Adatbáziskezelő rendszerek

Bevezetés

Az információ feldolgozás története korábban kezdődött, mint az elektronikus adatfeldolgozás. A múlt század végén az amerikai népszámlálás részére dolgozta ki Hollerith lyukkártyás rendszerét. Az ekkor megállapított kártya szabvány még az 1980-as években is használatban volt, az 1970-es években pedig még mechanikus Hollerith gépekkel (rendező, stb.) is lehetett találkozni Magyarországon. Ugyanakkor az első generációs számítógépekre nem csak az elektroncsövek, hanem a kis tárolókapacitás mellett az is jellemző volt, hogy azokat csak erre specializálódott szakemberek tudták használni. Nem csoda, hogy ezeket a gépeket elsősorban kalkulátorként alkalmazták. (Erre a célra is fejlesztették őket ki.) A második generációs gépek a fordítóprogramok révén már szélesebb felhasználói kör számára voltak elérhetőek, s már valamivel nagyobb kapacitású háttértárolóik is lehetővé tették az első információ-feldolgozó rendszerek megjelenését. Mindez fokozottan igaz a harmadik számítógépes generációra (operációs rendszerek megjelenése). Nem véletlen, hogy azt a mesterséget, melyet eleinte számítástechnikának neveztek, ma informatikának hívják: napjainkban a számítógépek fő felhasználási területe az információ-feldolgozás. Mára az információ feldolgozása igazi iparrá vált, az információban szegény országok más iparágakban sem lehetnek sikeresek.

Az információ feldolgozás alapvető módszerei

Napjainkig az információ feldolgozásnak három alapvető formája ismeretes: a folyamatszemléletű információ feldolgozás, az adatbázis-szemléletű információ feldolgozás, és a szakértői rendszerek.

A folyamatszemléletű információ feldolgozás

E rendszerek legfőbb jellemzője hogy azok célorientáltak, azaz készítésükkor először az elérendő *cél*t kell meghatároznunk, majd a cél ismeretében tervezzük meg az ahhoz vezető *folyamat*ot, s a folyamat által használandó *adatállomány*okat (file). Az állományok csak adatokat tartalmaznak: kapcsolatokat a folyamat írja le. A tervezés sorrendje tehát: cél folyamat állományok Az ilyen módon megtervezett adatállományok nem igényelnek túl nagy tárolókapacitást, hiszen csak a valóban felhasználandó adatokat kell tárolnunk. Másrészt az adatállományok feldolgozása igen gyors lehet, hiszen azok szerkezete optimálisan illeszkedhet az adott feladathoz. (Tisztázandó persze, hogy tekinthetünk optimális illeszkedésnek.) A leírt tulajdonságok indokolják, hogy miért használták az első információ feldolgozó rendszerek ezt a formát, s hogy miért használják még napjainkban is kisebb volumenű, alkalmi feladatok esetén. Ugyanakkor a folyamat szemléletű információ feldolgozás minden célhoz új folyamatot és állományokat kíván, függetlenül az esetleges közös adatok előfordulásától. Az így keletkező redundancia rossz tárolókihasználást eredményez - és ami még nagyobb baj - felidézi az inkonzisztencia veszélyét, vagyis azt, hogy egy-egy több helyen előforduló adat egyes példányainak különbözik a tartalma. Hátránya az effajta feldolgozásnak az is, hogy csak az eredeti kérdésre ad választ, a rendszernek ad hoc jellegű kérdések nem tehetők fel.

Optimális szerkezetű állományok

A programozással foglalkozók jól tudják, hogy arra a kérdésre: mi a jó program ismérve, nem lehet egyszerűen válaszolni. El kell döntenünk, milyen szempontból akarjuk a programokat összehasonlítani. Ha az információ feldolgozó rendszereket

vizsgáljuk, a helyzet ugyanilyen bonyolult. tárolóhely létrehozás Optimalizálási szempontok feldolgozási idő keresés változtatás felhasználói szempontok Könnyen belátható, hogy a felsorolt szempontok egyidejűleg nem elégíthetőek ki maradéktalanul, bármelyiket választjuk is ki az optimalizálás alapjául, az így adódó megoldás a többi szempont szerint messze lesz az optimálistól. Így pl. a tárkihasználás szempontjából ideális tömörített állományok feldolgozása nyilvánvaló többletidőt igényel, a könnyen kezelhető felhasználói felületek mind tár, mind idő tekintetében igen sokat követelnek. (Gondoljunk csak pl. a DOS-os és a WINDOWS-os programok hely- és időigénybeli különbségeire!) De ellent mondanak egymásnak az időre optimalizálás "al" szempontjai is: nyilvánvalóan pl. a keresést jelentősen meggyorsítja, ha egy állomány strukturált (mondjuk a keresési kulcs szerint rendezett), ugyanekkor a rendezettség megteremtése a létrehozáskor, megtartása új rekordok beszúrásakor komoly időigénnyel járhat. Az elmondottak ellenére általában kijelenthetjük, hogy a modern információs rendszerekben kiemelt szerepe van a keresésnek. Egyfelől a nagy hálózatok korában az időfaktor többnyire szorítóbb mint az elosztott információk helyigénye, másfelől az időre való optimalizáláson belül elmondhatjuk, hogy egy-egy változtatást (adatbeszúrás, törlés) megelőz az illető rekord keresése (van-e ilyen, s ha igeműn, hol). Ha mindehhez hozzávesszük, hogy egy-egy állományban (melyet egyszer hozunk létre) igen sokszor kell keresnünk, nem meglepő a keresés kiemelt szerepe. A keresés időigényét két tényező szabja meg, az egyik az átlagosan szüksége lépésszám (hány keresési lépést kell végrehajtanunk átlagosan egy-egy keresés alkalmával), a másik az egyes lépések időigénye. Ha feltesszük, hogy rekordjainkat azonos gyakorisággal keressük, egy-egy keresés időigényére alábbi képletet adhatjuk:$S_N (T_A + T_B + T_C)$
A képletben T_A az u.n. algoritmusidőt jelenti, azaz annak az algoritmusnak az átlagos végrehajtási idejét, mely kijelöli a legközelebb megvizsgálandó rekordot, T_B az u.n. behozási idő, azaz a tárolóról való beolvasás átlagos időigénye, végül T_C az összehasonlítási (komparálási idő, mely a beolvasott rekord kulcsának megvizsgálásához szükséges. Végül S_N az átlagosan szüksége lépésszám. Belátható, hogy központi tár esetén elesik a T_B tag, míg az általánosan használt mozgófejes lemezeknél T_A és T_C elhanyagolható T_B-hez képest. Másrészt míg T_C idejét nem tudjuk befolyásolni, T_A, T_B és S_N értéke a választott algoritmustól függ. Azt pedig, hogy milyen keresési algoritmusok között választhatunk az állomány szervezési módja, szerkezete határozza meg. Például ha egy adott nevű személy létező rekordját akarjuk egy rendezetlen állományból kikeresni, az u.u. lineáris keresést alkalmazhatjuk, azaz sorban meg kell vizsgálnunk a rekordokat, hogy azonosak-e az általunk keresettel. Ez esetben átlagosan a rekordok felét kell megvizsgálnunk (azaz S_N k N/2). Ha az illető rekordja nem található meg az állományban, ennek megállapításához már az egész állományt végig kell keresnünk (vagyis S'_N k N). Ha az állomány a névre rendezett, az utóbbi esetben is elég a rekordok felét megvizsgálni (S'_N k N/2). Így a struktúra megváltoztatása önmagában gyorsított az eljáráson. Más kérdés, hogy az ilyen módon strukturált állományokban már sokkal hatékonyabb keresési algoritmusok használatára is van remény.

A legfontosabb állomány struktúrák

1.) ***Szekvenciális állomány struktúrák*** ahol logikailag a rekordok mindegyikének egy megelőzője és egy rákövetkezője van (az első és az utolsó elem kivételével).
2.) ***Hierarchikus állomány struktúrák;*** ahol logikailag a rekordok mindegyikének egy megelőzője, de esetleg több rákövetkezője van (az első elem kivételével).
3.) **Hálós állomány struktúrák*;*** ahol logikailag a rekordok mindegyikének több megelőzője és több rákövetkezője lehet.

4.) **Asszociatív állomány struktúrák*;*** ahol nem a rákövetkezés ill. megelőzés adja a állomány struktúráját, más logikai szempont szerint lehet a rekordokat ill. rekordok egy csoportját kiválasztani.

Szekvenciális állomány struktúrák:

Fizikai szekvenciális állomány struktúrák Fizikai szekvenciális struktúráról akkor beszélhetünk, ha a *rekordok logikai és fizikai sorrendje megegyezik.* E zervezési forma legfontosabb előnye, hogy a természetesen itt is alkalmazható a lineáris keresés mellett annál jóval hatékonyabb keresési eljárások alkalmazását is lehetővé teszi.

