, és néhány, az adatszerkezeten végezhető müveletet. A reláció fogalma mellé rögtön be is vezeti a "normálforma" fogalmát.

Erről a normálformáról röviddel később ő maga mutatja meg, hogy ez csak az első normálforma ugyanis [CODD 71b] - ben bevezeti a második és a harmadik normálformát. Jelenleg óvatos becslések szerint is öt normálformáról beszélhetünk [KENT 83] . A különféle normálformáknak, a normálformáju relációkat előállitó algoritmusoknak nemzetközi és hazai viszonylatban is igen komoly, matematikai felkészültséget igénylő irodalmuk van [DEME 81].

Az elmélet tehát semmiképpen nem nevezhető egyszerünek az első, felületes vizsgálatok után. A gyakorlat pedig - mint általában - még kuszább, összetettebb képet mutat. Milyen adatbáziskezelő rendszer nevezhető egyáltalán relációsnak?

1970-ben erre még egyszerü volt a válasz: amik relációkon, tehát fizikai adatszervezés - független táblázatokon manipulálnak [CODD 70] . 1971-ben Codd javaslatot tett egy manipulációs nyelvre is [CODD 71a] , majd

az akkori és későbbi nyelvek "erősségére" - vagyis kifejezőképességükre, a nyelven leirható és az adatbáziskezelő rendszer által megvalósitandó müveletek minimális bonyolultságának jellemzésére - mércét állitott fel [CODD 71c]. Ekkor még - természetesen - nem léteztek magukat "relációsnak" nevező adatbáziskezelő rendszerek.

Az 1979-es válasz - a nagyszámu relációs adatbáziskezelő rendszer létrejötte után - már összetettebb és óvatosabb [CODD 79]. Az adatbáziskezelő rendszer tökéletesen relációs, ha támogatja

1/ a relációs adatmodell szerkezeti elemeit
2/ bizonyos beillesztési, felujitási, törlési szabályokat
3/ a relációs algebrát, ill. egy legalább azzal ekvivalens erősségü adatkezelő nyelvet

A csupán az 1/ , 2/ feltételeket teljesitő rendszer félig relációs.

1981-ben a meghatározás még árnyaltabbá vált [CODD 82]. Megjelentek ugyanis olyan rendszerek, melyek "relációsnak" nevezték magukat ugyan, de teljesitményük - főként az adatkezelő nyelv biztositotta lehetőségek terén - elég sok kivánnivalót hagyott maga után.

Az 1979-ben "félig relációs"-nak hivott rendszereket Codd 1981-ben már csak "táblás"-nak /tabular/ nevezi. Minimálisan relációs rendszerek azok, melyek lekérdező nyelve képes a három alapvető relációs müvelet /SELECT, PROJECT, JOIN/ realizálása. A relációsan teljes rendszernek már az elsőrendü predikátumkalkulussal ekvivalens lekérdező nyelvvel kell rendelkeznie, a tökéletesen relációs rendszernek pedig kezelnie kell emellett a hiányosan megadott relációsorokat is [LIPS 79], [CODD 79], továbbá támogatni kell az adatok integritását is.

Az "egyszerü" és "matematikailag megalapozott" relációs adatmodell, és az ezekre támaszkodó rendszerek tehát korántsem könnyen áttekinthetőek, hiszen az alapvető fogalmak

is folyton változnak, fejlődnek. Jelen tanulmány célja nem lehet a rendteremtés, inkább a különféle elképzelések, vélemények ismertetésére szorítkozik.

0.1. A relációs adatmodell

Az adatszerkezetek - a programozási nyelvek és az adatbáziskezelés elméletében használt értelemben egyaránt - a logikai adatok szerkezetével /pl. verem, sor, fa/, és az ezeken definiált müveletekkel /a verem legfelső elemének kiemelése, uj sorelem beillesztése, fa valamilyen stratégia szerinti bejárása/ jellemezhetők az adatszerkezet felhasználója számára. A relációs adatmodellt adatszerkezetként kezelve először a logikai adatokat, majd a rajtuk definiált müveleteket irjuk le. Az adatok "felhasználó-orientáltak", a számitógéphez nem értő felhasználó számára ismerős, szemléletes fogalmakkal irhatók le. A müveletek absztraktabbak, az uj rendszerek ezeket igy nem is használják, de a különféle lekérdező nyelvek alapjául ezek szolgálnak.

0.1.1. A relációk

A relációs adatbázisok alapegysége a reláció, egyszerüen táblázatnak tekinthető, melynek adatai oszlopokban ill. sorokban helyezkednek el. További, gyakran használt analógia a "rekord" szemlélet, ahol a relációnak a file, egy relációs sornak egy rekord, a sor elemeinek pedig a rekord mezői felelnek meg.

Név	Alapbér	Szül.év	Osztály neve	Oszt.helye
Kiss Pál	5400	1946	Munkaügyi	I. em
Nagy Elek	4800	1954	Játékok	II. em
Simon János	6000	1944	Elektromos cikkek	II.em
Kovács Zsuzsa	3200	1964	Játékok	II. em
Nagy Elek	3200	1964	Textil	III. e-
.	.	.	.	
.	.	.	.	
.	.	.	.	

/1. ábra/

Az egyszerü "táblázatos" vagy "rekordos" szemléletet néhány szabály pontositja [SAND 81]:

- minden táblázat csak egy rekordtipust tartalmaz;
- minden sor rögzitett számu, saját névvel rendelkező mezőből áll;
- a mezők mind különbozők és egyszerüek /atomikusak/, ismétlődő csoportok, és összetett mezők /olyanok, melyek maguk is relációk/ nem megengedettek /ez [CODD 70] első normálformája/;
- minden rekord egyedi - duplikátumok nem megengedettek;
- a rekordok sorrendje közömbös;
- a mezők értékeiket egy meghatározott értékkészletből /domain/ veszik;
- ugyanaz az értékkészlet több különböző mezőhöz is felhasználható;
- uj táblázatok hozhatók létre két létező táblázat azonos értékkészlethez tartozó mezői értékeinek egyezése alapján.

Egy mező értékkészlete általános, alkalmazás-független halmaz /egész számok alsó és felső határ között/, vagy pedig alkalmazás-függő /az áruház osztályainak nevei/ lehet. [CODD 79] javasolja minden relációhoz az E-értékkészlet, ill. egy pótlólagos mező bevezetését. Az E-értékkészlet /E mint entitás/ lényegében a relációs sorok belső azonositóinak halmaza volna és a megfelelő mező egyszerüen a sor azonositását szolgálná azzal, hogy a sor belső azonositóját tartalmazná a felhasználó számára hozzáférhető módon. Ez az elképzelés az entitás-kapcsolat /entity-relationship/ modell [CHEN 76] felé általánositaná a relációs adatbázisokat, ugyanis a felhasználó igy könnyebben definiálhatná a különböző táblázatba tartozó sorok /entitások/ közötti összefüggéseket - reláció formában /hiszen minden sornak egyedi, explicite hivatkozható "neve" lenne/.

Másik érdekes általánositása a relációs modell értékkészlet lehetőségeinek az absztrakt adatbázis adattipusok. Itt a mező értékkészlete absztrakt adattipus, melyre a különféle rendszerek különböző definiciós lehetőségeket biztositanak. Az absztrakt adattipus értékkészletü reláció jellegzetes példája az SDLA rendszer [KNUT 80].

A jelenleg müködő adatbáziskezelő rendszerek rendszerint nem támogatják a bevezetett "értékkészlet" fogalmat teljes általánosságban. Általában az operációs rendszer által támogatott értékkészletekre /karaktersorozat, egész szám, stb./ szoritkoznak, ezek közül választhat a felhasználó.

A relációt precizebben a matematikai fogalommal /értékkészletek Descartes-szorzatának részhalmaza/ szokás definiálni. Megemlitendő, hogy egy relációnak, és minden egyes oszlopának /mezőjének/ saját névvel kell rendelkeznie, melyre hivatkozni lehet /ld. 1. ábra "Dolgozó" relációja/.

Az adatszerkezet része az integritási feltétel. Ez valamilyen, az adatbázis adatainak állapotára, köztük lévő összefüggésekre utaló állitás, melynek az adatbázis konzisztens állapotában érvényesnek kell lennie. Néhány példa:

"Minden dolgozónak legfeljebb egy közvetlen főnöke van."

"A fizetések /időben/ nem csökkennek."

"Az intézet dolgozói alapbéreinek összege nem lehet több, mint az intézeti béralap."

Ezeket az állitásokat a felhasználónak kell megadnia valamilyen magas szintü nyelven, és a rendszernek automatikusan biztositania kell az állitások igazságát minden módositás esetén.

A jelenlegi gyakorlat általános integritási feltételek kezelésétől elég messze van [CODD 82]. Az INGRES rendszer egyváltozós állitások kezelésére /pl. fizetés < 30000 & fizetés > 2000/ képes. Az ORACLE Version 3. ezen kivül uj érték bevitelénél ellenőrzi, hogy az érték más reláció valamelyik sorában már létezik-e

/pl. ha a "dolgozó" relációban az "osztály" mező értéke "elektronika", az "osztályok" relációban ellenőrzi, hogy van-e ilyen nevü osztály/, és felujitási eljárások más felujitások automatikus elvégzését implikálhatják /pl. ha az "elektronika" osztályon dolgozó "N. Wiener" nevü dolgozót törli a felhasználó, a "dolgozó" relációból, akkor a rendszer az "osztály létszáma" mező értékét eggyel csökkenti az "osztályok" reláció megfelelő sorában/. Az IBM SQL/DS rendszere csupán annyit biztosit, hogy egyes mezőkben nem enged meg "nulla" értéket. [DIEC 81] .

Nem foglalkozunk a különféle relációs normálformákkal. Ezek lényegében véve a relációk épitésére vonatkozó javaslatok, azt célozzák, hogy a felhasználó "célszerüen" definiálja relációit /pl. csak egy dologra vonatkozzon a reláció, ne keveredjenek benne két különböző entitás tulajdonságai, mint ahogyan az 1. ábrán látható relációban a dolgozó adatai keverednek az osztályáéval/. Ezek a normálformák jobb szerkezetü adatbázis felépitést tesznek lehetővé, de egy relációs adatbázis implementálásához nem szükségesek [SAND 81] .

0.1.2. A relációkkal végezhető müveletek

A relációs modell egyik legrokonszenvesebb vonása, hogy viszonylag könnyen használható és "erős" felhasználói nyelveket lehet hozzá konstruálni. Napjainkra a manipulációs nyelvek száma elég sok tucatra becsülhető, és gyarapodásuk sebessége nem csökken. Ezeknek a nyelveknek az értékelése és összehasonlitása nem könnyü feladat.

A magas szintü nyelvek alapjául alapvetően két matematikailag megalapozott formális mechanizmus szolgál: a relációkalkulus és a relációalgebra. Mind a kettőt még Codd vezette be [CODD 70], [CODD 71a], [CODD 71c] . A tovább-

biakban egyikre sem adunk pontos definiciót - ez elég hosszadalmas és fárasztó - inkább példákkal illusztráljuk a formalizmusokkal specifikált lekérdezések stilusát. /Matematikailag preciz leirásuk megtalálható pl.[ДРИБ 82]-ben/. Egy lényeges vonásukat - amit a belőlük készült nyelvek is megőriztek - emeljük csak ki. A müveletek relációkon definiáltak, és eredményük is mindig reláció.

A relációalgebra müveletei egy vagy két relációt használva operandusként állitanak elő uj relációt. A fontosabb müveletek felsorolása következik:

Projekció /projection/: egy adott reláció egyes oszlopait kiválogatva, állit elő uj relációt, miután a duplikátumokat kidobta. Ha például valaki az 1. ábra "Dolgozó" relációjából csak az osztályok adataira kiváncsi, a

Dolgozó [Osztály neve,Osztály helye]

projekcióval kapja válaszként az 1. ábra "osztály" relációját

Osztály

Osztály neve	Osztály helye
Munkaügyi	I. em
Játékok	II. em.
Elektromos cikkek	II. em.
Textil	III. em.
.	
.	
.	

2. ábra

Korlátozás /Restriction, select/: A relációnak egy adott feltételt kielégitő sorait választja ki. A feltétel eredeti formájában [CODD 70] csak a sor két mezőjének összehasonlitása lehetett. Ez elégtelennek bizonyult, igy a definició kiegészült az egyes mező valamilyen konstanssal történő összehasonlithatóságával. Tehát például a

Dolgozó [Fizetés > 5000]

korlátozás az 1. ábra relációjának első és harmadik sorát adja eredményül, az ábrán látható többi sor kiesik.

Illesztés /join/: Két relációból /legyenek A és B/ készit egy harmadikat /C/ olyan módon, hogy A és B egy-egy sorát egymás mögé illeszti, ha a két sor egyes mezőire egy megadott feltétel teljesül. Ha például A reláció az 1. ábra relációja B pedig a

Főnök	Osztály neve	Osztályvezető neve
	Játékok	Barna József
	Munkaügyi	Kiss Pál
	Elektromos cikkek	Simon János
	Textil	Károly György
	...	...

3. ábra

reláció, a

Dolgozó [Dolgozó.Osztály neve=Főnök.Osztály neve] Főnök

illesztés eredménye az 1. ábra relációja lesz két uj oszloppal kiegészitve. Az egyikben minden dolgozónál az osztály-

vezetője neve szerepel - ami megegyezhet a saját nevével, ha ő az osztály vezetője, a másik, az "Osztály neve" oszlop, tehát minden sorban kétszer ismétlődik ugyanaz az osztálynév.

A relációalgebrának ezen kivül vannak halmazelméleti müveletei /egyesités, különbség, Descartes-szorzat, metszet/. Ezeket a halmazelméleti értelmükben kell venni, de alkalmazásuk persze értelemszerüen korlátozott /pl. két reláció csak akkor egyesithető, ha oszlopaik száma és azok értékkészletei megegyeznek, és az egyesités után - hogy reláció legyen az eredmény - a duplikátumokat el kell hagyni/.

A felsorolt müveleteken kivül egyéb müveletek is bevehetők a relációalgebrába, de már a felsoroltak is redundánsak, tehát egyesek levezethetőek a többiből /pl. az illesztés a Descartes-szorzatból és a korlátozásból./

A relációkalkulus nem más, mint az elsőrendü predikátumkalkulus felhasználása relációs lekérdezésekre. A változók a relációs sorokon vagy a reláció oszlopain /értékkészletein/ definiáltak [LACR 83] . A Codd-féle kalkulusból néhány példa:

Az osztályok adatai a "Dolgozó" relációból:

{D.Osztály neve, D.Osztály kódja: D ∈ Dolgozó}

Az 5000-nél többet kereső dolgozók adatai:

{D: D ∈ Dolgozó, D.Fizetés > 5000}

A "Dolgozó" és a "Főnök" reláció illesztése:

{D,F: D ∈ Dolgozó, F ∈ Főnök, D.Osztály neve=D.Osztályvezető neve}

A relációkalkulus szolgált alapul az első relációs lekérdező nyelvhez az ALPHA-hoz [CODD 71a] /ld. I.2.1./.

Sem a relációalgebra, sem pedig a relációkalkulus nem tartalmazza a müködő rendszerekben elterjedt, és igen hasznos aggregátum függvényeket /összeg, átlag, elemek száma/.

Ugyancsak hiányoznak belőlük a módositó /beillesztés, felujitás, törlés/ müveletek is.

A későbbiekben látni fogjuk, hogy a rendszerek által használt nyelvek általában valamilyen átmenetet képeznek az algebra és a kalkulus között - ez a felhasználó számára a legkényelmesebb. Ugy tünik, nehezen dönthető el, hogy a procedurálisabb algebra, vagy a kalkulus a kényelmesebb-e. Egy felmérés [WELT 81] arra az eredményre jutott, hogy az összetett kérdést könnyebb procedurális nyelven megfogalmazni /kb. 10 %-kal volt nagyobb a sikeres válaszok aránya/, az egyszerü kérdéseknél a pszichológusok nem találtak szignifikáns eltérést.

Lényeges elvi eredmény [CODD 71c] , hogy amilyen kérdés megfogalmazható relációalgebrával, az kalkulussal is megadható, és viszont, tehát a két nyelv kifejezőereje megegyezik. Bármilyen olyan nyelvet, melyben a relációalgebra vagy kalkulus müveletei megfogalmazhatóak relációsan teljesnek nevezünk.

0.2. Történeti áttekintés

Codd az alapvető [CODD 70] dolgozatot 1970-ben publikálta, és még ebben az évben az MIT projektet kezdeményezett relációs adatbáziskezelő rendszer készitésére /az igazsághoz tartozik, hogy Codd a dolgozat eredményeit már 1969-ben egy belső IBM anyagban /Research Report RJ 599/ közzétette azok számára, akik ilyenhez hozzájutnak/. A projekt eredményeképpen 1970-ben már müködött a MADAM rendszer. Ez PL/1-ben készült H6000 gépre, és a MULTICS virtuális memória lehetőségeit használta ki. A rendszer később /1971/ összeolvadt az ugyancsak MIT-s RDMS-sel. Lényeges motivum, hogy a MADAM/RDMS külön tárolja a relációs kapcsolatokat, és külön az adatmezők értékeit, oly

módon, hogy a relációkban minden adatot egy rögzitett hosszuságu azonositó reprezentál. [KIM 79]

1971-ben Codd három nevezetes dolgozatot közöl. [CODD 71a] az ALPHA nyelv leirása, [CODD 71b] inditja utjára a normálformák és a velük kapcsolatos matematikai kérdések lavináját, [CODD 71c] vezeti be a relációs teljesség /relational completeness/ fogalmát, megmutatva, hogy a relációkalkulus müveletei kifejezhetők a relációalgebra segitségével. A bizonyitás konstruktiv, és a lényege a következő redukciós algoritmus:

1. A kalkulus változói relációs sorokon vannak definiálva. Legyenek a megfelelő relációk $S_1, S_2, \ldots S_n$.
2. Képezzük a $D=S_1 \times S_2 \times \ldots \times S_n$ Descartes-szorzatot.
3. Távolitsuk el D-ből azokat a sorokat, melyek nem elégitik ki a predikátum feltételeit.
4. A kvantorok hatását fejezzük ki a relációs algebra müveleteivel, majd végezzük el ezeket a müveleteket D-n.
5. Projekcióval megkapjuk D-ből a kivánt relációt.

Maga az algoritmus gyakorlatilag nem megvalósitható, mivel a 2. és 3. lépések igen memória és időigényesek. Mégis az algoritmus jelentős, mert felhivja a figyelmet a relációs adatbázisok müködésének véleményünk szerint döntő fontosságu kérdésére: a keresési stratégia megválasztására. Egyébként az algoritmus tökéletesitett változata - a Palermo-algoritmus [ДРИБ 82] - müködő rendszer része [REBS 82].

Az IBM első relációs rendszere 1971-ben készült el Angliában IS/1, majd az uj változat 1973-tól PRTV /Peterlee Relational Test Vehicle/ néven. Ez relációs algebrát használ lekérdező nyelvként. Lényeges vonása, hogy az önálló nyelvhez a felhasználó PL/1 procedurákat illeszthet. Az adatokat süritve tárolja, és az ehhez szükséges kódolást

dekódolást mikroprogram végzi /a software megoldás tul lassunak bizonyult/. A rendszer keresési stratégiáját optimizáló program először átalakitja az algebrai kifejezéseket, majd megkisérli az optimális elérési ut kiválasztását. Az optimizálási algoritmus felhasználja [HALL 76] és [SMIT 75] /ld. 2.2/ dolgozatait. [KIM 79, TODD 76].

A másik, ebben az időszakban /1972/ kezdődő IBM projekt /ez Massachusetts-ben/ az RM, majd XRM rendszer. Az RM bináris, az XRM már tetszőleges fokszámu relációk tárolására alkalmas rendszer. A rendszer interface-e igen alacsony szintü, a felhasználónak belső azonositókkal kell dolgoznia. Hasonlóan a MADAM/RDMS-hez az XRM is külön-külön tárolja a relációkat és az adatmezők értékeit. A keresést index támogatja. [KIM 79, ASTR 75]

Az XRM-re épült 1974-ben a SEQUEL, 1976-ban a QBE /Query by Example/ rendszer. Az előbbi rendszer célja tulajdonképpen a SEQEL nyelv használhatóságának ellenőrzése, kisérleti implementáció volt, melynek ugy vágtak neki, hogy a rendszerből csak a tapasztalatokat viszik át az uj kisérleti rendszerbe - ez a System-R nevet kapta. [CHAM 81]. A Query by Example jellegzetessége a grafikus adatkezelő nyelv. [KIM 79]

A SEQUEL-ből kinövő uj kisérleti rendszert fejlesztették tovább az IBM jelenlegi "hivatalos" relációs adatbáziskezelő rendszerévé /SQL/DS/. A fejlesztési munka 1975-től egészen 1981-ig tarott. Az SQL/DS rendszer IBM nagygépeken DOS/VSE operációs rendzser alatt fut.

Az INGRES /Interactive Graphics and Retrieval System/ kisérleti vállalkozásnak indult 1973-ban a Berkeley egyetemen. Olyan sikeresnek bizonyult, hogy 1981-től bekerült az üzleti forgalomba. VAX-11 gépeken fut.

1979-ben jelentették be az ORACLE relációs adatbáziskezelő rendszert. A Relational Software Inc. készitette és

forgalmazza, eredetileg PDP-11 gépekre, 82-től azonban IBM-re is. Ez a rendszer lényegében véve a System-R projekt célkitüzéseinek megvalósitása, ugyanazt az adatkezelő nyelvet /SEQUEL 2 vagy SQL/ támogatja mint az, feltehetően a belső megoldások is hasonlóak.

A nem-mikrogépes relációs rendszerek közül tudományos és üzleti szempontból egyaránt a három utolsónak emlitett rendszer - SQL/DS, INGRES, ORACLE - tünik a legérdekesebbnek. Sajnos az ORACLE müködéséről nem sokat tudunk, a másik két rendszerre azonban a továbbiakban sürün fogunk hivatkozni. Az üzleti részről még: 1981 szeptemberében, az SQL/DS még nem volt forgalomban /1982 februárjától tervezték a termék kibocsátását/, az INGRES-nek 15, az ORACLE-nak 80 installálását jelnetették [DIEC 81].

1982-től indul meg a mikrogépes relációs adatbáziskezelő rendszerek áradata. Az 1981-ben forgalmazott adatbáziskezelők közül csak a CONDOR deklarálta magát relációsnak [BARL 81], ez is "korlátozott /limited/ relációs rendszer" a cikk szerzői szerint. 1983 februárjában Toulouseban már relációs rendszerek mikrogépes implementációjáról és felhasználásáról szervez az INRIA konferenciát [WORK 83]. Itt a csak az amerikai rendszereket áttekintő [MARY 83] dolgozat hét rendszert ir le /igaz ebből kettő biztos, hogy még a "minimálisan relációs" rendszerekkel szembeni [CODD 82] követelményeket sem elégiti ki, lévén, hogy nem tudja az illesztést/, de emliti, hogy ez csupán önkéntes kiemelés. A létező, és általunk ismert mikrogépes rendszerek közül a továbbiakban jónéhányra fogunk hivatkozni.

Nem emlitettük még ebben az igen felületes áttekintésben az adatbázis gépeket /data base machine/. Ezek relációs adatkezelést támogató speciális célgépek és a rájuk épülő rendszer. 1981-ben már három ilyen rendszert forgalmaztak [SNYD 82], tehát létező dolgok, de jelen tanulmány

ezekkel nem foglalkozik. Ugynacsak hasonló okból maradtak ki, az osztott, és a természetes nyelvü interface-t biztositó rendszerek is, noha mind a két témakör tekintélyes irodalommal, és müködő rendszerekkel /SDD-1, PLANES, stb./, rendelkezik.

1. A FELHASZNÁLÓI INTERFACE

Egy rendszer használhatósága szempontjából döntő fontosságu az a mód, ahogy a felhasználó a rendszert látja, az a forma, amelyben igényeit közölheti, és az eredményeket megkapja. A relációs adatbáziskezelő rendszerek felhasználói interface-eit áttekintve elég változatos a kép.

A legelterjedtebbek az önálló /stand alone/ nyelveken hozzáférhető rendszerek. Ezek a nyelvek interaktivak - a relációs rendszerek ad hoc kérdésekre adott válaszadó képességeit igy lehet kihasználni - de vannak olyan rendszerek, melyek batch lehetőségeket is támogatnak. A terminálon megjelenő információk, és a rendszerrel szembeni igények megfogalmazása általában szöveges, de léteznek ennél kényelmesebb - de legalábbis látványosabb - grafikus rendszerek is.

Az önálló nyelv mellett - rosszabb esetben - szükség van adatbázis hozzáférésre programozási nyelvből is. Sok rendszer biztositja ezt a lehetőséget oly módon, hogy egy programozási nyelv - gazdanyelv /host language/ - utasitáskészletét relációs adatbáziskezelő utasitásokkal egészíti ki. Kellemes, ha ezek az uj utasitások megegyeznek a rendelkezésre álló önálló, adatkezelő nyelv utasitáskészletével /vagy legalább annak részhalmazát alkotják/. [CODD 82] ezt a tulajdonságot "univerzálisan relációs"-nak nevezi, és több érvet hoz fel mellette. Léteznek persze rendszerek, ahol ez nincs meg - mikrogépen nem ismerünk olyan rendszert, mely univerzálisan relációs lenne - de sokszor szükséges legalább kettős - önálló és gazdanyelvből való - hozzáférés lehetősége, függetlenül attól, hogy a két nyelv megegyezik-e.

Igen érdekes az az irányzat, mely a gazdanyelvek és az adatkezelő nyelvek teljes összeolvadása mellet tör lándzsát. Itt lényegében véve arról van szó, hogy az absztrakt adattipus konstrukciókat támogató nyelvek minimális bővitéssel relációk kezelésére alkalmassá tehetők. Ezek a bővitések

szervesen illeszkednek bele "stilusukban" is a befogadó nyelvbe, fogalmaik annak fogalmaival keverednek, annyira, hogy nem lehet "gazdanyelvről" és "adatnyelvről" /data sublanguage/" beszélni, hanem uj egységes programozási-adatkezelő nyelv jön létre /PASCAL/R [SCHM 77, ALAG 81], MODULA/R [REIM 83]/. Ez persze azt is jelenti, hogy a megfelelő forditóprogramokat módositani kell, nem lehet a gazdanyelv - adatnyelv konstrukcióknál szokásos előforditási technikát alkalmazni [REIM 83].

Ebben a részben először az önálló nyelvek közül irunk le néhányat, előbb az adatdefiniciós lehetőségeket /adatbázis, reláció, index, stb. létrehozása/, majd a lekérdező és módositó nyelvet tárgyalva. Külön fejezet foglalkozik a magasszintü nyelvekből való adatkezeléssel, bemutatva a különböző irányzatokat.

Az egyes nyelvek használatát [CHAM 76] minta adatbázisán mutatjuk be. Ez a következő relációkat tartalmazza.

Dolgozó(Törzsszám, Név, Részlegkód, Besorolás,Főnök, Alapbér)
Részleg(Részlegkód, Részlegnév, Cim)
Felhasználás(Részelgkód, Cikkszám)
Szállitás(Szállitó, Cikkszám)

A Dolgozó és Részleg relációk egy vállalat dolgozóiról és részlegeiről, a Felhasználás reláció a részlegek által felhasznált anyagokról, a Szállitás pedig ezek szállitóiról tartalmaz adatokat.

1.1. Adatdefiniciós lehetőségek

Egy adatbáziskezelő rendszer használata mindig az adatok definiciójával kezdődik. A felhasználó /adatbázis adminisztrátor/ itt irja le az adatbázis szerkezetét, hogy milyen adatokat kiván használni, sokszor azt is, hogy milyen

módon. A logikai adatszerkezet mellett a fizikai tárolásra vonatkozó utmutatást is adhat a rendszernek - az egyes adatbáziskezelők adatfüggetlenségét jellemzi, hogy mennyi ilyen jellegü információt lehet, mennyit kötelező megadni, és ez milyen következményekkel jár majd az adatbázis létezése során.

Igen nagy előrelépés a CODASYL tipusu adatbázisokhoz képest, hogy a relációs rendszereknél az adatdefiniciós lehetőségek sokkal dinamikusabbak, és egy-egy döntésnek a következményei sokkal kisebbek. Mig a CODASYL-nál a séma változtatása igen nehézkes - általában teljes adatbázis ujraszervezést igényel - a relációs rendszerekben bármikor definiálható vagy törölhető egy reláció, vagy index. A CODASYL séma explicite irja le a fizikai elérési utakat, mig a relációs rendszerekhez a fizikai elérésre csak igen óvatos utalások - index szervezése, kapcsolat létrehozása - vannak, és ezek is legfeljebb valamilyen elérési lehetőség hatékonyságát, de nem a létezését befolyásolják.

1.1.1. SQL/DS

a/ Reláció létrehozása

A reláció nevét, és az oszlopok neveit, valamint tipusaikat kell megadni. A felhasználó megtilthatja egyes oszlopokban a nulla értéket. A Részleg reláció definiciója pl.:

```
CREATE TABLE RÉSZLEG
       (RÉSZLEGKÓD(CHAR(2),NONULL),
        RÉSZLEGNÉV(CHAR(12) VAR),
        CIM(CHAR(20) VAR))
```

Az SQL/DS valamennyi IBM/370 adattipust támogat [DIEC 81].

b/ Szinonima definiálása relációhoz

DEFINE SYNONYM ÜZEMEGYSÉG AS RÉSZLEG

c/ Index létrehozása

Az SQL/DS, mint általában a relációs rendszerek az elérés meggyorsitására indexeket használ. Ezeket ő képnek /image/ nevezi. Ezek - éppen ugy, mint a relációk - dinamikusan, bármikor, bármilyen oszlop /vagy oszlopkombináció/ szerint létrehozhatók, vagy törölhetők. Létezésük nem befolyásolja a reláció lekérdezhetőségét, ha valamilyen oszlop szerint nincs index, azért a relációból kiválasztható pl. az a sor, ahol az illető oszlopban 2568 áll. Mindebből adódik, hogy az "index" fogalmának bevezetése - noha fizikai szervezésre utal - nem csorbitja az adatfüggetlenséget.

Az index további utalást tartalmazhat a fizikai adatelhelyezésre. A CLUSTERING tulajdonság előirja a rendszernek, hogy az index definiálta sorrendben egymáshoz közel elhelyezkedő sorok az adatbázisban is egymás közelében helyezkedjenek el. /Érdekes lenne tudni, hogy ha egy létező relációhoz definiál az ember CLUSTERING indexet, csinál-e valamit a rendszer, tehát megmozgatja-e reláció már létező sorait. Valószinü, hogy a CLUSTERING csak a reláció jövőjére vonatkozik./

Az index másik tulajdonsága a UNIQUE lehet. Ez azt jelenti, hogy azok a sorelemek melyekre az indexet szervezzük a reláció kulcsai, vagyis nem lehet a relációnak két olyan sora, ahol ezeknek az elemeknek az értéke egyenlő.

A következő példa a Dolgozó relációra készit az Alapbér szerint indexet:

CREATE IMAGE FIZIND ON DOLGOZÓ (ALAPBÉR)

d/ Kapcsolat létrehozása

Szintén fizikai elérést meggyorsitó mechanizmus, melynek - elvben - nem lenne helye egy relációs nyelvben, de éppugy mint az index, ez sem jelent adatfüggőséget. A kapcsolat /link/ két relációnak azokat a sorait köti össze /pointerekkel/, ahol a kapcsolatot meghatározó adatmezőkben az adatok értéke megegyezik. Ez nyilvánvalóan az olyan illesztéseket gyorsitja meg, ahol az illesztés feltétele egyenlőség /o.l.2./. A kapcsolat is lehet CLUSTERING, ilyenkor a rendszer az összetartozó sorokat egymás közelében próbálja elhelyezni.

Kapcsoljuk össze a Dolgozó és a Részleg relációk sorait a Részlegkód alapján! A kapcsolat /nyilvánvaló 1:n lesz/ egy Részleg sorához tartozó Dolgozó sorai legyenek rendezve Besorolás és Fizetés szerint!

```
CREATE LINK L
FROM RÉSZLEG RÉSZLEGKÓD
TO DOLGOZÓ RÉSZLEGKÓD
ORDER BY BESOROLÁS FIZETÉS
```

e/ Nézőpont /view/ létrehozása

A nézőpont tulajdonképpen nem más, mint az adatbázis relációiból létrehozott uj reláció, amely létrehozása után éppen ugy használható /kérdezhető, felujitható, stb./, mint bármely másik reláció. A létrehozás tulajdonképpen lekérdezéssel történik. A lekérdezés eredménye - a relációs modellben természetesen /o.l.2./ - reláció, de ahelyett, hogy ez nyomtatás, vagy terminálra irás után elveszne, nevet kap és megőrződik.

Definiáljuk a Programozási Osztályt, mint részleget! /A lekérdezés formalizmusa 1.2.3.-ban található./

```
DEFINE VIEW PROGRAMOZÁSI_OSZTÁLY
  SELECT DOLGOZÓ.NEV,DOLGOZO.ALAPBER,RÉSZLEG.CIM
  FROM DOLGOZÓ,RÉSZLEG
  WHERE DOLGOZÓ.RÉSZLEGKOD=RÉSZLEG.RÉSZLEGKÓD
  AND DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'
```

Az uj relációnak - mert innentől a nézőpont annak számit - három oszlopa lesz, és a 'programozó' besorolásu dolgozók nevét, fizetését, munkahelyének cimét tartalmazza. Egyébként a rendszer fizikailag nem hozza létre az uj relációt, pusztán a nézőpont definicióját jegyzi meg, és a nézőpontra való hivatkozásoknál azt helyettesiti be a hivatkozás helyére.

f/ Reláció létrehozatala és feltöltése létező relációkból

Formálisan nagyon hasonlit a nézőponthoz:

```
ASSIGN TO PROGRAMOZÁSI_OSZTÁLY
  SELECT DOLGOZÓ.NÉV,DOLGOZÓ.ALAPBÉR,RÉSZLEG.CIM
  FROM DOLGOZÓ,RÉSZLEG
  WHERE DOLGOZÓ.RÉSZLEGKÓD=RÉSZLEG.RÉSZLEGKÓD
  ADN DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'
```

de lényegét tekintve más. Mig a nézőpont - noha önálló relációnak számit minden szempontból - a későbbiekben függeni fog azoktól a relációktól, melyekből létrejött /ha azok változnak ő is megfelelően változik, és viszont/ a feljebb lérehozott reláció teljesen független a létrehozó /Dolgozó,Részleg/ relációktól. Az uj reláció a két reláció megfelelő adatainak a létrehozás pillanatában rögzitett állapotát tükrözi, és függetlenül attól, hogy a két reláció adatai hogyan változnak a jövőben, ez az állapot nem módosul automatikusan. Itt tehát tényleg uj reláció

létrehozataláról és ennek adatokkal való feltöltéséről van szó, mig a nézőpont valóban az, amire a neve utal: egy adatbázis "ablak", melyen benézve tulajdonképpen nem uj relációt látunk, hanem a régi relációkat, csak speciális módon, másként válogatva, csoportositva. Az SQL/DS garantálja, hgoy a nézőpontok és a többi reláció konzisztens marad, de az ASSIGN-nal létrehozott reláció és a létrehozó relációk között minden kapcsolat megszakad. Itt nyilvánvalóan az uj reláció fizikailag is létrejön.

A nézőpont nyilvánvalóan "erősebb" lehetőség - ennek megfelelően a megvalósitása is nehezebb lehet.

g/ Reláció bővitése

Jó lehetőség az SQL/DS-ben, hogy szükség esetén, létező relációk uj oszlopokkal bővithetők. Az uj oszlopok a létrehozás pillanatától /"definiálatlan mező" értékü adatmezőkkel/ léteznek. A mezők a szokásos módositó utasitásokkal kaphatnak értéket.

Kiegészitjük a Részleg relációt a Létszám oszloppal:

```
EXPAND TABLE RÉSZLEG
  ADD COLUMN LÉTSZÁM(INTEGER)
```

h/ Reláció, nézőpont, index, kapcsolat megszüntetése

Ez is bármikor megtehető, pl.

```
DROP VIEW PROGRAMOZÁSI_OSZTÁLY
```

h/ Megjegyzés

```
COMMENT ON THE VIEW PROGRAMOZÁSI_OSZTÁLY:
  'EZ NEM IS EGY OSZTÁLY, CSAK AZ ÖSSZES PROGRAMOZÓ
   BESOROLÁSU DOLGOZÓ HALMAZA'
```

[CHAM 76]

1.1.2. INGRES

Az utasitások tartalmukban nagyon hasonlóak az SQL/DS-éihez, ezért általában nem magyarázzuk őket, legfeljebb azokat, amelyek nincsenek meg, vagy másképp vannak itt mint ez SQL/DS-ben.

a/ Reláció létrehozása, törlése

```
CREATE RÉSZLEG(RÉSZLEGKÓD IS CHAR(2), RÉSZLEGNÉV IS
                CHAR(12), CIM IS CHAR(20))
DESTROY RÉSZLEG
```

/Nincsenek változó hosszuságu karaktersorozatok, 1,2,4 byte fixpontos, 4 és 8 byte lebegőpontos számok, és max. 255 byte hosszu karaktersorozatok vannak./

b/ Reláció másolása

Ez vagy létező reláció adatait irja ki adatfile-ba, vagy az adatfile-ból tölt fel egy relációt adatokkal.

```
COPY RÉSZLEG(RÉSZLEGNÉV IS CHAR(40), RÉSZLEGKÓD IS CHAR(10))
    FROM RÉSZLEGEK
```

Az utasitás a Részleg relációt feltölti a Részlegek nevü file-on található adatokkal. A file logikai rekordjának felépitése a formátumlistán látható: a részlegnév 40, majd utána a részlegkód 10 karakter hosszu. A rendszer felhasználva a Részleg reláció definicióját elvégzi a szükséges konverziót, és feltölti a reláció két oszlopát.

Reláció adatfile-ra irása ugyanigy történik 'FROM' helyett 'TO' kulcsszót használva. Speciális adatfile-ra irás a reláció terminálra irása, pl.:

```
PRINT RÉSZLEG
```

c/ Tárolási struktura változtatása

Ez lényegében véve index definiálását, ill. a fizikai tárolás előirását jelenti. A fizikai tárolást 2.3.2-ben irjuk le, itt csak annyit emlitünk meg, hogy ezek az utasitások éppen ugy nem jelentenek fizikai adatfüggőséget az INGRES számára, mint a kép ill. kapcsolat az SQL/DS-nél. Megjegyezzük még, hogy a relációt létrehozó utasitásban nem történik utalás a szervezési módra. [DIEC 81]-ből kiderül, hogy a rendszer a legegyszerübb tárolási módra a heap-re - rendezetlen, soros elérésü file - készül fel, és a tárolási struktura változatásával adható meg a kivánt tárolás.

Legyen Részleg reláció index-szekvenciális file-ként tárolva a Részlegkód szerint!

```
MODIFY    RÉSZLEG TO ISAM ON(RÉSZLEGKÓD),
```

és csináljunk hozzá a Részlegnév szerint egy indexet!

```
INDEX ON RÉSZLEG IS NÉVIND(RÉSZLEGNÉV)
```

Az INGRES egyébként az indexet közönséges relációként kezeli, melynek oszlopai a relációnak azok az oszlopai, amelyek szerint az index készült /jelen esetben a részlegnév/ és plusz még egy oszlopa a reláció sorainak azonositóját tartalmazza.

d/ Reláció feltöltése létező relációkból

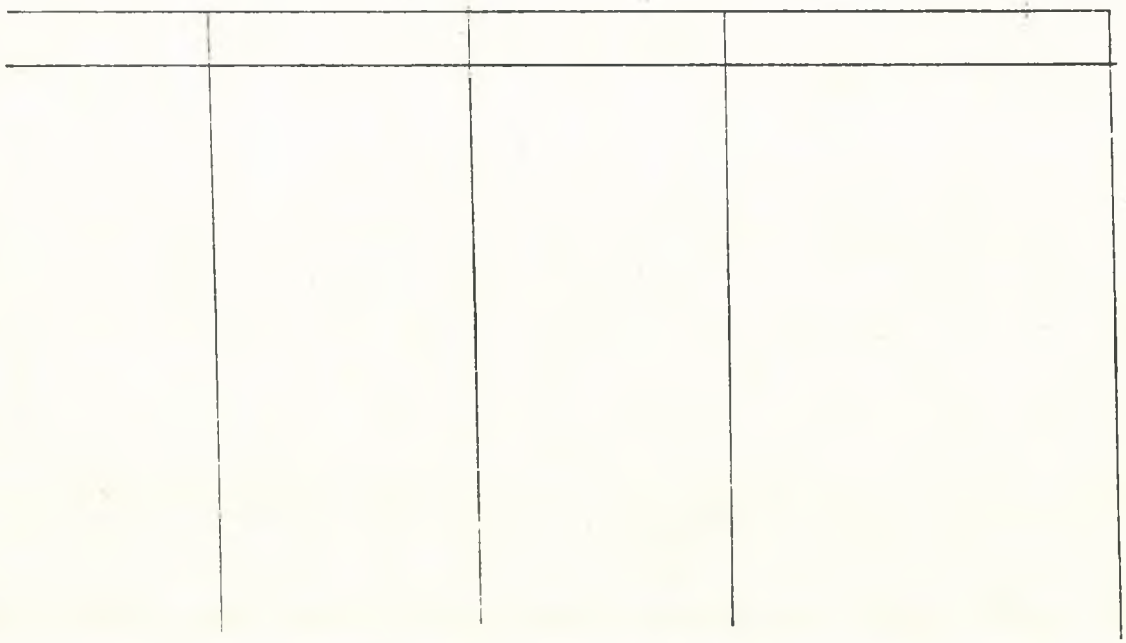
Ugyanaz , mint az SQL/DS-ben.
Például az 1.1.1 f-ben definiált Programozási Osztály itt is megadható: /miután egy CREATE definiálta/

```
RANGE OF D IS DOLGOZÓ
RANGE OF R IS RÉSZLEG
RETRIEVE INTO PROGRAMOZÁSI_OSZTÁLY
        D.NÉV, D.ALAPBÉR, R.CIM
        WHERE D.RÉSZLEGKÓD=R.RÉSZLEGKÓD
        AND D.BESOROLÁS='PROGRAMOZÓ'
```

[STON 76, EPST 77]

1.1.3. QBE

A QBE /Query By Example/ rendszer grafikus interface-t biztosit a felhasználó számráa. A reláció táblázat, és ennek a táblázatnak a "csontvázába" /ld. 4. ábra/ helyezi be a szöveges információt a felhasználó, ha igényel valamit a rendszertől, ill. a rendszer válaszadásnál.



