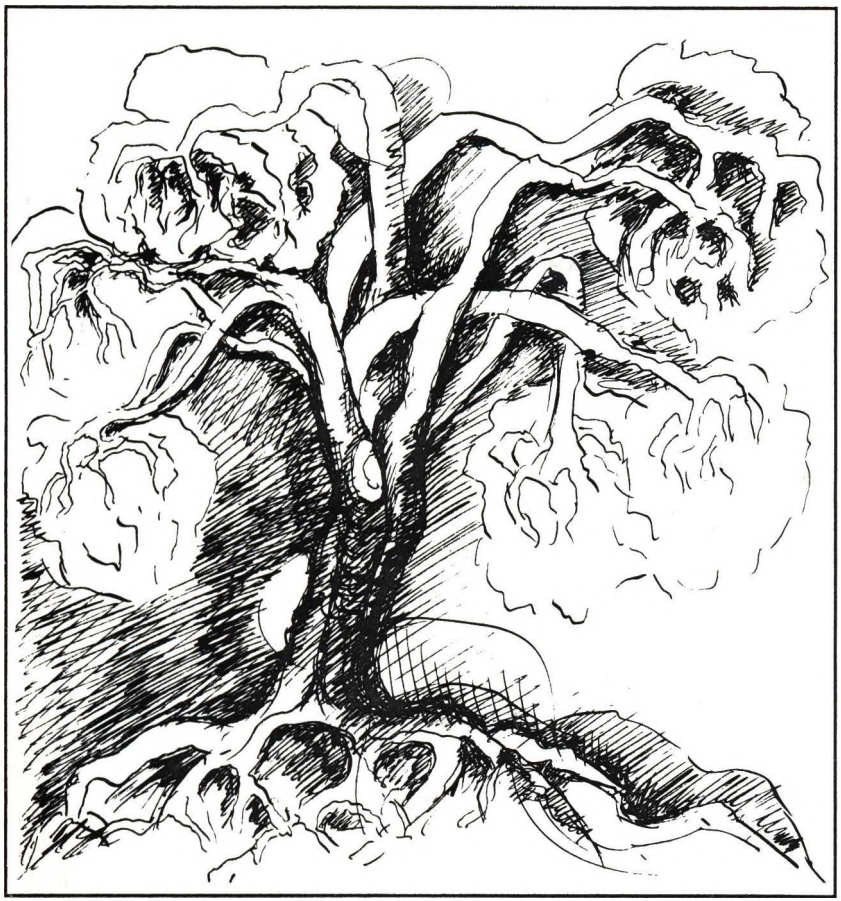
RELÁCIÓS ADATBÁZISKEZELŐK III. MAGYARORSZÁGI KONFERENCIÁJA

1997. január 28-30.



RELÁCIÓS ADATBÁZISKEZELŐK III. MAGYARORSZÁGI KONFERENCIÁJA

1997. január 28-30

FAIR Információs Rendszerek
Neumann János Számítógéptudományi Társaság

Tisztelt Résztvevő!

Szeretettel köszöntöm Önt a Relációs Adatbázis-kezelők Harmadik Magyarországi Konferenciáján.

A kiadvány, amit a kezében tart, a konferencián elhangzó előadások nagyobb részének írott anyagát tartalmazza, a tervezett elhangzás sorrendjében. Remélem, hogy a kötet megkönnyíti mind az előadások követését, mind pedig a későbbi felidézését.

Remélem azt is, hogy a konferencián ismertetett gondolatok, tapasztalatok, megoldási ötletek megkönnyítik az Ön munkáját is, s így a konferencia elérni legfőbb célkitűzését.

Szeretnék itt is köszönetet mondani mindazoknak, akik munkájukkal, ötleteikkel, biztatásukkal, tapasztalataik közreadásával, vagy más módon segítettek abban, hogy ez a konferencia létrejöjjön.

További munkájában sok sikert kívánok:

Dr. Ecsedi-Tóth Péter
FAIR Információs Rendszerek

1997. Január 28.

Adatkezelés a népességnyilvántartásban

Farkas Lajos - Meggyes István
KÖNYV K & H Rt.

A nagyméretű, országos információs rendszereket csak távolról ismerő kollégákban joggal vetődhet fel a kérdés, hogy egy személyi és lakcím adatokat tartalmazó nyilvántartásban miféle gondok, problémák merülhetnek fel, hiszen az viszonylag egyszerű feladatnak tűnik. Magunk is ezt hittük, amíg szembe nem találkoztunk a meglepő valósággal, melyből most csupán néhány szemelvényt van módunk bemutatni. Reményeink szerint talán sikerül érzékeltetni, hogy egy országos alapnyilvántartást létrehozni igen érdekes, szép munka, de ugyanakkor meglehetősen „kemény dió".

A magyar népességnyilvántartás (továbbiakban ÁN) az elmúlt több mint húsz éves története alatt számos adatkezelési korszakot élt át, s ma is egy újabb váltás előtt áll.

A kezdetek idején, az 1970-es évek elején a világ a mainframe gépekben és ahhoz kapcsolódó terminál hálózatban gondolkodott. Nem véletlen tehát, hogy az ÁN információs rendszerének felépítését is így képzelték el. Induláskor ennek szokás szerint nem voltak meg a feltételei. Csupán arra telt, hogy az akkori gépbeszerzési gyakorlatnak megfelelően - hogy ne legyünk kiszolgáltatva egyetlen "imperialista" cégnek sem - a „vetésforgó" elv szerint éppen egy Honeywell-Bull gépet vásároltak, amit bérlet formájában használhatott az Állami

Népességnyilvántartó Hivatal (ÁNH) is. Ezért egy centralizált gépi és kb. 1600 helyi kartonos nyilvántartás felállítására volt lehetőség. Maga a központi gép teljesítményben majdnem annyit tudott, mint azok a PC-k, melyeket ma már lassan kiselejteznek.

A népességnyilvántartás előkészítésével megbízott számítástechnikai cég a HwB gép IDS adatbázis kezelőjével, normalizált adatmodellel álmodta meg a központi adatbázist. Az elképzelések és a realitások közötti szakadék miatt a központi adatbázis feltöltése a 10 millió polgár adataival legkevesebb két éves(!) gépi futást igényelt volna, s a karbantartáshoz a napi 24 órát még egy-két órával meg kellett volna toldani...

A mai ÁN nagy szerencséjére ez a megoldás nem volt követhető, más megoldást kellett keresni. A szerencse ebben az volt, hogy amíg a HwB gép feldolgozó képességére, szűkös háttér kapacitására való hivatkozással újra kellett tervezni a gépi megoldást, azalatt a háttérben pótlólag meg lehetett szervezni magát a népességnyilvántartást. Tudniillik ez utóbbi munkafázist, a rendszerszervezést a számítástechnikusok jó magyar szokás szerint mindenestül kifelejtették, s az egyébként jó koncepcionális anyagok alapján egyből program modulokat kezdtek írni. Az eredményt el lehet képzelni.

Az immár megszervezett ÁN központi „adatbázisa" 35 db., külön-külön 100 megás cserélhető lemezre írt, ISP(ISAM) szervezésű személyi adatokat tartalmazó szegmensből és egy IDS-sel kezelt, 100 ezer tételt tartalmazó megye-település-közterület (út, utca, tér stb.) adatbázisból állt össze. Ezt egészítették ki a kódszótárak, pl. az utónév

kódokhoz tartozó utónevek állománya, valamint az ÁN-ben érintett szervek nyilvántartása. Némi meta-adatbázis már itt is megjelent, hiszen a rendszer adathordozói között e nélkül már nem lehetett volna eligazodni (volt olyan input állomány, amelyik kétezer mágnesszalagra fért fel). Ez a megoldás, bár sok változatot ért meg, igen hosszú életűnek bizonyult. Csak 1993-ban sikerült felváltani, ami nem a fejlesztőkön múlott.

A „minden számítástechnika a központba" elv sokáig nem volt tartható. Megjelentek a decentralizált megyei és helyi adatrögzítő rendszerek, s utazott a floppy meg a mágnesszalag. A központi adatbázis is igen messze esett a legtöbb területi-helyi felhasználótól, azt is közelebb kellett vinni. Az első sikert a ZALASZÁM-mal közösen kifejlesztett, ESZR R-22-őn futó megyei adatbázis rendszer hozta.

A megyei adatbázis rendszer a központi rendszer adott megyére vonatkozó adatait kezeli, mint a központi rendszer követő adatbázisa.

Az R-22-őn semmilyen adatkezelő szoftver nem állt rendelkezésre, így magunk írtunk egy IBM CFMS-re némileg hasonlító, szuper helytakarékos adatkezelőt. A mai értelemben vett adatmodellezésről itt még igazából nem beszélhettünk, az adatmodell nem volt normalizált.

Ezzel a rendszerrel sajnos kicsit túlnyertük magunkat, mert a már jól működő deszka-modellt be kellett mutassuk a vezetésnek, s az rögtön elrendelte annak éles bevezetését. Azonnal tapasztalhattuk, hogy az üzemeltetés-automatizálási és más kiegészítő részek

nélküli, egyébként kész és a fejlesztés helyén, Zalaegerszegen kitűnően futó rendszer országos üzemeltetése hihetetlenül problémás. Beigazolódott, amit már sejtettünk, hogy egy országos rendszert tervezni és működtetni egészen más műfaj, mint amit a szakma korábban megszokott. Pótolva a szükséges részeket ez a modell a SZÜV számítóközpontjaiban, valamennyi megyében évekig eredményesen dolgozott.

A nagyméretű állományok adatkezelésének nehézségeit jelezte, hogy más fejlesztőknek sem VIDEOTON-os minigépen, sem R-55-ön IDMS-el nem sikerült működőképes megyei modellt létrehozniuk.

A megyei TAKEH (mai nevén TÁKISZ/FÁKISZ) hálózat DEC bázisú számítógépei és az elérhető DBMS hálós adatbázis kezelő rendszer teremtette meg a továbblépés lehetőségét. Olyan számítógépek, melyek a relációs adatbáziskezeléshez szükséges teljesítménnyel rendelkeztek, továbbra sem álltak rendelkezésre.

Az újabb változatot, a VAX-os megyei rendszert a HBMÖ IK-val dolgoztunk ki, felhasználva azt a rengeteg tapasztalatot, melyet az előző modellel nyertünk. Ez a rendszer korszerűségben jóval megelőzte az akkori központi rendszert, bár mára az új központi rendszer került ugyanilyen előnybe.

A megyei adatbázisok jellemzésére két dolgot emelünk ki. A HBMÖ IK-s kollégák egy olyan érdekes felhasználói lekérdező programot fejlesztettek ki, amellyel sokféle entry point-on lehet belépni és tetszés szerint lehet

továbbnavigálni az adatbázisban, a különböző láncokat váltogatva.

A másik kérdés, hogy a megyei adatbázisok megjelenésével két szintű, osztott adatbázis rendszer jött-e létre? Erre meglehetősen furcsa a válasz: egyszerre igen is és nem is. A felhasználók számára létrejött az osztott adatbázis, hiszen ma is jól működik, használják. Számítástechnikai szempontból vizsgálva viszont a "nem" a helyes felelet, mivel a két adatbázis szint között az adatbázis karbantartásban nem él on-line kapcsolat, s nem a szokásos adatbázis szinkronizációs eljárások működnek. Megjegyezzük, hogy az első megyei rendszer változatok kifejlesztésekor a központban a mai értelemben vett "adatbázisról" még nem is beszélhettünk, mivel ott a hagyományos fájl kezelés dominált, heti egyszeri batch karbantartással. A központi rendszer olyan módszerrel frissíti a megyei adatbázisokat, ahogy az az egymástól független, "idegen" rendszerek adatcseréjénél szokás, hiszen az igazi osztott adatbázis kapcsolat egyetlen feltétele sem volt adott. Viszont itt is érvényesült a latin mondás: navigare necesse est, másképpen fogalmazva a megyei rendszereket meg kellett csinálni.

A megyei modell "kisöccseként" készült el a helyi modell, a PC-s megoldásnak megfelelő dBASE-es fájlokkal, három különböző fejlesztőnél, három változatban. Ezek logikájukban már közel álltak a relációs adatkezeléshez. A helyi modell HBMÖ IK-s változata ma több mint ezer település polgármesteri hivatalában dolgozik.

Időközben az élet haladt, s megjelentek az on-line adatszolgáltatást kérő felhasználói igények is. A HwB-s

központi rendszer a töméntelen cserélhető lemezes állományával ennek kielégítésére elvileg alkalmatlan volt, ezért a gép-skanzen bővítésével most egy SIEMENS gép állt üzembe, amely saját adatbázisaként a központi adatbázis copy-ját kezeli. Ezen a feladathoz mérten szerény méretű gépen ráadásul igen komplikált, fonetikai eljárásokkal súlyosbított lekérdezéseket kellett megoldani.

A szokásos normalizált adatmodellről - amit ekkor a megyei modellben már eredményesen alkalmaztunk - itt sem lehetett szó. Ennek ellenére igen figyelemreméltó kompromisszumos megoldás született. A fonetikai kereső indexeket (speciális match-kódokat kell érteni rajta) egy PRISMA adatbázis kezelő rendszer kezeli, míg az egy-egy személy összes aktuális adatát tartalmazó rekordokat egy tömörítő algoritmus komprimálja össze igen kis méretűre, s így a teljes népesség adata felfért a gépre. Ez a rendszer ma is működik, s mintegy 10 ezer logikai terminál kapcsolódik rá.

A központi rendszer korszerűsítésére a lehetőséget két körülmény szerencsés alakulásával a rendszerváltás teremtette meg. Egyrészt csökkent annak az érdeknek a hatása, amely korábban eredményesen akadályozta a népességnyilvántartást abban, hogy megfelelő saját számítógéphez jusson, másrészt "állambácsi" zsebében találtunk némi maradék aprópénzt, amely elegendő volt arra, hogy a központi rendszer számára az első saját és korszerű, bár a feladathoz mérten változatlanul kis konfigurációjú gépet megvegyük. Ez egy IBM AS/400-B60-as modell volt. Az AS/400-as gép kitűnően bevált, ma is ennek egy többször bővített és korszerűsített változatán (F60-as modell) van a központi adatbázis.

Az AS/400-as gépeken az OS/400-as operációs rendszer szerves része egy integrált relációs adatbázis kezelő rendszer. Ma ez az adatbázis kezelő rendszer a teljessé vált funkciókkal együtt saját nevet kapott (DB2/400), mint az IBM nagy adatbázis kezelő családjának tagja.

A központi adatbázis 1993-ban relációs modellben, normalizálva készült el. Újdonság a modellben, hogy először jött létre egy teljes, lakáscím mélységű címnyilvántartás, amely a megindulása óta a címek teljes történetét tárolja, azaz minden adatbázis állapot visszakereshető belőle.

A központi relációs adatbázisunk teljesítmény próbáinál mi is találkoztunk azzal a gonddal, hogy egy nagyméretű, normalizált relációs adatbázis alá igen komoly számítástechnikai erőforrást kell tolni. A modellben egy személy adatait átlagosan 17 különböző relációs táblából lehet összeállítani. A szokásos módon, a logikai kapcsolatokon navigálva erre túl hosszú válaszidőket kaptunk, ami egy tömeges feldolgozást (pl. 8 millió polgár adatát kell egy futás keretében kivenni az adatbázisból) akár lehetetlenné is tehet. Éppen ezért a modellt "el kellett rontani", azaz a szöveg adatokat tartalmazó, a memóriában gyakorlatilag állandóan benntartózkodó lapokon lévő relációs táblákra utaló elsődleges kulcsokat beletettük a személyek alap táblájába. Ettől a hozzáférés igen felgyorsult.

Azt is meg kell mondani, hogy a harmadik normál formájú adatbázisoknak ettől még nagyobb hívei lettünk, mivel az egyébként kiváló minőségben elkészült központi adatbázisban az egyetlen kisebb probléma éppen a

modell "elrontásából" származott. Úgy gondoljuk, hogy itt együtt kell utalni a két érintett klasszikusra, Murphy-re és Halassyra. Érdemes odafigyelni rájuk.

Ma a népességnyilvántartás nagy változás előtt áll. Az adószám, a TAJ szám bevezetése, az adatvédelmi jogszabályok, a felhasználói igények minőségi és mennyiségi növekedése, az új, biztonságos okmány-család (ID-kártya, jogosítvány stb.) bevezetése egyaránt azt követeli, hogy minden részében a kor színvonalára hozzuk fel a népességnyilvántartás számítógépes rendszereit. Ennek feltételei most végre kialakulni látszanak. Kibővülőben van a közigazgatási országos WAN hálózat, a megyékben korszerű DEC ALPHA-k jelentek meg, helyi szinten IBM RS/6000-es és más gépek is dolgoznak a PC-k mellett. A központban installálás alatt van egy IBM AS/400-53S típusú iker-szerver, amely igazi "erőgyár" a maga 8 db. RISC processzorával, 3.5 G operatív memóriájával, s a háttértár 300 G-je is elegendőnek tűnik.

Terveink (és a nagyrészt már működő kísérleti változatok) szerint

- a történeti adattárolást a személyi adatokra is kiterjesztjük,
- átállunk a client-server architektúrára,
- az adatbázisunkat valódi osztott adatbázisokká esszük, DB2 és ORACLE bázison,
- kihasználjuk a hálózat által nyújtott lehetőségeket,
- a teljes információs rendszert egy meta-adatbázisra épülő felügyelő rendszer alá helyezzük,
- egységes adatszolgáltatás engedélyező, naplózó adatbázist állítunk fel.

Tapasztalataink közül néhányat megemlítünk.

Egy ezer adatbázisból álló rendszernél a szokásos adatmodellezést meghaladó problémákkal kell megbirkózni. Ha nincs mód gyári osztott adatbázis kezelő szoftvert üzembiztos és teljes hálózat mellett használni, a megoldás legalább egy nagyságrenddel bonyolultabbá válik. Egy ekkora adatbázis rendszert emberi erőforrásokkal már nem lehet megfelelően kézbentartani, azt is automatizálni kell. Egy országos rendszerben az adatbázis megosztása is úgy változik, mint más rendszerben az adatok, s ezt tudni kell követni. Az adatbázisok szinkron állapotát folyamatosan ellenőrizni kell. Az 1 központ - sok száz végpont felépítésű rendszerben a szoftver változatok telepítése és ellenőrzése is megoldandó. Az egyértelmű adatkarbantartási jogosítás hiánya miatt (pl. akárhová költözhetünk vagy bárhol utolérhet életünk utolsó eseménye) a tranzakció ütközések kezelése és a géphiba, szoftver hiba miatti roll-back eljárás igen komplikálttá válhat. Az üzemzavarok kezelésére, az adatvesztések kiküszöbölésére is nagyobb figyelmet kell fordítani, mint egy szokványos méretű rendszernél.

Összegezve a tapasztalatokat úgy találtuk, hogy egy országos számítógépes információs rendszer kifejlesztése a rengeteg specialitása miatt mára már a szakmán belül önálló szakterületté vált.

A VÁLASZTÁSI INFORMÁCIÓS RENDSZER ADATBÁZISAI

Farkas Lajos - Málics Géza
KÖNYV IDOM

A hazai választások számítógépes támogatásában a kezdetek az 1985-ös országgyűlési választásokig nyúlnak vissza. Ekkor végeztük az első kísérletet a gépi szavazat-összesítésre, a SZÜV megyei számítóközpontjainak bevonásával. Az akkori adottságoknak megfelelően lyukkártya, ESZR gépek, batch feldolgozás, telexes adattovábbítás volt az eszköztár, s a központban egy COMMODORE 64-es gép dolgozott. Bármennyire meglepő is mai szemmel, ez a rendszer jól vizsgázott. Igaz, az akkori feladat nem volt bonyolult.

A ma is használatos informatikai megoldások első, időtállónak bizonyult változata 1989-ben, Zalaegerszegen jelent meg. A ZALASZÁM-mal együtt készítettük el az állampárti parlament utolsó időközi választását támogató választókerületi számítógépes rendszert, PC-kre építve. Talán van, aki emlékezik rá, itt született meg az első ellenzéki győzelem. A szavazatösszesítést az újságírókon kívül többek között az USA nagykövetsége is a helyszínen követte. Ekkor szembesültünk azzal, hogy milyen fontos a gyors, pontos, részletes tájékoztatás. Szerencsénkre számítottunk az újszerű igényekre is, ezért a rendszer sikeresen vizsgázott.

A PC-ken dBASE-es adatbázis volt, már az adatmodellezés irányába mutató megoldással. A PC-k szakosított feladatokat láttak el, adatbevitelt, összesítést, tájékoztatást végeztek. A több gépen lévő adatbázisok szinkronjának problémájával már itt találkoztunk, amire csak hálózatba kötött gépekkel lehet optimális megoldást kialakítani. Az adatkapcsolatot a gépek között floppy és hálózat egyaránt szolgálta.

Az 1989-es emlékezetes "négy igenes" népszavazásnál már országos PC-s rendszer dolgozott. A 20 területi rendszerben a rögzítő PC-k mellett külön összekapcsolt összesítő és kommunikációs PC-k dolgoztak. A hálózat bérelt telefon vonalakat használó BBS szoftverrel ment. A központi PC-ket NOVELL háló kötötte össze.

A népszavazásnál bevált megoldás kiterjesztésével készültünk az 1990-es parlamenti választásokra. Az eredmény szállóigévé vált: "NINCS ADAT". A közvélemény és sajnos a szakma nagyobb része is úgy értelmezte, hogy ez a hazai számítástechnika kudarca volt. Azonban már akkor is akadt néhány kolléga, aki rájött, hogy valójában egészen más történt. A sikertelenség valódi oka megtalálható a választásokat megelőző két hónap sajtójában, amiről nem érezzük kompetensnek magunkat bármilyen értékelést adni. Az első választási fordulóban a sikeres működést egyetlen közvetlen tényező akadályozta meg: a rendkívül feszített ütemű fejlesztést a finisben, a választás előtt két hónappal **leállították**. A fejlesztést végül kormányfői döntésre mégis folytatni kellett. A kiesett több mint három hét súlyos és kivédhetetlen hátrányba hozta a fejlesztőket.

Maga az összesítő rendszer a választás napjára jó minőségben elkészült, de erre az alkalomra felépített országos TAF hálózat és a TV-s tájékoztatás első teljes próbája a katasztrofális időhiány miatt már maga a választás éjszakai éles üzem volt. Csodák meg nincsenek.

A fentiek alátámasztása helyett csupán két "ellenőrző kérdést" említünk meg, s mindenkire magára bízzuk a választ:

- egy rossz programrendszerrel egy napos késéssel ugyan, de az utolsó jegyzőkönyv beérkezése után egy-két órával el lehet-e készülni 22.000 jegyzőkönyv teljesen (!) hibátlan feldolgozásával?
- egy rosszul tervezett, elrontott rendszer a második fordulóban képes lett volna-e olyan sebességgel ontani a pontos adatokat, hogy a TV riporterek alig győzték elemezni és ismertetni, s éjjel egy órára már a végeredmény is megszületett?

Tiszta lelkiismerettel állíthatjuk, hogy a szakmánknak a történtek miatt nincs szégyenkezni valója.

Maga a rendszer tipikus PC-s technológiájú megoldás volt. A 148 egyéni választókerületi székhely települést, a 20 megyét, a parlamenti központot és a TV-t telefonvonalas, BBS-el dolgozó hálózat kötötte össze. A központban dupla NOVELL LAN épült ki. Az egyes munkamozzanatokat külön-külön PC-k végezték kezelői beavatkozásra, a fájl szerver gépeket elérve.

A rendszerszervezést az ÁNH végezte. A programrendszert és a dBASE adatbázist a CONTROLL Kisszövetkezet munkatársai *mintaszerűen* tervezték meg

s dokumentálták. Az adatbázis harmadik normál formára hozva, a relációs adatkezelés elvei szerint épült fel. Az illetéktelen behatolás, hamisítás ellen számos eszköz és csapda épült be a rendszerbe, a politikai szituáció miatt több is, mint ami szakmailag indokolt volt. Az egész rendszer gondos munka volt.

Az ÁNH-s csapat a történtek után 1993-ban, a társadalombiztosítási önkormányzati választások alkalmával kapcsolódott be ismét a szavazatösszesítési feladatok ellátásába. Az informatikai megoldások az 1989 óta máig konzekvensen követett és fejlesztett irányvonalat tükrözték, míg a számítástechnikában lényeges fordulat állt be. A megyék és központ között állandósult a TAF kapcsolat, a megyékben és a központban DEC VAX gépek vették át a feldolgozást. A PC-k adatbeviteli és tájékoztatási munkákat végeztek.

A rendszer belső felépítésén ekkor még jelentősen meglátszott, hogy a hazai fejlesztők igen nehezen tudnak elszakadni a PC-s hagyományoktól, s egy normalizált modell nagy ellenkezést vált ki. A kliens-szerver megoldás is idegen. Az ellenérvek: bonyolult, felesleges, lassú, egyszerűbb és könnyebb hagyományos módon bütykölni. Egy kis teljesítményű PC esetében ebben volt is valami.

Az 1994-es országgyűlési képviselő választásoknál már kiszélesedett a gépesített feladatok köre. Ekkorra már 17 különböző számítógépes rendszer dolgozott, részben közös, részben saját adatbázissal, a névjegyzék készítéstől a jelöltek adatainak összegyűjtésén keresztül a különböző összesítő és tájékoztató, nyomdai,

költségvetési rendszerekig. Külön rendszer készült a rendszerek tesztelésére, minőség ellenőrzésére.

A rendszer központjában a jelölt, lista állítást regisztráló és a választás éjszakáján működő összesítő rész adatbázisa állt. Érdekessége a dolognak, hogy ez az adatbázis induló állapotát egy másik, népességnyilvántartási adatbázisból vette át (választási körzetek), s maga is számos más adatbázisnak (pl. a végleges szavazatösszesítést támogató rendszernek) adott át adatokat. Ezért egymással összefüggő adatbázisok láncolata jött létre.

Magát a szavazatösszesítést végző adatfeldolgozó rendszer 3+1 szintű volt:

- a központi rendszer, a Duna palotai , TV-s és a párt tájékoztató rendszerekkel kiegészítve,
- a 20 területi rendszer,
- a 148 egyéni választókerületi (OEVK) székhely település rendszerei

hálózatba kötve, utóbbiak kapcsolt vonalon. Ezt egészítették ki opcionálisan a helyi települési rögzítő rendszerek, melyekkel floppy-s kapcsolat volt. Az OEVK székhely településekre IBM RS/6000-es gépek kerültek, melyek a helyi LAN-ba kötve fájl szerverként dolgoztak.

A megyei szint alatti adatbázisok induló állapotát a területi adatbázisból egy generáló rendszer állította elő, a választás napját közvetlenül megelőzően. Az előkészítési szakaszban csak a központi és a területi adatbázisok éltek. A központi és a területi VAX gépeken ekkor használtunk először relációs adatbázisokat, a DEC RDB rendszerével.

A legnagyobb problémát két összefüggő dolog, a négy szintű osztott adatbázis szinkronja és a kommunikációs kapcsolatok hibatűrő megoldása jelentette. A feladat jellegéből adódóan minden körülmények között működőképes megoldást kellett kialakítani, ezért a TAF mellett adathordozó szállításos és faxos adattovábbításra is fel kellett készülni. Faxon, floppyn meg a kétfázisú commitment control elég ritkán működik...

A kor színvonalának közelébe az 1994 decemberi önkormányzati és kisebbségi választások rendszerével jutottunk. Magyarországon ez a legbonyolultabb választási feladat, ami a rendszeren és az adatbázisokon egyaránt tükröződött.

Az adatfeldolgozó rendszer struktúrája lényegében azonos volt az országgyűlési választásoknál kialakítottal. Ez esetben is több viszonylag önálló és egy nagyobb, a jelöltállítást és a szavazatösszesítést kiszolgáló adatbázis láncolatát hoztuk létre A megoldás gerincét adó rendszerek valamennyi adatbázisa egy közös, harmadik normál formájú logikai adatmodell teljes illetve a szinttől függő részleges megvalósításával állt elő.

Az említett közös modell mintegy 50 relációs táblából állt. A relációs táblákat négy csoportba foglaltuk:

- a jelölő szervezetek és a jelöltek adatai,
- a választókerületi struktúra,
- az indított jelöltek, listák adatai,
- a szavazóköri és választókerületi eredmények.

A megyei és területi adatbázisok között adatbázis triggerek teremtették meg az állandó szinkront. A központi feldolgozó rendszer tiszta SQL alapú volt, s az adatbázis integritását ökölszabályokkal(constraint) is védtük.

A központban a felhasználókkal olyan kliens PC-k tartották a kapcsolatot, melyekről levettük a billentyűzetet. A felhasználók Visual Basic-es, grafikus felülettel s egyetlen egérrel találkoztak. Ez a megoldás komoly sikert aratott, mivel még a számítógéptől idegenkedők is maguktól, külön mini-oktatás nélkül azonnal meglehetősen összetett lekérdezéseket tudtak indítani. A felhasználóknak kizárólag „választásul" kellett némileg tudniuk, még úgy sem nagyon. A kliens PC-k ODBC-n keresztül érték el a szerver VAX-okat.

A választási adatok komplex lekérdezése volt az első igazi nehézség. A közel 400 ezer sorból álló adatbázisra rá lehet indítani egy olyan egyszerű kérdést, mint pl. „hogyan állnak a pártok?". Ennek a megválaszolása az eredetileg rendelkezésre álló VAX-on az elfogadható egy-két másodperc helyett több mint negyedórát tartott volna, ha a többi felhasználó addig nem kérdez semmit. A PC-k előtt viszont - tapasztaltuk - egymást lökdösik az újságírók és más érdeklődők, mindegyik ezer dologra kíváncsi. Közben a TV műsorát is pillanatok alatt ki kell szolgálni, bármilyen adatot kérnek is.

Ezért készítettünk egy „kivonat" adatbázist, amely az összes, előre elképzelhető kérdésre előregyártott formában tartalmazta a válaszok adatait. Ebből a tájékoztató PC-k már azonnal tudtak válaszolni, s a TV-ben lévő grafikus rendszer is át tudta szívni az adatokat.

Egy probléma azért maradt. A kivonat adatbázist időnként, optimálisan 10 percenként frissíteni kell, közben fogadni kell a területről érkező adatokat, s el kell végezni a központi feldolgozást. Ehhez egy VAX 7000-es gépet kellett bérelnünk, hogy a megfelelő számítási teljesítmény meglegyen. Ez sem volt még elegendő, mivel egy újabb, aktuális eredmény előállítása a lemezműveletek hatalmas mennyisége miatt - sok száz, az adatbázis kisebb-nagyobb részét feldolgozó SQL kell a kivonat adatbázis frissítéséhez - még így is egy órába telt volna, miközben 30-40 kliens PC is állandóan terheli a szervert. Fel kellett tehát bővíteni a VAX memóriáját olyan méretűre, hogy RAM diszket generálva az egész adatbázis beleférjen. Így már kellően gyors volt a rendszer.

Az országban dolgozó több mint 2000 gép (VAX, RS/6000, többféle PC) nagy része hálózatban működött. A központi rendszer a KÖNYV telephelyén dolgozott. A tájékoztatást végző kliens PC-k távol voltak(TV, Duna Palota), ahová a LAN-t 8 Mbit-es optikai kábellel terjesztettük ki. Tartalékként két mikrohullámú csatorna állt készenlétben. Az MTI-vel és az azt igénylő pártok székházaival kapcsolt vonalas összeköttetést építettünk ki.

A rendszer tervezését-szervezését az Országos Választási Munkacsoport tagjaiként végig kézbentartottuk, de a megvalósításban már számos cég dolgozott együtt: a KÖNYV, az IBM, a DIGITAL, a DIREKT, a HBMÖ IK, a MATÁV, a BM Híradástechnikai Szolgálat, a Komárom megyei és a Szolnok megyei TÁKISZ-ok, a PROPIXEL, a ZALASZÁM és még sokan mások. Igazi csapatmunka volt.

A választási adatbázisokkal szerzett tapasztalatok közül megemlítünk néhányat.

Egy ilyen összetett, sok adatbázisú, sok fejlesztős rendszer igen komoly koordinációt igényel. Ez csak úgy oldható meg, ha a tervezés egyetlen műhelyben összpontosul. Különösen fontos a koncepcionális tervezés, a korrekt, normalizált adatmodell. Ha különböző okok miatt nincs mód kész, gyári osztott adatbázis kezelő szoftvert használni, a megoldás igen bonyolulttá tud válni. A kliens-szerver architektúra, a grafikus felhasználói felület, a relációs adatbázis kezelés mindegyike már önmagában igen nagy erőforrást igényel. Ha ezek hatása összeadódik, ráadásul egy nagyon művelet-igényes feladattal találkozik össze, igen komoly erőforrás és hálózati teljesítmény, a korábbiaknak a többszöröse kell az eredményességhez. Az egész rendszer tervezésénél is alaposabban meg kell mozgatni a „szürke állományt". Sajnos valamennyi eddigi választásnál a fejlesztésre elképzelhetetlenül rövid idő állt rendelkezésre, ami a normális fejlesztési technológia betartását lehetetlenné tette.

Ma már az 1998-as választások előkészítésével foglalkozunk. Új IT stratégia kialakításán dolgozunk, amelyben már kibővült, erős hálózat, integrált adatbázis, ORACLE és más, korszerű szoftverek szerepelnek. Elmozdulunk az INTERNET/INTRANET alkalmazások irányába is, WEB szerverek rendszerbe integrálásával. Reméljük, hogy a szükséges fejlesztési időnek most már közel fele rendelkezésünkre fog állni, s ez lenne az igazi előrelépés.

ADATMINŐSÉG-IRÁNYITÁSI RENDSZER KIALAKITÁSA

Dr. Mezey Gyula főtanácsos KÖNYV

1. Népességnyilvántartás pontossága, időszerűsége, teljessége

Aki azt igéri, hogy a nyilvántartása teljes, pontos, időszerű, az a lehetetlent igéri. Könnyen lehet, hogy a ténylegesen történt esemény, és az arról begyűjtött adat -- két teljesen különböző dolog. De ha ettől el is tekintenénk, akkor is a tudomány kénytelen felismerni, hogy minden jelenség leirása véges pontosságú, sőt az instabil rendszereket éppen a pontatlanság jellemzi. Ha az adatok egy teljes körére várunk, a már begyűjtöttek egy része válik időszerűtlenné. Ha a begyűjtött adatokat sokkal hosszabb ideig kezeljük (történeti és képi adatbázisok), akkor a sokkal nagyobb adatvolumen túlterheli az adatfeldolgozó és elemző kapacitásainkat, és gazdaságtalanná teszi az adatkezelést. Ha ennek terheit végtére a közigazgatási szolgáltatásainkat igénylő polgárokra háritjuk, politikai elégedetlenséget "aratunk". A jó kompromisszum kialakitását segitik a (D.P.Ballou, H.L.Pazer 1995) által az adatfolyamok, (W.Page, P.Kaomea 1994) által pedig a képfolyamok rendszertervezőjének pontosság-időszerűség kompromisszum döntéseit támogató eljárások. Mindenesetre megfogalmazható (adott körülmények között) egy igazgatási nyilvántartással szemben elvárható pontosság teljesitése. A hasonlóakhoz viszonyitás eljárását alkalmazza az ún. **benchmarking** módszere.

A népességnyilvántartás vonatkozásában bizonyára az egyik alapvetőbb pontatlanság, ha új belépőként nem rögzit, kilépőként, passzivként, nem regisztrál a valóságnak megfelelően, vagyis ha nem minden (vagy éppen több) személyt tart nyilván: a személynyilvántartás nem teljes. Ennek a hibának a súlya azért nagy, mert nem csupán a közigazgatás, hanem a magánszféra számára is a polgár azonositása nélkülözhetetlen. A népesség minden egyes tagjának nyilvántartásbavételére ezért általában mindenütt létrehoztak anyakönyvi nyilvántartást, néhány országban pedig ezen felül még népességnyilvántartást is. Arra nézve, hogy a szervezeti keretek, a társadalmi szerkezet, az infrastrukturális fejlettség milyen mértékben befolyásolják a nyilvántartás pontosságát, alig van közvetlen adatunk.

Rendelkezünk szóbeli információkkal arról, hogy egyazon országon belül az egyik megoldásról a másikra áttérve mennyivel tett ez pontosabb mérést lehetővé. Igy pl. Dániában egy decentralizált, kiforrott, területi (regionális) szinten már jól működő népességnyilvántartásról a központi népességnyilvántartásra átállva, annak a teljessége kb. 2%-al javult. A skandináv országokban, ahol rendkivül magas az állami szektor aránya, magasfoku az állami gondoskodás, a mindezt megalapozó adórendszer követelménye is, hogy a költözködést be kell jelenteni, a népességnyilvántartások ma már igen pontosak, pl. Svédországban a központi személynyilvántartás hibaaránya ezrelék alatti, és még ezt a csekély hibát is az ötévenkénti népszámláláskor feltárva korrigálni tudják.

Olyan országban viszont, ahol az állami szektor aránya kicsi, az adórendszer eltérő, a költözködéseket nem kell bejelenteni (pl. Angliában és Walesben), két népszámlálás között(10 év) egy-egy település lakosságszámának a becslése 2-5%-al tért el az utóbb megszámlált lakosságtól. Az ún. Longitudinal Study tapasztalatai szerint két egymásutáni népszámlálási állomány összekapcsolásakor kb.91% csak a sikeres összekapcsolások aránya. Nagy-Britanniában azonban a világháboru alatt volt kötelező lakcimbejelentés és népességnyilvántartás, ezt vette át később az egészségügy, s ennek pontosságát ma 2-5%-ra teszik, jelenleg pedig van országos, településenként vezetett helyiadó nyilvántartás és külön választási lajstrom. Aligha tévedünk sokat, ha azt feltételezzük, hogy népességnyilvántartás nélkül, csupán népszámlálásokkal a lakosságszám mérési pontossága 5-9% körüli lehet, a helyi és területi szint működésére épitkező regionális népességnyilvántartás esetén ez a pontosság 2%-hoz állhat közel, mig központi népességnyilvántartásnál 1% alatti pontosság

várható. Interpolációval arra következtethetünk, hogy egy csupán települési szinten működő népességnyilvántartás talán kb. 4% -os pontosságú (teljességű) lehet nyugateurópai viszonyok között. Mindamellett a statisztikusok egy része úgy véli, hogy (Ph.Redfern 1989) céljaira a népszámlálás amugyis szükséges, illetve alkalmasabb, mint a népességnyilvántartás. A lakosságszámlálás olyan pontosítási módját, mint a népességnyilvántartással kombinált periódikus statisztikai adatfelvétel, viszont az adatvédelmi jogszabályok tiltják.
A népességnyilvántartásra főleg egy-egy konkrét személy pontos azonosítása érdekében, egyrészt a hatósági munkához, bűnüldözéshez, másrészt különféle szolgáltatások igénybevételekor a jogosultság igazolásához **operative** van szükség. Habár az általános hadkötelezettségen alapuló honvédelem, a bűnüldözés, és az adóhatósági munka hatékony végrehajtását bizonyára elősegiti az, hogy az állam pl. adóalanyait egyértelműen és egyedileg azonositani tudja, nincsen elegendő közvetlen adatunk arra, hogy pl. az adócsalások megelőzésében mennyire hatékony egy központi, egy regionális, vagy egy települési népességnyilvántartás. A honvédelem átalakulása, a települések régiók növekvő politikai sulyának az adórendszerben való tükröződése, a jóléti államtól való távolodás megrázzák a központi népességnyilvántartás eresztékeit, az adatvédelem korlátozza mozgásterét és a személyi számot, átfogalmazza az adatminőség fogalmát, a biztonságos információrendszer irányába állitja a közigazgatási információrendszerek fejlesztési céljait. Felmerül a kérdés, hogy ha az országos, kötelező személyi szám --vagy akár utódai, az ágazati személyazonositók-- a központi népességnyilvántartáshoz kötődő jelenség, akkor a történelmi fejlődés milyen más személyazonositási megoldások felé tereli a közigazgatást (és a magánigazgatást), hiszen az EU-ban a magyar népességnyilvántartás semmiképpen sem lesz központi(a többi ágazati nyilvántartás mégkevésbé). Megjegyezzük, hogy pl. Svédországban is ma már a népességnyilvántartás elosztott és történeti adatbázisait megyei szintre telepitik, a központ szerepe csökkenőben van, döntően a személyi szám központi kiosztása marad rá. Ha ágazati, vagy még annál is szűkebb (csak egy-egy szervezeten belüli) hatókörü személyi azonositókkal, ugyanakkor szorosabb nemzetközi szálakkal (európai rendőrség és közigazgatás? nélkül is) kell számolni a jövőben, akkor milyen megoldások jöhetnek szóba ?
Jelenleg a népességnyilvántartásnál az OEP és az APEH igazolványai és igazgatási azonositói kapcsán nagyszabásu adategyeztetés és hibajavitás is történik. Az eredmények részletezése nélkül is három dolgot meg lehet állapitani:
-- a természetes személyazonositó adatoknak nagyobb szerepe lesz a jövőben
-- nagyobb hitelességre és pontosabb személyazonositásra kell törekedni
-- minőségirányitási rendszert kell kialakitani

1.1. Adaptálható minőségirányitási modell

Terjedelmi okokból most csak erre az utóbbi vonatkozásra kell szoritkoznunk. A minőségirányitási rendszer kiindulópontja az "ügyfelek" és a "beszállítók" tisztázása. Némely esetben az ügyfelek és a beszállítók köre átfedi egymást. A közigazgatás tipikusan olyan terület, ahol az ügyfelek közszolgáltatást kapnak egy közigazgatási szervtől, amely számára azonban maguk is szolgáltatnak, mégpedig legalább két dolgot : elintézendő ügyeket (feladatokat), valamint azok elintézéséhez szükséges információt, és ecélból adattermékeket "gyártanak le"(kitöltött alapbizonylat), adnak át a közigazgatási szerv iratgyüjtő rendszerének. Milyen **modellre** támaszkodhatunk a minőségirányitási rendszer megtervezésénél ? Az ún. minőségkör irányitását ipari gyártmányokra egy 20 szakaszból álló modellel irja le az ISO 9000-9004, de az adattermékekre ezt a modellt nem lehet egyszerüen ráhúzni. (Y.R.Wang és tsai 1995) kialakitottak egy 7 szakaszra tagolt adatminőség-modellt, aminek a középpontjában adattermék rendszerfejlesztése--üzemelés--karbantartása 3 szakasz áll. Mi úgy véljük, hogy az adatgyüjtés ("adattermék-gyártás" és "szétszerelés")--tárolás--szolgáltatás("adattermék-összeszerelés") 3 szakaszát a középpontba állitó modell (lásd Mezey 1995-6) valamivel

jobban tükrözi az információrendszer egésze, az AB-ok, és a szolgáltatásba ágyazott adattermékek közötti viszonyokat, és alkalmasabb lehetne a fentiek egymáshoz viszonyitott minőségének konzisztens index-rendszerrel való mérésére.