A.) **Bináris, logaritmikus keresés**

E közismert keresési eljárás lényege, hogy a megvizsgálandó állomány ill. állományrész középső rekordjának kulcsát hasonlítjuk össze a keresett rekordéval, s így eldönthetjük, hogy

a.) megtaláltuk a keresett rekordot,

b.) a keresett rekord a vizsgált állomány bal felében van,

c.) a keresett rekord a vizsgált állomány jobb felében van. A keresést ezután már csak az előzőlépésben vizsgált állomány felében kell folytatnunk (ha egyáltalán). Belátható hogy bináris keresés esetén e az átlagosan szükséges lépésszámra mind sikeres, mind sikertelen esetben $S_N = \log_2 N$.

B.) **Peterson-féle keresés**

Az előző módszer továbbgondolásából adódik ez a módszer. A gondolatmenet lényege az, hogy az állományt az egyes lépésekben ne két hosszra egyenlő részre osszuk, hanem két olyan részre, melyekben a keresett rekord előfordulása egyenlően valószínű. (Pl. egy nevekre rendezett nyilvántartásban Ács Aba rekordját nyilván az állomány elején, Zöld Zoltánét a végén célszerű keresni.) Az ilyen felezéshez nyilván ismerni kell a kulcsok eloszlását. Feltehetjük pl., hogy azok eloszlása egyenletes. Ekkor $S_N = 1/2 \log_2 N$. Mindebből nem következik azonban, hogy a Peterson-féle keresés mindig hatékonyabb a binárisnál. Ennek egyik oka az, hogy kulcsaink eloszlása messze eshet az egyenletestől, s ilyenkor e módszer hatékonysága jelentősen romlik. A másik ok, hogy a Peterson módszer minden lépése arányosítást, azaz "igazi" osztást igényel. A bináris keresésnél erre nincs szükség, az itt használt felezés a rendkívül gyors léptetéssel megvalósítható. Mivel központi tárbeli használatnál, az egyes lépések időigényét az algoritmusidő határozza meg, hiába igényel a Peterson féle keresés kevesebb lépést, ha azok jóval lassabban hajthatók végre. Bármilyen gyorsak a fizikai szekvenciális állománystruktúránál használható keresési eljárások, elhamarkodott kijelentés lenne azt állítani, hogy az ilyen állományok feldolgozása $N < 4$ esetén gyorsabb, mint a struktúrával nem rendelkezőké. Az állomány kezelésekor ugyanis a keresésre fordított időn kívül figyelembe kell venni azt is, melyet a karbantartásra, azaz a rendezettség megtartására kell fordítanunk. Mindezt összevetve azt mondhatjuk, hogy fizikai szekvenciális struktúra használata elsősorban a statikus állományoknál javasolt, azaz ott ahol egy-egy struktúrát érintő változtatásra (pl. beszúrás) sok keresés esik.

Logikai szekvenciális állomány struktúrák Logikai szekvenciális struktúráról akkor beszélünk, ha a *rekordok logikai és fizikai sorrendje nem egyezik meg.* A logikai sorrendet ez esetben legtöbbször a rekordokban elhelyezett mutatók (pointerek adják) meg. (Meg lehetne adni a logikai sorrendet valami külső táblázat segítségével is, ezt azonban a szakirodalom általában az indexelt szervezés részeként tárgyalja: v.ö. pont.) A mutatórendszerek lehetnek egy és kétirányúak, ill. gyűrűsek. Az állomány, és a sokszor ugyancsak speciális

listaként kezelt üres helyek kezdetét egy u.n. indító tábla tartalmazhatja. E szervezési mód legfőbb előnye, hogy alkalmazásakor az állomány logikai struktúrájának megváltoztatása igen könnyű, nem igényli a rekordok mozgatását, csupán a mutatók módosítása szükséges. További jelentős előny, hogy fizikailag ugyanaz az állomány több mutatórendszer alkalmazásával többféle logikai szekvenciális rendbe is szervezhető. (E logikai szekvenciáknak nem kell minden esetben valamennyi rekordot tartalmazniok.) A felsorolt előnyök mellett a logikai szekvenciális szervezésnek komoly hátrányos tulajdonságai is vannak. Elsősorban azt kell megemlítenünk, hogy e szervezési forma mellet nem állnak rendelkezésünkre olyan hatékony keresési algoritmusok mint a bináris, vagy a Peterson féle. Nem kisebb probléma, hogy mozgófejes lemez használata esetén - ha a rekordok véletlenszerűen helyezkednek el az állományban - a keresés során szükséges pozícionálások száma a feldolgozást nagyon lelassítja. (Egy C cilindert elfoglaló állomány esetén két rekord elérése között átlagosan C/3 cilindernyi pozícionálásra van szükség.) A fenti hátrányok kiküszöbölésére, vagy legalább csökkentésére használhatóak az u.n. "***csaknem fizikai szekvenciális"*** struktúrák. E struktúrák lényege, hogy az állományt - mely logikai szekvenciális, így mutatókkal rendelkezik - létrehozásakor a mágneslemezen fizikailag is szekvenciálisan helyezzük el. Beszúráskor nem szükséges a fizikai szekvencialitást megtartani, az új rekordot a - miden cilinderen külön-külön fenntartott - u.n. cilinder túlcsordulási területre helyezzük, és a logikai rendbe a mutatók segítségével illesztjük be. Látható, hogy így elkerülhejük a sok pozícionálásból adódó időveszteséget. Ezen túlmenően azonban e szervezési formának az is előnye, hogy mellette alkalmazhatjuk az u.n. "kupacos" keresési módszert. A keresési módszer lényege, hogy a rekordokat (N db.) G kupacra osztjuk, majd először e kupacok első elemeit hasonlítjuk össze a keresett rekord-dal. Így vagy megtaláljuk a keresett rekordot, vagy legalább meghatározhatjuk, hogy melyik kupacban van. Ezután ezt a kupacot kell végigkeresnünk. Ha az egyes kupacok kezdőelemei az elsődleges (nem a túlcsordulási) területen vannak, a keresés elég gyors lehet. Természetesen kulcskérdés a kupacok méretének megválasztása. Bizonyítható, hogy ha az egyes kupacok méretét kN-re választjuk, a szükséges lépésszámra S_N =kN adódik. Az eddigiekben mindig feltételeztük, hogy az egyes rekordokat azonos gyakorisággal keressük vissza (ezt a soronkövetkező bekezdés kivételével a továbbiakban is feltesszük majd). A logikai szekvenciális szervezés azonban lehtőséget ad erősen különböző keresési gyakoriságú rekordok olyan szekvenciába szervezésére, melynél a sorrendet a keresési gyakoriság csökkenő sorrendje szabja meg. Ilyen szervezés fizikai szekvenciális struktúrában is lehetséges, a logikai szekvenciális struktúra esetén azonban alkalmazhatóak az u.n. önrendező algoritmusok is, melyek a gyakoriság ismerete nélkül, ill. változó gyakoriság mellett is lehetővé teszi a gyakoriság szerinti szervezést. (A legegyszerűbb ilyen algoritmus minden rekordot annak keresése után az első helyre csatol. Ha nem is érhető így el pontos gyakoriság szerinti rendezés, azt ez az egyszerű eljárás is biztosítja, hogy a gyakran használt rekordok mindig az állomány elején legyenek találhatók.)

A hierarchikus állomány struktúrák

A hierarchikus (faszerű) struktúrák számítógépes ábrázolására egyaránt elterjedtek mind a belső mutatós (pointeres), mind a külső mutatós (táblázatos) eljárások.

A.) **Belső mutatós eljárások**

a.) Visszavezetés szekvenciális struktúrákra. E módszereknél a fa egy megadott módon való bejárását rögzítjük. A legjellegzetesebb ilyen módszer az u.n. a left-list módszer. Sajnos az eljárás

információvesztéssel jár: az eredeti fastruktúra nem állítható vissza belőle egyértelműen.

b.) Több mutató használata. A rekordokban lehet annyi mutatót elhelyezni, ahány rákövetkezője (gyermeke) van az illető rekordnak. Az ábrázolás teljesen egyértelmű, hátrányos tulajdonsága viszont, hogy - mivel a rákövetkezők száma tág határok között változhat - vagy igen rossz tárolókihasználással, vagy változó hosszú rekordok használatával kell számolnunk.

c.) Külön kapcsolórekordok használata. Látszólag bonyolítja a helyzetet, ha az információt hordozó rekordokon kívül egy újabb rekordtípust vezetünk be, mely csak a kapcsolatok leírására szolgál. Ugyanakkor ezzel a megoldással elérhető, hogy egy-egy rekordban (a kapcsolórekordokat is beleértve) legfeljebb két mutató legyen, sőt az is, hogy ezek egyike "oldalra", másika "lefelé" mutasson, ami a struktúrát igen áttekinthetővé teszi.

d.) Gyűrűk alkalmazása. Szokásos megoldás a fában "függőleges" és "vízszintes" gyűrűket szervezni, melyekkel ugyancsak megoldható, hogy egy-egy rekord ne tartalmazzon kettőnél több pointert.