4. ábra

a/ Reláció definiálása

A felhasználó kitölti a képernyőn megjelenő "csontváz"-at, pl. igy

RÉSZLEG	RÉSZLEGKÓD	RÉSZLEGNÉV	CIM
	CHAR 2	CHAR 12	CHAR 20

5. ábra

/Jellemző a QBE-re, hogy a gyakorlatlan felhasználó megkérheti a rendszert, hogy irja ki, hogy milyen adatokat kell megadnia. Miután a reláció nevét is az oszlopokét megadta, a relációnév - jelen esetben "Részleg" - alá irt"P." /print/ hatására megjelenik a

RÉSZLEG	RÉSZLEGKÓD	RÉSZLEGNÉV	CIM
TYPE			
LENGTH			
KEY			
DOMAIN			
SYSNULL			

6. ábra

táblázat/.

b/ Reláció bővitése

Szintén grafikusan történik, lényegében ugy, mint a definiálás. A rendszer a felhasználó kivánságára kiirja a Részlegtábla definicióját, az első oszlopot a 6. a többit pedig az 5. ábrán látható módon. A felhasználó az üres oszlopot tölti ki, ugy mint reláció definiálásakor. Az uj oszlop adatai felujitásukig "definiálatlan mező" /sysnull/ értékkel birnak.

c/ Reláció törlése, relációnév változtatása

A lekérdezett táblába /pl. 6. ábra/ a reláció neve elé "D." ill "U." irásával. Névváltoztatás esetén a régi nevet át kell irni uj névvé. Oszlop törlése ugyancsak

az oszlopnév elé irt "D."-tal, névváltoztatás "U."-tal történik.

d/ Reláció feltöltése létező relációkból

Ugyanaz a funkciója, mint az INGRES esetében /1.1.2.d/ A formalizmusának lényege - egy lekérdezés eredményét tároló uj relációként - is ugyanaz, de persze mindez QBE-ben. A felhasználó itt három táblával dolgozik. A Dolgozó tábla:

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	...	ALAPBÉR
		XXX	N1	PROGRAMOZÓ		5R

A részleg tábla:

RÉSZLEG	RÉSZLEGKÓD	RÉSZLEGNÉV	CIM
	N1		AB

Az uj programozási Osztály tábla:

PROGRAMOZÁSI_OSZTÁLY	NÉV	ALAPBÉR	CIM
	XXX	5 R	AB

Az aláhuzott nevek változók. Ezeknél nem a konkrét név számit, hanem az, hogy hol jelenik meg. Pl. A Dolgozó tábla "Név" oszlopában az "XXX" helyett állhatna "KISS PÁL" is, a fontos az, hogy a Programozási Osztály tábla Név oszlopában ugyanaz álljon, jelezve, hogy a lekérdezés eredményeként kapott akármilyen nevet - legyen az "XXX" vagy "KISS PÁL",

vagy bármi más - kell az uj tábla Név rovatába irni. Ugyanaz a helyzet az "5 R", "AB" változókkal is. Az N1 segédváltozó, a feltétel megfogalmazásához, a Dolgozó és Részleg táblák illesztéséhez szükséges /ld. 1.1.2.d feltételének első tagját/. A 'Programozó' konstans jelzi, hogy csak azokat a sorokat kell kiválogatni, ahol a besorolás 'programozó' /ld. 1.1.2.d. feltételének második tagját/. /A lekérdező nyelv leirása megtalálható 1.2.5-ben/.

e/ Nézőpont definiálása

A nézőpont a QBE-ben ugyanazt jelenti, és ugyanugy viselkedik, mint az SQL/DS nézőpontja /1.1.1e és 1.1.1.f/. Definiálása a d-ben leirt módon történik, de a "Programozási Osztály" név elé odairandó a "VIEW" kulcsszó.[ZLOO 77]

1.1.4 Általános áttekintés. Mikrogépes rendszerek

a/ Reláció létrehozása

Ez a rendszereknél gyakorlatilag egyformán történik. Alapkövetelmény, hogy bármikor lehessen uj relációt definiálni /legalábbis a dolgozatok szerzői sehol nem emlitik, hogy előre rögzitett relációs sémával dolgoznának.

[EBER_83] dolgozat egy TRS 80/I-re készült rendszert ir le. Ez csak karaktersorozat tipusu adatokat enged meg, de ezek a rendszerbe való bekerüléskor automatikusan ellenőrizhetők a következő módokon:

Formátumlista: legegyszerübb példán keresztül megmutatni. Ha valamilyen oszlopban szerepelhető adatok formátumaként "AB %$$"-t adunk meg, a rendszer csak olyan adatokat enged meg az oszlopban, melyek első két karaktere 'AB' utánuk számjegy /0-9/ következik, a két utolsó karakter tetszőleges.

Megszámlálható, vagy intervallum adattipus. A programozási nyelvekből átvett ötlet: előbbinél fel kell sorolni a megengedett adatértékeket, utóbbinál az intervallum /lexikografikusan/ alsó és felső határát.

A rendszerbe kerülő értékek ellenőriztetése elég könnyen kivitelezhető, hasznos gondolat, hagyományos filekezelő rendszerekben elég gyakori /főleg mikrogépeken/.

[KAMB 83] egy igen erős séma definiciós és restrukturálási lehetőségekkel biró rendszert ismertet. Itt egy-egy oszlop definiálásánál a szokásos adatok mellett megadható kétféle null érték /"nem létezik" "nem tudom" null/ megengedése, vagy meg nem engedése is. A rendszer megengedi a halmaz értékü oszlopokat, vagyis azokat ahol egy adatmezőben egyszerre több érték is szerepel. /Például a "Könyv" relációnak "Szerző" oszlopában szerepelhet értéknek a (Aho, Ullman) pár, és lekérdezésnél a megfelelő sor ugy a Szerző='Aho', mint a Szerző='Ullman' feltételnek eleget fog tenni./
A reláció definiálásakor az oszlopok, mint halmazok között 1:1 és 1:n kapcsolatok irhatók elő.

b/ Relációk átszervezése

[KAMB 83] dolgozat ismertetését folytatjuk. A szokásos lehetőségek /uj oszlopok beillesztése, létezők törlése/ megengedi

- az oszlop jellemzőinek /tipus, hossz/ változtatását;
- az a/ pontban ismertetett attributumok /null-érték, 1:n kapcsolatok, stb./ változtatását;
- egy oszlop több részre vágását, vagy több oszlop egyesitését /pl. a "Dátum" oszlop szétvágható "Év", "Hónap" és "Nap"-ra/ és még egy sor más lehetőséget.

A rendszerrel kapcsolatban meg kell jegyeznünk, hogy

a példák alapján ugy tünik, hogy könyvtári nyilvántartásra használják, ezért szükséges a séma fantasztikus rugalmassága. Egyébként Z-80 alapu japán gépen fut CP/M alatt 10 MB-os Winchester lemezt használ. A rugalmas adatdefiniciós és átstrukturálási lehetőséget elég speciális belső szervezéssel éri el.

c/ Index létrehozása

Az SQL/DS és INGRES esetében ez explicite, a felhasználó utasitására történik /1.1.1.c és 1.1.2c/. A QBE-nél erre nem találtunk utalást, gyanitjuk, hogy ugy történik, mint az a/ pontban már emlitett [EBER 83] leirta rendszernél. Ott ugyanis a felhasználó a reláció definiálásakor kijelöli azokat a mezőket, melyek a reláció kulcsai /ezt a fogalmat [CODD 70] vezette be, itt egyszerüen mint közvetlen elérési - nem emlitik, hogy azonositási - lehetőséget használják/. A kijelölt kulcsokra készit ezután a rendszer automatikusan indexet.

Ennek a módszernek hátránya /lehet/ az indexek statikus jellege. Hacsak a rendszer nem engedi meg a kulcsok ujradefiniálását menet közben /ez elég valószinütlen/, akkor az indexek, s velük együtt a hatékony közvetlen hozzáférés utvonalai egyszer s mindenkorra ki vannak jelölve.

Vannak rendszerek, ahol azért nincs indexet létrehozó utasitás, mert nincs index. Ilyen például a VIDEBAS [BLAN 83], mely igen sajátos implementáció - minden adatmező szerint lerendezi a relációt, és több példányban, index-szekvenciális file-okként tárolja. Ennél a rendszernél /DEC-system 10-en fut természetesen nagy lemezekkel/ nincs szükség indexre. Az RQL esetében [MASR 83] az indexek hiánya elég lassu müködéshez /több mint 1 órás várakozási idők/ vezet /viszont 48 K-s APPLE II-n fut/.

Három változattal találkoztunk tehát:

- explicit indexdefiniálás /SQL/DS/;
- implicit indexdefiniálás /[EBER 83]/;
- nincs indexdefiniálás, mert nincs index.

Elképzelhetőnek tartanánk egy másik, elég kényelmes és hatékony /de vajon könnyen megvalósitható?/ változatot. A . rendszer maga döntene indexek létrehozásáról, megszüntetéséről a saját statisztikái alapján. Ily módon, valamiféle stratégia szerint annak jóságától függően előbb vagy utóbb el lehetne jutni oda, hogy a sürün használt adatokra létezne index. Az explicit indexdefiniálás majdnem ezt csinálja, csak ő a statisztikát a felhasználóval /adatbázis adminisztrátor/ vezetteti.

d/ Reláció másolása külső file-ról/ra

Elég könnyü lehet megvalósitani, nyilvánvalóan szükséges müvelet, tehát eléggé elterjedt, ld. pl. a PRTV "relációs file"-ja [TODD 76]. A mikrogépeken ritkábban fordul elő, feltehetően a viszonylag kevés tárolható adatot terminálról is be lehet vinni, és mivel nincs multiprogramozás nem érdemes külön előkésziteni az adatokat.

e/ Reláció feltöltése más relációkból

Ugy tünik ez bonyolultabb ügy, mint az előző, nem elterjedt lehetőség. [KAMB 83] rendszere nem uj relációt szervez más relációkból, hanem átstrukturálja a rendszert, a generáló /feltöltő/ relációk eltünnek, és az uj reláció veszi át helyüket.

f/ Kapcsolat, nézőpont definiálása

Előbbivel csak az SQL/DS-nél, utóbbinál pedig az SQL/DS-

nél és a QBE-nél találkoztunk.

A kapcsolat igen hatékonyá tud tenni egyes müveleteket /illesztések egyenlőség alapján/ de megvalósitása körülményes pl. egy olyan rendszerben, ahol a sorok B-fában helyezkednek el, fizikai elhelyezkedésük szüntelenül változik, igy csak szimbolikus pointer vagy indirekt cimzés alkalmazható. Ez még a kisebbik baj, de az amugy is igen bonyolult optimizálási algoritmusokba /2.2/ nehéz beilleszteni a kapcsolatot - legalábbis erre gondolunk.

A nézőpont megvalósitása sem lehet egyszerü. A felujitása körül bonyolult esetekben logikai problémák merülnek fel. Egyébként [CHAM 76] a nézőpont felujitásával kapcsolatosan korlátozásokat is emleget - tehát a felhasználó komolyabb bonyodalmakba is keveredhet. Egyébként [DIEC 81] sem az SQL/DS sem pedig az ORACLE lehetőségei között nem emliti a kapcsolat és a nézőpont definiálását.

g/ A relációs modell kiterjesztése

Izgalmas kisérlet ebben az irányban a LIDAS rendszerre épülő interaktiv adatdefiniciós interface, a GAMBIT [BRAG 83, REBS 83]. Entitásokkal és kapcsolatokkal [CHEN 76] dolgozik - ezt az irányzatot már [CODD 79] sürgette - és erős szemantikai kifejezőerővel bir.

A rendszer grafikus technikát használ, de a relációs szimbolumok helyett /táblázatok/, az entitás-kapcsolat modell dobozait /ez jelképez egy entitást/ és összekötő vonalait /az entitások közötti kapcsolat/ használja. Ez az első lépés tehát, az entitások és ezek egymás közötti kapcsolatainak megadása.

Második lépésként az entitásoknak, mint relációknak a megadására kerül sor. Először a szemantikailag fontosabb oszlopokat jelöli meg a felhasználó - egy oszlop fontosságát

az jelzi, hogy több entitásban is szerepel ilyen módon tartva fenn a közöttük fennálló kapcsolatot /globálisnak nevezi őket a GAMBIT/ - majd a többi, lokális jellegü adatot. Mindez persze grafikus segitséggel.

A harmadik lépésben integritási feltételek adhatók meg /1:n kapcsolat, kulcs tulajdonság,stb./. Ezek eléggé bonyolultak is lehetnek, ugyanis a befejező lépés az adattipus kialakitásában a müveletek /tranzakciók/ megadása, és ezzel a lépéssel valamennyi definiált entitás és kapcsolat a rajtuk definiált müveletekkel együtt MODULA/R absztrakt adattipussá válik. Szó szerint ugyanis a GAMBIT az adatdefinició befejezése után MODULA/R /1.3.2/ programot generál, és - forditás után - ez lesz a továbbiakban a futó program.

Nagyon rokonszenves vonásai a rendszernek a szemantikaorientáltság, és az, hogy az adatbázis tervezéséhez nyujtott software támogatás valósággal kényszeriti a felhasználót a lépésenkénti finomitás /stepwise refinement/ célszerü tervezési módszerére. Ugy tünik viszont, hogy ezekért azzal fizet, hogy definiált sémája statikus lesz. Egy már feltöltött adatbázis szerkezetét csak ugy lehet módositani /reláció oszlopainak változtatása, szemantikus összefüggések változtatása, stb./, ha ujradefiniáljuk az egészet, ami uj MODULA/R programot generál és ez - legalábbis ugy gondoljuk, nem találtunk rá utalást - nem képes a régi adatbázison futni, ujra kell szervezni azt.

1.2. Lekérdezés, módositás

Az adatdefinició vagy kizárólag az adatbázis adminisztrátor feladata, vagy ha mások is definiálhatnak uj adatokat, az ritkán történik meg, egy "mezei user" nem kell, hogy jól ismerje az adatdefiniciós lehetőségeket. Ezzel szemben

a lekérdező, módosító nyelv az, melyen a felhasználók széles köre naponta az adatbázishoz fordul. Lényeges szempont az ilyen nyelvek tervezésénél a felhasználóhoz való alkalmazkodás /user friendliness/.

Először a két klasszikus nyelvet, a relációkalkulust és a relációalgebrát ismertetjük, majd sorba vesszük az SQL/DS, az INGRES és a QBE rendszereket, végül külön paragrafusban a többieket. A nyelvek bemutatásánál nem törekedhetünk teljességre, csak lehetőségeket és a stilust próbáljuk érzékeltetni néhány példán keresztül.

Valamennyi itt szereplő relációs nyelvnek van egy kiemelkedően fontos, a többi adatkezelő nyelvtől megkülönböztető vonása: egy nyelvi utasitás egy vagy több teljes relációval dolgozik, a müveletek eredménye pedig mindig egy teljes reláció. Ez eltér a relációs modell előtti adatkezelés "egy utasitás egy logikai rekord" elvétől.

1.2.1. Relációkalkulus – ALPHA

Az ALPHA az első relációkalkulus /0.1.2/ alapu nyelv. Szerzője Codd mint gazdanyelvbe beépülő adatkezelő nyelvet / sublanguage/ definiálta [CODD 71a], de jellege /egy-egy utasitás nem sort, hanem teljes relációt dolgoz fel/ igazából nem olyan. A gazdanyelvre utal viszont a "munkaterület" fogalma: Ezen a területen kommunikál a felhasználó az adatbázissal. Itt kapja meg egy-egy lekérdezés eredményét, itt végzi az adatbázis módositását, a beillesztéseket. Persze a munkaterület tulajdonképpen a többi, már egyértelmüen önálló relációs nyelvben is teremthető amikor szükség van rá, csak más formában /1.1.1.f, 1.1.2.d, 1.1.3.d/.

a/Egyszerü lekérdezés

Válasszuk ki azokat a részlegeket, ahol programozók

programtervezők dolgoznak!

```
GET W (DOLGOZÓ.RÉSZLEGKÓD):
        DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ' V
        DOLGOZÓ.BESOROLÁS='PROGRAMTERVEZŐ'
```

A Dolgozó reláció olvasását a GET utasitás végzi, mely a W munkaterületre azoknak a soroknak Részlegkód elemét adja át, melyre a ":" után álló feltétel teljesül. A müveletet nem elég ugy elképzelni, hogy a GET soronként olvas, és a megfelelő sorok részlegkódjait szépen egymás után W-be irja: ugyanis miután mindezt megtette, még kiszüri a duplikátumokat is - összhangban a reláció definiciójával /O.1.1./. Megtehettük volna, hogy a kódokat nagyság szerint rendezve kérjük. Ehhez mindössze az utasitás végére kell biggyeszteni az

```
UP DOLGOZÓ.RÉSZLEGKÓD
```

sort.

b/ Beépitett függvény

Noha a relációkalkulus nem tartalmaz beépitett függvényt, már Codd felismerte ezek szükségességét, és az ALPHA-ba beillesztette őket. Ezek egy reláció sorainak számát, egy oszlopban szereplő különböző értékek számát, egy oszlop összegét, átlagát, minimális, maximális elemének nagyságát, stb. adják vissza.

Hány programozónak van 4000 forintnál nagyobb alapbére?

```
GET W(COUNT(DOLGOZÓ.TÖRZSSZÁM)):
        DOLGOZÓ.ALAPBÉR > 4000
        DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'
```

c/ Összetett /több relációt érintő/ lekérdezés

Keressük ki minden programozó nevét, alapbérét és a részlege cimét!

```
GET PROGRAMOZÁSI_OSZTÁLY(DOLGOZÓ.NÉV,DOLGOZÓ.ALAPBÉR
                                        RÉSZLEG.CIM):
                DOLGOZÓ.RÉSZLEGKÓD=RÉSZLEG.RÉSZLEGKÓD
                DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'
```

A specifikált adatok most a beszédesebb PROGRAMOZÁSI_OSZTÁLY munkaterületre kerülnek. A két reláció /Dolgozó, Részleg/ közötti kapcsolatot közös adatuk, a Részlegkód teremti meg. A GET minden programozónál megkeresi a megfelelő Részlegkódu Részleg sort /ha a Részlegkód nem azonositó, akkor sorokat/, és abból illeszti hozzá a cimet a dolgozó adataihoz igy állitva elő a kivánt relációs sort /ha a Részlegkód nem azonositó, sorokat/. Relációalgebrai nyelven a Részleg és a Dolgozó relációt illesztjük össze, és a

```
DOLGOZÓ.RÉSZLEGKÓD=RÉSZLEG.RÉSZLEGKÓD
```

pusztán azért szerepel, mert az az illesztés feltétele.

Keressük ki azokat a dolgozókat, akik alapbére nagyobb, mint a főnöküké!

```
RANGE DOLGOZÓ VEZETŐ
GET W(DOLGOZÓ.NÉV,VEZETŐ.NÉV):
      DOLGOZÓ.FŐNÖK=VEZETŐ.TÖRZSSZÁM
      DOLGOZÓ.ALAPBÉR>VEZETŐ.ALAPBÉR
```

Az első utasitás definiálja a Dolgozó reláció sorain értelmezett Vezető változót. Ezt tulajdonképpen ugy képzelhetjük el, mint a Dolgozó "reláció" másolatát, vagy mint

a Dolgozó sorain futó cursor-t. A GET utasitás első feltétele nem más, mint a Dolgozó reláció önmagával - illetve Vezető nevü tükörképével való illesztésének feltétele. Minden Dolgozó sorhoz kikeressük a főnöke sorát /nyilvánvalóan az a sor lesz, amelyben a Törzsszám megegyezik a Dolgozó sor Főnök elemével/. Miután ez megvan, ellenőrizzük a két sorban, hogy a dolgozó fizetése nagyobb-e mint a főnöké, és ha igen a dolgozó és főnöke neve a munkaterületre kerül.

d/ Lekérdezés csoportositott adatok szerint

Keressük meg azokat a részlegeket, melyekben 10-nél több programozó dolgozik!

A feladatot két lépésben oldjuk meg. Első lépésként kiválogatjuk a Dolgozó relációból a programozók törzsszámait és részlegkódjait.

```
GET W(DOLGOZÓ.TÖRZSSZÁM,DOLGOZÓ.RÉSZLEGKÓD):
    DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'
```

A munkaterületen lévő relációra az ICOUNT csoportszámláló beépitett függvényt fogjuk használni. Az ICOUNT(R,A,B) első paramétere az az R reláció, amin használjuk, és müködése abból áll, hogy a második paramétereként megadott oszlopnév A minden rögzitett értékére kikeresi a hozzá tartozó valamennyi különböző B /a reláció egy másik oszlopának neve/ értéket. Az

```
ICOUNT(PROGRAMOZÓ,RÉSZLEGKÓD,TÖRZSSZÁM)
```

beépitett függvény tehát /ha a Programozó reláció valóban a programozókat tartalmazza/ éppen azt adja vissza amire szükségünk van: az egy részlegben dolgozó programozók számát. Mivel W-ben az előző válogatás eredményeként éppen a programozók adatai vannak, igy a

```
RANGE W PROGRAMOZÓ
GET W1(PROGRAMOZÓ.RÉSZLEGKÓD):
    ICOUNT(PROGRAMOZÓ,RÉSZLEGKÓD,TÖRZSSZÁM) > 10
```

lekérdezés éppen a 10-nél több programozót alkalmazó részlegek kódjait /duplikátumok nélkül természetesen!/ helyezi el W1 munkaterületen.

e/ "Az Összes" tipusu lekérdezés

Keressük meg az olyan szállitókat, akik az 50 kódu részleg által felhasznált összes cikkszámot szállitják!

Ismét két lépésben oldjuk meg a feladatot. Először kiválogatjuk azokat a cikkszámokat, melyeket az 50 kódu részleg használ.

```
GET W(FELHASZNÁLÁS.CIKKSZÁM): FELHASZNÁLÁS.RÉSZLEGKÓD=50
```

Most tehát az olyan szállitókat kell kiirnunk, melyekre az összes W-ből vett cikkszámra létezik olyan Szállitás sor, ahol éppen ő a Szállitó. Mivel a relációkalkulus megengedi a kvantorok használatát ez igy irható:

```
RANGE SZÁLLITÁS LÉTEZŐ
RANGE W MINDEN
GET W1(SZÁLLITÁS.SZÁLLITÓ):
   ∀ MINDEN ∃ LÉTEZŐ:(MINDEN.CIKKSZÁM=LÉTEZŐ.CIKKSZÁM
           SZÁLLITÁS.SZÁLLITÓ=LÉTEZŐ.SZÁLLITÓ)
```

f/ Uj sor illesztése relációba

Meglehetősen egyszerüen megy: a megfelelően strukturált W munkaterületen kapnak az egyes sorelemek értéket, majd a PUT utasitás irja be az uj sort a relációba:

```
W.TÖRZSSZÁM=9286
W.NÉV='KISS PÁL'
W.RÉSZLEGKÓD=52
PUT W(DOLGOZÓ)
```

g/ Lekérdezés eredményének illesztése relációba

Illesszük be a "Programozási Osztály" nevü relációba az összes programozó nevét, alapbérét és annak a részlegnek a cimét ahol dolgozik!

```
GET W(DOLGOZÓ.NÉV,DOLGOZÓ.ALAPBÉR,RÉSZLEG.CIM):
      DOLGOZÓ.RÉSZLEGKÓD=RÉSZLEG.RÉSZLEGKÓD
      DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'
PUT W(DOLGOZÓ)
```

A GET W-ben összegyüjti a kivánt adatokat /ld, c//, a PUT pedig a relációba illeszti őket.

h/ Egy sor törlése

Töröljük az 5618 törzszsámu dolgozót!

```
HOLD W(DOLGOZÓ): DOLGOZÓ.TÖRZSSZÁM=5618
DELETE W
```

A HOLD utasitás ugyanazt csinálja, mint a GET, csak egyben figyelmezteti a rendszert, hogy módositás következik /megfelel a modern rendszerek "locking"-jának is/. /Mellesleg az, hogy a HOLD-ra szükség van, az a nyelv alkotóinak realizációs elképzeléseiről is árulkodik: ha nem az utasitások szekvenciálisan egymás után következő végrehajtásával müködik a rendszer nincs szükség HOLD-ra /ld. pl. SQL/DS, 1.2.3./. A DELETE a munkaterületen lévő sorokat törli.

i/ Összetett lekérdezés eredményének törlése

Töröljük azokat a részlegeket, melyek egyetlen dolgozót sem alkalmaznak!

```
RANGE DOLGOZÓ D
HOLD W(RÉSZLEG):
    ¬∃D:(D.RÉSZLEGKÓD=RÉSZLEG.RÉSZLEGKÓD)
DELETE W
```

j/ Felujitás

Adjunk minden programozónak 10 % béremelést!

```
HOLD W(DOLGOZÓ): DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'
ALAPBÉR=1.1 * ALAPBÉR
UPDATE W
```

A keresést a HOLD ugyanugy végzi, mint mindig. Az értékadás az oszlop valamennyi elemére vonatkozik /akár a PL/1 tömbmüvelete/. A felujitás az UPDATE hatására következik be. [DATE 77]

1.2.2. Relációalgebra

A most ismertetésre kerülő nyelv [DATE 77]-ből származik. Nem egyezik meg pontosan semelyik relációalgebrát használó rendszer nyelvével, de az eltérések pusztán szintaktikai jellegüek, valamennyi ilyen nyelv ugyanazokat a müveleteket használja, melyek közül a szokásos halmazelméleti müveleteken kivül a projekció, a korlátozás és az illesztés /O.1.2/ bir számunkra jelentőséggel. Codd nevezetes eredménye [CODD 72c] szerint már ezek a müveletek garantálják a relációs teljességet /O.1.2./.

A relációalgebrában a relációkalkulus "munkaterület"-ének nincs pontos megfelelője, viszont egy művelet eredményeként előálló reláció a továbbiakban felhasználható, azzal további müveletek végezhetők. Megjegyezzük, hogy egy több lépésből álló müveletsor zárójelek alkalmazásával egyetlen, bár több müveletből álló lépéssé alakitható. Mi itt az érthetőség érdekében általában több lépésben hajtunk végre mindent .

a/ Egyszerü lekérdezés

Válasszuk ki azokat a részlegeket, ahol programozók vagy programtervezők dolgoznak!

```
SELECT DOLGOZÓ WHERE BESOROLÁS='PROGRAMOZÓ' UNION
SELECT DOLGOZÓ WHERE BESOROLÁS='PROGRAMTERVEZŐ GIVING T
PROJECT T OVER RÉSZLEGKÓD GIVING EREDMÉNY
```

Az első lépés T relációt állitja elő. Ez két reláció egyesitése. Mind a kettő a Dolgozóból keletkezett, az egyik a programozók, a másik a programtervezők Dolgozóból kiválasztott sorait tartalmazza /korlátozás/. Mivel csak a részlegekre /azok kódjaira/ vagyunk kiváncsiak a második lépésben

ezt az oszlopot emeljük ki projekcióval az Eredmény relációvá. Ez a lépés egyben garantálja a duplikátum részlegkódok megszünését is. Az eredmény rendezésére /v.ö. 1.2.1a/ itt nincs lehetőség.

b/ Beépitett függvény

Itt nincs, igy 1.2.1.b kérdését relációalgebrával nem tudjuk megválaszolni. /Egyébként "tiszta", beépitett függvények nélküli relációkalkulussal sem tudnánk./ A relációalgebra kiegészitése beépitett függvényekkel sokkal bonyolultabb, mint a relációkalkulusé. [CODD 71c]

c/ Összetett /több relációt érintő/ lekérdezés

Keressük ki minden programozó nevét, alapbérét és részlege cimét!

```
SELECT DOLGOZÓ WHERE BESOROLÁS='PROGRAMOZÓ'GIVING PROGRAMOZÓ
JOIN PROGRAMOZÓ AND RÉSZLEG OVER RÉSZLEGKÓD GIVING ADATOK
PROJECT ADATOK OVER NÉV,ALAPBÉR,CIM GIVING EREDMÉNY
```

Az első lépés - a korlátozás - kiválogatja a programozókra vonatkozó sorokat, a második mindegyik sor mögé odailleszti a megfelelő Részleg sort /erre a Cim miatt van szükség/, a harmadik kiválasztja a kért adatokat tartalmazó oszlopokat.

d/ Lekérdezés csoportositott adatok szerint

1.2.1.d-ben megkerestük a 10-nél több programozót foglalkoztató részlegeket. Ezt itt nem tudjuk megtenni, ismét csak a beépitett függvények hiánya miatt /nincs

lehetőség egy reláció sorainak leszámolására/.

e/ "Az összes" tipusu lekérdezés

Keressük meg az olyan szállitókat, akik az 50 kódu részleg által felhasznált összes cikkszámot szállitják!

A kérdés kényelmes megválaszolásához uj relációalgebrai müveletet vezetünk be, az osztást /division/. Ez a müvelet egy kétoszlopos relációból /A/ és egy másik oszlopból /B/ /egyetlen oszlopból álló relációból/ állit elő, egy egyoszlopos relációt /C/, a következő módon:

Legyen az A reláció egy sora /x,y/! Az x elem csak akkor kerül be C-be, ha A minden B-ben előforduló z értékre tartalmazza az /x,z/ párt /feltételeztük, hogy A második oszlopa és B ugyanazon az értékkészleten vannak értelmezve/.

Fontosnak tartjuk megjegyezni, hogy a most bevezetett müveletet a 0.1.2.-ben bevezetettekből előállitható, igy nem jelenti a relációalgebra bővitését. Most pedig a lekérdezés:

```
SELECT FELHASZNÁLÁS WHERE RÉSZLEGKÓD=50 GIVING ADATOK
PROJECT ADATOK OVER CIKKSZÁM GIVING CIKKSZÁMOK
DIVIDE SZÁLLITÁS BY CIKKSZÁMOK GIVING EREDMÉNY
```

A megoldás a szinte épp a feladatra szabott osztással igen egyszerü: az első két lépés előállitja az 50 osztály által felhasznált összes cikkszámot: az első lépés a Felhasználás adataiból válogatja ki az 50-es osztályra vonatkozókat, a második kiemeli a szükséges oszlopot /a másik oszlopban ugyis csupa 50 szerepelt/. Az osztás-definiciója szerint - éppen a kivánt eredményt adja/még egyszer hangsulyozzuk levezethetőségét a többi müveltből/.

f/ Uj sor illesztése relációba

DOLGOZÓ UNION {9286, 'KISS PÁL',52} GIVING DOLGOZÓ

Annyi az uj benne, hogy a GIVING rész egy uj reláció helyett egy már meglévőt tartalmazza, s az iródik /logikailag/ ujra.

g/ Lekérdezés eredményének illesztése relációba

Illesszük be a "Programozási Osztály" nevü relációba az összes programozó nevét, alapbérét és annak a részlegnek a cimét, ahol dolgozik!

A c/-ben leirt lekérdezést kell még egy sorral kiegészíteni:

PROGRAMOZÁSI_OSZTÁLY UNION EREDMÉNY GIVING PROGRAMOZÁSI_OSZTÁLY

h/ Egy sor törlése

Töröljük az 5618 törzsszámu dolgozót!

DOLGOZÓ MINUS {5618,?,?,?,?,?} GIVING DOLGOZÓ

/A MINUS a halmazelméleti különbség./

i/ Bonyolult lekérdezés eredményének törlése

Töröljük azokat a részlegeket, melyek egyetlen dolgozót sem foglalkoztatnak!

PROJECT DOLGOZÓ OVER RÉSZLEGKÓD GIVING LÉTEZŐ_RÉSZLEG
JOIN RÉSZLEG AND LÉTEZŐ_RÉSZLEG OVER RÉSZLEGKÓD GIVING T

Az első lépés kiválogatja azoknak a részlegeknek a kódját, amelyben legalább egy dolgozó van. A második ezt az oszlopot illeszti a részleg relációhoz, elérve ezzel, hogy T-ben már csupán azok a sorok szerepeljenek, melyek részlegkódja a "Létező részleg" relációban benne van. Most már csak a két - tökéletesen azonos - Részlegkód sor egyikétől kell megszabadulni:

```
PROJECT T OVER RÉSZLEG.RÉSZLEGKÓD AND RÉSZLEGNÉV AND CIM
        GIVING RÉSZLEG
```

j/ Felujitás

1.2.1.j-ben minden dolgozónak 10 %-os béremelést adtunk. Ez itt nem megy, mert nincsenek aritmetikai kifejezések. Egyébként [DATE 77] nyelvén nincs speciális felujitási lehetőség, helyette azt emliti, hogy a régi sorok MINUS-szal törölhetők a felujitottak pedig UNION-nal hozzácsatolhatók a relációhoz. [DATE 77]

1.2.3. SQL/DS

Az SQL vagy SEQEL 2 nyelv természetesen már "igazi" relációs nyelv, müködő adatbáziskezelő rendszer része. Az adatdefiniciós lehetőségeit 1.1.1.-ben irtuk le, itt most a lekérdező-módositó részét vizsgáljuk. Megemlitjük még, hogy a lekérdezések - ahogy a következő példákban szerepelnek - eredményei közvetlenül terminálra kerülnek. Ha a felhasználónak más szándékai vannak az eredménnyel 1.1.1.e és 1.1.1.f-ben tárgyalt lehetőségek állnak a rendelkezésre.

a/ Egyszerü lekérdezés

Válasszuk ki azokat a részlegeket, ahol programozók

vagy programtervezők dolgoznak!

```
SELECT RÉSZLEGKÓD
FROM DOLGOZÓ
WHERE BESOROLÁS IN 'PROGRAMOZÓ','PROGRAMTERVEZŐ'
```

Az első sor a kívánt adatot, a másik a kérdezett relációt a harmadik a feltételt specifikálja. Tulajdonképpen 1.2.1.a, "áramvonalasabb" formája, jól olvasható angol mondat. A feltételt irhattuk volna

```
WHERE BESOROLÁS='PROGRAMOZÓ' OR
      BESOROLÁS='PROGRAMTERVEZŐ'
```

formában is. Elérhetjük - az ALPHA-hoz hasonlóan - az eredmény rendezettségét is az

```
ORDER BY RÉSZLEGKÓD
```

sor WHERE után biggyesztésével.

Ami eltérés az ALPHA-tól - és általában a relációkkal dolgozó nyelvek filozófiájától - az, hogy az eredményből kapott adathalmazból a duplikátumok nincsenek kiszürve /ezért nem is nevezzük az eredményt "reláció"-nak/, vagyis minden részlegkódot annyiszor kapunk meg, ahány programozó vagy programtervező dolgozik az illető részlegben. Természetesen ez elkerülhető, de ezt az igényt külön jelezni kell

```
SELECT UNIQUE RÉSZLEGKÓD
```

formában a rendszernek. [CHAM 76] ezt avval indokolja, hogy a duplikátumok kiszürése nagyon költséges müvelet /ez nyilván igaz/, és csak akkor érdemes elvégezni, ha ez valóban szükséges. Meggyőző hiszen sokszor fordul elő, hogy szemantikus meggondolásokból biztosak lehetünk abban, hogy az eredmény nem tartalmaz azonos sorokat. Olyan is előfordulhat,

hogy nem baj, sőt esetleg egyenesen kívánatos, hogy a duplikátumok benne maradjanak az eredményben.

b/ Beépitett függvény

Hány programozónak van 4000 forintnál nagyobb alapbére?

```
SELECT COUNT(*)
FROM DOLGOZÓ
WHERE ALAPBÉR > 4000 AND BESOROLÁS='PROGRAMOZÓ'
```

Ez is nagyon hasonlit a megfelelő ALPHA kérdésre /1.2.1.b/. Ami talán magyarázatra szorul az a COUNT(*) kifejezés. A "*" a teljes sort jelzi az összes mezővel COUNT(*) pedig a sorok számát - amire szükségünk van. Az SQL - az ALPHA-hoz hasonlóan /1.2.1.b./ - széleskörü beépitett függvénykészlettel rendelkezik.

c/ Összetett /több relációt érintő/ lekérdezés

Keressük ki minden programozó nevét, alapbérét, és a részlege cimét!

```
SELECT DOLGOZÓ.NÉV,DOLGOZÓ.ALAPBÉR,RÉSZLEG.CIM
FROM DOLGOZÓ,RÉSZLEG
WHERE DOLGOZÓ.RÉSZLEGKÓD=RÉSZLEG.RÉSZLEGKÓD
   AND DOLGOZÓ. BESOROLÁS='PROGRAMOZÓ'
```

Ugyanazok a megjegyzések vonatkoznak erre a lekérdezésre is, mint ALPHA nyelvü megfelelőjére /1.2.1.c/.

Keressük ki azokat a dolgozókat, akik alapbére magasabb mint a főnöküké!

```
SELECT D.NÉV,F.NÉV
FROM DOLGOZÓ D,DOLGOZÓ F
WHERE D.FŐNÖK=F.TÖRZSSZÁM AND
      D.ALAPBÉR > F.ALAPBÉR
```

Összevetve megint az ALPHA-val /1.2.1.c/, jól látható a RANGE áramvonalasitott formája. Jobban olvasható, kényelmesebb, de a lényege ugyanaz: A D és F a Dolgozó reláció két példánya. Az olyan sorpárokat keressük a két táblában, ahol a D-beli dolgozónál a Főnök mezőben az F-beli dolgozó törzsszáma van /tehát az F sorral reprezentált dolgozó a D sorral reprezentált főnöke/; és a D sorban az alapbér összege nagyobb, mint az F-ben.

d/ Lekérdezés csoportositott adatok szerint

Keressük meg azokat a részlegeket, melyekben 10-nél több programozó dolgozik!

```
SELECT RÉSZLEGKÓD
FROM DOLGOZÓ
WHERE BESOROLÁS='PROGRAMOZÓ'
GROUP BY RÉSZLEGKÓD
HAVING COUNT(*)>10
```

Ez első ránézésre nem hasonlit annyira ALPHA-beli megfelelőjére /1.2.1.d/, pedig hasonló a végrehajtás logikája. Először az első három sor szerinti lekérdezés kiválogatja a Dolgozó reláció, programozókra vonatkozó sorait. Utána részlegkód szerint létrejönnek a csoportok, majd a csoportok közötti kiválasztásra vonatkozó HAVING /mögötte mindig a csoport egészét jellemző beépitett függvénynek - COUNT, SUM, AVG, stb. - kell állnia a feltételben/ müködik, és kiszüri a 10-nél több sort tartalmazó csoportokat.

A GROUP állhat a HAVING csoportszürő feltétel nélkül is. Pl.

```
SELECT RÉSZLEGKÓD, AVG(ALAPBÉR)
FROM DOLGOZÓ
GROUP BY RÉSZLEGKÓD
```

lekérdezés a részlegek kódját, és az ott dolgozók átlagbérét adja vissza.

e/ "Az összes" tipusu lekérdezés

Keressük meg az olyan szállitókat, akik az 50 kódu részleg által felhasznált összes cikkszámot szállitják!

```
SELECT SZÁLLITÓ
FROM SZÁLLITÁS X
WHERE
   (SELECT CIKKSZÁM
    FROM SZÁLLITÁS
    WHERE SZÁLLITÓ=X.SZÁLLITÓ)
CONTAINS
   (SELECT CIKKSZÁM
    FROM FELHASZNÁLÁS
    WHERE RÉSZLEGKÓD=50)
```

A lekérdezést a következőképpen lehet "magyarra forditani": a Szállitás reláció X másodpéldányából válogatjuk ki azokat a szállitókat, amelyekre fennáll az, hogy kiválasztva a Szállitó relációból az összes általuk szállitott cikkszámot /ezt a WHERE-t követő SELECT csinálja/ az igy kapott halmaz tartalmazza az 50-es részleg által használt összes cikkszámot /a CONTAINS utáni SELECT generálta halmaz/.

A lekérdezés illusztrálja az SQL relációalgebrai /halmazalgebrai/ lehetőségeit. A SELECT-ek eredményei halmaznak tekinthetők, ha halmazelméleti müveleteket /a CONTAINS mellett más egyéb, pl. UNION, MINUS, INTERSECTION is van/ alkalmazunk rájuk. Ez azt is jelenti, hogy a duplikátumok kiszürése automatikusan megtörténik. Nincs lehetőség az összes relációalgebrai lehetőség ismertetésére

- a/-ban láttuk az IN-t - , de szeretnénk hangsulyozni: az SQL nem tekintehtő tisztán relációkalkuluson alapuló nyelvnek. /Már csak azért sem mert nincsenek benne kvantorok/.

f/ Uj sor illesztése relációba

```
INSERT INTO DOLGOZÓ(TÖRZSSZÁM,NEV,RÉSZLEGKÓD) :
   <9286, 'KISS PÁL', 52 >
```

g/ Lekérdezés eredményének illesztése relációba

Illesszük be a "Programozási Osztály" nevü relációba az összes programozó nevét, alapbérét és annak a részlegnek a cimét ahol dolgozik!

```
INSERT INTO PROGRAMOZÁSI OSZTÁLY
  SELECT DOLGOZÓ.NÉV,DOLGOZÓ.ALAPBÉR,RÉSZLEG.CIM
  FROM DOLGOZÓ,RÉSZLEG
  WHERE DOLGOZÓ.RÉSZLEGKÓD=RÉSZLEG.RÉSZLEGKÓD AND
        DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'
```

h/ Egy sor törlése

Töröljük az 5618 törzsszámu dolgozót!

```
DELETE DOLGOZÓ
WHERE TÖRZSSZÁM=5618
```

i/ Összetett lekérdezés eredményének törlése

Töröljük azokat a részlegeket, melyek egyetlen dolgozót sem alkalmaznak!

```
DELETE RÉSZLEG R
WHERE
   (SELECT COUNT(*)
    FROM DOLGOZÓ
    WHERE DOLGOZÓ.RÉSZLEGKÓD=R.RÉSZLEGKÓD)=0
```

j/ Felujitás

Adjunk minden programozónak 10 % béremelést!

```
UPDATE DOLGOZÓ
SET ALAPBÉR=ALAPBÉR * 1.1
WHERE BESOROLÁS='PROGRAMOZÓ'
```

[CHAM 76]

1.2.4. INGRES

Az INGRES rendszer lekérdező-módositó nyelvét QUEL-nek /QUERY Language/ hivják. Az 1.1.2-ben ismertetett adatdefiniciós parancsok nem tartoznak a QUEL-be, önálló "INGRES utility commands"-nak nevezi őket [STON 76].

Az INGRES mint ezt látni fogjuk - inkább nevezhető relációkalkulus alapu nyelvnek, mint az SQL, noha sok közös vonásuk van, lényegében csak kulcsszavak, és a bonyolultabb lehetőségek - beépitett függvények, halmazmüveletek, stb. - különböznek. Az alább következő lekérdezések eredménye itt is rögtön terminálra kerül - ha másként kivánja a felhasználó, ugy 1.1.2.d lehetősége áll rendelkezésére.

a/ Egyszerü lekérdezés

Válasszuk ki azokat a részlegeket, ahol programozók vagy programtervezők dolgoznak!

```
RANGE OF D IS DOLGOZÓ
RETRIEVE(D.RÉSZLEGKÓD)
WHERE D.BESOROLÁS='PROGRAMOZÓ' OR
      D.BESOROLÁS='PROGRAMTERVEZŐ'
```

A RANGE ugyanaz, mint az ALPHA-ban /1.2.1.c/, de itt ha relációval akar dolgozni az ember, minden esetben meg kell

adni rajta egy változót, a reláció nevére nem lehet hivatkozni, mint ezt a megfelelő ALPHA és SQL lekérdezések /1.2.1.a és 1.2.3.a/ teszik. Az SQL "IN" lehetősége /1.2.3.a/ itt nincs meg.