1.2. Minőségmutatók rendszere

A minőséget irányitó vezető számára az adatminőség-mutatók rendszere hasznos volna. Szabvány, vagy ajánlás erre még nincsen, kutatók ugyan kidolgoztak az információrendszer egészének minőségére vonatkozó sokváltozós függvényeket, de ezek alkalmazhatóságáról nincs egyetértés. A felhasználói vélemények felmérései alapján megállapitható volt, hogy a legfontosabb minőségparaméterek a pontosság, teljesség, időszerűség, megbizhatóság.
Az **információrendszer megbizhatóságával** kapcsolatban kutatók pl. az alábbi három mértéket dolgozták ki :

- üzemi megbizhatóság,
- relativ megbizhatóság (a felhasználó elvárásaihoz képest),
- abszolut megbizhatóság (a valósággnak megfelelés).

Mások pl. az **adattermék minőségét** is azzal mérik, hogy egy alkalmazás számára az adatbázis :

- milyen megbizható adatokat tárol,
- mekkora az adatbázis ún. szemantikus integritásának (tartalmi sértetlenségének) és
- az alkalmazásnak az előirásokat hűen követő "megvalósitási pontossága".

Adattermékek minőségét gyakran az előállitásuk alapjául szolgáló **adatbázis integritásának biztosítására** vezetik vissza. A szemantikai integritást elemző statisztikai eljárásokat is dolgoztak ki. A kutatások elsősorban a pontosságot tekintették minőségmeghatározónak, amit előbb a **szintaktikus(alaki) és szemantikus korrektség** fogalmára egyszerüsitettek, viszont később a szemantikus integritás, **séma integritás, stb. korlátai** fogalommal még bővitették. Azonban mások kimutatták, hogy a szemantikus integritás-korlátokkal nem is lehet minden pontossági követelményt egységesen meghatározni. Mivel a teljesség és időszerűség mellett még számos tényező befolyásolja az adatminőséget (pl.T.C.Redman 1992, az adatminőség 20 összetevőjét különbözteti meg) , célszerünek látszik más eljárásokat alkalmazni. Az adatbázisok fogalmi tervezését minőségszempontból kiegészitő eljárásokra ötleteket mutatnak be (és számos kérdést hagynak nyitva) a legújabb kutatások.
Kiséreljük meg ezért legalább néhány fogalom tisztázását és csoportositását. Miben különbözik információrendszer és nyilvántartás, illetve nyilvántartás és adattermék, "gyártott" adattermék és "szerelt" adattermék ? Mi köze a szoftver minőségnek az adatminőséghez, vagy a hálózat minőségéhez ? Hogyan lehet biztositani, hogy a minőségmutatók konzisztens rendszert alkossanak ? Melyek a releváns minőségparaméterek ? A fentiekre adandó válaszhoz először egy absztrakt modellre van szükségünk. Felfogható egy szervezet információs rendszere úgy mint egy csatorna. A falát hardver, átviteli, és szoftver-eszközök, a működtető emberi erőforrás és az azt irányító szabályzatok alkotják. Belül (sohasem adatok, hanem mindig --) adattermékek áramlanak. Ezen az alapon az adatminőséget jellemző paraméterek világosan elválaszthatók pl. a szoftver minőségét jellemző paraméterektől. A csatorna csak látszik madártávlatból egyetlen csatornának, közelről alcsatornák alkotta hálózat. Célszerü lehet az alcsatornákat kategorizálni a bennük áramló adattermékek adathordozója szerint (papirfolyam, adatfolyam, felvételezett kép-folyam, stb.). Hasonlósági módszer alkalmazásával e folyamokra adaptálhatjuk a Fényes Imre által megalapozott általános transzportelmélet idevágó elemeit. E matematikai modell bármely fizikai, kémiai, műszaki folyamatra alkalmazható, amennyiben energiaáramlás mellett legalább még egy ún. extenziv mennyiség áramlik. Kivétel a hőáramlás, ahol csak az energia áramlik, itt azonban (az elmélet szimmetriáját megőrzendő fogalomnak) az ún. entrópia áramlását tételezik. Az információrendszer is műszaki folyamat, jellemzi entrópia. Ahogyan az általános transzportelméletet nem-műszaki folyamatokra (pl. pénzfolyamok) már

megkísérelték alkalmazni, információrendszerre is adaptálhatónak látszik, legalább abban a tekintetben, hogy megmagyarázza, miért ilyen és nem másféle minőség-paramétereket mérünk, minőségmutatókat vezetünk be . Az általános transzportelmélet adaptálása információrendszer adatminősége vonatkozásában elméleti alapozó kutatásokat igényelne, hiszen tisztázni kell az ún. extenziv mennyiségeket, az ún. mérlegegyenleteket, vezetési tényezőket, forrássürüségeket, valamint az egyértelmüségi feltételeket. Ugyanakkor ezt az ISO 9000-el összhangban tehetnénk De -- ma még a leggyakoribb paraméter a **pontosság** sincs általánosan elfogadott módon definiálva, bár többnyire a **valóságnak megfelelő adatot** értenek alatta. Adathibák súlyos (mellékes) voltának értékelése pedig viszonylagos, "attól függ", helyesebben szervezetfüggő.
A szervezet egészének céljai és rendeltetése szempontjából kell súlyokat, prioritásokat minden egyes hibatipushoz külön rendelni azért, hogy a vezetői intézkedéseket az éppen legfontosabb hibafajták megszüntetésére lehessen koncentrálni. A CobiT (az ISACF módszertana) az IT erőforrásait 5, folyamatait 32 kategóriába csoportositja, és rögziti az ezek közötti tipikus kapcsolatokat, majd a 32 folyamathoz összesen 280 kivánatos ellenőrzési célt állit fel (általános esetet feltételezve). Egy konkrét helyzetben ezekből lehet leválogatni azokat, amelyek egy specifikus esetben éppen szükségesek (a vezetés, az ügyfelek, és a belső ellenőrzés számára). Az átfogó modellben megfogalmazott összefüggések alapján ebből már kiadódnak az 5 IT-erőforrással való gazdálkodásra, az éppen oda javasolható ellenőrzési megoldások kiválasztására mutató teendők, illetve a SAC-ban is részletezett konkrét megoldások. Hogyan juthatunk el eddig a "leválogatásig" ? Az, hogy a 280 ellenőrzési célból adott helyzetben melyik releváns, **attól függ,** hogy az egyes **ellenőrzési célok nem-teljesülése mekkora kockázattal jár a szervezet egészének rendeltetésszerü működése** szempontjából. Tehát explicite **ismertnek** kell már **előzetesen** lenni a szervezeti **célhierarchiának** és a fenti **kockázatok becsült értékeinek.** Feltehető, hogy ezek az adatok abban a szervezetben, ahol az Olvasó dolgozik, még egyáltalán nincsenek kimunkálva, előzetes elemzéseket nem végeztek és gyorsabban kell tenni valamit a minőségért, mint ahogy egy ilyen alapozó munka és elemzés elkészül. Mit lehet tenni ebben a helyzetben ?
Összegezve azt állapithatjuk meg, hogy adattermékek, nyilvántartások, információrendszerek minőségének mérésére kutatások folynak, de sem hazai, sem EU, sem nemzetközi szabvány erre nincs. De még a közigazgatási szolgáltatások minőségének mérésére sem létezik olyan adaptálható ajánlás, vagy szabvány, mint amilyen a távközlés (CCITT), vagy a szoftver minőségi mutatói tekintetében már létezik. Nehogy elkezdjük ezek kidolgozását, mert az nem térülne meg. Ha az EU szintjén lesznek majd ezekre szabványok, akkor azt úgyis át kell venni. A hazai Adatvédelmi tv. által bevezetett jellemzők, mint pl. tisztességes adatfelvétel, stb., mérhető paraméterekre, minőségmutatókra való lebontása ugyancsak várat még magára. Ezért mi az Önök helyében ma még csak néhány olyan egyszerü, üzemi szinten egyértelmü, az ISO 9000-nek nem ellentmondó, és magától értetődő mutatót képeznénk, mint amelyeket (Adatvédelmi tv.-ben rögzitett szempontokhoz is igazodva) a példa kedvéért alább bemutatunk.

1.3. Javaslat üzemi szintü adatminőség-mutatókra

Feltesszük, hogy az adatminőséget az alábbi dimenziókban biztosan mérni kell (annak magyarázatát, hogy miért éppen ezekben, az Adatvédelmi tv.7.§-a adja):

- Teljességi vetület (szállítás, tárolás, működés),
- Helyességi (megbízhatóság és pontosság) vetület,
- Időszerüségi vetület (átlagos átfutási idő, késések).

A teljességi és a helyességi vetületet hibaaránnyal mérjük.

1.3.1. A darabszám-szerinti teljességi vetület tényezői:

Iratteljesség: Követelmény az, hogy irat el ne vesszen a papírfolyamban.
Adatteljesség: Követelmény az, hogy az iratról rögzített adatrekord el ne vesszen az adatfolyamban.
Képteljesség: Követelmény az, hogy az iratról felvételezett kép (image) el ne vesszen a képfolyamban.
Azt, hogy miért éppen ezt a 3 tényezőt választottuk, az adat- és iratgyűjtési alrendszer (ezt tekintjük az információrendszer adatminősége szempontjából főfolyamatnak) 3 jelentősebb alcsatornára szétbontása indokolja.
Ha több alcsatornát (pl. hang, mikrofilm, kártya, stb.) definiálhatunk egy konkrét helyzetben, akkor értelemszerűen ugyanannyi újabb mutatót is számításba vehetnénk.

Az iratteljességi arány: $I_t = \frac{\text{elveszett iratok száma}}{\text{összes iratok száma}}$

Az adatteljességi arány: $A_t = \frac{\text{elveszett adatrekordok száma}}{\text{összes adatrekord}}$

A képteljességi arány: $K_t = \frac{\text{elveszett képrekordok száma}}{\text{összes képrekordok száma}}$

Mindhárom fenti mértéket azonos időszakaszra vonatkoztatva az abban az időszakban érvényes teljességi arányszámok eredője adja az adatminőség teljességi összetevőjét.
Amennyiben multimédia iratok gyűjtése kapcsán más (pl. audio, mozgókép, mikrofilm, stb.) állományokat is tartalmazna az adattár, akkor a fenti hibaarányokhoz hasonlóak rájuk vonatkoztatva is képezhetőek lennének.

1.3. 2. Az információtartalom-helyességi vetület tényezői:

Irathelyesség: Követelmény az, hogy az irat ne hamisítvány, vagy utánzat legyen.
Adathelyesség: Követelmény az, hogy a papíriratról mágneses adathordozóra rögzített rekord adatmezőinek tartalma feleljen meg a papírirat megfelelő rovatai tartalmának, valamint az alapnyilvántartásnál működő input-ellenőrzés által képviselt feltételeknek (amelyek a változás-jelentésnek a valóságtól, pontosabban az előzőleg több oldalról is megfigyelt eseményektől való eltérései kimutatását célozzák).
Képhelyesség: Követelmény, hogy a papíriratról való képfelvételezéstől (vagy képlemezre történő átmásolástól) kezdve az iratkép, (fénykép, rajz, térkép, stb.) információtartalmából ne veszítsen, így minősége ne romoljon.

Az irathelyességi arány: $I_h = \frac{\text{ellenőrzésen fennakadt hamis iratok}}{\text{összes kibocsájtott iratok}}$

Az adathelyességi arány: $A_h = \frac{\text{ellenőrzésen fennakadt rögzített rekordok}}{\text{összes rögzített rekordok}}$

A képhelyességi arány: $K_h = \frac{\text{lecsökkent minőségű képek(képromlás)}}{\text{összes felvételezett képek}}$

1.3.3. Az időszerűségi vetület tényezői:

Iratátfutási idő: a papírfolyam egy iratának átlagos átfutási ideje

Adatátfutási idő: az adatfolyam egy input tételének átlagos átfutási ideje

Képátfutási idő: a képfolyam egy image-ének átlagos átfutási ideje

Az átfutási idő értelmezhető a munkafolyam, vagy valamely részfolyam(papír, kép, adat) teljes hosszára, vagy annak csupán egy szakaszára is.
A hibaarányokat periodikusan (pl.karbantartási ciklusonként) decentrumonként, azon belül minden egyes beszállitóra, minden alapbizonylattipusra (illetve eseménytipusra), azon belül minőségmutatónként külön-külön összesitjük és a vezetői értékeléshez szükséges összetett mutatók képzéséhez felhasználjuk. Természetesen valamennyi mutató-átlagérték mellett statisztikai jellemzőket (pl. szórás, stb.) is képezni és elemezni szükséges.

2. Az adatminőség szempontjából kritikus szakaszok

2.1. Az adattermékek beszállitóival való kooperáció minőségorientált fejlesztése

Először is a piaci és érdekviszonyokat kell tisztázni. A minőség ettől is függ. Például az állami népességnyilvántartás más szolgáltatásokat nyújtó igazgatási szerveket szolgál ki, ugyanakkor azonban közvetlen szolgáltatásokat is teljesít ügyfelei irányába. Ma egy országos közigazgatási alapnyilvántartásnak a profiljába eső szolgáltatások tekintetében egyrészt tipikusan monopolhelyzete van, és meglehet, hogy ügyfelei, valamint szolgáltatásainak minősége ezt meg is szenvedik. Másrészt azonban általában a neki **beszállitók meg vele szemben vannak monopolhelyzetben** és neki a beérkező adattermékek nem kielégítő minősége miatt jelentős, ebből adódó **"felhasználói" vesztesége van**. Az alapnyilvántartás minőségirányításának költségeivel (visszajelzés, hibák kijavítása, stb.) a beszállítók által elkövetett hibák, ellenőrzési hiányosságok "árát" fizeti meg. Az adattermékek közvetlen beszállítói (népességnyilvántartás esetében pl. nagyrészben az önkormányzatok). A népességnyilvántartás elosztott rendszerré továbbfejlesztése tehát kedvező lehet takarékossági szempontból is.
Nem tudjuk pontosan, hogy mekkora a rossz kapott adatminőség miatti többletköltség, de ipari analógiák alapján 2-3% selejt a kibocsájtott végtermék előállítási költségét kb. 20%-al is megnövelheti. Az iparban a kereskedelemből és kooperációból beszerzett gyártmányösszetevők aránya kb. 30%, de **az alapnyilvántartás lényegében valamennyi adata beszállítóktól, kooperációból származik**. Tehát amig az iparban a beszállítóknak, kooperálóknak csak egy kisebb hányada van a vevővel szemben monopolhelyzetben, az alapnyilvántartásnál szinte valamennyi. Könnyü tehát belátni, hogy az **alapnyilvántartás adatminőségét a kapott adattermékek minősége határozza meg döntően** és az alapnyilvántartás részéről ehhez hozzáadott minőség zömmel a saját belső minőségellenőrzése, minőségbiztosítása, esetleg minőségirányítása által teljesített eredmény. Ezek együtt határozzák meg az alapnyilvántartás minőségét.

Természetesen más szervezetek esetében inkább lehet adatpiacról beszélni. Egy műszaki terv mint adattermék saját "gyártású", és kevés kapott elemet tartalmaz. De nem is szokták gyakran adattermékként árulni -- általában több profitot hoz, ha a műszaki gyártmányt értékesitik mint végterméket. Mégis -- megélnek a tervezőintézetek, és egyre több az adattermék is, amelyek minőségét pl. az ISO 9000-vel biztosithatjuk **Piacra** szánható tipikus adattermék az igazgatási folyamatok nyilvántartásaiból **mellékterméként "épithető"** olcsón össze, illetve ezt **érdemes eladni**. Ilyen újra-és-újraképződő potenciális adattermék csaknem minden irodai munkahelyen található a nyilvántartásokban.

Az ISO 9001 előírja, hogy a (be)szállítónak az egyedi hibák kijavításán túl elemeznie kell a hibaokokat és az ismételt előfordulásokat megelőző intézkedéseket kell tennie, ezeket ellenőriznie kell.
Az ISO 9004 ajánlása az, hogy külön feladatként kell kijelölni a korrekciós tevékenység koordinálását, nyilvántartását és megfigyelését a szervezeten belül, azonban a változások számos más feladatkört is érinteni fognak. A hibaokok és hatásaik közötti összefüggések elemzése a folyamat(ok) eljárások, vagy termékek (részbeni) megváltoztatásához, javításához vezet(het)nek el. A hibaokok egymás közti függései gyakran egy alapokra vezethetők vissza (15.5.).
Mit jelent hát az adattermékek beszállítóival való kooperáció minőségorientált fejlesztése? A saját minőségbiztosítási rendszerünknek a beszállítói rendszerekben való messzemenő elismertetését, az adatgyűjtő decentrumokban érvényesítését.
Az ISO 9001 a beszállítókkal, kooperációs partnerekkel kapcsolatban kétféle minősítést különböztet meg. Az egyik maguknak a beszállítóknak a minősítése (4.6.4.), a másik a beszállított termék minősítése.

2.2. A beszállítók minősítése

Még akkor is, ha egy beszállító az alapnyilvántartás számára az illető közigazgatási szerv irányából "monopolizált" szállító, az adatokat beszállító szervek minősítése csak **haszonnal** jár. Akár azért, mert az ISO 9004 ajánlását követve (9.4.) a beszállítóval minőségbiztosítási megállapodást kötve, -- abban az alapnyilvántartás által előírt kötött ügyrendü minőségbiztosítási rendszert rögzítve --, ott az adott beszállító tipikus hibáira irányítható a figyelem. Akár azért, mert ha a beszállító éppen az alapnyilvántartás decentrumaként és módszertani irányítása alatt működik, akkor az oktatás és a "gyártásközi ellenőrzés" eljárásai éppen azokban a vonatkozásokban és azokban a decentrumokban kerülnek majd előtérbe, amelyek ott a legtöbb hibát engedték eddig tovább.
Amint arra már utaltunk, az esetek döntő részében a beszállított adattermék olyan, amit később éppen a beszállító fel is kíván (mint "vevő") használni. Az ISO 9001 a vevő által beszállított terméke kezelésére nézve (4.7.) nem tesz érdemi megkülönböztetést, míg az SAQ (Schweizerische Arbeitgemeinschaft für Qualitätförderung) ajánlása szerint ezeket a termékeket ugyanugy kell kezelni, mint a többit, de mégis a vevővel szembeni fokozott információs kötelezettség áll fenn, ha ilyen termék elvész, megsérül, használhatatlanná válik, egyszesmind a vevő tulajdonát képező termék felelős őrzésének rendjét szabályozni kell.
Végezetül a beszállitókat -- ahol lehet -- arra kell ösztönözni, hogy legalább az ISO 9002 előírásainak megfelelő minőségbiztositási rendszert hozzanak létre és ezt független szervezettel tanusittassák.
A beszállitók minősitésére például a szakirodalomban a (Maas, Brown, Bossert 1990) által ismertetett eljárások alkalmasak.

2.3. A kapott adattermék vizsgálati mélysége beszállítójának minősítése alapján

Minden egyes adatbeszállitót jellemző adathalmaz egy-egy pontot jelöl ki a sokdimenziós térben. Keressük ebben a térben azt az egyenest, amelyre e pontokat vetitve egyértelmü, a valóságot (a pontok egymáshoz való viszonyát) pontosan tükröző beszállitói adatminőségi sorrendet kapunk. Ezzel kapcsolatban több probléma is felmerülhet, igy pl.:
--Csak abban a térrészben elhelyezkedő adatokat láthatjuk, amelyen (adatvédelem) belül hozzáférési jogosultsággal rendelkezünk.
--Kérdés, hogy a tér dimenzióit az igazgatási eseménytipusok, vagy a beszállító szervezetek feszitik-e ki ?

--Létezik-e egyáltalán a feltételezett egyenes ? Feltéve, hogy igen, akkor is a pontoknak az egyenesre vetitésével (esetleg jelentős) információvesztés járhat együtt. A probléma megoldására alkalmazható a többdimenziós skálázás(MDS), amely sokdimenziós térbeli ponthalmazhoz egy kisebb dimenziószámu (itt : egyenes) redukált térben elhelyezkedő azonos elemszámu ponthalmazt rendel egyértelmü megfeleltetéssel.
--Ha a feladaton lazitunk, pl. csak jó és rossz adatbeszállitói kategória megkülönböztetésére van szükség, akkor azt a hipersikot keressük (helyette egydimenziós tér esetén az ún. vágási értéket), amely elválasztja a két kategória elemeit, ezt pedig pl. diszkriminancia-analizissel érhetjük el.

Egy nyilvántartásba összegyüjtött, fokozatosan bővülő tapasztalataink alapján a fentiek végrehajtásával egy adatbeszállitói (jósolt) minőségi valószinüséget rendelhetünk minden egyes beérkező alapbizonylathoz, ami az iratra vonatkoztatva **vezérelheti az adatellenőrzés kiterjedt voltát, mértékét**, alaposságát. Az adatellenörzés végrehajtásakor azokat a szempontokat kell figyelembevenni, amelyek kimutathatóan jellemzőek (relevánsak) az:
-- adatszolgáltatásban résztvevőkre
-- eseménytipusokra.

3. Adatellenőrzési terv

Ezt a tervet a választott technológia (itt: nagyvonalú rendszerterv) ismeretében lehet kialakítani, hiszen az adatellenőrzéshez alkalmas ellenőrzési módszerek, eszközök és pontosság, a mérőeszközök áteresztőképessége és a kezelők hatásköre a rendszertervhez egyrészt igazodik, másrészt a javasolt rendszerterv megváltoztatásához is vezethet. A meglevő (vagy a tervezett) **adatellenőrzés** hatását mérő(szimuláló) és elemző operációkutatási modelleket (D.P.Ballou , H.L.Pazer 1985) fejlesztettek ki. Ezeket továbbfejlesztették (D.P.Ballou és társai 1993) az **adattermékek minőségét, időszerüségét, és költségét mérő** rendszerré, amely a lényeges minőségparamétereket az adattermék-összeszerelés folyamatának lépésein követi végig (Data manufacturing analysis matrix).
Amikor a rendszertervező kijelöli az **ellenőrzési pontokat**, akkor már részleteiben ismerni kell, hogy a munkafolyamban pontosan milyen objektumok áramlanak. Majd el kell döntenie, hogy ezek adatait és működésvezérlési adataikat **milyen módszerekkel** és **milyen mélységben ellenőrzi**. A tárolás későbbi szakaszában ezeken felül szükséges még rendszeres mintavételes ellenőrzés (információrendszer-auditálás keretében) is.

Az adat- és iratgyüjtési alrendszer kezdeti szakaszában a hagyományosan használatos eljárásokat, mig a végső szakaszában, illetve az adatbáziskarbantartás elvégzése után a tárolás szakaszában egymás után még **öt** további **szürő** felállitását javasoljuk:

-- ún. CAAT-eljárások alkalmazása a kétes tranzakciók automatikus detektálására
-- minőségbiztositási kapu (iratpéldányok darabszám szerinti teljességellenőrzése)
-- ún. credit-scoring elemzés (a kapun átjutott iratok vizsgálatának mélységét vezérli a kapun át nem jutott iratok "beszállitóira" vonatkoztatott statisztikák alapján)
-- szakértői rendszer (a történeti adatbázisok összefüggései alapján ellenőriz)
-- ún. auditáló-rendszer (statisztikai mintavételezéssel)

3.1. Történeti adatbázisok szerepe adathibák feltárásában

A továbbfejlődés egyik fő iránya a számítógépes történeti adatbázisok és az ún. temporális (időben előre is mutató) adatbázisok kialakítása. Ismertek a közigazgatás nagy hagyományu statisztikai vagy pénzügyi **idősorok**at kezelő adatbázisai, ezek azonban általában a rendszeresen végrehajtott ú.n. **szinkronizált adatgyüjtések** adatait tartják nyilván. Ugyanakkor

a jelenlegi számítógépes rendszereink (kevés kivétellel) a nyilvántartott egyedekről, objektumokról mindeddig jószerével csak **a legutolsó ismert állapotukat kifejező** adatrekordjaikat, az ún. statikus, vagy másnéven **aktuális adatbázist** tudták kezelni. **Ennél pedig többre lenne szükség a történeti adatok célszerű hasznosítása és a jobb adatminőség érdekében.**

Ismert tény, hogy minden ellenőrzés ellenére még egy ipari termék mindendarabos MEO-ján is átcsúszik néhány % hibás gyártmány. Nos, rá kell arra mutatni, hogy (függetlenül attól, hogy minden darabot ellenőriznek) **adatterméknél -- az ipartól eltérően** -- a már ellenőrzött adat sem biztosan hibátlan adat és **elvileg is teljesen lehetetlen** a beszállított adattermékekből **a hibák 100%-os kiszűrése**, hiszen a "mérőeszközünkben" sem bizhatunk meg teljesen, mivel a **mérő etalont is adatok alkotják**, amelyeket ugyan hibátlannak fogadtak el, de **utólag** ezekről **bármikor kiderülhet**, hogy mégis **tévesek voltak**. Következésképp hibás lehet bármelyik más, ezen hibás adat (etalon) segítségével már ellenőrzött, és akkor hibátlannak talált, majd eltárolt adat. Rendkívül **nagy** lehet tehát a **hibaterjedés**, sok a **rejtett hiba**, amelyeknek nyomára viszont egy később feltárt hibából "visszagörgetéssel" el lehet jutni, majd javítani. Figyelembe kell venni, hogy mindaddig, amig az adat a nyilvántartásban van (és ez esetleg 100 évet is meghaladhat) érhetik megkésett változásjelzések, javítások, sztornók. Ezeket a **történeti adattárat** felhasználva úgy kell átvezetni, hogy két nézetet is adjon :

- a javítás utáni (esetleg visszamenőleges hatályu változások, javítások) helyzet mai nézetét, és
- egy múltbeli, akkor látott állapotot (az utóbb rárakódott megkésett változások ismerete nélkül).

Az egyedre vonatkozó, régebbi állapotait kifejező adatok hatékony kezelését napjainkig (a nagyon terjedelmes történeti adatállományok miatt) általában nem tudták számítógéppel végezni, hanem egészen hagyományos módon működő okmány- és okirattárak látták el mindmáig. Annyi történt, hogy a mikrográfiát alkalmazták, és a mikrofilmtekercsek , illetve a tekercsen elhelyezkedő egyes felvételek (automatizált) visszakeresésére szolgáló számítógépes programokat irtak. Ma viszont már lehetőségünk van ún. tömegtároló-rendszerek alkalmazására, ami lehetővé teszi nagy digitalizált adatmennyiségek gazdaságos kezelését

A gépi lehetőségek tehát adottak, időszerűvé igy most válik az alapnyilvántartásokra annyira jellemző, **aszinkron** működésü, **eseményvezérelt**, történetiséget hordozó adatbázisok szervezése. Ezeknek a képi adatbázisokhoz szoros közük van két szálon is:

- a képi adatbázis feltöltéséhez a multbanyúló alapbizonylatok, papíriratok rendelkezésre állnak,
- **a képi adatbázis katalógusaként** történeti adatbázisnak célszerű rendelkezésre állni (az aktuális képi adatbázishoz az aktuális AB, mig a történeti képi AB-hoz a történeti AB).

3.2. Képi adatbázisok

Az adatbázisok jövőbeli továbbfejlesztését meghatározó alapvető tényező egyszerűen megfogalmazható:

a digitalizált, adatbázisban kezelt **adattartalom bővülése** minden tekintetben.

Igy például a jelenleg kezelt egyedtipusokra vonatkozó tulajdonságtipusok száma több esetben bővül (pl. geokóddal vagy fényképpel, aláírással, stb.)

Ez azt jelenti, hogy általában szükséges lesz a nyilvántartások, már működő adatfeldolgozó rendszerek ún. strukturált (rekordokat tartalmazó) adatbázisai mellé **képi adatbázisok illesztése** úgy, hogy azok a strukturált adatbázistól **függetlenül is** tudjanak működni, mégse alakuljon ki két egymással konkurráló, párhuzamos adatbázis (strukturált és képi), tehát pl. az adatbázisok karbantartását integráljuk.

Várható, hogy a későbbiekben a képi adatbázisokon kívül (amelyek ma még általában **nem mozgó** képeket tárolnak), egyrészt **mozgó képanyagot**, másrészt **hang**felvételeket tároló **multimédia** adatbázisokat is kell a szöveges adatbázisokhoz **illeszteni.**

Amikor egy már hosszabb ideje működő adatfeldolgozó rendszerhez egy olyan új (jelen esetben: **grafikus**) iratkezelő (DIP) **alrendszert kell illeszteni**, amelynek képi adatbázisa papíriratok digitalizált képeit tárolja, képfelvételre és továbbításra alkalmas eszközöket, adathordozókat, berendezéseket kell beszerezni és a digitalizált grafikus adatok (a továbbiakban : képek) gyűjtésére, átvitelére szolgáló információs alcsatornát nyitni.
Ugyanakkor azonban a gazdaságosság azt követeli, hogy ezt az új iratgyűjtő rendszert a régi adatgyűjtő rendszerrel összeillesszük, integráljuk.
Hasonlóan az **adat- és az iratgyűjtés egybe**: irat-és adatgyűjtő rendszerbe integrálásához, valamint a **strukturált és a képi adatbázis integrálásához**, célszerű a két adatbázisból igényelt információkat kifelé **egyetlen szolgáltatási felület**et képezve célszerű szolgáltatni, azaz a kétféle szolgáltatást nyujtó működést integrálni kell.
Nézzük meg, hogy a **történeti, és a képi adatbázis** integrálása miféle kérdéseket vethet fel :

A mikrofilmes, vagy más analóg(pl.videokazettán, stb.) adathordozón való tároláshoz képest a számítógépes digitális tárolással kapcsolatban ki kell emelni, hogy az okiratok képmásai tárolásának céljára csak az a megoldás felel meg, ha a kitöltött alapbizonylatok papíradathordozón látható alakját annak legapróbb grafikus részleteivel együtt (kézírás, pecsét, stb) teljes hűséggel (hasonmás) másolatban tároljuk. Tehát az iratbevitelkor a bevitel legkisebb egysége az irat, de végig a számítógépes rendszerben kezelés során általában ezt a legkisebb egységet (mint egy faxot) szokták a felhasználók elérni. Az irat egyes kijelölt mezőinek a tár egy másik részére másolása pl. OCR (optikai karakter felismerés) céljából nyílik lehetőség és általában csak a bevitelkor. Törlés vagy módosítás egy iraton belül meg nem engedhető manipulációnak számít. Az irat letárolt képe itt nem kódolt adatokhoz, hanem egy ún. bit-térképhez (bitsorozathoz) kötött, azon egyetlen bit megváltoztatása is már hamisítás lehet. Hamisításra természetesen a legnagyobb elővigyázat mellett is nyílhat lehetőség, erre a biztonsági kérdéseknél visszatérünk.
Számítógépes rendszerben egy irat nem csupán szöveget, számadatokat, hanem vonalas (rajzos) ábrát, képet (szines fénykép, fax) sőt pl. a telefonbeszélgetések bitsorozatait (hangot) is jelenthet. A fenti iratformák egységes kezelését a különféle ún. analóg formájukból egységesre átalakitott "közös" digitális alakjuk teszi lehetővé (multimédia irat).

Ha az irat képét egy ún. iratfeldolgozó (DIP, azaz Document Image Processing) rendszer tároló-alrendszerébe vettük fel (pl. videoval, vagy lapolvasóval scanneltük), akkor a **visszakeresés** pl. háromféle eszközzel történhet :

rekordszerűen strukturált adatoknál ABKR (adatbáziskezelő rendszerrel), vagy szöveges strukturálatlan adatoknál KVR (könyvtári visszakereső rendszerrel), vagy telefoni lekérdezés hanganyagánál KÉR (kérdésre válaszadó --QAS-- rendszerrel).

Ha az alapirat jólstruktúrált nyomtatványűrlap, akkor az ABKR-el ellátott számítógépre, ún. adatbázisszerverre van szükség. Ezesetben a struktúrálatlan (tehát pl. irodai levelezést, ügyintézést segítő iratokat) akkor lehet jól az iratkezelő(DIP) rendszerben visszakeresni, ha pontosan tudjuk az ügyszámot, illetve az iratazonosítót. Ún. tezaurusz (fogalmi kapcsolatok rendszere), deszkriptorok(leiró adatok), stb., itt nem fognak rendelkezésre állni, mint a KVR-nél. Ún. másodlagos kulcsokkal való keresés inditására azonban szintén szükség lehet.
Amennyiben a beszerzésre felkínált iratkezelő (DIP) rendszer adatbázisszervere önmagában nem csupán generikus(elsődleges kulcsról inditott) keresést, hanem másodlagos kulcsokról indított keresést is el tud végezni, az hasznos funkció.
Ha nyomtatványűrlapokat akarunk eltárolni, akkor a tároló-alrendszer lehetne vagy egy:

= ABKR kiterjesztése, ahol struktúrált és struktúrálatlan (kép, fax, stb.) adatokat egyaránt kezelhetünk. Ennek számos előnye van, mert a konkurrencia ellenőrzés, az elérés-ellenőrzések, a lekérdezések optimalizálása készen átvehető, valamint a tervezés adatmodellre épül. Hátrányai, hogy a sok "egymásrahelyezett" szoftver-réteg lerontja a tároló-alrendszer potenciálisan sokkal jobb teljesítőképességét, és ha az ABKR-kiterjesztés nem támogatja ugyanazon adattipusokat, amiket az iratkezelő (DIP) rendszer előír, akkor még kiegészítő szoftverrel kell ellátni,

= ad-hoc file-rendszer létrehozását csupán a kevesebb szoftver-réteg miatti gyorsabb működés indokolná, minden egyéb ellene szól,

= ABKR és ad-hoc megoldások vegyítése lehetséges, ha az ABKR-ünk nem kezelne bizonyos adattipusokat (nagy objektumok: BLOB), de itt nehéz az optimalizálás és összeillesztgetés.

Gyakran az elsőként említett ABKR-kiterjesztésű tároló-alrendszer változat a legelőnyösebb. Elterjedt az, hogy az ún. index-adatbázist (másszóval : logikai AB-t, magyarul : katalógust) kezelő valamely eredetileg általános célú ABKR-t két irányban is kiterjesztett változatban használják:

Majdnem mindig az irat elhelyezési címét az indexadatbázisból elérhető, de fizikailag elkülönült, ún. nyúlványadatbázis tartalmazza. Az elérni kívánt irathalmazból az igényelt iratok részhalmazának tárban való kiválasztásához meg kell adni a kulcsmezőket, azok sorrendjét és az irathalmaz fizikai elhelyezési címét. Ez azt jelenti, hogy az általános célú ABKR-ek egy ún. nyúlványadatbázissal vannak kiegészítve, amely az egyes iratoknak a tárolóban való fizikai elhelyezési címeire mutat.

Némely DIP-rendszernél lehetséges (pl. az SQL AB-kezelő nyelv kiterjesztésével), hogy az ún. adatbázis-adminisztrátor előre helyeket foglaljon le az optikai lemezen úgy, hogy oda majd később iratokat szúrhasson be (optical clustering). Ennek az egyszerírható (WORM) optikai lemezeknél van jelentősége. Az idősorrendben történő (szekvenciális) optikai letárolás rendje felborul ugyan, egy idő után (sok kihasználatlan hely mellett) az előre foglalt helyek is csak betelnek, de addig lehetőség van arra, hogy az eltérő időben beérkezett, viszont logikailag szorosan kapcsolódó iratok is szekvenciálisan elérhetőek legyenek és nem szükséges az érintett, polcon levő WORM-okat sem a juke-box-ra feltenni, cserélgetni. Ez előnyös, hiszen egy irat visszakeresését a mechanikai mozgás rendkivül lelassitja, mig ha ez utóbbi lehetőség fennáll, akkor azt jelenti, hogy az optikai lemezkötet tartalmában nem csupán sorosan kereshetünk, hanem egyféle indexszekvenciálishoz hasonló (Hierarchikus fileszervezési) lehetőségünk is van. Az ún. optical clustering(optikai klaszterálás, azaz iratcsoportositás) enyhíti (ha meg nem is oldja) az iratbevitel túlcsordulási problémáit. Ugyanakkor viszont, hogy a bevitelkor ez a technika ne okozzon jelentős lassulást, a bevitt iratokat mágneslemezen

(esetleg ujrairható ún. MO optikai lemezen) pufferelni (az átviteli sebességkülönbségek kiegyenlitése ún. munkatárolók közbeiktatásával) kell.

Amennyiben az indexadatbázis mérete azt lehetővé teszi, a rendszer teljesítményét megnövelné, ha mágneslemezek helyett ún. RAM diszkeken, (diszk emulátor ahol nincs mechanikai mozgás) tárolhatnánk. Ha pedig az indexfilet nem is RAM (Read Access Memory), hanem akár mindjárt egy ún. REM (recognition memory) fogadná be, akkor a párhuzamos keresések miatt a válaszidő lényegesen tovább csökkenne. Egy irat elérését a tartalmát kifejező adatok segítségével hajthatnánk végre. Ez végső fokon azt jelenti, hogy a tartalma segítségével "címzünk" meg egy iratot.

Amennyiben a gyorsabb működés lehetőségei kiegyenlítik, előtérbe kerülnek a lassan végrehajtható, de a jelenleginél még tömörebb képet szolgáltató kódolási lehetőségek (pl. ún. superimposed coding) amelyek az adatátvitel és adattárolás szűk kapacitásait oldják fel.

A teljesítményt befolyásolja az iratkezelő (DIP) rendszer index-AB-ának elérési módja. Az elsődleges kulcsra működő (pl. személyazonosító jel) két leghatékonyabb elérési mód : vagy az ún. hash-táblák vagy az ún. B-fák. A másodlagos kulcsokkal működő (pl. név, anyja neve, stb.) három leghatékonyabb elérési mód : vagy az ún. file-inverzió, vagy az ún. szignatura fileok, vagy az ún. multiattributumos hash-táblák (de ez meg a file növekedését tűri nehezen). Kisebb adatbázisoknál az inverzió, nagyoknál a szignatúra fileok hatékonyabbak.

A fenti szempontok segítenek a megfelelő iratkezelő (DIP) rendszer kiválasztásában.

Nested adatbázisok implementációs tapasztalatai[1]

Cserges Enikő
(ELTE, Információs Rendszerek Tanszék
Budapest, Múzeum körút 6-8
e-mail: cserges@ullman.inf.elte.hu)

Hajas Csilla
(KLTE, Matematikai és Informatikai Intézet
Debrecen, Pf. 12, 4010
e-mail: hajas@math.klte.hu)

1. Bevezetés

Az előadás célja a nested adatmodellen alapuló, az SQL nyelv általánosításának tekinthető lekérdező nyelv (NSQL) definiálása, és a hozzátartozó interpreter implementációjának bemutatása.

A nested modell lehetővé teszi bonyolultabb szerkezetű objektumok reprezentációját is, lehetőségei azonban ennek ellenére is korlátosak. Az objektum-orientált adatmodellek már sokkal több lehetőséggel rendelkeznek, hiszen itt az objektumorientált tervezési, fejlesztési módszerek összes előnyei megjelennek. Az objektum-orientált adatmodellt tekinthetjük a nested adatmodell kiterjesztésének Ebben a megközelítésben a nested adatmodell egy közbeeső lépés az objektum-orientáltság felé.

Az objektum-orientált adatbáziskezelők implementációjának egyik sarkallatos pontja a bonyolult szerkezetű objektumok tárolása. Erre egy lehetőség az, ha az objektum-orientált adatbázisunkat egy már meglevő adatbáziskezelő fölé építjük. A nested adatmodell lehetőséget nyújt arra, hogy az objektumhierarchiákat nested relációkkal tárolhatóvá tegyük.

2. A nested adatmodell

A beágyazott relációs adatmodell a klasszikus relációs adatmodell olyan kiterjesztése, amely nem-első normálformájú relációkkal foglalkozik, azaz feloldja az értékek atomi jellegére tett megszorítást.