B.) **Külső mutatós eljárások**

a.) Táblázatok használata. E táblázatokban az egyes rekordokhoz tartozó bejegyzések adják meg a rákövetkezőket (esetleg a megelőzőt is). Mivel a rákövetkezők száma itt is erősen változhat, a táblázatok ábrázolásánál ugyanazokkal a problémákkal kell szembenéznünk, mint a több mutatós rendszer alkalmazásánál.

b.) Bináris mátrixok használata. Bináris mátrixokban ábrázolhatjuk, hogy egyes rekordok között van-e kapcsolat. A bináris jellegen kívül az teszi - az egyébként nagyméretű mátrixokkal operáló - módszert használhatóvá, hogy hierarchikus állományoknál elég a mátrix felét tárolni (a másik félmátrix csak 0 elemeket tartalmaz).

A hálós állomány struktúrák:

A.) Belső mutatós eljárások

a.) Visszavezetés a hierarchikus struktúrákra. Megkisérelhetjük a hálós struktúrákat hierarchikusra visszavezetni. Ez esetben úgy kerülhetjük el az egyes elemek többszörös tárolását, hogy azok helyett - egy kivételével - u.n. virtuális elemeket alkalmazunk (azaz olyan elemeket, melyek adatokat nem, csak azok helyére való utalást tartalmaznak.

b.) Több mutatós eljárások. Ezek az eljárások ugyanúgy használhatók, mint a hierarchikus esetben, az előnyök és a hátrányok is ugyanazok lesznek. Itt azonban nem alkalmazhatóak azok az eljárások (külön kapcsolórekordok, gyűrűk), melyek kihasználják a szintek létezését: a "vízszintes", a "függőleges" fogalmát.

B.) Külső mutatós eljárások

a.) Táblázatok használata. A táblázatok alkalmazását illetően semmi lényeges különbség sincs a hálós és a hierarchikus eset között.

b.) Bináris mátrixok használata. A bináris mátrixok is ugyanúgy használhatók, mint a hierarchikus állományoknál, itt azonban általában nem lesz elegendő a fél mátrix tárolása.

Asszociatív állomány struktúrák Indexelt állomány struktúrák

A.) A sűrű indexelés

E szervezési formánál az adatállomány minden tételéhez tartozik az index állományban egy index bejegyzés: egy kulcs cím pár. Ilyen módon több szempont alapján több index is szervezhető, az adatállományhoz új rekordok könnyen fűzhetők (a végére), az index bejegyzés ismeretében a keresés gyors. Ugyanakkor nagyméretű index állományokban kell keresni, és azokat karban is kell tartani. Kulcskérdés tehát az index állomány szerkezetének helyes megválasztása. Sok szempontból az u.n. bináris fák (olyan fák, melyek miden elemének legfeljebb két gyermeke van) használata tűnik előnyösnek. Ha egy bináris fa kiegyensúlyozott (azaz az utolsó ill. és az utolsó előtti szint kivételével minden elemnek valóban két leszármazottja van), a keresés igen gyors benne: közelítőleg $\log_2 N$ lépésre van szükség átlagosan. Ugyanakkor a kiegyensúlyozottság megtartása új elemek beszúrásánál igen időigényes. A gyakorlatban indexállományok készítésére elsősorban az u.n. B-fákat (Bayer fákat) használják. (Ilyen struktúrát alkalmaz a közismert Turbo Access, a dBASE, a Clipper, a FoxPro, stb.) A B-fa lényege, hogy az indexállomány lapokra van osztva, egy-egy lapon (a gyökérelemet kivéve) n és 2n közé eső számú bejegyzés, és a bejegyzéseknél eggyel több pointer van. (Ez az n szám jellemző a szervezésre.) Az ilyen állományok könnyen karbantarthatóak, és a bennük való keresés elég gyors: közelítőleg $\log_n N$ lépésre van szükség átlagosan.

B.) A ritka indexelés E szervezési formánál nem minden rekordhoz, csak bizonyos jelzőpontokhoz tartozik bejegyzés. Természetesen az eligazodáshoz ez esetben valamilyen rendezettség szükséges. Az indexek általában több szintűek, az alsó szintű indexekben (sok bejegyzés) való eligazodást újabb ritka index segítheti. Jól példázza a ritka indexelést a szokásos telefonkönyv. Ilyen szervezés mellett igen gyors a keresés, és az állományba aránylag könnyen lehet új rekordokat beszúrni.Ugyanakkor ilyen index rendszer csak egy szempont alapján szervezhető.

C.) Az index-szekvenciális szervezés A ritka indexelés egyik legnépszerűbb megvalósítása az index-szekvenciális szervezés. Kifejezetten mágneslemezre tervezték: így a legalsó (legritkább) index a sávindex. (A sávokat technikailag amúgy sem lehetséges másképpen, mint szekvenciálisan feldolgozni.) Fő index Az index szintek:Cilinder index Sáv index Minden egyes cilindert három részre osztanak, hogy az azon levő területeket fejmozgás nélkül lehessen elérni. Index terület A lemez felosztása: Elsődleges adatterület Túlcsordulási terület Az állomány létrehozásakor a rekordok az elsődleges adatterületre kerülnek, melyet csak részben töltenek fel. A feltöltés rendje fizikai szekvenciális. Ekkor rögzítik az indexeket. Ha az elsődleges adatterület betelik, a rekordok a túlcsordulási területre kerülnek, mutatókkal ellátva, azaz logikai szekvenciális rendben. Az adatok keresése az index rendszeren keresztül történik. Természetesen az index-szekvenciális állományokat időről időre újra kell szerveznünk, azaz valamennyi rekordot az elsődleges területre kell tölteni. Ellenkező esetben a feldolgozás nagyon lelassulhat.

A direkt állomány szervezés A direkt szervezésnél a rekordok kulcsai és címei között egy leképezés hozza létre a kapcsolatot. Természetesen kívánatos lenne, hogy ez a leképezés egy-egy értelmű legyen, vagyis, hogy a leképezés ne rendeljen azonos címeket különböző kulcsokhoz (látjuk majd, hogy ez a követelmény nem mindig teljesíthető).

A.) A leképezések A fenti követelményeknek eleget tévő leképezést **közvetlen** leképezésnek nevezzük. Erre példa egy olyan elképzelt állomány, mely a teljes magyar lakosság adatait tartalmazza, s melynél a kulcsként használt személyi szám egyben a rekord állományon belüli sorszáma. (Bináris számítógépben ábrázolt

kulcsokat minden további nélkül tekinthetünk numerikus értékeknek, s az sem okozhat problémát, hogy címként az állományon belüli sorszámot tekintjük.) A közvetlen leképezések sajnos többnyire katasztrofálisan rosszul gazdálkodnak a tárterülettel, s így gyakorlatilag használhatatlanok. (Megjegyezzük, hogy tekinthetjük ilyen szervezésnek a sűrűn indexelt állományokat is, ezekre nem vonatkozik a rossz tárkihasználásról szóló megjegyzés.) A jobb tárkihasználás érdekében követelményeinkből engedni kényszerülünk. Az u.n. **hashing** algoritmusok olyan leképezések, melyek - lehetőleg nem túl sokszor - megengedik, hogy különböző kulcsokhoz ugyanaz a cím tartozzék (ezek az u.n. **szinonimok**). Cserébe jobb tárki-használásra számíthatunk. A tapasztalat azt mutatja, hogy 80-85%-os tárkihasználás mellett 20%-nél nem több szinonim elfogadható érték. A leggyakoribb hashing algoritmusok: a.) a csonkítás melynél lhagyjuk a kulcs olyan jegyeit, melyek nagy valószínűséggel csak kevés értéket vesznek fel (pl. a már említett személyi számnál a hónapok tizes jegyei, stb.), és a maradék számot tekintjük az állományon belüli sorszámnak, b.) a maradék módszer melynél a kulcsot egy m modulussal osztjuk, s a sorszám az osztás maradéka lesz. (Célszerű modulusként prímszámot választani.) E módszer előnyös tulajdonsága, hogy az eredetileg egyenletes elosztást nem rontja el.