Az INGRES a duplikátumokat másképpen kezeli, mint az eddig vizsgált rendszerek. Egy terminálra irandó lekérdezés eredményéből sosem szüri ki őket, és erre nincs is nyelvi lehetőség. Viszont, ha a lekérdezés eredményét egy uj relációba irányitjuk /1.1.2.d/, akkor a duplikátumok kiszürése automatikusan megtörténik /sőt le is rendezi az uj relációt/.

b/ Beépitett függvény

Hány programozónak van 4000 forintnál nagyobb alapbére?

```
RANGE OF D IS DOLGOZÓ
RETRIEVE (COUNT(D.ALL))
WHERE D.BESOROLÁS='PROGRAMOZÓ' AND
      D.ALAPBÉR > 4000
```

Lényegében megegyezik az SQL-beli megfelőjével /1.2.3.b/. Az ALL az egész sort jelöli, a COUNT a szokásos számláló függvény. Az INGRES-nek is megvannak a szokásos beépitett függvényei.

c/ Összetett /több relációt érintő/ lekérdezés

Keressük ki minden programozó nevét, alapbérét és a részlege cimét!

```
RANGE OF D IS DOLGOZÓ
RANGE OF R IS RÉSZLEG
RETRIEVE(D.NÉV,D.ALAPBÉR,R.CIM)
WHERE D.RÉSZLEGKÓD=R.RÉSZLEGKÓD AND
      D.BESOROLÁS='PROGRAMOZÓ'
```

Keressük ki azokat a dolgozókat, akiknek alapbére nagyobb, mint a főnöküké!

```
RANGE OF D IS DOLGOZÓ
RANGE OF F IS DOLGOZÓ
RETRIEVE (D.NÉV,F.NÉV)
WHERE D.FŐNÖK=F.TÖRZSSZÁM AND
      D.ALAPBÉR > F.ALAPBÉR
```

d/ Lekérdezés csoportositott adatok szerint

Keressük meg azokat a részlegeket, melyekben 10-nél több programozó dolgozik!

```
RANGE OF D IS DOLGOZÓ
RETRIEVE (D.RÉSZLEGKÓD)
WHERE COUNT(D.ALL By D.RÉSZLEGKÓD WHERE
            D.BESOROLÁS='PROGRAMOZÓ) > 10
```

A COUNT beépitett függvényben lévő két konstrukció közül a BY... végzi a csoportositást, a WHERE... pedig a programozók kiválogatását /nem szükségképpen ebben a sorrendben, és nem is biztos, hogy külön lépésben zajlanak ezek a müveletek/. A COUNT az igy kapott csoportok tagjait számlálja.

A BY-nak az INGRES-ben értelme csak beépitett függvény /lehet más is, nem csak a COUNT/ belsejében van. A WHERE feltétele kivehető a beépitett függvényből, de ezzel a lekérdezés értelme megváltozik. A

```
RANGE OF D IS DOLGOZÓ
RETRIEVE (D.RÉSZLEGKÓD)
WHERE COUNT(D.ALL BY RÉSZLEGKÓD) > 10  AND
      D.BESOROLÁS='PROGRAMOZÓ'
```

lekérdezés azt csinálja, hogy végigmegy a dolgozókon, és ha egy programozót talál, akkor leszámolja a részlege összes dolgozóját, és ha ezek száma 10-nél nagyobb, akkor

kiirja a részlegkódot. A lekérdezés eredménye tehát a 10-nél nagyobb létszámu, legalább egy programozót alkalmazó részlegek kódjai lesznek - mindegyik annyiszor kiirva ahány programozójuk van.

Elég nyilvánvaló, hogy az INGRES BY lehetősége és az SQL GROUP BY-a egymás megfelelői, és mindkettő őse az ALPHA ICOUNT /ISUM,IAVG stb./ beépitett függvénye /1.2.1.d./, noha annál jobbak.

e/ "Az összes"tipusu lekérdezés

Keressük meg az olyan szállitókat, akik az 50 kódu részleg által felhasznált összes cikkszámot szállitják!

A választ éppen ugy, mint az ALPHA-nál /1.2.1.e./ két lépéssel kapjuk meg. Először előállitjuk egy önálló relációban - ezt persze előbb létre kell hozni /ld. 1.1.2.d/ - az 50-es részleg által felhasznált összes cikkszámot:

```
RANGE OF F IS FELHASZNÁLÁS
RETRIEVE INTO CIKKSZÁMOK (F.CIKKSZÁM)
WHERE F.RÉSZLEGKÓD=50
```

A következő lépésben kapjuk az eredményt:

```
RANGE OF S IS SZÁLLITÁS
RANGE OF SX IS SZÁLLITÁS
RANGE OF C IS CIKKSZÁMOK
RETRIEVE (S.SZÁLLITÓ)
WHERE COUNT(C.CIKKSZÁM WHERE C.CIKKSZÁM=SX.CIKKSZÁM
            AND SX.SZÁLLITÓ=S.SZÁLLITÓ) =
      COUNT(C.CIKKSZÁM)
```

A lekérdezés logikája szerint azokat az S sor-beli szállitókat irjuk ki, melyekre fennáll a következő: azoknak a C-be tartozó cikkszámoknak a szama, melyekre letezik olyan SX sor,

amely sorban a szállitó éppen az S szállitója, a cikkszám pedig a C-beli cikkszám /vagyis S.szállitó szállitja.C.cikkszámot/ megegyezik a C-be tartozó összes cikkszám számával. Megjegyezzük, hogy az eredményben minden szállitókód annyiszor szerepel, ahány különböző sorban fordul elő a Szállitás relációban.

A lekérdezés logikája megegyezik az ALPHA megfelelő lekérdezésének /1.2.1.e./ logikájával. A különbség az, hogy az INGRES nem használ kvantorokat, ugyanis nincs rájuk szükség. Az egzisztenciális kvantor /∃/ implicite minden feltételben benne van: igy definiált a lekérdezés szemantikája. Az univerzális kvantor /∀/ pótolható pl. a COUNT beépitett függvénnyel; a $\forall x\ P(x)$ predikátum ekvivalens a $COUNT(\neg P(x)) = 0$-val [DATE 77].

Ennél a lekérdezésnél szembeötlőek az INGRES és az SQL /1.2.3.e/ közötti különbségek. Lényegében itt az INGRES erősebb relációkalkulus-orientáltságáról van szó, az SQL relációalgebrai logikája /a CONTAINS/ nincs meg benne, igy bizonyos dolgokat körülményesebb QUEL-ben leirni. Az SQL logikája direkt: egy-egy SELECT előállitja a két halmazt /egy-egy szállitó által szállitott cikkszámokét, ill. az 50-es osztály által felhasznált cikkszámokét/ és a CONTAINS összehasonlitja őket.

f/ Uj sor illesztése relációba

```
APPEND TO DOLGOZÓ (TÖRZSSZÁM=9826,NÉV='KISS PÁL',
                   RÉSZLEGKÓD=52)
```

g/ Lekérdezés eredményének illesztése relációba

Illesszük be a "Programozási Osztály" nevü relációba az összes programozó nevét, alapbérét és annak a részlegnek a cimét

ahol dolgozik!

```
RANGE OF D IS DOLGOZÓ
RANGE OF R IS RÉSZLEG
APPEND TO PROGRAMOZÁSI OSZTÁLY(NÉV =D.NÉV,
            ALAPBÉR=D.ALAPBÉR,RÉSZLEGKÓD=R.RÉSZLEGKÓD)
WHERE D.RÉSZLEGKÓD=R.RÉSZLEGKÓD AND
      D.BESOROLÁS='PROGRAMOZÓ'
```

h/ Egy sor törlése

Töröljük az 5618 törzsszámu dolgozót!

```
RANGE OF D IS DOLGOZÓ
DELETE D
WHERE D.TÖRZSSZÁM=5618
```

i/ Összetett lekérdezés eredményének törlése

Töröljük azokat a részlegeket, melyek egyetlen dolgozót sem alkalmaznak!

```
RANGE OF R IS RÉSZLEG
RANGE OF D IS DOLGOZÓ
DELETE R
WHERE COUNT(D.ALL WHERE D.RÉSZLEGKÓD=R.RÉSZLEGKÓD)=0
```

j/ Felujitás

```
RANGE OF D IS DOLGOZÓ
REPLACE D(ALAPBÉR=1.1*ALAPBÉR)
WHERE D.BESOROLÁS='PROGRAMOZÓ'
```

[STON 76, EPST 77].

1.2.5. QBE

Mint erről 1.1.3-ban már beszámoltunk, a QBE grafikus nyelv, a felhasználó a rendszer által képernyőre irt táblázatokba "full-screen" módban a megfelelő helyekre szövegeket irva tartja a kapcsolatot a rendszerrel. Alapgondolata, hogy mindent példákon keresztül ért meg a rendszer. A felhasználó beir a táblázat egy-két sorába egy megoldást -"igy néz ki egy olyan sor, amilyenre szükségem van"- és a QBE kikeresi az adatbázisból az összes "olyan"sort.

a/ Egyszerü lekérdezés

Válasszuk ki azokat a részlegeket, ahol programozók vagy programtervezők dolgoznak!

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
			P. EGY	PROGRAMOZÓ		
			P. KETTŐ	PROGRAM-		
				TERVEZŐ		

Az aláhuzott EGY és KETTŐ a változók, vagy mintaelemek. Az aláhuzatlan szövegek konstansok. Milyen legyen a kivánt lista? Két mintasor van megadva, mind a két sor jó. A két sor kitöltetlen oszlopaiban lévő értékek a lekérdezés szempontjából közömbösek. A Részlegkódban lévő értékeket kell kiirni - erre utal a P. - de csak a mintának megfelelő sorokból, tehát azokból ahol a Besorolás programozó /első sor/ vagy programtervező /második sor/.

A rendszer a duplikátumokat automatikusan kiszüri. Rendezés

megadható ha P. helyett P. AO. /Print Ascending Order./ szerepel.

b/ Beépített függvény

Hány programozónak van 4000 R-ot meghaladó alapbére?

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
P.CNT.ALL				PROGRAMOZÓ		>4000

A mintasorból jól látni, hogy milyen sorokat kell a rendszernek kiválogatni az adatbázisból: a Besorolás programozó, az Alapbér pedig 4000-nél nagyobb. Az egész sort jelképező dolgozó oszlopban van a kiiratást jelző "P". A mögötte álló CNT, a számláló /eddigiekben COUNT/ beépitett függvény. Az ALL halmaz képzését jelenti /duplikátumok kiszürése nélkül/, a CNT. ALL tehát a halmaz számosságát jelzi - ezt kell kiirni. /Az ALL. megfelelője az SQL/DS-ben a "SET"/.

c/ Összetett /több relációt érintő/ lekérdezés

Keressük ki minden programozó nevét, alapbérét és a részlege cimét!

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
		XXX	N1	PROGRAMOZÓ		Y

RÉSZLEG	RÉSZLEGKÓD	RÉSZLEGNÉV	CIM
	N1		AB

VALAMI	IZÉ	MINDEGY	Q
	P.XXX	P.Y.	P.AB

A lekérdezéshez három tábla szükséges. A Dolgozó oszlopaiban megadjuk, hogy csak a programozók érdekelnek, és mintaértékeket irunk a Név, Részlegkód, Cim oszlopokba, vagyis ezeknek az értékeit kívánjuk használni. A Részleg tábla Részlegkód mezőjében ugyanaz a mintaérték /N1/ áll, mint a Dolgozóéban - egy konkrét Dolgozó sorhoz tehát azt a Részleget kell megtalálni, ahol a Részlegkód a Dolgozóéval egyezik /de csinálhatja a rendszer forditva is, a részlegekhez keresve a dolgozókat, az illesztés feltétele akkor is változatlan/. A Részleg relációra a Cim miatt van szükségünk, mint azt a mintaérték /AB/ jelzi. A harmadik tábla csak a listázás miatt kell, mint azt azt oszlopnevek is jelzik. Ebben kapjuk majd meg az eredményt. Ha a harmadik tábla nem lenne, és a Dolgozó, ill. Részleg táblában P.XXX-et, P.Y-t ill. P.AB-t adunk meg, akkor két, egymástól független listát kapunk: egyiken a programozók nevei és alapbérei. másikon a részlegeik kódjai szerepelnek majd, de ezeket összeilleszteni - a duplikátumok kiszürése miatt - nem lehetne.

Keressük ki azokat a dolgozókat, akik alapbére több, mint a főnöküké!

Nagyon elegánsan, egy táblával megoldható:

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
	D	P.				SOK
		P.			D	>SOK

Az ábrán látható két sor közül a felső a főnöké, az alsó a beosztotté. Ezt a viszonyt a felső sor Törzsszám és az alsó sor Főnök mezőjében álló azonos érték /D/ jelzi. A főnök alapbére SOK /valamennyi/ a beosztotté >SOK /nagyobb, mint valamennyi/. A két név irandó ki.

d/ Lekérdezés csoportositott adatok szerint

Keressük meg azokat a részlegeket, ahol 10-nél több programozó dolgozik!

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
	All S1		P. R	PROGRAMOZÓ		

CONDITIONS
CNT.ALL.S1 > 10

A megoldáshoz egy speciális QBE eszközt, a feltétel-dobozt /condition-box/ is igénybe kell venni. A Dolgozó táblázatból kiderül, hogy a programozokat kell kivalogatni, P.R a Részlegkódban azt tudatja, hogy csoportositani kell a Részlegkód szerint a sorokat és a megfelelő részlegkódok kiirandók. A kiiratás feltétele a feltétel-dobozban van

megadva - az egy csoportba tartozó törzsszámok halmazának /ALL.S1/ tagszáma 10-nél nagyobb kell hogy legyen.

e/ "Az összes tipusu lekérdezés

Keressük meg az olyan szállitókat, akik az 50 kódu részleg által felhasznált összes cikkszámot szállitják!

FELHASZNÁLÁS	RÉSZLEGKÓD	CIKKSZÁM
	50	ALL.CIKK

SZÁLLITÁS	SZÁLLITÓ	CIKKSZÁM
	P.S	ALL.CIKK x

A megoldás lényegében az SQL/DS "CONTAINS" lehetőségének /1.2.3.e./ grafikus megfelelője. A Felhasználás táblában az 50-es részleg által felhasznált összes cikkszámot jelképezi az ALL.CIKK. A Szállitás tábla jelzi, hogy Szállitó szerint csoportositva a cikkszámokat, azokat a szállitókat kell kiirni, ahol a kapott cikkszámhalmaz megegyezik ALL.CIKK-kel, ill. még további elemet is tartalmazhat /x/.

A QBE logikájában igen hasonlit az SQL/DS-hez, szinte annak grafikus megfelelője, erősségben sem marad el tőle, ugyanugy tartalmaz relációkalkulus és algebra jellegü lehetőségeket, mint az.

f/ Uj sor illesztése relációba

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
I.	9826	KISS PÁL	52			

A Dolgozó oszlopban álló I. miatt kerül be a fenti /hiányosan megadott/ sor a Dolgozó táblába.

g/ Lekérdezés eredményének illesztése relációba

Illesszük be a "Programozási Osztály" nevü relációba az összes programozó nevét, alapbérét, és annak a részlegnek a cimét, ahol dolgozik!

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
		XXX	N1	PROGRAMOZÓ		Y

RÉSZLEG	RÉSZLEGKÓD	RÉSZLEGNÉV	CIM
	N1		AB

PROGRAMOZÁSI_OSZTÁLY	NÉV	ALAPBÉR	CIM
I.	XXX	Y	AB

Teljesen ugy megy, mint a c/-ben tárgyalt lekérdezés, csak az eredményül kapott reláció /ezelőtt valamikor definiálnunk is kellett, hogy ez a beillesztés menjen/ és a kiirást jelző P. helyett ezuttal az egész sorra vonatkozó, beillesztést jelző I. áll.

h/ Egy sor törlése

Töröljük az 5618 törzsszámu dolgozót

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
D.	5618					

i/ Összetett lekérdezés eredményének törlése

Töröljük azokat a részlegeket, melyek egyetlen dolgozót sem alkalmaznak!

RÉSZLEG	RÉSZLEGKÓD	RÉSZLEGNÉV	CIM
D.	R1		

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
¬			R1			

A rendszer végigmegy a Részleg táblán, és megpróbálja törölni sorban az összes sort. A Dolgozó tábla kitöltése miatt ezt azonban csupán azoknál a soroknál teszi meg, ahol a Részlegkód /R1/ a Dolgozó tábla egyetlen sorában sem fordul elő.

j/ Felujitás

Adjunk minden programozónak 10 % béremelést!

DOLGOZÓ	TÖRZSSZÁM	NÉV	RÉSZLEGKÓD	BESOROLÁS	FŐNÖK	ALAPBÉR
U.				PROGRAMOZÓ		1.1×S1 S1

[ZLOO 77].

1.2.6. Egyéb rendszerek

A rendszereket a felhasználói interface vizsgálata két csoportra osztja: az egyikbe a relációsan teljes /0.1.2./ rendszerek tartoznak, a másikba azok, melyek nem rendelkeznek ezzel a tulajdonsággal. Utóbbiaknál ez a gyakorlatban az illesztés hiányában mutatkozik meg, ami viszont azt jelenti, hogy a felhasználó egy kérdéssel csak egy relációra kérdezhet rá. A felhasználói interface döntő jelentőségü jellemzőjének tartjuk a relációs teljességet. /Érdekes kompromisszum ebből a szempontból a dBASE II, forgalmazott adatbáziskezelő rendszer. Ebben létezik ugyan illesztés /JOIN/, de javasolja annak mértékletes használatát, mert nagy file-okra időigényes [ASHT 81]./

A nem relációsan teljes rendszereket az [EBER 83] dolgozatban leirt, 1.1.4.-ben emlitett rendszer példáján mutatjuk be. A felhasználói interface kellemes kérdez-felelek játékból áll, melynek során először a lekérdezendő relációt, annak kiirandó oszlopait, majd a lekérdezés feltételeit lehet megadni. Lehetőség van rendezés kérésére is. Az adatmódositások hasonlóan zajlanak.

A relációs rendszerek általában az 1.2.1.-1.2.5.-ben ismertetett "nagy" nyelvek valamelyikét használják - többé kevésbé átalakitva. E szerint csoportositottuk mi is a rendszereket.

A relációalgebrát használó rendszerek őse a PRTV /ld. 0.2/ volt. Az INGRES-ről és a SEQUEL-ről szóló első publikációk /[STON 76 , ASTR 75]/ után a relációalgebrát kényelmesebben olvasható, és kevésbé procedurális nyelvek váltották fel. A relációalgebra szemmel láthatóan a mikrogépeken indul uj virágzásnak. Ugy gondoljuk, ennek elsősorban a könnyebb implementálhatóság az oka /ezt a feltevést [DEEN 83] explicite megerősiti/.

Az RQL müveletei az illesztés, korlátozás /kiválasztás/ és a projekció. Az algebrai müveleteket kiegészitették beépitett függvényekkel /pl. SUM-oszlop összege/, és a korlátozást általánositották több, Boole művelettel összekapcsolt feltétel irányába. Mivel az algebra procedurális jellegü - egy-egy lekérdezés több lépésből állhat /ld. 1.2.2./ - és a gépen egy-egy lépés válaszideje elég hosszu lehet, - a tapasztalatok szerint 1 perc és 1 óra közötti idők vannak - batch lehetőséget is biztositottak a felhasználónak. A "USE OF" parancs hatására a rendszer az utasitásokat a parancsban megadott nevü file-ról várja, és nem kell a gép mellett ülni várva az aktuális lépés végrehajtására, hogy a következőt be lehessen gépelni. [MAST 83] /ld, még 1.1.4./

Az _MRDBS_ a korlátozást, projekciót, permutációt és az illesztést használja. Természetesen a halmazelméleti müveleteket is implementálja. A szerzők szerint ezzel a rendszerrel is vannak hatékonysági problémák - főként természetesen az illesztéssel. [REVE 83]

Az INGRES QUEL-jét implementálta az R-DBMS rendszer. Ez INTEGRA-1001 gépen fut, amiről azt biztosan tudjuk, hogy nem mikrogép, mert a rendszer egyes moduljai a 220K nagyságot is elérik. Az alkotók szerint megfelelő overlay-esitéssel 16 bites mikroprocesszorokra is alkalmassá lehetne tenni a rendszert.

Érdekes továbbfejlesztés az INGRES-hez képest a rekurziv lekérdezés. Az "alkatrész tipusu" relációknál használható. Ha van egy alkatrészekből összeállitott szerkezet, amelynek alkatrészei maguk is összetett szerkezetek és igy tovább, akkor a relációs nyelvek nem képesek olyan lekérdezesek, mint pl. "Keressük meg X gép elemi /alkatrészekre tovább nem bontható/ alkatrészeit!" megválaszolására. /Relációs nyelven, itt a reláció tranzitiv lezárásának lekérdezéséről van szó/. [APS1 83]

Az SQL-szerü nyelveket használó rendszerek közül az 1.1.4-ben már emlitett VIDEBAS rendszert emeljük ki. Ez az SQL bonyolultabb lehetőségeit /pl. GROUP BY, rendezés előirása, stb./ is tudja.[BLAN 83]

A QBE grafikus nyelvét használja az RDBAS rendszer. Ez 32K-s /!/ HP21MX-en fut - erősen szegmentálva, egyszerre mindig csak egy kis programrész tartózkodik a memóriában - RTE operációs rendszer alatt. /Hatékonyságára vonatkozó adatot nem ismerünk/. Ehhez a rendszerhez is létezik ügyes batch parancsfile lehetőség. A felhasználó a szokásos módon /1.2.5./ definiálja a müveletet, majd önálló nevet ad neki, és a rendszer az azonnali végrehajtás helyett csupán megjegyzi a nevet és a müveletet. Ezek a felhasználói parancsok parametrizálhatóak, és még a paraméterek megadására felszólító üzenetek is külön megadhatók a parancs definiálásakor. [HERM 83]

A TITAN UCSD-PASCAL-ban készült. APPLE II-n 54K-ban fut. Uj lehetőség benne - a QBE-hez képest - reláció irási lehetősége lemezfile-ba, illetve feltöltése lemezfile-ról /v.ö. 1.1.2.b/. [FALQ 82]

Mind a két rendszer - a közleményekben látható példák szerint-relációsan teljes, tehát a QBE-nek legalábbis jelentős részét képesek megvalósitani. A bonyolultabb lehetőségekre /1.2.5.d,e/ vonatkozó utalást nem találtunk, de azért elég meglepőnek tünnek az implementációk, első ránézésre a QBE látszik a legnehezebben megvalósithatónak /a szokásos adatbázis-problémák mellett még a szövegesnél bonyolultabb grafikus interface-t is meg kell csinálni./

A LIDAS rendszer HIQUEL /Hierarchical Interactive Query Language/ lekérdező nyelve is QBE-szerü grafikus technikát használ, de ez a nyelv lényegesen különbözik a QBE-től. Már az egy relációs lekérdezések megfogalmazásában, a feltételek megadásánál jelentkezik különbség, de

lényegi eltérés az összetett lekérdezéseknél van. A HIQUEL hierarchikus strukturák definiálására ad lehetőséget természetes nyelvü kapcsolatok segitségével. Egy könyvtári adatbázisban pl. A "Persons" és a "Borrowed-items" relációk "hierarchikus nézőponttá" /semmi köze az SQL/DS "né-. zőpont"-jához/ kapcsolhatók össze a "borrowed items OF A person" kapcsolattal. Az ábrán látható lekérdezés eredménye Miller vagy Smith urak által kölcsönzött könyvek jegyzéke lesz

PERSONS
NAME
MILLER
SMITH

BORROWED-ITEMS OF A PERSON
TITLE

[REBS 83, URSP 83]

A mikrogépes rendszereknél váltak elterjedtté az adatszerkesztők /data base editor/. Ezek a szövegszerkesztőkhöz /text editor/ hasonló stilusban irják képernyőre és engedik módositani egy reláció sorait. Adatszerkesztő lehetőség van pl. az MRDSA [MERR 83] és a dBASE II [ASHT 81] rendszerekben. Az előbbi a szerkesztő mellett még PASCAL rutinokból hivható relációalgebrai nyelvvel is rendelkezik - méghozzá általánossal, olyan extra lehetőségekkel, mint a többfajta definiálatlan értéknek megfelelő különböző illesztések.

Külön emlitjük az APPLE /Access Path Producing LanguagE/ nyelvet. Ez az RSX11-M operációs rendszer PDP 11-en létező DATARETRIEVE utility-jának általánositása. Jellegzetessége, hogy a felhasználónak még a relációkat sem kell megadnia, mindössze az adatok nevét. Az 1.2.1.c.-1.2.5.c lekérdezés

/a programozók neve, alapbére és részlegük cime itt igy irható/:

```
SELECT NÉV,ALAPBÉR,CIM
WHERE BEOSZTÁS='PROGRAMOZÓ'
```

A rendszer maga találja meg a relációkat, melyben a hivatkozott oszlopnevek vannak, és kapcsolja őket össze közös oszlopnevek alapján. Ha több lehetséges összekapcsolás van, a felhasználó választ közülük.

Ez igy nagyon egyszerü és elegáns, de a módszer elég rugalmatlan, hiszen csak a közös névvel rendelkező oszlopok szerint illeszthetők relációk. Ez igen komoly korlátozás, és a nyelv nyilvánvalóan nem relációsan teljes. [PATN 83]

A batch lehetőség továbbfejlesztésével találkozunk a dBASE II rendszernél. Itt a parancsfile-ban nem csak adatbázis utasitások nevei, hanem programozási nyelv-szerü vezérlőutasitások /DO WHILE, GO TO,stb./ is lehetnek. Egy-egy ilyen parancsfile szabályos program, végrehajtandó adatbázis és programnyelv-szerü vezérlő utasitásokkal. Mindez nem ugyanaz, mint egy programozási nyelvbe beépülő adatkezelő nyelv, itt ugyanis nincs lehetőség pl. egy lekérdezési eredményből programozási nyelvü feldolgozással statisztika készitésére.[ASHT 81]

1.3. Adatbáziskezelés programozási nyelvekből

Az eddigiekben megvizsgáltunk néhány relációs adatkezelő nyelvet. Ennek alapján nyugodtan állithatjuk, hogy valamennyi sokkal egyszerübb, mint egy programozási nyelv, és emellett elég bonyolult megfogalmazásu kérdésekre is képes választ adni. Mégis ugy tünik, hogy szükség van a hagyományosabb módszerre, a programozási nyelvekből végezhető adatkezelésre is. A vizsgált nyelvek egyike sem rendelkezik ugyanis egy programozási nyelv számoló, vezérlő

utasitásainak erejével /ha rendelkezne, maga is programozási nyelvvé válna,és elvesztené fő vonzerejét - egyszerüségét/. Az önálló nyelvek nem alkalmasak pl. statisztikák készitésére, és nem irhatók le bennük olyan bonyolult logikáju felujitások, mint pl. egy bérszámfejtés.

A magasszintü nyelvekből való adatkezelés sokféle módon megvalósitható, pl. az MRDSA-ban alkalmazott rutinhivástól a MODULA/R saját forditóprogrammal implementált beágyazott adatkezelő utasitásáig. Először ismertetjük az SQL/DS és az INGRES PL/1 és C gazdanyelvü adatkezelő interface-ét, majd megvizsgáljuk programozási-adatkezelő nyelveket.

1.3.1. Gazdanyelvek és adatnyelvek

Az INGRES és az SQL/DS rendszerek lehetőségeit fogjuk röviden ismertetni. Mind a két rendszer univerzálisan relációs /ld. 1./, vagyis a programozási nyelvbe beépülő adatkezelő nyelv mind a kettőnél megegyezik az önálló adatkezelő nyelvvel, a QUEL-lel ill. az SQL-lel. Mind a kettőnél előforditót /preprocessor/ alkalmaznak. Ez az adatkezelő utasitásokat a gazdanyelv CALL utasitásaira alakitja /megfelelő paraméterekkel/, és a gazdanyelv forditóprogramja már gazdanyelvü programot kap.

Az INGRES-ben a C programozási nyelvet a beépülő QUEL-lel EQUEL-nek /Embedded QUEL/ nevezik. Az EQUEL-t a következő szabályok definiálják:

a/ Minden C nyelvü utasitás EQUEL nyelvü utasitás /vagyis az EQUEL programban bármilyen C utasitás szerepelhet/.

b/ Minden QUEL utasitás EQUEL nyelvü utasitás /tehát az EQUEL a QUEL teljes kifejezőerejével bir/. A QUEL utasitásokat az utasitás előtt álló " # # " jelzi.

c/ A QUEL utasitásokban mindenhol szerepelhetnek C változók, de ezek deklarációját is "# # "-nek kell megelőzni.

d/ Mig a RETRIEVE utasitás /1.2.4./ a QUEL-ben egész relációkkal dolgozik, az EQUEL-ben egy hivásra csak egy sort ad vissza. Egymás utáni hivásai soronként mennek végig azon a relación, amit a megfelelő QUEL hivás egyszerre adna visszza. Ennek szintaktikája:

```
RETRIEVE /változólista/
         WHERE /feltételek/

##{
C-blokk
##}
```

A C-blokk minden RETRIEVE után egyszer hajtódik végre. Ez végzi az aktuális sor feldolgozását.

a/ Irjuk ki néhány dolgozó nevét és alapbérét! A kiirandó dolgozók törzsszámait a C program olvassa be.

```
main( )
{
   ##char NÉV[20];
   ##char TÖRZS[5];
   ##int ALAP;
while (READ(TÖRZS))
      {
   # # RANGE OF D IS DOLGOZÓ'
   # # RETRIEVE NÉV=D.NÉV,ALAP=D.ALAPBÉR
   # # WHERE D.TÖRZSSZÁM=TÖRZS
            ##{
            PRINT NÉV," FIZETÉSE ",ALAP," FORINT" ;
            ##}
      }
}
```

A NÉV, és ALAP változók a dolgozók neveit, alapbéreit tartalmazzák majd a lekérdezés eredményeképpen. A TÖRZS-be olvassuk be a törzsszámokat, igy ez szerepel majd a feltételben. A "while" utasitás vezérli a programot, sorban olvasva a törzsszámokat, addig mig el nem fogynak. A ciklus két utasitást - RETRIEVE és PRINT - tartalmaz. A RETRIEVE mindig a frissen beolvasott törzsszámra keres, - ezt a TÖRZS-ből veszi /az adatbázisban ténylegesen kereső rutin már minden alkalommal D.TÖRZSSZÁM=konstans tipusu feltételt kap/- és az eredményt a NÉV és az ALAP változókba rakja. A PRINT már csak megfelelő formátumban kiir.

Erős lehetőség, hogy a QUEL utasitásokban maguk az adatbázis reláció és oszlopnevek is változók lehetnek, módot adva ezzel pl. tetszőleges önálló adatkezelő nyelv interface-ének megprogramozására EQUEL-ben /SQL interface az INGRES-hez/. Erre is egy egyszerü példa:

b/ Valamilyen reláció egyik oszlopának beolvasott összes értékére hivjunk meg egy PROCESS nevü rutint. Legyen ennek paramétere a másik oszlopban lévő megfelelő érték!

```
main( )
{
## int ÉRTÉK1,ÉRTÉK2;
## char RELÁCIÓ[13],OSZLOP1[ 13],OSZLOP2 [13];

READ(RELÁCIÓ);
READ(OSZLOP1);
READ(OSZLOP2);
## RANGE OF R IS RELÁCIÓ
while(READ(ÉRTÉK1 ))
      {
 ## RETRIEVE ÉRTÉK2=R.OSZLOP2
 ## WHERE R.OSZLOP1=ÉRTÉK1
      ##{
      PROCESS(ÉRTÉK2);
      ##}
  }
}
```

A lekérdezni kivánt reláció nevét RELÁCIÓ-ba, a két oszlopét OSZLOP1 és OSZLOP2-be olvassuk, az első oszlop adott értéke ÉRTÉK1, a másodiké - ezt a RETRIEVE olvassa be - ÉRTÉK2. A RANGE utasitás szokásos helye a RETRIEVE előtt lenne, - oda is helyezhettük volna - de az INGRES emlékezik a definicióra, s mindaddig mig R-t ujra nem definiáljuk, az a RELÁCIÓ változóban lévő relációnéven lesz meghatározva.[STON 76]

Az IBM SQL/DS gazdanyelvként PL/1-et és COBOL-t használ, mi a példákban PL/1-gyel dolgozunk. /Mellesleg az INGRES gazdanyelvként a C-n kivül FORTRAN-t és Pascal-t, az ORACLE COBOL és PL/1 mellett Assembler-t, FORTRAN-t, C-t és Pascal-t is kinál [DIEC 81]./

A PL/1-SQL tetszőleges PL/1 és SQL utasitásokból áll. Az SQL utasitást "$" előzi meg. Ugyancsak $ kell, hogy legyen az első karaktere az SQL utasitásokban használt PL/1 változóknak. A beépitett SQL kiegészül néhány speciális utasitással. Ezek közül néhány:

```
$LET  C1  BE
  SELECT NÉV INTO $X
  FROM DOLGOZÓ
  WHERE BESOROLÁS=$Z;
```

Ez az utasitás C1-et deklarálja mint lekérdezést, mely a Dolgozó relációból azoknak a nevét, akiknek besorolása megegyezik a $Z változó aktuális értékével a $X változóban adja vissza. Az utasitás hatására semmi sem történik még, az SQL/DS tudomásul veszi a deklarációt. A

```
$ OPEN C1;
```

megnyitja a lekérdezést. Ez a felhasználó számára annyit jelent, hogy C1 utasitás input - tehát felhasználó szolgáltatta - értékeit megjegyzi a rendszer, és azokon többet változtatni nem lehet. Ez lényeges eltérés az EQUEL

filozófiájától, mely a C változóknak mindig az aktuális értékét vette. Egy-egy sor beolvasása a

```
$ FETCH C1;
```

hatására történik. A lekérdezés lezárását

```
$ CLOSE C1;
```

utasitás végzi.

c/ Irjuk ki néhány dolgozó nevét és alapbérét! A kiirandó dolgozók törzsszámát a PL/1 program olvassa be.

```
KIIR:PROC;
     DCL
     EOF BIT(1) INIT ('0' B),
        $NÉV CHAR(20),
        $TÖRZS CHAR(5),
        $ALAP BIN FIXED;
          < RENDSZER INTERFACE DEKLARÁLÁS >
    $LET C BE
      SELECT NÉV,ALAPBÉR INTO $NÉV,$ALAP
      FROM DOLGOZÓ
      WHERE TÖRZSSZÁM=$TÖRZS;
    ON ENDFILE(SYSIN) EOF='1'B;
    GET LIST($TÖRZS);
    DO WHILE (¬EOF);
      $OPEN C ;
      IF SYR_CODE¬=0  THEN CALL BAJ('OPEN');
      $FETCH C;
      IF SYR_CODE¬=0  THEN CALL BAJ('FETCH');
      $CLOSE C;
      IF SYR_CODE¬=0  THEN CALL BAJ('CLOSE');
      IF SYR_CODE=0  THEN
         PUT LIST($NEV,' FIZETÉSE ',$ALAP,' FORINT');
      GET LIST($TÖRZS);
    END;
 END KIIR;
```

A rutin lényegében véve az a/-ban látható C program PL/1 megfelelője. Két megjegyzést teszünk:

A $OPEN és $CLOSE utasitások nem helyezhetők a cikluson kivülre. Ekkor ugyanis a program mindig az elsőnek beolvasott törzsszámu dolgozó adatait irná ki, hiszen az input változók értékeit $OPEN-nél egyszer s mindenkorra megjegyzi.

Szemben a EQUEL-lel itt jól láthatóan van hibakezelés. /Az EQUEL-ben is van, mint [STON 76] egy mellékmondatából kiderül, de a cikkben közölt példákból a részletek nem derülnek ki./ A SYR_CODE változóban adja vissza a rendszer a hibakódot. Ezt - jelen programban kicsit furcsán - a BAJ rutin feldolgozhatja, és megteheti a megfelelő intézkedéseket.

d/ Valamilyen reláció egyik oszlopának beolvasott összes értékére hivjunk meg egy PROCESS nevü rutint! Legyen ennek paramétere a másik oszlopban lévő megfelelő érték!

Ennek a feladatnak a megoldásához meg kell ismerkednünk a $PREPARE és $EXECUTE utasitásokkal. A $PREPARE formátuma:

$PREPARE ⟨utasitásnév⟩ AS ⟨karaktersorozat⟩;

Például a

$PREPARE C AS SZÖVEG;

értesiti az SQL/DS-t, hogy a SZÖVEG nevü változóban SQL/DS utasitás lesz, és azt végre szándékozunk hajtatni. A végrehajtás

$EXECUTE ⟨utasitásnév⟩ USING ⟨változósorozat⟩

hatására következik be. Pl. a

```
SZÖVEG='UPDATE DOLGOZÓ SET ALAPBÉR=? WHERE NÉV=?';
$EXECUTE C USING $UJBÉR, $NÉV
```

utasitások a

$UPDATE DOLGOZÓ SET ALAPBÉR=$UJBÉR, WHERE NÉV=$NÉV;

utasitások végrehajtását eredményezik.

Most pedig nézzük a feladatot!

```
PRÓBA:PROC;
     DCL (SÉRTÉK1,SÉRTÉK2) BIN FIXED(31),
          EOF BIT(1) INIT('0'B),
          UTASITÁS CHAR(100) VARYING;
          <RENDSZER INTERFACE DEKLARÁCIÓ>
          SPREPARE U AS UTASITÁS;
          IF SYR_CODE ¬=0  THEN CALL BAJ;
     GET LIST(UTASITÁS);
     ON ENDFILE(SYSIN)  EOF='1'B;
     GET LIST(SÉRTÉK1);
     DO WHILE(¬ EOF);
       SEXECUTE U USING SÉRTÉK2, SÉRTÉK1;
       IF SYR_CODE ¬=0 THEN CALL BAJ;
       ELSE CALL PROCESS(SÉRTÉK2);
       GET LIST(SÉRTÉK1);
     END;
END PRÓBA;
```

A b/-ben tárgyalt C programmal szemben itt a teljes utasitást beolvassuk /ott csak a lekérdezni kivánt reláció és a kérdésben szereplő oszlopok neveit kellett/. Az utasitást tartalmazó kártya:

SELECT ?=<oszlopnév2> FROM <relációnév> WHERE<oszlopnév1>=?

Itt a <...> a konkrét neveket jelölik. A $EXECUTE a "?"-ek helyére"irja" a $ÉRTÉK2-t és $ÉRTÉK1-et igy hajtva végre a lekérdezést. Meg kivánjuk jegyezni, hogy természetesen itt is megoldható lett volna, hogy csak a neveket, és nem a teljes utasitást kelljen beolvasni /a PL/1 azok alapján összeszerkeszthette volna a megfelelő karaktersorozatot/.

Figyelemre méltó, hogy a $EXECUTE biztositotta parametrizálási lehetőség erősebb, mint amit az EQUEL lehetővé tesz. Az EQUEL-ben ugyanis az utasitás neve nem parametrizálható,

annak explicite szerepelnie kell [STON 76] a $EXECUTE viszont csak egy karaktersorozat tipusu változóval dolgozik, amibe bármi irható, tehát az utasitás fajtája is futás közben dönthető el. [CHAM 81]

1.3.2. Adatkezelő-programozási nyelvek

Az előző paragrafusban vizsgált két beépülő nyelv utasitásai szintaktikájukban, koncepciójukban teljesen elütnek a gazdanyelv stilusától. A két nyelv közötti kapcsolatot a közösen használt gazdanyelvi változók teremtik meg, ezeken mint "paramétereken" keresztül kommunikál a két nyelv. Az elkülönülést formai jegyek /##, $/ is hangsulyozzák.

A most ismertetendő nyelvek célja éppen ellenkező: alkotóik elképzelése szerint az adatkezelő rész elválaszthatatlan egységet alkot a befogadó programozási nyelvvel; azzal nem csak változókon át kommunikál, de utasitásai felépitésükben, koncepciójukban, szintaktikájukban a programozási nyelv szerkezetére, fogalmaira, stilusára támaszkodnak.

Ezt a lehetőséget a programozási nyelvek "absztrakt adattipus" fogalma teremti meg. Az absztrakt adattipus tulajdonképpen nem más, mint adatok együttese és a rajtuk definiált müveletek. A klasszikus példa a verem, ahol a definiáló adatok a verembe helyezhető adatok tipusa, és egy Boole-változó, mely jelzi, hogy van-e elem a veremben, és a müveletek: elem kivétele a veremből, elem verembe illesztése, a verem ürességét eldöntő függvény.

Absztrakt tipusu változókat gyakorlatilag bármelyik, szubrutinhivási művelettel rendelkező programozási nyelvben lehet implementálni. A verem például egy tömbbel és egy egész értékü változóval implementálható, és egyszerü szubrutinok a müveletek. Az adatok és a müveletek leirására alkalmas eszközökön kivül azonban ahhoz, hogy absztrakt

adattipusról beszéljunk szükség van legalább még egy további eszközre is, az adott tipusu változók generálásának lehetőségére. Példánkban ez azt jelenti, hogy miután definiáltuk a "verem" absztrakt adattipust, a nyelv fogadja el az A,B,C STACK stb. változókat mint "verem" tipusunak deklaráltakat /és generálja automatikusan az implementálásukhoz szükséges változókat ugy, ahogy a tipus definiciójában meghagytuk/.

Az adatkezelő-programozási nyelvekben a relációk absztrakt adattipusok lesznek. Nem a "reláció", mint olyan, hanem minden reláció egyfajta tipust alkot. A "reláció" fogalma tipusképző - a nyelvben definiált - "mód"-ként jelentkezik /mint a PASCAL array,record, stb./. Ennek segitségével definiáljuk a Dolgozó relációt: /Az alábbi az [ALAG 81] cikk javasolta konstrukció szellemében készült, de a PASCAL/R rendszerben [SCHM 77], vagy a MODULA /R-ben [REIM 83] ugyanigy megy a reláció definiálása/.

```
type  dolgozótipus = record  törzsszám: integer;
                             név: string;
                             részlegkód: integer;
                             besorolás: string;
                             főnök: integer;
                             alapbér: integer;
                     end;
var dolgozó  : relation of dolgozótipus;
```

Az első tipusdefinició a reláció egy sorát adja meg, PASCAL rekord formájában. A második definició a dolgozó nevü változót a "relation" mód segitségével relációként generálja. /Ezzel ekvivalens lett volna a:

```
type dolgozóreláció = relation of  dolgozótipus;
var dolgozó: dolgozóreláció;
```

felirás/.

[SCHM 77]és [REIM 83] javaslataiban kulcs megadása is szerepel;

type dolgozóreláció = relation ⟨törzsszám⟩ of dolgozótipus;

A "törzsszám" kulcsnak azonositónak kell lennie, és a reláció soronkénti olvasásánál növekvő értékei szerinti sorrendben kapjuk meg a sorokat.

[REIM 83] előirja, hogy a relációsornak megfelelő rekordban csak strukturálatlan adattipusok szerepelnek. Ez valóban összhangban van az első normálforma /a reláció/ definiciójával /O.1.1./. [SCHM 77] erre nem tartalmaz utalást, [ALAG 81] pedig - később látni fogjuk - pointert is engedélyez oszloptipusnak.