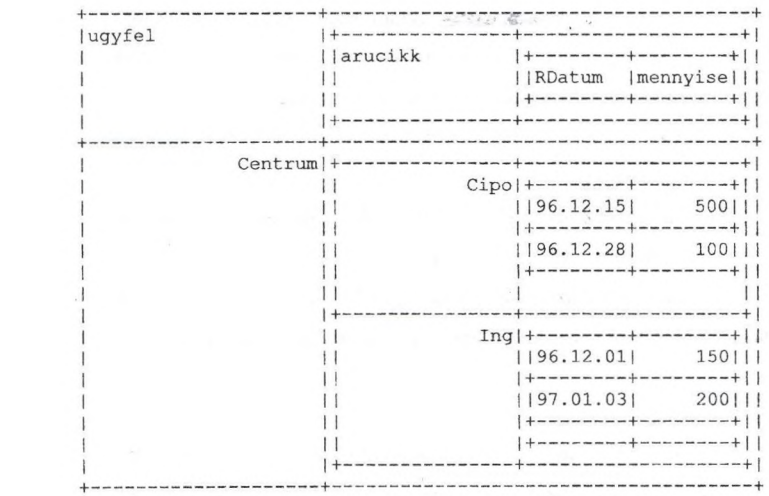
---

[1]Ez a munka az OTKA-T-016-933 pályázat támogatásával készült.

Egy beágyazott reláció sémája attribútumok halmaza, ahol az attributúmoknak kétféle típusát különböztetjük meg. Egy attribútum vagy atomi, vagy összetett (strukturált, reláció-értékű, tábla-értékű). Egy összetett attribútum rendelkezik saját sémával, ami nem más, mint (atomi vagy összetett) attribútumok egy halmaza. Az attribútumok egymásbaágyazása tetszőleges szintű lehet. Az egyszerűbb kezelhetőség érdekében előírjuk, hogy egy relációs sémán belül az attribútumnevek egyediek legyenek.

Egy adott (beágyazott) relációs séma feletti konkrét reláció (reláció előfordulás) sorok egy halmaza, ahol egy sor a sémában előforduló minden attribútumhoz egy megfelelő típusú értéket rendel. Ez azt jelenti, hogy egy atomi attribútumhoz atomi érték tartozik, egy összetett attribútumhoz pedig egy az attribútum sémája feletti reláció előfordulás. Így egy relációba belső relációk lehetnek beágyazva. A beágyazott modell relációit egymásbaágyazott táblákkal szemléltetjük. Az attribútumoknak oszlopok felelnek meg.

Példa: Tegyük fel, hogy egy kereskedelmi vállalatnak van egy nagy raktára, amiről számítógépes nyilvántartást vezet. Tegyük fel továbbá, hogy az adatbázisnak van egy RENDELES nevű táblája, amely azt tartalmazza, hogy az egyes ügyfelek miket rendeltek, valamint hogy a rendelések mikor történtek és milyen mennyiségben.



3. A reprezentáció

Az egyes relációk atomi attribútumait egy (sima) relációba fogjuk össze a megfelelő sorazonosítóval együtt és minden összetett attribútumhoz egy reláció készül. Ez a tárolási

forma alkalmas relációs algebrára való visszavezetésre, elég általános, elfogadható hatékonyságot eredményez különböző körülmények között is.

Ez tehát azt jelenti, hogy minden egyes összetett attribútumhoz felveszünk egy SQL táblát, amibe az atomi attribútumok közvetlenül átkerülnek, az összetett attribútumok helyett pedig egy logikai típusú mezőt veszünk fel, ami azt jelzi, hogy az adott helyen nullérték áll, vagy valódi érték tartozik hozzá. Ez lehetővé teszi, hogy megkülönböztessük azt a két esetet, hogy egy összetett mező nullérték, vagy a beágyazott tábla az üres reláció. Annyi táblánk lesz, ahány összetett attribútumunk volt. Ez csak a reláció sémájától függ, a tábla létrehozásakor eldől.

A sorazonosítót két mezővel oldjuk meg, amelyek minden reprezentáló táblában megtalálhatók. Az egyik az adott sort az egész adatbázisra nézvést egyedien azonosítja. (Az egyedi azonosítót a rendszer biztosítja, egy **ujsor** nevű tábla segítségével, amelyik tartalmazza a legutoljára kiosztott sorazonosító értékét.) Ennek neve **fel**, ezen keresztül lehet egy sorra hivatkozni. A másik neve **le**, és annak a szülőtáblabeli sornak az azonosítóját tartalmazza, amelyikhez az adott sor tartozik. Ennek a mezőnek az értéke legkülső szintű tábláknál érdektelen. Beágyazott tábláknál pedig ez a mező kapcsolja össze az egy beágyazott értékhez tartozó sorokat.

Azonban nem elegendő az adatokat tárolni, szükség van strukturális információkra is, a relációk sémáját bármikor reprodukálni kell tudni. Szükség van tehát meta-adatok tárolására. Ezt egy katalógus táblával oldjuk meg, amelyik minden attribútumhoz egy sort tartalmaz, tárolva az adott attribútum sémában elfoglalt helyét, és típusát.

A katalógus öt oszlopot tartalmaz, ezek:

Ős szülő (CHAR(32)): Annak a (legkülső szintű) beágyazott táblának a neve, amelynek a sémájában az adott attribútum szerepel. Legkülső szintű táblák esetén itt NULL áll. Így különböztetjük meg a legkülső szintű attribútumokat a belső attribútumoktól.

Szülő (CHAR(32)): Az attribútumot közvetlenül tartalmazó attribútum neve. Legkülső szintű táblák esetén NULL.

Attribútum (CHAR(32)): Az attribútum neve. Ez legkülső szintű táblák esetén a tábla neve, beágyazott attribútum esetén az attribútum neve.

Típus (CHAR(32)): Az attribútum típusa. Atomi attribútum esetén itt az attribútum típusának SQL megnevezése, összetett attribútum esetén a reprezentáló tábla neve áll.

Sorszám (INTEGER): Az attribútum sorszáma az őt tartalmazó attribútumban. Ez mondja meg, hogy sorrendben hányadik attribútumként lett definiálva ez az attribútum. Legkülső szintű táblák esetén ez NULL.

Az összetett attribútumokat reprezentáló táblák nevét mindig egy aláhúzásjellel kezdjük, így azok neve nem egyezhet meg egyetlen atomi típus nevével sem. A katalógus neve: __katalogus.

4. Az NF2-SQL nyelv

Jelölések:

< ... >	nem terminális, később kifejtett vagy kifejtendő
... | ...	pontosan az egyik rész helyettesíthető be
[...]	opcionális, elhagyható rész
{ ... }	0, 1 vagy többször ismétlődő rész
NAGYBETŰ	kulcsszó
/* komment */	megjegyzés

A relációs adatbáziskezelő rendszerek funkcióit három csoportba szokás osztani. Az egyik funkció az adatbázis elemek definiálását és megszűntetését szolgálja, ez az adatdefiníciós nyelv (DDL). A másik az adatok módosítását szolgálja, ezt az adatmanipulációs nyelv (DML), a harmadik és talán legfontosabb az adatok lekérdezésére szolgáló lekérdezőnyelv. A bemutatott NSQL nyelv is e három funkciót látja el. A nyelv definiálásánál felhasználtuk [2]-t és [8]-at.

<utasitas>::- <ddl utasitas>; |
<dml utasitas>; |
<dql utasitas>; |
<help utasitas>;

<help utasitas>::- HELP [<tablanev>]

4.1 Az adatdefiníciós nyelv (DDL)

<ddl utasitas>::- <create utasitas>|
<drop utasitas>

CREATE utasítás

<create utasitas>::- CREATE TABLE <tablanev> <tabladefinicio> |
CREATE TABLE <tablanev> [<semadef> AS <lekerdezes>|
<semadef>::- (<attribútum> [<semadef>] [, <sema>])
<tabladefinicio>::- (<oszloplista>)
<oszloplista>::- <attribútum> <tipusmegadas> [,<oszloplista>]
<tablanev>::- /* SQL szabvanyos tabla nev */
<attribútum>::- /* SQL szabvanyos attribútum nev */

Eddig nincs is különbség a beágyazott, és a sima SQL között. Azonban a beágyazott SQL nyelv tartalmaz egy további típust is, a tábla típust, amelyet a TABLE kulcsszó vezet be, és utána egy teljes tábladefiníció következhet. Ezzel lehet a beágyazott táblákat definiálni.

<tipusmegadas>::- INT |
CHAR (<egesz kifejezes>) |
TABLE <tabladefinicio>

Példa: Az előző példában szereplő RENDELÉS tábla létrehozása.

```
CREATE TABLE rendeles
 ( ugyfel CHAR(20),
   rendelesek TABLE
    ( arucikk CHAR(15),
      RTetelek TABLE
       ( RDatum CHAR(8),
         mennyiseg INTEGER)));
```

DROP utasítás

<drop utasitas>::- DROP TABLE <tablanev>

4.2 Adatmanipulációs nyelv (DML)

A beágyazott modellben külön érdekesség a beágyazott táblák módosítása. Egy beágyazott táblába például sorok beszúrása az anyatábla egy (ez esetben összetett) attribútumának a

módosítását jelenti, igy ezt egy UPDATE utasításba ágyazott INSERT utasítással lehet elérni. Ezzel az adatmanipulációs nyelv egy rekurzív szerkezetet kap.

<dml utasitas>::- <insert utasitas> |
<delete utasitas> |
<update utasitas>

INSERT utasítás

Az INSERT utasítás paraméterei a dmlobjektum neve, és a beszúrandók. Azért dmlobjektumot, és nem táblanevet mondunk, mert ugyanez az insert utasítás lesz majd használatos az update utasításnál beágyazott táblák bővítésére is.

<insert utasitas>::- INSERT INTO <dmlobjektum> <beszurando>
<dmlobjektum>::- <tablanev> |<attribútum>
<beszurando>::- [(<attribútumlista>)] VALUES (<sorlista>)
<attribútumlista>::- <attribútum> [,<attribútumlista>] |
<sorlista>::- <sormegadas> [, <sorlista>]
<sormegadas>::- (<erteklista>)
<erteklista>::- <beagyazott kifejezes> [,<erteklista>]

Az NSQL nyelv bővítést a beágyazott kifejezés fogalmánál tartalmaz. Beágyazott kifejezés a megfelelő oszlop típusának megfelelő kifejezés kell legyen. Az egész, és karakterkifejezés ugyanaz, mint a sima SQL-ben, azonban egy bonyolult fogalom az összetett kifejezés. Ez lehet egy táblanév, vagy egy táblakonstans.

<beagyazott kifejezes>::- <egesz kifejezes> |
<karakter kifejezes> |
<tablanev> |
<tablakonstans>
<tablakonstans>::- [(<attribútumlista>)] (<sorlista>) |

A táblakonstans nagyon hasonlít a beszúrandóra, hiszen a funkciója is hasonló.

DELETE utasítás.

Az NSQL DELETE utasítását egy dmlobjektumra alkalmazzuk, mert szerepelhet egy update utasítás belsejében, ilyenkor egy beágyazott táblából töröl értékeket.

<delete utasitas>::- DELETE FROM <dmlobjektum>
[WHERE <logikai kifejezes>]
<logikai kifejezes>::- (<logikai kifejezes>) |
<logikai kifejezes> AND<logikai kifejezes> |
<logikai kifejezes> OR <logikai kifejezes> |

<atomi allitas>

Az atomi állítás az attribútumokra vonatkozó predikátum lehet.

<atomi allitas>::- <oszlop kifejezes> <operator> <oszlop kifejezes> |
< <attr sor> > [NOT] IN <halmazdef>|
[NOT] EXIST <halmazdef>

<attr sor>::- <oszlop kifejezes> [, <attr sor>]

<halmazdef>::- <attribútum>|
(lekerdezes)

<oszlop kifejezes>::- (<oszlop kifejezes>)|
<term>

<term>::- <attribútum>|
<egesz kifejezes>|
<karakter kifejezes>

<operator>::- < |
<= |
> |
>= |
= |
!=

<egesz kifejezes>::- /* SQL szabvanyos egesz konstans */

<karakter kifejezes>::- /* SQL szabvanyos karakter konstans */

UPDATE utasítás

Első paramétere ennek is egy dmlobjektum, hiszen vonatkozhat egy önnálló táblára, vagy egy másik update utasítás belsejében egy beágyazott táblára.

<update utasitas>::- UPDATE <dmlobjektum> <alias>
SET <ertekadas lista>
[WHERE <logikai kifejezes>]

Az értékadás lista egy vesszővel elválasztott felsorolása értékadásoknak. Az értékadások egy attribútumból, egyenlőségjelből, és egy megfelelő típusú kifejezésből állnak. összetett attribútumoknál a kifejezés lehet konstans kifejezés, ahogy azt már az insert utasításnál láttuk, valamint a megfelő attribútumra vonatkozó adatmanipulációs utasítás. llyenkor az a szabály, hogy az adatmanipulációs utasításban szereplő dmlobjektum meg kell egyezzen az egyenlőségjel bal oldalán álló attribútummal.

<ertekadas lista>::- <ertekadas> [, <ertekadas lista>]

<ertekadas>::- <attribútum> = <beagyazott kifejezes> |
<attribútum> = (<dml utasitas>)

4.3 Lekérdezõ nyelv

A lekérdező utasítás tulajdonképpen egy vagy több subselect halmazműveletekkel összekapcsolva, és opcionálisan a végén egy ORDER BY rész.

```
<dql utasitas>::-        <lekerdezes>
                         [ORDER BY <order oszlop> [ASC|DESC]
                         {, <order oszlop> [ASC|DESC] }]
<order oszlop>::-        <attribútum>|<egesz kifejezes>
<lekerdezes>::-          <subsel> [ <setoper> <lekerdezes>]
<setoper>::-             UNION
```

Egy subselect lehet SELECT, a NEST, vagy az UNNEST utasítás. A NEST és az UNNEST a beágyazott oszloptípusmegadását és kezelését teszi lehetővé.

```
<subsel>::-              <select utasitas> |
                         <unnest utasitas> |
                         <nest utasitas>
```

A **SELECT** utasítás

A SELECT utasítás több részből áll: egy oszloplistából, egy táblalistából, egy keresési feltételből, valamint egy GROUP BY és egy HAVING részből..A SELECT utasítás adatok lekérdezésére szolgál, meg kell adni, hogy mely táblák, mely oszlopaiban levő, és mely sorok adatait szeretnénk lekérdezni. A táblákat egyszerű felsorolás formájában kell megadni, a kívánt oszlopokat szintén fel kell sorolni, de az összetett attribútumok esetén nem szükségszerű, hogy a teljes összetett attribútumot kérjük, arra is lehet szűkítést adni, egy SELECT mondat formájában. A sorok szűkítése egy keresési feltétellel lehetséges.

```
<select utasitas>::-     SELECT <oszlop lista> FROM <tabla lista>
                         [WHERE <logikai kifejezes>]
                         [GROUP BY <attribútum> {, <attribútum> } ]
                         [HAVING <logikai kifejezes>] |

                         SELECT * FROM <tabla lista>
                         [WHERE <logikai kifejezes>
                         [GROUP BY <attribútum> {, <attribútum> } ]
                         [HAVING <logikai kifejezes>]
<oszlop lista>::-        <oszlop megjeloles> |[ ,<oszlop lista> ]
<oszlop megjeloles>::-   <oszlop kifejezes> [ <alias> ] |
                         <select utasitas> [ <alias> ]
<tabla lista>::-         <tablanev> [ <alias> ]
```

{ , <tablanev> [<alias>]}|
(<lekerdezes>) [<alias>)

<alias>::- <aliasnev> |
AS <aliasnev>

A csillag az oszlop lista helyén azt jelenti, hogy az összes oszlopot kérjük. Az oszloplistában szereplő SELECT utasítás FROM-listájában az adott szint összetett attribútumai, és táblanevek szerepelhetnek. Az oszloplistában egy összetett attribútum neve áll, az azonos értékű, mintha a (SELECT * FROM az attribútum neve) állna.

A WHERE feltételében az egész sorra kell feltételt megszabni, ezért csak olyan attribútumok szerepelhetnek benne, amelyek a FROM mezőben felsorolt táblák attribútumai, alsóbbszintű attribútumok nem.

Az **UNNEST** utasítás

Az UNNEST utasítás csökkenti a reláció beágyazási szintjét oly módon, hogy egy összetett attribútum attribútumatit egy szintel feljebb emeli, ezáltal az összetett attribútum megszűnik, helyette csak az attribútumai lesznek.

<unnest utasitas>::- UNNEST <tablanev> ON <attribútum>|
UNNEST (<lekerdezes>) ON <attribútum>

Az UNNEST utasításnak két paramétere van, a tábla neve, amelyiken a műveletet végre kívánjuk hajtani, és a kibontandó összetett attribútumneve.

A **NEST** utasítás

A NEST utasítás a beágyazott relációs algebra nest műveletének a funkcióját látja el, növeli a beágyazottság mértékét oly módon, hogy a tábla attribútumainak egy megadott csoportjából egy új összetett attribútumot képez, azokat a sorokat, amelyek a maradék attribútumokon megegyeznek, egy sorrá fogja össze, és a megadott attribútumok értékei pedig az új összetett attribútum sorait képezik.

<nest utasitas>::- NEST <tablanev> ON <attribútumlista> [<alias>]|
NEST (<lekerdezes>) ON <attribútumlista> [<alias>]|

<attribútumlista>::- <attribútum> [, <attribútumlista>]

<aliasnev>::- /* SQL szabvanyos alias nev */

A táblanév annak a táblának a neve kell legyen, amelyiken a műveletet végrehajtjuk, az attribútum lista azokat az attribútumokat tartalmazza, amelyekből az új összetett attribútumot kell képezni, az alias nev pedigaz új összetett attribútum neve.

5. Az implementációs környezet és eszközök

A fejlesztés főleg C nyelvben történt, az INGRES adatbáziskezelő rendszert használatával, IBM RS/6000 gépen, AIX operációs rendszer felügyelete alatt.

A lexikális elemzés

A lexikális elemzést a UNIX rendszer LEX nevű lexikális elemzőgenerátora által előállított program végzi. A LEX-nek reguláris kifejezésekkel lehet leírni az egyes lexikális elemeket, és minden ilyen elemhez egy C utasítást lehet megadni, amely akkor kerül végrehajtásra, ha az elemző automata az input szövegben az adott lexikális elemet felismerte.

Leíró adatstruktúrák

A szintaktikai elemzés outputja az input szöveget leíró adatstruktúra. Ennek minden adatot tartalmaznia kell, ami a további elemzéshez, fordításhoz és végrehajtáshoz szükséges.

Szintaktikai elemző

A szintaktikai elemző feladata az input szöveg szintaktikai egységeinek elemzése, és adatstruktúrákká kódolása, azaz a megfelelő adatstrúkturák feltöltése. Az elemző a UNIX rendszer YACC programjával készült, amely LALR1-es típusú elemző.

NSQL utasítások végrehajtása

A szintaktikai elemzés felépített egy leíró adatstruktúrát, amely a végrehajtandó NSQL utasításokat tartalmazza. Az utolsó feladat ezek végrehajtása, azaz a sima SQL-t használó algoritmusok megírása.

Hibakezelés

A C nyelv nem tartalmaz nyelvi eszközöket a hibák és kivételes helyzetek kezelésére, ezért a programban egy hibakezelési mechanizmust kell megvalósítani. Ez a mechanizmus a függvények visszatérési értékén keresztül történik.

Kapcsolat az adatbázismotorral

Az NSQL utasítások végrehajtása közben sima SQL utasításokat kell végrehajtani, amelyek legtöbbje dinamikusan, a program futása közben alakul ki.

Egy külön modul szolgál az adatbázis motorral való kapcsolattartásra, így a programnak lényegében csak ez a része függ az adatbázis motortól. (Néha magát az SQL utasítást a végrehajtó algoritmus állítja elő, így az SQL szintaxisától ezek a kódrészletek is függnek, de az SQL szintaxisnak az a része, amit a program használ, szabványosnak tekinthető).

Irodalomjegyzék

[1] BENCZÚR A., BELSŐ Z., CSERGES E., HAJAS CS., HERNÁTH ZS.,KISS A., KOCSIS A., KONCZ K., KOVÁCS GY., MÁRKUS T., NIKOVITS T., SZÉP T. Negyedik generációs szoftverfejlesztési környezet kidolgozása relációs adatbáziskezelő rendszerek felhasználói felületének kibővítésével. ELTE TTK Ált. Számítástudományi Tanszék 1996
A multi-, nested és fuzzy relációs adatmodell SQL jellegű nyelveinek implementálása. Kutatási-fejlesztési jelentés.

[2] BUSSCHE, J.V. Complex object multi-level fixpointing queries. MFDBS 91

[3] Van GUCHT, D., FISER, P.C. Multilevel nested relational structures. Journal of Computer and System Sciences., Vol. 36, No. 1., pp 77-105, 1988.

[4] KOVÁCS, GY., BENCZÚR A., CSERGES E. Nested Relations and ODMG Collections. Copernicus Large Parallel Database Project, Deliverable Report, 1995

[5] KOVÁCS GY., HAJAS CS. A beágyazott relációs adatmodell reprezentálása relációs adatbáziskezelő rendszerek felhasználásával, Relációs Adatbáziskezelők II.. Magyarországi Konferenciája, 1995

[6] LEVENE, M.,LOIZOU, G. The nested universal relational data model. Journal of Computer and System Sciences. Vol. 49, no. 3. pp. 683-717, 1994

[7] GYSSENS, M., PAREDAENS, J., VAN GUCHT, D. A uniform approach towards handling atomic and structured informations in the nested relational database model. J. ACM Trans. Datab. Syst., Vol. 36, No. 4, pp. 790-825, 1989.

[8] OZSOYOGLU, G., HAFEZ, A. Near-Optimum Storage Models for Nested Relatoins Based on Workload Information. IEEE trans. Knowl. Data Eng., Vol 5, No. 6, pp. 1018-1038, 1993.

[9] ROTH, M.A., KORTH, H.F. and BATORY, D.S. SQL/NF: A query language for ~1NF relational databases. Information Systems, Vol. 12, No. 1, pp. 99-114, 1987.

[10] ULLMAN, J.D. Principles of Database and Knowledge-Base Systems, Vol 1. Computer Science Press, Rockville, Md, 1988.

Adatbázis adminisztráció PLATINUM eszközökkel

Firnága László IQSOFT Rt

Az elmúlt egy-két év során a statisztikai adatok (és saját tapasztalataim) azt mutatják, hogy a licencek eladása csökken és a hangsúly a kész rendszerek, adatbázis alkalmazások fejlesztése felé tolódik el. A első napokban, hetekben jól működő rendszerek lassan adatokkal telítődnek és folyamatosan híznak és ennek lehetséges következményeként lassulnak. Ez az a pont ahol már egy kicsit más szakemberekre van szükség mint a fejlesztéseknél.

Az **IQSOFT Rt.** -felismerve az üzemeltetést elősegítő szofteverek iránti egyre növekvő igényt- elvállalta a PLATINUM cég termékeinek disztribúcióját. A **PLATINUM *Technology*** egy amerikai szoftverforgalmazó cég, mely úttörő szerepet tölt be az informatikai piacon. Termékeik nagyon jó minőségben látnak napvilágot, erről a világ több részén működő kutató és fejlesztő laboratóriumai gondoskodnak. Ezek a relációs adatbázis rendszerek üzemeltetését elősegítő termékek lehetőség szerint több adatbázis kezelő rendszerrel is együtt tud működni (Oracle, Sybase, Informix, ...)

A elkövetkezőkben egy pár figyelemre méltó terméket szeretnék bemutatni:

PLATINUM Tsreorg:

- A ***TSreorg*** Oracle, Sybase, SQLServer, Informix adatbázisok táblastruktúrájának a frissítésére alkalmazható. Feladata - mint neve is mutatja - az organizálás, szervezés. Akkor van rá szükség, ha egy táblaterület szemetes, tehát ha valamit kitöröltek, átírtak, létrehoztak, vagy át kellett mozgatni más területre.
- A ***Tsreorg*** gyorsítja az adatbázis-kezelő rendszer működését. Ugyanis ha egy szegmens (tábla, index, cluster, stb.) adatai töredezettek, ha pl. egy tábla a fájl teljes területén szét van szóródva, akkor mindig meg kell keresni, hogy melyik az aktuális adat, melyik az előző és melyik a következő. A ***TSreorg*** ezen a gondon segít. Az egész folyamat hasonlítható ahhoz, mint amikor egy DOS gépen Speed Disk-kel tesszük rendbe a winchesterünket. Persze sok a különbség, pl. a ***TSreorg***-nak meg lehet mondani, hogy ezt a fájlt fűzze össze, ebből a táblából csináljon egy egységet, meg lehet mondani azt is, hogy az egész táblaterületet generálja újra. Van még ezenkívül egy sor más szolgáltatása is: át lehet vele szervezni egy tábla struktúráját, a műveleteket előre lehet egy bizonyos időpontra ütemezni, ki lehet *egyensúlyozni* a táblákat, indexeket, stb.
- A felhasználó egyedi igényei szerint utasíthatja a ***TSreorg***-ot, hogy az milyen elvek szerint rendezze újra a lemezterületet?
- Elmondható az is, hogy a ***TSreorg*** fő előnye a sebességnövekedés. A fent említett műveleteket el lehet végezni pl. az Oracle Exporttal vagy Oracle Importtal, vagy éppen kézzel is, csakhogy az mind az Oracle-on keresztül megy. A ***TSreorg*** közvetlenül az adatfájlból szedi ki az információt, és ezáltal optimális esetben a művelet sebessége kilencszer gyorsabb, de kb. ötször biztosan gyorsabban dolgozik, mint az Oracle saját maga.
- A ***TSreorg***-gal elért sebességnövekedés magánál az átszervezési folyamatnál, és az átszervezés utáni feldolgozások sebességénél jelentkezik mivel gyorsabban végzi el az átszervezési munkát, és az eredmény gyorsabb reakcióidő az adatbázisból.
- *Az átszervezés ideje függ az adatbázis nagyságától.* Az átszervezés sebessége attól is függ, hogy mire használják azt a táblát. Az állandóan használt, dinamikusan módosuló táblákat - pl. egy AVALON, SAP, Oracle Financials, stb. alkalmazás esetében - állandóan reorganizálni kell annak elkerülésére, hogy ha ma egy művelet elvégzésére egy másodpercet kell várni, ez az idő holnapra ne két másodpercre nőjön.

PLATINUM SQL-Spy:

- Az ***SQL-Spy*** egy Windows/Win 95/Windows NT környezetben futó monitorozó eszköz, mely Oracle, SYBASE, a Microsoft SQLServer és az AIX-es DB2 re is használható (akár egy időben).

- Az adatbázis-kezelő rendszer lelki világából, a működési procedúrákból állít elő különböző diagramokat: CPU kihasználtság, input-output hozzáférés, írás-olvasás, átlagos, logikai, fizikai hozzáférések, stb. Mindezekből képes diagramokat készíteni és lehet új diagram típusokat is definiálni. Nemcsak diagramokat tud megjeleníteni, hanem a legkülönbözőbb információkat is ki lehet vele nyerni pl. az Oracle-ból.
- Összesen 60 féle diagramot és hétféle információs ablakot ismer, attól kezdve, hogy kik vannak bejelentkezve, egészen addig hogy éppen milyen SQL procedúrák vagy függvények futnak az Oracle-ban. Az összes paramétert ki lehet nyerni az Oracle-ból, és ezeket szépen, vizuálisan megmutatja. A nagy előnye az, hogy konfigurálható, új ablakok hozhatók létre, új adatokat lehet egyszerre megjeleníteni. Mindezt egyszerű SQL parancsokkal érhetjük el. Meg lehet nézni a már korábban definiált SQL utasításokat, vagy pl. grafikusan megjeleníthető a CPU kihasználtság. Generálhatunk olyan új diagramokat, amelyek nekünk jobban tetszenek, több adatot mutatnak, más felbontásban vagy más időközönként. Ha az adat értéke túllép egy magadott küszöbértéken, akkor automatikusan indítható egy előre megadott művelet, amely lehet egy SQL utasítás, egy operációs rendszer művelet vagy egy UNIX-os parancs (remote). Van benne egy SQL Script nevű modul is, amellyel SQL DDL és DML parancsokat lehet szerkeszteni nagyon egyszerűen. Az ***SQL-Spy*** segítségével rögzíteni lehet az adatbázis-kezelő rendszerben lejátszódó eseményeket, ezek később megnézhetők.
- Az ***SQL-Spy***-t nemcsak adatbázis adminisztrátorok használhatják, hanem azok is, akiket érdekel az adatbázis-kezelő rendszer belső világa. Ha pl. egy cég vesz egy 10 felhasználós Oracle-t, akkor nem biztos, hogy alkalmazni fog egy adatbázis adminisztrátort, de mindig lesz egy szakember, aki többet szeretne megtudnia többieknél jobban ért az Oracle-höz, és számára az ***SQL-Spy*** nagyon hasznos segédeszköz lehet.

PALTINUM DBVision és ServerVision:

- A ***DBVision*** adatbázis monitorozó, a ***ServerVision*** pedig UNIX szerver monitorozó eszköz. Ebből következik, hogy a ***DBVision*** adatbázis adminisztrátoroknak, a ***ServerVision*** pedig adatbázis adminisztrátoroknak és UNIX adminisztrátoroknak való. A ***DBVision*** segítségével egész napon, héten vagy hónapon keresztül rögzíteni lehet az adatbázis-kezelő rendszerben vagy az operációs rendszerben lejátszódó eseményeket, és ezek később percről percre visszajátszhatók.
- A lehetőségeket érzékeltetéséhez lássunk egy példát. Panaszkodnak a felhasználók, hogy éjfél körül nagyon lelassul a rendszer. Az adatbázis adminisztrátor persze nem fog egész éjjel ott ülni, de bemegy reggel, és elkezdi visszapörgetni az eseményeket. Látja, azért nem mentek az Oracle lekérdezések, mert a fájl hozzáférés nagyon lassú, mégpedig azért, mert éjjel 12-kor indítják a UNIX-os backup-okat, ezek pont azt a winchestert foglalják le, amelyen az Oracle is dolgozna. Tehát felszabadítja az adminisztrátort az alól, hogy órákon keresztül bámulja a képernyőt, vajon rendben mennek-e a dolgok.
- Plusz szolgáltatás, hogy küszöbértékeket lehet beállítani. Az adminisztrátornak nem is kell ott ülnie, csak akkor kell odafigyelnie, ha "megszólal a csengő"- akkor eldöntheti, hogy a riasztást kiváltó ok igényel-e beavatkozást. Pl. ha lassú az Oracle lekérdezés, akár egy TSreorg futtatás is elindítható. Vagy pl. ha egy tábla töredezettsége meghalad egy bizonyos százalékot, akkor a ***TSreorg*** automatikusan elindítható (a ***TSreorg***-ot ugyanis parancssorból is el lehet indítani). Egyébként a ***TSreorg***-on belül is meg lehet adni az időzítést, pl. hogy holnap 12-kor induljon vagy minden második nap, vagy az év egy bizonyos napján.
- A PLATINUM ***DBVision*** és ***ServerVision*** olyan szervezeteknek ajánlható, melyek tevékenysége az egész országra kiterjed, vagy több, párhuzamosan futó rendszerük van. Több mint 100 féle előre beépített értékfigyelő rendszere minden apró mozzanatot (kérésre) értékel, archivál. Szükség esetén, figyelmeztet, közbelép. Nagy UNIX-os adatbázis rendszerek *nélkülözhetetlen* terméke.

PALTINUM Desktop DBA, Enterprise DBA:

- A ***Desktop DBA*** szoftver két felhasználói csoportnak ajánlható Az egyik - logikus módon - az adatbázis adminisztrátorok (röviden: DBA-k), a másik pedig a fejlesztők. A DBA-kat a következők miatt érdekelheti. Egyrészt, mert grafikus felületen lehet kezelni több adatbázist, akár különböző típusúakat is. Egy grafikus képernyőn, egy felületen egyszerre több adatbázis lehet nyitva. Ez a grafikus felület egyszerű, átlátható. Ha valamit az ember elront, azonnal üzenetet kap, hogy vigyázzon, ha átír egy objektumot, mert erre hivatkozó objektumok is léteznek. Tehát elég jól megoldották benne a védelmet, hogy a DBA véletlenül se csinálhasson szamárságot.
- A több adatbázis jelenti-e azt is, hogy az egyik adatbázis pl. Oracle, a másik pedig pl. Sybase. és ezek között sémákat is képes mozgatni, tehát objektumok definícióját - adatokat természetesen nem. A grafikus felületből következik, hogy sokkal átláthatóbb az egész - tehát sokkal egyszerűbb megnézni, hogy éppen milyenek az egyes beállítások, hol és mit kell változtatni, hol vannak a kritikus pontok - mindaz, ami egy DBA napi munkájához szükséges. Különösen nagy előnye a szoftvernek, hogy nagyon gyors. A cache-elési mechanizmusa kiváló, amit egyszer megnéztünk, azt egy listán ott tartja benn a RAM-ban, és csak akkor frissíti, hogyha a felhasználó direkt erre kényszeríti. Tehát nincs folyamatos adatbázis-hozzáférés. Ezenkívül olyan finomságot építettek be, mint hogy egy sémát képes kipakolni egy fájlba, pl. egy user összes objektumának a felépítését, a kreáló szkripteket adatok nélkül. Képes továbbá arra, hogy összehasonlítson - akár különböző adatbázisok között is - sémákat és legyártsa azt a szkriptet, amely a különbségeket kijavítja. Persze meg kell mondani, hogy melyikből keletkezzen a másik.
- A ***Desktop DBA*** használatának nincs adatbázis mérettől függő felső korlátja, általában lefedi a teljes skálát, pici adatbázisoktól a nagyokig. Főleg amiatt, hogy az igazán nagy adatbázisokban nem az adatszerkezet a bonyolult, hanem egyszerűen sok az adat.

PALTINUM Plan Analyzer for Oracle:

- Ez fejlesztőknek való szoftver. Arra szolgál, hogy kényelmesen meg lehessen állapítani, hogy pl. egy SQL utasítás miért lassú. Itt nemcsak lekérdezésről van szó, mert insert-ben és upgrade-ben is lehet beágyazott select. Nagyon szép grafikus felületben lehet dolgozni. Bár az Oracle behozta az új optimalizálási módszerét, a Cost Base Optimizer-t, ez egyelőre nem teljesen kiforrott technológia, ezért mindig figyelni kell, hogy mit csinál az optimalizáló. Megjelentek a "hint"-ek is, amelyekkel meg lehet mondani, hogy az optimizer bizonyos dolgokat hogyan csináljon meg, pl. egy select-ben milyen sorrendben kapcsolódjanak össze a táblák. Ezeket a műveleteket ellenőrizni kell. Mindez Oracle alverziónként is változik. A ***Plan Analyzer for Oracle*** lehetővé teszi tesztek készítését. Grafikusan kirajzolja, hogy mit fogok csinálni, és pirossal jelzi, hogy baj lesz, végignézhetem lépésenként, hogy mi fog történni, hogy mennek majd az eredménysorok. De ennél sokkal izgalmasabb az, hogy megmondhatom, hogy milyen kliens oldali eszközről fogom használni. Fontos, hogy ismeri az SQLWindows-t, az SQLForms-ot, az SQLPlus-t, a Power Builder-t, tehát azt lehet mondani, hogy általános. Meg lehet azt tenni, hogy többször tesztelek egy-egy select-et, különböző típusokkal: a régi típusú Rule Based optimalizálóval, a Cost Based-del (ennek is van két formája - az első sor jöjjön le gyorsan, illetve az egész eredményhalmaz gyorsan jöjjön le, ez mást-mást jelent), majd kipróbálhatom, hogy "hint"-ekkel telezsúfolva milyen eredményt adna. Már az is nagyon jó dolog, hogy utána mindezt grafikusan ábrázolja, és mint fejlesztőnek, van rá befolyásom, hogy melyiket használja az Oracle. Megállapíthatom, hogy ebben az adott Oracle verzióban SQLWindows kliensből ez a bizonyos optimalizációs technika sokkal jobb eredményt ad, mint a többi. Sajnos, egyáltalán nem biztos, hogy a költség alapú (Cost Based) jobbat szolgáltat, mint a szabály alapú (Rule Based). Az igazán izgalmas az, hogy a lekérdezéseket és a hozzájuk tartozó statisztikákat le tudom tenni adatbázisba. Ha ezek után pl. egy Oracle verziót váltok, megtehetem azt, hogy az új Oracle verzióra egy automatizmus segítségével összehasonlítást végzek. Előfordulhat, hogy kapok egy jelzést: itt bizony gond van, mert van egy select-em, amely jobb eredményt adna, hogyha más optimalizálási módot választanék. Jó, hogy ezt egy mozdulattal megtehetem.

- Az adatbázis-hangolásnak két része van: a szerver oldali, ami a DBA feladata, és a kliens oldali. A szakmában ismeretes egy bizonyos diagram, két görbével. Az egyik görbe azt mutatja, hogy a fejlesztés különböző szakaszaiban mennyire költséges egy alkalmazás hangolása, a másik pedig azt mutatja, hogy mekkora a várható eredmény. Mind a két görbe exponenciális, de a költség nagyon kicsi az alkalmazás-fejlesztés stádiumában, amikor pl. a ***Plan Analyzer for Oracle***-t érdemes használni. Ezzel szemben óriásira nő a költség a kész alkalmazások hangolásakor. A várható eredmény pedig ugyanilyen mértékben csökken. Tehát a tervezési és fejlesztési stádiumban pici ráfordítással nagyon nagy eredményeket lehet elérni, míg a DBA iszonyú ráfordítással hetek munkájával 5-10%-os gyorsulásért küzd. Tehát egy ilyen termékkel rengeteg költséget lehet megtakarítani. Ha fejlesztők a fejlesztés során használják ezt az eszközt, az elkészült alkalmazás akár 70-80%-kal is gyorsabb lehet.

ObjectStore

Ismerkedés az objektum-orientált adatbáziskezeléssel.

Németh Miklós, IQSOFT RT, 1997
nemeth@iqsoft.hu

Az ObjectStore az **Object Design, Inc. (ODI)** (USA) cég terméke.

Az ObjectStore-hoz kapcsolódó más ODI termékek:

•OpenAccess - SQL92 elérést biztosító ODBC interfész ObjectStore adatbázisokhoz.

•DBConnect - C++ programok számára leegyszerűsíti a relációs adatbázisok kezelését.

•Inspector - ObjectStore adatbázisok böngészését biztosítja.

•Performance Expert - ObjectStore adatbázis hangoláshoz szükséges információkat állít elő.

•Spatial Object Manager - 2,3 és n-dimenziós geometriai és geográfiai adatbázisok kezelését támogató objektumkönyvtár.

•Text Object Manager - Full-text keresést és dokumentum kezelést támogató objektumkönyvtár, ami a Verity Topic technológiájára épül.

•Video Object Manager - A videót önálló adattípusként (nem csupán BLOB-ként kezelő objektumkönyvtár.

•Audio Object Manager - Az audio adatokat önálló adattípusként kezelő objektumkönyvtár.

•Java Object Manager - Java appletek ObjectStore-ban történő tárolását, adminisztrálását és web alkalmazások felé történő biztosítását támogató osztálykönyvtár.

Miről lesz szó?

- Az ObjectStore helye az OO fejlesztési folyamatban
- ObjectStore alkalmazások architektúrája
- ObjectStore adatbázisok (sémák)
- ObjectStore-ban tárolt adatok kezelése

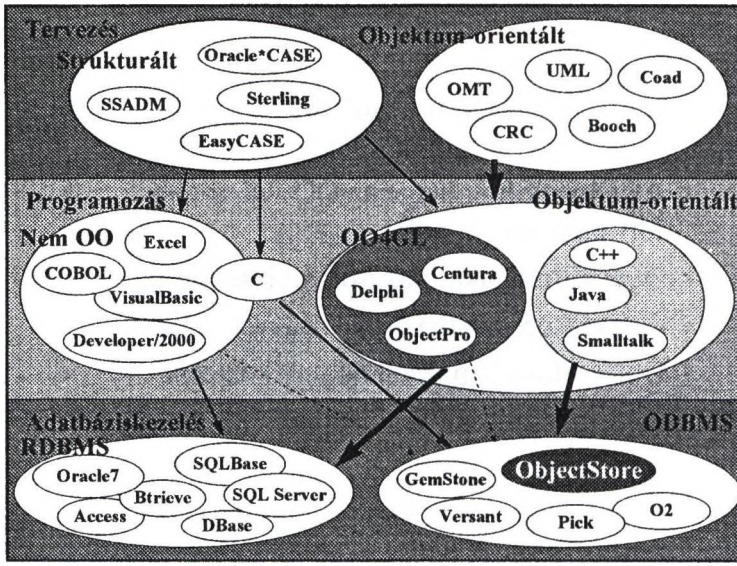
Az ObjectStore-hoz kapcsolódó további ODI termékek:

•ObjectForms - Kibővíti a HTML-t és így lehetővé válik ObjectStore adatok közvetlen kezelése illetve HTML-lapok dinamikus generálása.

•Image Object Manager - Különféle formátumú (bmp, gif, pcx, tif) képi adatok kezelését támogató objektumkönyvtár.

•HTML Object Manager - HTML lapok tárolását, kezelését támogató objektumkönyvtár

•Time Series Object Manager - Idősorok kezelését támogató objektumkönyvtár



A **vastag nyíl** azt jelzi, hogy jelenleg (1996 - 97) ezek az utak a legperspektivikusabbak. A **normál vastagságú nyíl** azt jelzi, hogy ez az út is járható és jól ismert. A **vékony szaggatott nyíl** azt jelzi, hogy ezen az úton is lehet menni, de ez még nem járt és nincsenek is mindenütt jelzőtáblák.