B.) **A szinonimok kezelése** Hashing algoritmusok esetén külön kell gondoskodnunk a szinonimok kezeléséről. a.) A külső láncolás E módszernél egy külön területen tároljuk a szinonimokat, melyeket mutatólánc köt össze az eredeti (hashing algoritmus által adott) címmel. Mivel előre nem lehet a szinonimok számát tudni, a külön szinonimterület fenntartása akadálya a jó tárkihasználásnak. b.) A belső láncolás Itt is mutatórendszer köti össze a szinonimot eredeti címével, azonban azt az elsődleges terület egy üres (és lehetőleg az eredeti címhez közel eső) helyére tesszük. A tárterület kihasználása jó lesz, azonban lehetséges, hogy a szinonim által elfoglalt helyre később igényt tart annak "jogos" tulajdonosa, mely rekordból így "műszinonimot" csinálunk. Ennek kezelése az állomány feldolgozásakor több lépést igényel. c.) A Peterson módszer (nyílt módszer) A rekordok ugyanott helyezkednek el, mint a belső láncolásnál, azonban nem használunk mutatókat. Az állomány struktúrája egyszerű, feldolgozása azonban több lépést igényel, mint ha a belső láncolást alkalmazzuk. d.) Többszörös hashing Ha a kiszemelt hashing algoritmus szinonimra vezet, alkalmazhatunk újabb és újabb ilyen algoritmusokat, mígcsak a probléma el nem hárul. (A gyakorlatban két hashing algoritmust használnak, az esetlegesen még maradó szinonimokat az a.)-c.) módszerek valamelyikét alkalmazzák.) e.) Bucket-ek (bugyrok) alkalmazása Lehetséges az egyes címekhez nem egy, hanem több rekordnyi helyet rendelni. Így kevesebb lesz a szinonimok kezelésére fordítandó idő, ugyanakkor minden keresésnél az érintett bucket-et (pontosan átlagosan annak a felét) meg kell vizsgálni. A módszer használata akkor indokolt, ha a tároló fizikai tulajdonságai (és a rekordok rövidsége) amúgy is csak több rekord egyidejű ki- ill. bevitelét teszik lehetővé.

C.) **A szinonimkezelő módszerek összehasonlítása** E témakörnek komoly, és nagy matematikai apparátust megmozgató irodalma van, itt csak igen durva heurisztikus okoskodásra van módunk. Központi tárban való használat esetén, előnytelen a többszörös hashing

számára a jelentős algoritmusidő (egy hashing algoritmus lefuttatása jóval hosszabb, mint egy cím átírása, vagy egy regiszter növelése, amit a láncolás ill. a nyílt módszer igényel). A nyílt módszer több lépést igényel mint a láncolások, s algoritmusideje azokéval összemérhető. Jóllehet a belső láncolás hajszállal több lépést igényelhet a külsőnél (a "műszinonimok" miatt), nagyobb súllyal esik latba a jó tárkihasználás. Így általában a belső láncolást tudjuk javasolni. Mozgófejes lemeznél előnytelenek a sok fejmozgást igénylő módszerek: ilyen a többszörös hashing és a külső láncolás. Bár a belső láncolás elméletileg kevesebb lépést igényel a nyílt módszernél, ha a szinonim rekord ugyanazon a sávon van, mint az eredeti helye, a gyakorlatban ez az előny nem érvényesül. Ugyanakkor, ha a szinonim az eredeti helyet közvetlenül követi (és ez igen gyakori eset), a nyílt módszer gyorsabb is lehet (nem kell a mutatókat feldolgozni). Végeredményben tehát ilyen esetekben többnyire a nyílt módszer javasolható.

Az adatbázis szemléletű információfeldolgozás

Egyfelől a tárolókapacitások robbanásszerű növekedése, másfelől a felhasználói körnek a rendszertámogatások által lehetővé tett bővülése indokolta az új információ-feldolgozási forma megjelenését az 1960-as években. E szemlélet lényege, hogy a rendelkezésre álló adatokból indulunk ki, az összes adatot és a közöttük fennálló összes kapcsolatot egy integrált adatbázisba gyűjtjük, és valamennyi felhasználó valamennyi kérdéséhez ezt az adatbázist (vagy ennek egy részét) használhatja. A felhasználók nagy részének nincs szüksége a teljes adatbázisra, sokszor egy-egy felhasználónak nincs is joga az adatbázisban mindenhez hozzáférni. Egy-egy felhasználói nézet (view) tehát csak az adatbázis egy részét láthatja. Ezen belül is szabályozandó, hogy a látott adatokkal mit tehet (más egy adatot megnézni, vagy pl. megváltoztatni). Szükséges tehát, hogy legyen valaki, aki a felhasználókénál nagyobb hatáskörrel rendelkezve, ezeket a jogokat kiossza és a munkát felügyelje. Ez a személy, vagy csoport az **adatbázis felügyelő**. Az ő joga és kötelessége az adatbázist megtervezni, elkészíteni a teljes logikai struktúrát leíró **sémát**, valamint az egyes nézetek által látott adatbázis-rész logikai struktúráit leíró **alsémákat**. Ugyancsak az adatbázis felügyelő feladata az adatbázis működtetése során a felhasználókkal való kapcsolattartás, az adatbázis logikai és fizikai struktúrájának a felhasználói igényeknek megfelelő módosítása, valamint az adatbázis tartalmának rendszeres mentése. A séma és alséma leírások az u.n. **DDL** (Data Definition Language: Adatleíró Nyelv) nyelven készülnek ezt a nyelvet többnyire csak az adatbázis felügyelet használhatja. A felhasználók a lekérdezéseket, adatváltoztatásokat a **DML** (Data manipulation Language: Adatmanipuláló Nyelv) nyelven eszközölhetik. E nyelveknek két nagy csoportja van. Egy **Host Language** (Beépített Nyelv) azoknak a programozóknak áll rendelkezésére, akik az általuk korábban jól megismert magasszintű nyelven kívánnak dolgozni, ezt a nyelvet egészíti ki a rendszer adatbázisok kezelésére alkalmas utasításokkal, eljárásokkal. Egy **Self Contained Language** (Önálló Nyelv) teljesen kerek önálló rendszer, ami lehet programozási nyelv éppúgy, mint egy paraméterezéssel kezelhető felhasználói felület (erre példa a légi- stb. társaságok helyfoglalási rendszerei). A tervezéskor az adatbázis felügyelőnek figyelembe kell vennie az adott információ-feldolgozás előzményeit, és nyitva kell hagynia a későbbi változtatások lehetőségét. Ezt segíti elő a **logikai** és **fizikai adatfüggetlenség**. A logikai adatfüggetlenség azt jelenti, hogy az adatok logikai struktúrájában történő változtatás ne érintse az ezen változtatást nem igénylő felhasználók munkáját. Hasonló követelmény a fizikai struktúrával kapcsolatban a fizikai adatfüggetlenség. A

két követelmény együttes teljesítése esetén beszélhetünk **adatfüggetlenségről**. Ugyancsak ügyelni kell a tervezéskor biztonság (géphiba, természeti csapás stb. elleni védelem), a titkosság (illetéktelen hozzáférések megakadályozása v.ö. 6.pont), továbbá a pontosság (lehetőleg ne kerüljön az adatbázisba hibás adat) követelményeire. Mivel az adatbázisokat többnyire egyidejűleg is többen használják, gondoskodni kell a változtató tranzakciók alatt az adatok lezárásáról, az ebből adódó patthelyzetek felismeréséről, a sikeres tranzakciók véglegesítéséről, a sikertelenek visszagörgetéséről. Végezetül nem hagyhatók figyelmen kívül a válaszidő és a költségek kérdése sem. Az adatbáziskezelő rendszerek építenek az operációs rendszerek data management rutinjaira, így rendszerint a fizikai input/output lebonyolítását az operációs rendszerre bízzák. Az adatbázis kezelő rendszereket illetően ezidő szerint három modell (approach) ismeretes: a hierarchikus, a hálós, és a relációs.

A hierarchikus modell

A legrégebbi modell, ma már nem használatos. A csoport jellegzetes képviselője az IBM által készített IMS. Lényege, hogy az adatokat fákban tároljuk, ahol a fák egy-egy szögpontja a **szegmens** (segment) adatokat és a további szegmensekre utaló mutatókat tartalmaz. A fák gyökérelemei hagyományos állományokba vannak szervezve. Az egyes nézetek a számukra **érzékeny szegmenseket** (sensitive segment) látják. A modell a hálós struktúrájú feladatok leírására csak korlátozottan alkalmas, ilyen esetekben a lekérdezés hatékonysága erősen függ az adatbázis struktúrájával.