A reláció adatainak definiálása után lássuk a relációval végezhető müveleteket! [SCHM 77] és [ALAG 81] nyomán:

```
var d: dolgozótipus;
d.törzsszám=9826;
d.név='Kiss Pál';
d.részlegkód=52;
dolgozó=dolgozó+[d];
```

A programrészletben az egyik müvelet az elemi relációképző, a "[ ]". Ez a "relációt alkotó sor" tipusu változóból egy egyetlen sorból álló relációt készit /a PASCAL halmazképző mintájára/ A "+" relációkat egyesit, /a "-" különbséget, a "*" metszetet képez/. Mindez a PASCAL halmaz müveleteinek megfelelően történik, tehát a reláció mindeddig ekvivalens a halmazzal /leszámitva a kulcs megadását/.

A következő konstrukció az általános relációképző. [SCHM 77]és [ALAG 81] alapján mutatjuk be, megemlitve, hogy a MODULA/R hasonló konstrukciója ettől csak szintaktikájában különböző [REBS 83]:

Válogassuk ki a programozókat a dolgozók közül, és

helyezzük el nevüket, alapbérüket, és részlegük cimét a Programozási Osztály relációban /ld. 1.2.1.c-1.2.5.c./!

```
var  programozási_osztály: relation of  név: string;
                                        alapbér:integer;
                                        cim:string;
programozási_osztály:=[each d.név, d.alapbér, r.cim
                       for  d,r
                       in   dolgozó,részleg
                       where  d. besorolás='programozó' and
                              d.részlegkód=r.részlegkód ]
```

A relációképző valóban relációs müvelet: relációkon definiált, és a| müvelet eredménye ugyancsak reláció.

A másik müvelet, amit bemutatunk szintén teljes relációkkal dolgozik, ebben hasonlit az előzőhöz. A kettő közötti lényeges különbség,hogy mig a relációképző eredménye uj reláció, addig a "foreach" utasitás eredménye tetszőlegesen - a PASCAL keretein belül-előirható /MODULA/R-ben ilyen utasitás nem létezik/:

Adjunk minden programozónak 10 % béremelést! /1.2.1.j-1.2.5.j/

```
foreach
in dolgozó
where  d.besorolás='programozó'
do d. alapbér=1.1*d.alapbér
```

A "foreach" több relációval is képes egyszerre dolgozni, és a do után bármi állhat, amit a PASCAL eltür.

A "foreach" természetesen képes a relációképző helyettesitésére /a do után elemi relációképzőt irva/, mégsem mondható hogy feleslegessé teszi azt. Ezen az alapon ugyanis maga a "foreach" is felesleges, mert a PASCAL-ban van ciklusképzés

/kiegészitésképpen | mindössze három egyszerü, relációsorokon értelmezett beépitett függvényre van szükség - a konstrukciót [SCHM 77] tartalmazza/. Éppen ezek az utasitások azok, melyek a PASCAL-ból PASCAL/R-t csinálnak elegáns PASCAL stilusu relációs müveleteikkel. /Ugy az EQUEL, mint a PL/1-SQL soronként dolgozza fel a relációt/.

A bevezetett müveletek erejét a következő példával /1.2.1.e.-1.2.5.e/ jellemezzük. Képezzünk relációt olyan szállitókból, akik az 50-es kódu részleg által felhasznált összes cikkszámot szállitják!

```
[each s.szállitó
 for s
 in szállitás
 where [each sx.cikkszám
        for sx
        in szállitás
        where s.szállitó=s.xszá±litó]
        >=
       [each c.cikkszám
        for c
        in felhasználás
        where c.részelgkód=50]]
```

A megoldás logikája azonos az SQL/DS-évei /1.2.3.e/ kihasználva a PASCAL halmazmüveletét />=/.

A két konstrukció /relációképző, "foreach"/ nyilván a relációalgebra valamennyi müveletét képes előállitani /PASCAL utasitások segitségével/. [SCHM 77] és a MODULA/R azonban ezek mellé még kvantorokat is alkalmaz, [ALAG 81] hivatkozva arra, hogy a nyelv nélkülük is relációsan teljes az egyszerüség kedvéért elveti őket.

[ALAG 81] indexet is definiál a relációhoz. Ehhez semmilyen uj konstrukcióra nincs szükség. Az index nem más, mint egy bináris reláció, melynek egyik oszlopában az indexelt mező különböző értékei, a másodikban pedig a megfelelő sorokra mutató pointerek /azért ezek csak formálisan egyeznek meg a PASCAL pointerekkel/ állnak. A Dolgozó reláció Név szerinti indextáblázata például:

```
var névindex: relation of record  név: string
                                  ref: ↑dolgozótipus
```

nagyon szemléletes. Az indexet a "createimage" eljárás hivása generálná. A pointerek csak olvashatóak lennének, változtatni csak az adatbáziskezelő rendszer változtathaja őket.

A cikk - helyes óvatossággal - külön szintre helyezi az indexeket. Az egyszerü felhasználó ezt a konstrukciót jobb, ha nem használja ez a "második szint" lehetősége. Algoritmus készithető - [ASTR 75] nyomán - "első szinten" indexek használata nélkül irt programok index segitségével történő gyorsitására.

A dolgozat egy harmadik szintet is definiál. Ez a relációkat soronként manipuláló rutinok halmaza.

2. IMPLEMENTÁCIÓ

Egy absztrakt adattipus implementálása tulajdonképpen egy leképezés létrehozását jelenti. Ez az adattipus felhasználójának a fogalmait - adatokat, müveleteket - a megvalósitó adatokra és müveletekre képezi le.

Ha verem tipusu változót akarunk FORTRAN-ban implementálni, a triviális megoldás egy tömb kijelölése, ahol a verem elemeit elhelyezzük, és egy egész értékü változó, mely a veremben aktuálisan tárolt elemek számát jelzi. A felhasználói interface itt nyilván a verem a rajta végezhető müveletekkel, a közeg, melyben implementáljuk a "FORTRAN'gép" vagyis a számitógép, ahogyan azt a FORTRAN-on keresztül látjuk.

Az "absztrakt adattipust", melynek implementációjával ez a rész foglalkozik, az 1. részben irtuk le. Persze ahány rendszer, annyi tipus - a verem esetében is több lehetőség van pl. a tulcsordulás kezelésére - de valamennyinél - amint ezt láttuk - az alapmodell a reláció és a rajta definiált müveletek /O.1./.

Összehasonlitva a többi "nagy" adatmodellel, a relációs implementálása nehéz feladatnak tünik. Jellemző, hogy mig a CODASYL DBTG 1971. évi jelentése után 1-2 évvel már forgalmaztak olyan hosszutávon is nagysikerü rendszereket, mint az IDMS vagy a DMS/1100 addig [KIM 79] - a szép számu kisérleti rendszer mellett - csak két olyan rendszert emlit, mely forgalmazási céllal készült /MAGNUM és QBE/, de [SNYD 82] áttekintése az 1981-es adatbáziskezelő rendszerekről ezeket már nem emliti. Ezzel együtt éppen 1981 volt a relációs adatbáziskezelők megjelenésének éve a piacon /[SNYD 82] / - 11 évvel Codd első cikke után.

/Léteznek rendszerek, melyekben csak a második leképzés definiált [KISS 83]. Ezek vagy önállóan, az adatbázis assembleren, vagy - az első leképzés megvalósitásával -

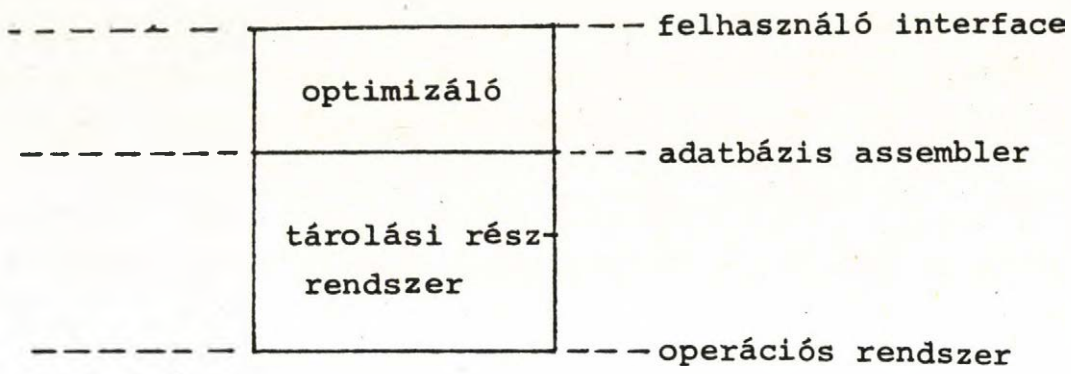
tetszés szerinti, erre épithető felhasználói interface-n keresztül használhatók./

A relációs modell triviális implementációja a szekvenciális file /ld. O.l.l./. Ez a legegyszerübb adatszervezés felel meg a modell logikai egyszerüségének, egyes fogalmaknak direkt megfelelője van /sor-rekord, reláció-file, stb./. A probléma ezzel a megoldással az, hogy gyakorlati célokra tul lassu. Ennek ellenére sok mikrogépes rendszer használja a szekvenciális file implementációt /dBASE II, MRDBS, stb./

Ebben a részben a relációs adatbáziskezelő rendszerek implementálásáról lesz szó. Az első fejezetben a rendszerek - a software - felépitéséről, a másodikban és a harmadikban két - általában külön interface-szel elválasztott - összetevőjükről, az adattárolási rendszerről és a felhasználói nyelvet ennek a rendszernek az interface-ére lefordító optimizáló programról.

2.1. A rendszer architekturája

A felhasználói interface-t implementáló leképezést általános gyakorlat szerint két, egymást követő részleképezés alkotja /ld. 7. ábra/. Első lépés a felhasználói igény megértése, és az igény kielégitéséhez szükséges müveletsor megtervezése. Ezt a müveletsort belső nyelvre - a reláció soronkénti elérését lehetővé tevő, rutinhivásokból álló "adatbázis assembler"-re forditja az első leképzés. A második leképzés hajtja végre a müveleteket. Ennek "felhasználói interface"-e az adatbázis assembler, és az implementáció közege általában az operációs rendszer file-kezelője.



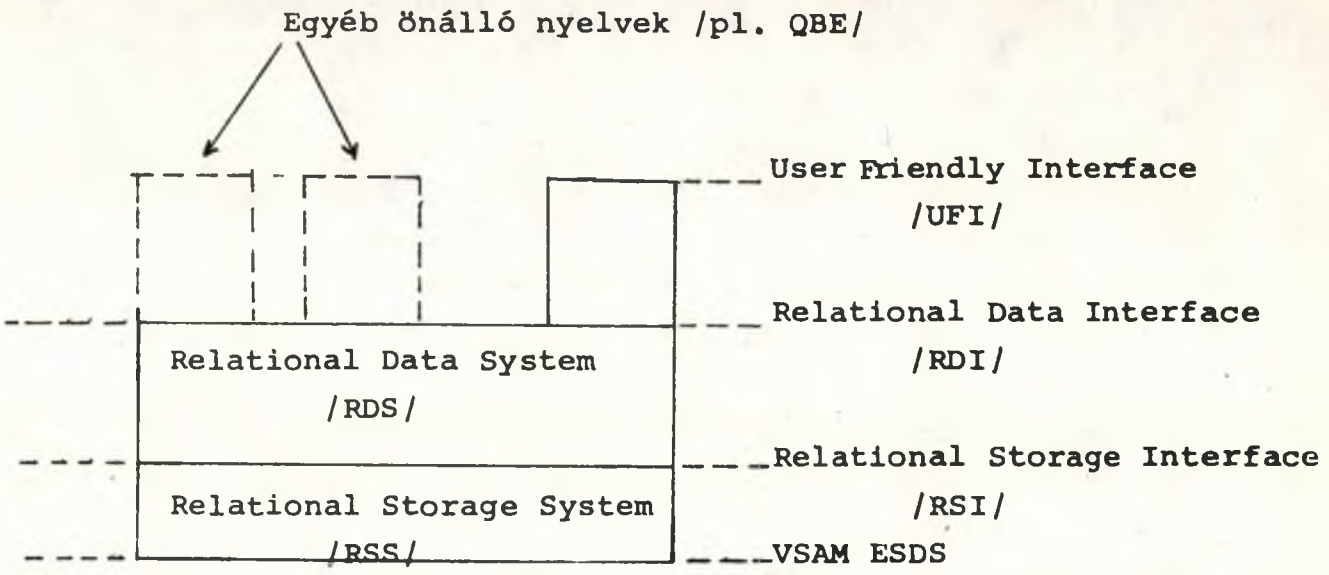
7. ábra

A két lépésnek megfelelően általában az egyes rendszerek is két részre oszthatók. Az első leképezést megvalósitó részt "Optimizáló"-nak /Optimizer/ fogjuk nevezni. A név legfontosabb feladatából adódik: a felhasználói igény lehető leghatékonyabban történő kielégitésének módját kell megtalálnia.

A második leképzés a tárolási részrendszer. Ennek feladata az adatok fizikai elérése - rendszerint az operációs rendszer file-jaival, de ennek megkerülésével is lehetséges - az indexek automatikus karbantartása, használata stb.

2.1.1. SQL/DS

A 7. ábrán látható általános felépités az SQL/DS /System R/ esetén a következő módon /8. ábra/ realizálódik



8. ábra

Alulról fölfelé haladva: a tárolási részrendszer /RSS/ VSAM ESDS szervezési módu file-ra támaszkodik. Ez igen lényeges döntés: azt jelenti, hogy a rendszer lényegében nem használja az operációs rendszer biztositotta file-szervezési lehetőségeket. A VSAM ESDS /Virtual Sequential Access Method - Entry Sequenced Data Set/ lényegében nem több, mint a blokk sorszáma szerinti közvetlen elérést támogató file-szervezési mód. Azt tehát, hogy egy relációs sor hol található, az RSS-nek megának kell tudnia, semmiféle operációs rendszer lehetőség /pl. egy index-szekvenciális file index-táblázata/ ebben nem segiti. Szabadságot élvez viszont természetesen a blokkokon belüli szervezésben. Egy operációs rendszer feletti szint ugy dönt arról, hogy melyik adatát hova /az operációs rendszer szempontjából melyik blokkba/ helyezi, ahogy akar.

Az RSS feladata a külső tárolóterülettel való gazdálkodás, az indexek karbantartása, az elérési utak realizálása. Emelett ezen a szinten történik a konkurrens hozzáférés vezérlése, a mentés és visszaállitás.

Az RSS bemenő nyelvét az "adatbázis assemblert" RSI-nek nevezi az SQL/DS. Ez a reláció egyszerü, soronkénti elérését biztositó utasitásokon kivül adatdefiniciós, visszállitó, tranzakció-kezelő utasitásokat is tartalmaz. Az RSI-vel mint felhasználói interface-szel az RSS komplett adatbáziskezelő rendszert alkot. Ez persze egy igen kényelmetlen rendszer, a felhasználói interface-nek ismernie kell olyan fogalmakat, mint "szegmens", tudnia kell arról, hogy létezik-e index valamilyen relációra, vagy van-e pointeres kapcsolat két reláció között, stb.

Ennek az adatbáziskezelő rendszernek a felhasználója az RDS /a 7. ábrán Optimizáló/. A feladata nyilvánvaló - a felhasználó nyelv utasitásait "forditja" RSI programmá. Ehhez fel kell ismernie azokat, és a belső táblázatok alapján megkeresnie a legkedvezőbb "elérési utat". Ez annyit jelent, hogy ő az, aki a különböző tulajdonságu indexek, kapcsolatok szövevényében eligazodva eldönti, hogy egy lekérdezés megválaszolásához pl. melyik oszlop szerinti indexet érdemes használni, vagy egy illesztést milyen algoritmussal érdemes csinálni. /Innen a neve: Optimizáló/.

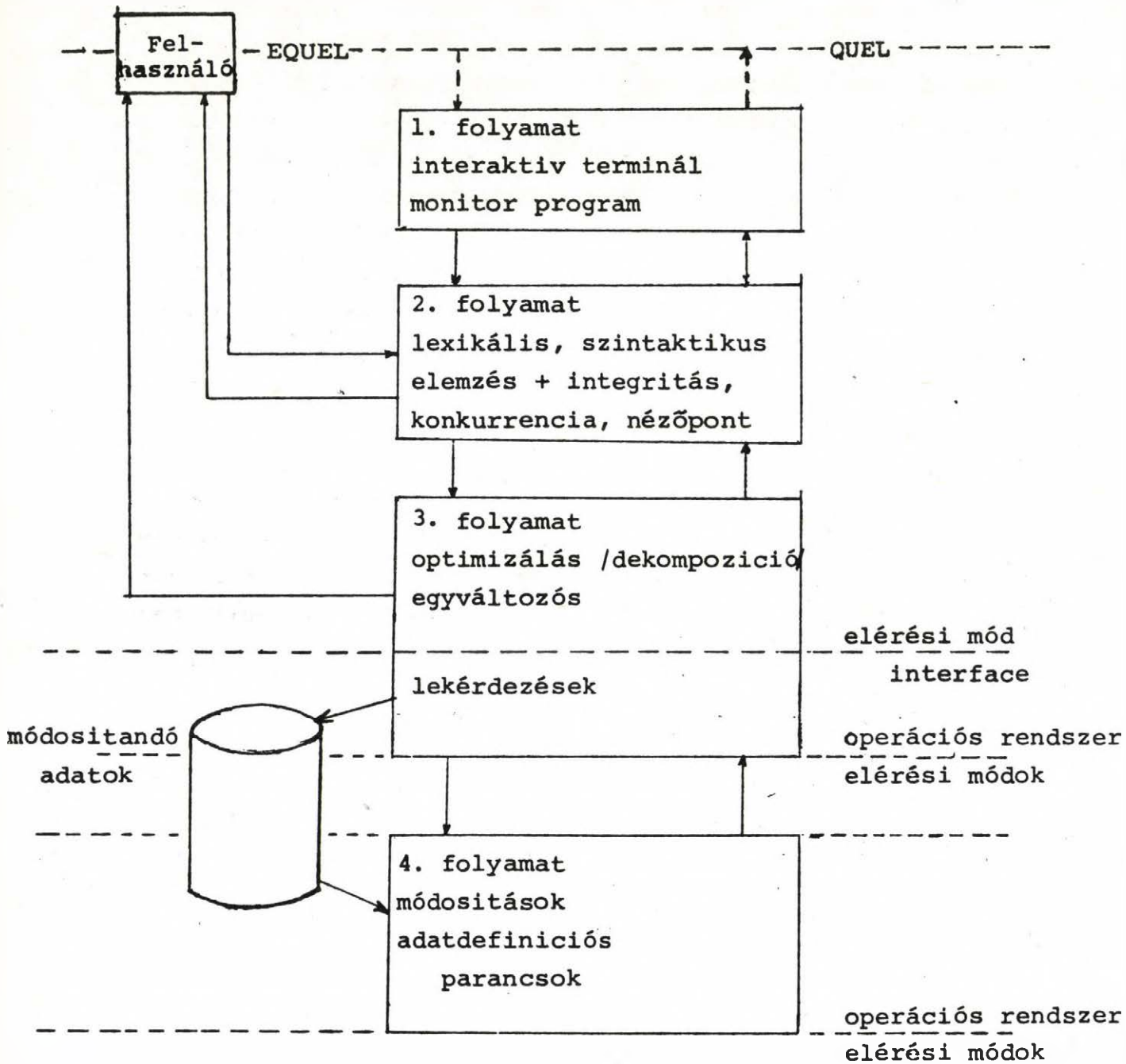
Az RDS bemeneti nyelve az RDI. Ez nem más, mint programozási nyelvbe beépülő SQL /leirását ld. 1.3.1./, tehát ez már a felhasználó számára hozzáférhető interface.

Szellemes az önálló SQL nyelvü interface megoldása. /Ezt az ábrán UFI jelöli/. A terminál előtt ülő felhasználó egy PL/1 - SQL programmal kommunikál. A párbeszédet ez folytatja a felhasználóval, ez adja a hibaüzeneteket, stb. A program - éppen ugy, mint bármely más közönséges felhasználói program - az SQL/DS-sel a beágyazott SQL

utasitásokon keresztül kerül kapcsolatba. Itt használják ki a rendszer alkotói a PREPARE és EXECUTE biztositotta lehetőségeket - vagy ennek a megoldásnak a kedvéért került a nyelvbe a PREPARE és az EXECUTE? - ugyanis ezek teszik lehetővé, hogy az UFI tetszőleges relációkra vonatkozó tetszőleges parancsot végre tudjon hajtani az RDS-sel /1.3.1./. A dologban az a szép, hogy nem kell külön kezelni az önálló és a programozási nyelvből érkező igényeket, az SQL felhasználó a terminál előtt egy PL/1 programot futtat - noha ezt nem tudja - és igy az ő ad hoc igénye az RDS-hez már az UFI-n, vagyis a szokásos módon, egy PL/1-SQL programon keresztül érkezik.

Ugyanigy lehet más önálló relációs nyelvekhez /pl. QBE/ interface programot készitени. /Ez is elegáns - a felhasználó - ha nem tetszik neki az SQL -saját maga tervezhet, és felhasználói eszközökkel implementálhat egy szimpatikusabb nyelvet a rendszer fölé./ A többi önálló nyelvet szimbolizáló két téglalapot szaggatott vonalból rajzoltuk - jelezve, hogy ilyenek létezéséről nem tudunk. [ASTR 76, BLAS 81]

2.1.2. INGRES



9. ábra

A 9. ábra az INGRES szerkezetét mutatja be. Az ábrán látható téglalapok UNIX folyamatokat, a nyilak pedig csöveket szimbolizálnak. Ezeket a fogalmakat magyarázzuk el először:

A folyamat /process/ a UNIX operációs rendszerben egy program által cimezhető /virtuális/ memóriadarabot jelent. Ennek maximális nagysága PDP-11/40-en 64K, 11/45-ön vagy 11/70-en 128K /byte/. Egy felhasználói program több folyamatból állhat. Ezek a folyamatok egymástól függetlenül tevékenykednek, és szinkronizációs utasitásokkal, file-okon és csöveken keresztül tartanak egymással kapcsolatot.

A cső /pipe/ egy egyirányu kommunikációs csatorna folyamatok között. Egy folyamat másiknak küldött üzenetei a folyamatok szempontjából olyanok, mintha egy file-on keresztül tartanák a kapcsolatot, melyre az üzenő folyamat irni tud, a vevő pedig leolvashatja róla az üzenetet. Valójában a UNIX szervezi meg a küldemény célba jutását ennél hatékonyabb módon, és ő gondoskodik az adás és a vétel szinkronizálásáról.

Lássuk most tehát a rendszer müködését! Mint az ábra tetejéről látható, aszerint, hogy a QUEL-ből /önálló adatkezelő nyelv/ vagy az EQUEL-ből használjuk a rendszert, változik a folyamatok kapcsolata. A QUEL felhasználó parancsait interaktiv terminálkezelő monitor fogadja, és a rendszer üzeneteit is ez továbbitja /a QUEL interface és a monitor közötti nyil az ábrán persze nem csövet, hanem terminálra irást ill. onnan olvasást jelent/. A program munkaterületre olvassa a QUEL utasitásokat, igy a felhasználónak lehetősége van ott felépiteni, módositani és tárolni az utasitásait. A monitor paraméterezhető makro lehetőséget is tartalmaz [DIEC 81]. Kihasználva a UNIX standard lehetőségeit a monitor inputját file-ként

megadva nem interaktiv módon is lehet QUEL parancssorozatot futtatni /ilyen lehetőség más rendszerekben is van - ld. 1.2.6./.

Az EQUEL-t használva a felhasználó programja lép a monitor folyamat helyébe a strukturában. Ilyenkor a program megkerüli a monitort, és csövön keresztül közvetlenül a szintaktikus elemzőnek /2. folyamat/ küldi a QUEL parancsokat /az előforditó által az EQUEL utasitások helyére programba illesztett INGRES hivások paraméterei ezek a parancsok - ld. 1.3.1./. Az eredményről a 2. folyamat eredménykódot küld, lekérdezésnél pedig a sorokat a 3. folyamat küldi vissza.

[STON 76] két érvet hoz fel a kettős folyamatstruktura mellett, megemlitve, hogy a monitort meg lehetett volna irni EQUEL-ben és közönséges EQUEL programként futhatna. /Ez az SQL/DS megoldás - ld. 2.1.1./. Először is a monitor még az EQUEL létezése előtt készült el, és nem tekintették sürgősnek az átirását /nem tudjuk, hogy 1976 óta változott-e a helyzet/. A második érv szerint a futást valamennyivel lassitaná, ha a 3. folyamat ahelyett, hogy közvetlen csövön át elküldené a felhasználói programnak a megtalált sorokat, a 2. folyamaton keresztül adná át az elsőnek.

A 2. folyamat a szintaktikus elemző. Feladata még emellett a konkurrens hozzáférés megszervezése és a nézőpont, valamint az integritási feltételek realizálása. A nézőpontot - ugyanaz, mint az SQL/DS-é /ld.1.1.1./ - 1976-ban még nem implementálták /emiatt maradt ki 1.1.2.-ből/, de [KIM 81] már müködőként emliti.

A lekérdezés processzor a 3. folyamat. A 7. ábra Optimizálójának a ténylegesen csak legkedvezőbb elérési ut kiválasztásával foglalkozó részét /a szintaktikus elemzés a 2. folyamatban megvolt/ és a tárolási rész-

rendszer programjainak a lekérdezéshez szükséges részét tartalmazza. A két részrendszer érintkező felülete - az INGRES elérési mód /access method/ interface-nek nevezi - itt is pontosan definiált, soronkénti elérést biztositó rutinokból áll. Látni fogjuk hogy ez a folyamat nem véletlenül tartalmazza mind a két részrendszer legfontosabb elemeit: egy-egy részlekérdezés /a több relációt érintő lekérdezés egy relációs részlekérdezésekre bomlik az INGRES-ben / eredménye döntően befolyásolja a további elérési utat.

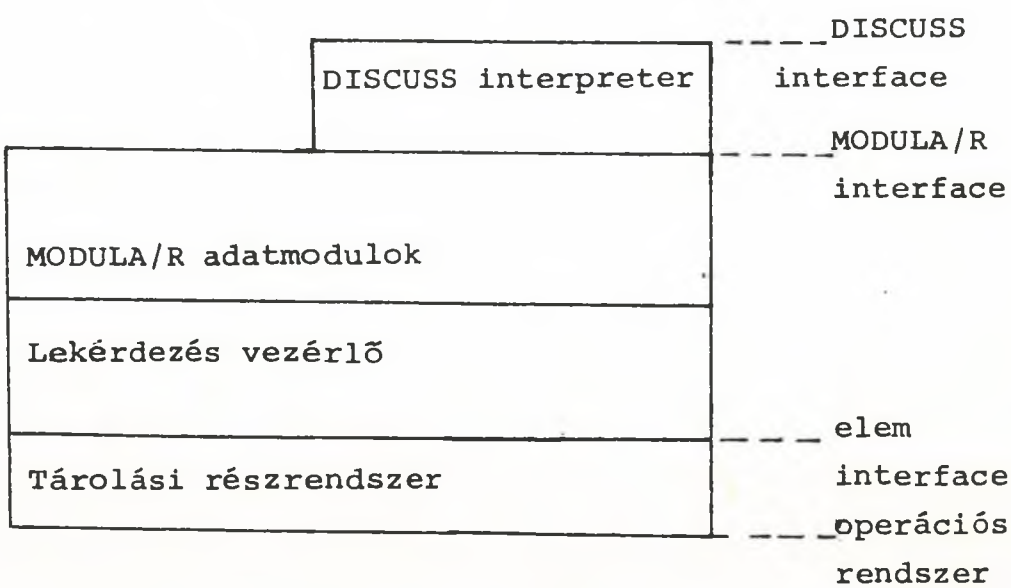
A módositó /törlés, beillesztés, felujitás/ és adatdefiniciós utasitásokat realizálja a 4. folyamat. Ez a folyamat 8 overlay-ből áll. A módositandó adatok kiválogatása - lekérdezésként - a 3. folyamatban történik. Az adatok speciális file-ban kerülnek át a 4. folyamathoz /9. ábra/. Emellett a megoldás - a módositandó adatok előválogatása - mellett [STON 76] több érvet is felsorakoztat: három - elég szélsőséges - példát hoz olyan felujitásra, ahol a módositás soronkénti előrehaladása - több sort érintő felujitásokról van szó - befolyásolja a még hátralévő módositásokat, ugyanis a felujitás feltételeinek vagy a már felujitott sorok még egyszer, vagy miattuk más sorok eleget fognak tenni, s igy indokolatlanul változnak meg sorok. Negyedik érve a visszaállithatóságra vonatkozik. [STON 76] megjegyzi, hogy ez a megoldás /ő késleltett módositásnak - deferred update - nevezi/ igen költséges és olyan felhasználót emlit, aki inkább vállalta volna, hogy nem adhat meg bármilyen módositást a rendszernek, ha azok már a 3. folyamatban lezajlanak.

A folyamat-struktura kialakitásánál az elsődleges szempont a 64 K-s memóriakorlát volt. A rendszer tervezője nagyszerü összefoglaló - értékelő [STON 80] cikkében

emliti, hogy a folyamatszerkezet igen jelentős sebességcsökkenéshez vezetett. /Ebben a cikkben már egy 5 folyamatból álló rendszerről ir, ahol a vezérlésnek 8 folyamaton kell áthaladnia, ami 8 UNIX scheduler hivást, csőlétesitést jelent/. Emliti, hogy talán helyesebb lett volna kihasználni a PDP-11/70 biztositotta 128 K-t - ezt a kisebb gépekkel dolgozó felhasználók kedvéért nem tették meg - de igazi megoldásként egy 32 bites mikroprocesszorral müködő gép használatát javasolja. Az INGRES-t VAX-11-en forgalmazzák. A rendszer 180 K byte rendszerterületet + felhasználónként 90 K byte memóriát igényel. [DIEC 81] Ez a rendszer egyetlen folyamatból áll [ALLM 82] [STON 76].

2.1.3. LIDAS

1.1.4., 1.2.6. és 1.3.2.-ben már volt szó erről a rendszerről. Emlékeztetünk arra, hogy 128 K 2 byte-os szó központi memóriáju LILITH személyi számitógépen fut, MODULA-2-ben programozták. A rendszer felépitése a 10. ábrán látható.



10. ábra

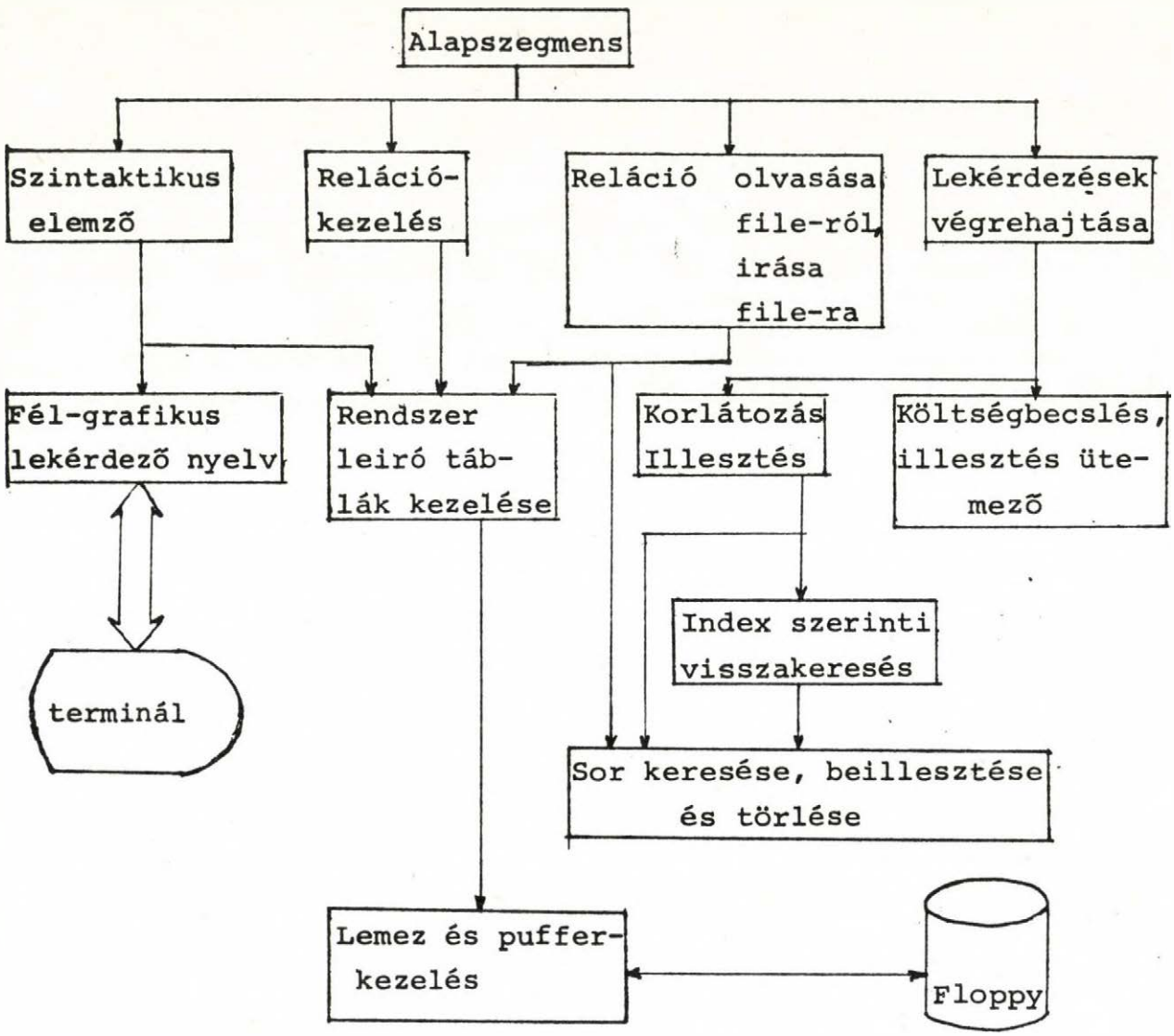
A rendszer két felhasználói interface-t biztosit, az egyiket a számitógéphez nem értő felhasználó, a másikat a programozó-felhasználó számára. Az előbbi a DISCUSS /Database Interface Specified for a Casual User of a Small System/. Ez adatszerkesztőt /1.2.6./, ürlapdefiniálót, és a HIQUEL lekérdező nyelvet /ld. 1.2.6./ tartalmazza. Az utóbbi a MODULA/R adatkezelő-programozási nyelv /ld. 1.3.2./.

A rendszer jellegzetessége a relációs séma definiálásakor keletkező MODULA/R adatmodulok. Ezeket a GAMBIT interaktiv adatdefiniciós rendszer /ld. 1.1.4./ generálja, és a MODULA/R forditóprogram készit belőlük futtatható modulokat. A relációknak, mint absztrakt adattipusoknak /adatszerkezet + müveletek/ a definicióit tartalmazzák. A felhasználó a definiciós fázis után már csak az ott definiált módositó tranzakciókat használhatja, és ezeket csak a létrehozott modulokon keresztül. A módszer hátránya rugalmatlansága, előnye viszont, hogy komplikált konzisztencia-feltételek megadását teszi lehetővé /ld. 1.1.4./.

A lekérdezés-vezérlő /query evaluation manager, QEM/ feladata az elérési ut optimizálása. A tárolási részrendszerrel a szokásos adatbázis assembleren /7. ábra/ keresztül tart kapcsolatot, csak itt azt elem interface-nek /element interface/ hivják [REBS 82, REBS 83]

2.1.4. Néhány mikrogépes rendszer

A TITAN rendszer lényegében a QBE /nem tudjuk pontosan mennyire teljes/ implementációja PASCAL-ban /7000 sor/, APPLE II mikrogépen 56 K byte memóriával /ld. 1.2.6./. Felépitése a 11. ábrán látható:



11. ábra

Az ábra tetején látható Alapszegmens a vezérlő része a rendszernek. Ez állandóan a memóriában tartózkodik. Egy szinttel lejjebb 4 egymástól független szegmens helyezkedik

el, az Alapszegmens ezeket tölti be a memóriába aszerint, hogy mikor melyikre van szükség.

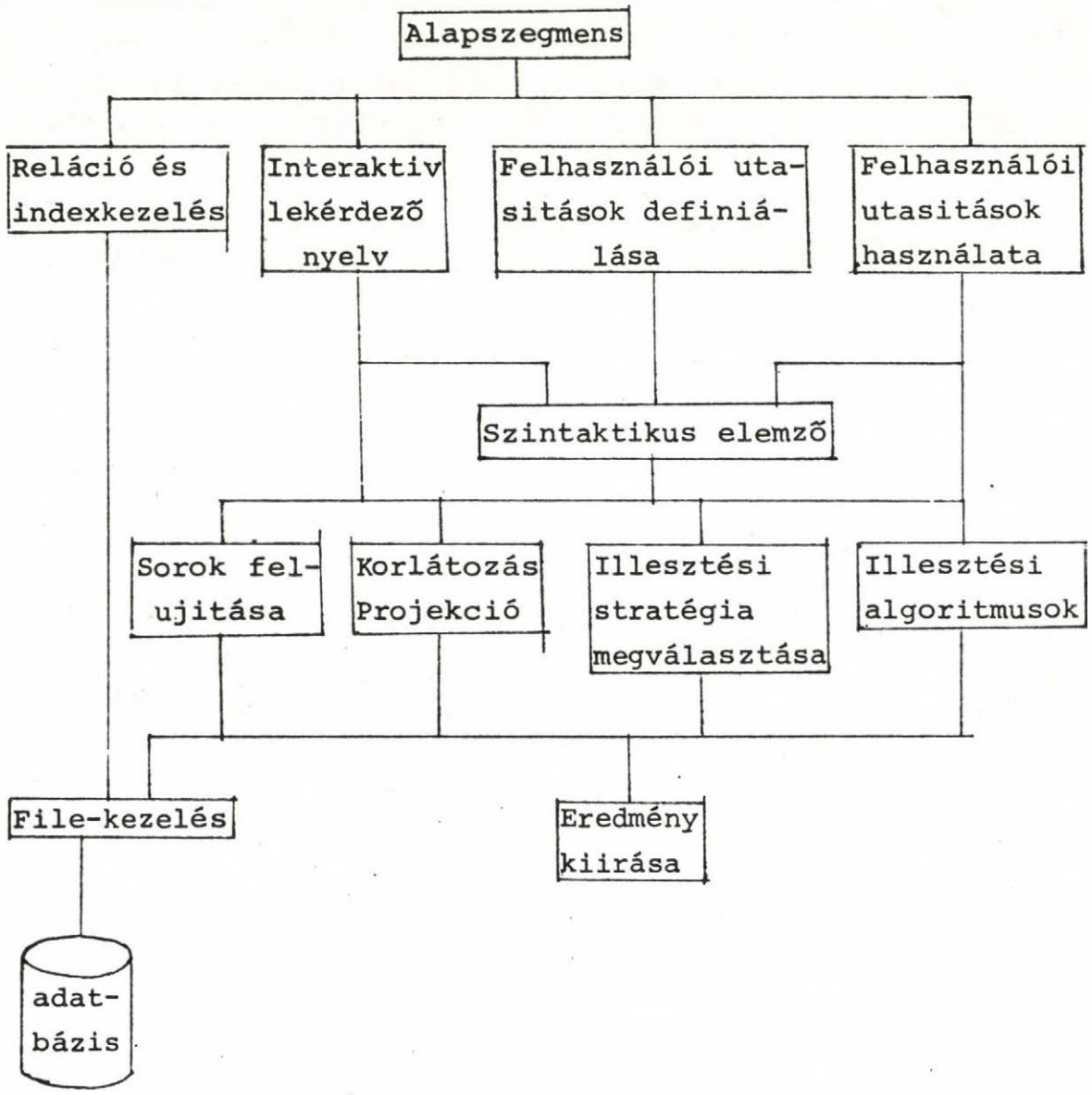
Balról jobbra haladva az első Szintaktikus elemző és képernyőkezelő szegmens. Ez kommunikál a felhasználóval.

A Relációkezelő szegmens tartalmazza a relációkat leiró táblákkal végzett müveleteket /reláció definiálása, törlése, listázás/.

A Relációt file-ra iró és onnan olvasó szegmens a WRITE és READ, a QBE-hez képest ujdonságnak számitó parancsokat /ld. 1.2.6./ hajtja végre.

A Lekérdezések végrehajtása szegmens a legösszetettebb. Elég jól elválik benne az optimizáló és a tárolási részrendszer. Elkülönitett blokkban vannak a keresési stratégiát összeállitó, a relációalgebrai szintü müveleteket végző, ill. az indexben és a relációkban soronként mainipuláló programrészek. [FALQ 82]

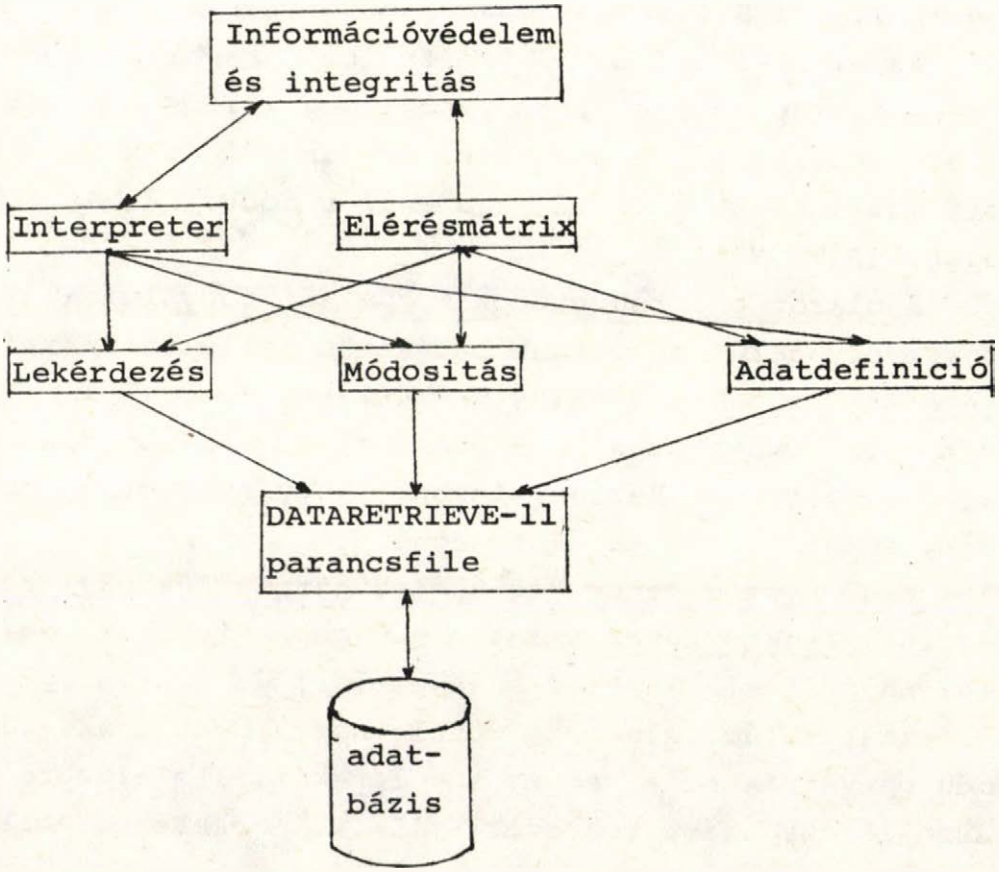
Az RDBAS rendszer még jobban szegmentált, ami érthető, hiszen egy 32 K-s HP21MX-en implementál egy QBE-szerü grafikus nyelvet /ld. 1.2.6./ A 12. ábrán valamennyi doboz önálló szegmenst jelöl:


Alapszegmens
Reláció és indexkezelés
Interaktiv lekérdező nyelv
Felhasználói utasitások definiálása
Felhasználói utasitások használata
Szintaktikus elemző
Sorok felujitása
Korlátozás Projekció
Illesztési stratégia megválasztása
Illesztési algoritmusok
File-kezelés
Eredmény kiirása
adatbázis

12. ábra

Figyelemreméltó a TITAN és az RDBAS szerkezetének hasonlósága. A két speciális lehetőségtől - TITAN: file-ok irása, olvasása, RDBAS: felhasználói utasitások definiálása /ld. 1.1.2./ - ugyanazok a blokkok szerepelnek a két rendszerben: Lekérdező nyelv, Szintaktikus elemző, Illesztési stratégia megválasztása /optimizálás/, Illesztési algoritmus, Korlátozás /+ Projekció/ algoritmus, Reláció /és index/ kezelés. Ugy tünik az RDBAS-nál nincs interface-szel elkülönitett tárolási részrendszer, hiszen sort módositó program van, de kereső nincs. [HERM 83].