A **strukturált tervezési** gombócban némely krumpli **CASE eszközt** és az ahhoz tartozó módszertant jelent. Az **UML**-t (Unified Modeling Language) még nem véglegesítették, és csupán a fogalmakkal és jelöléssel foglalkozik.

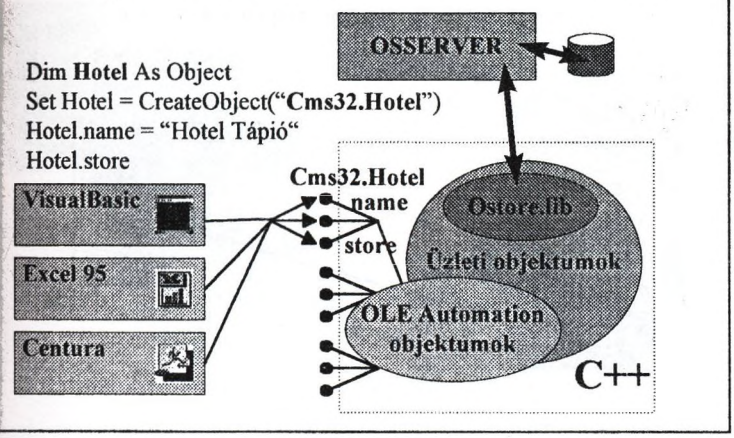
A **C** nyelv formálisan nem OO nyelv de rugalmassága (univerzálissága) miatt kiválóan lehet OO programozásra is használni. A C programok ODBMS-ekhez való illesztése teljesen megoldott.

Az **Excel**-t is felvettük a programozási eszközök közé mivel programozhatósági (VBA) lehetőségei ma már nem maradnak ela legtöbb 4GL rendszerétől.

A sárga körbe helyezett nyelvek tekinthetők a **szabványos OO nyelveknek**, és csak ezeknek az illesztése megoldott ODBMS-ekhez. Az OO4GL-ek jelenleg csak RDBMS-ekkel használhatók közvetlenül (azaz minden extra ráfordítás nélkül).

Az **ObjectStore** a legelterjedtebb ODBMS, Magyarországon az ODBMS-ek közül csak az ObjectStore-t forgalmazzák, és mindjárt két cég (IQSOFT, HiCare).

ObjectStore illesztése 4GL-ekhez



A **bogyós végű faágak** dispatch interface-ket jelentenek, a **nyilas végű faágak** ezen interfészek hívását.

Az OLE Automation objektumok nevei (pl. "Cms32.Hotel") a Windows 95/NT registryben is tárolódnak.

Az ObjectStore-t csak szabványos OO nyelveken lehet programozni: C++, Java, Smalltalk, C. 4GL támogatást csak közvetetten lehet elérni.

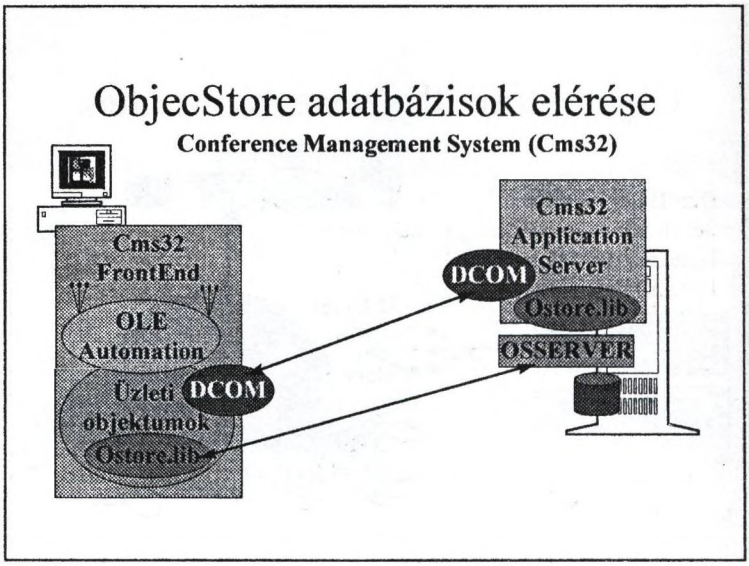
A Windows platformokon a C++ fordítók közül csak a **Microsoft Visual C++ 4.1**-et támogatja. Az **MFC 4.1** ObjectStore-hoz módosított változata lehetővé teszi, hogy különösebb erőfeszítés nélkül készítsünk OLE Automation objektumokat ObjectStore-hoz, így az OLE Automationt támogató 4GL-ekből felprogramozhatók az ObjectStore-ra alapuló üzleti objektumaink.

Ha olyan 4GL eszközünk van, ami az OLE Automationt csak ActiveX-eken (OCX) keresztül támogatja, akkor **ActiveX objektumokat** kell készítenünk.

Az üzleti objektumok normális perzisztens C++ osztályok, amelyek (normálisan) függetlenek az MFC-től, OLE Automation-tól.

A **programrészlet** azt mutatja be hogyan használja egy Visual Basic program üzleti objektumainkat.

A **szaggatott vonallal határolt négyzetben** található komponensek C++ - ban készülnek.



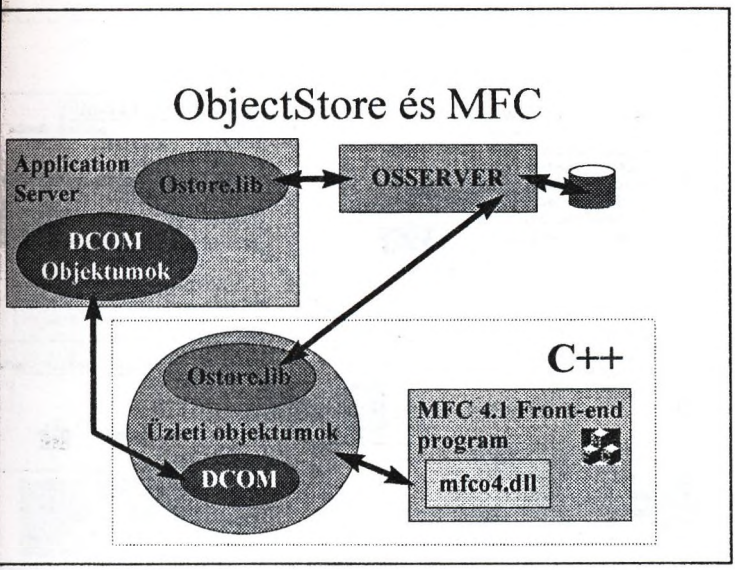
Egy többfelhasználós (kliens-szerver) rendszer esetén **alkalmazás-szervert** (application server) célszerű készíteni. Az alkalmazás-szerver **DCOM** (vagy estleg CORBA (Iona Orbix, VisiGenic VisiBroker)) objektumokként hozza nyilvánosságra szolgáltatásait.

A **front-end programok** az üzleti objektumaikon keresztül **közvetlenül** (az Ostore.lib révén) érik el az ObjectStore szervert, vagy a **DCOM**-mal (CORBA) az alkalmazás-szervert.

Az **OLE Automation**-ra csak a front-end programoknak van szüksége. Mivel a DCOM bináris interfész 4GL-ekből közvetlenül nem használható csak az OLE Automation protokollon keresztül. A front-end üzleti objektumok az OLE Automation objektumok révén teszik elérhetővé a perzisztens objektumokat a (legkönnyebben 4GL-ben elkészíthető) felhasználói felület moduljainak.

Egy bonyolultabb lekérdezés útja: `Excel :automation> automationObjektum :dcom> alkalmazásSzerverObjektum :query(ostore.lib)> ObjectStore`

Közvetlen adatmanipuláció: `Centura :automation> automationObjektum > üzletiObjektum :ostore.lib> ObjectStore`



Az **MFC implementáció előnyei** 4GL programozással szemben:

•A front-end programok sokkal **gyorsabbak** lesznek mivel közvetlen gépi kódú programok készülnek, ami bizonyos körülmények között (lassú CPU-jú, kevés RAM-os PC-k) fontos szempont lehet.

•A lassú **OLE Automationra** alapvetően nincs szükség. **DCOM** (vagy CORBA) használható az **alkalmazás szerver** megvalósítására.

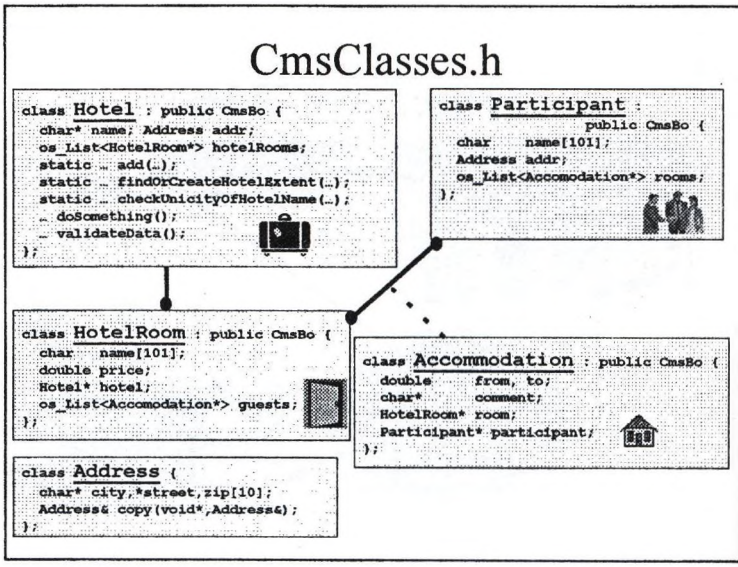
•Csak **C++** -ban lehet programozni, ami a lehető legrugalmasabb (legáltalánosabb) alkalmazások készítését teszi lehetővé. A C++ általánosan elfogadott szabványos nyelv míg a 4GL-eket a szakma egy jelentős része nem "fogadja be".

A **4GL-ek előnyei** az MFC-vel szemben:

•A mai **Pentium** dömpingben a 4GL-ek is elfogadható sebességet produkálnak.

•Bizonyos fokú OLE Automation támogatást az MFC alkalmazásokba is be kell építeni mivel a **riportozást, vezetői lekérdezéseket** teljes egészében vagy nagyrészt az Excelre kell bízni.

•A 4GL-ek sokkal **integráltabb** és vizuálisabb (nem csak dialog boxok vannak) fejlesztőkörnyezetet biztosítanak mint az MFC wizardok (pl módosítás, törlés).



Az ObjectStore adatbázisok létrehozásához nincs szükség különösebb SQL-szerű szkriptekre. A C++ -ban szokásos módon definiáljuk az alkalmazás (üzleti) osztályait.

Az ObjectStore rendszer garantálja, hogy olyan (front-end) program, amelyik inkompatibilis az adatbázissal, nem tud bejelentkezni az adatbázisba.

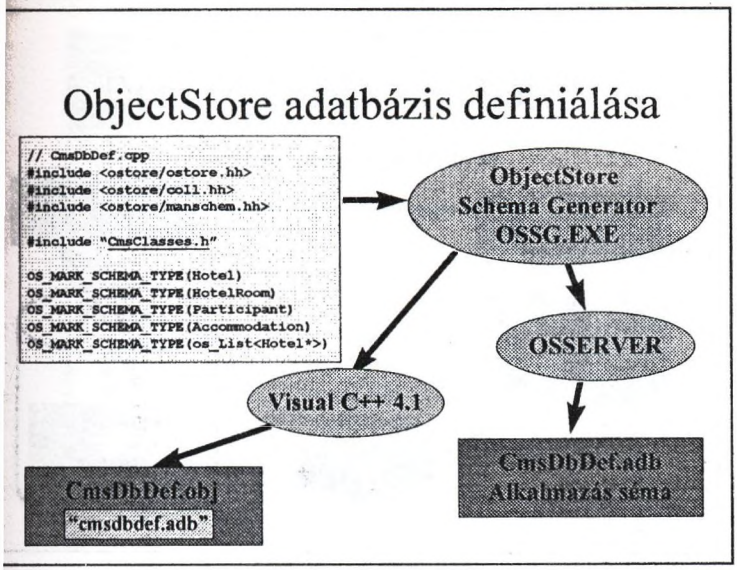
Nincs szükség külön adatbázis-installálásra sem, az ObjectStore automatikusan elvégzi az adatbázis installálását, amikor az első program létrehozza.

A **séma-evoluálás** során az ObjectStore automatikusan elvégzi a **triviális adatbázismódosítási (konvertálási)** feladatokat. A fejlesztőknek azonban egész sor eszköz áll rendelkezésre, hogy a séma-evoluálást az igényeknek megfelelően alakítsák (**testreszabott séma-evoluálás**). Magát az evoluálást a fejlesztőknek kell explicit kezdeményezniük (OSSEVOL.EXE vagy os_schema_evolution::evolve() hívásával).

A fenti objektum-diagramon az **OMT és az UML jelölését egyesítve** ábrázoltuk az asszociációkat.

Az **Address** osztály csak mint aggregált tagváltozó (member variable) jelenik meg az adatbázisban, így **saját definiálású adattípusként** kezelhetjük, akinek nincs azonosítója (identification).

Minden perzisztens tárolású azonosítható osztályt a **CmsBo** alaposztályból származtattuk, aminek még nagy hasznát vesszük (pl polimorf listák).



A sémagenerálásnak két célja van: (1) előállítsa az ObjectStore adatbázis létrehozásához (séma-installálás), ellenőrzéséhez, és evoluálásához szükséges **alkalmazás sémafájlt** (cmsdbdef.adb); (2) előállítsa az alkalmazás-programokhoz linkelendő **séma tárgymodult**.

A **sémadefiníciós forrásfájlban** (cmsdbdef.cpp) az ObjectStore adatbázisban tárolandó osztályokat kell megjelölni (OS_MARK_SCHEMA_TYPE).

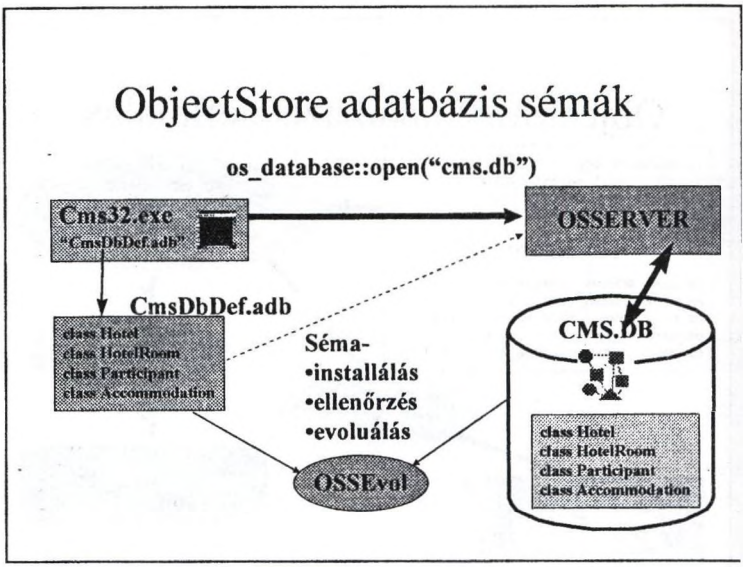
A generált **séma tárgymodulba** (cmsdbdef.obj) begenerálódik az **alkalmazás sémafájl elérési útja** is, amit az OSSETASP.EXE segédprogrammal módosítani lehet.

A **séma tárgymodulba** generálódnak az OS_MARK_SCHEMA_TYPE által megjelölt osztályok statikus **get_os_typespec** eljárásainak implementációi is.

Perzisztens objektum létrehozáskor a **new** operátornak paraméterként át kell adni a létrehozandó osztály get_os_typespec eljárása által visszaadott os_typespec* típusú értéket.

A generáláshoz az OSSG az **ObjectStore szervert** és a **Visual C++** kompájlert használja.

Egy ObjectStore adatbázisa **tárolja** az adatbázisban tárolható (és tárolt) **objektumok leírását**, így lehetőségünk van -a Meta Object Protocol(MOP) API segítségével- általános (adminisztrációs) programok írására mint amilyen az **OSBrowse.Exe**, **ObjectStore Inspector**.

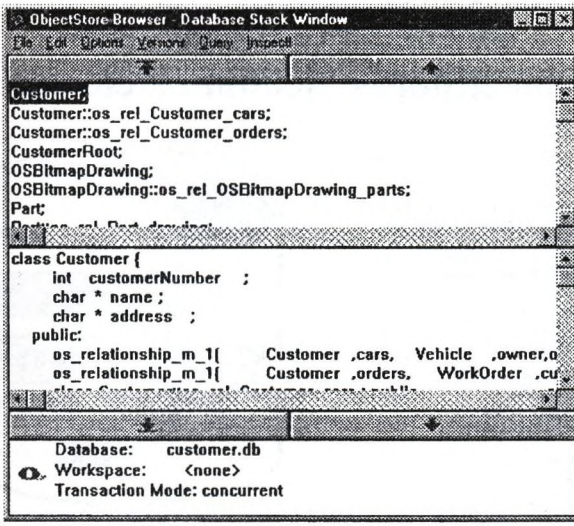


Az alkalmazás-programok (cms32.exe) tartalmazzák (a generált cmsdbdef.obj-ben) az alkalmazás sémafájl (cmsdbdef.adb) elérési útját.

Egy ObjectStore adatbázis megnyitásakor az ObjectStore három feladatot végez.

Ha az adatbázist most hoztuk létre -az open-nel ezt megtehetjük-, akkor az alkalmazás sémafájl alapján az ObjectStore **installálja** a sémát a létrehozott adatbázisba. **Ha már létezik** a megnyitott adatbázis, akkor leellenőrzi, hogy az adatbázist megnyitó program (cms32.exe) alkalmazás sémája **kompatibilis**-e a megnyitott adatbázis sémájával. Ha nem kompatibilis, akkor nem engedi megnyitni az adatbázist. Ekkor két eset lehetséges: a programunk régi és kész szerencse, hogy nem tudott bejelentkezni az adatbázisba. Az adatbázis szerkezetét módosítani kellett és programunk már ezen új séma szerint működik. Ekkor a fejlesztőknek **explicit evoluálni** kell az adatbázis vagy az **OSSEvol.Exe segédprogrammal** vagy -bonyolult módosulások esetén- egy erre a célra készített **evoluáló programmal** (ami lehet része a cms32-nek).

Az **OSSEvol.Exe** egy olyan segédprogram, aminek ha megadjuk az alkalmazás sémafájl és az evoluálandó adatbázis elérési útját (és opcionálisan néhány evolúciós paramétert), akkor az ObjectStore szerverrel elvégezteti a séma-evoluálást.



Az ObjectStore sztenderd fejlesztőcsomagban található az **OSBrowse.Exe**, amivel az ObjectStore adatbázisokat lehet böngészni.

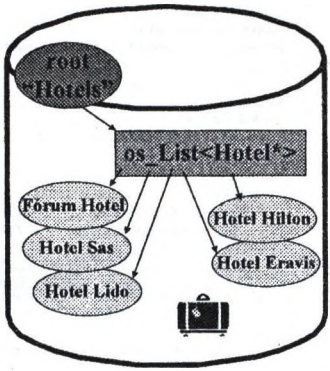
A képen a TestRide nevű mintaprogram customer.db adatbázisa sémájának egy részlete látható: a legfelső panelben láthatók az osztályok, a középső panelben pedig a kiválasztott (Customer) osztály definíciója.

Látható, hogy a kapcsolatokat úgy vette fel a tervező, hogy azok automatikusan szinkronizálódjanak: **os_relationship_m_1**. A mintapéldában nem ezt alkalmazzuk.

Az OSBrowse-nál sokkal elegánsabb kivitelezésű az **Inspector**, ami még arra is képes, hogy egy adatbázisnak felrajzolja az OMT vagy Booch jelölésrendszer szerinti objektum-modelljét.

ObjectStore objektumok elérése

```
os_database* db; os_database_root* r;
os_List<Hotel*>* hotels; Hotel* h;

db = os_database::open("cms.db");
OS_BEGIN_TXN(…,os_transaction::update)
{
  r = db->find_root("Hotels");
  hotels = (os_List<Hotel*>*)
    r->get_value();
  os_Cursor<Hotel*> hc(hotels);
  for (h = hc.first();
       hc.more;
       h = hc.next()) {
    h->doSomething();
  }
} OS_END_TXN(…)
db->close();
```



Az RDBMS-eken elkényelmesedett fejlesztőt hidegzuhanyként éri, hogy az ObjectStore esetén neki kell gondoskodni arról, hogy az **adatbázisban létrehozott objektumokat meg is találja**. (Elvileg a MOP segítségével is megtalálhatók az objektumok, de ennek nem ez a rendje).

Az ObjectStore adatbázisokban **kiindulási (root) pontokat** hozunk létre, amelyeket globális kereséssel megtalálhatunk.

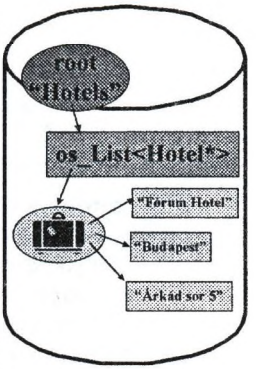
Mielőtt elkezdenénk ömleszteni az adatokat az adatbázisba gondosan meg kell tervezni, hogy milyen módon is akarjuk az adatokat elérni (kollekciók, indexek, pointerek).

Egy ObjectStore adatbázisba **semmi sem kerül be magától** (sem egy kollekcióba, sem egy indexbe) mindenről nekünk kell gondoskodni.

A helyzet azért nem ennyire drámai, mivel az ObjectStore számos segítséget ad ahhoz, hogy a triviális dolgok automatikusan menjenek (pl. kapcsolat (relationship) szinkronizálás, automatikus indexkarbantartás, lekérdezés optimalizálás).

Perzisztens ObjectStore objektumok létrehozása

```
// hotel.cpp
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "CmsClasses.h"

Hotel* Hotel::add(os_database* xdb,
  char* xname,Address& xaddr)
{
  os_List<Hotel*>* hotels; Hotel* h;

  hotels = findOrCreateHotelExtent(xdb);
  if (!checkUnicityOfHotelName(hotels,xname))
    return NULL;
  h = (Hotel*)new(xdb,get_os_typespec()) Hotel;
  h->name = AllocStr(h,xname);
  h->addr.copy(h,xaddr);
  hotels->insert(h);
  return validateData() ? h : NULL;
}
```



Feltételezzük, az **add** eljárást update tranzakción belül hívja meg egy vezérlőobjektum.

Az add statikus eljárás és az újonnan létrehozott Hotel objektum pointerét adja vissza. Ha sikertelen az eljárás (NULL visszatérési érték), akkor a vezérlőnek abortálni kell a tranzakciót.

A **checkUnicityOfHotelName()** saját eljárás során nekünk kell ellenőrizni, hogy szerepel-e már a listában a paraméterben megadtott nevű hotel objektum.

Az **egyediség ellenőrzését** az ObjectStore **indexek** (os_index_path) esetén a kollekcióhoz (pl. os_List<Hotel*>) való rendeléskor meg lehet adni:

```
hotels->add_index(idx,os_index_path::no_duplicates
  | os_index_path::signal_duplicates);
```

Az ObjectStore index-szolgáltatásai kifinomultak és komplexek.

Az indexeknek az adatbázisbeli létrehozását, nyilvántartását, kollekciókkal való összerendelését az alkalmazás üzleti objektumainak kell végezni.

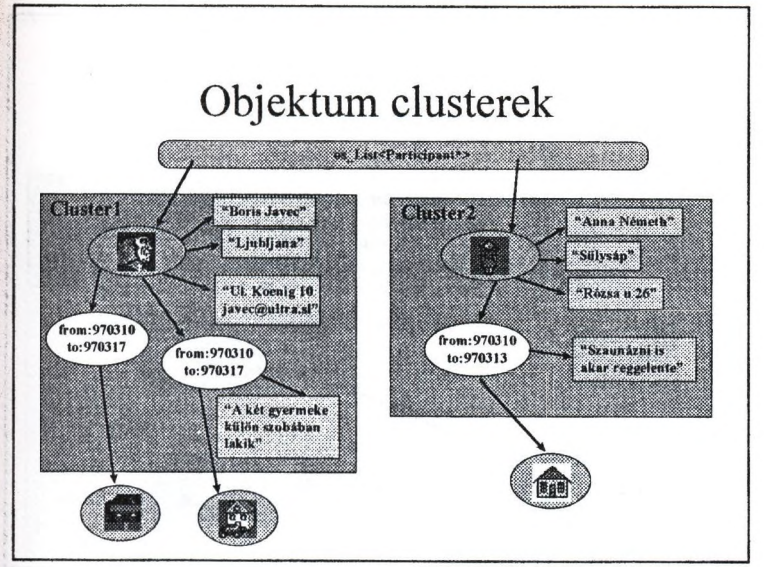
AllocStr

```
char* AllocStr(void* xo,char* xs)
{
  char* s;
  s = (char*)new(os_segment::of(xo),
    os_typespec::get_char())
    char[strlen(xs) + 1];
  strcpy(s,xs);
  return s;
}
```

- **A sztringek is dinamikusan allokálhatók.**
- **Az AllocStr a sztringet abba a segmensbe helyezi, amelyikbe az objektum is található: os_segment::of(void*)**
- **Ha véletlenül az xo nem perzisztens cím, akkor is működik az AllocStr.**

Íme a **findOrCreateHotelExtent** statikus eljárás. Extentnek nevezik azon objektumok összességét, amelyek egy osztályhoz tartoznak. Az extent objektumait szokás egy listában összefogni, ha keresni kell közöttük.

```
os_List<Hotel*>* Hotel::findOrCreateHotelExtent(
  os_database* xdb)
{
  os_database_root* r;
  os_List<Hotel*>* hotels;

  if (r = db->find_root("Hotels")) {
    if (hotels = (os_List<Hotel*>*)r->get_value())
{
      return hotels;
    }
  } else {
    r = xdb->create_root("Hotels");
  }
  hotels = &os_list<Hotel*>::create(xdb);
  r->set_value(hotels);
  return hotels;
}
```



A lila ellipszisek (a férfi- illetve a kislány-kép) a **Participant**, a világoskékek (ház-képek) a **HotelRoom**, a fehérek (from: to:) az **Accommodation** osztály objektumai.

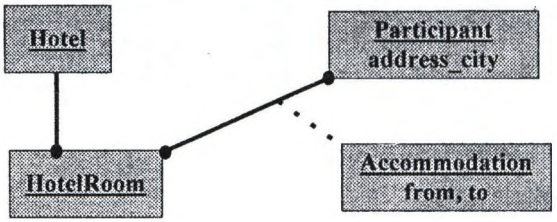
Az ObjectStore **objektum clusterei** (os_object_cluster) a legrugalmasabb objektumösszerendelési megoldásokat is lehetővé teszik.

Az ObjectStore objektum clusterezése **nem dől el végérvényesen az adatbázis létrehozásakor**. Hogy egy objektum melyik clusterbe kerül, azt dinamikusan határozza meg egy általunk kidolgozott tárolásvezérlő (object storage manager).

Mivel az ObjectStore objektumokkal és nem pedig táblákkal dolgozik, ugyanannak az osztálynak a példányait többféle clusterben is elhelyezhetjük.

Lekérdezések

Kérdezzük le az adatbázisban tárolt hotel lista alapján azokat a Hotel objektumokat, ahol 1996.01.01 és 1996.01.31 között lakott Budapesti konferencia-résztvevő (Participant)



RDBMS-ekben **nem lehet struktúrája az attribútumoknak** (mint az address esetén) ezért az address attribútumot elemeire bontva tároljuk (address_city, address_street, stb.)

A fenti adat/objektummodellben kiemeltük a lekérdezésbe szereplő attribútumokat (from, to, address_city).

SQL Select

```
select * from Hotel h where
exists (select * from HotelRoom r where
  exists (select * from Accommodation a where
    exists (select * from Participant p where
      address_city = 'Budapest'
      and p.id = a.participant_id
    )
    and from >= 19960101 and to <= 19960131
    and a.hotelroom_id = r.id
  )
  and r.hotel_id = h.id
)
```

A piros színű dőlt betűs részek a tényleges keresési feltételt emelik ki, a kék álló betűs részek a kapcsoló feltételeket jelentik.

Egy RDBMS-nek a fenti SELECT optimalizálásához 8 index kell:

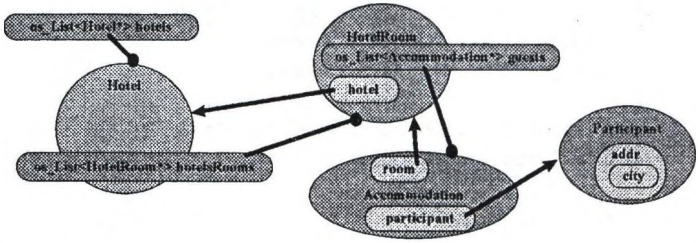
- Accommodation(from,to)
- Participant(address_city)
- Hotel(id) - elsődleges kulcs
- HotelRoom(id) - elsődleges kulcs
- HotelRoom(hotel_id) - idegen kulcs
- Accommodation(hotelroom_id) - idegen kulcs
- Accommodation(participant_id) - idegen kulcs
- Participant(id) - elsődleges kulcs

A fenti 8 indexből 6 csupán azért kell, hogy a legtermészetesebb kapcsolatokat felhasználhassuk.

A fenti SELECT-et lehet egy kicsit egyszerűsíteni és a szubselectekből három táblás kapcsolást lehet készíteni de ez a lépés már a "kúrálási" folyamat része, ugyanis abban reménykedünk, hogy akkor majd gyorsabb lesz az RDBMS.

Lekérdezések

```
os_Collection<Hotel*>& qr;

qr = hotels->query("Hotel*","(*hotelRooms)
   [:(*guests)[: from >= 19960101
     && to <= 19960131
     && strcmp(participant->addr.city,
               \"Budapest\") == 0:]:]");
```



A lekérdezéseket mindig kollekciókon végezhetjük. A **query** mindig kollekciót (halmazt ad vissza), míg a **query_pick** egy elemet.

Az eredményhalmaz elemei tartozhatnak többféle de azonos ősel rendelkező osztályhoz is (**polimorf listák**); íme miért fontos, hogy valamennyi perzisztens osztály egyetlen közös őstől származzon.

Ha a kollekcióhoz (perzisztens) **indexek** is tartoznak, akkor az ObjectStore lekérdező motorja felhasználja azokat a feltételek kiértékelésénél.

Az ObjectStore lekérdezésekben a **kapcsolás (join) pointerek mentén** folyik, amelyek sokkal gyorsabb válaszidőket eredményezhetnek.

A lekérdezés kifejezésbe **saját definiálású függvényeket** is megadhatunk, ami igen rugalmas lekérdezéseket tesz lehetővé. Ráadásul az ObjectStore lekérdezés kifejezésbe megadott **függvények gépi kódban íródnak** és így sebességük messze túlszárnyalja az SQL motorok interpretált függvényeit.

A lekérdezések igazán akkor hatékonyak többfelhasználós környezetben, ha **alkalmazás szerverben** valósítjuk meg őket.

Lekérdezés optimalizálás

```
os_List<Hotel*>* Hotel::qryHotelsWithBpGuestsIn96Jan()
{
os_Collection<Accommodation*> aq; Accommodation* a;
os_List<Hotel*>* hq = new os_List<Hotel*>;

aq = accommodations.query("Accommodation*",
  "from >= 19960101 && to 19960131 &&
   strcmp(participant->address.city,\"Budapest\")");
os_Cursor<Accommodation*> ac(aq);
for (a = ac.first();ac.more();a = ac.next()) {
  hq->insert(a->room->hotel);
}
return hq;
}
```

Feltételezzük, hogy az **accommodations** statikus változó és valamennyi Accomodation objektumra hivatkozó kollekció.

Miután leválogattuk a megadtott feltételnek megfelelő Accommodation objektumokat (az **aq** kollekcióban) , könnyen előállítható a Hotelek listája.

Egy Accommodation objektumból eljutni egy Hotel objektumig csupán két pointerhivatkozás; ennek a sebességével nem tudnak versenyezni az RDBMS-ek, a joinjaikkal.

A lekérdezést végző eljárást tovább optimalizálhatjuk, azzal ha pl. lekérdezzük a rendelkezésre álló kollekciók elemszámait:

```
if (accommodations->cardinality() <
                participants->cardinality() * 2)
```

és ezután dinamikusan döntjük el a **lekérdezési stratégiát**.

Egy ObjectStore adatbázist a lekérdezések szempontjából úgy fel tudunk programozni, hogy annak sebességével egy RDBMS nem vetekedhet. Nem véletlen, hogy valós idejű rendszerek esetén, a telekommunikációs iparágban, a hadseregeknél **előszeretettel dolgoznak objektum-orientált adatbáziskezelőkkel**. Egyes mérések szerint bizonyos feladatok esetén egy ODBMS 100 - 1000-szer is gyorsabb lehet mint egy RDBMS.

Egy elem lekérdezése

```
Hotel& h;
h = hotels->query_pick(
  "Hotel*","strcmp(name,\"Hotel Sas\") == 0");
```

Paraméterezett lekérdezés

```
Hotel& h;
os_coll_query& q = os_coll_query::create_pick(
  "Hotel*","strcmp(name,(char*)XNAME) == 0",db);
...
os_bound_query bq(q,
  os_keyword_arg("XNAME",xname));
h = hotels->query_pick(q);
```

A **query_pick** egyetlen objektumot ad vissza.

A **query_pick nem egyszerűen a query speciális esete**, mivel az ObjectStore **más algoritmusokat használ**, ha csak egyetlen elemet kell keresnie mintha egy egész halmazra fel kell készülnie.

A **paraméterezett lekérdezéseknek** akkor van értelme, ha a lekérdezést az **adatbázisban tároljuk** (os_coll_query). A tárolt lekérdezést később tetszőleges konkrét paraméterértékkel lefuttathatjuk (xname).

Az RDBMS-ek viewi nem paraméterezhetők.

A **query** lekérdezés kifejezés sztringben a **kollekciók elemszámára** is lehet hivatkozni:

```
hotels->query("Hotel*",
  "(*hotelRooms)[::].cardinality() == 0");
```

Relációs adatbázisok tervezése OMT módszertannal

Fazekas Zsuzsanna, IQSOFT Rt.

Napjainkban a relációs adatbázis-kezelők és az objektum-orientált programozási nyelvek világát éljük. E két terület azonban eltérő absztrakciós szinten mozog, különböző fogalmakkal operál. Ezt a szakadékot valahogyan át kell hidalni.

Az egyik választási lehetőségünk az, hogy strukturált módszertant használunk, a rendszert relációs fogalmakkal közelítjük meg, az adatbázis-táblák definiálására koncentrálunk. Ennek az a hátránya, hogy a programozónak szinte újra kell terveznie a rendszert, hiszen a fejlesztőkörnyezete osztályokat, műveleteket és attribútumokat ismer. Tehát a munka számára azzal kezdődik, hogy megkeresi az objektumokat a rendszerben.

Jobbnak tűnik az a választás, hogy az OO fejlesztési környezethez OO tervezési módszertant használjunk. Ekkor azonban meg kell fogalmaznunk néhány, az OO tervezési módszertan alapelvein kívüli szabályt is, amelyek lehetővé teszik azt, hogy a rendszertervből megbízható, jó relációs adatbázist tudjunk generálni. Az IQSOFT-nál mindennapi gyakorlatunkban a Rumbaugh és társai által 1991-ben definiált **Object Modeling Technique** (OMT) módszertant használjuk, a PLATINUM Technology Paradigm Plus CASE eszközének segítségével.

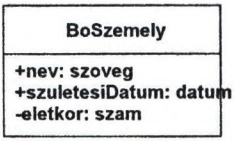
Az első és egyik legfontosabb szabályunk az, hogy az alkalmazás az adatbázissal csak **üzleti objektum**okon (Business Object) keresztül kommunikál. Az alkalmazásban tehát nincsenek "kósza" SQL-utasítások, valamennyit jól meghatározott helyen találjuk meg. Ha bármelyik osztálynak – legyen az egy dialog-box, táblázat-ablak vagy épp egy riport – szüksége van adatbázisban tárolt adatra, akkor azt az üzleti objektum megfelelő műveletének meghívásával éri el.

A másik nagyon fontos alapelv az, hogy minden egyes osztályhoz hozzárendelünk egy úgynevezett **objektum-azonosítót**. Ez az objektum-azonosító lesz az adatbázisban is az elsődleges kulcs.

Nézzük ezek után, hogyan tudjuk az OO és a relációs világ fogalmait megfeleltetni egymásnak!

Objektum-orientált fogalom	Relációs fogalom	
osztály (class)	tábla	
attribútum	oszlop	
művelet	tárolt procedúra	
objektum	sor	rendszerint nem modellezzük
asszociáció		
aggregáció		
generalizáció		
	index	
	view	
	constraints	

Az **adatbázistáblák** osztályként való modellezésére két választási lehetőségünk van. Az egyik, hogy az üzleti objektumok attribútumaiból és egyéb jellemzőiből generáljuk az adatbázistáblát. Az ilyen osztályok perzisztensek és modell szerepet játszanak. A tervezés során lehetőségünk kell legyen arra, hogy megjelöljük azokat az attribútumokat, amelyekből adatbázis-oszlopnak kell keletkeznie. Eddigi gyakorlatunkban azt a nem túl "tisztességes" megoldást követtük, hogy a generálandó attribútumokat publikusnak állítottuk be.

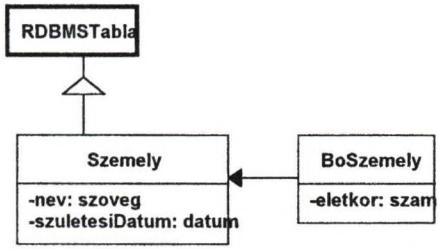



```
CREATE TABLE BoSzemely(
  nev                VARCHAR(50) NOT NULL,
  szuletesiDatum     DATE NOT NULL,
  Szemely_Id         INT NOT NULL PRIMARY KEY
);
```

Szerencsére a Paradigm Plus legújabb (3.5-ös) verziója ilyen trükköt nem követel, mivel ettől kezdve az attribútumoknak is be lehet állítani perzisztens tulajdonságot. Ennek a megoldásnak az mindenképpen hátránya, hogy az üzleti objektum műveleteiről is valamiképpen el kell tudnunk dönteni, hogy azok tárolt eljárásként az adatbázisba kerülnek-e.

Tisztább megoldásnak tűnik, ha külön modellezzük az üzleti objektumot és az adatbázis-táblát egy-egy osztállyal. Ez esetben az üzleti objektum tranziens. Az eljárás nélküli

perzisztens osztályok adattáblaként jelennek meg, a szerepük modell, és valamennyi attribútumukból oszlop keletkezik az adattáblában.

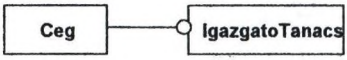



```
CREATE TABLE Szemely(
  nev                   VARCHAR(50) NOT NULL,
  szuletesiDatum        DATE NOT NULL,
  Szemely_Id            INT NOT NULL PRIMARY KEY
);
```

Ha egy perzisztens osztálynak műveletei is vannak, akkor az az adatbázisban **tárolt eljárásként** jelenik meg. Ekkor az osztály vezérlő szerepet játszik. Az attribútumai package szintű változók, kurzorok, típusok, konstansok, stb, a műveletei pedig procedúrák vagy függvények. Minden művelethez tartozik egy dokumentumfájl, amelyben az eljárás törzse szerepel. A csomag deklarációja tehát automatikusan generálható.
A másik nagyon fontos terület az **osztályok közötti kapcsolatok** leképezése az adatbázisba. Az OMT az osztályok között három alapvető kapcsolatot ismer: az asszociációt, az aggregációt és a generalizációt. Az asszociációk a szokásos 1:1, 1:n, m:n kapcsolatoknak feleltethetők meg, s ily módon idegen kulcsokként jelennek meg az adatbázisban. Pontosabban ez az 1:n, m:1 kapcsolatokra igaz.


```
CREATE TABLE Szemely(
  nev                   VARCHAR(50) NOT NULL,
  szuletesiDatum        DATE NOT NULL,
  Szemely_Id            INT NOT NULL PRIMARY KEY
);
ALTER TABLE Szemely ADD Ceg_Id INT NOT NULL;
ALTER TABLE Szemely FOREIGN KEY fk_sze_c (Ceg_Id)
  REFERENCES Ceg ON DELETE RESTRICT;
```

Az 1:1 kapcsolatokat az adatbázis generálása előtt felül kell vizsgálni, és vagy össze kell vonni a táblákat, vagy az asszociáció egyik oldalát opcionálissá kell tenni. (Különben az alkalmazás során beleütközünk a tyúk és a tojás problémájába, egyik táblába se tudunk adatot beszúrni, mivel a másikban nincs meg a megfelelő sor).




```
CREATE TABLE Ceg(
... ...
  Ceg_Id                INT NOT NULL PRIMARY KEY
);
ALTER TABLE Ceg ADD IgazgatoTanacs_Id INT;
ALTER TABLE Ceg FOREIGN KEY fk_ceg_i (IgazgatoTanacs_Id)
  REFERENCES IgazgatoTanacs ON DELETE RESTRICT;

CREATE TABLE IgazgatoTanacs(
... ...
  IgazgatoTanacs_Id     INT NOT NULL PRIMARY KEY
);
ALTER TABLE IgazgatoTanacs ADD Ceg_Id INT NOT NULL;
ALTER TABLE IgazgatoTanacs FOREIGN KEY fk_it_ce (Ceg_Id)
  REFERENCES Ceg ON DELETE RESTRICT;
```

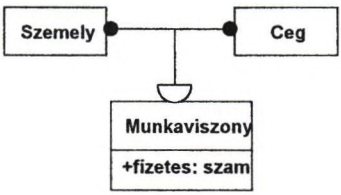
Bonyolultabb kérdés az **m:n kapcsolatok** esete. Gyakorlatunkban az m:n asszociációkat csak akkor szükséges explicit módon feloldani, ha az asszociációnak további jellemzőit is meg akarjuk adni. Azaz ha csak az a fontos, hogy adott személy mely cégeknél dolgozik illetve adott cégnél kik dolgoznak, akkor elegendő egyszerű m:n asszociációval modelleznünk.