A hálós modell

A hierarchikus modell ki nem elégítő volta miatt hamarosan szükségessé vált az új modell kidolgozása. Sok individuális próbálkozás után a CODASYL bizottság által létrehozott DBTG (Data Base Task Group) jelentései (1969-1971) hozták létre azt a terminológiai és metodológiai közös alapot, amin a hierarchikus rendszereket fejleszteni lehetett. Mintegy két évtizedig ez a jelentés volt a nagy adatbáziskezelő rendszerek tervezésének alfája és omegája. A DBTG jelentés terminológiai javaslataiból már használtunk néhányat: ilyen pl. a séma, az alséma, DDL, DML stb. A legfontosabb metodológiai újítás a **set** fogalmának bevezetése volt. A set egy rekordokból álló kétszintű fa, melynek gyökéreleme a **tulajdonos** (owner), levelei a **tagok** (members). Természetesen egy-egy rekord lehet az egyik set-ben tulajdonos, egy másikban tag, s ugyanígy lehetséges, hogy egy rekord több set-nek is tagja legyen. A set-ek segítségével a legbonyolultabb hálós kapcsolatok is leírhatók. (Esetleg kapcsolórekordok bevezetésével.) A másik fontos új fogalom a terület (area), mely együtt kezelendő rekordok egy halmazát jelenti. A hálós adatbáziskezelő rendszerek egyik fontos reprezentánsa a Magyarországon is elterjedt IDMS (Integrated Data Management System), melyben a fentieken kívül rendelkezni lehet a set-ek tagjainak rendezéséről, a rekordok valamilyen hashing algoritmus szerinti elhelyezéséről, indextáblák készítéséről stb. is. E tulajdonságok a hálós adatbáziskezelő rendszereknek nem kötelező, de gyakori tartozékai. DML-ként a DBTG jelentés a COBOL nyelv host language-ként való használatát javasolta (akkoriban szinte kizárólag kötegelt (batch) feldolgozást használtak, s az elterjedt magasszintű nyelvek közül csak a COBOL volt alkalmas adatok manipulálására). A COBOL-t sok helyen hamar felváltotta a PLI, majd az interaktív feldolgozás térnyerésével a más, ezt kihasználó felhasználói felületek is.

A relációs modell

A relációs modell ötlete Codd 1970-es cikkéből származik, így lényegében egyidős a DBTG jelentéssel. Mégis csaknem húsz évet kellet várni arra, hogy ez a modell átvegye a vezető szerepet. Az ok: a számítógépek kapacitásának és sebességének kellet növekednie ahhoz, hogy a relációs modellt hatékonyan implementálni lehessen. Mindenesetre napjainkban szinte kizárólag e modell alapján készülnek adatbáziskezelő rendszerek. A **reláció** ebben az értelemben tulajdonképpen egy **táblázat** (table) a gyakorlati szakirodalom így is említi. A táblázat oszlopai a **tulajdonság**ok (domains), a sorai az ***n-es***ek (tuples). A hagyományos terminológiának megfelelően azonban gyakran nevezik azonban a sorokat rekordnak, az egyes oszlopokhoz tartozó értékeket mezőnek (field). A táblázattal kapcsolatos alapvető feltételezések, hogy abban ne legyenek teljesen megegyező tartalmú sorok vagy oszlopok, továbbá a sorok és az oszlopok sorrendje ne hordozzon információt. Azt a mezőt, vagy mezőkészletet, mely a sort (a sor többi elemét) egyértelműen meghatározza, ***kulcs***nak nevezzük. Karbantartási anomáliák és információvesztések elkerülése végett a relációkat célszerű **normalizálni**. A normalizálás legfontosabb lépései az alábbi **normál formák**on át vezetnek. a.) 1NF A relációt elsőrendben normalizáltnak nevezzük, ha annak mezője elemi értéket (nem relációt) tartalmaz. b.) 2NF A relációt másodrendben normalizáltnak nevezzük, ha elsőrendben normalizált, és amennyiben valamelyik mezőjének azonosításához egy összetett kulcs szükséges, nincs olyan mező, melynek azonosításához elegendő ennek egy része. c.) 3NF A relációt harmadrendben normalizáltnak nevezzük, ha másodrendben normalizált, és a nem kulcs jellemzői nem függenek egymástól. d.) BCNF A relációt BOYCE-CODD értelemben normalizáltnak nevezzük, ha a fentieken kívül teljesül az is, hogy egyetlen nem kulcs jellemző sem határozza meg egy összetett kulcs valamelyik összetevőjét (nincs kulcstörés). Bizonyítható, hogy minden reláció felbontható ilyen, normalizált relációkra, ezt a műveletet nevezzük dekompozíciónak. A relációk legfontosabb tulajdonsága, hogy azok exakt matematikai eszközökkel kezelhetőek. Ilyen eszközrendszer a **relációs algebra** és a **relációs kalkulus**.

A relációs algebra

A.) A relációs algebra alapműveletei*:*

1.) Unió: RkS Az R és az S reláció unióján azt a relációt értjük, melyben szerepelnek mindazon sorok, melyek akár az R, akár az S relációban szerepelnek. Az R és az S reláció (és természetesen az eredmény reláció) sorhossza meg kell egyezzék.

2.) Különbség: R-S Az R és az S reláció különbségén azt a relációt értjük, melyben csak azok a sorok szerepelnek, melyek benne vannak az R, de nincsenek benne az S relációban.

3.) Direkt szorzat: RkS A r sorhosszúságú R és a s sorhosszúságú S reláció direkt szorzatán azt a relációt értjük, melynek r+s hosszúságú soraiban minden R-beli sort minden S-beli sor előtt előfordul.

4.) Projekció: k (R) Az R reláció i1,i2,...ik osylopaira való projekción azt a relációt értjük, mely az R relációból úgy keletkezik, hogy elhagyjuk a felsoroltakon kívüli oszlopokat.

5.) Szelekció: k (R) Az R relációnak az F feltétel melletti szelekcióján azt a relációt értjük, mely az R relációból úgy származtatható, hogy abból csak az F feltételnek eleget tévő sorokat hagyjuk meg.Az F feltétel az [i]k [j] (a sor i-edik és j-edik eleme között fennáll k) és az [i] kc (a sor i-edik eleme és a c konstans között fennáll k) elemi kifejezések logikai műveletekkel (k, k,k) való összekötéséből állhat. K tetszőleges összehasonlító operátort (k, k, kk, kk, k,k) jelenthet.

B.)* A következmény műveletek*:

A soronkövetkező műveletek voltaképp nélkülözhetőek, de a gyakorlatban jól lehet őket használni.

1.) Metszet: RkS RkS = R - (R - S)

2.) Hányados: RkS Az R és az S reláció hányadosán azt a relációt értjük, melyre igaz, hogy (Rk S)k S összes sora R-beli sor. (Feltesszük, hogy r > s, és s k 0.) A hányados valóban következmény művelet. Legyen ugyanis T = k (R), továbbá V = k_1 ((TkS)-R). Ekkor belátható, hogy RkS = T - V.

3.) Feltételes kapcsolat (k join): R * S=k (RkS) [i]k[j] [i]k[r+j]

4.) Természetes kapcsolat (natural join): R * S Hasonló a feltételes kapcsolathoz de itt a szelekció feltétele az, hogy az azonos nevű oszlopokban azonos érték szerepeljen. Természetesen a szelekció után projekcióra is szükség lesz, az azonos tartalmú oszlopok egyikét meg kell szüntetni.

C.) **Példa a relációs algebra alkalmazására** Tekintsük az alábbi adatbázist: legyen három relációnk.

a.) Autók: jele A Mezői: rendszám, gyártmány, típus, szín, ...

b.) Emberek: jele E Mezői: szemszám, név, foglalkozás, cím, ...

c.) Kapcsolat: jele K Mezői: forgrsz, szemszám.

Feladat*:* keressük ki a sárga opelek tulajdonosainak foglalkozását.

A megoldás*:* R1 =k_l (A) R2 =k (R1) R3 = R1 * K rendszám=forgrsz R4 = k (R3) R5 = R3 * E R6 = $k_{ás}$ (R5) Természetesen mindezt egy formulába is foglalhatjuk: k_f(k (k (k (A))*K)*E). rendszám=forgrsz