Végezetül nézzük meg, hogyan realizálódik az APPLE nyelv /ld. 2.1.6./! A 13. ábra ennek a felépitését mutatja:



13. ábra

Ez a szerkezet az összes eddigitől különbözik, ami nem meglepő, hiszen maga a rendszer sokkal egyszerübb. A megvalósitás a PDP DATARETRIEVE-11 utility-jára támaszkodik, minden adatokkal kapcsolatos müveletet ez végez el /mint ez az ábráról is leolvasható./

A felhasználó először az Interpreterrel kerül kapcsolatba. Ez a nagyon egyszerü parancsnyelv utasitásait értelmezi /ellenőrizve közben, hogy a felhasználó milyen adatok milyen jellegü használatára jogosult/, és jellegüktől függően inditja a három feldolgozás előkészitő modul valamelyikét. Ezek fő feladata az elérési ut meghatározása, hiszen az APPLE-felhasználó az általa használni kivánt relációk neveit sem adja meg, csupán az adatmezők /oszlopok/ neveit /ld. 1.2.6./.

Az elérési ut az Elérésmátrixból kapható. A rendszer azonos oszlopnevek alapján próbál meg illesztésekkel jutni el az inputtól az outputig. Ha ez sikerül az eredményül kapott müveletsort file-ra irja - és a többi már az utility dolga. [PATN 83]

A mikrogépes rendszerek közül két bonyolultat és egy nagyon egyszerüt mutattunk be. Közös jellemzőjük az erős szegmentáltság, az overlay használata. Ez nyilvánvalóan a szük memóriakapacitásból adódik, tehát másképp nem lehetne megirni őket. Más kérdés viszont, hogy az overlay valószinüleg sokat ront a hatékonyságukon.

Ezek a rendszerek kevésbé szigoruan, és más szempontok szerint strukturáltak, mint a nagygépre irottak. Nem feltétlenül létezik adatbázis assembler, és miután egy-egy lekérdezést relációalgebrai müveletsorra bontottak, külön-külön modulok végzik el az egyes müveleteket. /Az algebrai nyelvet használó szépszámu rendszer - 1.2.6. - esetében ez egészen természetes/.

Nem térünk ki külön a relációsan nem teljes rendszerek szerkezetére. Ez a relációs rendszerekéhez képest elég triviális, hiszen egyszerre csak egy relációval - ez általában egy file-t jelent - dolgoznak.

2.2. Optimizálási algoritmusok. Optimizáló implementációk

Mint a 7. ábrán látható, a relációs adatbáziskezelő rendszerekben a felhasználói igények kielégitése /legalábbis logikailag/ két lépésben történik. A tárolási részrendszer feladata egy reláció hatékony manipulálása.

· Az Optimizáló bontja le a sokszor komplex, több relációt érintő lekérdezéseket vagy módositásokat egy-relációs müveletek sorozatára.

Elég nyilvánvaló, mekkora hatást gyakorol az optimizáló algoritmus a rendszer hatékonyságára. Feladata, a rendelkezésre álló külső és központi memóriában, a legrövidebb idő alatt végrehajtható müveletsor kiválasztása az általában nagy számu lehetségesből. Döntenie kell a lépések sorrendjéről, indexek, szervezési módok használatáról vagy nem kihasználhatóságáról, az egyes müveletek elvégzéséhez rendelkezésre álló algoritmuskészletből az adott helyzetben legjobban alkalmazható megválasztásáról stb.

Az ismertetendő optimizálási algoritmusok heurisztikusak, és azt sem tudjuk megmondani, melyik a "legjobb" /még a "legjobb" fogalmát sem olyan egyszerü definiálni/. A feladat bonyolultságát jól jellemzi a következő eredmény: ha tekintjük a lekérdezéseknek az illesztés, összehasonlitás és az egyenlőség feltételü korlátozás /kiválasztás/ generálta nyelvét /nem lesz relációsan teljes!/, akkor az erre készithető optimizálási algoritmus bonyolultsága a lekérdezés méretétől exponenciálisan függ. Két kérdés ekvivalenciájának eldöntése már az NP-teljes feladatok körébe tartozik.

Egy, a fentinél szükebb körü lekérdezés osztályra sikerült csupán $O(n^4)$ bonyolultságu algoritmust találni /n a kérdésben szereplő változók száma/. És ezek az absztrakt algoritmusok, a tárolási szerkezet kinálta lehetőségeket /index, közvetlen hozzáférés, stb./ nem veszik figyelembe! [AHO 79]

2.2.1. A Palermo-algoritmus. LIDAS implementáció

Ez az 1972-ben publikált eljárás lényegében véve Codd redukciós algoritmusának /0.2./ javitása, hatékonyabbá tétele. Codd algoritmusának legköltségesebb része nyilvánvalóan a 2. és 3. lépés, a lekérdezésben részt vevő relációk Descartes-szorzatának képzése, majd az igy kapott, feltehetően hatalmas méretü relációból a lekérdezés feltételeinek eleget tevő sorok kiválasztása.

A javitott algoritmus alapötlete szerint nem kell Descartes szorzatot képezni, ehelyett elegendő speciális indextáblákat létrehozni, azokban összegyüjteni a válaszként szóba jöhető sorok közvetlen elérését lehetővé tevő pointereket. Az indextáblák összevetésével kapjuk majd meg a válaszként adandó relációt.

Kétfajta indextáblát definiálunk: az értéklista egy adott T változó adott A oszlopára készül és egyszerü inverz file. Legyen a_j az A-ban szereplő tetszőleges érték, azoknak a soroknak a mutatói pedig, ahol az A oszlopban a_j áll $p_{j1}, p_{j2}, \ldots, p_{jk}$! Ekkor az $\{a_j, (p_{j1}, \ldots, p_{jk})\}$ $j=1,2,\ldots,m$ elemek halmazát nevezzük értéklistának.

Párlista olyan T és U változók A és B oszlopainak értékeire készül, melyekre a lekérdezés $T.A \theta U.B$ /θ vagy $<$ vagy $>$, vagy $=$/ alaku feltételt tartalmaz. A párlista ekkor $\{(p_{k_1}, p_{k_2}, \ldots p_{k_n}), (q_{k_1}, q_{k_2}, \ldots, q_{k_m})\}$ $k=1,2,\ldots,p$ elemek halmaza, ahol a p-k olyan T sorokra mutató pointerek ahol

$T.A=a_k$, a q_k pedig olyan U sorokra mutató pointerek, ahol $U.B=b_k$, és teljesül az $a_k \Theta b_k$ feltétel. Nyilvánvaló, hogy ha T.A-ra vagy U.B-re már van értéklista, a párlista elkészitése leegyszerüsödik.

Az algoritmust relációkalkulus terminológiában /1.2.1./ irjuk le. Az indextáblákat épitő részalgoritmust reláció invertálásának nevezzük. A reláció invertálása a következőképpen történik:

a/ ha a reláción definiált változó szabad /nem kvantoros/, akkor a relációnak a lekérdezésben használt oszlopaira vonatkozó projekcióját állitjuk elő, és azzal dolgozunk.

b/ Ha az r_i változó /akár szabad, akár kvantoros/ valamilyen $r_i \Theta r_j$ összehasonlitásban szerepel és az r_j változónak az összehasonlitásában szereplő oszlopára már van értékindex, akkor elkészitjük az összehasonlitás párindexét. Ha az r_j-nek nincs megfelelő értékindexe, az r_i összehasonlitásban szereplő oszlopára készül értékindex. Az értékindex készitésénél az r_i-re vonatkozó egyváltozós feltételek figyelembe veendők, és csak az ezeket kielégitő sorok pointerei kerülhetnek be az értékindexbe.

Maga az algoritmus ezek után a következő:

1. Minden lekérdezésben szereplő változóra megbecsüljük az invertálásához szükséges memória méretét. A becslések alapján nagyság szerint növekvő sorrendbe rendezett L változólistát hozunk létre.

2. A lista első /következő/ relációját invertáljuk, és kihuzzuk a megfelelő változó nevét a listából.

3. Ha L üres, ugrás a 4. lépésre. Ha nem, uj becsléseket készitünk, szükség esetén átrendezzük L-t, és visszatérünk 2-re.

4. Az érték és párindexből a lekérdezési feltételnek

megfelelően uniókkal és metszetekkel egyetlen közös index-táblát hozunk létre. /Ez lényegében Codd algoritmusának harmadik lépése után létrejövő Descartes-szorzatnak felel meg. Innen az algoritmus megegyezik a redukciós algoritmussal, vagyis:/

5. A kvantorok hatásának megfelelő műveletek elvégzése.

6. Projekcióval megkapjuk a kivánt relációt /miután a pointerekkel sorokkal helyettesitettük/.

Az algoritmus LIDAS /ld. 2.1.3., 1.3.2./ implementációját példán keresztül mutatjuk be:

Az adatbázis két relációból áll. Ezek:

Dolgozó	Név	Részlegkód	Besorolás	Alapbér	Főnök
	Csipke Rózsika	FBI	pozitiv hős	12	Gonosz Mostoha
	Hétfejü Sárkány	CIA	tüzokádó	9	Gonosz Mostoha
	Gonosz Mostoha	FBI	boszorkány	10	Hétfejü Sárkány
	Hó Fehérke	CIA	pozitiv hős	3	Gonosz Mostoha
	Hétkis Törpe	CIC	bányász	7	Hó Fehérke

Részleg	Részlegkód	Cim
	FBI	az Óperenciás tengeren is tul
	CIC	az Üveghegyen innen
	CIA	az Üveghegyen innen

Az összes pozitiv hős és az összes Üveghegyen innen elhelyezkedő részlegben alkalmazott "dolgozó" nevének és besorolásának kiválasztása a következő MODULA/R lekérdezéssel valósitható meg:

```
{<* d.név,d.besorolás *> OF EACH d IN dolgozó:
 (d.besorolás='pozitiv hős')
  OR SOME r IN részleg
  ((r.cim='az Üveghegyen innen') AND (d.részlegkód=r.részlegkód))}
```

Az optimizálás első lépését már a MODULA/R forditóprogram végrehajtja: az adott predikátumot diszjunktiv normálformára hozza:

```
{<* d.név,d.besorolás *>OF  EACH d IN dolgozó:
  SOME r IN részleg
 ((d.besorolás='pozitiv hős') OR
  (r.cim='az Üveghegyen innen') AND (d.részlegkód=r.részlegkód))}
```

/Mivel diszjunktiv normálformáról van szó, a müveletvégzés sorrendje szempontjából természetesen az AND prioritást élvez az OR-ral szemben./

A második lépés a relációk invertálása. /Megjegyezzük, hogy az invertálás helyett a relációs adatbázis standard indexei használhatók /ha vannak/, de itt ezzel a lehetőséggel nem foglalkozunk./ A LIDAS minden T változóra és i-edik konjunkcióra /T,i/ értéklistát, hoz létre. A /T,U,i/ párlisták létrehozása Θ müvelettel összekapcsolt T és U változónként és i konjunkciónként történik. /A LIDAS ezt Gyüjtő fázisnak nevezi/.

A példában az Optimizáló először az r-Részlegen értelmezett-változót invertálja. Az 1. konjunkcióban nem szerepel, a másodikban kétszer is. A program itt felismeri azt a lehetőséget, hogy az r.cim='az Üveghegyen innen'

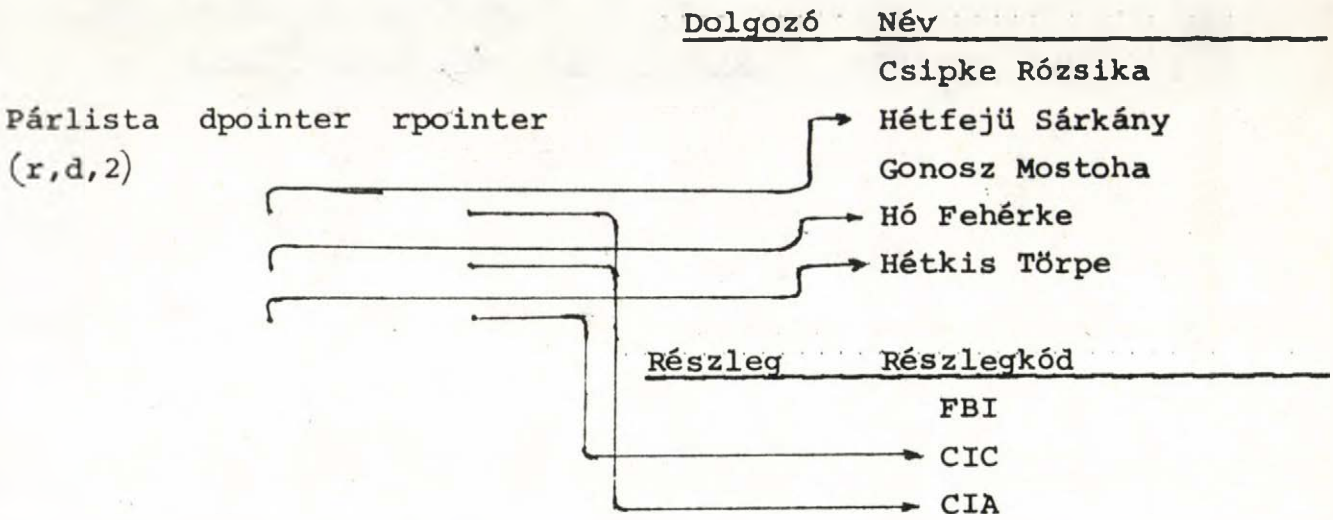
taghoz tartozó értéklista és a d.részlegkód=r.részlegkód taghoz tartozó párlista egyetlen párlistává olvasztható össze, melyben csak azok az r sorok szerepelnek majd, melyekre mind a két feltétel igaz. A párlista létrehozásának első lépéseként - a Palermo algoritmus szellemében - el kell készíteni az r. részlegkód szerinti értéklistát. Ezt tehát már ugy teszi, hogy csak azokat a részlegkódokat szerepelteti, melyek az Üveghegyen innen helyezkednek el, vagyis melyekre fennáll az r.cim='az Üveghegyen innen' feltétel. Ez az értéklista a következőképpen néz ki:

értéklista (r,2) Részlegkód	pointer	Részleg Részlegkód
CIC	→	FBI
CIA	→	CIC
	→	CIA

Most a d- Dolgozón értelmezett-változó következik. Az első tagra /d.besorolás='pozitiv hős'/ értéklista készül:

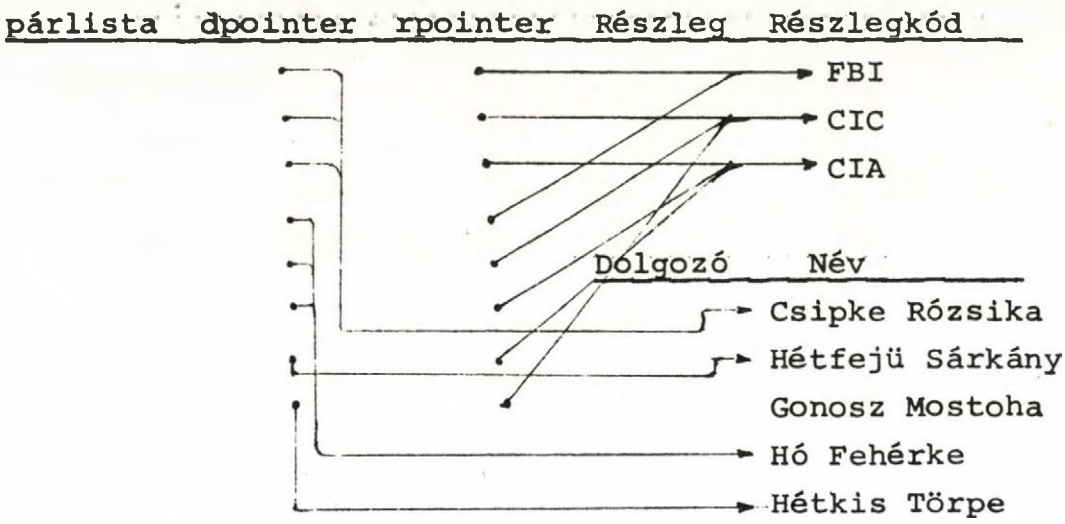
értéklista(d,1) Besorolás	pointer	Dolgozó Név
pozitiv	→	Csipke
hős		Rózsika
pozitiv		Hétfejü
hős	→	Sárkány
		Gonosz Mostoha
	→	Hó Fehérke
		Hétkis Törpe

A második tagra - felhasználva az (r,2) értéklistát - elkészithető az (r,d,2) párlista:



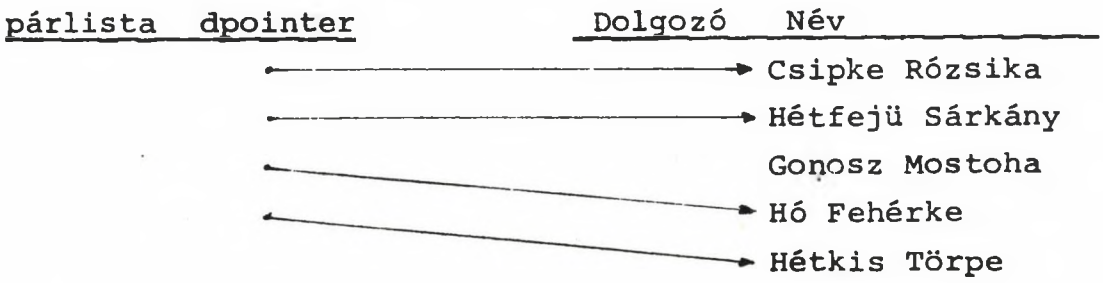
Az invertálás /a LIDAS terminológiában az első fázis/ ezzel befejeződött. A második lépés a Kombinációs fázis. Ez először az AND kapcsolatban álló listák közös elemeinek kiválogatásából /gyakorlatilag egy merge/, majd az igy kapott listák egyesitéséből áll.

Esetünkben az AND-ek elvégzése szükségtelen, az (r,d,2) párlistát már ugy épitettük fel, hogy mind a két AND viszonyban álló feltétel teljesüljön. Az OR viszonyban álló listákat kell egyesiteni. Ehhez (d,1) értéklista és az összes Részleg sor Descartes szorzatát vesszük, és ezt egyesitjük (r,d,2) párlistával. Az eredmény elég kusza pointerdzsungel:



/Az első hat sor a Descartes-szorzat, az utolsó kettő pedig /r,d,2/ párlista 1. és 3. sora. A második sor már a Descartes-szorzatban szerepelt, igy az egyesitésnél kiesett./

A Palermo algoritmus szerint a kvantorok következnek. A SOME /egzisztenciális kvantor/ projekciót, az ALL /univerzális kvantor/ osztást /ld. 1.2.2./ jelent. Példánkban a SOME a párlista dpointer-re való projekcióját jelenti. Az eredmény



A LIDAS harmadik fázisa a Konstrukció. A pointerek az aktuális sorokra cserélődnek, majd projekció következik:

Eredmény	Név	Besorolás
	Csipke Rózsika	pozitiv hős
	Hétfejü Sárkány	tüzokádó
	Hó Fehérke	pozitiv hős
	Hétkis Törpe	bányász

A rendszer alkotói értékelést is adnak az algoritmusról. Eszerint elsősorban a munkaterület minimális szinten tartására való törekvés miatt választották ezt az algoritmust.

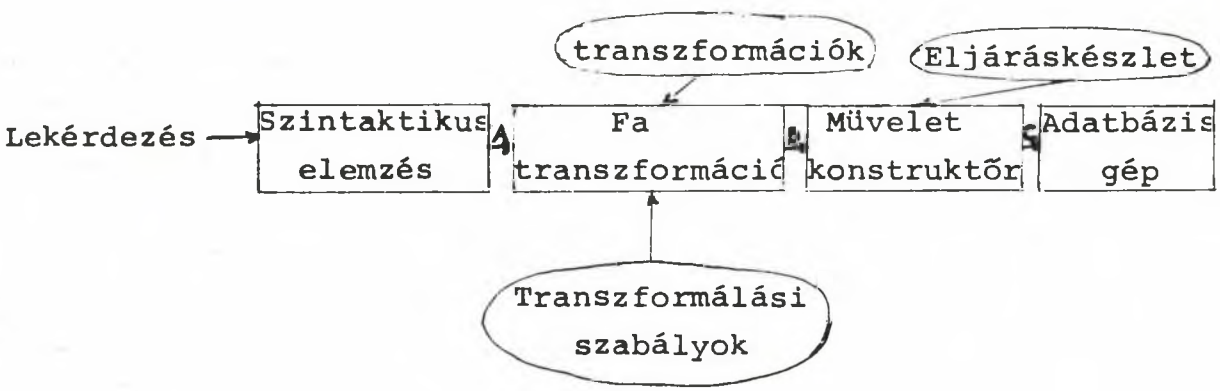
Második észrevételük szerint azokra a változókra, melyek csak egy kétváltozós összehasonlitásban /változó1 Θ változó2/ szerepelnek már a Gyüjtő fázisban kiértékelhető a kvantor. Ez a mi esetünkben - SOME r, és az r csak egy kétváltozós összehasonlitásban /r. részlegkód= d.részlegkód/ szerepel - azt jelentette volna, hogy az /r,d,2/ párlista rpointer oszlopát figyelmen kivül hagyva, a dpointer oszlopot egyesithettük volna /d,1/ értéklistával, elintézve ezzel Kombinációs fázist. A müködő LIDAS Optimizáló valójában igy csinálta volna.

Még egy megjegyzés: ha egy lekérdezendő reláció feltételeket kielégitő elemei a reláció egy végigolvasásával kiválogathatók /tehát pl. ha nem szerepel kétváltozós összehasonlitásban a relación értelmezett változó/, akkor nincs értelme értéklistát készitени. Ezt az esetet az Optimizáló felismeri, és egy, az itt ismertetettől különböző algoritmust használ. [REBS 82]

2.2.2. A Smith -Chang algoritmus. PRTV implementáció

Az eredeti [SMIT 75] cikk az algoritmust többprocesszoros számitógépeken futó rendszerek, vagy adatbázis gépek Optimizálójának szánta, de ettől függetlenül, tradicionális architekturáju gépeken is jól felhasználható ötleteket tartalmaz. Az algoritmus a hipotetikus SQUIRAL /Smart Query Interface for a Relational Algebra/ adatbáziskezelő rendszer része. Erről a rendszerről elég annyit tudnunk, hogy a felhasználói interface relációalgebra, az adatbázis assemblere pedig /7. ábra, 2.1./ a három klasszikus müveletet: illesztést, korlátozást /kiválasztást/ és projekciót realizáló eljárásokat tartalmaz. Egy-egy müveletre többfajta eljárás is rendelkezésre áll, hiszen nyilvánvaló pl. hogy más algoritmussal érdemes illeszteni két, az illesztendő oszlopok szerint rendezett relációt, és másképp két rendezetlent.

Az algoritmus elvi vázlata a 14. ábrán látható:

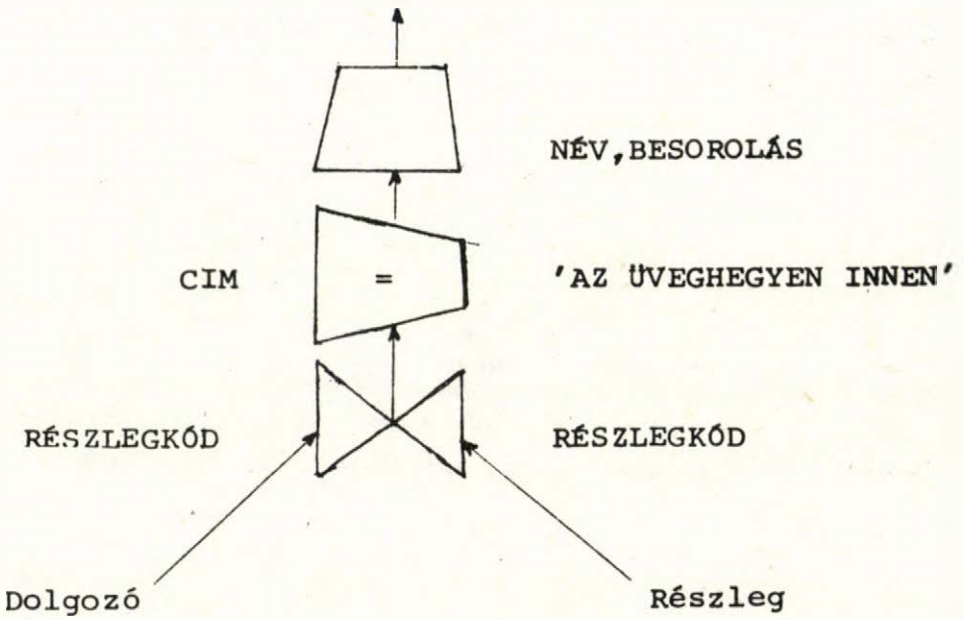


14. ábra

A lekérdezésből a Szintaktikus elemzés fát készit. A fa levelei a lekérdezésben szereplő adatbázis relációk, gyökere a válasz reláció, a közbülső csomópontok az egyes elvégzett müveletek után kapott ideiglenes relációk. 2.2.1-ben bemutatott példánk egyszerüsitve: az Üveghegyen innen elhelyezkedő részlegekben alkalmazott dolgozók neveinek és besorolásainak kiválasztása a relációalgebra nyelvén /1.2.2./:

JOIN DOLGOZÓ AND RÉSZLEG OVER RÉSZLEGKÓD GIVING T1
SELECT T1 WHERE CIM='AZ ÜVEGHEGYEN INNEN' GIVING T2
PROJECT T2 OVER NÉV,BESOROLÁS GIVING T3

Ennek a müveletsornak a 15. ábra fája felel meg:



Jelölés:
Projekció -  , korlátozás -  , illesztés - 

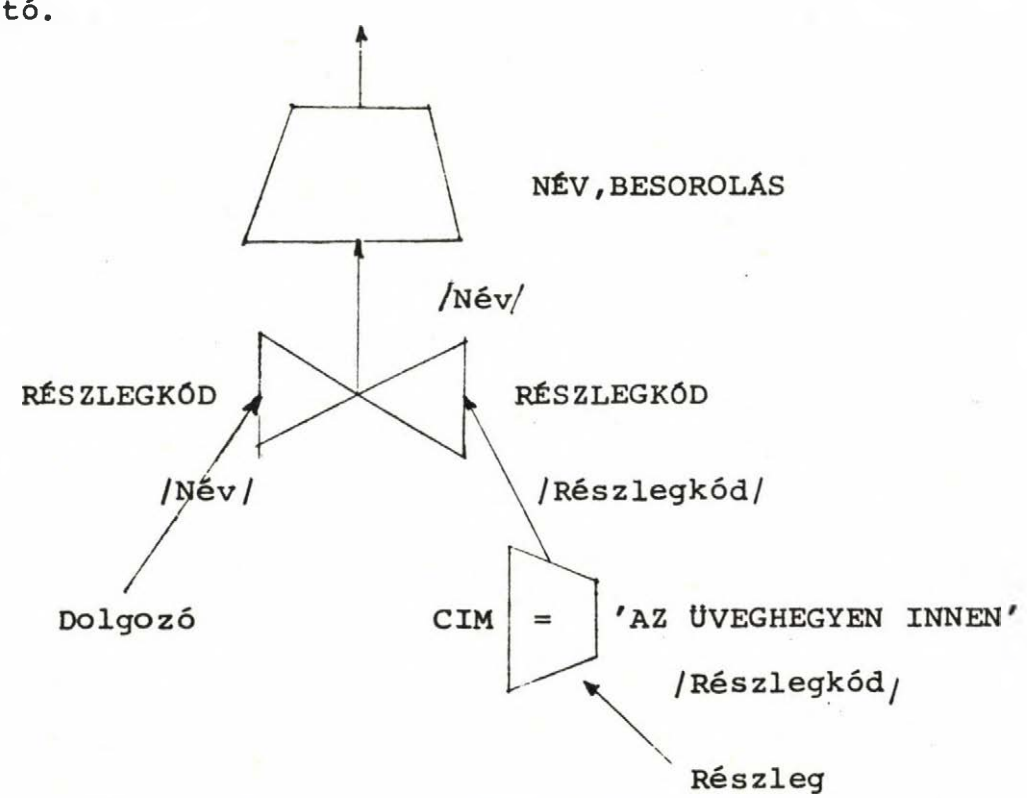
15. ábra

A második lépés, a kapott fa transzformálása következik. Ez meghatározott készletbe tartozó transzformációk megfelelő szabályok szerinti alkalmazásával történik. Ezekre még visszatérünk, egyelőre csak annyit jegyzünk meg, hogy esetünkben célszerűnek tünne az illesztés és a kiválasztás sorrendjét megcserélni. Ezzel két ponton is határozott sebességnövekedést érnénk el:

a/ csökkenne az illesztendő sorok száma a Részleg relációban, és ez az illesztésnél - algoritmustól függő mértékben - mindenképpen kedvező

b/ a válogató algoritmusnak az illesztés eredményeként kapott, nyilván nagyméretü reláció helyett elegendő a Részleg relación végigmenni /ami mellesleg tárolt reláció, tehát közvetlen elérést támogató indexe lehet/.

A transzformáció eredményeként kapott fa a 16. ábrán látható.



16. ábra

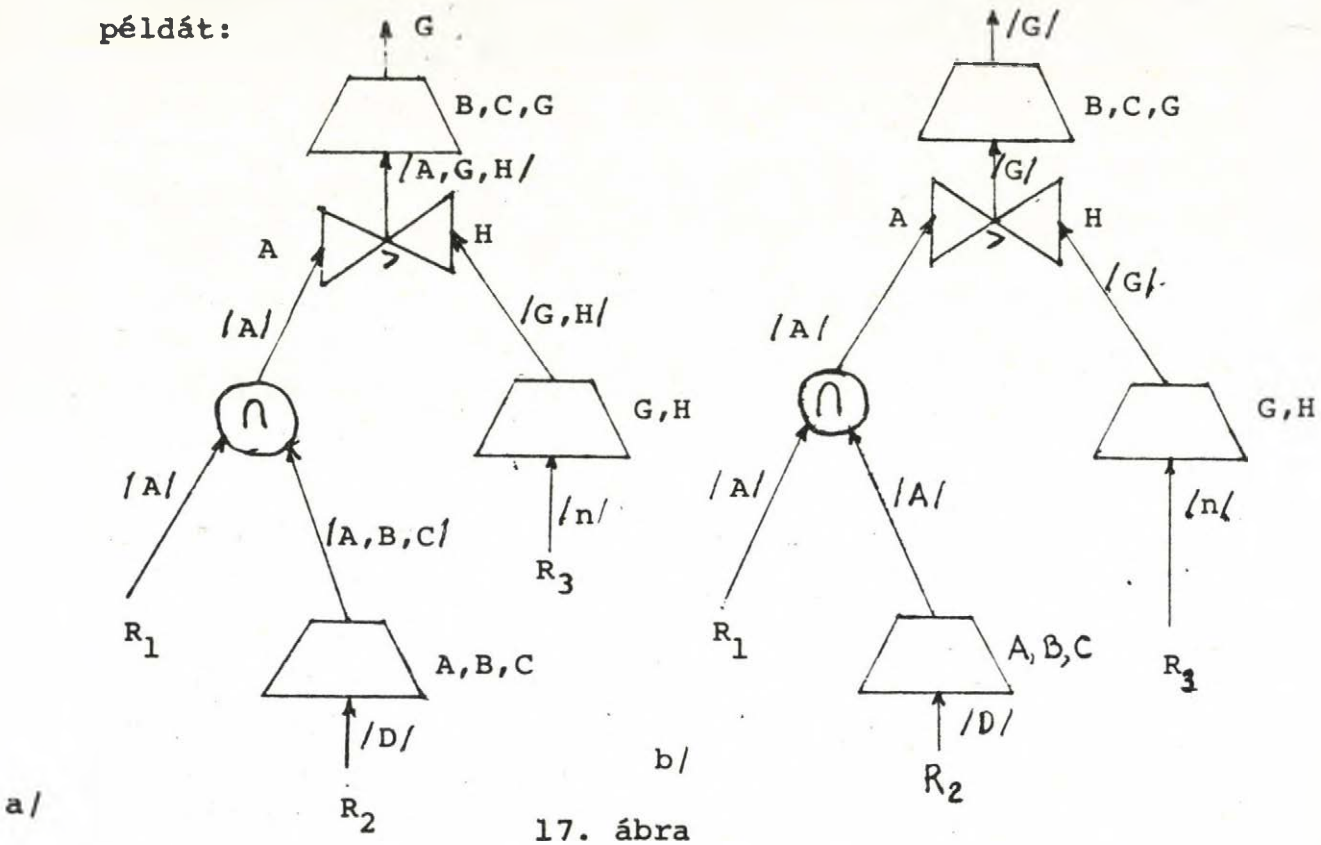
Ez lesz a Fa transzformáció /a 14. ábrán B-vel jelölt/ kimenetén. A Művelet konstruktőr feladata a rendelkezésre álló müveletvégző eljárások közül a legkedvezőbbeket kiválasztani. [SMIT 75] a választás döntő tényezőjének a reláció rendezettségét tartja, és a kiválasztási algoritmus mozgástere lényegében az ideiglenes relációk rendezettségének megválasztása.

A Művelet konstruktőr először a levelektől a gyökérig járja be a fát, és minden reláció mellé bejegyzi az un. legkedvezőbb rendezettséget. Ez nem más, mint az előző művelet hatékony megvalósitása után kapott rendezés. Ezek sorozata tulajdonképpen előzetes végrehajtás terv. Példánkban tételezzük fel, hogy a Részleg a Részlegkód, a Dolgozó a Név szerint rendezve van az adatbázisban. /Ezt a nyil mellé irt "/Részlegkód/" ill. "/Név/" jelöli a 16. ábrán/. A korlátozásra - tegyük fel - az eljáráskészlet a reláció szekvenciális végigkeresését kinálja hatékony eljárásként. /Lehet ott persze indexen keresztüli hozzáférés, meg sok más is/. Ez a művelet nem változtat a rendezettségen, tehát a korlátozás eredményeként kapott reláció Részlegkód szerint rendezett marad. Ez jól is jön nekünk egy elég hatékony illesztéshez, mely után az eredményül kapott reláció Név szerint rendezve lesz.

Az első menet tehát a legkedvezőbb rendezettségek kiosztása. Lehet persze több kedvező rendezettség is,ilyenkor valamennyit fel kell tüntetni. [SMIT 75] - egy ott definiált eljáráskészlet alapján - táblázatban adja meg a rendezettségek kiosztásának szabályait.

A második menet a gyökértől a levelekig járja be a fát. Ennek feladata olyan esetekben döntést hozni, amikor több kedvező rendezés is létezik. Ez a döntés a müveletet végrehajtó eljárások legkedvezőbbjét választja meg - meghatározva ezzel a rendezettséget is. Esetünkben nincs

választási lehetőség, de a 17. ábrán erre is láthatunk példát:



17. ábra

A fiktiv R_1,R_2 és R_3 relációkra hajtjuk végre a 17. ábrán látható müveletsort. Az a/ ábrán a Művelet konstruktőr első, a b/-n a második menete utáni állapot látható /b-n mindegyik lépéshez odaírhattuk volna a kiválasztott eljárás nevét/. R_1 reláció A,R_2 D szerint rendezett, R_3 rendezetlen. Ennek megfelelően alakulnak a 17.a, ábrán a rendezettségek. /Például az R_2 reláció első projekciójánál mindenképpen le kell rendezni a relációt a három oszlop A,B és C szerint, de megválasztható, hogy milyen legyen a rendezési sorrend: ABC, BAC vagy CBA - a többi nem érdekes - és ezzel a kimenő reláció rendezettsége/.

A második bejárás utáni eredményt mutatja a b ábra. Az utolsónak végrehajtandó projekcióhoz nyilván az input G szerinti rendezettsége a kedvező - hiszen G azon

oszlopok egyike, melyekre projektálunk. Az illesztés kimenetének G szerint rendezettnek kell lennie - ez a bemenet G szerint rendezettségét is meghatározza, s.i.t. [SMIT 75].

A PRTV rendszerben használt /ld. 0.2./. optimizálási technikát ismertető [HALL 76] cikk [SMIT 75]-ből két nagyon fontos ötletet - a fa-transzformáció már ott /noha nem teljesen általánosan megadott/ alapelvét veszi át és finomitja:

a/ a korlátozások összevonása egyetlen korlátozássá
b/ az unáris müveletek minél korábbi elvégzésére való törekvés

Tulajdonképpen elég nyilvánvaló mind a két elv: a/ elv szerint nem érdemes annyiszor végigmenni egy relación, ahány különböző feltétel van az egyes tagokra, célszerübb egyszer beolvasni a relációt, és egyszerre ellenőrizni az összes feltétel teljesülését. A b, elv mellett nyilvánvalóan a relációméretek minél korábbi csökkentése szól. /A korábbi példán is láttuk az előnyeit/. Ezeket az elveket HALL 76 átfogalmazza és kiegészíti:

a/ a korlátozásokat minél korábban kell elvégezni
b/ a projekciósorozatokat össze kell vonni
c/ fel kell ismerni, és kiszürni a felesleges müveleteket /pl. nyilvánvalóan nem érdemes elvégezni az /AUB/-/BUA/ müveletsort/
d/ a több helyen szereplő tagokat egyszer kell előállitani.

Vizsgáljuk meg sorban ezeket az elveket!

Az a/-hoz rögtön meg kell jegyezni, hogy noha triviálisnak tünik, valójában nem az. Az illesztés és korlátozás sorrendjének megcserélése körültekintést igényel. Példaképpen megemlitjük, hogy ha a 2.2.1.-ben szereplő lekérdezésnek nem az egyszerüsitett, hanem az eredeti változatát

- pozitiv hősök, és az Üveghegyen innen elhelyezkedő részlegekben alkalmazott dolgozók nevei és besorolásai - kísérreljük meg optimizálni, nyilvánvalóan nem tudnánk az illesztés és a kiválasztásnk sorrendjét megcserélni. [TODD 76] a PRTV Optimizálóról irva megjegyzi, hogy általában az illesztés és a korlátozás felcserélése a korlátozás részekre vágásával történik /már amikor ez megtehető/. Az egyik rész az egyik, a másik a másik input relación hajtható végre, egy harmadik részt maga az illesztés realizál, a negyediket pedig az illesztés eredményén kell elvégezni.

A másik probléma, hogy nem mindig érdemes a korlátozást korán végrehajtani. Például, ha két nem diszjunkt reláció egyesitését cseréljük fel csak gyengén szürő korlátozással, rosszabbul járhatunk, mintha az egyesités eredményén hajtanánk végre korlátozást.

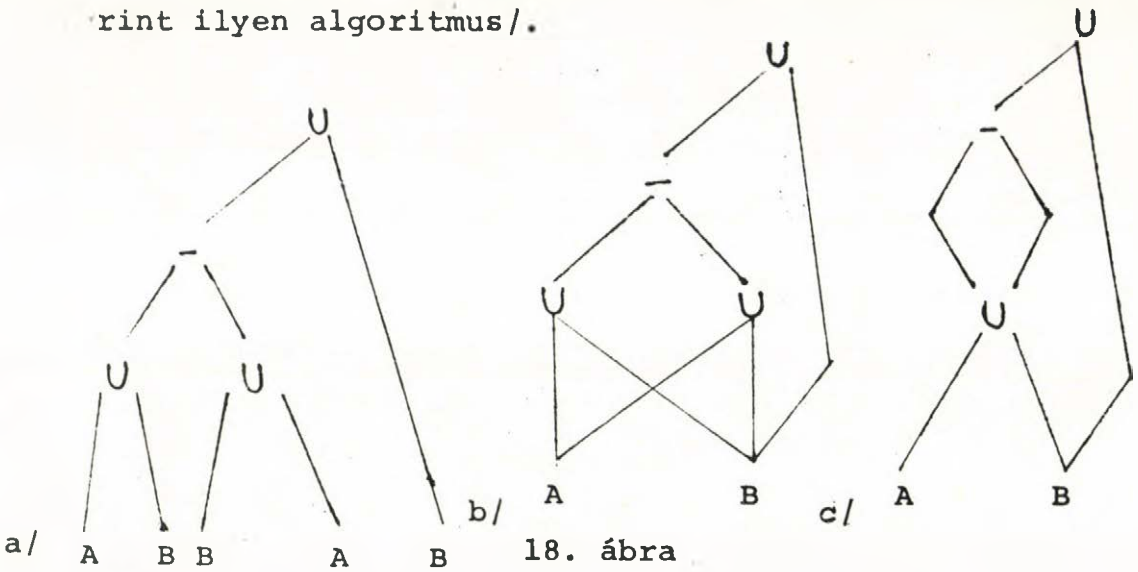
A b, elv helyessége nyilvánvaló. A projekció rendezést jelent, - a duplikátumok kiszürése - és jobb csak egyszer rendezni, mint többször. A gyakorlatban egyébként ez az elv egy másikkal egészül ki:

a rendezést igénylő projekciót minél később,

a rendezést nem igénylőt minél előbb kell végrehajtani [TODD 76].

A c, és d, pontok megvalósitása igen szemléletes transzformációsorozattal történik. A 18. ábrán /AUB/-/BUA/UB kifejezések a 15. ábrára emlékeztető fa ábrázolása látható. A b, ábrán összevontunk egy-egy csucsba A-t és B-t elrontva ezzel a fát, de észrevéve, hogy A kétszer B pedig háromszor szerepel. A b-ből a c, ábrába a müveleti jelek vizsgálata révén juthatunk - mind a két müveleti jel A-val és B-vel van összekötve. c, ábráról már látható, hogy AUB-t elég egyszer létrehozni, és kétszer felhasználni - sőt elképzelhető, olyan algoritmus, amely tudja, azt, hogy ha a "-" jelhez a két operandus azonos csomópontból fut be, akkor ez

a rész egyszerüen elhagyható. /A PRTV Optimizáló [HALL 76] szerint ilyen algoritmus/.