Ez a kapcsolat az adatbázisban egy újabb táblaként jelenik meg, amelynek attribútumai a "Szemely" és a "Ceg" azonosítója, valamint a saját objektumazonosítója.

```
CREATE TABLE dolgozik (
  dolgozik_Id           INT NOT NULL PRIMARY KEY
);
ALTER TABLE dolgozik ADD Ceg_Id INT NOT NULL;
ALTER TABLE dolgozik FOREIGN KEY fk_dol_c (Ceg_Id)
  REFERENCES Ceg ON DELETE RESTRICT;
ALTER TABLE dolgozik ADD Szemely_Id INT NOT NULL;
ALTER TABLE dolgozik FOREIGN KEY fk_dol_s (Szemely_Id)
  REFERENCES Szemely ON DELETE RESTRICT;
```

```
CREATE UNIQUE INDEX dolgIDX1
  ON dolgozik (Ceg_Id, Szemely_Id);
```

Ha azonban további tulajdonságai is fontosak a "Ceg"-"Szemely" kapcsolatnak, például a fizetést is nyilván akarjuk tartani, akkor ezt egy asszociációs osztályban adhatjuk meg.




```
CREATE TABLE Munkaviszony(
  fizetes                NUMBER NOT NULL,
  Munkaviszony_Id        INT NOT NULL PRIMARY KEY
);
ALTER TABLE Munkaviszony ADD Ceg_Id INT NOT NULL;
ALTER TABLE Munkaviszony FOREIGN KEY fk_mv_ce (Ceg_Id)
  REFERENCES Ceg ON DELETE RESTRICT;
ALTER TABLE Munkaviszony ADD Szemely_Id INT NOT NULL;
ALTER TABLE Munkaviszony FOREIGN KEY fk_mv_sz (Szemely_Id)
  REFERENCES Szemely ON DELETE RESTRICT;
CREATE UNIQUE INDEX munkaIDX1
  ON Munkaviszony (Ceg_Id, Szemely_Id);
```

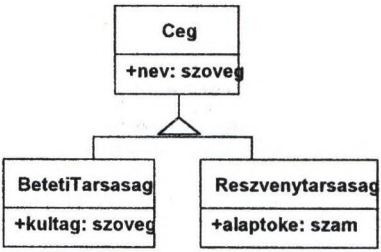
Az **aggregációk** értelmezése teljesen hasonló az asszociációkéhoz, mindössze két megkötésünk van:

- egy aggregáció soha nem lehet m:n kapcsolat,
- az aggregált osztályra nem hivatkozhatnak más osztályok.

Érdekesebb kérdés a **generalizációk** leképezése relációs adatbázisba. Erre is többféle megoldás kínálkozik. Az egyik lehetőség, hogy egyetlen táblában tároljuk az objektum-hierarchia összes osztályát, az attribútumokat egyesítjük. Ennek az a nyilvánvaló hátránya, hogy minden sorban lesznek olyan oszlopok, amelyek üresen maradnak, illetve a leszármazottakban a mezők kötelező kitöltését adatbázis-szinten nem lehet biztosítani. (Az attribútumok esetleges újradefiniálása pedig ezzel a módszerrel egyáltalán nem oldható meg.)

A másik lehetőség ennek a fordítottja, az összes leszármazottból keletkezik egy-egy tábla, és ezen táblákba beletesszük az ős összes attribútumát. Ebben az esetben viszont az őshöz tartozó attribútumok unicitását kell feláldoznunk.

A harmadik, általunk leggyakrabban alkalmazott módszer esetén mind az ősből, mind a leszármazottakból egy-egy tábla keletkezik. Ezen táblákban egy-egy objektum-azonosító szerepel, az ős és a leszármazott esetében ezen két azonosító értéke megegyezik.




```
CREATE TABLE Ceg(
  nev                   VARCHAR(50) NOT NULL,
  Ceg_Id                INT NOT NULL PRIMARY KEY
);

CREATE TABLE BetetiTarsasag(
  kultag                VARCHAR(50) NOT NULL,
  BetetiTarsasag_Id     INT NOT NULL PRIMARY KEY
);
ALTER TABLE BetetiTarsasag FOREIGN KEY fk_bt_ce (BetetiTarsasag_Id)
  REFERENCES Ceg ON DELETE RESTRICT;

CREATE TABLE Reszvenytarsasag(
  alaptoke              NUMBER NOT NULL,
  Reszvenytarsasa_Id    INT NOT NULL PRIMARY KEY
);
ALTER TABLE Reszvenytarsasag FOREIGN KEY fk_rt_ce
  (Reszvenytarsasag_Id) REFERENCES Ceg ON DELETE RESTRICT;
```

Például ha a betéti társaságok közé felvesszük a "Varázsló Bt."-t, "Óz" kültaggal, akkor a "Ceg" táblába ("Varázsló Bt", 10001), míg a "BetetiTarsasag" táblába az ("Óz", 10001) sort kell beszúrnunk.

Nem beszéltünk még az RDBMS-k egyik alapvető eszközének, a **view**-nak a modellezéséről. A view-kat olyan osztályokkal modellezhetjük, amelyek perzisztensek és view szerepet játszanak. Mivel minden osztályhoz hozzárendelünk még egy strukturált leíró állományt is, ebben az állományban definiálhatunk egy olyan szekciót is, amelyben a view-t definiáló SQL-műveletet adhatjuk meg. Ugyanezen állomány különböző szekció

szolgálhatnak arra, hogy az indexeket és a különböző constraint-eket megadjuk. Ezen állományok valójában azt a célt szolgálják, hogy a CASE eszközt a saját tervezési megoldásainkkal könnyen kiegészítsük.

Láthatjuk tehát, hogy viszonylag egyszerűen lehet objektum-orientált tervezést használni relációs adatbázison alapuló objektum-orientált környezetben készülő rendszerekhez. Így használhatjuk az objektum-orientált fejlesztés előnyeit, és külön munka nélkül generálhatunk relációs adatbázist.

Objektum-orientált CASE eszköz támogatása relációs, objektum-relációs és objektum-orientált adatbáziskezelő rendszerek segítségével

Esettanulmány

Faragó Gergely, Dr. Gajdos Sándor, Máté Attila, Moskovits Péter, Németh István, Urbán Péter, Wagner Kornél

gajdos@ttt-202.ttt.bme.hu, mosko@ttt-202.ttt.bme.hu

Budapesti Műszaki Egyetem - Távközlési és Telematikai Tanszék

A cikk az objektum-orientált, a relációs és az objektum-relációs adatbáziskezelők kínálta lehetőségeket hasonlítja össze példákon keresztül az UML objektum-orientált analízis és tervező módszertanhoz készített grafikus szerkesztő példája kapcsán. Az UML és a feladat megoldásához használt Java nyelv rövid bemutatása után a feladat során az adatbáziskezeléssel kapcsolatosan felmerült lehetőségeket, kérdéseket taglalja. Elsőként egy objektum-orientált, majd a relációs, végül pedig egy objektum-relációs alapú adatbázis objektum-orientált Java programhoz kapcsolódását elemzi. Kitér az adatbázisokhoz illesztés technikai - ODBC, JDBC - és tervezési - egységes felület az adatbáziskezelők felé, hierarchiák és öröklődés, virtualitás, stb. - kérdéseire. Végül értékeli a különböző adatbáziskezelők alkalmazásának lehetőségeit.

Kulcsszavak: objektum-orientált-, objektum-relációs- és relációs adatbáziskezelés, Java, UML

Bevezetés

Az objektum-orientált (OO) koncepciók az utóbbi években folyamatosan egyre nagyobb teret nyernek a szoftverfejlesztés számos területén. Mind gyakoribbak az olyan alkalmazások, ahol nagy tömegű adat megbízható, perzisztens tárolása is követelmény. Ezekben az esetekben az adatbáziskezelők és az OO elveken alapuló alkalmazások együttműködése kézenfekvően jelentkezik. Esettanulmányunkban arra kerestük a választ, hogy három különböző adatmodellt támogató adatbáziskezelő mennyire integrálható hatékonyan egy OO programozási nyelvvel. Vizsgálataink alapjául egy grafikus szerkesztő megvalósítása szolgált. Ez a Java nyelven készülő szerkesztő a Unified Modeling Language (UML) elnevezésű OO módszertant támogatja. A probléma felvetésével elsődleges célunk a különböző - OO, relációs és objektum-relációs (OR) - adatbáziskezelők által nyújtott lehetőségek tanulmányozása volt, nem pedig egy program (termék) vagy részének kifejlesztése.

Az OO fejlesztés lelke - talán az egyéb fejlesztési módszereknél is hangsúlyosabban - a tervezés. Az elkészítendő rendszert több aspektusból (statikus osztályszerkezet, kommunikáció-adatáramlás, vezérlés, stb.) kell elemezni ahhoz, hogy a majdan elkészülő program megfelelő minőségű lehessen. Többek között ezt a folyamatot rögzítik az OO módszertanok: pl.: OMT, ObjectOri, UML stb. Az UML ezek közül a legújabb. Jelölésében ugyan új, de lényegében a meglévő rendszerek (Booch-93, OMT, OOSE) előremutató komponenseit foglalja magába. Jelentőségét - többek között - az adja, hogy benne egységesedni látszanak az OO analízis és tervezés főbb irányzatai. Grafikus szerkesztőnk - amely egy CASE eszköz része lehet - UML-ben

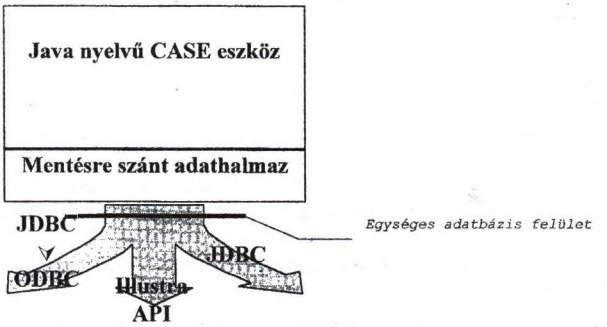
leírt rendszerek osztály-hierarchia diagramjainak szerkesztésére szolgál. A cikkben szereplő összes osztálydiagram UML jelölésekkel készült (2., 4., ábra).

A grafikus szerkesztő programozási nyelvének a Java-t választottuk. Bár a célnak elvileg bármelyik OO támogatással rendelkező nyelv megfelelt volna, a Java több szempontból perspektivikusnak tűnt. Természetesen a Java magába foglalja az OO paradigma csaknem összes olyan jellemzőjét (absztrakt adattípus, öröklődés, egységbezárás, polimorfizmus, késői kötés, stb.), ami tanulmányunk elkészítéséhez nélkülözhetetlen volt. *Platformfüggetlensége* lehetővé teszi, hogy egy heterogén számítógéppark egységesen váljék használhatóvá. *Többszálúsága* a párhuzamos programozás lehetőségét kínálja fel. Implementálhatunk *kliens-szerver* alapú, valamint *elosztott* rendszereket, támogatást kapunk *multimédia* tulajdonságok integrálására is. Számunkra mindezeken túl a legfontosabb a Java *adatbázisokhoz* való illeszthetősége volt a JDBC (Java Database Connectivity) adatbázis-illesztőn keresztül.

A tárolási feladat

A grafikus szerkesztő implementálásának kritikus része a megrajzolt ábrák grafikai elemeinek, illetve azok szemantikai tartalmának adatbázisba való elmentése volt. Ez utóbbira a diagramok programkódba való lefordításának megkönnyítéséhez van szükség. A tároláshoz alapvetően két feladatot kellett megoldani. Az első - technikai jellegű - a Java és az adatbáziskezelő közti kommunikáció biztosítása, a második az adatbázis sémák illetve az elmentés folyamatának megtervezése, kidolgozása volt.

Tekintettel arra, hogy terveink szerint három, alapvetően különböző - *relációs, OR és OO* - adatmodellnek megfelelően kell eltárolni az adatokat, az UML szerkesztő adatbázisok felé mutatott felülete egységes. Az 1. ábrán ezt a viszonyt szemléltetjük.



1.ábra: A CASE eszköz és az adatbáziskezelők kapcsolódása

Ennek az egységes adatbázis felületnek az alábbi főbb tulajdonságai vannak:

- Minden adatbázisban tárolt osztály - közvetve, vagy közvetlenül - egy perzisztens alaposztályból származik.
- Az alaposztály tartalmazza az objektumtárolás és -betöltés osztályfüggetlen részeit. Az osztályspecifikus részeket virtuális függvények valósítják meg.
- Az objektum tagváltozóinak elmentéséhez és betöltéséhez a mentés és betöltés függvények az ősosztály hasonló célú függvényeit használják.

A Java és az objektum-orientált adatbáziskezelés

Kísérleteinket az *ObjectStore* OO adatbáziskezelő rendszer Java nyelvű alkalmazások számára készített változatával, a tisztán Java-ban írott *ObjectStore PSE* (Persistent Storage Engine) *Pro*-val a végeztük. Választásunk azért esett erre az adatbáziskezelőre, mert rendelkezik az OO adatbáziskezelők [5] csaknem minden fontos tulajdonságával, ugyanakkor létezik szabadon letölthető változata is [3]. *Az ObjectStore PSE Pro*-nál a maximálisan tárolható adatmennyiség 100MB-ban van korlátozva, ami a problémánk kísérleti jellege miatt természetesen nem jelentett korlátozást.

Az ObjectStore maximálisan támogatja egy Java nyelvű alkalmazás adatainak adatbázisban való tárolását. A tárolás alapgondolata, hogy egy speciális adatbázisbeli "pontot", egy objektumot (gyökér) az adatbáziskezelőnek tárolásra át kell adni. Mindaz, ami innen öröklődéssel és referenciákkal elérhető, tárolásra kerül. Ezek után már csak annyi a feladat, hogy körültekintően ki kell választani ezt/ezeket a speciális pontot/pontokat. Ha például egy személyt és az ő gyerekeit akarjuk eltárolni, akkor létre kell hozni az objektumokat, majd a szülőhöz a gyerekeit referenciaként hozzá kell rendelni. Ezek után nincs más hátra, mint a szülőt adatbázis-gyökérként megjelölni, majd tárolásra az adatbáziskezelőnek átadni. A beolvasás hasonlóan egyszerűen történhet.

Tetszőleges osztályhierarchia leképezése relációs adatmodellre

Napjainkban a relációs alapú rendszerek meghatározó jelentőségűek. Programozói felületük a szabványos SQL felületen keresztül gyakorlatilag egységes. A nagy kérdés az, hogy a szoftvertechnológia különböző irányzatai által diktált tempóval mennyire tudnak lépést tartani. Kísérletünk a választott feladat összetettsége miatt nem pusztán a konkrét feladat megoldására irányult, hanem arra az általános kérdésre kerestük a választ, hogy hogyan lehet egy OO programozási nyelven írott adat-, illetve objektumstruktúrát a lehető legáltalánosabban relációs adatmodellre leképezni.

Míg a Java nyelven írott ObjectStore PSE igazán természetes módon kapcsolódik a Java nyelvű alkalmazásokhoz, a Java és a relációs adatbáziskezelők összekapcsolásáról külön kellett gondoskodni. A Java adatbázisokhoz kapcsolódó szabványos felülete a JDBC (Java Database Connectivity). A relációs világban általánosan használt felület az ODBC (Open Database Connectivity), melyet a legtöbb rendszer ismer. Mivel a JDBC még fiatal, az adatbáziskezelők tanulmányunk készítésekor még nem, vagy alig támogatták azt. A megoldás tehát egy JDBC-ODBC bridge alkalmazása volt. Választásunk - az amúgy nem túl bő skálából - a hordozhatóság biztosítása végett a Sun által készítettre esett [4]. A JDBC-ODBC bridge használatával kialakított környezet nagy előnye, hogy a program számára lényegtelen, hogy valójában milyen adatbáziskezelő áll mögötte, egyedül annyi szükséges, hogy rendelkezzék ODBC felülettel.

A tárolás megtervezésekor három fontos tulajdonságot kellett tekintetbe venni, melyek a komplexitás, ill. az OO-ság következményeként lépnek fel: a *hierarchikus felépítést*, az *öröklődést*, valamint a *statikus változók* kérdését.

Hierarchikus felépítés

Az objektumokat alapvetően kétféleképpen tárolhatjuk a relációs adatbázisban. Az egyik módszer szerint az öröklődési fának csak a levelei kerülnének bele a táblákba, a közbenső öröklődési lépcsőket nem tárolnánk. Ezzel a módszerrel egyrészt az a probléma, hogy a relációs adatbázisban nem látszik az objektum-modell hierarchikus felépítése, így az nem reprezentálja kellő szemléletességgel az objektum-hierarchiát. A másik probléma onnan ered, hogy elképzelhető, hogy két osztály objektumai, és ezek összes leszármazottai egymással kapcsolatban állnak. Ebben az esetben az öröklődési fa levelein található leszármazottak közti kapcsolat - amit azok valamelyik ősénél definiáltunk - idegen kulcsok bonyolult hierarchiájával lenne csak megvalósítható. Gondot okozna az is, ha egy ősosztály összes objektumát szeretnénk lekérdezni, hiszen ehhez külön-külön kellene lekérnünk az egyes származtatott osztályok példányait. Ez csak igen rugalmatlanul oldható meg.

Ezen megfontolások alapján úgy érdemes eltárolni az objektumokat, hogy minden ős és származtatott típushoz (osztályhoz) létrehozunk egy relációs táblát, melyben csak az ősosztályhoz képest új tagváltozók kerülnek eltárolásra. Ez azt is jelenti, hogy egy objektumpéldány eltárolásához nem csak a saját típusához tartozó, hanem az összes felmenőjéhez tartozó táblába is kerül bejegyzés, azaz az objektum az öröklődési hierarchiának megfelelően szét van szabdalva annyi részre, amennyi az őseinek a száma, plusz egy (önmaga).

Öröklődés

Modellünkben az *öröklődést* egy `is_a` kapcsolatnak feleltettük meg. A fent leírtak szerint minden egyes objektum eltárolásakor az összes, ősét reprezentáló táblába is kerül bejegyzés. Ahhoz, hogy egy objektum összes attribútumát (adattagját) összegyűjtsük a táblákból, az objektum őseit reprezentáló táblákból az objektumhoz tartozó sorokat is ki kell nyerni. Ebből következik, hogy az objektum levélben lévő része pontosan hozzárendelhető az ős-táblák egy-egy sorához, a kapcsolat aritása 1:1. Tekintettel arra, hogy az objektumnak a táblákból történő összeállítása általában az objektum levél részétől történik, az idegen kulcsot mindig a leszármazottnál érdemes tárolni. Az összeállítás a levélből kiindulva az ős táblákon felfelé haladva az összes kapcsolódó információ kigyűjtésével történik.

Statikus változók

Külön megfontolást igényel az OO terminológia *statikus változó* fogalma illetve azok modellünkre történő leképzése. A statikus változó durván annyit jelent, hogy lehetnek olyan változók, amelyek egy adott osztályba tartozó objektumok számára közösek. Ha az egyik objektum megváltoztatja az értékét, a másik rögtön látja ezt. Analóg a procedurális nyelvek globális változó fogalmával, a globalitás esetünkben az osztályra vonatkozik. Megvalósítására két megoldás kínálkozik. Az egyik szerint minden objektumhoz tároljuk - nyilván az osztályhoz tartozó tábla minden sorában ugyanazt - az értéket. Ez azzal jár, hogy ha valamelyik objektum megváltoztatja a statikus változó értékét, akkor minden sorban egyből változtatni kell. A másik megoldás, hogy minden olyan osztályt reprezentáló táblához, ahol statikus változó van, létrehozunk egy rögzítetten egyetlen sorból álló táblát, amiben a statikus változókat tároljuk. Ezzel elkerülhetjük a frissítési anomáliát.

Az első megoldás esetén a problémát - a redundáns tárolás mellett - az jelenti, hogy amellett, hogy a változó értékének módosítása esetén atomi művelettel kell megváltoztatni az összes, ugyanezen változót reprezentáló sor megfelelő attribútumát. A második megoldás -

egysoros táblák létrehozása - nem megszokott. Számunkra didaktikailag mégis tisztábbnak tűnt, ezért ezt választottuk. Megjegyezzük azonban, hogy ha csak egy mód van rá a konverzió nehézkessége miatt érdemes elkerülni a statikus változók használatát.

Objektumok mentése - visszatöltése

A *mentésnek* fontos része még az objektumok szétdarabolása során használandó idegen kulcsok generálása és megfelelő helyekre történő behelyettesítése. Az idegen kulcsoknak az objektumok és az őseik összerendelése során jut szerep. Mivel az elmentendő hierarchiában megkülönböztetünk objektum-példány és annak részei, valamint objektumok közti kapcsolatokat, a mentés - ennek megfelelően - két részből áll.

Az első rész az objektum-hierarchia táblákba képezése. Mindez a levéltől az ősökön fölfelé haladva rekurzív módon, történik:

Vesszük a következő objektumpéldányt. Erre alkalmazzuk a *save* függvényt, ami ha az adott szintnek még van őse, akkor meghívja a *save* függvényt, ha nincs, akkor pedig táblába írja a szükséges adatokat. Az egyedi azonosító generálása is ebben a lépcsőben történik, amit visszaad a rekurzióban visszafelé lépkedő *save* függvénynek.

Ez vázlatosan így néz ki:

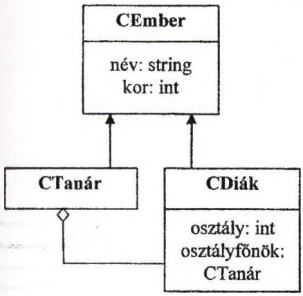
```
first=true

id save(boolean first)
{
     if(parent exists)
          {parentid=parent.save(false)}
     id=savetable(first)
     return id
} /* save */

id savetable(boolean first)
{
     if(first) {ehhez a sorhoz tartozik egy objektum}
          // ez a visszatolteshez kell
     id=azonosito generalasa
     mentes a tablaba
     return id
} /* savetable */
```

A mentés második fázisában az objektumok közötti kapcsolatok letárolása a feladat. Ehhez végig kell ismét mennünk az összes példányon, és ahol van másik példányra hivatkozás, oda idegen kulccsal annak a példánynak az első fázisban már legenerált azonosítóját írjuk be.

A *visszatöltéshez* minden tábla sorához kell plusz egy bit (ld. példány mező a 3. ábrán), amelyik megmondja, hogy az a sor egy példányhoz tartozik-e, vagy pedig egy példány őse (ez természetesen még a mentésnél kerül beállításra). Itt is két lépésre van szükség. Először végigmegyünk az összes táblán. Ha az adott sor egy példány, akkor létrehozunk egy olyan objektumot és az őstáblákból feltöltjük az adatmezőit. A más példányokra hivatkozó referenciákkal itt nem törődünk. Ezek után újból végigmegyünk az összes táblán, és ha van idegen kulcs, akkor az ahhoz tartozó példány referenciáját betesszük a hozzá tartozó változóba.

Példa:

Tekintsük a következő objektum-hierarchiát:



Jelölés:
Nyíl: öröklés - gyerektől a szülő felé
Kapcsolat: több (*) : egy (◇)

2.ábra: Öröklődés UML-ben leírva

```
class CEmber
{
    String          név;
    int             kor;
}

class CTanár <- CEmber
{
}

class CDiák <- CEmber
{
    int             osztály;
    CTanár          osztályfőnök;
}
```

Az adatbázisba mentés a fent leírtak alapján történik. A mentéshez létrehozott relációs séma:

```
CEmber (int id, boolean példány, char[50] név, int kor)
CTanár (int id, boolean példány, int superid)
CDiák (int id, boolean példány, int superid, int osztály, int ofőid)
```

Az adatbázis feltöltése:

```
CEmber ember1=new CEmber("Attila",22);
CEmber ember2=new CEmber("Peter",25);
CTanar tanar1=new CTanar("Sandor",35);
CDiak diak1=new CDiak ("Kornel",22,4,tanar1);
CDiak diak2=new CDiak ("Peter",23,3,tanar1);
```

A relációk tartalmát az eltárolás után (kiemelve az idegen kulcsok által mutatott mezőt) 13. ábra mutatja:

CEmber

id	példány	név	kor
1	T	Attila	22
2	T	Peter	25
3	F	Sandor	35
4	F	Kornel	22
5	F	Peter	23

CTanár

id	példány	superid
6	T	3

CDiák

id	példány	superid	osztály	ofőid
7	T	4	4	6
8	T	5	3	6

3. ábra: Öröklődés ábrázolása a relációs konverzió után

A relációs séma minden táblájában megtalálható az egyedi azonosító (id), és az a bit, ami az adott relációbeli sorról azt mondja meg, hogy az példány, vagy ős. A hierarchiában a nem legfelső szintet reprezentáló táblákban megtaláljuk azokat az idegen kulcsokat is, amelyek a gyerekektől mutatnak az ősök felé. A példából az is jól látszik, hogy az adattag nélküli CTanár osztály számára annak ellenére is készült egy reláció, hogy abban az osztályban nem volt változó deklarálva. Ez szükséges is, hiszen az osztályfőnökre való hivatkozás enélkül csaknem megoldhatatlan lenne.

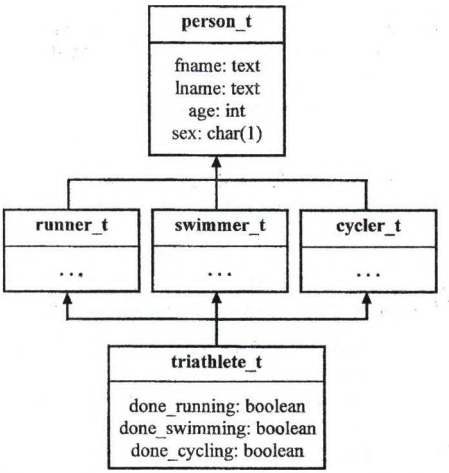
Az objektum-relációs adatmodell főbb objektum-orientált tulajdonságai

Tanulmányunkhoz az Informix cég Illustra nevű termékét használtuk.

Mivel az OR alapú rendszerek relációs modellre próbálják leképezni az OO struktúrákat, ezért megfontolásaikban nagyon hasonlítanak az általunk az előző fejezetben leírtakhoz. Itt is megtaláljuk a hierarchia ábrázolásának kétféle lehetőségét. A különbség adatábrázolás szempontjából annyi, hogy a szétszabdalt objektumokra irányuló lekérdezéseket - akár a többszörös[1] öröklődési kapcsolatokkal is - az adatbázis-szerver állítja össze és fejti ki. A többszörös öröklődés szintaktikájára érdemes egy rövid példát is tekintenünk.

Származzon a futó, úszó és a biciklis típus az embertől (person) - ezt a programkódot nem mellékeltük -, majd ettől a háromtól származzon az olyan atléta, aki fut is, úszik is és kerékpározik is (triathlete).

[1] A Java nyelv nem ismeri a többszörös öröklődés fogalmát, ezért az előzőekben ennek leképzésével nem is foglalkoztunk. A többszörös öröklődés relációs modellre való leképzése statikus változók eltárolásának mintájára plussz tábla felvételével képzelhető el. (További megfontolást igényel, ha egy alaposztály az öröklési fában több úton is elérhető.)



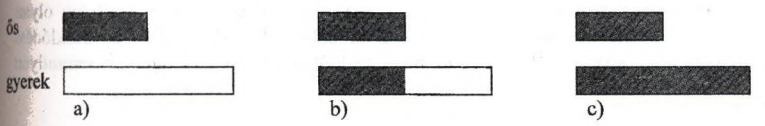
4. ábra: A többszörös öröklődés UML modellje

Származtatáshoz az under kulcsszót kell használni:

```
create type person_t(
     fname  text,
     lname  text,
     age          int,
     sex          char(1)
);

...

create type triathlete_t(
     done_running boolean,
     done_swimming boolean,
     done_cycling boolean
) under runner_t, swimmer_t, cyclist_t;
```

Az Illustra megkülönböztet típusöröklést (fent) és táblaöröklést. Az alapvető különbség a kettő között, hogy táblaöröklésnél létrejön egy adatszerkezet is, amelyben különböző - az alaptípusból származó - típusokat tárolhatunk (ennek minden további következményeivel - ld. lent), valójában azonban ugyanannak a megoldásnak a kétféle szemszögből való megvilágításáról van szó:

```
create table ős of type person_t;

create table gyerek of new type child_t(
     child_code char(3)
) under ős;
```



5. ábra: A táblaöröklléssel létrejött és a SELECT utasítás által érintett táblák

```
select * from ős;
```

Ez a lekérdezés visszaadja az ős tábla négy attribútumból álló sorait (5/a ábra), a gyerek tábla sorainak azt a négy attribútumát, amit az ős táblából örökölt, és minden további őstől leszármazott tábla őstől örökölt 4 attribútumát[2]. Ha azt akarjuk, hogy csak az alaptábla sorai jelenjenek meg, az only kulcsszót kell használnunk (5/b ábra):

```
select * from only(ős)[3];
```

Az ún. korrelációs változó használatával egy ősosztály összes leszármazottjának teljes egészében történő listázása oldható meg (5/c ábra). A

```
select p from ős p;
```

lekérdezés az ős, a gyerek és az esetleg belőlük származott összes sort adja vissza. Az ilyen típusú lekérdezés akár különböző típusú sorokat is adhat vissza (*jagged rows*).

Lehetőségünk van a relációs modellben megszokottnál rugalmasabb sémadefinícióra; terminológiájában és tulajdonságaiban nagyon hasonlít az OO tervezésre. Az adattípusok az öröklődésen túl rekurzív módon tartalmazhatják is egymást. Az Illustra adattípusai lehetnek *konstruált* (tömb, halmaz, referencia), *összetett* (C-struct jellegű), valamint *alap típusok* (char, integer, real, stb.) tetszőleges komplexitásban.

További, az objektum-orientáltságra nagyon is jellemző tulajdonságokat is implementáltak az Illustrában. Lehetőség van a polimorfizmus, valamint a *függvények késői kötésének* kihasználására. A redundáns tárolás elkerülésére használhatunk *származtatott attribútumokat*. A származtatott oszlopot egy függvény implementálja, így valójában nem tárolódik, értéke mindig érvényes.

Összegzés

A Java az elosztott, platformfüggetlen, hálózati, multimédia alkalmazások ígéretes programozási nyelve, várhatóan nagy jövőnek néz elébe. Ezért különösen fontos, hogy mennyire illeszthető meglévő adatbáziskezelőkhöz, azok közül is melyiket érdemes használni.

Tapasztalataink szerint a Java nyelvhez legegyszerűbben az ObjectStore PSE kapcsolódott. Teljesen elrejtve a programozó elől, hogy a mélyben mi is történik oldja meg a tárolási ill. lekérdezési feladatot.

Bonyolult, ám kifejezetten érdekes a feladat, ha relációs adatbáziskezelővel kell az OO nyelven írott alkalmazás adattárolását megoldani. A feladat nemcsak a két látszólag alapvetően ellentétes adatfelfogás összeegyeztetése miatt magával ragadó, hanem amiatt is, hogy a rendszer tervezői a két rendszer előnyeinek és hátrányainak, valamint a megoldandó problémának ismeretében több helyen is a megoldás "szépségét" figyelembe véve dönthetnek. Talán ez is az oka, hogy az OR adatbáziskezelők kicsit különböznek egymástól, helyenként ugyanarra a részkérdésre más és más megoldást kínálnak.

Nem nehéz a helyzetünk, de alapos végiggondolás szükséges akkor is, ha OR adatbáziskezelő áll rendelkezésünkre. Az Informix cég Illustra nevű termékét jónéhány olyan tulajdonsággal ruházták fel, ami megkönnyíti az OO modellezést: API szintjén az öröklődési kapcsolatok közvetlenül ábrázolhatók. Hátránya, hogy az SQL-en kívül nem támogat semmilyen

---

[2] Ez a tárolási, illetve lekérdezési mód teljes egészében megfelel a korábban említett CEmber tábla szerkezetének illetve listázásának.

[3] Az only kulcsszó a CEmber tábla *példány* attribútumának figyelembevételével történő listázást jelent.

szabványos adatbázis hozzáférési felületet. Az OO adatbáziskezelőkkel szemben előnye, hogy általánosabb - relációs - lekérdezésekre is lehetőséget biztosít, ugyanakkor nehézkesebb az objektum-hierarchiák kezelése.

Kapcsolódó irodalom, források

[1] *Illustra Users's Guide*

[2] Grady Booch, Ivar Jacobson, James Rumbaugh: *The Unified Modeling Language for Object-Oriented Development*. Documentation Set. Version 0.91 Addendum. UML Update.

[3] ObjectStore homepage: http://www.odi.com/

[4] Official Javasite: http://www.javasoft.com/

[5] Moskovits Péter: *Objektum-orientált adatbáziskezelő rendszerek*. Kecskemét, 1996. I. Országos Objektum-orientált Konferencia. (Elérhető a *http://gedeon.ttt.bme.hu/~mosko/interest.html* címen.)

A cikkel kapcsolatos információk a *http://gedeon.ttt.bme.hu/~mosko/interest.html* címen érhetők el.

AZ INFORMIX által támogatott OLAP eszközök

dr. Balogh Kálmán

Informix Technology Center Hungary
1063 Budapest
Bajnok u. 13.
Tel: (06-1)-302-33-88/117
Fax: (06-1)-302-33-95

E-mail(Internet): kbalogh@informix.hu

Bevezető

- dimenzionális modell
  - szükségessége
  - tény- ill. dimenziótáblák
  - dimenzióelemek és dimenzió attribútumok
  - 'csillag' és 'hópihe' séma

- OLAP felhasználói elvárások
  - természetes üzleti fogalmak használata
  - az adatok ad-hoc elérhetősége
  - nagy teljesítmény
  - komplex analízis lehetősége
  - könnyű használhatóság, tanulhatóság
  - a meglévő PC-s eszközök kihasználása

- Technikai elvárások
  - karbantarthatóság, kiterjeszthetőség
  - skálázhatóság
  - nyíltság különböző front-end eszközök felé

A MetaCube technológiája - az elvárások teljesülése

- teljesítménnyel kapcsolatos kérdések
  aggregátumok kezelése, háttér feldolgozás
- lehetőségek az SQL korlátain túl
  - összehasonlítások
  - 'top line reporting'
  - speciális időalapú analízis
- üzleti fogalmak használata
- dinamikus 'metaadat'
- a MetaCube architektúrája
- OLE2 támogatás
- szoftverkomponensek
  - MetaCube Explorer
  - MetaCube for Excel
  - MetaCube Warehouse Manager
  - MetaCube Warehouse Optimizer
  - MetaCube Agents
    architektúra

Business Objects

Windows NT 4 kontra NetWare 4.11

Babócsy László
BKV Rt Informatika

A Windows NT 4 megjelenése óta számtalan összehasonlítás látott napvilágot, mely a NetWare 4-es rendszerével hasonlítja össze. Jelen értekezés is ezt célozza, bár egy kissé szakítva az eddig megjelent összehasonlítások metodikájával.
Az összehasonlítás alapja a két rendszer azonos szolgáltatásainak, felépítésének összevetése és nem a táblázatba foglalt és kipipált vagy hiányzó megfeleltetés.
Előljáróban azonban egy fontos tulajdonságot kell tisztázni. A Windows NT alapvetően a UNIX rendszer Microsoftos megfelelője. A felépítése és működése is az alkalmazás szerverek - mint a UNIX - tulajdonságait viseli. Tulajdonképpen tehát nem is a Windows NT és NetWare hanem a Windows NT kontra UNIX és NetWare összehasonlítás lenne a teljes spektrum. Jelenleg csak a fájl és nyomtató szolgáltatások összehasonlítására szorítkozom. Az összehasonlítás másik aspektusa mindkét rendszerrel együtt szállított Internet/Intranet-es környezet. A NetWare-t is gyakorlatilag IntraNetware-ként kellene összevetni, de jelen összehasonlítás nem terjed ki a rendszerek ilyen irányú kiterjesztéseinek összevetésére sem.

Az összehasonlítás részei:
1. **Rendszer felépítés**
2. **Címtár - tartomány szervezés**
3. **Fájl és nyomtatás szerviz**
4. **Hibatűrés**
5. **Titkosság**
6. **Távoli elérés**
7. **Hálózati tulajdonságok**
8. **Kereszt támogatás**
9. **Egyéb**
10. **Összefoglalás**

Vizsgáljuk meg az egyes területeket!

Rendszer felépítés

NetWare 4
A NetWare 32 bites felépítéssel az elsők között jelent meg. A 32 bites felépítésnek köszönhetően sebességben az Intel alapú PC-k számára megfelelő platformként szolgált az alkalmazás szerverek kategóriában is, mint pl. az OracleWare példája is mutatja. A felépítését tekintve egy kernelhez csatlakoznak a különböző NLM-ek, melyek a NetWare specialitásai. Ezek a betölthető modulok a moduláris működés alapjait jelentik, hiszen csak azok a modulok kerülnek betöltésre, melyek a rendszer működéséhez elengedhetetlenek. Az Intel alapú működés fontos tulajdonsága még, hogy a kernel és az NLM-ek is a processzor 0 ringjén futnak. A védett működéshez a 2-es ring használata szükséges. Ez azt jelenti, hogy az a program, mely a 0-s ringen fut, képes a rendszer működését leállítani, amelyik pedig a 2-esen az nem. Ez utóbbi esetben csak a program működése áll meg. A NetWare-ben alapvetően minden NLM a 0 ringen fut, de lehetőség van a 2-es ring

használatára is, ha még nem teljesen betesztelt NLM-et kíván futtatni. A sebességbeli különbségek még jelentősek, a 0-ás ringen futó processz javára. Képes az SMP keretén belül 32 processzor támogatására.

Windows NT

A Windows NT is 32 bites rendszer, de mikrokernel architektúrájú. Ez azt jelenti, hogy a rendszer egy kis része, a kernel fut csak a 0-és ringen, a többi rész a 2-es ringen fut. A processzor függetlenség is könnyen megvalósítható, hiszen csak a mikrokernel kis része íródott assemblerben, a többi része C-ben, melyet könnyű más processzorra is átrakni. A Windows NT nem csak Intel hanem RISC, PowerPC stb. Processzorokon is fut, bár a nagyszámú installáció az Intel platformon történt. A felépítéséből fakadóan a működése megbízható, ez már nem az a Windows amit esetleg megszokott. Ha Windows NT -ről beszélünk alapvetően két rendszerről beszélhetünk, ha nem pontosítjuk, mivel a Windows NT lehet munkaállomás és lehet szerver is. Az előbbit a nagyteljesítményű munkaállomások operációs rendszerének szánják, míg az utóbbit a hálózati erőforrás-szerverek működtetésére alakították ki. Mindkét változat ellátja az alapvető operációs rendszer feladatokat, a Windows NT Szervert azonban kiegészítették néhány további hálózati funkcióval.

A Windows NT szerver és a Windows NT munkaállomás azonos kernellel rendelkezik. A különbség a különböző működés módok illetve funkciókból eredő hangolásban rejlik. A különbségek részletezése a 1. Táblázatban található.

Jellemző	NT Munkaállomás	NT Szerver
Memóriaigény	Javasolt: 16 MB vagy több	Javasolt: 32 MB vagy több
Processzorok száma (SMP)	1-2	1-32
Hibatűrő lemezkezelés	—	Tükrözés, duplikálás, szoftveres RAID 5
Bejövő telefon-kapcsolatok	1	256
Fájl- és nyomtató-megosztás	Legfeljebb 10 egyenrangú kapcsolat.	A hozzáférési licencek (CAL- ok) számától függ.

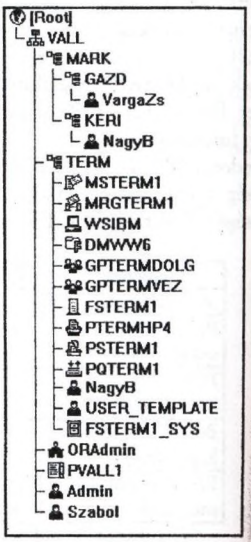
1. Táblázat Windows NT szerver és munkaállomás összehasonlítás

A Windows NT munkaállomást úgy hangolták, hogy a lehető legjobb teljesítményt biztosítsa a lokálisan bejelentkezett felhasználó számára. Ezzel ellentétben a Windows NT szerver beállításai azt célozzák, hogy a hálózati felhasználók kiszolgálásakor nyújtsa a maximumot. A billentyűzet és az egér figyelése, a gyors grafika kevésbé fontos egy kiszolgálón, mint a fájlok és nyomtatók megosztott használatának biztosítása, a hálózati alkalmazások nagy sebességű végrehajtása.

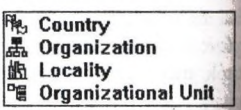
A Windows NT 4 új tulajdonsága még a Win32 alrendszer megjelenítés részének 2 ringből a 0-s ringbe költöztetése. Ezáltal azonban csak a grafikus munkaállomásként használt gépek teljesítménye növekedett, a szerverek számára a grafikus megjelenítés ideje nem döntő. A grafikus képernyő a szerver oldalon erőforrásokat köthet le, amelyek a kiszolgáló teljesítmény hangolásánál figyelembe kell venni.