A relációs kalkulus

A másik relációk kezelésére használatos matematikai módszer a relációs kalkulus. Az alábbi formájú kifejezéseket vizsgáljuk: {t | k(t)}. Ennek jelentése: azokat a t sorokat tekintjük, melyek kielégítik a k (t) formulát. A formula következő atomokból épülhet fel: a.) R(s) azt jelenti, hogy az s sor része az R relációnak, b.) u[i] k v[j] azt jelenti, hogy az u sor i-edik és a v sor j-edik eleme között fennáll a k viszony, c.) u[i] k c azt jelenti, hogy az u sor i-edik eleme és a c konstans között fennáll a k viszony. Az atomok önmagukban is formulák, de azokat a logikai műveletek jeleivel (k, k,k) összeköthetjük, ezenkívül a k (van olyan, hogy) és a k (minden olyan) és a zárójelek is használhatóak. A relációs kalkulus eszközeivel minden felírható, amit a relációs algebrával kifejezhetünk. Ez egyszerűen belátható az alapműveleteknek megfelelő kifejezések felírásával.1.) Egyesítés {t | R(t) k S(t)} 2.) Különbség {t | R(t) k kS(t)} 3.) Direkt szorzat {t|(kkku)(k v)(R(u) kS(v)k t[1]=u[1]k.....k t[r]=u[r]k t[r+1]=v[1]...k t[r+s]=v[s]} 4.) Projekció{$t^{(k)}$ | (ku)(R(u) kt[1]=u[i1] k...k t[k]=u[ik]} 5.)Szelekció {t | R(t) kF} Ugyanakkor készíthetünk olyan kifejezéseket is, melyek a relációs algebrában nem fejezhetők ki. Pl. {t | kR(t)}. Könnyen látható, hogy ennek a kifejezésnek nem sok értelme van. Az effajta kifejezések kizárásával létrehozhatjuk az u.n. **biztos kifejezések** körét. Ehhez szükség van a DOM függvény bevezetésére. Egy k formulához rendelt DOM(k) azoknak az elemeknek a készletét jelenti, melyek a k formulában közvetlen, vagy közvetett módon említésre kerülnek. Ahhoz, hogy egy kifejezés biztos legyen, az alábbi feltételeknek kell eleget tennie. 1.) Ha egy t sor kielégíti a k formulát, miden komponense tagja DOM(k)-nek. 2.) kminden (ku)(k (u)) formájú részkifejezésére ahol u kielégíti k(u)-t, u minden komponense tagja DOM(k)-nak. A biztos kifejezések készletéről már belátható, hogy equivalens a relációs algebra kifejezéskészletével.

Lekérdezés relációs rendszerekben

Az első lekérdező rendszerek pusztán a relációs algebra vagy kalkulus számítógépes megvalósítását tűzték ki célul. (Ilyen például az IBM által kifejlesztett **ISBL**.) Ez azonban nem elégíthette ki a felhasználói felületekkel kapcsolatos egyre magasabb igényeket. Viszonylag korai nyelv az ugyancsak IBM termék **QBE** (Query by Example), mely táblavázak (skeleton) részbeni kitöltése után a táblázat fennmaradó részének kitöltésével ad választ a felhasználó kérdéseire. Ez a technika a legújabb rendszerekben is megtalálható. A QBE változó vektorok alkalmazásával tette lehetővé az egyes relációk összekapcsolását.

A.) **Az SQL** Manapság szabványnak tekinthető az **SQL** (Structured Query Language), melyet szinte minden korszerű adat-báziskezelő rendszer "ért".Az SQL - némiképp eltérően a hagyományos szokásoktól - négy nyelvet, ill. résznyelvet tartalmaz.

- **a.)** **A DDL** Ezen lehet táblákat, nézettáblákat (view) és indexeket létrehozni (CREATE), táblákat módosítani (ALTER), valamint táblákat, nézettáblákat és indexeket megszüntetni (DROP).
- **b.)** **A DCL** (Data Control Language) E nyelv feladata kettős, egyrészt ide tartozik a jogosítványok kiosztása és visszavonása (GRANT, REVOKE), másrészt e nyelven lehet a tranzakciókat véglegesíteni, ill. visszagörgetni (COMMIT, ROLLBACK).
- **c.)** **A DML** E résznyelv tartalmazza a rekordok beszúrásához (INSERT), módosításához (UPDATE) és törléséhez (DELETE) szükséges utasításokat.
- **d.)** **A QUERY** Ez a résznyelv egyetlen utasításból áll (SELECT), mely azonban a legösszetettebb kérdések megfogalmazására is alkalmas.

B.) **A negyedik generációs nyelvek (4GL)** A modern adat-báziskezelő rendszerek még az SQL-beli "programozást" is meg akarják takarítani a felhasználónak. Különféle paraméterezhető form, report, menu stb. generátorok teszik ezt lehetővé.

Osztott adatbázisok kezelése A nagy hálózatok és az ezek következtében sok távoli lekérdezés a 80-as évekre erősen megnövelte a kommunikációs költségek részarányát az adatbázis-kezelés költségein belül. Ez a tény sugallta az ötletet: próbáljuk meg az adatokat a felhasználás közelében elhelyezni. Az eredmény: egy **fizikailag megosztott, de logikailag egységes adatbázis*.*** Az osztott jelleg a felhasználó számára nem látható (transzparens), ugyanakkor több jellegzetes előnnyel és hátránnyal jár.

Az előnyök*:*

1.) A kommunikációs költségek már említett csökkenése.
2.) Mindenki a számára ismerős adatokat gondozza.
3.) Egy-egy csomópont kiesése esetén a többi adatai továbbra is elérhetőek.
4.) Lehetséges a moduláris tervezés, a rugalmas konfigurálás.
5.) Hosszabb idő alatt a rendszer gépei akár ki is cserélhetők.

A hátrányok*:*

1.) A rendszer bonyolultabb és sebezhetőbb lesz,
2.) Nem könnyű minden csomópontra egyformán jó személyzetet találni, másrészt, ha találunk fenyeget a szuboptimalizáció veszélye.
3.) Mindig valamennyi gépnek működnie kell.
4.) Többféle hardvert és szoftvert kell a rendszernek kezelnie és "összehoznia".
5.) Bonyolult a jogosultságok ellenőrzése (a jogosultságokat leíró táblázatokat hol tároljuk: egy csomópontban, vagy mindenütt?). Külön problémát jelent, ha feladjuk a redundancia-mentesség elvét. Erre okot szolgáltathat az is, ha nem eldönthető egy-egy adatról, hogy hol használják legtöbbet, de biztonsági

okokból is dönthetünk egy-egy adat többszörözése mellett. Ilyen estekben biztosítanunk kell, hogy az egyes példányok tartalma azonos legyen, azaz a rendszer **konzisztens** maradjon. Az első osztott rendszerek megvalósításához nagy segítséget nyújtott az u.n. **backend gépek** felhasználása. Az adatok elhelyezése tervezéskor az adatbázis felügyelő feladata (Mivel az osztott rendszereknél megkell különböztetnünk központi és csomóponti adatbázis felügyeletet, pontosabban a központi adatbázis felügyeleté). A már működő rendszerek automatikusan is képesek a felhasználás gyakoriságát figyelve a kópiák számán és elhelyezkedésén változtatni.

Az adatbázis felügyelő számára több elemzési módszer áll rendelkezésre döntései meghozatalakor.

a.) **A forrás-nyelő elemzés** Elsősorban a felhasználás gyakoriságát kell vizsgálnunk.Nem mindegy azonban, hogy a felhasználó csomópont nyelő (azaz csak olvassa az adatot, s így a felhasználáskor elég a legközelebbi példányt megtalálnia), vagy forrás (mely az adatot létrehozza, vagy módosítja).

b.) **Az ABC elemzés** Az adatok kategóriákba sorolhatók "nélkülözhetetlenség" szerint. Ezekhez a kategóriákhoz különböző minimális kópiaszámot rendelhetünk.

c.) **Az érzékenység elemzés** A különböző csomópontok költség/teljesítmény aránya más és más lehet. Nyilvánvaló, hogy a teljesítményt "kevésbé érzékeny" csomópontoknál növelni, az "érzékenyebbeknél" csökkenteni kell. Említettük már a konzisztencia megőrzésének fontosságát. Ez a **szinkronizációs protokollok** feladata. E protokollok vagy biztosítják az **erős konzisztenciát** (vagyis azt, hogy egy-egy adat kópiái egyszerre változzanak meg), vagy a **gyenge konzisztencia** (amikor az adatbázis hosszabb-rövidebb ideig inkonzisztens marad) mellett gondoskodnak arról, hogy ez ne okozhasson problémát. A konzisztencia mérőszáma a **koherencia**, mely erős konzisztenciánál azonosan 1 értékű, gyenge konzisztenciánál pedig 1-hez tart. Következtetésképp a koherencia konvergenciája a konzisztencia feltétele.

A szinkronizációs protokollokat két részre oszthatjuk. A központosított protokollokban az egyik (nem feltétlenül mindig ugyanaz) csomópont szerepe kitüntetett, az osztott protokolloknál ilyen nincs.

A.) Központosított protokollok

a.) A központi zárellenőrzés Egy állandó központ végzi el a változtatásokhoz szükséges lezárásokat. Működése hasonlít az osztatlan adatbázisoknál megszokotthoz. A központ sérülésére érzékeny, csak statikus adatbázisoknál használható, erősen konzisztens módszer.

b.) A zseton módszer Egy, a csomópontok között körbejáró "zseton" jelöli ki az alkalmi központot, mely a változtatásokhoz szükséges lezárásokat elvégezheti. Igen kedvelt, erősen konzisztens módszer.

c.) Az elsődleges példány módszer Egy adat kópiáit egy szekvenciába szervezzük, s a változtatások csak ennek mentében történhetnek. Mivel a tranzakciók nem előzhetik meg egymást, nem okoz gondot, hogy a módszer gyengén konzisztens. Előnye, hogy az ideiglenesen működésképtelen csomópontok újra munkába állítása egyszerű

C.) **Osztott protokollok** Az időbélyeg módszer E módszernél a tranzakciók elindításuk időpontjából és a csomópont azonosítójából egy időbélyeget kapnak, mely meghatározza végrehajtásuk sorrendjét. A távolságból adódó

félreértéseket rendszeres dummy üzenetek küldésével kerülik el. A módszer gyengén konzisztens.