18. ábra

[SMIT 75, HALL 76, TODD 76].

2.2.3. Astrahan és Chamberlin TID algoritmusa

Ez az algoritmus SEQUEL /Structured English QUery Language/ /ld. 0.2./ lekérdezések interpretálására készült. Két lényeges körülmény befolyásolta müködésének filozófiáját:

1. A SEQUEL nyelv - az SQL /ld. 1.2.3./ elődje - nem támogatta az "illesztés" tipusu lekérdezéseket, helyettük a "részlekérdezést" ajánlotta /az SQL-ben mind a kettő létezik/. A következő lekérdezés azoknak a dolgozóknak a nevét válogatja ki 2.2.1. rész Dolgozó relációjából, akik a főnöküknél magasabb alapbért kapnak:

SEQUEL:

```
SELECT NÉV
FROM D IN DOLGOZÓ
WHERE ALAPBÉR >
      SELECT ALAPBÉR
      FROM DOLGOZÓ
      WHERE NÉV=D.FŐNÖK
```

SQL:

```
SELECT D.NÉV
FROM DOLGOZÓ D,DOLGOZÓ F
WHERE F.NÉV=D.FŐNÖK AND
      D.ALAPBÉR > F.ALAPBÉR
```

/ld. 1.2.2.c/. Megjegyezzük mellesleg, hogy SEQUEL-ben nem tudnánk a dolgozó nevével együtt a főnökét is előszedni, mint azt SQL-ben 1.2.3.c.-ben tettük.

2. A SEQUEL realizálása XRM-en /ld.0.2./ alapult. Ennek jellegzetessége, hogy külön tárolja az adatmezők értékeit, és külön a relációs sorokat, mint az értékekre mutató pointer n-eseket. Ebből adódott, hogy a rendszer készitői arra törekedtek, hogy a válogatásnál minél kevesebb sort kelljen megvizsgálni. Ehelyett inkább igyekeztek maximálisan kihasználni az indextáblázatokat, és azok alapján felépitett, a válasz szempontjából szóba jöhető sorok XRM belső azonositóit /tuple identifier - TID/ tartalmazó listákkal manipuláltak. [CHAM 81].

Az algoritmus három program együttese. A LALR/k/ tipusu Szintaktikus elemző /Parser/ a SEQUEL utasitásokat ismeri fel, és alakitja át. Bennünket most elősorban a legbonyolultabb rész a WHERE ábrázolása érdekel. Minden WHERE feltételből egy fa lesz, melynek levelei az elemi feltételek, gyökere a válasz, egyes ágai pedig a feltételeket összekötő OR ill. AND müveletek. Ebben a paragrafusban néhány példán keresztül vizsgáljuk az algoritmus müködését 2.2.1. adatbázisán. Az első:

```
SELECT *
FROM DOLGOZÓ
WHERE BESOROLÁS='POZITIV HŐS'  AND
      /NÉV='HÉTFEJÜ SÁRKÁNY' OR
       RÉSZLEGKÓD='FBI'/
```

Ez azoknak a pozitiv hősöknek az adatait irja ki, akik vagy az FBI-nak dolgoznak, vagy Hétfejü Sárkánynak hivják őket. A WHERE fája a 19. ábrán látható.

A második példa megegyezik az 2.2.1.-ben tárgyalttal: Az összes pozitiv hős és az Üveghegyen innen megtalálható részlegekben alkalmazott valamennyi dolgozó neve és besorolása irandó ki:

```
SELECT NÉV,BESOROLÁS
FROM DOLGOZÓ
WHERE BESOROLÁS='POZITIV HŐS' OR RÉSZLEGKÓD IN
      SELECT RÉSZLEGKÓD
      FROM RÉSZLEG
      WHERE CIM='AZ ÜVEGHEGYEN INNEN'
```

Ehhez a lekérdezéshez a Szintaktikus elemző két fát generál a két WHERE-nek megfelelően /20.ábra/, és valamilyen módon jelzi a két WHERE beágyazottsági viszonyát.

Az algoritmust alkotó másik két önálló egység az Optimizáló /Optimizer/ és a Válogató /Scanner/. Mig a Szintaktikus Elemző a fák előállitásával befejezte tevékenységét, ez a két programrész a továbbiakban egymást és rekurzive saját magukat hivogatva elég bonyolult vezérlési szerkezetben többször is fut.

Az Optimizáló feladata a kérdés megválaszolásához beolvasható sorok minimizálása. Ezt indexek segitségével és azonositó /TID/ listák egyesitésével, és metszésével éri el.

Természetesen a kiiráshoz - ha nem előbb - mindenképpen szükség van a tényleges sorokra. Ezeket a Válogató választja ki az adatbázisból. Ugyancsak az ő feladata a feltételeknek eleget tevő sorokból projekcióval kiválasztani a kiirandó elemeket. A Válogató feladata az Optimizálóhoz képest elég egyszerü, igy a továbbiakban csak hivatkozni fogunk rá.

Az Optimizáló müködését három lépésben ismertetjük:

A/ A vizsgált WHERE-ben szereplő elemi feltételek osztályozása:

A legegyszerübb tipust P1-gyel jelöljük. Ez olyan, hogy létező index alapján azonnal előállitható a feltételt kielégitő sorok azonositóinak listája. Példa adatbázisunkban létezzen a Dolgozó relációhoz index Név és Részlegkód, a Részleg-hez pedig Részlegkód szerint! Ekkor a 19. ábrán szereplő feltételek közül a "NÉV=HÉTFEJÜ SÁRKÁNY" és a "RÉSZLEGKÓD= FBI" nyilván P1 tipusuak. Megjegyezzük, hogy a P1 tipusu feltételek egyváltozósak. A SEQUEL szintaktikáját figyelembe véve ez azt jelenti, hogy a lekérdezés többi WHERE blokkjától független. Ez nem feltétlenül jelenti a változó=konstans /lista/ tipusu feltételt. A 20. ábra "RÉSZLEGKÓD IN" feltétele is P1 tipusu, ugyanis /csupán az szükséges, hogy külső blokkra ne hivatkozzon a feltétel, és ez itt teljesül./

A P2 tipusba sorolt feltételek is függetlenek, csak ezekre nem létezik index, megoldásukhoz a Válogató hivására van szükség. Ilyen a 19. ábra "BESOROLÁS=POZITIV HŐS" és a 20. ábra "BESOROLÁS=POZITIV HŐS" és "CIM=AZ ÜVEGHEGYEN INNEN" feltételei.

A P3 tipusu feltételnél a bal oldali változóra van index, de a jobb oldalon külső blokkra történik hivatkozás. Ilyen szerepel harmadik példánkban, mely azoknak a részlegeknek a kódjait válogatja ki, melyek több dolgozót is foglalkoztatnak:

```
SELECT RÉSZLEGKÓD
FROM R IN RÉSZLEG
WHERE  1 <
        SELECT COUNT(*)
        FROM DOLGOZÓ
        WHERE RÉSZLEGKÓD=R.RÉSZLEGKÓD
```

Itt a"RÉSZLEGKÓD=R.RÉSZLEGKÓD" feltétel P3 tipusu, hiszen a Részlegkód oszlop szerint van index a Dolgozó relációban. A P3 tipusu feltételek lényeges, az algoritmusban kihasznált jellemzője, hogy a jobb oldal adott értéke mellett a

feltétel az index alapján azonnal kiértékelhető.

A P4 tipusu feltétel annyival rosszabb a P3 tipusunál, hogy nem létezik rá index, de annyival jobb a P5 tipusunál, hogy érdemes létrehozni ugyanis létezése csökkenti a beolvasandó sorok számát. Harmadik számu példánkban, ha nem lenne index a Részlegkód szerint, érdemes lenne létrehozni azt. Az történik ugyanis, hogy a magasabb szintü blokkban sorra veszi majd elő a részlegkódokat és adja tovább a beágyazott blokknak a részlegben dolgozók számának előkeritésére /rekurzive hiva önmagát/. Ha nincs index Részlegkód szerint, a blokkban minden adott részlegkódra végig kell menni a Dolgozó relación, viszont ha egyszer létrehozzuk azt, attól kezdve már használható. Az index létrehozásával a P4 tipusu feltételből P3 tipusu lesz.

A P5 tipusu feltételre nem is érdemes indexet létrehozni, mert az nem csökkentené a beolvasandó sorok számát. Ilyen lenne a "RÉSZLEGKÓD=R.RÉSZLEGKÓD" feltétel is a harmadik példában, ha nem COUNT, hanem pl. SUM(ALAPBÉR) szerepelne a beágyazott SELECT-ben.

Az Optimizáló A/ lépése tehát a fenti módon osztályozza a feltételeket a külső blokkban, majd a beágyazottban is. A külső blokkban mindenesetre csak P1 és P2 tipusu feltételek vannak, hiszen nem tartalmazhat hozzá képest külső blokkra hivatkozásokat - mivel ilyenek nincsenek. Igy az Optimizáló B/ lépése következik - P3, P4 vagy P5 predikátumot tartalmazó blokkoknál ehelyett a C-re kerül sor, mint azt látni fogjuk.

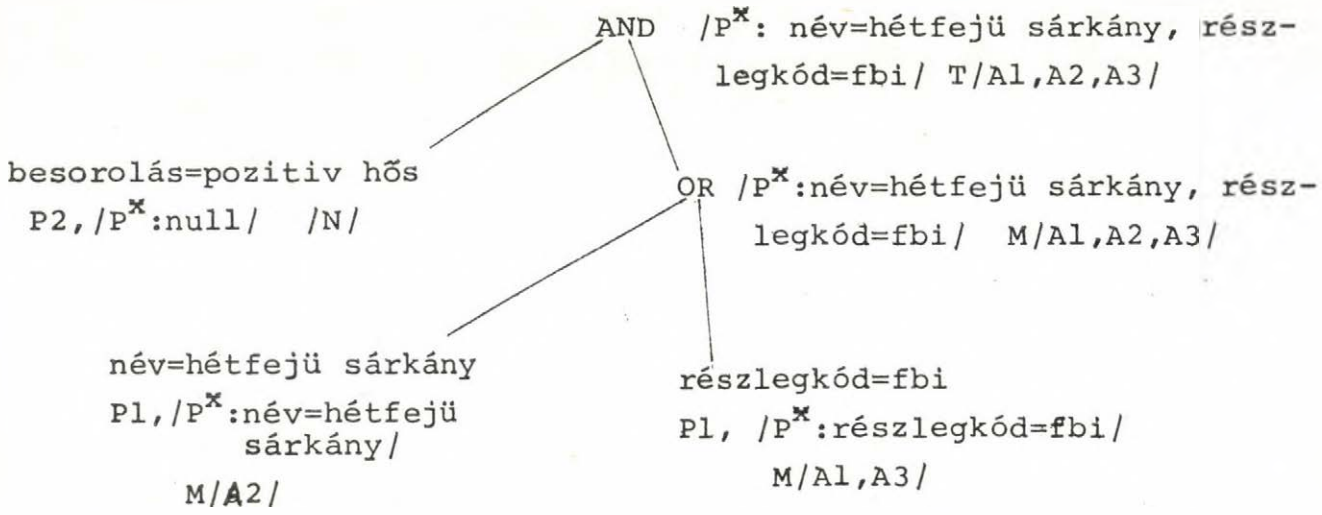
B/ Ennek a lépésnek az első feladata /B1/ annak megállapitása, hogy a blokkban szereplő P1 feltételek mennyire képesek a beolvasandó sorok számát csökkenteni. A fa minden csomópontjához hozzárendeljük az indexből megoldható, a beolvasandó sorok számát korlátozó feltételek P^* listáját az alábbi módon:

a/ a fa leveleinél P^* = feltétel P1, P^*=null P2 tipusu

feltétel esetében.

b/ AND csomópontra $P^x=P^x.L \cup P^x.R$, ahol $P^x.L$ a bal, $P^x.R$ a jobb oldali leszármazott P^x listája.

c/ OR csomópontra, ha $P^x.L$ vagy $P^x.R$=null, akkor P^x=null egyébként $P^x=P^x.L \cup P^x.R$



19. ábra

A 19. ábrán látható első példán részletesebben megnézzük a P^x listák képzését: Először a levelekhez állitjuk össze a listát aszerint, hogy P1 vagy P2 tipusu a feltétel. Mivel az OR egyik leszármazottja sem null, a P^x listája a két vagy - kapcsolatban álló feltétel lesz. Az AND listája megegyezik az OR csomópontéval, hiszen az avval AND kapcsolatban álló másik feltétel P2 tipusu, igy a listája null.

Ezzel a lépéssel tulajdonképpen a feltételek poziciójá értékeltük. Hiába van index ugyanis egy feltételre, ha az

vagy-kapcsolatban áll egy P2 tipusuval. Ahhoz, hogy ezt a vagy-feltételt kielégitő sorokat megkapjuk, mindenképpen végig kell menni a relación a P2 tipusu feltétel miatt, és ilyenkor nyilván nem érdemes a másik tagra indextáblát használni.

A B/ lépés következő feladata /B2/ válasz szempontjából szóba jöhető sorok azonositóinak kiválogatása minden csomópontra. Az azonositólista mellett minden csomópont indexet is kap, ez jelzi, hogy a csomópontban lévő lista mit jelent. M /megoldott/ az index, ha a csomópontra pontosan ismerjük azokat a sorokat /azonositóikat/, melyek eleget tesznek a csomópont képviselte feltételnek. T /talán/ az index, ha pontos azonositólistánk nincs ugyan, de van egy olyan listánk, mely biztos tartalmazza az összes megfelelő sor azonositóját, de bővebb is annál, P2 tipusu feltételek miatt ezek az azonositók még ellenőrzésre szorulnak. N indexet kap a P^*=null listáju csomópont.

Az indexek és a listák készitése a levelekre nyilvánvaló: P^*=null listáju levél N indexü; P^*=feltétel listáju levél M indexü, az azonositólistát az indextáblából kapjuk. A többi csomópontra a közvetlen leszármazottak határozzák meg a listát. AND csomópontra:

	M(L2)	T(L2)	N
M(L1)	M(L1∩ L2)	T(L1∩ L2)	T(L1)
T(L1)	T(L1∩ L2)	T (L1∩ L2)	T(L1)
N	T(L2)	T (L2)	N

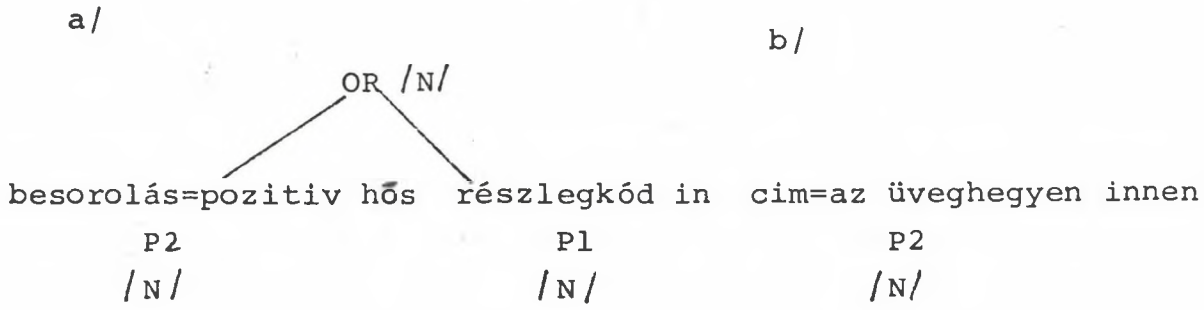
OR csomópontra:

	M(L2)	T(L2)	N	
M(L1)	M(L1∪L2)	T(L1∪L2)	N	
T(L1)	T(L1∪L2)	T(L1∪L2)	N	
N	N	N	N	

ahol L1 és L2 a két leszármazott azonositólistája, M,T és N pedig az indexeik.

Első példánknál a "NÉV=HÉTFEJÜ SÁRKÁNY" feltételnek a Dolgozó reláció második sora tesz eleget /A2 azonositó/, a "RÉSZLEGKÓD=FBI"-nak az első és harmadik. Ezzel a két listával a vagy-feltételre "pontos" /csak a feltételnek eleget tevő sorok azonositóit tartalmazó/ listát kaptunk, igy az OR csomópont indexe M, listája pedig a két lista egyesitése. Az AND csomópont, mivel a Besorolásról nincs index T cimkét kap /eddig biztosan jó soraink bizonytalanná válnak, lehet, hogy ezekben a sorokban a Besorolás nem mindenütt "pozitiv hős"/, az azonositólista változatlan /azért afelől nyugodtak lehetünk, hogy ezeken kivül más sorok nem jöhetnek szóba/.

Nézzük meg második példánkat is /20. ábra/!



20. ábra

/Az Optimizáló a belső blokkra is elvégezte a B1-B2 lépéseket /ugyanis csak egy P2 tipusu feltételt tartalmaz/. Észrevehetjük, hogy a "RÉSZLEGKÓD"-ra vonatkozó feltétel, noha P1 tipusu, mégis N cimkét kapott. OR tipusu csomópontnál ugyanis, ha a csomópont cimkéje N lesz, a leszármazottakat is át kell cimkézni, hiszen nem érdemes azonositólistát készitени hozzájuk. Feltételezzük - nem találtunk rá pontos utalást - hogy a program igy is tesz.

A B lépés utolsó feladata B3 következik: a Válogató hivásával be kell olvasni a sorokat, és kiválogatni közülük a feltételeknek eleget tevőket. Ez azonositólistáink segitségével történik.

A fa gyökerétől indulunk el. Ha a gyökér cimkéje M, készen is vagyunk, az azonositólistán szereplő azonositóju sorokat a Válogató beolvassa, a megfelelő elemeket kiirja. Ha a gyökér cimkéje N vagy T, akkor az azonositólistán szereplő azonositóju sorokat /N esetében az egész relációt/ be kell olvasni, és a sorokat ellenőrizni, lefelé haladva a fán:

a/ T vagy N cimkéjü csomópontban, ha van leszármazott, rekurziv hivással meg kell nézni a két leszármazottat, Azt, hogy a csomópont képviselte feltételt kielégiti-e a sor, a leszármazottak vizsgálatából kapott értékek konjunkciója ill. diszjunkciója dönti el a feltétel tipusától függően.

b/ M cimkéjü csomópontnál egyszerüen meg kell nézni, hogy az éppen vizsgált azonositó benne van-e az azonositólistában. /Ez hatékony XRM müvelet/.

c/ N cimkéjü leveleknél az azonositó meghatározta sor feltételben szereplő elemét be kell olvasni, és ellenőrizni a feltétel teljesülését.

Első példánkban /19. ábra/ csak az A1 azonositóju sorban lesz a gyökérnél lévő listán szereplő azonositóju sorok közül a Besorolás éppen "pozitiv hős", igy ez az egy sor a megfelelő. Ezzel az első példában feltett kérdés választ nyert.

A második példában a gyökér cimkéje N, tehát soronként megy végig a relación. Az első sornál a Besorolás értéke éppen "pozitiv hős", igy ezt a sort - lévén a gyökérben OR - el is fogadja a program. A második sornál a Besorolás értéke nem megfelelő, igy a másik ág, az összetett feltétel vizsgálata következik.

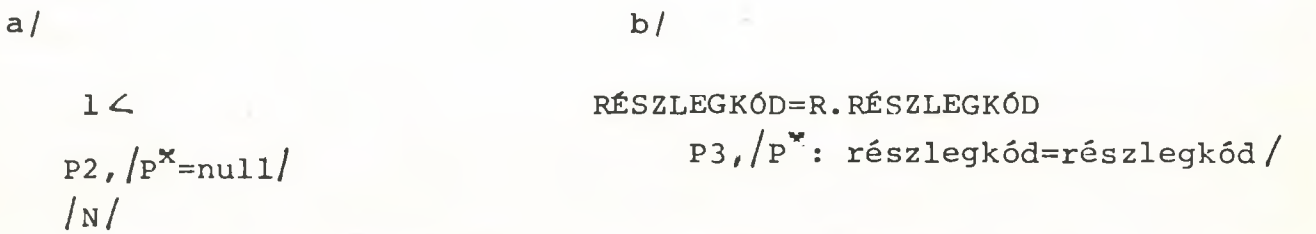
A Válogató ezen a ponton rekurzive meghivja saját magát. Ezt annak megfelelően teszi, hogy a beágyazott blokkban a B1 és B2 lépések megvoltak, most B3 - a Válogató hivása - következik. A Részleg reláció átfésülése - a P2 feltétel megoldása - eredményeként egy listát ad, melyen az Üveghegyen innen elhelyezkedő részlegek nevei /CIC,CIA/ szerepelnek. Ezzel a probléma ekvivalenssé vált a

```
SELECT NÉV,BESOROLÁS
FROM DOLGOZÓ
WHERE BESOROLÁS='POZITIV HŐS' AND
      RÉSZLEGKÓD IN /CIC,CIA/
```

lekérdezéssel. A Válogató rekurzive hivott példánya befejezi müködését.

Az eredeti Válogató a külső blokk kiértékelését folytatva a második sorban lévő részlegkódot ellenőrzi, és mivel az szerepel a listán, elfogadja. A harmadik sor értékelésénél a Besorolás nem jó, a Részlegkód sem szerepel a listán, tehát azt elveti, de a negyediket és ötödiket elfogadja.

Nézzük most meg a harmadik példát /21. ábra/!



21. ábra

Az a/ ábra a külső blokk, egy P2 tipusu feltétel. A B1 és B2 lépés végrehajtásának eredménye az ábráról leolvasható. Mivel a belső blokk P3 tipusu feltételt tartalmaz az Optimizáló B/ lépése helyett erre a C/ hajtódik végre a következőképpen:

C/ Ez a lépés arra készült fel, hogy a P3, P4,P5 tipusu feltételek miatt a blokk nem oldható meg egy hivással. A feldolgozási stratégia szerint a magasabb szintü blokk feldolgozásakor a B3 lépésben amikor a sorok értékelése van hátra, minden egyes sorra meg kell vizsgálni a beágyazott blokkot. Ilyenkor a magasabb szintü blokk átadja az éppen vizsgált sorból azokat az értékeket, melyre a beágyazott értékeléséhez szükség van - jelen esetben a Részlegkódot, az első hivásra "FBI"-t, másodikra "CIC"-t, harmadikra "CIA"-t

Az Optimizáló igyekszik előkésziteni a további hivásokra a blokkot:

a/ a blokk P1 és P2 tipusu feltételeinek azonnali kiértékelésével /létrehozva a feltételeknek megfelelő sorok azonositóját/

b/ a P4 tipusu feltételek P3 tipusuvá alakitásával /index létrehozása/

Az előkészités menete nagyjából a B1-B2 lépések algoritmusai szerint történik. Első lépés annak megállapitása, hogy melyek a beolvasandó sorok számát korlátozandó feltételek. Ez B1 algoritmus szerint történik, azzal a különbséggel, hogy a P1 tipusu feltételek mellett most már a P2, P3, és P4 tipusu feltételek is értékesek számunkra, /a többszöri kiértékelés miatt/ igy a P5 feltételekre lesz csak P^x=null, a többi levélnél P^x=feltétel. A levelek P^x listái után a csomópontokra B1 algoritmus állitja elő a P^x listákat.

A következő lépésben a P^x listák alapján B2 algoritmusával

cimkézzük meg a csomópontokat, és egyuttal létrehozzuk az azonositólistákat is. Itt a következő különbségekre hivjuk fel a figyelmet:

a/ A P2 tipusu feltételek is kiértékelésre kerülnek, egy speciális válogatás létrehozza az azonositójukat

b/ Ha valamelyik P1 vagy P2 tipusu feltétel beágyazott blokkot tartalmaz ennek a blokknak a megoldása rekurziv hivással megkezdődik

c/ A P3 és P4 tipusu feltételeket általában nem tudjuk megoldani, de a P4-re létrejön az indextábla ezzel P3-má alakitva azt.

Lássuk most a B3 lépést a külső blokkra! A Válogató az első részlegkódot próbálja /"FBI"/. Az összetett feltétel értékeléséhez rekurzive hivja önmagát a belső bokkra. Itt a P3 tipusu feltétel "RÉSZLEGKÓD=FBI"-já alakul át, igy P1 tipusuként az indextáblából - egy előkészitő lépéssel - megkapja az azonositólistát /A1,A3/. A sorok számát kell visszaadni csupán /COUNT/ igy nincs szükség a sorok olvasására, a visszaadandó érték 2. A második Válogató példány befejezi müködését, az eredeti pedig elfogadja az "FBI"-t, mint a megoldás reláció egy sorát.

Másodiknak a "CIC"-vel hivja önmagát a Válogató. Az azonositólista egy elemből /A5/ áll, igy ez elvetendő. A harmadik hivás a megoldás relációhoz csatolja a "CIA"-t is. [ASTR 75]

Az algoritmust elemezve, a System R történetét átfogó és ismertető [CHAM 81] cikk megállapitja, hogy:

1. Az Optimizálónak nem csak a sorok olvasásának költségét kell figyelembe vennie, hanem az azonositólisták létrehozásával és manipulálásával keletkező veszteségeket is. Ezek elég tekintélyesek lehetnek.

2. Mértéknek a "beolvasott sorok száma" helyett az I/O-müveletek száma is a CPU-idő sulyozott összege megfelelőbb mutató lenne.

3. Drága megoldás külön tárolni az adatmezők értékét, és külön a rájuk mutató pointerek alkotta sorokat - noha. az XRM védelmében meg kell jegyezni, hogy hosszu adatmezőértékeknél ez a megoldás előnyökkel jár.

4. A "részlekérdezés" mellett be kell vezetni az "illesztés" lekérdezést is. Az Optimizáló szempontjából ennek szimmetriája jobb optimizálási stratégiához vezethet.

5. Nagyobb sulyt kell fektetni a sürün előforduló egyszerübb kérdések hatékony optimizálására.[CHAM 81] szerint a fenti algoritmus tulzottan komplex lekérdezés-orientált.

2.2.4. Az INGRES-ben használt dekompoziciós algoritmus

A módszert a következő példán mutatjuk be: Válogassuk ki azoknak a pozitiv hősöknek, és főnöküknek nevét, akik az Üveghegyen innen dolgoznak, és alapbérük nagyobb, mint a főnöküké.

```
RANGE OF D IS DOLGOZÓ
RANGE OF F IS DOLGOZÓ
RANGE OF R IS RÉSZLEG
RETRIEVE (D.NÉV,F.NÉV)
WHERE     D.FŐNÖK=F.NÉV  AND
          D.RÉSZLEGKÓD=R.RÉSZLEGKÓD AND
          D.BESOROLÁS='POZITIV HŐS' AND
          R.CIM='AZ ÜVEGHEGYEN INNEN' AND
          D.ALAPBÉR > F.ALAPBÉR
```

A dekompoziciós algoritmus leirása két cikkben - [STON 76] és [WONG 76] - is megtalálható, és a két változat lényegesen

különbözik. Mi először a tökéletesebb [WONG 76]- féle változatot vizsgáljuk, majd a két változat különbségéről szólunk.

Az alapgondolat mind a két dolgozatban közös: a többváltozós, bonyolult lekérdezéseket egyváltozósak sorozatára kellene lebontani. Ennek a nyilvánvaló módszernek - valamennyi optimizálási algoritmusnak ez a célja - a megvalósitására két eljárást javasolnak:

1. Helyettesités /tuple substitution/: Tetszőleges n-változós Q lekérdezés /n-1/-változósak sorozatára bontható oly módon, hogy egyik változóját sorban egymás után kicseréljük a változó értelmezési tartományaként szolgáló reláció soraival:

$$Q(X_1,X_2,\ldots,X_n) \rightarrow \{Q_\alpha'(X_2,X_3,\ldots,X_n) \mid \alpha \in R_1\}$$

Látható, hogy n-1 darab helyettesitéssel önmagában elérhető a cél, a dolognak csak az a szépséghibája, hogy ez nem más, mint a Codd-féle redukciós algoritmus /O.2./ Descartes-szorzata, és gyakorlatilag megvalósithatatlan.

2. Leválasztás /detachment/: Ez a lekérdezés olyan Q' és Q'' lekérdezések sorozatára bontását jelenti, melyeknek csak egy közös változójuk van. Példánk lekérdezése nyilván felbontható pl. igy:

```
RANGE OF D IS DOLGOZÓ
RETRIEVE INTO POZITIV_HŐSÖK (D.NÉV,D.FŐNÖK,D.ALAPBÉR)
WHERE D.BESOROLÁS='POZITIV HŐS'

RANGE OF D IS POZITIV_HŐSÖK
RANGE OF F IS DOLGOZÓ
```

```
RANGE OF R IS RÉSZLEG
RETRIEVE (D.NÉV,F.NÉV)
WHERE D.FŐNÖK=F.NÉV  AND
      D.RÉSZLEGKÓD=R.RÉSZLEGKÓD  AND
      R.CIM='AZ ÜVEGHEGYEN INNEN'  AND
      D.ALAPBÉR > F.ALAPBÉR
```

A két eljárást megvizsgálva látható, hogy a leválasztás általában olcsóbb, mint a helyettesités. Esetünkben az ötsoros Dolgozó relációból hármat kiszür az első lekérdezés, lényeges megtakaritást eredményezve a második lekérdezésben, ha azt ezek után pl. tisztán helyettesitésekkel kivánjuk megoldani. Ez önmagában véve persze nem döntő, előfordulhatott volna, hogy az első lekérdezés egyetlen sort sem szür ki. A másik - ennél érdekesebb - probléma az, hogy hogyan érdemes két részlekérdezésre bontani egy lekérdezést. Nyilvánvaló az is, hogy általában csak leválasztással nem lehet egyváltozós lekérdezésekre visszavezetni egy lekérdezést, szükség lesz helyettesitésre is.

A két emlitett dolgozat az optimális szétválasztások meghatározásának módszerében különbözik. [WONG 76] algoritmusa négy lépést tartalmaz:

a/ A redukció /reduction/ feladata a lekérdezés komponensekre bontása vagyis a leválasztások megállapitása. Ennek az algoritmusnak ismertetéséhez néhány fogalmat kell bevezetni:

Legyen Q a

$$\text{RANGE OF}(X_1,X_2,\ldots,X_n)\ \text{IS}\ (R_1,R_2,\ldots,R_n)$$
$$\text{RETRIEVE}\ T(X_1,X_2,\ldots,X_m)$$
$$\text{WHERE}\ B''(X_1,X_2,\ldots,X_m)\ \text{AND}$$
$$B'(X_m,X_{m+1},\ldots,X_n)$$

általános lekérdezés! $X_1,\ldots,X_n$ változók $R_1,\ldots,R_n$ relációkon definiáltak, és $X_1,\ldots,X_m$ változóknak a speciális alaku feltételeknek - B" AND B' - eleget tevő sorait, ill. ezeknek egyes elemeit kivánjuk kiválogatni/. Kézenfekvő Q-t két komponensre bontani. Az első Q' :

RANGE OF $(X_m,X_{m+1},\ldots,X_n)$ IS $(R_m,R_{m+1},\ldots,R_n)$

RETRIEVE INTO R'_m $(T'(X_m))$
WHERE $B'(X_m,X_{m+1},\ldots,X_n)$

Itt $T'(X_m)$ a második, Q" komponensnek szükséges információ. Q" a következő lekérdezés:

RANGE OF $(X_1,X_2,\ldots,X_m)$ IS $(R_1,R_2,\ldots,R'_m)$

RETRIEVE $T(X_1,X_2,\ldots,X_m)$
WHERE $B"(X_1,X_2,\ldots,X_m)$

Az ilyen leválasztást nevezzük redukciónak. Ha a leválasztás eredményeképpen B' nem függ X_m-től, Q-t diszjunkt /disjoint/ lekérdezésnek nevezzük. Ez azt jelenti, hogy az eredeti Q lekérdezésben a B' feltétel akár el is hagyható /ha legalább egy $R_{m+1},\ldots,R_n$ sorkombinációra igaz - ellenkező esetben Q eredménye üres halmaz/. Látható tehát, hogy Q' és Q" között az egyetlen kapcsolat az X_m közös változó. Q-t összefüggő /connected/ lekérdezésnek mondjuk, ha nem választható le belőle diszjunkt lekérdezés. Végül Q irreducibilis, ha nem választható le belőle a fenti módon semmiféle Q' /nem redukálható/.

Példánk nyilvánvalóan összefüggő lekérdezés. Ha elhagynánk belőle a "D.RÉSZLEGKÓD=R.RÉSZLEGKÓD" feltételt nyilvánvalóan le tudnánk belőle választani a

```
RANGE OF R IS RÉSZLEG
RANGE OF D IS DOLGOZÓ
RETRIEVE INTO D"(D.NÉV,D.FŐNÖK,D.BESOROLÁS,D.ALAPBÉR)
WHERE R.CIM='AZ ÜVEGHEGYEN INNEN'
```

diszjunkt lekérdezést.

Lássuk tehát a leválasztási algoritmust! Nyilvánvaló, hogy pl. példánk lekérdezése többféleképpen is felbontható. [WONG 76] algoritmusának nagyon lényeges alapgondolata, hogy a felbontásnak redukcióval kell történnie. A dolgozat algoritmust ad arra, hogyan végezhető el a redukció:

Feltesszük, hogy a WHERE feltétel konjunktiv normálformára van hozva. A feltétel un. karakterisztikus mátrixát /incidence matrix/ ugy képezzük, hogy sorai a normálforma $F_1,F_2,\ldots,F_m$ tényezőinek, plusz egy sor a RETRIEVE-ben szereplő eredmény-listának, oszlopai pedig az $X_1,X_2,\ldots,X_n$ változóknak feleljenek meg, a_{ij} eleme pedig aszerint legyen 1 vagy 0, hogy a F_i tényezőben szerepel-e az X_j változó. Példánk karakterisztikus mátrixa /a feltételeket felülről lefelé számozva/:

	D	F	R
F1	1	1	0
F2	1	0	1
F3	1	0	0
F4	0	0	1
F5	1	1	0
E	1	1	0

vagy az áttekinthetőség kedvéért összevonva az azonos sorokat:

	D	F	R
/F1,F5,E/	1	1	0
F2	1	0	1
F3	1	0	0
F4	0	0	1

Egyszerü algoritmussal ellenőrizhető a lekérdezés összefüggősége: Vegyük sorban az oszlopokat, és egy adott oszlopra azokkal a sorokkal, ahol az oszlopban 1 áll végezzünk elemenként logikai vagy müveletet. A müvelet eredményét tartsuk meg, az operandusaiként szolgáló sorokat huzzuk ki! Az első lépés, a D oszlop vizsgálata után mátrixunk igy néz ki:

	D	F	R
/F1,F5,E,F2,F3/	1	1	1
F4	0	0	1

Ha végeredményként nem egy sorból álló mátrixot kapunk /ilyen lesz példánk karakterisztikus mátrixa, ha az F2 sort elhagyjuk belőle/, akkor a lekérdezés nem összefüggő, és a sorok reprezentálják a diszjunkt komponenseket.

A redukciós algoritmus azon az észrevételen alapul, hogy a redukálható lekérdezés a közös változó elhagyásával nem összefüggővé válik. Hagyjuk el tehát sorban a változókat, /a karakterisztikus mátrix oszlopait/, és ha valamelyik elhagyásával nem-összefüggő lekérdezést nyerünk, akkor megvan a közös változó. A következő lépéshez elő kell késziteni az un. redukált karakterisztikus mátrixot. Ez a karakterisztikus mátrix sorainak átrendezésével nyerhető. Felülről lefelé a sorrend a következő:

/1/ egyváltozós sorok /kivéve az eredménylistát/
/2/ a közös változót tartalmazó, és az eredménylistát nem tartalmazó sorok
/3/ az eredménylistát nem tartalmazó további sorok
/4/ az eredménylista

Ez esetünkben:

	D	R	F
F3	1	0	0
F4	0	1	0
F2	1	1	0
/F1,F5,E/	1	0	1

Megjegyezzük még, hogy noha példánkban a redukció eredménye két leválasztott irreducibilis komponens, ez nem mindig van igy a redukció eredménye több komponens is lehet.

b/ A redukált karakterisztikus mátrixot a lekérdezés ütemező /subquery sequencing/ veszi át. A mátrix első többváltozós sorát veszi, hozzáteszi azokat az egyváltozós sorokat, melyekben a többváltozós sor változói szerepelnek, és ebből lekérdezést képez, majd a felhasznált sorokat kihuzza a mátrixból. Az előállitott lekérdezést a helyettesitő eljárásnak továbbitja. Következő hivásakor uj lekérdezést generál, amig el nem fogy a mátrix. Esetünkben két lekérdezést generál:

```
RANGE OF D IS DOLGOZÓ
RANGE OF R IS RÉSZLEG
RETRIEVE INTO D1(D.NÉV,D.FŐNÖK,D.ALAPBÉR)
WHERE D.BESOROLÁS='POZITIV HŐS' AND
      R.CIM='AZ ÜVEGHEGYEN INNEN' AND
      D.RÉSZLEGKÓD=R.RÉSZLEGKÓD

RANGE OF D IS D1
RANGE OF F IS DOLGOZÓ
RETRIEVE (D.NÉV,F.NÉV)
WHERE D.FŐNÖK=F.NÉV  AND
      D. ALAPBÉR> F.ALAPBÉR
```

c/ A többváltozós irreducibilis lekérdezést helyettesitéssel oldjuk meg. Ennek első - döntő jelentőségü - lépése annak eldöntése, hogy melyik változót - illetve annak értékkészleteként szolgáló relációt - jelöljük ki helyettesitésre. Ezt a döntést több tényező befolyásolhatja.

Legyen $Q(X_1,X_2,\ldots,X_n)$ a vizsgálandó lekérdezés, $X_1,X_2,\ldots,X_n$ értelmezési tartománya $R_1,R_2,\ldots,R_n$! Tegyük fel, hogy X_i-t választjuk ki soronkénti helyettesitésre. Minden $\alpha \in R_i$-re $Q_i(\alpha)$ lekérdezés generálódik. Durva becslés alapján ekkor a Q lekérdezés megválaszolásának költsége:

R_i számossága x Q_i megválaszolásának költsége

Az első, kézenfekvő gondolat tehát a legkisebb számosságu R_i-nek megfelelő X_i kiválasztása. Ez azonban nem mindig optimális, mert

a/ az egyváltozós részlekérdezések feldolgozásával $R_1,R_2,\ldots,R_n$ számossága megváltozhat. Persze az is kérdés, hogy érdemes-e mindegyik egyváltozós részlekérdezést külön feldolgozni. Ha a benne résztvevő változó a helyettesitésre kijelölt, nyilván érdemes. Az INGRES egyszerüen valamennyi egyváltozós lekérdezést feldolgozza mielőtt döntene a változókiválasztásról, de [WONG 76] más módszert is közöl.

b/ X_i megválasztása nyomán létrejövő Q_i lekérdezés bonyolultsága i-vel igen változó lehet.

[WONG 76] a következő közelitéseket javasolja. Nyilvánvaló, hogy $C(Q)$-val jelölve a Q lekérdezés minimális költségét:

$$C(Q) = \min_i \{ \sum_{\alpha \in \overline{R}_i} C(Q_i(\alpha)) \}$$

ahol $\overline{R}_i$, R_i egyváltozós lekérdezések után még szóbajöhető

soraiból összeállitott reláció. Ezek után, ha $C(Q_i(\mathcal{A}))$-t i-től függetlennek tételezzük fel, akkor a minimális számosságu $\overline{R}_i$-t kell helyettesitésre kijelölni. 1976 januárjában ez volt az INGRES stratégiája. [WONG 76] ennél bonyolultabb és pontosabb módszereket is közöl, az optimális helyettesités megválasztására.

Példánk első lekérdezésének feldolgozása az egyváltozós részlekérdezések feldolgozásával kezdődik. Ezek létrehozzák a POZITIV_HŐS relációt /a Dolgozó reláció 1. és 4. sora tartozik bele/, és az ÜVEGHEGYEN_INNEN relációt /a Részleg 2. és 3. sorából/. Ezután lekérdezésünk ekvivalens lesz a

```
RANGE OF D IS POZITIV_HŐS
RANGE OF R IS ÜVEGHEGYEN_INNEN
RETRIEVE INTO D1 ( D.NÉV,D.BESOROLÁS,D.ALAPBÉR)
WHERE D.RÉSZLEGKÓD=R.RÉSZLEGKÓD
```

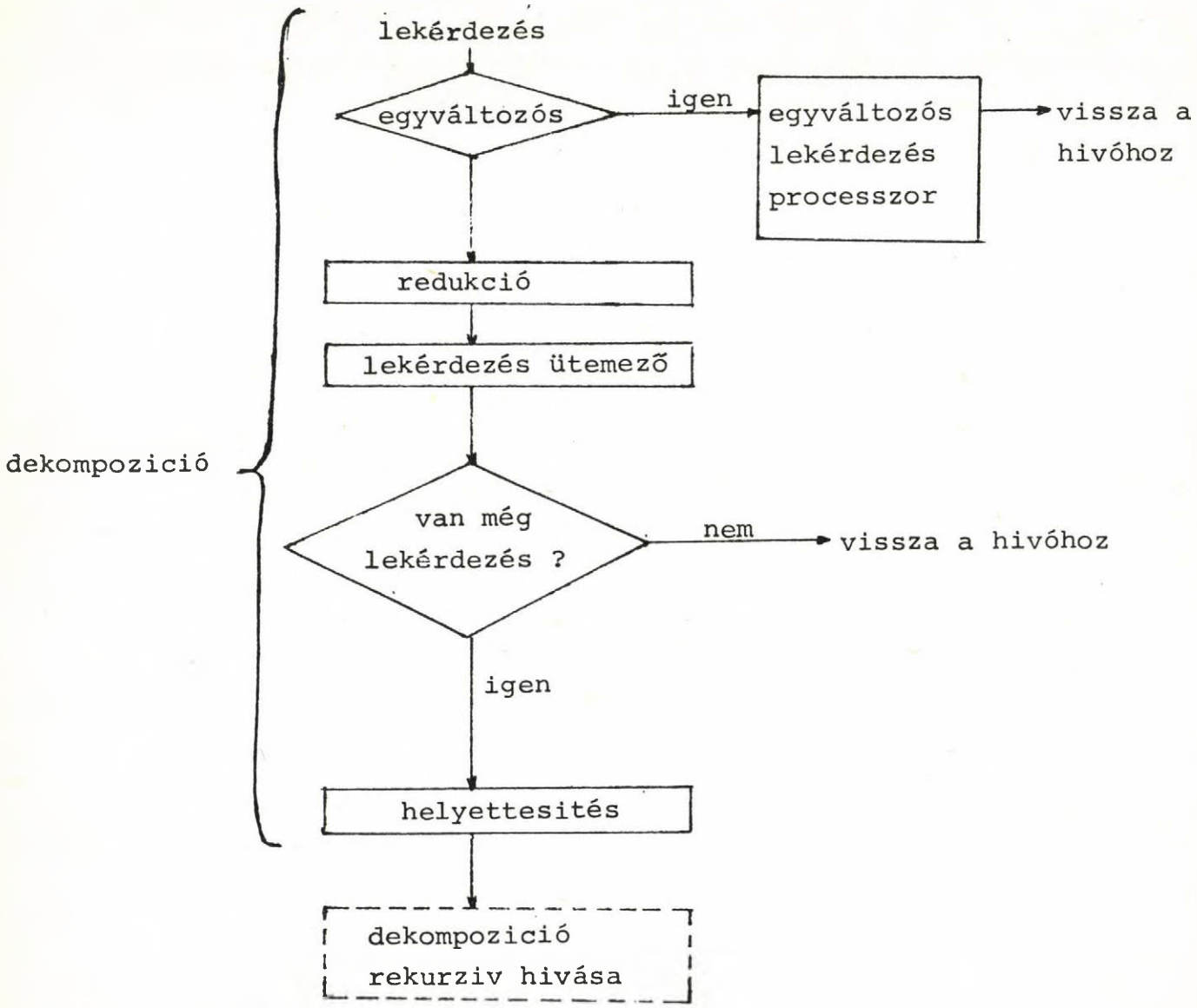
A két reláció számossága egyaránt 2, jelöljük ki helyettesitésre pl. R-t. Az algoritmus R első sorát helyettesitve generálja a

```
RANGE OF D IS POZITIV_HŐS
RETRIEVE INTO D1(D.NÉV,D.BESOROLÁS,D.ALAPBÉR)
WHERE D.RÉSZLEGKÓD='CIC'
```

lekérdezést, és rekurzive hivja önmagát a redukció lépéstől kezdve. Jelen esetben nem lesz szükség dekompozicióra, igy a rekurziv példány mindössze az egyváltozós lekérdezéseket megoldó processzort hivja meg, és visszatér. Üres listával, ugyanis egyik pozitiv hősünk sem dolgozik a CIC-nél. Az eredeti példány most a következő részlegkódot /CIA/ helyettesiti, és ismét rekurziv hivás következik. Ennek eredményeképpen a Dolgozó reláció 4. sora /Hó Fehérke/

bekerül a D1 relációba.