A legnagyobb különbség a két rendszer között a címtár és a tartomány szervezésben van jelenleg, ezért ezek bemutatásának szentelem a legnagyobb teret.

A címtár szolgáltatással jelenleg csak a NetWare 4 rendelkezik. Az NDS, a Novell Directory Services jelenleg a legszélesebb körben használt hálózati címtár szolgáltatás, rövidesen több platformon is elérhető, többek között Windows NT -n is.
Az NDS a NetWare 4-es legfontosabb újdonsága. Az NDS egy információs adatbázis a hálózati környezet információinak tárolására, hozzáférésére, karbantartására és használatára. A NDS tehát egy objektum orientált címtár szolgáltatás implementáció, mely követi a nemzetközi szabvány, az X.500 előírásait. Az NDS architektúrája gondoskodik róla, hogy az információs rendszerünket egy teljes rendszerként érzékeljük, függetlenül attól, hogy mely része hol helyezkedik el. Lehetővé teszi a korábbi szerver, szerverek és a szerverenként definiált felhasználók metodika megszűnését, és a hálózatba történő bejelentkezéskor a hálózati jogaink birtoklását a teljes hálózatra vonatkoztatva, a hálózat összes szerverén, globálisan. A globalitás azt jelenti, hogy egy hálózaton belül az összes szerver megosztva használ egy adatbázist.



A címtár adatbázis tehát a logikai erőforrások összessége egy hálózatra vonatkoztatva.
A címtár fa: Az NDS egy hierarchikus, többszintű felépítésű fa struktúrát alkot. A fa struktúrában az objektumokat rendszerezzük.
Az NDS séma: Az NDS séma a címtár fa felépítésének szabályait tartalmazza. A séma definiálja az információk típusát és tárolásának módját. Az alap séma (basis schema) a rendszerrel kapott objektumleírások összefoglaló neve. Ezt az objektumgyűjteményt - ha szükséges - módosíthatjuk, és kibővíthetjük.
Objektumok: A címtár objektumok az információs alapegységek. Egy objektumban definiált a típusa és tartalmazza a tulajdonságai összetevőit (properties) és a tulajdonságait leíró adatokat. Ezek az információk tárolódnak a címtár adatbázisban.
Az objektumok háromfélék lehetnek, a [Root], a konténer objektum (container object) és a levél (leaf) objektum.
A konténer objektum tartalmazhat további konténer objektumokat és levél (leaf) objektumokat is, segítségével lehet hierarchiába szervezni az objektumokat. A konténer objektumra vonatkoztatott jogosultságok minden, a konténerben szereplő objektumra igazak. A képen látható konténer objektumok részei az alap NDS sémának. A Levél objektumok nem tartalmazhatnak további objektumokat. Az egyedi objektumokat reprezentálják, mint a felhasználók, nyomtatók, szerverek stb. Konténer objektumon belül készíthetünk levél objektumokat.



Minden levél objektumnak van egy neve, a CN (Common Name). A felhasználói objektum esetén a CN a login név.
A címtár természetesen a köteteket is leképezi, a kötet objektumban. A kötet objektum a kötet tulajdonságait írja le, mint pl. a szerver és a fizikai kötet neve.

A kötet objektum neve alapértelmezetten a szerver nevéből, egy aláhúzás karakterből és a kötet fizikai nevéből áll.

A címtár felépítéséből , kiterjedt elhelyezkedéséből fakadóan biztosítani kell a különböző szervereken keresztüli megfelelő működést, több földrészen keresztül is. Ezt biztosítják az alábbi lehetőségek is.

A szerverek belső óráit **szinkronizálni** kell, mivel a több szerveren történő egyidejű munkavégzés eredménye, az adatbázis integritás csak így biztosítható.

Minden NDS adatbázis művelethez hozzákapcsolódik az elvégzésének ideje is, ezt idő-bélyegnek (time stamp) hívják. Az idő-bélyegek biztosítják a nagytávolságú összeköttetések esetén is a korrekt módosítások végigvezetését. Ez különösen több földrészt átívelő hálózatoknál lehet nagy probléma.

Az idő szinkronizálás az ún. idő szervereken keresztül történik. A szerverek belső óráit szinkronozza a szolgáltatás, ezen keresztül a bejelentkezéskor a munkaállomások órái is erre szinkronizálnak.

A **partíciók** az NDS adatbázis logikai részei. Az NDS információinak partíciói a felhasználók számára láthatatlanok, az egész NDS-t egy egységként látják. A partíció a címtár fa egy részfája vagy ága.

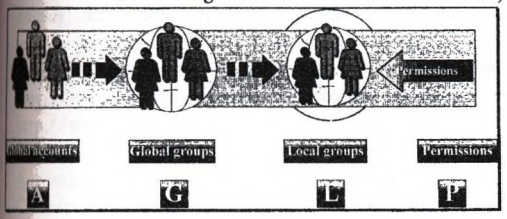
A **replikák** az NDS adatbázis partícióinak fizikai másolatai. Segítségével hibatűrő képességekkel rendelkezik az NDS (a másolatról a sérült előállítható), másrészt az adathozzáférést gyorsíthatjuk meg segítségével nagytávolságú és kis sebességű vonalak esetén. A replika nem vonatkozik az állomány rendszerre.

Tartomány (domain)

A tartomány egy valamilyen szempontból összetartozó gépek speciális halmaza. Ezek a gépek egy hálózaton keresztül képesek kommunikálni egymással, de főleg egy kitüntetett géppel, a tartomány vezérlővel. A jogosultságok egy speciális adatbázisban, a SAM (Security Accounts Manager)-ben tárolódnak, a tartomány vezérlőn. A felhasználók bejelentkezése sem a saját gépre történhet, hanem a tartományba. A bejelentkezési jogosultság vizsgálat ezután a tartomány vezérlőn történik. Lehetőségünk van egy gépről több tartomány elérésére is, de csak ha ezek között speciális kapcsolat van (trust) .

A tartomány adatok természetesen a tartomány vezérlőn tárolódnak. Az adatok biztonsága érdekében azonban nem elég egy szerveren tárolnunk ezeket. A SAM adatbázis tehát installáláskor készül el, ha PDC-t választunk. Tehát egy szerverről az installáláskor dől el, hogy PDC vagy BDC lesz, vagy csak a hálózatban egy szerver, mely nem tartalmaz SAM adatokat.

A meghatalmazás (trust) gyakorlatilag azt jelenti, hogy egy tartományban lévő erőforrások (resources) használatára meghatalmaz valakit egy másik tartományból. A fiók (account) egy felhasználói fiókot jelent. Lehetőség van egyirányú és kétirányú (két egyirányú) meghatalmazás készítésére is. Ezen a meghatalmazáson keresztül lehet bejelentkezni, erőforrásokat elérni több tartományon keresztül is. Mivel az erőforrások megosztása nem része a tartomány adatbázisnak, ezért a megfelelő szervezéssel segíthetünk a jogok kézben tartása érdekében, ami lényege, hogy felhasználókat csak globális fiókokhoz rendel, ezeket globális csoportokba helyezi,



NT a FAT fájl rendszerrel nem képes a jogosultságok kezelésére a saját gépen, ezért az NTFS fájl rendszer használható a megfelelő biztonság érdekében. Az NTFS használata a nagy lemezek esetén is indokolt, bár a lemez szektorcsoport beállítása nem lehetséges, a szektorcsoport szuballokációját sem lehet beállítani. A tömörítés a megszokott és bevált röp-tömörítés alapú, de nem lemezekre, hanem fájlokra, könyvtárakra vonatkoztatva. Az NTFS fájl szerveren lassabb, mint a FAT, mivel nehezen cache-elhető. Lehetőség van azonban a fájl rendszeren belüli replikálásra, mely adott könyvtárakban található fájlokra terjed ki és automatikusan a változásokat is követi.

A nyomtatás szerviz is hasonló, a kliens a nyomtatókra küldött munkáit a megosztott nyomtató kinyomtatja. Természetesen kisebb különbségek találhatók a két nyomtatási metódus között.

Hibatűrés

A hibatűrés címszóval konkrétan a lemezes rendszerek, és a fájl szerver védelmét értem.

A lemezes rendszerek esetén megtalálható mindkét rendszernél a tükrözés, a duplikálás. A Windows NT képes szoftveres RAID5 megoldásra is, ami azonban csak elvi előny, hiszen a hardveres megoldások sokkal gyorsabbak. A NetWare régóta rendelkezik hibatűrő megoldásokkal, melyek közül az SFTII most is kiemelkedik. Ez gyakorlatilag azt jelenti, hogy két párhuzamosan kapcsolt szerver dolgozik, ha a fő szerver meghibásodik, a tartalék átveszi a szerepét. A NetWare rendelkezik a TTS-sel amely egy tranzakció kezelő rendszer. A Windows NT -ben a fürtözési megoldás lesz ehhez hasonló, amely most még csak 3. Party megoldásban kapható. A UNIX-os rendszerek tulajdonsága a fürtözés és kiterjedhet a tartalék gép erőforrásainak használatára is.

Titkosság

A számítógéppel szemben támasztott biztonsági követelmények a meghibásodás, ellopás vagy illegális használat elleni védelmet jelentik. A védelem kiterjedhet egy vagy több számítógépre és beletartozik a számítógép részeinek, a hardver, szoftver valamint a tárolt adatok biztonságára is.

Az Egyesült Államokban a Védelmi Minisztérium (Department of Defense (DOD)) National Computer Security Center (NCSC) 1983-ban közzétett TCSEC azaz (Trusted Computer System Evaluation Criteria) azaz Megbízható Számítógép Rendszerek Értékelési Szempontjai, ismertebb nevén az Orange Book vagyis Narancs Könyve tartalmazza azokat a szempontokat, melyeket a számítógép rendszereknek teljesíteniük kell a megfelelő biztonsági előírások érdekében. A Red Book vagyis Vörös Könyv tartalmazza a számítógépes rendszerek hálózati kiterjesztését, tehát a Narancs könyv hálózati kiterjesztése. Létezik még a Blue Book, vagyis Kék könyv, mely a Narancs könyvben meghatározott rendszerek alrendszereire vonatkozik. A biztonsági előírásokat osztályokba sorolták, a főbb osztályok az A, B, C, D osztályok. A fokozott biztonság az A osztálynál jelenik meg, míg a D osztály a legkevésbé megbízható rendszer. A főbb osztályokon belül további alosztályok kerültek meghatározásra. A teljes struktúra az alábbi.

- **A1** Ellenőrzött tervezés
- **B3** Védett, titkos tartományok
- **B2** Strukturált védelem
- **B1** Megjelölt Titkosítási védelem
- **C2** Vezérelt Hozzáférés védelem

GSNW szolgáltatás használatával. Segítségével a kliens oldalról elérjük a vegyes hálózatunkban található Windows NT és NetWare 2.x,3.x,4.x szervereket, és a fájl és nyomtató megosztást használhatjuk mindkét szerverről, valamint a Windows NT alkalmazás szerver tulajdonságát is kihasználhatjuk. A **Gateway Services for NetWare** (GSNW) a Windows NT -t használja mint gateway, tehát a kliens oldalról a NetWare szervereket a Windows NT -n keresztül érhetjük el. Ez a megoldás ideális a 3.x -es NetWare szerverek központi menedzselésére, mivel a szolgáltatásba a felhasználói menedzsment is benne foglaltatik. Ez azt jelenti, hogy a Windows NT mint egy NetWare 3.x-es szerver a felhasználókat képes menedzselni a NetWare 3.x-es szervereken. Ez a megoldás a NetWare-es kliensek számára elérhetővé teszi a Windows NT szervert. A **File and Print Services for NetWare** (külön termék) pedig lényegében egy 3.12 NetWare emuláció, a legfontosabb segédprogramokkal egyetemben.
A NetWare a kliens oldalon támogatta eddigiekben a Windows NT -t, most jelenik meg a Windows NT munkaállomás központi menedzselését megvalósító alkalmazás. Természetesen mindkét rendszer támogatja a másikról történő áttérést. A NetWare a 3.12 szervereket az úgynevezett bindery emuláción keresztül támogatja és képes az NDS-be szervezni ezeket.

Egyéb

Fontos a központi adminisztráció, melyet mindkét rendszer támogat. A NetWare-ben egyrészt integrálva van az NWAdmin-ban a legtöbb funkció, a Windows NT -ben több különálló segédprogram áll a rendszergazdák rendelkezésére. Említésre méltó még a Windows NT -nél a felhasználói profilok kezelése, mely a teljes user környezetet magába foglalja. A NetWare-ben pedig a NAL a NetWare Application Launcher, mely segítségével alkalmazásokat is kezelhetünk az NDS-ben, mint objektum.

Összefoglalás

A két hálózati operációs rendszer képességei konvergálnak egymáshoz. Amilyen tulajdonsággal rendelkezik az egyik, azt idővel a másik is megvalósítja. A választást mindenképpen az alkalmazás környezet vizsgálatával kezdje, és vizsgálja meg, valójában az alkalmazás mit igényel. A kliens oldalon gyakorlatilag látható egy Windows NT munkaállomás terjedés a Windows 95 rovására, főleg a vállalati területen, hiszen ha a nagyobb hardvert úgyis meg kell venni (W95), akkor érdemesebb a Windows NT -t egyből meglépni és kihagyni egy lépcsőfokot, mellyel jelentős költségek takaríthatók meg. A statisztikai adatokra hívom még fel a figyelmet, nehogy félrevezessék. A licenszelési politika eltérő a két cégnél. A Windows NT-t 5 vagy 10 felhasználósként kell megvenni (relatív olcsó), majd a user licenceket egyenként kell hozzávenni. A NetWare-nél ellenben a rendszer megvételekor kell megvenni a user licencet is, ami bővíthető (ez magasabb költség). A Microsoft ezért eladott darabot ad meg általában (ha megad), a Novell pedig user licencet. A téma részletesebb vizsgálódás után is tartogat meglepetéseket, ezért javaslom a részletesebb vizsgálódást a konkrét esetekben.
További információk: megtalálhatók az Interneten, a http://www.novell.com és a http://www.microsoft.com címeken, az előadás anyaga pedig a http://www.datanet.hu/neten címen érhető el.
A részletesebb információk a Microsoft TechNet CD-jén valamint a Novell Support Connections CD-jén találhatók.

C1 Diszkrét Titkosítási védelem
D Minimális védelem

Az egyes szintek tartalmazzák az előző szint tulajdonságait. Az USA kormányhivatalai a C2 szint alatt nem használhatnak rendszereket. A C2 szint esetén az objektumok (fájlok) elérésének korlátozása, ennek ellenőrzése, az auditálás a lényeges tulajdonságok.

A számítástechnikai rendszerek biztonságának európai szabványa az Information Technology Security Evaluation Criteria (ITSEC) , vagyis Információ Technológia Biztonsági Értékelési Szempontjai.

A NetWare rendelkezik a C2 Red Book követelményeinek kielégítéséhez szükséges képességekkel. (Mivel nincs kliens NetWare ezért nincs értelme Orange Book-ról beszélni).

A Windows NT jelenleg az amerikai C2-es szint Orange Book-ban meghatározott (azaz egyedi PC-kre vonatkozó) követelményeit teljesíti, ez az európai E2-nek felel meg. A Red Book követelményrendszerének (a teljes hálózat biztonságának) vizsgálata jelen pillanatban még tart.

A szigorú C2-es szintnek megfelelő biztonság eléréséhez pl. a következő beállításokat mindenképpen el kell végezni.

- A merevlemezen csak NTFS fájlrendszert installálhat.
- A hálózati részek installálását le kell tiltania.
- Le kell tiltani az OS/2 és POSIX alrendszerek futását.

A teljes hatás eléréséhez gondoskodnunk kell a hardver védelméről is. A gépet kulccsal és jelszóval védhetjük, illetve biztonságos (nem szabadon megközelíthető) helyiségbe helyezhetjük.

Az auditálás része mindkét rendszernek. A NetWare-ben ez elkülönülhet az adminisztrátor tevékenységétől, tehát az adminisztrátor is auditálható.

Távoli elérés

A Windows NT a RAS (Remote Access Service) szolgáltatáson keresztül valósítja meg a távoli elérést, mely gyakorlatilag ugyanolyan kapcsolat, mintha LAN lenne, csak más az interfész. A NetWare-nél ez külön termék, a NetWare Connect, mely képes a dial-out-ra is. Az alapelvei megegyeznek.

Hálózati tulajdonságok

A legfontosabb a protokollok kezelése. Mindkét rendszer képes az útvonal választásra (routing), a TCP/IP, és az IPX/SPX protokoll kezelésére, bár a Windows NT egy sajátos IPX implementációt készített , mely nem teljesen egyezik meg a NetWare-évél. Nem ismeri a packet burst-öt (több csomag átvitele egyszerre) , az NLSP-t (korszerű útvonalválasztó protokoll) , a LIP-et (nagy méretű csomagok átvitelét). A Windows NT a NetBeui-n kívül a DLC protokollt is kezeli. A NetWare-ben külön termék, de lehetőség van a nagytávolságú szinkron adatvitelre, mely kompresszált adatokkal történhet, és LAN-WAN-LAN viszonylatban is működik.

Kereszt támogatás

A Windows NT vezet ezen a területen. A **Client Services for NetWare** (CSNW) segítségével a kliensek oldaláról lehet elérni mind a Windows NT , mind a NetWare szervereket. Természetesen ilyen esetben a NWLink protokoll használata kiegészül a

amely globális csoportokat ezután helyi csoportokhoz rendel. Az erőforrás jogosultság megadás csak a helyi csoportok számára történik, felhasználó fiók nem kap közvetlenül jogosultságot.

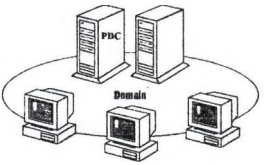
Az előző oldali ábrán bemutatott A-G-L-P betűk az angol szavak rövidítései, az ACCOUNT (felhasználói fiók), a GLOBAL (globális csoport), a LOCAL (helyi csoport), a PERMISSION (jogosultság)-ot jelöli.

Tartomány modellek

A tartomány kialakításakor több szempontot is figyelembe kell vennünk. A legfontosabb szempont a fizikai hálózat felépítése. Ha több helyi hálózatot összekötő lassú WAN vonalakkal dolgozunk, ezeket külön tartományokba célszerű szervezni, de működő megoldást kaphat, ha egy tartományt készít el és a távoli helyekre BDC-ket telepít. A következő fő szempont a felhasználók száma. Egy tartományt 30000 felhasználóig célszerű tervezni, ha ennél több felhasználót akar adminisztrálni, több tartományt kell terveznie, akkor is, ha egy fizikai LAN-ba kapcsolt a hálózatunk. Az alábbi tartomány modellek mutatják az előző problémákra adott válaszokat.

Egyedi tartomány modell

Mint a neve is mutatja, egyetlen tartományunk van. Ha nem ütközünk korlátokba, célszerű ezt a modellt választanunk. Ez tervezési és erőforrás szükséglet szempontjából a legkevesebb ráfordítást igényli. Minimum egy PDC-t kell installálnunk. A felhasználók fiókjainak definiálása is egyszerű, nincs szükségünk meghatalmazások definiálására.

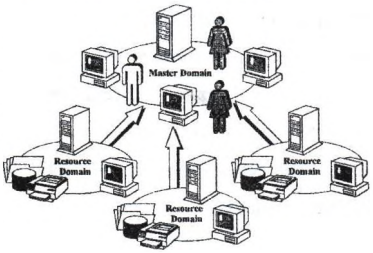


Fő (Master) tartomány modell

A következő modell már bonyolultabb, bár még mindig a lehető legjobban adminisztrálható és könnyen áttekinthető. A lényege, ha több telephelyen lassú WAN vonalakon keresztül kell összekapcsolni több tartományt, de a központi adminisztrációt fenn kívánja tartani. Ilyenkor a távoli helyeken lévő tartományok erőforrásait a fő tartományban található felhasználói fiókokon keresztül lehet elérni. Itt alkalmazható az A-G-L-P előzőekben leírt szabály.

Természetesen a fő tartomány modellben már meghatalmazásokat is kell definiálni. Ezek még egyszerűek, könnyen kézben tarthatók, mivel egyirányú meghatalmazások.

A további tartomány modellek már bonyolultak, nem jól adminisztrálhatók, mutatják, hogy a bonyolultság egy szintjén túl már a tartomány modell nem elegendő.



Fájl és nyomtatási szerviz

A fájl kiszolgáló szolgáltatások alapjaiban hasonlóak, hiszen különböző jogosultságok szerint a felhasználók számára fájlokat osztanak meg. A jogosultságok, és az öröklődési rendszer a NetWare-ben részletesebb, finomabb beállításokat tesz lehetővé. A jelentős különbség a fájl rendszerekben van. A NetWare egy saját fájl rendszerrel dolgozik, mely alapvetően FAT jellegű, de képes a blokk méretet beállítani installáláskor, és a nagy blokkok megosztására is képes (blokk szuballokáció). A kompresszálás a NetWare-ben azt jelenti, (ha beállítja) a régen nem használt fájlok kerülnek tömörítésre, amikor a szerver éppen nem dolgozik. A Windows

Az Informix
objektum-relációs adatbázis kiszolgálója

dr. Balogh Kálmán

Informix Technology Center Hungary
1063 Budapest
Bajnok u. 13.
Tel: (06-1)-302-33-88/117
Fax: (06-1)-302-33-95

E-mail(Internet): kbalogh@informix.hu

A jelenlegi relációs adatbáziskezelők elsősorban az on-line tranzakciófeldolgozás igényeit elégítik ki. Emellett azonban egyre nagyobb az igény döntéstámogatásra, és az alfanumerikusakon túl sokféle összetett adat (pl. idősorozat, dokumentum, multimédia, térkép) kezelésére. Az adatbázisok elérési lehetőségeit pedig szélesre kell tárni a lehetséges felhasználók előtt.

Az RDBMS gyártók front-end eszközeikben már évek óta objektumorientált fejlesztést kínálnak. Az objektumorientáltságból fakadó előnyök azonban nem használhatók ki teljes mértékben, ennek ugyanis a kiszolgálók relációs volta korlátokat szab.

Az Informix már 1995 végétől rendelkezik az *Illustra* objektum-relációs kiszolgálóval (ORDBMS), amely a fenti igényeknek tesz eleget. Az Informix *Universal Server*e pedig '96. végére készült el a már most is sokoldalúan alkalmazott Illustra megoldásainak, és korábbi, dinamikusan skálázható architektúrájú RDBMS-ének általánosításával. Az objektumorientálttá kiterjesztett relációs architektúra szépsége, hogy a legkülönfélébb alkalmazásoknak természetes utat biztosít, így az intranet/internet szolgáltatás lehetősége is szerves következménye.

Jönnek a nem hagyományos adattípusok

Míg a hagyományos relációs adatbáziskezelő rendszerek bizonyulnak a legalkalmasabbaknak alfanumerikus adatok (betűk és számok) kezelésére, a technológiai fejlődés már új utakat nyitott az összetett és előreláthatatlan szerkezetű információk előtt, beleértve a video, audio, térképi, 2D és 3D, idősorozat, grafikai, szöveges és Web-alapú adattípusokat, melyek a hagyományos RDBMS technológiával nem kezelhetők hatékonyan. Ilyen adatok egyre

tömegesebben jönnek létre, és felhasználásukra a fokozódó piaci elvárásoknak megfelelni kívánó vállalatok nyomására már eddig is sok speciális megoldás született. Melyek is ezek az igények?

Vegyük például a pénzügyi szektort, ahol a broker cégek új, hatékony alkalmazásokat használnak a beruházási döntéseknél szükséges komplex pénzügyi információk elemzéséhez. Az ilyen szervezeteknek óriási mennyiségű összetett pénzügyi adatot kell kezelniük anélkül, hogy az információt nagyszámú elemi rekordra tördelnék, ami a lekérdezéseket reménytelenül elbonyolítaná, és az eredmény előállítása kivárhatatlanul elnyúlna. Sokkal jobb megoldás, ha a pénzügyi adatot megtartják eredeti formájában - időben változó információ sorozatként, ami meglévő portfoliókezelő kereskedelmi rendszereikkel is könnyebben integrálható.

Új adattípusokat igényel a vizuális média ágazat is, ahol az analóg technológiáról digitálisra történő hirtelen átállás újjáformálta számos piaci szegmens (publikálás, fotózás, film és TV műsorok, különleges effektusok, oktatási és szórakoztató média) arculatát. Ahogy a hardver és szoftver technológia egyre fejlettebbé és gazdaságosabbá válik, ezek a cégek a relációs adatbázis technológiát a digitális információ tárolására és kezelésére kívánják használni, mint multimédia könyvtárakat.
A probléma abból adódik, hogy ezek a cégek multimédia információikat tényleges tartalmuk (pl. vizuális jellemzőik, összetételük) alapján is szeretnék visszakeresni, nemcsak fáradságosan hozzájuk rendelt, és csak korlátozott tulajdonságleírást biztosító kulcsszavak szerint. Erre pedig a hagyományos RDBMS technológia nem képes.

Miért van szükség korlátlanul kiterjeszthető szerverre?

Az Informix Illustra szervere már most támogatást nyújt pl. multimédia tartalom kezeléséhez anélkül, hogy feláldozná a relációs modell hatékonyságát. Sőt, az Illustra szerver igény szerint kiterjeszthető bármely új adattipussal, így a felhasználási területek számára saját alkalmazás-specifikus adattípusok hozhatók létre. A szükséges új adattípusok meghatározása egy-egy speciális alkalmazási terület mély ismeretét feltételezi, minden ilyet tehát az ORDBMS szállító nem építhet be előre a termékébe. Az új típusú adatok igény szerinti kezeléséhez kiterjeszthető szerver technológiára van szükség.

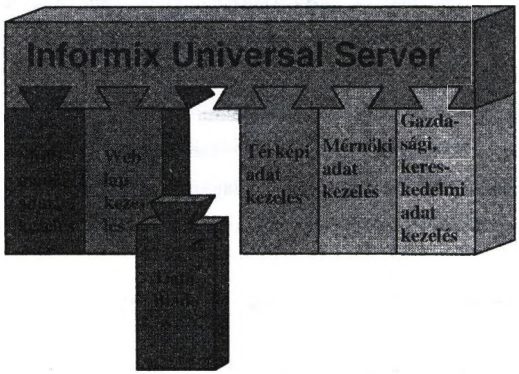
A kiterjeszthetőség fogalma viszonylag új keletű az adatbázis szerverek területén. Míg az objektum orientált alkalmazás-fejlesztő eszközök, mint az INFORMIX-NewEra már évek óta

lehetővé teszik, hogy osztálykönyvtárakat a felhasználók könnyedén integráljanak, ha az applikációtól kívánt új funkciók ezt megkövetelik, addig a szerverek kiterjeszthetősége, új típusú adatokat tároló és kezelő funkciója forradalmian új.

Hogyan valósítható meg a korlátlan kiterjeszthetőség?

A kiterjesztés *DataBlade modul*okkal - csatlakoztatható szoftver kiegészítőkkel (osztálykönyvtárakkal) történik, amelyek kibővítik az Illustra szerver általános célú képességeit. Ezek a kiegészítések az egyes alkalmazások (adattípusok) speciális igényeihez hangolt tárolási és kezelési funkciókat kínálnak.

A DataBlade modul Illustra szerverbe történő behelyezése analóg egy speciális penge univerzális nyélbe való bepattintásával. Ha új funkcióra van szükség, csak "bepattintjuk" a DataBlade modult a szerverbe, és abban a pillanatban már hatékonyan tárolhatjuk, kereshetjük vissza és kezelhetjük az új típusú nem hagyományos adatot.



A szerver korlátlanul bővíthető DataBlade-ekkel

Az új adattípusok - éppen mert osztályokként valósulnak meg - egymással összekombinálhatók, egymásra építhetők, specializálhatók, így az egyes alkalmazások konkrét igényeinek megfelelő osztályok és objektumbázisok hozhatók létre.

DataBlade technológia a BLOB-okkal szemben

Mivel egy tisztán relációs adatbáziskezelő strukturálatlanul, nem tipizált bináris, nagy objektumként (binary large object - BLOB) tárolja a különféle nem hagyományos adattípusokat, nem képes *tartalom alapján lekérdezni* ilyen adatot. Ez a megközelítés csak korlátozott összehasonlítási lehetőséget kínál arra, hogy az egyik objektumot a másikkal összevessük. Nem tesz lehetővé hatékony keresést, ami pedig minden kezelő művelet alapja.

Ezzel szemben a DataBlade technológiával megvalósítható a különféle adattípusok hatékony szerver-oldali kezelése. Sőt, DataBlade modulok segítségével az alkalmazási logika a kliens oldalról átvihető a szerverre, így megnövelve a teljesítményt, és egyszerűsítve az alkalmazást.

Egy DataBlade modulban az adatok tárolási szerkezete, a kezelő műveletek (metódusok) és az indexelési mechanizmus egymáshoz a megvalósított adattípusok sajátosságainak megfelelően alkalmazkodik, így biztosíva az ORDBMS hatékony működését. Az adatbáziskezelő lekérdezés optimalizálója figyelembe veszi a különböző adatszerkezetek, kezelő műveletek és indexek speciális költségeit.

A relációshoz képest általánosított indexelés *többdimenziós lekérdezések* hatékony végrehajtását is lehetővé teszi, ami például adatáruházi, *OLAP* (On-Line Analytical Processing) alkalmazások, *döntéstámogatás* esetén fontos. A többdimenziós lekérdezések problémái a *térképi alkalmazások*on keresztül válnak szemléletessé, amely természetes alkalmazási területe ezeknek a problémáknak.

Az elérhető DataBlade Modulok

2D/3D térbeli DataBlade Modul
Image DataBlade Modul
VIR (Vizuális Információ Keresés)DataBlade Modul
Text DataBlade Modul
Idősorozat DataBlade Modul
WEB DataBlade Modul.

További DataBlade Modulok fejlesztés alatt állnak és kibocsátásuk a közeljövőben várható. Ilyen például az OLAP DataBlade, amely az Informix meglévő, *MetaCube* nevű OLAP eszközét valósítja meg DataBlade technológiával.

DataBlade modulok kombinált alkalmazása

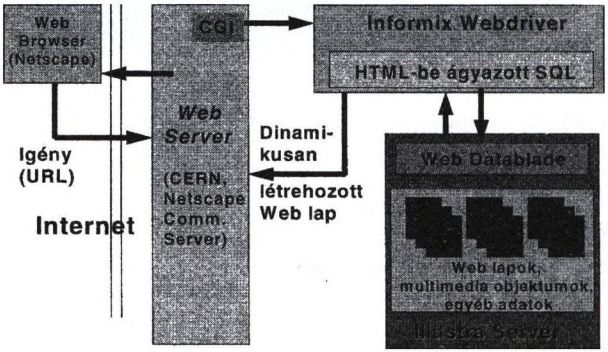
Mivel az alapul szolgáló objektum-relációs technológia segítségével a kód újrahasználható, az új DataBlade modulok hasznosítani tudják a más DataBlade modulok által meghatározott funkciókat és adattípusokat. Ezáltal speciális DataBlade modul kifejlesztése meglévő általánosakból gyorssá és könnyűvé válik.

Például egy döntéstámogató alkalmazás természetesen építhető az OLAP DataBlade-re. Ezt pl. pénzpiaci alkalmazásnál kiegészíthetjük az Idősorozat DataBlade Modullal, és ha helyszíneknek jelentősége van, a térképi DataBlade-del. Végül a felhasználók számára az alkalmazást a WEB DataBlade Modul teheti szervezeten belül és kívül egyöntetűen elérhetővé.

Az Illustra Web architektúrája

Az Illustra Internet szolgáltatást lehetővé tevő architektúráját ismertetjük, amely természetesen a Web DataBlade-en alapul.

Az Illustra szerver Web architektúrája



A bemutatott architektúra több, különböző képességű termék közös jellemzőit mutatja be. Pl. az Illustra Webdriver-e esetén az architektúra finomítható: a Web és az adatbázis szerver különböző gépeken is elhelyezhető, a Webdriver maga is kliens és szerver komponensekre

tagolható. A Webdriver kiszolgáló része a rendszerbe egyszer betöltve, konfigurálható mennyiségű szálon át állandóan tartja a kapcsolatot az adatbáziskezelővel. A Web szerverrel (pl. valamelyik Netscape szerverrel) a Webdriver kliensei kapcsolódnak össze. Ez az architektúra skálázható, rugalmasan és hatékonyan alkalmazkodik a terheléshez, amire egyszerűbb megoldások közel sem képesek ilyen mértékig.

Az objektum-orientált és az objektum-relációs DBMS-ek összehasonlítása

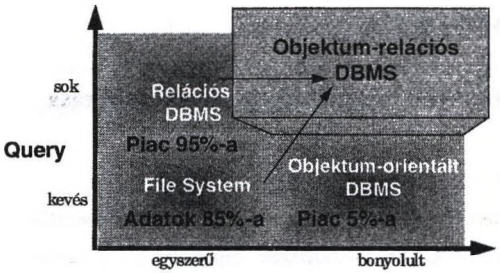
Az objektum-orientált DBMS-ek jellemzői

- nem túl nagy számú összetett adatstruktúrát hatékonyan kezelnek, így olyan speciális alkalmazási területeken, mint pl. a CAD és CASE adattárházak (repository-k), a tisztán relációs DBMS-eknél hatékonyabbak
- adatkezelő nyelvük C++ vagy Smalltalk alapú, így a lekérdezési lehetőségek korántsem olyan gazdagok és rugalmasak, mint amit az SQL-ben megszoktunk
- általában csak az objektumokat tárolja a szerver, minden műveletet az így szükségképpen erőforrásigényes kliensek végeznek (még a függvényértékektől függő indexek kezelését is!); erős a hálózati átbocsátóképességre vonatkozó igény; az objektumok osztályai az alkalmazásokba építendők be.

A fentiekkel ellentétben az objektum-relációs DBMS-ek

- általános célúak
- adatkezelő nyelvük az SQL objektum-orientált kiterjesztése (a készülő SQL3 szabvány szerinti)
- a szerver mind az osztályokat, mind az objektumokat tárolja és kezeli, így a kliensek és a hálózat erőforrásigénye sokkal kisebb az osztályokat a klienseken kezelő megoldásénál, sőt az egész rendszer kisebb erőforrásigényű.

Piaci trendek



Adatok komplexitása

Prognózis 1996-ra : 1 Mrd. $
Prognózis 2000-ig : A piac megtízszereződik
(nagyobb lesz az RDBMS piacnál)

Az *ObjectTeam*, a vállalati szintű információs rendszer fejlesztés eszköze

Balogh Kálmán

Informix Technology Center Hungary
1063 Budapest
Bajnok u. 13.
Tel: (06-1)-302-33-88/117
Fax: (06-1)-302-33-95

E-mail(Internet): kbalogh@informix.hu

Az *ObjectTeam* a legnagyobb CASE technológiára szakosodott cég, a *Cayenne* objektumorientált, a szoftverek teljes életciklusát támogató terméke. A cég és termékeinek rövid bemutatása után az ObjectTeam kódgenerálási, valamint a nagy szervezetek egészét átfogó, elosztott információs rendszerek készítését és a csoprtmunkát támogató képességeit tekintem át.

A Cayenne CASE eszközei

Februárban jött létre a Cayenne cég az amerikai *Cadre* és a *Bachmann* egyesülésével; ez a koncentrálódás a második lépcsője annak a folyamatnak, amelynek során egy évvel korábban a Cadre és a holland *Westmount* egyesült. A burlingtoni központú Cayenne tagvállalatainak együttes éves árbevétele 70 milló dollár. Ezzel az eredménnyel a világ 50 legnagyobb szoftveres vállalata közé tartozik, egyben a legnagyobb, amely a CASE technológiára szakosodott. A Cayenne-t létrehozó két cég összesen 50 ezer licencet adott el. A fejlesztés Amerikában, Hollandiában és Indiában folyik. A Westmountot az *InTeC* elődje, az *OpenSoft* képviselte Magyarországon, de a többi terméknek nem volt magyaroszági "helytartója", így ezután az InTeC forgalmazza a teljes Cayenne termékskálát.

A Cayenne termékei közül a hagyományos, strukturált módszertanon alapuló *VantageTeam* és az objektumorientált *ObjectTeam* mind technológiai, mind Informix-támogatás szempontjából testvérek. A teljes életciklust átfogó, ügyfél-kiszolgáló architektúrájú CASE eszközökkel adatbázis alapú információs rendszerek tervezhetők és hozhatók létre. A szerveroldali platform vagy valamilyen UNIX, vagy WindowsNT lehet; az előtéri eszköz pedig UNIX-on, Windows NT-n vagy Windowson futhat.

Bár a Cayenne-nek vannak más CASE eszközei is - így a TeamWork hagyományos alapú, elsősorban valós idejű termelésirányítási rendszerek készítésére való; az Ensemble C-ből visszafejti a kódot, a GroundWorks adatbázis-modellezésre, a Terrain pedig adatbázis-tervezésre, -generálására és visszafejtésre használható -, az InTeC kínálatában a VantageTeam és az ObjectTeam áll a fókuszban.

Referenciaként említethetjük a Hungarocamiont és az APEH-et, utóbbinál elsősorban a VantageTeamet használják. Az oktatási intézmények közül például a Pénzügyi Főiskolán és a Műszaki Egyetemen tanítják a Cayenne CASE eszközeit, de maguk az InTeC is használja a Cayenne-t néhány projektjénél. A Windows NT-s MS SQL Serverrel működő változat támogatására a közeljövőben új partnert von be az InTeC. Jelenleg két-három munkatársa foglalkozik - nem teljes munkaidőben - a CASE eszközök támogatásával és értékesítésével, hosszú távon azonban perspektivikus területnek tartják a CASE technológiát, amelyre tapasztalataik szerint egyre inkább megérik a magyar piac.

A VantageTeam és az ObjectTeam kódgenerálási képességei

Nyitott kódgenerátoraik nemcsak Informix, hanem Oracle, CA/Ingres, Sybase serverre is generálhatnak SQL kódot, beleértve tárolt eljárásokat. Ami a front-end oldalt illeti: az ObjectTeam a NewErára, az Informix alkalmazás particionálást lehetővé tevő objektumorientált, grafikus előtéri eszközére is előállíthat kódot, beleértve beágyazott SQL utasításokat, és összekapcsolódhat a NewEra fejlesztőrendszerrel. Hasonló képességekkel rendelkezik a strukturált módszertant támogató VantageTeam az INFORMIX-4GL-re vonatkozólag. A harmadik generációs programozási nyelvek közül a C-re, C++-ra, Adára, Javára, Corba IDL-re, SmallTalkra és Visual Basicra vonatkozóan képes SQL utasításokat generálni a két CASE eszköz. A kódgenerátorok nyitottak, úgynevezett TCL nyelven készülnek, és forrásszinten, dokumentációval együtt rendelkezésre állnak. A TCL felület révén a kódgenerátorok és a repository (modelladatbázis) is módosíthatók. Ez kiegészül a CASE eszköz felhasználói felületének testreszabhatóságával, és ezek együtt teszik lehetővé az integrálást más fejlesztőkörnyezetekkel, döntéskövető (DOORS) és dokumentációkészítő eszközökkel (FrameMaker, Interleaf, Word) is.

Generálás szempontjából a következő képességek érdekesek leginkább:

- a generálás eredménye: front-end és back-end kódok, célnyelvek
- a generálás lépései, eszközei (a CAD diagram, a TCL nyelv, template-ek)
- a generálás befolyásolása, testre szabása

- az újrafelhasználás lehetőségei
  - az újragenerálás módja (round-trip engineering)
  - verzió és konfigurációkezelés
  - osztálykönyvtárak használata.

Csoportmunka támogatás

Az ObjectTeam nagy szervezetek egészét átfogó, elosztott információs rendszerek készítését is többféle képességgel támogatja.

A felhasználók szerepük, jogaik szerint csoportosíthatóak. A hozzáférési, alkalmazási jogokat a szerepdefiníciókban határozzuk meg a csoportoknak megfelelően, majd az egyes felhasználókhoz a megfelelő szerepnévvel rendeljük a kívánt jogokat.

A felhasználók egy közös repository-n dolgoznak, amely tetszőlegesen finomítható verzió, sőt konfigurációkezeléssel van ellátva. A különböző fejlesztésekhez tartozó információk átfedhetik egymást, megkönnyítendő az újrafelhasználást.

A tervezési fázisban megfelelő diagramtípus szolgál a rendszer architektúrájának a kialakításához, pl. a gépek, felhasználók és funkciók egymáshoz rendeléséhez, valamint ezek csoportosításához, alrendszerek kialakításához.

Az ObjectTeam multiplatformos kliens-szerver architektúrája maga is hozzájárul a többfelhasználós igények hatékony kielégítéséhez.

Információs rendszer fejlesztése és bevezetése az OLAJTERV Rt-ben

Nagy Péter (OLAJTERV Rt.) és Fodor Imre (FAIR Kft.)

Bevezetés

Az OLAJTERV Rt. vezetése elhatározta, hogy a cég versenyképességének javítása érdekében az információs rendszerének technológiájában megpróbálja fokozatosan elérni az iparilag fejlett európai országok színvonalát. Az információs rendszer célja annak biztosítása, hogy a vállalat stratégiai elképzeléseihez, céljaihoz illeszkedve erőforrásként, eszközként segítse azt.