Szakértői rendszerek

Az információkezelés legfiatalabb ága az u.n. szakértői rendszerek készítése. Felépítésük ill. elkészítésük alapvető sémája a következő. A felhasználó egy **felhasználói felület**tel (user interface) áll kapcsolatban, mely program a felhasználó és a rendszer közötti dialógust vezérli. A felhasználói felület a **következtető rendszer**hez (inference engine) kapcsolódik. Ez a rendszer "lelke" mely a **tudásbázis** és esetleg egyéb adatbázisok felhasználásával a válaszokat kidolgozza. Az **okoskodás**, következtetés (reasoning) lehet **adat**- vagy **célvezérelt**. A tudásbázis szabályok és állítások gyűjteménye, melyet a **tudásmérnök** (knowledge engineer) tölt fel egy **területi szakértő** (domain expert) tudásának számítógépben tárolható alakra "fordításával" (a szakértői rendszerek - ezidő szerint legalábbis - csak egy-egy elég szűk tudományterületet ölelnek fel). Fontos, hogy az eredeti és a számítógépes tudás közötti **fordítási szakadék** (semantic gap) a lehető legkisebb legyen. Számos szakértői rendszerek készítésére alkalmas keretrendszer (shell) áll a felhasználók rendelkezésére, melyben a felhasználói felület, a következtető rendszer és a tudásbázis struktúrája adott. Egyes rendszereknél azonban ezek a komponensek is többé-kevésbé változtathatók. A fejlesztés, változtatás a **rendszermérnök** (system engineer) feladata. A szakértői rendszereknek nem csak biztos állításokat kell kezelniök, mind a rendszerbeli tudás, mind a felhasználó válaszai tartalmazhatnak bizonytalanságot. Ezt a -100 és 100 közé eső **bizonyossági tényező**vel (CF = certainty factor) írhatjuk le. Az egyszerű állításokon és összefüggéseken kívül e rendszerek speciális adatelemeket, összetett és hierarchiákban megjelenő **váz**akat (frame) és aktív u.n. **démon**okat is tartalmazhatnak. Alapvető elvárás a szakértői rendszerekkel kapcsolatban, hogy következtetéseiket megmagyarázzák, s a modern rendszerek tanulni is képesek.

Az adatok bizalmas kezelése

A modern ipar, kereskedelem, államigazgatás, stb. (azaz a társadalom) zavartalan működéséhez azt jól kiszolgáló informatikai rendszereket igényel. Alapvető fontosságú ezen rendszerek védelme a behatolás szabotázs, stb. ellen. Az USA-ban már a 80-as években kimutatták, hogy egy-egy számítógépes bűncselekmény nagyságrendekkel nagyobb kárt okoz, mint egy átlagos bankrablás. A védelemnek alapvetően három formája van:

a.) **fizikai védelem** mely az adatokhoz való illetéktelen hozzáférést fizikailag próbálja megakadályozni (Zárt, őrzött számítóközpont, stb.. A modern rendszerekben az adatok ilyenfajta védelme a műholdas adatátvitel elterjedése óta alig lehetséges, hisz az ilyen vonalak vételének megakadályozása fizikailag lehetetlen.),

b.) **az ügyviteli védelem** mely a követendő biztonságtechnikai szabályokat, kötelező viselkedési módokat, továbbá a kötelező dokumentálás rendjét írja elő (elsősorban a felelősségkérdését szabályozza),

c.) az algoritmikus védelem olyan algoritmusok összessége, mely a fentieket hatékonyan elősegítő, kiszolgáló algoritmusokat tartalmaz. (Mi csak ezzel foglalkozunk.) Az algoritmikus védelem is tovább osztható: a **rejtjelezés** és az **üzenethitelesítés** lehetővé teszi az adatok védetlen közegen való továbbítását, a **felhasználó azonosítás** a rendszert használó személyek egyértelmű azonosítására szolgál (ugyanez a feladata számítógépek kapcsolata esetén a **partner azonosítás**nak), a **hozzáférés védelem** megakadályozza az egyébként jogos

felhasználót abban, hogy hatáskörét túllépje, végül a **digitális kézjegy** az elküldött üzenetek letagadását akadályozza meg.

A rejtjelezés Régóta ismeretes, hogy a védhetetlen közegen átjuttatandó x üzenetet valamilyen $y = E(x)$ transzformációval torzítani érdemes, természetesen úgy, hogy az üzenet címzettje képes legyen az inverz $x = D(y)$ transzformációra. A transzformációktól elvárjuk, hogy (legalábbis az üzenet "érvényességi idején" belül) kívülálló számára megfejthetetlenek legyenek, a címzett ugyanakkor gyorsan és könnyen megértse azokat. Mivel nem könnyű nagyszámú megfelelő E-D párt előállítani, nem egy, hanem kétváltozós transzformációkat célszerű alkalmazni, ahol a másik változó az esetenként cserélhető kulcs. Így a transzformációk akár nyilvánosak lehetnek, csak a kulcsok titkos kezelését kell megoldani. Ha az E transzformációnál alkalmazandó K_r rejtőkulcsból ennek D transzformációnál alkalmazandó K_f fejtőkulcs párja könnyen meghatározható, **konvencionális kódolás**ról beszélhetünk, ellenkező esetben a kódolás **nyilvános (rejtő) kulcsú**.

A.) A konvencionális kódolás A legelső konvencionális kódolási eljárás Julius Caesar nevéhez fűződik: üzenteiben a latin abc minden betűje helyett a rákövetkező harmadikat használta. Főleg katonai körökben alkalmazták a következő eljárásokat.

a.) Helyettesítés A nyílt szöveg minden betűjét megadott rend szerin egy másikkal helyettesíthetjük. (Pl. az abc-hez egy abc permutációt rendelünk.) A nyelvi sajátosságok (betűk gyakorisága) alapján megfejthető.

b.) Periodikus helyettesítés Az előzőhöz hasonló de több helyettesítési szabályt használunk periodikusan cserélve (pl. több abc permutációt). Ez is megfejthető: azonos betűkombinációkból lehet a periódus hosszára következtetni, s ezután az egyszerű helyettesítésnél leírtakhoz hasonlóan okoskodni.

c.) Kulcsfolyamos rejtés A helyettesítést aperiodikussá tesszük. A helyettesítést egy szöveg vezérli, az abc hosszának megfelelő számú különböző abc permutációból álló mátrixban a rejtendő szöveg betűjének megfelelő oszlopban és a kulcsszöveg soron következő betűjének megfelelő sorban álló betű lesz a rejtett szöveg következő betűje. A agyméretű mátrix tárolását elkerülendő, választhatjuk az i-edik sort az abc i lépéses eltolásának. A betűk reprezentálhatóak az abc-n belüli sorszámaikkal. Ekkor a kódolás (és a dekódolás) leegyszerűsödik a kódolandó (dekódolandó) és a kulcsszöveg megfelelő betűinek (mod abc hossz) való összeadására. Sajnos ez a kódolás is feltörhető (egy betűsorozatból két szöveg bontható ki!). A ulcsfolyamos rejtéshez hasonló kódolási eljárás a számítástechnikában is alkalmazható: ha a a kódolandó és a kulcsfolyam mod 2 összeadása a kódolás, ugyanez lehet a dekódolás módja is. A kulcsfolyamot valamilyen véletlenszám generátor állíthatja elő, ekkor kódolási kulcsként csak ennek elindítási módját kell a partnereknek egyeztetniök. Sajnos az előbbiekben vázolt módszer nagyon érzékeny a szinkronitásra (egy bit kiesése az egész üzenetet értelmetlenné teszi). Ezért az üzeneteket többnyire blokkokba rendezik, s így küldik el.