Az algoritmus most visszatér a lekérdezés ütemezőhöz, és a második lekérdezést is feldolgozza - teljesen hasonlóan az előzőhöz. Itt D1 egyetlen sora kihull a rostán. Az algoritmus blokkdigaramja a 21. ábrán látható.



21. ábra

[STON 76]-ban közölt algoritmus és a fent ismertetett között a leglényegesebb különbség a redukció hiánya. A lekérdezés - példánknál maradunk továbbra is - azonnal /ha nem egyváltozós/ a helyettesitő eljáráshoz kerül. Ez leválasztja róla az egyváltozós lekérdezéseket és az egyváltozós lekérdezés processzor megoldja ezeket. Tehát lekérdezésünkből a

```
RANGE OF F IS DOLGOZÓ
RANGE OF D IS POZITIV_HŐS
RANGE OF R IS ÜVEGHEGYEN_INNEN
RETRIEVE (D.NÉV,F.NÉV)
WHERE D.FŐNÖK=F.NÉV  AND
      D.RÉSZLEGKÓD=R.RÉSZLEGKÓD  AND
      D.ALAPBÉR > F.ALAPBÉR
```

lekérdezés lesz, ahol a "Pozitiv_hős" és az "Üveghegyen_innen" relációk ugyanazok, mint az előbb. Ennek a megoldása helyettesitéssel történik, hasonlóan választva meg a helyettesitendő változót, mint az előbb. A programozástechnikai megoldás ugyancsak a rekurziv hivás.

Az ideiglenes relációk - esetünkben pl. "Pozitiv hős" "Üveghegyen innen" - szervezésével [STON 76] részletesen foglalkozik. Ezeket a közvetlen hozzáférés kedvéért hash-elve szervezi, a hash-kulcs megválasztása csak a probléma. Az "Üveghegyen innen" relációnál egyértelmü, hogy Részlegkód szerint érdemes szervezni, hiszen a továbbiakban ezt az oszlopát használjuk csak. A "Pozitiv hős" esetén más a helyzet, szóba jöhet a Főnök, és a Részlegkód is /az Alapbér egyenlőtlenségben szerepel, igy kiesik/. Ha pl. az algoritmus - [STON 76]szerint ilyenkor találomra dönt - a Főnök szerint hash-el, és az első helyettesitendő reláció az "Üveghegyen innen" lesz, akkor a "Pozitiv hős"

relációt ujra kell szervezni, hogy Részlegkód szerint közvetlenül elérhetővé váljon.

[STON 80] az INGRES dekompoziciós technikáját értékelve megállapitja, hogy egy fontos esetet rosszul kezel. Abban az esetben ugyanis, ha egyenlőség feltétellel kell illeszteni, gyakran a legcélszerübb megoldás a két relációt rendezni az illesztési feltételekben szereplő mezők szerint és aztán összefésüléssel /merge/ lehet az eredmény relációt megkapni. Ezt az esetet a cikk szerint külön kellene kezelni.

2.2.5. Az SQL/DS optimizációs módszerei. A "forditóprogram" implementáció

Az SQL/DS optimizálási stratégiájának alapját - [CHAM 81] szerint - Blasgen és Eswaran [BLAS 77]-ben közölt vizsgálatai képezik. Ők a három alapvető relációalgebrai müveletből - egyenlőség feltételü illesztés, korlátozás /kiválasztás/, projekció - álló lekérdezéseket vizsgáltak APL modelleket felhasználva. Cikkük 4 lehetséges algoritmust közöl ilyen lekérdezések megválaszolására:

1. Az illesztendő oszlopok szerinti index felhasználása: Tegyük fel, hogy az illesztendő oszlopok szerint mind a két relációban /R és S/ létezik index! Ekkor ezeket használva keressük az olyan párokat, melyek illeszthetők, vagyis a megfelelő elemeik egyenlőek. Tegyük fel, hogy találtunk egy ilyen párt! Ekkor az egyik /mondjuk $r \in R$/ sort beolvassuk, és ellenőrizzük, hogy a korlátozás feltételét kielégiti-e. Ha igen, akkor S indexét felhasználva, az összes r-hez illeszthető $s \in S$ párt megvizsgáljuk az S-re vonatkozó korlátozás szempontjából, és a megfelelőkre alkalmazzuk a projekciót, majd ideiglenes tárolóra helyezzük

őket. Most R indextáblájában keressük meg az összes r-évél egyező kulcsu /tehát illeszthető/ sort, ezekre alkalmazzuk a korlátozás feltételét, és a megfelelőket - a projekció után - az ideiglenes tárolón lévő s-ekkel illesztjük, az eredményt az output relációba helyezve.

2. A relációk rendezése: Végigolvasva a két relációt a korlátozó feltételt kielégitő összes sor megfelelő projekcióját W_1 és W_2 file-okra irjuk. A két file-t összerendezve, az eredményül kapott file szekvenciális olvasásával könnyen kapható az output reláció.

3. Többszörös olvasás: S sorait olvassuk sorban. Ha valamelyik s sora megfelel a korlátozás feltételének, akkor arra alkalmazzuk a projekciót, és az eredményt egy W_2', központi memóriában felépitett adatszerkezetbe /lehet fa, hash-tábla, rendezetlen adathalmaz, stb./ kiséreljük meg elhelyezni. Ha W_2'-ben nincs hely, és az s illesztendő mezőjében lévő érték kisebb, mint a W_2'-ben lévő sorok hasonló mezőjében szereplő maximális érték, akkor töröljük a maximális értékü sorokat, és s-t W_2'-be illesztjük, ellenkező esetben s-t egyáltalán nem illesztjük W_2' -be. Miután S-t végigolvastuk, R-ben keresünk a korlátozó feltételt kielégitő sorokat. Ha egy ilyen r sorra bukkantunk, megkisérelünk W_2'-ben hozzáilleszthető sort találni.

Ha S-ben több sor van, mint amennyinek a projekciója W_2'-ben elfér, akkor az előző eljárást ismételjük, természetesen az előző menetben felhasznált sorokat már kihagyva /ehhez az előző menetben használt maximális illesztési mezőértéket kell megjegyeznünk/.

4. Sorazonositó /TID/ algoritmus: Tegyük fel, hogy mind a két relációban ugy az illesztési oszlopokra, mint

a korlátozó feltételben érdekelt oszlopokra van index! Ez utóbbi indexeket használva a megfelelő sorok azonositóiból összeállitjuk az R' és S' file-okat, majd különkülön rendezzük őket. Ezek után az illesztendő oszlopok szerinti indexeket használva kikeressük azokat az /A1,A2/ azonositópárokat, melyek illeszthető soroknak felelnek meg, és ellenőrizzük, hogy A_1 ill. A_2 szerepel-e R' ill. S'-ben. Ha mindez teljesül, a sorokat beolvasva, a projekciót elvégezve kapjuk az eredmény egy sorát.

Ennek a négy illesztési algoritmusnak várható költségét becsüli [BLAS 77] olyan változók függvényében, mint

- a reláció sorainak száma;
- a reláció által lefoglalt adatlapok száma;
- az indexben szereplő különböző értékek száma;

stb. Ezeket a statisztikákat a rendszer vezetheti /az SQL/DS meg is teszi/. A számolást bonyolitja, hogy egy index lehet CLUSTERING tulajdonságu /ld. 1.1.1./, ami persze elérés szempontjából kedvező.

Megjegyezzük, hogy még nagyon sok, a fentiekhez hasonló algoritmus gyártható, csupán az ezekben felhasznált tárolási fogalmak /index, rendezés, sorazonositó/ segitségével, és uj fogalmakat /pl. hash/ bevezetve az egész még tovább bonyolitható.

[BLAS 77] több érdekes következtetésre jut:

- mindegyik algoritmusra létezik olyan gyakorlatban előforduló helyzet, melyben optimális /a fenti négy közül/;
- az index CLUSTERING tulajdonságának komoly jelentősége van. /Ehhez meg kell jegyezni, hogy az SQL/DS az egyes relációkat egymástól nem elkülönitve, közös adatlapokon tárolja, igy egy reláció sorai nagyon szétszóródhatnak - hacsak egy CLUSTERING

elemzés után. Az Optimizáló minden SELECT-re külön optimizál.

Az első lépésben történik a nézőpontok feldolgozása. Ez annyit jelent, hogy a nézőpont definiciója bekerül a kérdés feltételei közé. Pl. az 1.1.1. e-ben definiált "Programozási Osztály" nézőpontra vonatkozó

```
SELECT NÉV,CIM
FROM PROGRAMOZÁSI_OSZTÁLY
WHERE ALAPBÉR > 4000
```

lekérdezés átalakul a

```
SELECT DOLGOZÓ.NÉV,RÉSZLEG.CIM
FROM DOLGOZÓ,RÉSZLEG
WHERE DOLGOZÓ.ALAPBÉR > 4000 AND
      DOLGOZÓ.BESOROLÁS='PROGRAMOZÓ'  AND
      DOLGOZÓ.RÉSZLEG=RÉSZLEG.RÉSZLEGKÓD
```

lekérdezéssé. /Ezt a technikát Stonebraker javasolta [STON 76]-ban, és természetesen az INGRES is használja. Nézőpontokon kivül integritási, konzisztencia feltételek kezelésére is alkalmazható./

Most következik a tulajdonképpeni optimizálás, az elérési ut megválasztása. Sajnos az ezt részletesebben leiró, sürün hivatkozott dolgozatot - egy Boston-ban tartott konferencia kiadványában szerepel - nem sikerült megszerezni, de a források alapján a következőképpen képzeljük el:

A beérkező fát az Optimizáló a fentiekben vizsgált három-müveletes /illesztés, korlátozás, projekció/ lépésekből álló sorozatra bontja. Ezután fát képez oly módon, hogy a fa egy-egy szintje egy ilyen kétrelációs müveletnek feleljen meg. A döntési szabadság minden szinten az, hogy az eljáráskészlet melyik algoritmusával végezze

index nem helyezi őket egymáshoz közel/;
egyszerű számolással összehasonlitható két lehetséges elérési ut /módszer/.

Mindebből a következő javaslat adódik: az Optimizáló a lehetséges elérési utakat vegye számba, a költségeiket - durva előzetes válogatás után - becsülje meg /a rendszer által vezetett statisztikai adatok felhasználásával/, és válassza a legolcsóbbat! [BLAS 77].

A System-R rendszer fejlődését összefoglaló - elemző [CHAM 81] cikk az illesztés módszerei közül kettőt emel ki, mint olyat, melyek közül az egyik "az esetek nagy részében közel optimális". Ezek:

1. R korlátozó feltételt kielégitő soraihoz /ezeket R szekvenciális olvasásával vagy index használatával, vagy más módon kapjuk/ keressük ki S megfelelő sorait /indexet használva, de nem minden esetben/.

2. Ha nincs index, rendezzük össze a két relációt, és igy végezzük el az illesztést /[BLAS 77] 2.algoritmusa/.

Az 1. algoritmus szerintünk nem tul pontos, elég sok különböző változatát el tudjuk képzelni, és az adott helyzet dönti el, melyiket érdemes használni. A 2. igazsága elég nyilvánvaló /[STON 80] is hivatkozik rá - ld. 2.2.4./. Valószinüleg arról a szomoru tényről van szó, hogy még ilyen - az általános lekérdezéshez képest viszonylag egyszerü esetben sem létezik az adott helyzettől független, legjobb, vagy közel legjobb univerzális megoldás.

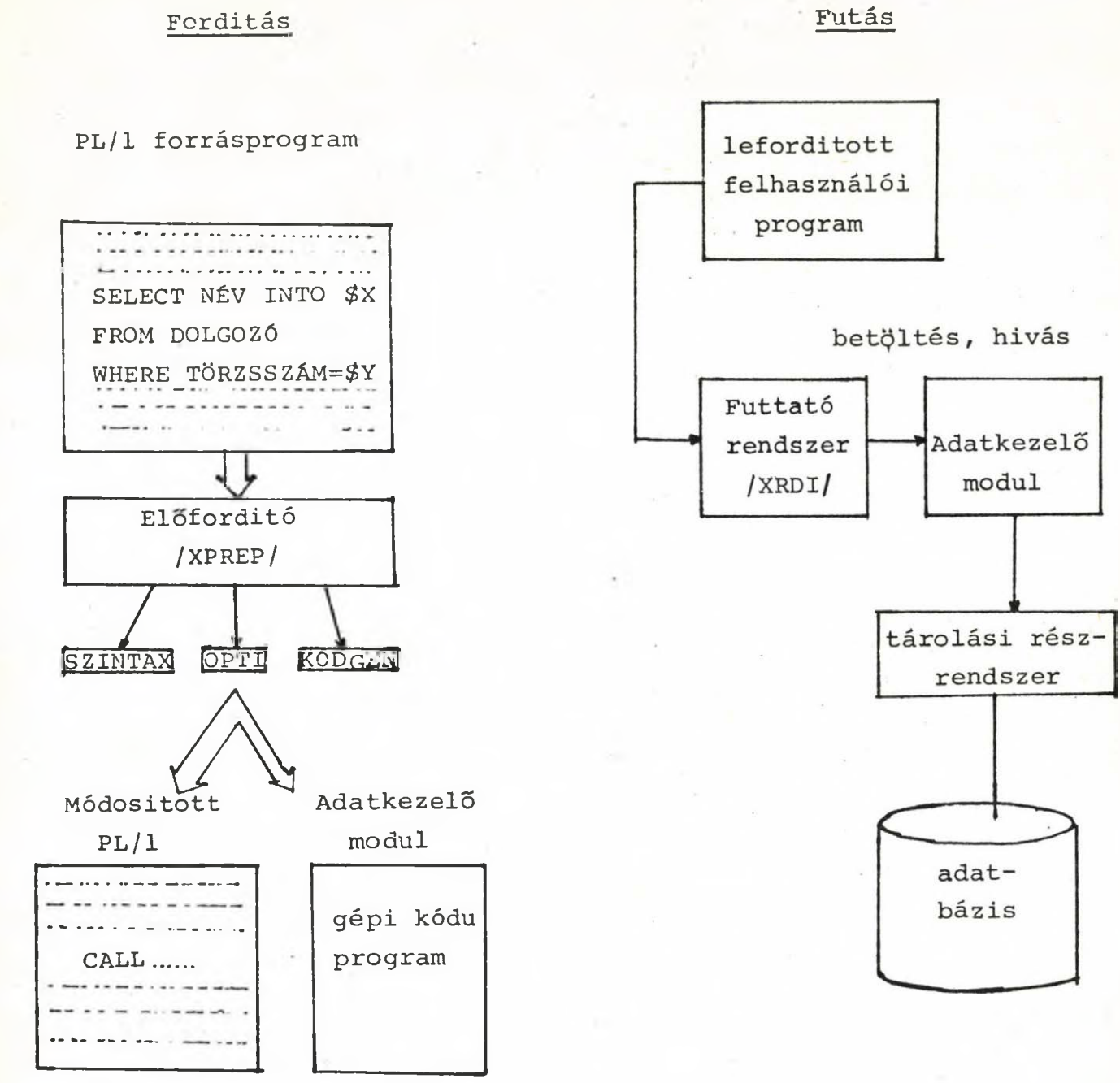
Lássuk most tehát az Optimizáló müködését!
A lekérdezésben szereplő minden egyes SELECT-nek /ezekből több is lehet egy kérdésben - ld. 1.2.3./ egy eredménylista /milyen oszlopokból áll majd az eredmény/ egy FROM-lista /a lekérdezésben résztvevő relációk/, és egy WHERE fa /a diszjunktiv normálformáju feltétel feltehetően 2.2.3.-ban leirt alakja/ felel meg a szintaktikai

el a szintnek megfelelő müveletet. Egy adott szint egy csomópontjából induló elágazások a szóbajöhető eljárásoknak felelnek meg. Minden szinten értékeli az ut költségeit, és az azonos eredményre vezetők közül a legolcsóbb utat tartja csak meg, megkapva végül az optimálist. Hogy egy-egy szinten miként mérlegel, és milyen algoritmusokat használ, arra vonatkozóan pl. [BLAS 77]nyujt támpontot. Az egyes algoritmusok értékelésének mérőszáma a lapozások és a tárolási részrendszer rutinjaihoz fordulások számának /utóbbi a CPU idő jó közelitése/ sulyozott összege. [CHAM 81]

A felhasználói interface nyelvének leképzését adatbázis assemblerre /vagy közvetlen adatmanipulációra/ interpreter vagy forditóprogram-szerü tevékenységnek is felfoghatjuk, egy magasszintü nyelv egy-egy utasitásának egy másik, végrehajtható nyelv egy vagy több utasitását kell megfeleltetni. Attól függően, hogy az igy kapott utasitások végrehajtása azonnal megtörténik, vagy csak tároljuk őket egy későbbi végrehajtás céljára, interpreterről vagy forditóprogramról beszélhetünk.

Ilyen szemszögből a 2.2.1.-2.2.4.-ben vizsgált algoritmusainktól - és általában a relációs adatbáziskezelők optimizálóprogramjaitól - eltérően az SQL/DS Optimizálót forditóprogramnak kell tekintenünk. Müködése a 22. a, és b, ábrákon látható:

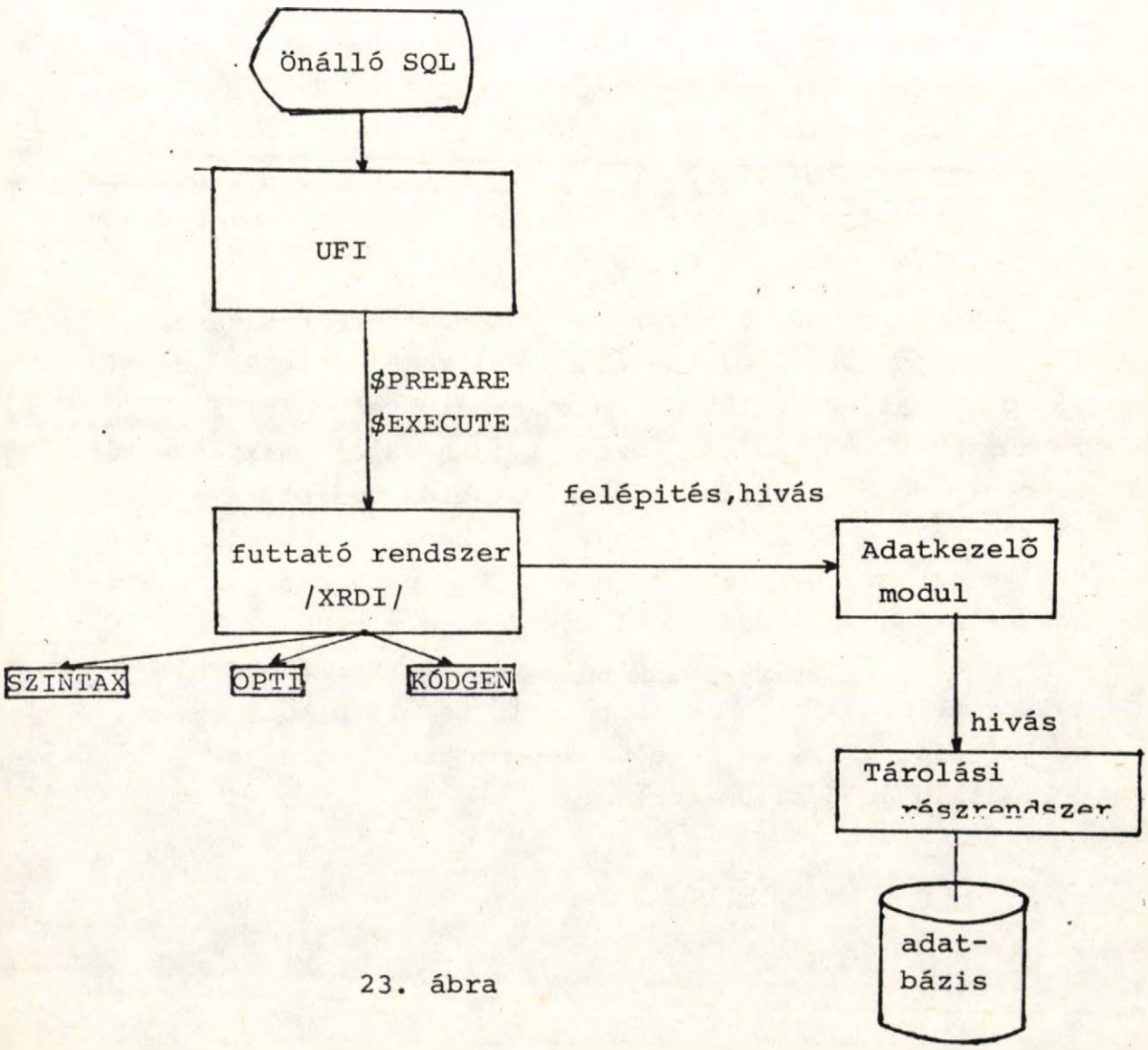
Az /1.3.1.-ben ismertetett/ adatkezelő résznyelv utasitásait tartalmazó PL/1 programot az Előforditó dolgozza fel. Megkeresi a programban lévő adatkezelő utasitásokat, megfelelő PL/1 eljárás-hivásokra cseréli őket, majd a módositott, most már "tiszta" PL/1 programot file-ra irja. Az adatkezelő utasitások feldolgozása három lépésben /Szintaktikus elemző, Optimizáló, Kódgenerátor/ történik, és eredménye a rendszer könyvtárában elhelyezett Adatkezelő Modul /Access Module/ lesz. Ez gépi kódu - vezérlésátadásokat, adatbázis assembler rutinok hivásait tartalmazó - program.


Fordítás
PL/1 forrásprogram
SELECT NÉV INTO $X
FROM DOLGOZÓ
WHERE TÖRZSSZÁM=$Y
Előfordító
/XPREP/
SZINTAX
OPTI
KÓDGEN
Módositott
PL/1
CALL .....
Adatkezelő
modul
gépi kódu
program
a/
Futás
leforditott
felhasználói
program
betöltés, hivás
Futtató
rendszer
/XRDI/
Adatkezelő
modul
tárolási rész-
rendszer
adat-
bázis
b/

22. ábra

Amikor a felhasználó futtatja a programot, az Előfordító által a programba helyezett első CALL végrehajtása teremti meg a kapcsolatot az SQL/DS futtató rendszerével. Ez betölti a megfelelő Adatkezelő Modult, és átadja neki a vezérlést. A modul az adatbázis assembler rutinokat használva az Optimizáló által megválasztott algoritmus szerint bonyolitja az adatok cseréjét a felhasználó programja és az adatbázis között.

Ad-hoc lekérdezés esetén a felhasználó utasitásait az UFI /User Friendly Interface, ld. 2.1.1./ fogadja. Ilyenkor a végrehajtás a 23. ábra szerint történik.



23. ábra

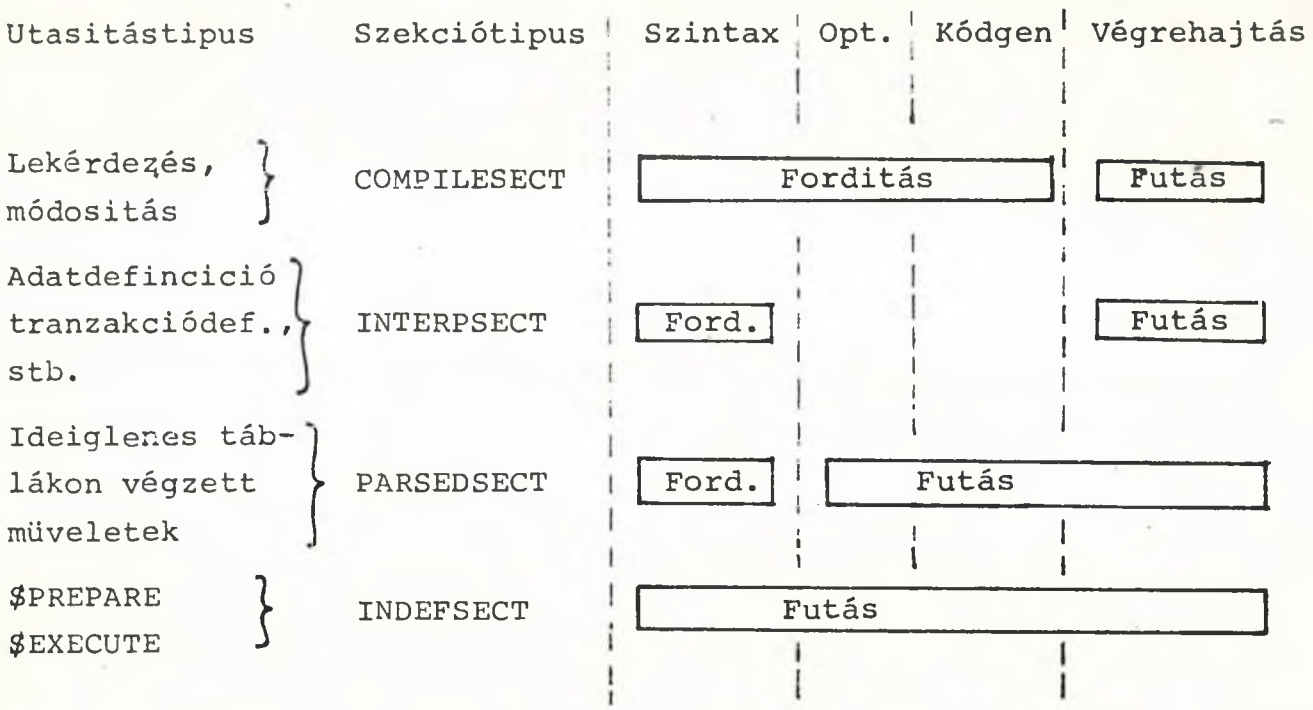
A terminálról érkező parancsokat az UFI fogadja, és megfelelő $PREPARE, $EXECUTE utasitásokon keresztül /ezek persze az előforditott UFI-ban már /PL/1 CALL-ok/ továbbitja a futtató rendszernek. Ez felismerve a $PREPARE és $EXECUTE utasitásokat elvégzi a forditás három lépését, felépiti és meghivja a megfelelő Adatkezelő Modult, amely a szokásos módon végzi az adatkezelést. /Megjegyezzük, hogy nem csak az UFI, hanem felhasználói program is tartalmazhat $PREPARE-t és $EXECUTE-ot. Ilyen esetekben a feldolgozás menete azonos a 23. ábrán láthatóval/.

A módszer előnyei /[CHAM 81a] szerint/

- a szintaktikus, név megfeleltetési, optimizálási, jogosultsági ellenőrzések nagy része a futás ideje helyett a forditását növeli. Ez főképpen a sokszor futtatott programoknál jelentős,
- az Adatkezelő Modul, mivel egy speciális programhoz készült, hatékonyabb, és sokkal kisebb, mint egy általános SQL interpreter.

Az INGRES rendszer interpretert és nem forditót használ /legalábbis 1980-ban/, de [STON 80] megállapitja, hogy tévedtek, alábecsülték a programozási nyelv interface fontosságát, és megfeledkeztek a [CHAM 81a] emlitette két előny - idő és memórianyereség - fontosságáról. A cikk konkrét számokat közöl a veszteség becslésére, külön kiemelve azt az időt, ami az egyes QUEL parancsoknál a felhasználó jogosultságának ellenőrzésére elmegy.

Most a forditás technikai részleteivel fogunk foglalkozni. A 24. ábra az Adatkezelő Modult alkotó egyes szekciótipusokat, és a velük kapcsolatos események időbeli alakulását illusztrálja:

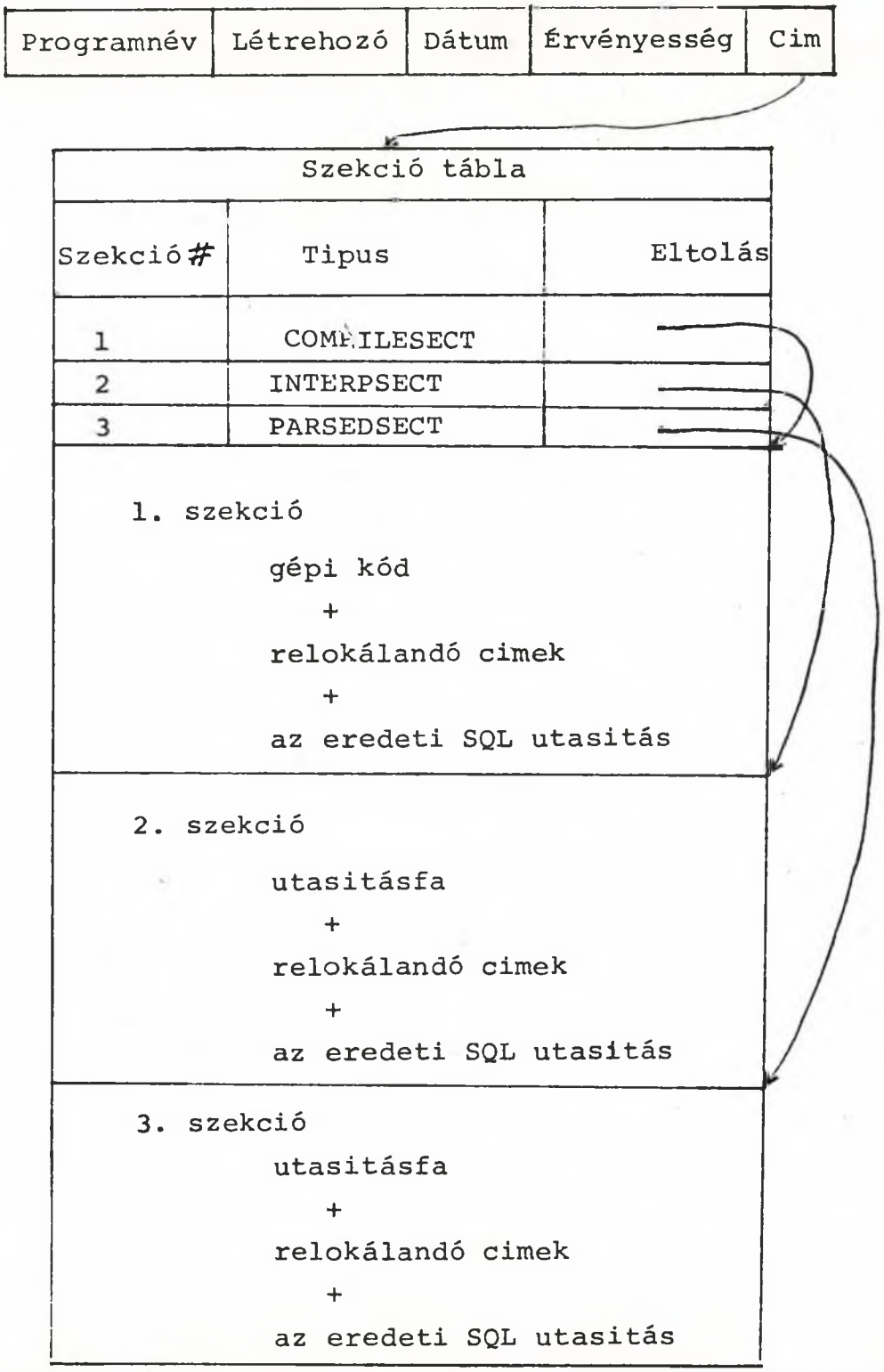


24. ábra

A COMPILESECT jelentése eléggé nyilvánvaló: az SQL utasitás végrehajtása. INTERPSECT olyan utasitásokra jön létre, mely az adatbázis logikai és fizikai szerkezetétől függetlenül mindig ugyanugy, és csak egyféleképpen hajtható végre. Amikor a program ideiglenes relációra hivatkozik, az még nem létezik, igy elérése nem optimizálható. Ilyenkor generál az Előfordító PARSEDSECT-et, melyet a futtató rendszer optimizál, és a generált kódot végrehajtja. Az INDEFSECT jelentése megint nyilvánvaló: a $EXECUTE utasitásokról forditáskor csak annyit tudunk, hogy melyik karaktersorozatban helyezkednek el - igy a szintaktikus elemzéstől a végrehajtásig minden futás közben történik - ez tulajdonképpen interpretálás.

A 25. ábra egy Adatkezelő Modult ábrázol:

A Rendszerkatalógusban tárolt leirás:



25. ábra

A leirásban lévő mezők jelentése elég nyilvánvaló, kivéve az "Érvényességet"-et. Ez a mező jelzi, azt, hogy az Adatkezelő Modul érvényes-e még, vagy ujra kell forditani. Előfordulhat ugyanis, hogy a program forditása és futása közötti időszakban az adatbázis fizikai szervezése megváltozik, pl. egy indexet töröl az adatbázis adminisztrátor, vagy egy erre jogosult felhasználó. Ilyenkor a rendszer megkeresi a katalógusban azokat a Modulokat, melyek ezt az indexet használták, és - az Érvényesség mező segitségével - érvényteleniti őket. Ha a futtató rendszer érvénytelen Modulra való hivatkozással találkozik, ujraforditja azt, anélkül, hogy a felhasználó erről tudomást szerezve.

Az Adatkezelő Modulok több szekcióból állhatnak. A 25. ábrán 3 különböző tipusu szekcióból álló Modult láthatunk, COMPILESECT az egyetlen, mely gépi kódu utasitásokat tartalmaz, a másik kettőben ehelyett a szekciót generáló utasitás Szintaktikus elemző által előállitott fája szerepel. Mind a három szekcióban eredeti formájában szerepel a generáló SQL utasitás - erre az ujraforditás miatt van szükség.

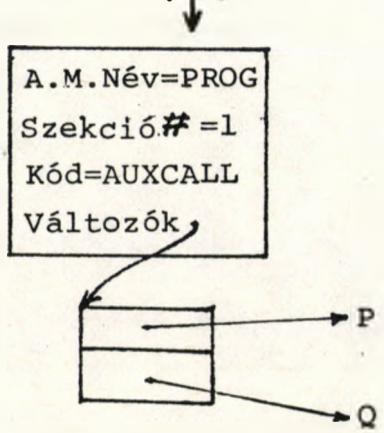
Egy felhasználói program forditását illusztrálja a 26. ábra:

Forrás PL/1

```
$UPDATE DOLGOZÓ
 SET ALAPBÉR=ALAPBÉR+$P
 WHERE TÖRZSSZÁM=$Q;
```

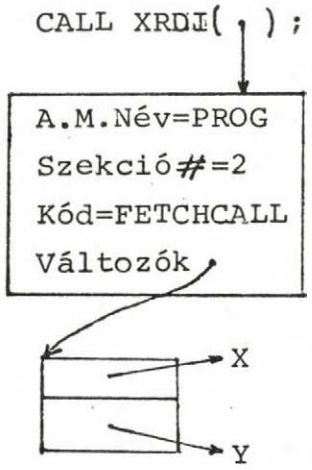
Módositott PL/1

```
CALL XRDI( ↓ );
```

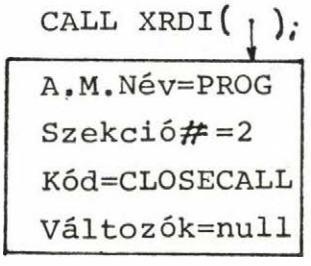


```
$LET C1 BE                                  /Nem történik XRDI hivás,
     SELECT NÉV,ALAPBÉR INTO $X,$Y          hiszen ez csak deklaráció.
     FROM DOLGOZÓ                           A fordító kijelöli a C1-
     WHERE BESOROLÁS=$A;                    nek a 2. szekciót/.
$OPEN C1;
```


```
$FETCH C1;
```




```
$CLOSE C1;
```



26. ábra

Az első XRDI hivás végrehajtja az SQL utasitást. Hivatkozik a program nevére, a szekcióra /ezt mindegyik XRDI hivás megteszi/. A müvelet kódja AUXCALL, a szekció gépi kódu utasitásainak végrehajtását eredményezi. Az utasitásnak ugy az input, mint output változói a hivás paraméterei lesznek. A második XRDI hivásnál - Cl nyitása - a rendszer megjegyzi az utasitás input adatait /v.ö. 1.3.1./, és felkészül a lekérdezés végrehajtására /a gépi kód "OPENCALL" paraméterü végrehajtásával/. A FETCH továbbitja az output paraméterek cimét, és végrehajtja az utasitást.

A $PREPARE utasitás hatására az Előfordító az Adatkezelő Modulban INDEFSECT-et hoz létre, a PL/1 programban pedig az utasitást speciális, "SETUPCALL" paraméterü XRDI hivásra cseréli. Futás közben ennek hatására a futtató rendszer /az XRDI/ a karaktersorozat tartalmát leforditja, és az INDEFSECT-et erre a COMPILESECT-re cseréli /persze csak a memóriában, és nem a Rendszerkatalógusban lévő példánynál/.

A $EXECUTE közönséges "AUXCALL" paraméterü XRDI hivást és ezzel a $PREPARE generálta COMPILESECT végrehajtását eredményezi. Ha ismét ugyanarra a karaktersorozatra /szekcióra/ vonatkozó $PREPARE utasitással /"SETUPCALL" paraméterü hivással/ találkozik az XRDI, a szekció régi tartalma - most már COMPILESECT - elvész, helyébe a karaktersorozat aktuális értéke szerint generálódik szekció, és a következő $EXECUTE már ezt fogja végrehajtani.

Még egy speciális hivást a "DESCRIBECALL" paraméterüt emlitünk. Ez a felhasználói program

$DESCRIBE ⟨utasitásnév⟩ INTO ⟨tömb⟩

utasitását helyettesiti, és hatására az XRDI a ⟨tömb⟩-ben elhelyezi az ⟨utasitásnév⟩ utasitásban szereplő mezők neveit és tipusait.

2.2.6. Összefoglaló megjegyzések

A megvizsgált algoritmusok néhány közös gondolatot - módszert, észrevételt - tartamaznak. Ezeket nézzük végig még egyszer, megfigyelve, hogy az egyes algoritmusoknál miképp bukkannak fel az azonos ötletek. Szolgálhat némi tanulsággal programozástechnikai - implementációs megoldások vizsgálata is.

Jól láthatóan /az algoritmusok alapján/ érdemes relációalgebrai terminológiát használni. A nem relációalgebrai nyelveknél is világos, hogy melyikben mit jelent a projekció, korlátozás, és az illesztés. Mivel a relációalgebra a legprocedurálisabb, a gép számára is a relációalgebra interface a legkönnyebben érthető, és az optimizálási algoritmusoknak is kiindulópontot ad, atomi müveleteivel, melyek sorozata alkotja a lekérdezést.

A Chang-Smith algoritmus fogalmazta meg az egyváltozós korlátozások mielőbbi elvégzésének és összevonásának elvét. Előnyei nyilvánvalóak: minél korábban érdemes a manipulálandó reláció méretét csökkenteni, és egy relációba tartozó feltételeket nem érdemes a reláció többszöri végigolvasásával ellenőrizni. Nézzük meg ennek az elvnek a megvalósulását és módosulásait az egyes algoritmusoknál!

A Palermo algoritmus minden relación csak egyszer megy végig, tehát az egy relációba tartozó feltételeket a lekérdezésben elfoglalt poziciójuktól függetlenül egyszerre ellenőrzi. Ügyes implementáció - az ismertetett LIDAS-ban erre nincs utalás, de feltételezzük, hogy ilyen - az értéklista készitésénél kihasználja az indexet /ha van/. Az egyváltozós korlátozások illesztés előtti elvégzését jelenti, hogy a párlisták épitésénél adott konjunkcióban lévő egyváltozós korlátozásokat figyelembe veszi.

A TID algoritmusnál is érvényesül ez az elv. Az egy-

változós korlátozások kitüntetett szerephez jutnak /P1 és P2 tipusu feltételek/, és noha az algoritmus index-középpontusága miatt elsősorban a P1-eket tudja jól felhasználni, a P3,P4,P5 feltételek /ezek jelentik az illesztést/ feldolgozása előtt a belső blokkban előbb a P1, majd a P2 tipusuak is kiértékelésre kerülnek.

Mindennek ellenére lehet olyan lekérdezést irni, ahol ez az elv csorbát szenved. Az alábbi lekérdezés azoknak a pozitiv hősöknek a nevét irja ki, akik 1-nél több dolgozót foglalkoztató részlegekben dolgoznak /2.2.1. adatbázisát használjuk/:

```
SELECT NÉV
FROM D IN DOLGOZÓ
WHERE BESOROLÁS='POZITIV HŐS'
      AND RÉSZLEGKÓD IN
          SELECT RÉSZLEGKÓD
          FROM R IN RÉSZLEG
          WHERE 1 >
               SELECT COUNT(*)
               FROM DOLGOZÓ
               WHERE RÉSZLEGKÓD=R.RÉSZLEGKÓD
```

Ennél a lekérdezésnél az algoritmus ha a külső blokk feltételeinél a "Besorolás"-t tartalmazót P2 tipusunak /nincs index "Besorolás"-ra/, a "Részlegkód"-ot tartalmazót pedig P1-nek találja, akkor lefelé haladva elvégzi a Részleg és Dolgozó relációk illesztését, anélkül, hogy a Részleg sorait a "Besorolás"-ra vonatkozó feltétel korlátozná /mint az pl. a Palermo-algoritmusnál megtörténhet/. Persze nyilvánvaló, hogy ezt a lekérdezést egyszerübben is meg lehet irni SEQUEL-ben, oly módon, hogy a korlátozást hajtsa végre előbb az algoritmus.