Az előadás témája az ezt kiszolgáló számítógépes információs rendszer kialakítása és bevezetése során szerzett legfontosabb tapasztalatok összefoglalása. Mivel két évvel ezelőtt a fejlesztés tapasztalatairól már tartottunk egy előadást, ezért ezt a témát csak érintjük és a hangsúlyt a rendszer bevezetésére tesszük.

A rendszer felépítése,

Az elfogadott rendszerfejlesztés stratégiai koncepció az információs rendszert három alrendszerre osztotta :

A ***projekt menedzsment rendszer*** a szerződött, illetve előkészítés alatt álló megrendeléseink kézbentartását (határidőre, adott műszaki tartalommal való elvégzését) hivatott biztosítani, továbbá a társaság piaci környezetével áll informális kapcsolatban.

Az ***ügyviteli rendszer*** elsődleges célja az, hogy pontos, naprakész információt biztosítson a társaság tervezett és tényszerű költségfelhasználásról, folyamatosan változó pénzügyi helyzetéről.

A harmadik a ***vezetői tájékoztató rendszer***, mely az ügyviteli és projekt mendzsment rendszer információit, szolgáltatásait integrálva, biztosítja a vállalat szempontjából meghatározó vezetői szintek folyamatos, szelektív, korrekt tájékoztatását, statisztikai adatok gyűjtését, azok alapján hatékony döntéstámogató információk kinyerését.

Eszközkörnyezet

A megvalósítás a korábban már bevált UNIX operációs rendszerre és ORACLE adatbáziskezelőre épült. A rendszer adatbáziskiszolgáló gépe egy Silicon Graphics Challange M típusú gép (150 Mhz MIPS processzor, 64 M RAM, 8 GB diszk), mely gépen kizárólag az ORACLE RDBMS adatbáziskezelő, valamint a SQL *Net hálózati kommunikációs szoftver fut.

Az ügyviteli alkalmazások egy ALR Pentium alkalmazás szerver gépen futnak, SCO UNIX operációs rendszerfelügyelete alatt. A munkahelyek DEC VT510 terminálok, melyek az épület különbözõ részeiben felállított öt db. terminál-szerver eszközön keresztül kapcsolódnak az ETHERNET hálózatra. Ezenkívül DOS-os kliensek is kapcsolódnak az adatbázishoz.

A vásárolt ***projekt menedzsment rendszer*** egy kliens-szerver alkalmazás, mely MS-Windows operációs rendszer alatt, PC-ken fut (5 munkahely), adatai viszont a központi ORACLE adatbázisban tárolódnak. A PC-k FTP PC/TCP szoftverrel kapcsolódnak az ETHERNET hálózatra.

A ***vezetõi tájékoztató rendszer*** tervezett munkahelyei szintén MS-Windows alapú PC-k, melyek FTP PC/TCP szoftverrel kapcsolódnak a hálózatra.

Fõbb célkitûzések

A rendszerrel szemben megfogalmazott fõbb célkitõzések a következõk voltak :

- Egységes fogalmi rendszer
- Egységes tranzakciókezelés
- Alkalmazások (alrendszerek) közötti egységes kommunikációs mechanizmus.
- Egységes dokumentumkezelés
- A számítógépes rendszer szolgáltatásainak átgondolt megtervezése
  - ⇒ egységes felhasználói felület (menük, adatbeviteli eljárások),
  - ⇒ segély rendszer,
  - ⇒ intelligens nyomtatás vezérlés.
- A számítógépes alkalmazás folyamatba illesztése, a vállalati tevékenységek szükség szerinti módosítása, mindezek dokumentálása az ügyviteli szabályzatokban.
- Magas színvonalú oktatás lebonyolítása az érintettek számára.

Módszertani háttér

A kitûzött célok, az egységes rendszer megvalósításához megbízható módszertant kellett választani az alábbiak miatt :

- egységes gondolkodás a team minden résztvevõje között,
- a feladatok egymásra épülõ, teljeskörû, kontrollált elvégezhetõsége,
- dokumentálás megfelelõ színvonala.

Mindezek érdekében az ORACLE CASE*Method módszertanát, illetve az erre épülõ CASE eszközeit használtuk fel a fejlesztés során.

Tapasztalatok a tervezésben

A jelentkezõ problémák:

- adatmodell-funkció modell viszonya,
- funkció modell - modul struktúra viszonya,
- módosítások, időbeli visszanyúlások.

Tapasztalatok a kivitelezésben

- Forms-ok bonyolultsága, adatbázis triggerek
- Riportok, nyomtatások
- Időközben felmerült magyarosítás problémái:
  ⇒ Adatbázis generálása után
  ⇒ Karakter készletek
  ⇒ Dátum problémák

Tapasztalatok a bevezetésben

A tervezési és fejlesztési témákkal összehasonlítva, lényegesen kevesebb szó esik a rendszerek beüzemeléséről. Pedig ennek a munkának a volumene összemérhető a fejlesztésre fordított munkamennyiséggel. Nem elhanyagolható az sem, hogy a beüzemelés során fellépő problémák mennyisége és milyensége lényegesen befolyásolja a rendszert használók véleményét, hozzáállását, segítőkészségét és tűrőképességét.

Tapasztalataink szerint üzemeltetés során leggyakrabban a következő problémákkal kell megküzdeni:

- Az induló adatállomány betöltése.
- Párhuzamos üzem esetén az új és a régi rendszerek működésének egyeztetése.
- Többfelhasználós rendszer lock problémái.
- Sebességi problémák.

Az induló adatállomány betöltése.

A bevezetendő rendszer tesztelése megtörténik a szokásos egyszerű próba adatokkal. Úgy tűnik, hogy nagyjából, vagy egészében renben működik minden. Ezután betöltjük a törzsadatokat, majd megpróbáljuk betölteni a meglévő rendszerek adatait, vagy felvinni a papíron létező adatokat. Ha elég intelligens az adatmodellünk, vagyis minden összefüggést constraint-tel ellenőrzünk, a betöltés első próbálkozásra rendszerint nem sikerül.

A sikertelenséget a következők okozhatják:

A rendszer tervezése során nem lehet figyelembe venni a korábbi rendszerben meglévő anomáliákat. Ezek rendszerint olyan esetek, amikor a mûködtető (indokoltan, vagy indokolatlanul) eltér a maga által meghatározott szabályoktól, vagy egy különleges eset kedvéért új szabályt állított fel. Ezeknek az eseteknek az ismertetése rendszerint kimarad a rendszerrel támasztott követelmények ismertetéséből. Tehát ez nem tekinthető tervezési hibának. Külön készülni rá nem nagyon tudunk, de a bevezetésre szánt időben ezt figyelembe kell venni. Az idő korrekt becslése szinte lehetetlen.

A második problémakör már rendszerint az adatfeltöltés tervezésének hibájára vezethető vissza. Tervezéskor arra figyelni szoktunk, hogy a rendszer által normál menetben bevitt adatok konzisztensek legyenek, azonban az induló adatok betöltésekor ez a konzisztencia már nem minden esetben teljesül. Ez általában a származtatott adatok esetén szokott előfordulni. Ezek a származtatott adatok léteznek a meglévő rendszerben is, azonban ezek közül sokat nem szándékozunk betölteni, hanem megírt algoritmusok segítségével állítunk elő. Gyakran előfordul, hogy az általunk előállított származtatott adatok nem egyeznek meg a régi rendszer megfelelő adataival. Pedig azt az információt kaptuk a megrendelőtől, hogy ezek az adatok az általunk használt algoritmusnak (szabálynak) megfelelően keletkeztek. Előfordul, hogy bizonyos tételeket az általunk ismert szabálytól eltérően egyedileg minősítettek, ami nem számít feltétlenül hibának. Azonban még hiba esetén is egyeznünk kell az induló adatállománnyal. Nem a bevezetésre kerülő rendszer feladata a múlt minden hibájának felderítése.

Párhuzamos üzem esetén az új és a régi rendszerek mûködésének egyeztetése.

Az alapdefiníció nagyon egyszerű, a régi és az új rendszernek párhuzamos üzemeltetés esetén ugyanazt az eredményt kell produkálnia. Azonban egy új rendszer legtöbbször minőségi előrelépést is jelent. Vagyis az új rendszer plusz funkciókat tartalmaz, és a fogalmi rendszere alapvetően különbözik a régitől. A kettő között nem lehet egy az egyben megfeleltetést alkalmazni.

Az új funkciók szempontjából nem létezik párhuzamos üzemeltetés. Ellenőrző listákat kell készíteni, és a felhasználók bevonásával kell ezeknek a funkcióknak a mûködését kontrollálni.

Az eltérő fogalmi rendszer okozza az igazi problémát. Mivel az egyezőséget valahogy mégis igazolni kell, meg kell találni a különböző fogalmi rendszerek közötti megfeleltetés algoritmusát. Ez azt igényli, hogy olyan szinten meg kell ismerni a régi rendszert, amilyen mélységig soha nem szándékoztunk. Ezután bonyolult algoritmusokat, illetve még

praktikusabban sok szinten keresztül végigvonuló konverziós táblákat kell felállítanunk, aminek segítségével megvalósíthatjuk a két rendszer közötti egyeztethetõség vizsgálatát. Ezt az egyezõséget papíron kell dokumentálni, tehát rengeteg olyan listakészítõ programot kell készíteni, amelyet késõbb soha nem használunk fel. Ezeknek a listakészítõ programoknak soha nincs elõre elkészített specifikációja. Formátuma és adattartalma többszöri egyeztetés és átdolgozás után alakul ki. Nagyon gyakran felmerül az az igény, hogy a régi rendszerbõl szeretnének olyan egyeztetõ listát, ami nem létezik. Ennek az elkészítése egy harmadik generációs programnyelv esetén megfelelõ dokumentáció hiányában legtöbbször lehetetlen. De nyitott rendszerek esetén is komoly energiabefektetést igényel.

Saját gyakorlatomban többször elõfordult, hogy nem egyezett a régi és az új rendszer. A bevezetndõ rendszerben hosszú keresés után sem találtunk hibát. A régi rendszer alapos átvizsgálása során derült ki, hogy a felhasználók abban követtek el olyan hibát, vagy hibákat, amelyeknek kivédésre az nem volt felkészítve. Tehát a próbaüzem sok esetben nem, csak az új rendszernek a tesztelésébõl, hanem a régi rendszer üzemeltetésébõl adódó hibáknak a felderítésébõl is áll. A bevezetés során erre szánt idõ becslése komoly bevezetési gyakorlatot igényel. Becslési alapként szolgálhat a régi és az új rendszer fogalmi rendszerének különbözõsége, valamint a régi rendszer hibakezelési szintje.

Többfelhasználós rendszer lock problémái.

A lock-olási problémák leggyakrabban tervezési, illetve programozási hibák eredményeként lépnek fel. Az általam tapasztalt leggyakoribb hibák a következõk voltak:

Egy adott felhasználói programm olyan pillanatban foglal le egy objektumot, amikor a lefoglalás után még felhasználói közbeavatkozásra volt szükség. Elõfordulhat, hogy a felhasználó nem folytatja a munkát, az adott objektum lefoglalva marad nagyon hosszú ideig. Cél az, hogy a lock olyankor kerüljön kiadásra, amikor a tranzakció már batch jelleggel lefut.

Különbözõ felhasználói programok az objektumokat nem ugyanabban a sorrendben foglalták le. Tipikus esély a dead lock létrejöttére.

Kellemetlen jelenséget okozhatnak a megszorítások. A hivatkozott táblára értelemszerûen létezik index, azonban a hivatkozó tábla megfelelõ oszlopára nem kötelezõ az index létrehozása. Ha ez a tábla kicsi, sebességi szempontból nem is célszerû az index létrehozása. Azonban a megszorítások miatt kikényszerített lock-ok mechanizmusa sokkal kellemetlenebb, amikor a hivatkozó tábla megfelelõ oszlopára nincs index, mint amikor van. Ha nincs ezekre az oszlopokra index gyakori a dead lock elõfordulása.

Ami már nem programozási hiba, az a clients-server architectúrában a clients process-ek leszakadása. Ebben az esetben a rendszeradminisztrátor által alkalmazható erõforrás korlátozás, vagy hálózati beállítás jelenthet megoldást. Ez a probléma terminál-server architectúrában nem jelentkezik.

Sebességi problémák.

Egy lekérdezés nagyon gyakran elemi adatokból történõ összegzést, csoportosítást jelent. Emiatt a lekérdezés munkaigényes, sokáig tart. Célszerû lehet adatbevitelkor a gyakori lekérdezéseknek megfelelõ összegzett adatok tárolása.

A normálási feltételeknek megfelelõ adatmodell sok táblából áll. Ennek következtében a lekérdezéseknél sok táblát kell összekapcsolni. Emiatt egy lekérdezés is sok tábla összekapcsolásán keresztül történik, emiatt sokáig tarthat. Akár több nagyságrendnyi futási idõ csökkenést eredményezhet, ha a legnagyobb adatszûkítést jelntõ lekérdezés eredményét egy ideiglenes táblában letároljuk, majd ehhez az adathalmazhoz kapcsoljuk hozzá a többi táblát.

Komoly sebességcsökkenést okozhatnak az adatbázis triggerek, amelyek az adatkonzisztencia biztosítása szempontjából rengeteg csábító lehetõséget tartalmaznak, azonban a batch jellegû feldolgozások sebességét erõteljesen csökkentik.

SAPIENS OBJECT POOL OBJEKTUM ORIENTÁLT ADATBÁZIS-KEZELŐ HASZNÁLATA ÉS PC-S FELÜLETEI

SZŐKE JÓZSEF
SZÉKELY ATTILA

DUNAFERR DUNAI VASMŰ RT.
SZÁMÍTÁSTECHNIKAI ÉS SZOLGÁLTATÓ INTÉZET

Cégünknél a Sapiens objektum orientált adatbáziskezelő rendszer egy rendkívül hatékony, a jelenlegi igényeket minden szempontból kielégítő fejlesztőeszköznek bizonyult.
Tulajdonképpen egy alkalmazásgenerátor, mellyel programozás nélkül végezhető el a kívánt fejlesztési feladat.
A rendszerteljeskörű bemutatására nincs lehetőség, ezért az alábbiakban egy összetett feladat egy részének nagymértékben leegyszerűsített példáján keresztül probálunk meg rávilágítani arra, hogy a Sapiens esetében mit is jelent a hatékonyság.

A példában használt kifejezések magyarázata:

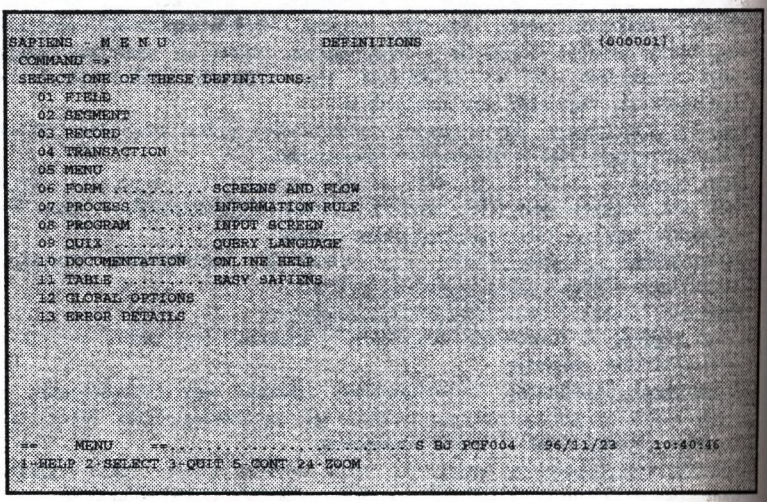
Entitás (Entity): A Sapiensben egy entitást egy más tábláktól nem függő tábla jelképez.
Kapcsolat (Relationship): A kapcsolat összetartozást jelent két vagy több entitás között. Függő táblák formájában jelenik meg a Sapiensben.
Objektum (Object): A Sapiensben egy objektum entitásként vagy kapcsolatként van definiálva.
Szegmens (Segment): Tulajdonképpen egyenértékű a táblával.
Független tábla (Indipendent Table): Egy tábla, amely nincs alárendelve más táblának.
Függő tábla (Dependent Table): Egy tábla, amely alá van rendelve a szülő táblájához.
Rekord (Record): Egy azon entitáshoz tartozó szegmensek összessége.
Tranzakció (Transaction): Az objektum orientált programozással foglalkozó szakirodalom sok helyütt üzenetként definiálja ezt a fogalmat. A tranzakció egy operációs kóddal ellátott összetartozó mezők csoportja, melynek feladata egy táblát módosítás vagy megjelenítés céljából elérni. Ez a lehetőség az, amivel az alkalmazás megjelenítési rétege érintkezésbe lép az adatbázissal. Egy objektum küldhet tranzakciót más objektumoknak, vagy akár saját magának is.
Esemény (Occurence): Egy egyszerű objektumhoz (tábla, szegmens, rekord) vagy üzenethez rendelt értékek csoportja az adatbázisban.
Mező (field): A mező a legkisebb, névvel ellátott adat a Sapiensben. Rendelkezik valamilyen értékkel, mely érték leírja az objektum egy jellemzőjét. A mező lehet akár azonosítója vagy egy tulajdonsága az objektumnak.
Képernyőblokk (Screen Block): Egy tranzakció eseményeinek csoportja, mely meghatározza a képernyőt vagy annak egy részét.
Képernyő (Screen): Képernyőblokkok egy csoportja.
Forma (Form): A forma egy megjelenítési objektum. Egy menü vagy összatartozó képernyőblokkok csoportja.
Attributumok (Attributes): Az attributumok feladata, hogy meghatározzák egy adat entitás (tábla, szegmens, rekord, mező) vagy egy megjelenítési forma (képernyő, tranzakció, forma, blokk) alapvető viselkedését.
Műveleti kód (Operation Code): Az adott művelet kódja, melyet minden egyes tranzakciónak tartalmaznia kell (pl.: beszúrás, módosítás, törlés, editálás stb.).

Egy alkalmazás felépítésének módjai

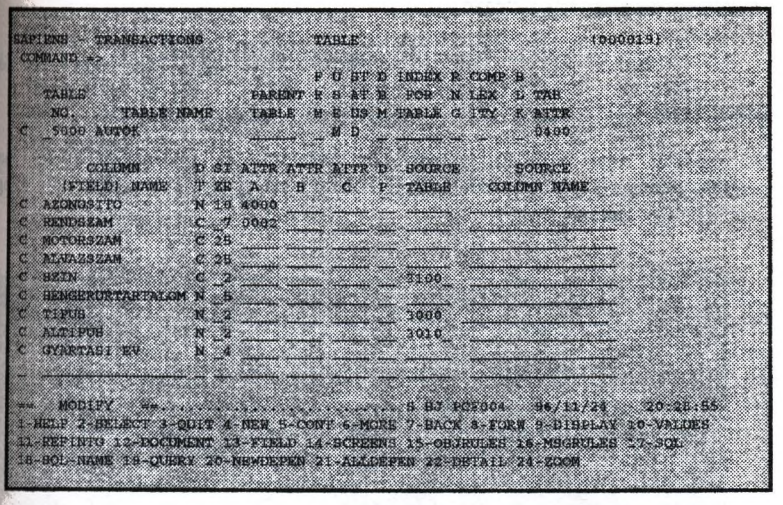
A *Sapiens* kétféle lehetőséget nyújt egy alkalmazás felépítésére:

1. A könnyebb és gyorsabb munkavégzés céljából készült egy ún. "*Könnyű Sapiens*" (Easy Sapiens), mely lényegesen leegyszerűsíti az alkalmazást fejlesztő egyén feladatát. Lényege abban rejlik, hogy az összetartozó adatokat (objektumokat) ***táblákban*** definiálja és ezen kívül automatikusan létrehozza az egyes táblákhoz tartozó ***tranzakciókat, formákat,*** stb. A későbbiek során ezek természetesen tetszés szerint módosíthatók.
Az *Easy Sapiens* tartalmazza mindazokat a szolgáltatásokat, melyeket a rendszer kínál, de használatával sok lépés leegyszerűsíthető és ezáltal az adott feladat megoldása kevesebb időt vesz igénybe.

2. Gyakorlottabb fejlesztők használhatják az ún. "*Alap Sapiens*"-t (Basic Sapiens). A különbség a két változat között az, hogy itt nincs lehetőség táblák definiálására, és minden egyes lépést egyenként végre kell hajtani a legalsó szinttől kezdődően (pl.: mező-, szegmens-, rekord-, tranzakció-, formadefiníciók).
Alkalmazása mélyebb ismereteket igényel, használata előtt célszerű az *Easy Sapiens*-el néhány egyszerű alkalmazást felépíteni.



Tábladefiníció (11. Easy Sapiens)



A képernyő két részből áll. A felső rész (Table No.,Table Name, stb.) a tábla jellemzőit mutatja, a második (Column Name, Type, stb.) a tábla mezőit írja le.

Table No.: A tábla száma, egyedi azonosító.
Table Name: A tábla neve.
Parent Table: Szülő tábla száma (ha van).

A fenti mezőket kötelező kitölteni. A többi mezőt vagy a rendszer tölti ki, vagy csak speciális esetben kell megadni.

A második rész a tábla mezőinek paramétereit tartalmazza:
Column Name: A mező neve.
Type: A mező típusa (numerikus, karakteres stb.).
Size: A mező hossza, a típus figyelembe vételével.
Attra, Attrb, Attrc: A mező szerepe. Meghatározza, hogy az adott mező milyen funkciót lát el a táblában (pl.: kulcs, entitáskulcs, stb.). Nyilvánvaló, hogy egy táblában elsődleges kulcsnak szerepelnie kell, a többi mezőt csak akkor kell szereppel ellátni, ha szükséges.
Dp: A tizedesjegyek száma.
Source Table: A mező forrástáblájának száma. Akkor alkalmazható, ha olyan mezőt kell szerepeltetni a táblában mely már másik táblában definiálásra került (pl.: mező egy törzsállományból).

A képernyő legalsó soraiban láthatók az adott képernyőn érvényes funkció billentyűk. A felsorolt billentyűk képernyőnként eltérhetnek.

A tábladefiníció következménye:

1. Automatikus tranzakció-, képernyő-, szegmens- és rekorddefiníció.

Adatfelvitel

A rendszer által definiált képernyőkön keresztül az update műveletek azonnal végrehajthatók. A példában adatfelvitel látható, melyet az "I" műveleti kód megadása tesz lehetővé (insert).

```
SAPIENS - TRANSACTION                    AUTOR                          (500000-005000)
MUVELETI KOD.....: I
AZONOSITO.....: 1234567890
RENDSZAM.....: NVA-847
MOTORSZAM.....: MOTORSZAM1
ALVAZSZAM.....: ALVAZSZAM1
SZIN.....: 3
HENGERURTARTALOM: 1984
TIPUS.....: 1
AZTIPUS.....: 1
GYARTASI EV.....: 1985
MUVELETI KOD.....: I
AZONOSITO.....: 1234567891
RENDSZAM.....: ELX-139
MOTORSZAM.....: MOTORSZAM2
ALVAZSZAM.....: ALVAZSZAM2
SZIN.....: 3
HENGERURTARTALOM: 1274
TIPUS.....: 2
AZTIPUS.....: 3
GYARTASI EV.....: 1986

--  MODIFY  --.....  S ED PCP004   96/11/24   20:32:02
1-HELP 2-SELECT 3-QUIT 4-NEW 5-CONT 7-BACK 8-FORW 24-ZOOM
```

Az egyes műveletekre vonatkozó ellenőrzéseket a rendszer automatikusan elvégzi (pl. már létező esemény beszúrása, nem létező esemény törlése stb.).

Az eljárások és azok szabályai

A Sapiensben az egyes objektumokhoz ún. eljárások kapcsolhatók, melyek szabályokból állnak. A szabályok teszik lehetővé a különböző számítások, adatmozgatások, ellenőrzések végrehajtását. Utasításkészletük rendkívül egyszerű. Akkor kerülnek végrehajtásra (alapértelmezésben), ha a szabályban szereplő legalább egy mező értéke megváltozik.

Három fő csoportjuk van:

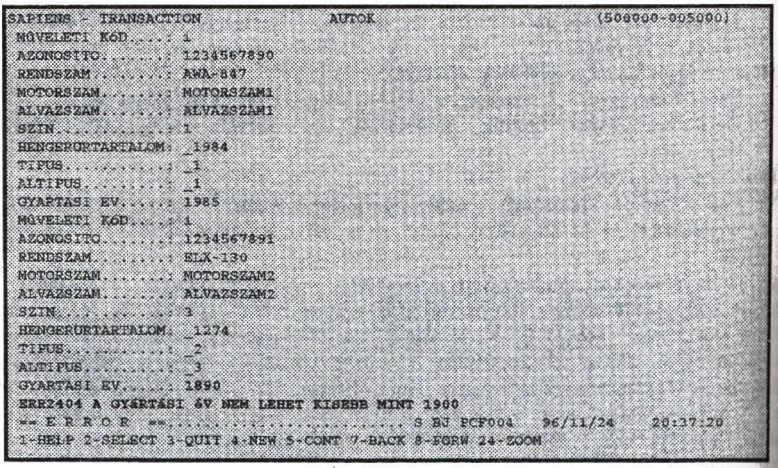
1. Műveletvégzés adott szegmensen belül:
   Local Rule (lokális műveletvézés, pl.: két mező összegzése egy harmadikba)
   Check Rule (lokális ellenőrzés, pl.: egy mező értéke nem nagyobb mint 5)
2. Műveletvégzés külső szegmensek mezőivel:
   Fetch Rule (mezők beemelése az aktuális szegmensbe másik szegmesből)
   Derivation Rule (mezők továbbítása az aktuális szegmensből másik szegmesbe).
3. Bármilyen eljárás hívása
   Call Rule (előzőleg már definiált eljárás hívása az adott szegmensből. A hívott eljárás bármilyen szabályt tartalmazhat).

Egy eljárás különböző szabályokat tartalmazhat megfelelő logikai sorrendben. A példában egy ellenőrző szabály látható mely hibaüzenetet generál, ha a gépkocsi gyártási éve kisebb mint 1900. A felső blokkban kötelezően ki kell tölteni a szabály számát és nevét, illetve attributumokon keresztül megadható, hogy a szabály milyen műveletek elvégzésekor működjön (pl.: csak beszúráskor, módosításkor és törléskor stb.). A középső blokkban meg kell adni a hibaüzenetet mely akkor generálódik, ha a szabály által ellenőrzött feltétel nem teljesül. A harmadik blokkban kerül megadásra a szabály által elvégzendő utasítások (jelen esetben ellenőrzés).

```
SAPIENS - TRANSACTIONS                CHECK RULE                        (000771)
 COMMAND =>
 C PROCESS: 500000    AUTOK ELJARAS
   RULE    : 10  NAME: GYARTASI EV ELLENORZESE
   ATTRIBUTES : 0200 _____                ERROR ATTR : _____
   ERROR FIELD: _5008 GYARTASI EV      SUBS. ERROR: _____ FF: __ FROM RULE: __
                          ERROR MESSAGE
 C  A GYARTASI EV NEM LEHET KISEBB MINT 1900
    NO                          T.O.L TEXT
 C   10  GYARTASI-EV > 1900.
 _   20  ____________________________________________________________________
 _   30  ____________________________________________________________________
 _   40  ____________________________________________________________________
 _   50  ____________________________________________________________________
 _   60  ____________________________________________________________________
 _   70  ____________________________________________________________________
 _   80  ____________________________________________________________________
 _   90  ____________________________________________________________________
 _  100  ____________________________________________________________________

 ==   MODIFY   ==..................... S BJ PCF004   96/11/24    20:35:30
 1-HELP 2-SELECT 3-QUIT 4-NEW 5-CONT 6-MORE 7-BACK 8-FORW
 9-COMPILE 10-PROCESS 11-SOURCE 12-DOCUMENT 13-SCROLRUL 14-LOCAL 15-FETCH
 16-DERIVE 17-CALL 18-ERROREXP 19-COMMENTS 24-ZOOM
```

Ha egy szegmenshez egy eljáráson belül több különböző műveletet elvégző szabály is tartozik, az adott műveleti lánc csak akkor kerül végrehajtásra, ha minden egyes szabály hiba nélkül végrehajtható.

Az előbbi szabály által generált hibaüzenet a következő képpen jelenik:



A beszúrási művelet sikertelen.

A pozitív gondolkodás

A fejlesztést nagymértékben elősegíti az ún. nevezett pozitív gondolkodás. A fogalom megértésére az alábbi példa szolgál:

Adott két szegmens "A" illetve "B". "A" szegmensben AM és "B" szegmensben BM mező szerepel.
Tegyük fel, hogy minden egyes "A" szegmensbeli "update" művelet esetén a következőnek kell teljesülnie "B" szegmensben:

BM = BM + AM

BM tehát az "A" szegmens eseményeihez tartozó AM értékek összege.
Természetesen ha AM értéke változik, a változást BM mezőnek is tükröznie kell.

Ez azt jelenti, hogy:
Beszúráskor: BM = BM + AM
Módosításkor: BM = BM + (AM módosítás után - AM módosítás előtt)
Törléskor: BM = BM - AM

A fenti három esetnek megfelelő műveleteket a pozitív gondolkodásnak köszönhetően egyetlen egy szabály elvégzi. A szabályban csak a beszúráskor elvégzendő műveletet kell definiálni. Természetesen a szabály működési köre nem korlátozható, minden "update" műveletkor futnia kell.

A lekérdezések

A Sapiens "English-like" lekérdezőnyelve az ún. QUIX. Saját - egyszerűen elsajátítható - utasításkészlettel rendelkezik, mely SQL utasításokat is tartalmazhat, ha a rendszer a DB2-vel együtt használatos (DB2 tábla a Sapiensben).
Az alábbi példa egy egyszerű lekérdezést mutat a autók táblára (szegmensre).

```
SAPIENS - TRANSACTIONS              QUERY DEFINITION               (000230)
 COMMAND =>
 OPERATION CODE..: C
 QUERY NO........: 500000
 QUERY NAME......: AUTOK AHOL TIPUS 1
 QUERY LANGUAGE..: E                       USER WORLD......: _7000
 FROM QUERY......: ______                  OUTPUT FORM.....: ______
 OUTPUT TRAN.....: _____
    NO                                 SOURCE LINE
 C  __5  FOR ALL AUTOK WHERE TIPUS = 1.
 C  _10  PRINT AZONOSITO RENDSZAM GYARTASI-EV.
 C  _15  TOP 'AUTÓK, AHOL A TIPUS 1'.
 _  _20  ________________________________________________________________
 _  _25  ________________________________________________________________
 _  _30  ________________________________________________________________
 _  _35  ________________________________________________________________
 _  _40  ________________________________________________________________
 _  _45  ________________________________________________________________
        USE 'EXECUTE' TO EXECUTE CURRENT QUERY


==   MODIFY   ==.............................S BJ PCE004   96/11/24    20:44:18
1-HELP 2-SELECT 3-QUIT 4-NEW 5-CONT 6-MORE 7-BACK 8-FORW 9-EXECUTE 10-COMPILE
11-DEMOQURY 12-DOCUMENT 15-QURYTRIG 16-WORLDS 24-ZOOM
```

A lekérdezés eredménye:

```
SAPIENS ObjectPool
  DATE 96/11/24                                                 PAGE    1
                       AUTóK, AHOL A TIPUS 1
                 ----------------------------------------
                 |     |            |          | GYARTASI |
                 | NO. | AZONOSITO  | RENDSZAM |    EV    |
                 ----------------------------------------
                 |   1 | 1234567890 | AWA-847  |   1985   |
                 ----------------------------------------

                    +++ QUERY 500000 COMPLETED +++
MSG2703 *** END OF DATA - PRESS "ENTER" TO CONTINUE ***










==>
```

A menük

Egyetlen definíciós képernyő megfelelő kitöltésével menü készíthető, mely akár képernyő vagy program, lekérdezés, illetve belső Sapiens parancs végrehajtását, illetve a végrehajtás eredményét jeleníti meg a felhasználó számára.

Az "On-line help"

A Sapiens-ben az összes definiált objektumhoz illetve azok elemeihez, "on-line" segítség definiálható. Ez egészen a legalsóbb szintig (mező) megtehető.
Az elkészített segítő információ az aktuális pl. mezőn állva a PF1 lenyomására megjelenik.

Az alkalmazás fejlesztése során a rendszer is rendelkezik Help-ekkel, melyre példát a következő ábrán láthatunk.

Segítség kérés a Tábladefiníciós képernyőn állva a DT (Data Type) mezőről:

```
SAPIENS ObjectPool
*** HELP FOR FIELD 1911 - ENTER DATA TYPE : DATA TYPE=C, INPUT LEN=001 ***
          The type  code determines the way  in which the  values of
          the field are stored in the  database.  There are a number
          of possible type code entries;

          (1) "N"  - numeric  field.  The  actual storage  type  is
              determined  by  the  field size  (the  length  of  the
              field)

               SIZE        CODE        MEANING
               1             Z         zoned decimal
               2-9           B         binary
               10-15         P         packed decimal

          Note that the numeric type may also be determined directly
          by entering the relevant code (see Z, B and P below).

          (2) "T"  - date  field.  The  "T" type  entry selects  the
              proper date format dependent on  the value entered for
              SIZE.  When "T" is entered, the allowed values for SIZE
              are 2, and 4 to 11.

===>
```

Az általunk készített Help-ek is hasonló módon jelennek meg a képernyőn.

A SAPIENS GRAFIKUS ALAPU FEJLESZTŐ KÖRNYEZETE: AZ OBJECT MODELER

A Sapiens fejlesztő rendszernek a Main Frame-s környezeten kívül számos, az alap környezethez kapcsolódó szolgáltatása is van:

- *SAPIENS WORKSTATION*

  Lehetővé teszi, hogy a nagygépes 3270 típusú képernyőket és az Object Modeler-el készített ablakokat grafikus felhasználói felületen keresztül láthassuk.
  A SWS rendelkezik még egy úgynevezett List Maneger-el is, amely a Main Frame-en írt lekérdezések használata esetén a listákat letárolja a PC-n egy Word-ben vagy Exel-ben használható formátumban.

- *SAPIENS OBJECT MODELER*

  Egy CASE típusú fejlesztőeszköz, mely segítségével egyszerűen, áttekinthetően modelezhetünk le bármilyen gazdasági folyamatot kiegészítve az Object Pool szabályleíró nyelvével és lekérdezési lehetőségeivel.

- *ODBC DRIVER*

  A Main Frame-es Object Pool által vezérelt összes adatbázis adata bármilyen ODBC-kompatibilis eszközből (MS Exel, MS Word, Borland Delphi, MS Visual Basic) elérhetővé válik a PC-n.

- *DDE KAPCSOLATOK*

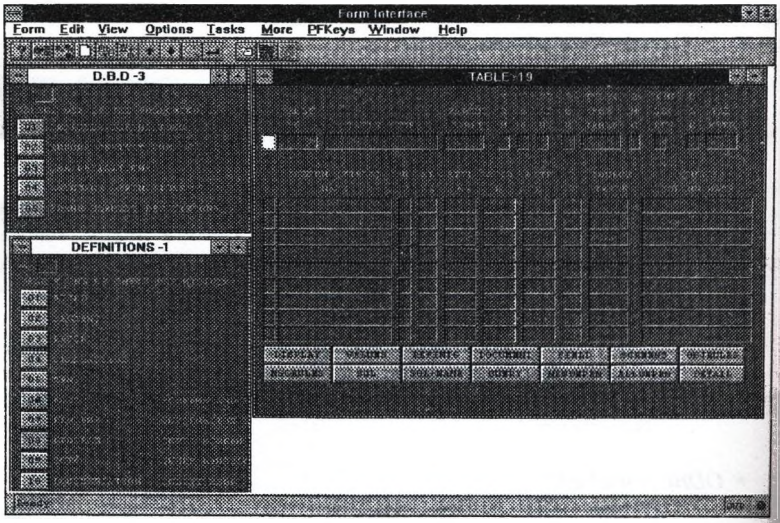
  Mind az Object Modeler, mind a Workstation képes dinamikus adatkapcsolatot kialakítani a Windows-os alkalmazásokkal (MS Exel, MS Word).

Az összes PC-s kapcsolat során a PC és a Main Frame között csak a tényleges felhasználói adatok közlekednek, az objektumok (képernyők, struktúrák) leírásait egy lokális adatbázis tartalmazza a PC-n, vagy a PC-s server-en.
Ha valamilyen módosítás történt a Main Frame oldalán az egyes objektumokon, akkor az objektum jellemzői az első használat során a tényleges felhasználói adatok feltöltése előtt automatikusan feltöltődnek a PC-re.
A felhasználói adatok csak egyetlen helyen, a Main Frame-en vannak tárolva.

SAPIENS WORKSTATION

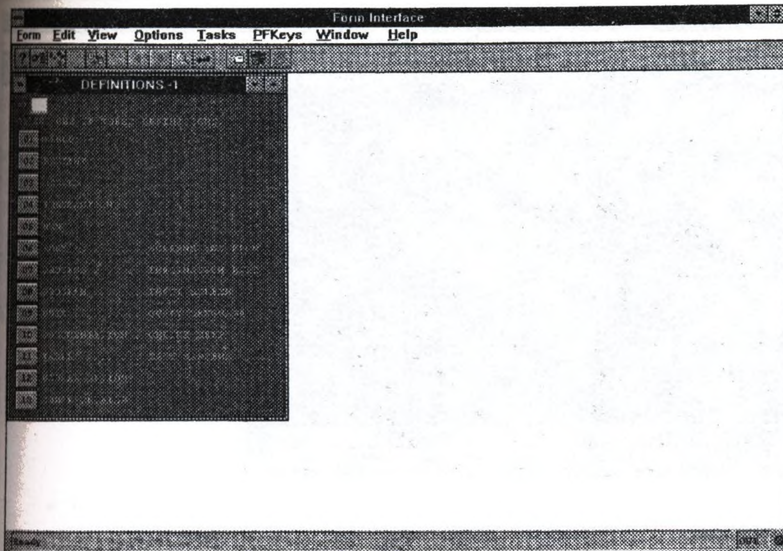
Ez a felhasználói felület egy grafikus megjelenítést tesz lehetővé a Main Frame-es képernyők alapján.

Az egyes képernyőknek megfelelő Windows-os ablakok természetesen együtt is szerepelhetnek a munka felületen, ezáltal áttekinthetőbbé téve az egész folyamatot.

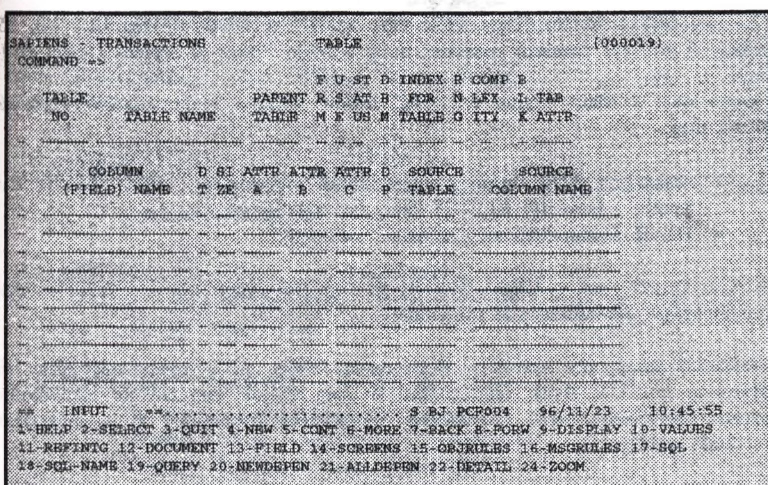


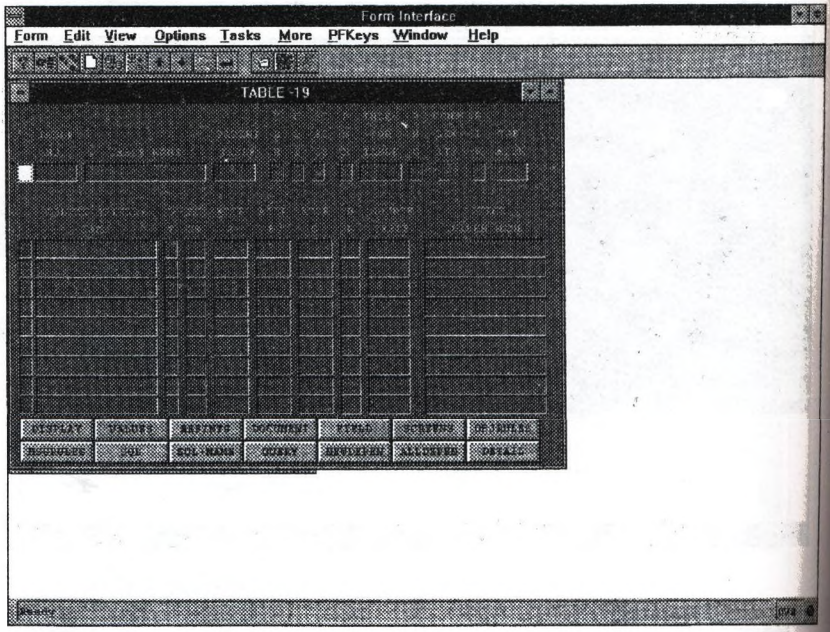
A következő ábrákon egy menü, és egy adatkezelő képernyő látható Main Frame-es és PC-s környezetben.

```
SAPIENS - M E N U                    DEFINITIONS                        (000001)
 COMMAND ==>
 SELECT ONE OF THESE DEFINITIONS:
   01 FIELD
   02 SEGMENT
   03 RECORD
   04 TRANSACTION
   05 MENU
   06 FORM ..... SCREENS AND FLOW
   07 PROCESS ..... INFORMATION RULE
   08 PROGRAM ..... INPUT SCREEN
   09 QUIK ..... QUERY LANGUAGE
   10 DOCUMENTATION . ONLINE HELP
   11 TABLE ..... EASY SAPIENS
   12 GLOBAL OPTIONS
   13 ERROR DETAILS




 ==>    MENU    ==>.......................... S BJ PCP004   96/11/23    10:40:46
 1-HELP 2-SELECT 3-QUIT 5-CONT 24-ZOOM
```



Adatkezelő képernyők:





A Sapiens Object Modeler egy PC-s környezetben (OS/2 , Win 3.1, Win 95) használható grafikus fejlesztő rendszer, mely a Main Frame-es Sapiens Object Pool-on alapul.
A kapcsolatot a Main Frame és a PC között kétfajta kommunikáció segítségével valósítja meg:

- valamilyen 3270 típusú kommunikációs software-en (IBM PC3270, Extra, Rumba) keresztül HLLAPI kapcsolat segítségével, vagy
- TCP/IP kommunikációs protokolon keresztül.