A **rejtjelötvözés** azt jelenti, hogy több egyszerű transzformáció egymás utáni alkalmazásával élünk. 1949-ben Shannon javasolta a **keverő transzformációk** bevezetését, mely váltakozva helyettesítések (s-réteg) és permutációk (p-réteg) egymásutánjából áll. A Shannon féle elvek továbbfejlesztése alapján készítette az IBM a 60- as évek második felében a **LUCIFER** berendezést (128 bites blokkok, 128 bites kulcs, 6-7 emberév fejlesztési munka). Ezt követte az egyetlen LSI áramkörrel elkészíthető **DES** (Data Encryption Standard, 64 bites blokk, 56 bites kulcs). Ez utóbbi 1977 óta USA szabvány. A DES-t számos támadás érte, de az idő a fejlesztőket igazolta. (Diffie, Hellman egy elméleti gépet is szerkesztett a DES-

sel kódolt üzenetek feltörésére. A gép fél nap alatt feltörte volna az üzeneteket, de teljesítmény-felvétele 2 Mw lett volna, így gondolatkisérlet maradt.)

C.) **A nyilvános (rejtő) kulcsú kódolás** E módszerek alapja valamilyen nehezen invertálható u.n. egyirányú (trap-door) függvény. Az MIT eljárásnál ahhoz, hogy a rejtőkulcsból a fejtőkulcs meghatározható legyen, egy többszáz jegyű számot kell prímtényezőkre bontania. Míg egy szám prímtényezőiből való előállítása (összeszorzása) egyszerű és gyorsan megvalósítható munka (néhány száz jegyű számokkal 1ksec alatt lehet műveleteket végezni, ugyanakkor a ma ismert leghatékonyabb eljárással egy 200 jegyű szám prímtényezőkre bontásához $3.8*10^9$ évre van szükség.) Egy másik ismert megoldás, a Merkle-Hellman módszer az u.n. lefedési feladaton alapszik.

A kulcsgondozás

Nyilvánvaló, hogy ha a kulcsok megfelelő védelme nem biztosított, az egész kódolási rendszer értelmetlen. A kulcsok gondozása három feladatbál áll.

A.) **Kulcsgenerálás** Az a művelet, melynek eredményeképpen a kulcsok előállnak. Egy valódi véletlenszám generátor segítségével végrehajtható.

B.) **B.) Kulcskiosztás**

a.) Alapkulcsok A kulcsok kiosztása történhet egy **alapkulcs készleten** keresztül, melyet rendszeren kívüli eszközökkel juttatnak el a résztvevőkhöz. Az alapkulcsokat, melyek gyakran nyilvános kulcsú rendszerhez tartoznak, csak a kulcsok cseréjéhez használják.

b.) Merkle "rejtvény" módszere A hívó fél n db (K_i, I_i) párt küld partnerének gyengén kódolva. Ez egyet kiválaszt, feltöri, és I_i-t visszaküldi. Ezzel a kommunikáció kulcsa meghatározott. A behatolónak átlagosan a párok felét kell ahhoz feltörnie, hogy megtudja, I_i-hez melyik K_i tartozik.

c.) A "hatványozós" módszer A módszer alapja, hogy az i és a j felhasználó kitalál egy-egy xi ill. xj számot. Egymásközt kicserélik az $Yi = k^{xi} \pmod p$, ill. az $Yj = k^{xj} \pmod p$ számokat (p prímszám, k a p elemű véges test egy primitív eleme), a kommunikáció kulcsa a $K = k^j$ szám (vagy annak valamilyen függvénye) lehet. Ennek előállítása Yj és xi vagy Yi és xj ismeretében egy egyszerű hatványozást igényel, a behatolónak viszont a diszkrét logaritmusképzés a feladata.

C.) **Kulcstárolás** Nehézségeket okozhat, ha a kulcsokat akár túl kevés, akár túl sok ember ismeri. (Az első esetben esetleg szükség esetén nem áll rendelkezésre a kulcs, a második esetben nagy a kiszivárgás veszélye.) Megoldást jelentenek az u.n. **(n,k) küszöbrendszerek**. E rendszerek lényege, hogy a kulcsot n db. (nem feltétlenül diszjunkt) részre osztva, bármelyik k kulcsrészletből a kulcs előállítható, de nincs olyan k-1 kulcsrészlet, amiből ez megtehető lenne. Ilyen küszöbrendszerek készítésére több matematikai módszer is rendelkezésünkre áll.

Felhasználó és partner azonosítás, hozzáférés védelem

A.) **A felhasználó azonosítás** a.) A jelszóvédelem Talán a legrégebbi felhasználó azonosítási mód a jelszavak használata. A módszer használhatóságát azonban csökkenti, hogy a túl hosszú és értelmetlen jelszavakat nehéz megjegyezni, ugyanakkor a rövid és értelmes szavak gyakran egyszerű próbálkozással

kitalálhatóak. Ezért a jelszavakat célszerű gyakran változtatni. Ez többféle alapon történhet a legbiztosabb valami előre rögzített algoritmust használni. A jelszavakat a számítógépes rendszer nem közvetlenül, hanem valamilyen egyirányú transzformáció alkalmazása után tárolja, így a tároló táblázatból sem lehet azokat megállapítani b.) fizikai azonosító használata A felhasználók azonosítására gyakran használnak valamilyen fizikai eszközt. A legelterjedtebbek a különféle azonosító kártyák. Ezek nehezen hamisíthatók, de könnyen el lehet őket lopni. Jelentősen fokozhatja a biztonságot a fizikai azonosító eszköz és a jelszó együttes használata. c.) személyi jellemzők Nem lophatók el, és nem hamisíthatók a különféle személyi jellemzők (ujj- vagy tenyérlenyomat, recehártya erezet, stb.). E jellemzők számí-tógépes tárolása és kiértékelése még nem teljesen megoldott, a biztató eredmények azonban azt sugallják, hogy a jövő útja erre keresendő.

B.) **A hozzáférés védelem** Nem elegendő kiszűrni a jogosulatlan behatolókat, a rendszernek azt is számon kell tartania, hogy a jogos felhasználók hatásköre mire terjed ki. A felhasználó által működtetett **ügynök folyamatok** hatáskörét a **hozzáférési mátrix** szabja meg. Ennek elemeit akár ügynökökhöz, akár adatokhoz kötötten tárolhatjuk.

C.) A partnerazonosítás A számítógép-számítógép kapcsolatokban is szükség lehet azonosításra. E célra fenntarthat minden számítógép pár egy-egy kulcsot, ez azonban egy n elemű hálózatnál n^2 kulcsot jelent. Folyhatnak az információcserék valamilyen hitelesítő központon keresztül, ez azonban a kommunikáció költségét növeli jelentősen. Megoldást jelenthet, ha minden csomópont egy, csak a hitelesítő központ által ismert kulccsal tud e központhoz bejelentkezni, s üzenet továbbítási igényét bejelenteni. A központi jelöli ki a kommunikáció kulcsát, melyet a fenti kulccsal kódolva a hívónak visszaküld. Ugyancsak küld emellett egy csomagot, mely a hívandó fél kulcsával kódolva tartalmazza a kommunikáció kulcsát, s a hívó megjelölését. Ha a hívó e csomagot a hívott félhez továbbítja a párbeszéd kettejük között folytatódhat, s az azonosítás is kielégítő biztonsággal történt meg. **Üzenethitelesítés, digitális kézjegy** Az üzenetek hitelesítéséhez két feltétele van, egyrészt ellenőrizni kell, hogy egy-egy blokkban az és csak az érkezett-e a címzetthez amit a feladó feladott, másrészt tudnunk kell azt is, hogy az egyes blokkok abban a sorrendben érkeztek e meg amilyenben feladták őket (ideértve azt is, hogy nem hiányzik-e közülük). Az első célhoz ellenőrző összegeket, a másodikhoz sorszámot célszerű a blokkban elhelyezni még a kódolás előtt. A **digitális kézjegy** arra szolgál, hogy segítségével a címzett megbizonyosodhassék egy üzenet feladójáról és bizonyíthassa, hogy az illetőtől kapott ilyen üzenetet. Olyasmire van szükség tehát mint az aláírás, ami könnyen azonosítható, de nehezen hamisítható. A cél elérhető úgy is, hogy a kényes kommunikációt egy hitelesítő központ közbeiktatásával végezzük (mintha tanú előtt beszélnénk) ez az u.n. **nem valódi digitális kézjegy**. Elegánsabb megoldás a **valódi digitális kézjegy**. Ehhez egy nyilvános kulcsú kódolási rendszer lehet felhasználni, mégpedig olyant, mely "megfordítható", azaz E(D(x)) = x (az általunk említett módszerek ilyenek). Használjuk a kódolási módszert fordítva, azaz

úgy, hogy a titkos fejtő kulccsal rejtünk (és nyilván a nyílt rejtő kulccsal fejtünk). Ekkor a címzett, ha a nyílt kulccsal fejthető üzenetet kap, biztos lehet abban, hogy azt csak a titkos kulcsot ismerő feladó küldhette. Ugyanakkor - mivel a címzett nem ismeri a titkos kulcsot - ha rendelkezik az üzenetnek a nyílt kulccsal megfejthető kódolt változatával, nyilvánvaló, hogy azt ő nem készíthette, azaz az üzenet a feladótól származik.