A PRTV algoritmusát leiró [HALL 76] cikk két fontos megjegyzést tesz evvel az elvvel kapcsolatosan: nem mindig lehet az illesztés előtt korlátozni, és nem is mindig érdemes. Az előbbit természetesen minden algoritmus figyelembe veszi, az utóbbit nem, ugyanis annak eldöntése, hogy mikor érdemes, mikor nem, nehezen becsülhető. Az SQL/DS algoritmus, mely a rendszer által vezetett statisztikákon alapul, tudja talán ezt a becslést legjobban elvégezni, és ennek alapján választani az optimális illesztés-korlátozási-projekció algoritmust.

Az INGRES [STON 76] algoritmusa elsőnek az egyváltozós korlátozásokat hajtja végre /már ha lehet, és az OR-ok ezt nem teszik lehetetlenné/. A módositott [WONG 76] algoritmus már azt emliti, hogy csak akkor érdemes ezt megtenni, ha a korlátozásban szereplő változó valószinüleg helyettesitésre kijelölt. /A helyzet azért bonyolult, mert a korlátozás eredményétől függ, hogy kijelöljük-e a változót helyettesitésre./

Az SQL/DS algoritmusa az egyváltozós korlátozások mielőbbi elvégzésének elvét tulajdonképpen igen kritikusan szemléli, és pl. [BLAS 77] négy algoritmusából is kiderül, hogy ennek az elvnek az érvényessége - legalábbis az SQL/DS fizikai szervezése mellett - nagymértékben függ a lehetséges elérési utaktól.

A Chang-Smith algoritmus projekcióra vonatkozó elve ennél egyszerübb és nyilvánvalóbb. Kisebb méretü /sorhosszuságu/ relációkkal kényelmesebb dolgozni, ésszerütlen lenne a felesleges adatot cipelni. Az algoritmusok általában - ésszerüen - ugy értelmezik ezt az elvet, hogy le kell vágni a szükségtelen adatmezőket, de a duplikátumok eltávolitásával /ami rendezést jelent/ nem foglalkoznak. Érdemes megemliteni még, hogy a sorazonositóval dolgozó algoritmusok /Palermo, TID, [BLAS 77] 4. számu/ szempontjából ennek az elvnek nyilván nincs jelentősége.

Az INGRES filozófiában alapvető szerep jut a helyettesitendő változó kijelölésének, [WONG 76] bonyolult algoritmusokat javasol /nem tudjuk, hogy implementálta-e valamelyiket az INGRES/. [STON 76] egyszerüen azt a változót helyettesiti, mely az egyváltozós korlátozások elvégzése után legkisebbé vált relación van értelmezve.

A Palermo-algoritmus ilyen szempontból még egyszerübb stratégiával dolgozik. Mindig azt a változót helyettesiti, amelyik a legkisebb relación van értelmezve /az egyváltozós lekérdezések figyelembe vétele nélkül/. [WONG 76] ez ellen több érvet is felhoz /ld. 2.1.4./

A Chang-Smith és a PRTV algoritmus a relációalgebrai interface miatt adott sorrendü müveleteket kapnak, a fát módositó algoritmus illesztési sorrendek felcserélésével nem foglalkozik /ilyen szempontból hátrány a relációalgebra nyelv/. Igaz viszont, hogy az illesztési algoritmus egy megadott készletből történő választása a helyettesitendő változó kijelölését /is/ jelenti.

A TID algoritmus a SEQUEL nyelv szerkezete miatt nem foglalkozik a helyettesitendő változó kijelölésével, hiszen azt az egymásba ágyazott blokkok sorrendje eleve - nem feltétlenül célszerüen - meghatározza.

Az SQL/DS statisztikai becslések alapján keresi a helyettesitések optimális sorrendjét.

A programozástechnikai megoldások közül kiemelkedő fontosságu a rekurzió. Ez érthető, hiszen a módszerek általában egyváltozós lekérdezésekre vezetik vissza a többváltozósakat, lépésenként csökkentve a változók számát. Ezt egészen pontosan láttuk a TID és a dekompoziciós algoritmusnál. A Palermo-algoritmusra ez nem igaz, ez inkább változónként egyszerü ciklust használó algoritmusnak tünik. A Chang-Smith algoritmusnál és az SQL/DS algoritmusnál a felépitett fák bejáráshoz, transzformálásához kell a rekurzió.

Mindenesetre célszerünek látszik rekurziót biztositó magasszintü nyelven megirni az Optimizálót. /Ez is a gyakorlat, hiszen: LIDAS-MODULA-2, TID és SQL/DS-PL/1, INGRES-C/.

Az implementációs eltérések közül kiemelkedően fontosnak tartjuk a "forditóprogram" megoldást /nem csak mi, a rendszer alkotói is [CHAM 81]/. Az előnyeiről volt már szó, most a hátrányait emlitjük:

- a kódgeneráláshoz szükséges többletidő;
- az Adatkezelő Modulok tárolásához szükséges lemezterület.

[CHAM 81a] szerint a kódgenerálási többletidő méréseik szerint általában az optimizáláshoz szükséges idő 1/3 része, egy-egy Adatkezelő Modul pedig 1000-1500 byte-ot foglal le.

2.3. A tárolási részrendszer

A tárolási részrendszer feladata a bemenő nyelvén érkező utasitások leképzése operációs rendszer file irás-olvasási utasitásokra és, hogy az I/O-müvelet egységén - legyen az az adatbázis lapja bonyolult belső szervezéssel, vagy egyszerüen egy adott reláció egy sorát tároló file blokkja - eligazodjon. Igyekeztünk óvatosan fogalmazni, mert sokfajta szervezés lehetséges. A 7. ábrán /2.1./ a tárolási részrendszer bemenő nyelve az adatbázis assembler. Ez elég általánosan elterjedt gyakorlat, de nem kizárólagos - pl. relációalgebrai interface-t használó kisebb rendszereknél teljesen hiányozhat az Optimizáló réteg, és legfeljebb szintaktikus elemzés után azonnal a relációkat tároló file-ok elérése következik, a felhasználó által adott müveletek alapján.

Két rendszert - SQL/DS, INGRES, - valamennyire

részletesen ismertetünk, és néhány mikrogépes rendszert is említünk. A nagyobb rendszereknél külön foglalkozunk az adatbázis assemblerrel.

2.3.1. SQL/DS

Az SQL/DS-ben az adatok szegmensekben tárolódnak. Egy szegmens fizikailag is egymást követő lapok sorozata. Egy szegmens egy vagy több VSAM ESDS file-ból áll, ezek blokkjai a lapok /2.1.1./. Egy szegmensen belül több reláció, index, belső katalógus információ is elhelyezkedhet, de minden struktura csak egy szegmenshez tartozhat, nem lóghat át másikba.

Az egy szegmenshez tartozó relációk tárolása a 4K byte méretü lapokon általában rendezetlenül történik, tehát egy lap több különböző reláció sorait tartalmazhatja. /A rendezetlenségen változtat a már többször említett CLUSTERING tulajdonságu index, melynek hatására a rendszer igyekszik a reláció sorait megadott sorrendben, egymáshoz közeli lapokon elhelyezni. A kapcsolat /ld. 1.1.1./ megadása is befolyásolja a sorok elhelyezését: ilyenkor a különböző relációkhoz tartozó összekapcsolt sorok kerülnek lehetőleg egymás közelébe./ Relációk csak a szegmens adatterületén - tehát sorok számára fenntartott lapjain - helyezkedhetnek el.

Egy lap felépitése a pointertömbös szervezési technikát követi. Eszerint a lapon a sorok tárolása egymást követően folyamatosan történik. Mindegyik sor elejére a lap alján elhelyezett tömb egy eleme mutat, megadva a sor elejének eltolását a lap elejétől. Ha valamelyik beillesztett sort törölni kell, vagy megváltozik a hossza, a többi sor megfelelő eltolásával megoldható, hogy a sorok továbbra is folyamatosan helyezkedjenek el. Eltolásnál persze a sorok

elejére mutató pointerek értéke is megváltozik, de a sorszámuk nem, tehát egy adott sor eltolását a lap elejétől továbbra is a tömb ugyanannyiadik eleme őrzi. Ez igen lényeges a sorazonositók szempontjából. Ennél a szervezési technikánál ugyanis a sorazonositó két részből - a lap sorszámából /az SQL/DS-ben szegmensen belüli sorszámából/, és a pointertömb sorra mutató elemének sorszámából - áll. /Ez a lapszervezési technika eléggé gyakori az adatbáziskezelő rendszereknél - ezt használja pl. az SQL/DS mellett az INGRES vagy IDS és az IDMS is./

Egy tárolt sort a lapon minden esetben egy leiró prefix előz meg. Ebben van a reláció azonositója, a kapcsolatot realizáló pointerek száma és maguk a pointerek, a sor tárolt elemeinek száma, stb. Ez a prefix biztositja a reláció kiterjeszthetőségének vagy két reláció közötti kapcsolat megadásának dinamikus lehetőségét /1.1.1./.

Az adatelérést gyorsitják az - SQL/DS-ben "kép"-nek /image/ nevezett - indexek. Ezek egy reláció tetszőleges oszlopkombinációira szervezhetők, és akárhány lehet belőlük /ld. 1.1.1./.

Index szervezésekor - ez a reláció létrejöttét követően bármikor kérhető - az SQL/DS a relációt tartalmazó szegmens erre a célra szolgáló lapjain hozza létre a fizikai strukturát. A lapok B-fa - konkrétabban VSAM KSDS /Key Sequenced Data Set/-szerü hierarchiát alkotnak, a levelekben a mutatók az adatlapokon elhelyezkedő sorok azonositóit tartalmazzák.

A kapcsolat - noha a felhasználói nyelvben szerepel /1.1.1./, és a tárolási részrendszer is felkészült rá - végül is ugy tünik, hogy kimaradt az SQL/DS fizikai elérést gyorsitó repertoárjából. [CHAM 81] szerint az Optimizáló /RDS/ nem használja őket, mert

· a lényeges kapcsolat /essential link/ - ezen a

rendszer számára ismeretlen jelentésü, felhasználó által definiált és fenntartott kapcsolatot árt - a relációs rendszer adatfüggetlenségének elvét sérti;

a lényegtelen kapcsolat /nonessential link/ - ezt a rendszer hozza létre azonos értékü mezőkkel rendelkező sorok között, és ő is tartja fenn - karbantartása tul költséges. Valóban, ha egy mező értéke megváltozik, nem elég, hogy hosszadalmas lehet valamennyi vele kapcsolatban álló sorban a pointert törölni - ez azért oda-vissza pointerezéssel megoldható - de utána meg kell keresni az uj mezőértékhez tartozó sorokat és összekapcsolni őket.

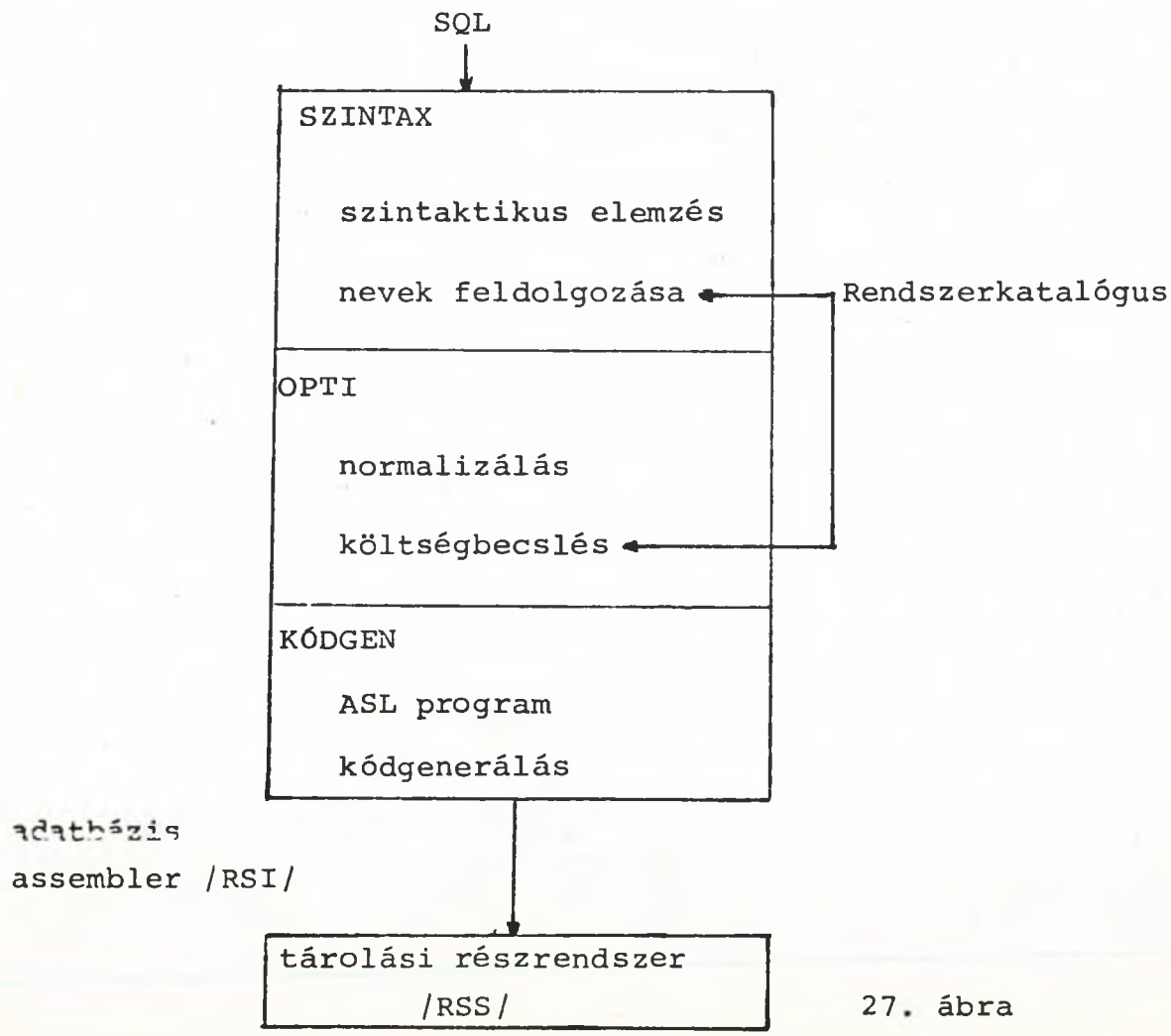
Most rátérünk az SQL/DS adatbázis assemblerének ismertetésére. Előljáróban szeretnénk rámutatni egy lényeges különbségre az SQL/DS és a többi rendszer Optimizálójának müködése között. Vizsgáljuk meg ehhez, hogy zajlik általában egy lekérdezés végrehajtása egy relációs adatbáziskezelő rendszerben!

Az Optimizáló müködésének eredményeképpen jön létre a válaszadás algoritmusa. A LIDAS esetében ennek első lépése egy döntés: az X változó értékelésével kell kezdeni. A döntésből adódik az algoritmus kezdete: tekintem először az első olyan tagját a diszjunktiv normálformának, melyben részt vesz az X, és értéklistát készitek rá. Ez az algoritmus két rutin hivását, és müködésük összehangolását jelenti ciklusban: az egyik rutin a feltételnek eleget tevő sorokat választja ki, a másik az előző által visszaadott sorok alapján készíti az értéklistát. Mind a két rutin az adatbázis assembler része lehet. Következő lépés: tovább kell menni a normálformában, ugyanezt megismételni a következő tagra, azután a következő változóra, s.i.t. Nem elég tehát, hogy kész rutinok vannak a relációk soronkénti manipulálására: ezek müködését vezérelni kell, fel kell

építeni a válaszadás algoritmusát az elemi müveletekből. Ezt a tevékenységet a LIDAS-nál vagy az INGRES-nél maga az Optimizáló látja el.

Más a helyzet az SQL/DS esetében. Itt futtatható, "önálló" kódot kell létrehozni az Optimizálónak, hiszen mikor az Adatkezelő Modul müködik, ő már nem lesz a memóriában, nem vezérelheti a tárolási részrendszer rutinjainak müködését - a kódnak tehát a vezérlési szerkezeteket, elágazási lehetőségeket, stb. is tartalmaznia kell.

Ebből adódik, hogy az SQL/DS esetében két szintről beszélhetünk: itt is megvan az adatbázis assembler rutinok készlete éppen ugy, mint a többi rendszerben, de ezek felett még egy önálló nyelv van: az ASL /Access Specification Language/. A 27. ábra a 22. és 23. ábrán már bemutatott SZINTAX, OPTI és KÓDGEN modulok kapcsolatát világitja meg:



27. ábra

Az OPTI tehát itt ASL nyelvü programot generál az algoritmus közvetlen végrehajtása /ill. adatbázis assembler rutinokkal való végrehajtatása/ helyett. Ez az ASL nyelvü program azután gépi kódu utasitássorozattá válik, melyben a vezérlési szerkezet RSI rutinhivásokon keresztül realizálja az Optimizáló által tervezett algoritmust.

Vegyük sorra először az RSI rutinokat!

- OPEN - relációban, indexben, kapcsolatban vagy korábban létrehozott listában keresés kezdeményezése. Keresésazonositót ad vissza.
- NEXT - bemenő paramétere az OPEN által visszaadott keresésazonositó. Az OPEN-nel megkezdett keresés következő elemét adja vissza. A NEXT-nek paraméterként a keresés objektumára vonatkozó feltételt is be lehet adni. Ilyenkor a feltételnek megfelelő következő sort adja vissza. Indexben való keresésnél nem az index, hanem az indexelt reláció sorait adja vissza.
- PARENT - paraméterei a NEXT-évei egyeznek meg, de ez a keresés a NEXT által beállitott aktuális sorral /az OPEN-ben megadott/ kapcsolatban lévő sort adja vissza.
- CLOSE - keresést zár le.
- CREATE } - relációt, listát, kapcsolatot, indexet
- DROP } /képet/ készit, szüntet meg.
- INSERT - sort relációba vagy listába illeszt.
- DELETE - sort relációból töröl,
- BUILDLIST - paraméterei: keresésazonositó, feltétel, rendezettség. A keresésazonositóval

megadott sorhalmaznak a feltételt kielégitő elemeiből rendezettség /csökkenő vagy növekvő/ sorrendü listát szervez.

Mint látni fogjuk, az RSI elég tipikus adatbázis assembler, leszámitva a listakészités lehetőségét. Erre azért lehet szükség, mert mig az interpreter tipusu Optimizálók maguk szervezik ideiglenes file-jaikat /rendezve, vagy hash-elve, vagy másképp/, az SQL-ben ez is az adatbázis assemblerre hárul.

Vizsgáljuk most meg az ASL-nyelvet! Nyilvánvalóan ennek átmenetet kell képeznie a nagyon magas szintü SQL és az alacsony szintü gépi kódu vezérlési szerkezetekből, /különböző tipusu ugrások/, értékadásokból és RSI rutinhivásokból álló "nyelv" között.

Az ASL program eljrásokból épül fel. Ezekből az első mindig egy SCANPROC, az ASL alapvető programszerkezete. Felépitése:

```
SCANPROC (paraméterek);
   {ideiglenes objektumok definiálása}
  SCAN sepcifikáció [WHERE feltétel] ;
  [IF korlátozás THEN]
  RETURN (kifejezések) ;
END;
```

Egy egyszerü példa: A

```
SELECT NÉV
FROM DOLGOZÓ
WHERE RÉSZLEGKÓD=50
```

lekérdezésnek a

```
SCANPROC A;
   SCAN DOLGOZÓ WHERE RÉSZLEGKÓD=50;
   RETURN NÉV ;
END;
```

ASL program felel meg. Ugyanez a Részlegkód szerinti "Részlegindex" nevü indexet használva:

```
SCANPROC A;
   SCAN DOLGOZÓ USING IMAGE RÉSZLEGINDEX AT 50;
   RETURN NÉV ;
END;
```

Az első változat forditása egy PL/1-szerü nyelvre:

```
PROC A;
  DCL STATUS STATIC INIT ('ZÁRVA') ;
  IF STATUS='ZÁRVA'
  THEN DO;
  azonositó=OPEN(DOLGOZÓ);
     STATUS='NYITVA';
  END;
  sor=NEXT(azonositó,<RÉSZLEGKÓD=50>);
  IF sor=NULL
  THEN DO;
     CLOSE(azonositó);
     STÁTUS='NYITVA';
     RETURN(NULL);
  END;
  RETURN(sor.NÉV);
END;
```

A SCANPROC által definiálható ideiglenes objektumok relációk, listák, indexek, és konstansok lehetnek. Ezeket az ASL BUILDPROC /és BUILDVALUEPROC/ eljárásai hozzák létre. Lássunk erre is egy példát! A

```
SELECT NÉV,RÉSZLEGKÓD
FROM DOLGOZÓ
ORDER BY RÉSZLEGKÓD
```

lekérdezésnek megfelelő ASL program pl.:

```
SCANPROC A;
   LET LIST L(N,R)=B;
   SCAN L;
   RETURN(N,R);
END;
BUILDPROC B;
   SCAN DOLGOZÓ;
   INSERT INTO LIST:NÉV,RÉSZLEGKÓD SORTED BY RÉSZLEGKÓD;
END;
```

Ennek a "forditása" két eljárást tartalmaz majd, a másodikban természetesen BUILDLIST hivással.

Egy példa illesztésre:

```
SELECT NÉV,RÉSZLEGKÓD
FROM DOLGOZÓ,RÉSZLEG
WHERE DOLGOZÓ.RÉSZLEGKÓD=RÉSZLEG.RÉSZLEGKÓD AND
      RÉSZLEG.CIM='BUDAPEST'
```

Legyen a Részlegkód szerint index ugy a Dolgozó, mint a Részleg relációban! Ekkor algoritmusunk a következő lesz: kiválasztjuk az összes budapesti részleget, majd a Részlegkód alapján az adott részlegben alkalmazott minden egyes dolgozó nevét a Dolgozó relációból. Ehhez uj ASL konstrukciót kell bevezetni. A

```
FOREACH TUPLE
     LET azonositó=scanproc-név (paraméterek)
RETURN
```

ciklusban hivja az adott nevü eljárást mindaddig, mig az nem ér a keresés végére, majd visszatér a FOREACH előtti SCAN-be:

```
SCANPROC A;
   SCAN RÉSZLEG USING IMAGE RÉSZLEGINDEX AT 'BUDAPEST';
   FOREACH TUPLE
       LET DNÉV=B(RÉSZLEGKÓD);
   RETURN DNÉV,RÉSZLEGKÓD ;
END;
```

```
SCANPROC B(RKÓD);
   SCAN DOLGOZÓ USING IMAGE DOLGOZÓINDEX AT RKÓD;
   RETURN(NÉV);
END;
```

Ennek az ASL programnak a forditása:

```
PROC A;
  DCL STATUS STATIC INIT('ZÁRVA');
  IF STATUS='ZÁRVA'
  THEN DO;
       azonositó=OPEN(RÉSZLEGINDEX,'BUDAPEST');
       STATUS='NYITVA';
    X:sor=NEXT(azonositó);
       IF sor=NULL
       THEN DO;
         CLOSE(azonositó);
         STATUS='ZÁRVA';
       END;
  END;
  sor1=B(sor.RÉSZLEGKÓD);
  IF SOR1=NULL THEN GO TO X;
  RETURN(sor1.NÉV,sor,RÉSZLEGKÓD);
END;
PROC B(RKÓD);
  DCL STATUS STATIC INIT('ZÁRVA');
  IF STATUS='ZÁRVA'
  THEN DO;
     azonositó=OPEN(DOLGOZÓINDEX,RKÓD);
     STATUS='NYITVA';
  END;
  sor=NEXT(azonositó);
  IF sor=NULL
  THEN DO;
       CLOSE(azonositó);
       STATUS='ZÁRVA';
  END;
  RETURN sor ;
END;
```

Ugy az előző forditás eljárása, mint az ebben a példában látott két eljárás szerkezete igen hasonló. A SCAN utasitás forditása az

```
IF STATUS='ZÁRVA'
   ...
   ...
   ...
END;
sor=NEXT;
IF SOR=NULL
   ...
   ...
END;
```

szerkezet. A FOREACH forditása a példa A eljárásában jól követhető.

A GROUP BY és a halmazmüveletek ASL forditására nem térünk ki. Szándékunk pusztán a nyelv stilusának érzékeltetése, és mind a két forditás /SQL-ről ASL-re és ASL-ről gépi kódra/ elvégezhetőségének illusztrálása volt. [ASTR76, CHAM 81, LORI79]

2.3.2. INGRES

Az INGRES - az SQL/DS-től eltérően - minden relációt külön file-ként tárol. Egy adatbázison belül négy file-tipus létezik:

Az <u>adminisztrációs file</u> az adatbázis adminisztrátor azonositóját és inicializáló információt tartalmaz. Ez az egyetlen file-tipus, mely nem relációt tárol.

A <u>katalógus /rendszer/ relációk file-jai</u> a rendszer adatainak leirását tartalmazza. Tulajdonosuk az adatbázis adminisztrátor, és tartalmukat bármelyik INGRES felhasználó lekérdezheti egyszerü RETRIEVE-vel, módositani azonban csak az adatdefiniciós parancsok képesek. Az egyes

relációk - RELATION, ATTRIBUTE, INDEX, PROTECTION, INTEGRITY - neveiből tartalmuk elképzelhető.

Az adatbázis adminisztrátor relációi a felhasználók számára lekérdezhetők, de nem módosithatóak - hacsak egyes felhasználóknak az adatbázis adminisztrátor nem engedélyezi.

A felhasználói reláció tulajdonosa a létrehozó felhasználó, és más felhasználó csak az ő engedélyével férhet hozzá.

A file-ok 512 byte hosszu rekordokból állnak. Ez operációs rendszer kötöttség, minden UNIX file fizikailag nem feltétlenül egymás után következő 512 byte-os rekordokból áll. Többek között a rövid lapmérettel indokolja [STON 76] a relációk egymástól elkülönitett tárolását.

Az adatbáziskezelő és az operációs rendszerek kapcsolatával általánosságban foglalkozó [STON 81] cikkből két idevágó gondolatot emelünk ki:

- a virtuális memóriát kezelő operációs rendszerek /a UNIX ilyen/ saját lapozási algoritmust használnak. Ez rendszerint LRU, ami az esetek jelentős részében elég jó, az INGRES esetén azonban sokszor van szükség:
  1. ujra nem használandó blokkok soros olvasására;
  2. blokkok ciklikus olvasása /1,2,...,n,1,2,...,n,.../;
  3. ujra nem használandó blokkok közvetlen elérésére.

  Ilyen esetekben az LRU rossz, ezért az adatbáziskezelő rendszerek általában belső lapozási mechanizmust használnak az operációs rendszer lapozása felett - ami viszont többletköltség
- a file rekordjai logikai sorrendjének és folytonosságának eltérése a fizikai szerkezettől hatékonyságcsökkenéshez vezet.

Végső soron, az értékelő [STON 80] cikk szerint az INGRES

alkotói a tévedések közé sorolják, hogy a UNIX file-kezeléséhez alkalmazkodtak, és nem alakitottak ki saját file-kezelést.

A lapszervezés az SQL/DS-ével egyező, annyi eltéréssel, hogy az INGRES file-okon kétfajta laptipus létezik: az elsődleges lapokat a file létrehozatalakor foglalja le a rendszer. Ehhez pointerrel lehet hozzáláncolva egy vagy több másodlagos lap. Ezek egy-egy elsődleges /vagy másodlagos/ lap tulcsordulásakor keletkeznek.

A tulcsordulási lapok létezése az INGRES file-szervezési módjaival függ össze. Egy reláció hash-elt vagy index-szekvenciális file, illetve ezek süritett változata - itt az adatokat a file-ra helyezés előtt összenyomják - lehet. Alapvetően kisebb relációk, ill. ideiglenes relációk részére létezik a rendezetlen tárolási mód. Mivel az INGRES az indexet is relációként kezeli, ennek szervezésére ugyanaz vonatkozik, mint bármely más relációéra /ld. még 1.1.2/.

Az INGRES az adatbázis assemblert AMI-nak /Access Method Interface/ nevezi. Kilenc rutinból áll, ezeket ismertetjük röviden:

OPEN /azonositó,mód,relációnév/. Az "azonositó" a program számára azonositja a "mód"-tól függően olvasásra vagy módositásra megnyitott "relációnév" relációt.

GET /azonositó, sorazonositó 1, sorazonositó 2, sor, mód/. A "mód"-tól függően vagy sorban "sorazonositól"-től "sorazonositó2"-ig adja vissza a "sor"-ban a sorokat, /szekvenciális elérés/, vagy csak a "sorazonositól" azonositóju sort /közvetlen elérés/.

FIND /azonositó, kulcs, sorazonositó, kulcstipus/. A "kulcs" alapján a "sorazonositó" értéket adja vissza. A "kulcstipustól" függően fogad el a "kulcs" értékü sor hiányában ahhoz legközelebbi alsó vagy felső kulcsértéket.

PARAMD /azonositó, leirás/. Ez a rutin relációra, a RARAMI /azonositó, leirás/ rutin pedig indexre adja vissza a kereséshez használható kulcs /ld. FIND/ jellemzőit.

INSERT /azonositó, sor/. Beillesztés.

REPLACE /azonositó, sorazonositó, uj sor/. Módosit.

DELETE /azonositó, sorazonositó/. Töröl.

CLOSER /azonositó/. Reláció lezárása.

[STON 76] megjegyzi, hogy a fizikai tárolás és az adatbázis assembler fenti szervezése jó lehetőséget ad a rendszer uj file-szervezési módokkal történő bővitésére. Valóban, az interface-en pusztán a FIND, PARAMI és PARAMD rutinokban lehet szükség változtatásra, feltéve, hogy az uj elérési mód alkalmazkodik az egységes lapszervezéshez és a sorazonositási mechanizmushoz.

2.3.3. Mikrogépes rendszerek

A mikrogépes relációs adatbáziskezelő rendszereket összehasonlitó cikkében F. Maryanski megállapitja: "Ha valaki egy standard adatbázis tankönyvet kinyit a fileszerkezeteket leiró fejezetnél, tiz-tizenöt különböző szervezési módot talál. Minden szervezési módhoz létezik legalább egy azon alapuló mikrogépes rendszer". [MARY 83]

Ugy tünik, a legelterjedtebbek a B-fák. Megitélésünk szerint fő előnyük abban rejlik, hogy egyszerre biztositanak gyors közvetlen hozzáférést kulcs alapján, és emellett a nagyság szerint soros elérést is támogatják. Ilyen "kétarcu" elterjedt szervezési módként csak az index-szekvenciális file-t emlithetnénk /figyelembe véve, hogy a dinamikus, lineáris, stb. hash még elég uj/, azonban ennek nehézkes és rossz esetben lassu tulcsordulás kezelése nem versenyezhet a B-fa elegáns, a tulcsordulást elkerülő dinamikus önátszervező technikájával.

/Azért megjegyezzük, hogy az INGRES alkotóinak [HELD 78] cikke néhány valós érvet tartalmaz az index-szekvenciális file mellett:

- a B-fában tárolt rekordokra mutató másodlagos indexek csak szimbolikus indexeket tartalmazhatnak, hisz egy rekord fizikai helye állandóan változhat;
- a konkurrens hozzáférés szervezése bonyolultabb a B-fára, ugyanis egy uj rekord beillesztése a teljes indexstrukturát megbolygathatja.

Végső soron, az ellenvetések ellenére, [STON 80] önkritikusan elismeri, hogy a B-fa előnyös tulajdonságai - nem kell időnként ujraszervezni, mint az index-szekvenciális file-t; egy adat eléréséhez beolvasandó lapok száma jól becsülhető - ellensulyozza a hátrányait./

Az általános B-fa megoldáson belül persze sok különböző változat lehetséges. A TITAN pl. nem is a kulcsot, hanem annak hash-függvényét tartja a B-fa index részében /a prefix B-fák alapgondolatához hasonlit az ötlet/, ezzel hely-megtakaritást érve el [FALQ 82]. Ugyancsak B-fát használ az /RDBAS [HERM 83] és a LIDAS [REBS 83] rendszer is.

Elég sok rendszer szervez egyszerüen szekvenciális file-okat, melyeket alkalmanként rendez / RQL[MAST 83], MRDBS [REVE 83], stb/. A forgalmazott rendszerek ezt nem engedhetik meg magunknak. A dBASE II pl. index file szervezését teszi lehetővé egy paranccsal. Az index kulcs szerinti lekérdezéseknél biztosit gyors /a leirás szerint 2 másodpercen belüli/ válaszidőt - érdekes viszont, hogy illesztésnél a rendszer nem használ indexet [ASHT 81].

Nagyon egyszerü - bár mikrogépesnek nem nevezhető - megoldást alkalmaz a VIDEBAS rendszer. Ez az összes

relációt index-szekvenciális file-on tárolja, méghozzá többször is, különböző kulcsok szerint rendezve, és szervezve az indexet. Mindehhez persze nagylemezeket használ. /A rendszer értékelésénél persze figyelembe kell venni, hogy készitői osztott adatbáziskezelő irányába kivánják fejleszteni./ Jellegzetessége, hogy a módositások nem kerülnek be azonnal a rendszerbe, hanem a "különbség" file-on tárolódnak, és egyszerre, a periódikus ujraszervezést végző programok viszik be őket. A megoldásnak sok előnye van: visszaállithatóság, az index-szekvenciális file-ok nem csordulnak tul, stb. [BLAN 83].

A PRECI/P rendszer hash-elt file-okban tárolja a relációit. Olyan hash technikát alkalmaznak mely megőrzi a rendezettséget. [DEEN 83]

Nem népszerü az adatösszenyomás a relációs rendszerek körében/[KIM 79] csak az INGRES-t és a PRTV-t emliti ellenpéldaként/. Ezért különösen érdekes a [KAMB 83] cikkben leirt, 1.1.4.-ben ismertetett rendszer. Ez is egyszerü szekvenciális file-lal dolgozik, de összenyomja az adatokat. Készitői szerint ezért nincsenek hatékonysági problémáik.

Még egy megjegyzés a mikrogépes rendszerekről: ezek éppugy, mint a nagygépes relációs adatbáziskezelők a rendszer katalógusát közönséges relációkként tárolják és kezelik. A megoldásnak az előnyei nyilvánvalóak.

I R O D A L O M

AHO 79 AHO A.V. SAGIV Y. - ULLMAN J.D. "Efficient optimization of a class relational transactions", ACM TODS 4,4/1979/, pp. 435-454.

ALAG 81 ALAGIC S. - KULENOVIC A: "Relational Pascal data base interface", The Computer Journal 24, /1981/ pp. 112-117.

ALLM 82 ALLMAN E. - STONEBRAKER M. "Observations on the evolution of a software system", Computer, June 1982, pp. 27-32.

APSI 83 APSINGIKAR V. - PHULE S. "Development of a portable relational DBMS", in [WORK 83] pp. 221-244.

ASHT 81 ASHTON-TATE Corp. "dBASE II Assembly language relational database management system",/1981/

ASTR 75 ASTRAHAN M.M. - CHAMBERLIN D.D. "Implementation of a Structured English Query Language", CACM 18, 10/Oct.1975/ pp. 580-588.

ASTR 76 ASTRAHAN M.M. et.al. "System R: relational approach to database management", ACM TODS 1,2 /June 1976/ pp. 97-137.

BLAN 83 BLANKEN H.M. "Another Approach to DBMS implementation", in [WORK 83] pp. 501-524.

BLAS 77 BLASGEN M.W. - ESWARAN K.P. "Storage and access in relational data base", IBM Syst. J. 4/1977/ pp. 363-377.

BLAS 81 BLASGEN M.W. "System R: an architectural overview" IBM Syst. J. 20,1/1981/ pp. 41-62.

BRAG 83 BRAGGER R.P. et. al. "GAMBIT: an interactive database design tool for data structures, integrity constraints and transactions", in [ZEHN 83]pp. 65-96.

CHAM 76 CHAMBERLIN D.D. et. al. "SEQUEL2: unified approach to data definition, manipulation and control", IBM J. of Res and Dev. 20,4/1976/ pp. 560-575 + 21,1/1977/ pp. 94-95.

CHAM 81 CHAMBERLIN D.D. et. al. "A history and evaluation of System-R", CACM 24,11/Oct.1981/ pp. 632-646.

CHAM 81 CHAMBERLIN D.D. et.al. "Support for repetitive transaction and ad hoc queries in System R", ACM TODS 6,1/March 1981/ pp. 70-94.

CHEN 76 CHEN P.P. "The entity-relationship model. Toward a unified view of data", ACM TODS 1,1/March 1976/ pp. 9-36.

CODD 70 CODD E.F. "A relational model for large shared data banks", CACM 13,6/June 1970/ pp. 909-917.

CODD 71a CODD E.F. "A data base sublanguage founded on the relational calculus", in ACM SIGFIDET Workshop on Data Description, Access and Control, /1971/ pp. 35-68.

CODD 71b CODD E.F. "Further normalization of the data base relational model", in Data Base Systems, Courant Computer Science Symposie, Vol. 6, Prentice-Hall, Englewood Cliffs, N.J., May 1971

CODD 71c CODD E.F. "Relational completeness of data base sublanguages" in Data Base Systems, Courant Computer Sicence Symposie, Vol. 6, Prentice-Hall, Englewood Cliffs, N.J., May 1971.

CODD 79 CODD E.F. "Extending the database relational model to capture more meaning", ACM TODS 4,4 /December 1979/ pp. 397-434.

CODD 82 CODD E.F. "Relational database: a practical foundation for productivity", CACM 25,2 /February 1982/ pp. 109-117.

DATE 77 DATE C.J. "An introduction to database systems" second edition, Addison-Wesley 1977

DEEN 83 DEEN S.M. "Some design aspects of Preci/P personal database system", in [WORK 83] pp. 441-456.

DIEC 81 DIECKMANN E.M. "Three relational DBMS", Datamation September 1981, pp. 137-146.

ДРИБ 82 ДРИБАС В.П. "Реляционные базы данных", Минск, изд. БГУ, 1982

DEME 81 DEMETROVICS J. - GYEPESI Gy. "Általános függőségek és lekérdezéssel kapcsolatos algoritmusok relációs adatmodellekben", SZTAKI Tanulmányok 118/1981

EBER 83 EBERLEIN W. - STEINBAUER D. "A menu-driven relational database system for a micro-computer", in [WORK 83] pp. 187-206.

EPST 77 EPSTEIN R. "A tutorial on INGRES", Memorandum No. UCB/ERL M 77/25 15 December 1977 University of California, Berkeley

FALQ 82 FALQUET G. et.al. "A portable relational data base management system for microcomputer", Microprocessing and Microprogramming 9 /1982/ pp. 17-25.

HALL 76 HALL P.A.V. "Optimization of a single expression in a relational data base system", IBM J. of Res. and Dev. 20,3/1976/ pp. 244-257.

HELD 78 HELD G. - STONEBRAKER M. "B-trees re-examined", CACM 21,2/1978/ pp. 139-143.

HERM 83 HERMIDA R. - RUZ J.J. "Assisted access to databases: the RDBAS approach", in [WORK 83] pp. 419-440.

KAMB 83 KAMBAYASHI Y. et.al. "A micro-computer-based relational database system with database preparation facilities", in [WORK 83], pp. 457-476.

KENT 83 KENT W. "A simple guide to five normal forms in relational database theory", CACM 26,2 /February 1983/ pp. 120-125.

KIM 79 KIM W. "Relational database systems" Computing Surveys 11,3/Sept. 1979/ pp. 185-211.

KISS 83 KISS O. "MADAM koncepció", SZTAKI Working Papers II/63, 1983

KNUT 83 KNUTH E. - RADÓ P. - TÓTH Á. "Az SDLA előzetes ismertetése", SZTAKI Tanulmányok 104/1980

LACR 83 LACROIX M. - PIROTTE P. "Relational model and Relational systems" in [WORK 83]

LIPS 79 LIPSKI JR,W. "On semantic issues connected with incomplete information databases", ACM TODS 4,3/Sept, 1979/ pp. 262-296.

LORI 79 LORIE R.A. - NILSON J.F. "An access specification language for a relational data base system", IBM J. Res. and Dev. 23,3/May 1979/ pp. 286-298.

MARY 83 MARYANSKI F. "Design, implementation, and use of relational DBMS on micro-computers" in [WORK 83] pp. 29-54.

MAST 83 MASTERS S. - DRISCOLL J.R. "RQL. A relational data base system for a low end micro-computer configuration", in [WORK 83] pp. 103-128.

MERR 83 MERRETT T.H. - CHIU G.K.W. "MRDSA: full support of the relational algebra on an APPLE II.", in [WORK 83] pp. 385-402.

PATN 83 PATNIK I.M. et.al.: "Access path query language for relational database systems", Software-Practice and Experience 13,/1983/ pp. 661-670.

REBS 82 REBSAMEN J. - REIMER M. - URSPRUNG P. - ZEHNDER C.A. "LIDAS - a database system for personal computer LILITH" Institut für Informatik der ETH Zürich 1982

REBS 83 REBSAMEN J. et.al. "LIDAS - the database system for the personal computer Lilith", in [WORK 83] pp. 291-316.

REIM 83 REIMER M. "Implementation of the database programming language Modula/R on the personal computer Lilith", in [ZEHN 83] pp. 49-64.

REVE 83 REVELL N. - SIMS R.J.: "MRDBS: a relational development system for a micro-computer", in [WORK 83] pp. 245-260.

SAND 81 SANDBERG G. "A primer on relational data base concepts", IMB Syst. J. 20,1/1981/ pp. 23-40.

SCHM 77 SCHMIDT J.W. "Some high level language constructs for data of type relation", ACM TODS 2,3/Sept.1977/ pp. 247-261.

SMIT 75 SMITH J.M. - CHANG P.Y. "Optimizing the performance of a relational algebra database interface", CACM 18, 10/1975/ pp. 568-579.

SNYD 81 SNYDERS J. "New trends in DBMS", Computer Decisions February 1982, pp. 100-133.

STON 76 STONEBRAKER M. - WONG E. - KREPS P. "The design and implementation of INGRES", ACM TODS 1,1 pp. 189-221.

STON 80 STONEBRAKER M. "Retrospection on a database system", ACM TODS 5,2/June 1980/ pp. 225-240.

STON 81 STONEBRAKER M- "Operating system support for database management", CACM 24,7/1981/ pp. 412-418.

TODD 76 TODD S.J. "The Peterlee Relational Test Vehicle-a system overview", IBM Syst. J. 15,4/1976/ pp. 285-308.

URSP 83 URSPRUNG P. - ZEHNDER C.A. "HIQUEL: an interactive query language to define and use hierarchies", in [ZEHN 83] pp. 97-118.

WELT 81 WELTY C. - STEMPLE D.W. "Human factors comparison of a Procedural and a Nonprocedural Query Language", ACM TODS 6,4/December 1981/ pp. 626-649.

WONG 76 WONG E. - YOUSSEFI K. "Decomposition - an algorithm for query processing", ACM TODS 1,3/1976/ pp. 223-241.

WORK 83 WORKSHOP on Relational DBMS
Toulouse /France/ February 14-15, 1983
/INRIA edition/

ZEHN 83 ZENHDER C.A. /editor/ "Database techniques for professional workstations", Institut für Informatik der ETH, Zürich 1983

ZLOO 77 ZLOOF M.M. "Query-by-Example: a data base language", IBM Syst.J. 16,4/1977/ pp. 324-343