A Sapiens Object Pool az Object Modeler-en keresztül fejlett grafikus felületet biztosít az alkalmazások elkészítésére és irányítására.

A fejlesztes során egy szabvány Windows-os felület áll a rendelkezésünkre, mely segítségével grafikus ábrák felhasználásával meg tudjuk tervezni az alkalmazásunk objektum szerkezetét, azok kapcsolatait, megjelenési formájukat és a működésüket befolyásoló szabályokat.

Az Object Modeler RAD fejlesztőeszközt alkalmazva azonnal, az elemzés és tervezés kezdeti szakaszától kezdve minden egyes módosítás a modelen azonnal láthatóvá válik a kész alkalmazásban is.

A model bármelyik elemének tulajdonságait a rendszer működése közben is meg lehet változtatni anélkül, hogy ez bármilyen problémát okozna a működés során.
A hibák feltárása után a javítások azonnal elvégezhetők a rendszeren, így lényegesen lecsökken az alkalmazói rendszerek "behangolásának" az ideje.
A gyakorlatban egy Windows ablakban folytathatjuk a fejlesztést, míg egy másik ablakban a kész eredményt tesztelhetjük párhuzamosan.
Az Object Modeler esetében a RAD-ciklus kiterjesztették az elemzés fázisára is, ezért az elméleti model online módon a működési ciklus tetszőleges pillanatában módosítható.

A modelalkodtás során három alapvető Objektum model közül lehet választani:

- ***BASIC OBJECT***

  Ez az objektum típus határozza meg az alkalmazói adatok csoportját, az adatok tárolási jellemzőit és kapcsolatrendszerüket.
  Ez az ábrázolás kiterjesztett ER diagramm formájában történik.

- ***PRESENTATION***

  A Basic Objektumban leírt adatok megjelenítését szabályozza (képernyők felépítése, GUI elemek használata), kiegészítve a megjelenítés kapcsolatrendszerével (menük, képernyőfolyamok).

- ***RULES***

  Mind a Basic, mind a Presentation szinthez lehet megadni olyan eljárásokat és szabályokat, melyek azok működését befolyásolják.

Az egyes objektumoknak három állapota lehet, melyeket a képernyőn különböző színek jelölnek:

- ***ANALISYS*** ***(KÉK)***

  Alap objektumok definiálása, típusuk meghatározása kapcsolatrendszerük felépítése.

- ***DESIGN*** ***(SÁRGA)***

  A kapcsolatrendszerek megjelenésének tervezése, megjelenítési felületek kialakítása.

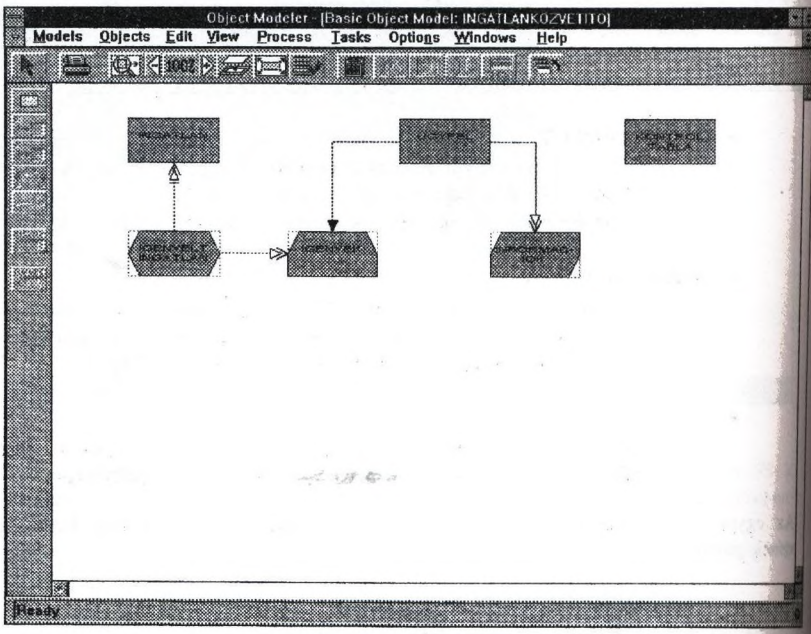
- ***IMPLEMENTATION*** ***(ZÖLD)***

  A model fizikai elemeinek felépítése (táblák, tranzakciók, ...), tárolása a Main Frame-en lévő Tudásbáziba.
  Tesztelés és a model használatbavétele.

Az egyes Objektumokat grafikus elemek szemléltetik, elősegítve ezzel a model áttekinthetőségét.

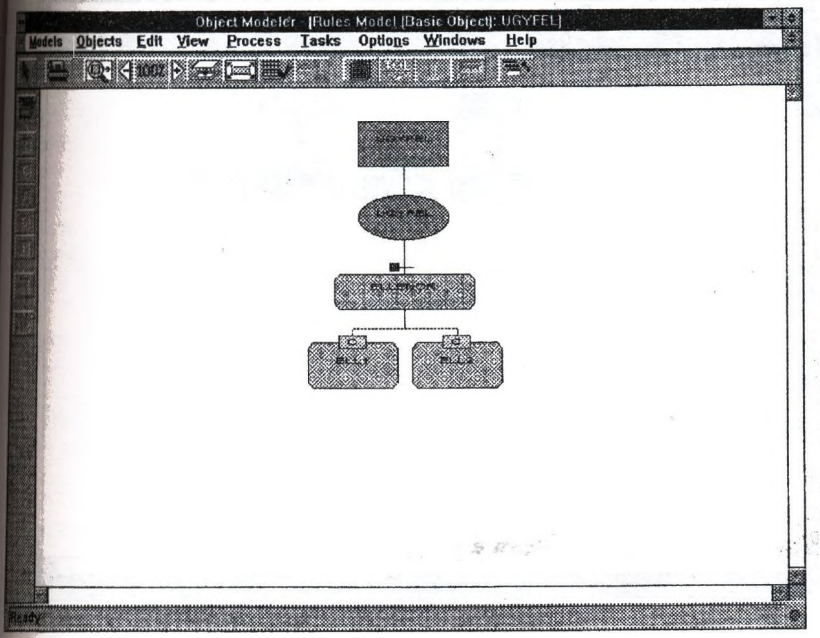
Mint az a bemutatásra kerülő példa is jól látható az alkalmazás struktúrája áttekinthető, és az egyes elemek kapcsolatai is jó látható.

BASIC MODEL



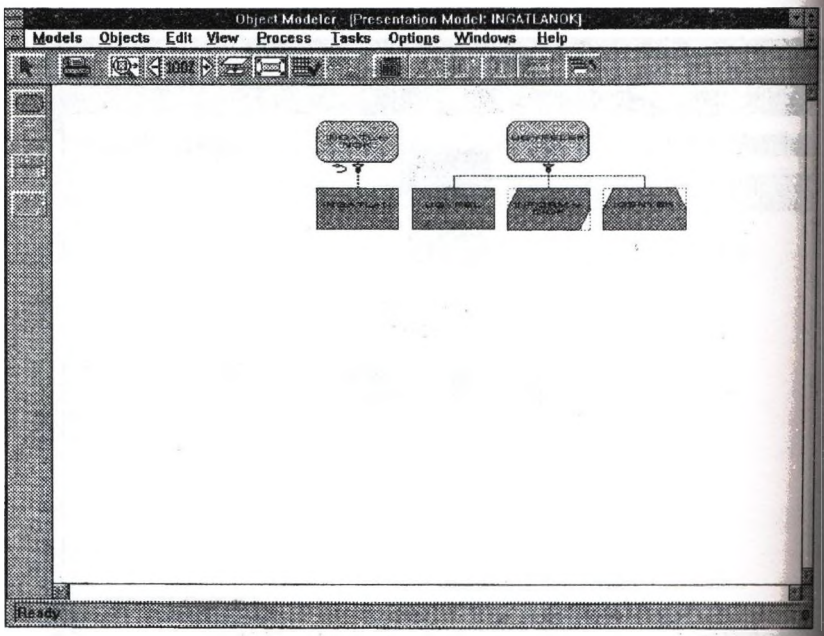
Az ábrán egy Ingatlanközvetítő iroda működését segítő nyilvántartó rendszer Basic model-je látható.
Az ábra bal oldalán látható oszlop tartalmazza a felhasználható objektumokat és azok paramétereit beállító gombokat.
Minden esetben csak az a funkció használható, ami az adott objektum szinten elérhető, a többit a rendszer automatikusan zárolja.

RULES MODEL



Itt szintén az ablak bal oldalán találhatóak az objektum működését befolyásoló kapcsolók.
A szabálytípust a hozzá tartozó eljárás kiválasztása után lehet megadni, majd a TOL gomb lenyomása után a tényleges szabály szöveget rögzíteni.
A hierarchikus struktúra jól mutatja, hogy az egyes szabályok hogyan kapcsolódnak egymáshoz, mi a végrehajtási sorrendjük (balról jobbra haladva következnek egymás után), milyen Basic objektumhoz kapcsolódnak, és ha van, akkor milyen másik objektumon végeznek műveletet, vagy hoznak onnan adatot (derivációs és fetch szabályok).

PRESENTATION MODEL



A Presentation modelnél egy megjelenítési képernyőfolyamot hozhatunk létre a felhasználók számára, melyen keresztül elérhetik az adatbázis objektumait, azokon műveleteket hajthatnak végre, lekérdezéseket használhatnak különböző paraméterek szerint.
Az egyes képernyők meghívásának típusait (menüből, egymást követően, valaminyen Pf billentyű hatására) az ábrákon elhelyezett kis nyilak jelölik.
A menü-beli vagy megjelenítési sorrend itt is balról jobbra halad.
A különböző megjelenítési paraméterek beállítási képernyőjét az adott grafikus elemen duplát kattinva érhetjük el.
A megjelenítést befolyásoló paraméterek természetesen máshogy hatnak egy 3270 típusú és egy Windows-os képernyőn (pl. listbox, select-box, ...).

Elegendő hely hiányában az itt közölt előadás anyag csak tematikai jellegű, a konferencia során bemutatásra kerülő ismeretek természetesen tágabb képet adnak a Sapiens Object Pool és Object Modeler felépítéséről és használatáról.

Dunaújváros, 1996 November 23.

A SAPIENS RENDSZER

LEHETSÉGES KAPCSOLATAI

RADA JÓZSEF

DUNAFERR RT

SZÁMÍTÁSTECHNIKAI ÉS SZOLGÁLTATÓ INTÉZET

A DUNAFERR RT, illetve a DUNAI VASMÛ már évtizedek óta különbözõ berendezésekkel, különbözõ operációs rendszerekkel, különbözõ feldolgozási módszerekkel végzi adatainak feldolgozását. Már kezdetben a nagygép, vagy ahogyan ma divatos mondani a mainframe irányában indult el a vállalat - természetesen nálunk is megjelentek a személyi számítógépek, a UNIX alkalmazások, stb. Az informatika sokszínûségét - és egyben bonyolultságát többek között nálunk is ez okozza. A rendszerek megfelelõ szintû integráltsága nélkül a rendelkezésre álló különbözõ adatok redundánsak, átláthatatlanok, gyakorlatilag használhatatlanok lesznek, nem felelnek meg funkcióiknak.

A rendszerváltás természetesen a DUNAFERR-t is érintette, mely az egykori nagyvállalat több kisebb társasággá alakulásában nyilvánult meg leglátványosabban. Ez az informatikában is jelentõs követelményeket eredményezett kezdve a más tipusú felhasználói igényektõl egészen a különbözõ jogosultságokig, mely addig - egy nagyvállalatról lévén szó - semmiféle problémát nem jelentett. Más aspektusból kellett tehát a - most már - vállalat csoport informatikai jövõjét tekintenünk.

Az 1990-es évek elején kialakított DUNAFERR koncepció szerint az addig kidolgozott, de az igényeknek és követelményeknek már csak részben, vagy egyáltalán nem megfelelõ nagygépes rendszereket kétféle módon kell kiváltani:

- A termelésirányítás és értékesítés területén SAPIENS alapú saját fejlesztésû rendszerekkel

- Az összes többi - pénzügy, számvitel, költség-, anyaggazdálkodás, stb. - területen pedig az SAP rendszer különbözõ moduljainak bevezetésével

A SAPIENS szabály alapú objektum orientált adatbáziskezelõ rendszer, mely hosszú tesztelést követõen került cégünknél megvételre. Szakítva a nagygépes környezetben még mindig domináns módszerekkel - például a programozás - különbözõ ún. szabályok alkotásán és attributumok beállításán alapszik, mely az eddig használt fejlesztési eljárásokhoz képest egy felhasználói rendszer kidolgozási idejét nagyságrendekkel rövidíti meg.

A SAPIENS erõforrás igénye teljesítményéhez mérten minimálisnak mondható.
Az alábbi minimális konfigurációval a rendszer képes mûködni:

A disk igények a következõk:

- 26 MB a SAPIENS tudásbázis számára
- 38 MB a rendszer könyvtárai számára

Központi tárigény :

- CICS használatakor : 0.3 MB UDSA a 16MB-os vonal alatt és1.8 MB EDSA a 16 MB-os vonal fölött
- IMS/DC esetében : 2.5 MB
- TSO használatakor pedig 3 MB felhasználónként

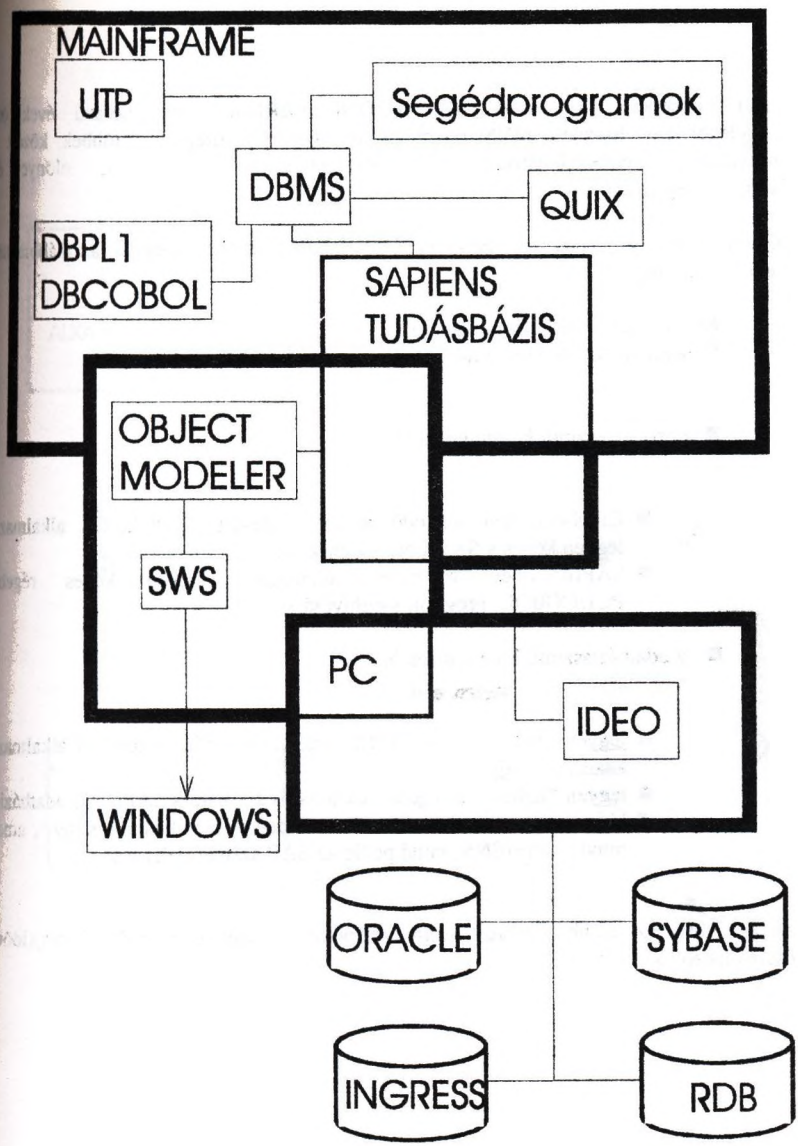
A PC-s környezet igénye 486/DX 66 MHz és 16 MB RAM,
MS-DOS 3.1, MS Windows 3.1, kommunikációs software, kommunikációs kártyával (IBM 3270 például)

Egyéb erõforrás igény a megtervezett és kidolgozott adatbázis méretétõl, valamint az egyidõben aktív felhasználók számától függ.

Hatalmas elõnye, hogy rendkívüli módon hordozható a különbözõ platformok között, úgymint az MVS, a VSE, a VM, vagy éppen a UNIX. Ma már megfelelõ teljesítményû, felhasználóbarát személyi számítógépes felületei is vannak. Ezek a SAPIENS WORKSTATION, OBJECT MODELER, IDEO.

A SAPIENS fõbb komponensei és azok alavetõ kapcsolatai az 1. ábrán láthatók.

- UTP : univerzális tranzakció kezelõ, mind batch, mind on-line módon mûködhet
- Segésprogramok : a mentéstõl a henyreállításon keresztül az adatbázisok struktúrájának megváltoztatásáig különféle segédeszközök állnak rendelkezésre
- DBMS : maga az adatbáziskezelõ rendszer a SAPIENS
- DBPL1/DBCOBOL : lehetõség van PL1 illetve COBOL nyelven írt programok illesztésére
- QUIX : egy "angol szerû" egyszerûen kezelhetõ lekérdezõ nyelv
- SAPIENS TUDÁSBÁZIS : a megalkotott képernyõk, definíciók, szabályok, az alkalmazások halmaza
- Object modeler : alkalmazás fejlesztésének PC-s eszköze
- SWS: SAPIENS WORKSTATION, a nagygép felé interface, a PC-s fejlesztést és használatot segítõ eszköz
- IDEO : egy PC-s adatbáziskezelõ rendszer, mely híd a nagygépes SAPIENS adatbázis és a különbözõ egyéb felsorolt adatbázisok között


MAINFRAME
UTP
Segédprogramok
DBMS
QUIX
DBPL1
DBCOBOL
SAPIENS
TUDÁSBÁZIS
OBJECT
MODELER
SWS
PC
IDEO
WINDOWS
ORACLE
SYBASE
INGRESS
RDB

1. ábra

Fenti koncepció figyelembe vette a DUNAFERR-nél már hosszú évek óta megelégedéssel használt MVS operációs rendszer lehetõségeit - többek között a minimális zökkenõkkel járó átmenet megvalósításának érdekében -, elõnyeit és hátrányait egyaránt.

Ki kellett alakítani tehát egy többrétegû kapcsolati rendszert, melynek az alábbiaknak kellett megfelelni:

- programszintû kommunikáció
- adatbázisszintû kommunikáció

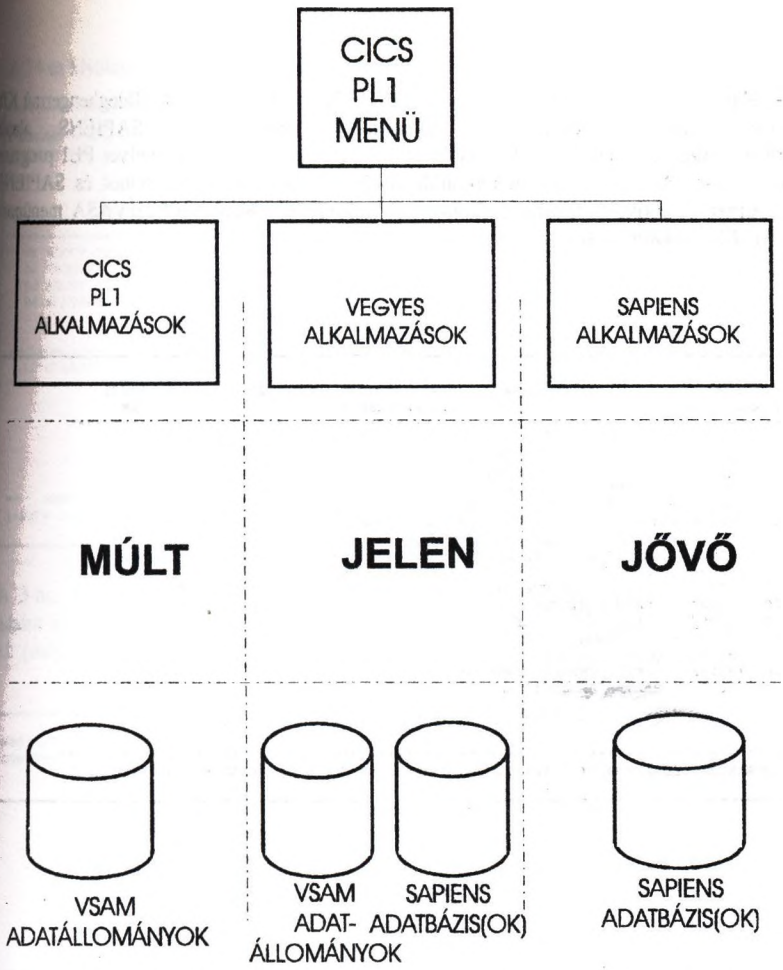
- a programszintû kommunikáció

  - CICS-ben már mûködõ régebben megírt PL1/COBOL alkalmazás legyen képes a SAPIENS meghívására
  - SAPIENS-ben fejlesztett alkalmazás legyen képes régebbi PL1/COBOL program meghívására

- az adatbázisszintû kommunikáció

  - legyen "látható" a SAPIENS rendszer számára a régebbi alkalmazás adatstruktúrája
  - legyen "látható" a régebbi alklamazás számára a SAPIENS adatbázis
  - legyen olyan adathalmaz - jelen esetben VSAM adatállomány -, amely mind a SAPIENS, mind pedig az SAP számára elérhetõ

A 2. Ábra az elõbb felsorolt kommunikációs igényeket szemlélteti az idõvel összekapcsolva.


CICS
PL1
MENÜ
CICS
PL1
ALKALMAZÁSOK
VEGYES
ALKALMAZÁSOK
SAPIENS
ALKALMAZÁSOK
MÚLT
JELEN
JŐVŐ
VSAM
ADATÁLLOMÁNYOK
VSAM
ADAT-
ÁLLOMÁNYOK
SAPIENS
ADATBÁZIS(OK)
SAPIENS
ADATBÁZIS(OK)

2. ábra

A feladat tehát egy már CICS-ben mûködõ - jelen esetben a DWA Hideghengermû Kft gyártáskövetési rendszere - felhasználói rendszerének SAPIENS alapú továbbfejlesztése, bõvítése. Az alábbi menü egy BMS képernyõ, melyet PL1 program küldött ki. A választható menüpontok között található PL1 programot és SAPIENS alkalmazást aktiváló egyaránt. A 74-es RAKTÁR TÉRKÉP MEGHÍVÁSA menüpont SAPIENS alkalmazást indít.

```
DUNAFERR            HIDEGHENGERMUI GYARTASKOVETESI RENDSZER        96/11/21
DUNAI VASMU                  VALASZTHATO TRANZAKCIOI                 HTEK
********************************************************************************

61  /HH12/ - REGISZTRATIV ALLOMANYOK LETREHOZASA                    BODA
62  /HH16/ - TERVEZETT ADATOK ROGZITESE                             BODA
63  /HH23/ - RENDSZER NYOMTATAS                                     PINTER
64  /HH24/ - ALTALANOS KARBANTARTAS                                 GOBOLOS
65  /HJ00/ - HENGERLESI TETELES TERMELESI PROGRAM                   RADA
66  /HS01/ - EXPORT ERTEKESITESI RENDSZER
67  /HS02/ - KESZLETRE VETEL ES "B" TIPUSU TIKETT NYOMTATASA        GUMINAR
68  /HS03/ - REP. OLLO HULLADEK ADATINAK ROGZITESE                  GUMINAR
69  /HS04/ - PACOLOI VEGLEMEZ ADATAINAK FELVITELE                   GUMINAR
70  /HS08/ - UZEMNEK VISSZA                                         GUMINAR
71  /HS09/ - LELTAROZAS MEGHIVASA                                   GOBOLOS
72  /HS10/ - VISSZARU JEGYZOKONYV                                   GUMINAR
73  /HS13/ - QUERIK KEZELESE                                        GUMINAR
74  /HS14/ - RAKTAR TERKEP MEGHIVASA                                GOBOLOS
75  /HS15/ - RAKTAR AKTUALIZALASA                                   GOBOLOS

              KEREM A VALASZTOTT TRANZAKCIO SZAMAT :      74

********************************************************************************
KOVETKEZO OLDAL:<SHIFT + PF8>,ELOZO OLDAL:<SHIFT + PF7>, KILEPES:<SHIFT + PF3>
```

A 74-es kiválasztása után az alábbi képernyõt kaptuk, melyet már a SAPIENS adott ki.

```
SAPIENS - M E N U                    LELTAR MENU                        (160009)
COMMAND =>
                              RAKTAR TERKEP MENU
                              ==================

VALASSZON AZ ALABBIAK KOZUL:
============================================================
  01 ADOTT TETEL TAROLASI HELYE
  02 TETELEK TAROLASI HELYE
  03 ADOTT RAKTAR KOTEGEI
  04 RAKTARHOZ ES TAROLASI HELYHEZ
                              TARTOZO TETEL

============================================================
  12 KILEPES




==   MENU    ==..................... R0990 S B1 HTEKT     96/11/21 13:43:55
1-HELP 2-SELECT 3-QUIT 5-CONT
```

A 3-as, ADOTT RAKTÁR KÖTEGEI menüpont kiválasztása után az alábbi input adatot kérõ képernyõ jelenik meg, ahol az alábbiak szerint a 02-es raktárt választottuk ki. (még mindig a SAPIENS alkalmazás dolgozik)

```
SAPIENS - QUERY-INPUT          LELTAR MENU-RAKT.TERKEP-KOTE       (160027-160001)
COMMAND =>

            RAKTAR TERKEP - TETELEK LISTAZASA ADOTT RAKTARRA
            ===============================================

                              RAKTAR SZAMA: 02



KEREM IRJA BE A RAKTAR SZAMAT...

MEGJEGYZES
 HA NYOMTATNI KIVAN, ELOSZOR NYOMJA MEG A <PF9> -ET,
 AZUTÁN TOLTSE KI A KEPERNYOT.


==   INPUT   ==..................... R0990 S B1 HTEKT     96/11/21 13:47:52
1-HELP 2-SELECT 3-QUIT 4-NEW 5-CONT 9-NYOMTAT
```

A query eredménye az alábbi képernyõn látható.

```
SAPIENS - APPLICATION GENERATOR
   DATE  96/11/21                                                PAGE     1
                    ADOTT TÁROLÁSI HELYEN LÉVÕ KÖTEGEK
                  RAKTÁR SZÁM:   2  TÁROLÁSI HELY:    1
             ---------------------------------------------------
             |      |   TÉTEL    | TERMÉK |                     |
             | NO.  | AZONOSÍTÓ  | FAJTÁJA |   TÁROLÁSI ÁLLAPOT  |
             ---------------------------------------------------
             |    1 |  3508201   | TÁBLA  | VÁGOTT KÉSZLET      |
             |    2 |  3508202   | TÁBLA  | VÁGOTT KÉSZLET      |
             |    3 |  3508204   | TÁBLA  | VÁGOTT KÉSZLET      |
             |    4 |  3886001   | TÁBLA  | VÁGOTT KÉSZLET      |
             |    5 |  3886002   | TÁBLA  | VÁGOTT KÉSZLET      |
             |    6 |  3886003   | TÁBLA  | VÁGOTT KÉSZLET      |
             |    7 |  3886004   | TÁBLA  | VÁGOTT KÉSZLET      |
             |    8 |  3886005   | TÁBLA  | VÁGOTT KÉSZLET      |
             |    9 |  3951006   | TÁBLA  | KISZÁLLÍTÁS ÁTVETTE |
             |   10 |  3951109   | TÁBLA  | KISZÁLLÍTÁS ÁTVETTE |
             |   11 |  4781001   | TEKERCS | KISZÁLLÍTÁS ÁTVETTE |
             |   12 |  4826201   | TEKERCS | KISZÁLLÍTÁS ÁTVETTE |
             |   13 |  4826801   | TEKERCS | KISZÁLLÍTÁS ÁTVETTE |
             |   14 |  4825701   | TEKERCS | KISZÁLLÍTÁS ÁTVETTE |
             ---------------------------------------------------
==>
```

Az előző képernyőn látható eredményt az alábbi egyszerű query készítette. Ez a kis példa a programok közötti kommunikációt, valamint a SAPIENS-ből látható nem SAPIENS adatbázis adatok elérhetőségét szemléltette.

```
SAPIENS - TRANSACTIONS              QUERY DEFINITION              (000230)
COMMAND =>
MUVELETI KOD....: C
QUERY NO........: 160001
QUERY NAME......: RAKT.TERKEP- TAROLASI HELY
QUERY LANGUAGE..: E                        USER WORLD......:   2
FROM QUERY......: ______                   OUTPUT FORM.....: ______
  NO                                SOURCE LINE
C  5  FOR ALL A475149
C 10    WHERE (TH @TH).
C 11  BREAK FRAME BY H P.
C 12  PRINT  K T INF.
C 13  DCL K NUM 7 TITLE'TÉTEL AZONOSÍTÓ'
C 14                                    AS KAZON OF A475140 USING(KAZON).
C 26  DCL H PIC'ZZ' TITLE'RAKTÁR SZÁM:' AS TH OF A475149.
C 27  DCL P PIC'ZZZ' TITLE'TÁROLÁSI HELY:' AS THPOZ OF A475149.
C 30  DCL T CHAR 7 TITLE'TERMÉK FAJTÁJA' CASE
C 31  WHEN TERM OF A475140 EQ 'E' THEN T AS 'TEKERCS'
C 32  WHEN TERM OF A475140 EQ 'A' THEN T AS 'TÁBLA'
C 33  OTHER T AS ' '.
C 40  DCL INF CHAR 20 TITLE 'TÁROLÁSI ÁLLAPOT' CASE
C 41  WHEN TH EQ 0 AND (THPOZ EQ 5 OR SZSZ OF A475140 NE 0) THEN
C 42       INF AS 'KI VAN SZÁLLÍTVA'
C 43  WHEN TH NE 0 AND AJSZ OF A475140 EQ 0 THEN
C 44       INF AS 'VÁGOTT KÉSZLET'
C 45  WHEN TH NE 0 AND AJSZ OF A475140 NE 0 THEN
C 46       INF AS 'KISZÁLLÍTÁS ÁTVETTE'
C 47  OTHER INF AS 'NINCS BERAKTÁROZVA'.
      USE 'EXECUTE' TO EXECUTE CURRENT QUERY

= MODIFY   ==..................... R0990 S B1 HTEKT    96/11/21 13:59:04
1-HELP 2-SELECT 3-QUIT 4-NEW 5-CONT 7-BACK 8-FORW 9-EXECUTE 10-COMPILE
11-DEMOQURY 12-DOCUMENT 15-WORLDS 24-ZOOM
```

Az első SAPIENS képernyőn a KILÉPÉS menüpontot kiválasztva a kiinduló PL1 menübe mehetünk vissza.

Fenti kis feladat jól mutatja, hogy a meglévő onüline rendszereinket korszerűsíthetjük, átalakíthatjuk, kicserélhetjük szinte anélkül, hogy azt a felhasználó észrevenné.

DIGITÁLIS ALAPTÉRKÉPI ADATÁLLOMÁNYOK KEZELÉSE ORACLE RENDSZERBEN

Dr. Mihály Szabolcs, Szendrő Dénes, Rátkai Györgyné dr.

Földmérési és Távérzékelési Intézet
1149 Budapest, Bosnyák tér 5., Tel.: 222-5116, Fax: 222-5112

Helyzetkép az adatbázis alkalmazásának környezetéről

Az ország ingatlan-nyilvántartási és kataszteri térképi ellátása az állami földügy és térképészet keretében történik, amely a Földművelésügyi Minisztérium szervezetében működő Földügyi és Térképészeti Főosztályból, a Földmérési és Távérzékelési Intézetből, továbbá a földhivatalok hálózatából áll. Az utóbbi 19 megyei és 115 körzeti földhivatalból, valamint a fővárosi, s ennek kerületeit ellátó földhivatalokból tevődik össze. A földhivatalok területi illetékességének megyei és körzeti határait a mellékelt ábra tartalmazza.

Az egyes körzeti földhivatalok által kezelt és az országosan nyílvántartott - a fővárost nem tartalmazó - adatmennyiségre a következő táblázat jellemző:

	Körzeti földhivatalok	*Országos összesítés*
Lakosok száma:	24 000 - 220 000	8 746 355
Térképek száma:	300 - 1 100	61 811
Földrészletek száma:	23 000 - 106 000	5 798 328
Tulajdonlapok száma:	24 000 - 140 000	6 676 737
Kérelmek évenkénti száma:	11 000 - 85 000	1 790 368
Számítógépesített földrészletek száma:	900 - 16 000	692 503
Térkép aktualizálások száma:	300 - 2 200	12 000

A földhivatalok által kezelt várható országos adatmennyiség 3 - 5 terabájt közötti értékre becsülhető.

A fenti adatok gyűjtése, kezelése és szolgáltatása hosszú időn keresztül analóg módon történt. Jelenleg folyamatban van az állami földügy és térképészet korszerűsítése, amelynek egyik legjelentősebb része a "Térképen Alapuló Kataszteri Rendszer Országos Számítógépesítése" (TAKAROS) és ennek kommunikációs hálózatba történő foglalása (TAKARNET) PHARE támogatással. Lényeges eleme az automatizálásnak az ingatlan nyilvántartási adatok és a kataszteri térképek folyamatban lévő digitalizálása, amelyhez nélkülönözhetetlen elemként szolgált a FÖMI keretei között OMFB és FM támogatással kidolgozott Digitális Alaptérkép (DAT) szabvány és szabályzat-rendszer. Ez utóbbi részletes utasításban írja le és szabályozza a fenti számítógépesítéshez is illeszkedő adatbázist.

Az adatbázis célja

A digitális alaptérképi adatállományokat kezelő adatbázis szerkezetének létrehozására, az adatbázis feltöltésére, az adatok megadott szempontok szerinti lekérdezésére és az adatállományok belső konzisztenciájának vizsgálatára és hitelesítésére szolgáló szoftver a megyei földhivataloknál történő használatra került kifejlesztésre. Célja az, hogy a digitális alaptérképi adatok tárolása, minőségellenőrzése és állami átvétele az országban egységes módon valósuljon meg. Kidolgozása az MSZ 7772-1 Szabvány és a DAT1. Szabályzat, valamint a földhivataloknál lévő hardver-, szoftver- és adatbáziskezelő rendszer figyelembe vételével történt meg.

Az adatbázis műszaki jogszabályi környezete

Az MSZ 7772-1 Szabvány a digitális alaptérkép (DAT) fogalmi modelljéről rendelkezik. Ez tartalmazza a fogalom-meghatározásokat, a geodéziai alapokat, a DAT-ban szerepeltetendő objektumok és attribútumaik leírását, az objektumok geometriai leírásmódját, a kapcsolatok alapvető formáit, az adatminőségi követelményeket, a DAT adatállományok ismertetésére szolgáló metaadatok leírását és a megjelenítés modelljét. Összhangban van az európai térinformatikai szabványosítással.

A DAT1. Szabályzat a digitális alaptérképek tervezésének, előállításának, változásvezetésének, dokumentálásának, minőségellenőrzésének, hitelesítésének és állami átvételének folyamatáról valamint adatszerkezetéről és adatcsere-formátumáról rendelkezik. Leírást tartalmaz még az MSZ 7772-1 Szabványhoz szükséges kiegészítő és részletező információkról is.

A DAT1. Szabályzat DAT-M1. jelű melléklete részletesen leírja a digitális alaptérképi adatbázis adattáblázatainak szerkezetét, tartalmát, formátumát és kezelésük módját, valamint az ún. közbenső adatcsere-formátumot. Meghatározza, hogy az adattáblázatok kezelésében milyen feladatokat kell ellátnia a FÖMI-nek, a megyei és a körzeti földhivataloknak és a földmérő cégeknek. Az állami alapadat és az alapadat kategóriák szemszögéből meghatározza az adattáblázatok elemeinek kötelező és opcionális használatát. Az MSZ 7772-1 Szabvánnyal összhangban van.

A digitális alaptérképek megjelenítésekor használandó jelkulcsokról a DAT1. Szabályzat DAT1-M2. jelű melléklete rendelkezik. Tartalmazza a jelkulcsok rajzát, a gépi kirajzoláshoz szükséges geometriai adatokat és megírásokat a megjelenítés méretarányának függvényében, az elhelyezésükre vonatkozó szempontokat, az alkalmazandó betűtípusokat és méreteket, s az alkalmazási mintákat. Az MSZ 7772-1 Szabványhoz illeszkedik.

Az MSZ 7772-1 Szabvány és a DAT1. Szabályzat valamint ennek mellékletei együttes figyelembevételével előállított, és az állami földmérés szervezeteihez átvételre benyújtott digitális alaptérképi adatállományok belső konzisztenciájának vizsgálatára és hitelesítésére a DAT1-M3. jelű mellékletben leírt szoftver szolgál.

A földmérési alaptérképek analóg, numerikus és digitális adatainak digitális alaptérképpé történő átalakításáról és minőségellenőrzéséről a DAT2. Szabályzat rendelkezik. Ez igazodik az MSZ 7772-1 Szabványhoz, valamint a DAT1. Szabályzathoz és ennek mellékleteihez.

A Magyarországon használt vetületi rendszerek közötti egységes követelmények és pontosság szerinti transzformációt, annak kiinduló adatait és számítási programját a DAT2 szabályzat DAT2-M1. jelű melléklete tartalmazza.

A térképészeti és térinformatikai digitális adatállományoknak egy magasabb, térinformatikai szintű, minimális információveszteséggel járó és országosan egységes adatcseréjéről a "Magyar Térinformatikai Adatcsereformátum" című MSZ 7771 Szabvány rendelkezik.

Az adatbáziskezelő szoftver nyelve és hardverkörnyezete

A digitális alaptérképi adatállományokat kezelő adatbázis szerkezetének létrehozására, az adatbázis adatokkal való feltöltésére, az adatok megadott szempontok szerinti lekérdezésére és az adatállományok belső konzisztenciájának vizsgálatát és hitelesítését végző szoftver kifejlesztésére az ORACLE relációs adatbáziskezelő és fejlesztő rendszer lehetőségeinek kihasználásával, a nemzetközi szabványnak megfelelő SQL (Structured Query Language) nyelven került sor. Az SQL programnyelv előnye az is, hogy nem csak a földhivataloknál

meglévő ORACLE környezetben használható, mivel szinte valamennyi korszerűnek mondható grafikus és/vagy táblázatos adatbáziskezelő rendszer - így a legtöbb térinformatikai rendszer is - az ezen a nyelven történő lekérdezést és fejlesztést támogatja. Így platform-függetlensége biztosított

Az ORACLE a világon legelterjedtebben használt adatbázis-kezelő rendszer, működik például MS DOS, WINDOWS, WINDOWS-NT, APPLE, MACINTOSH vagy MOTIF környezetben, egyedi PC-ken vagy kliens-szerver hálózati kiépítettségben. Az adathozzáférést DB2, DRDA, APPC, EDA/SQL, MS EXCEL, LOTUS 1-2-3, dBASE, s ASCII állományok felöl is biztosítja. Interface-i lehetővé teszik az SQL, C, C++, COBOL, FORTRAN, PASCAL és PL/I nyelven történő programfejlesztést is.

Magyarországon az ORACLE adatbáziskezelő rendszer alapkiépítése - amely a kifejlesztett alkalmazások futtatását is biztosítja - a TAKAROS rendszer telepítése kapcsán a földhivatalokban már megtörtént. Az ORACLE teljes eszközrendszerére csak a szoftvert fejlesztő helyeken van szükség.

Az adatbázis szerkezete

Az adatok között meglévő kapcsolatokat struktúráknak tekintve adatmodellekben gondolkodhatunk. Azokat az adathalmazokat, amelyeket modellbe foglalva kezelünk, adatbázisnak nevezzük. Adatbázison voltaképpen az adatoknak tulajdonságaikkal és kapcsolataikkal együtt való ábrázolását, tárolását értjük.

A hálós adatmodell gráffal, a hierarchikus adatmodell fa struktúrával ábrázolható, míg az esetünkben alkalmazott relációs adatmodell táblázatokkal definiálható. Ez utóbbiban minden egyedhalmazt egy táblázattal adunk meg, amelyben a táblázat oszlopai a tulajdonságok, sorai pedig az egyed értékei. Az egyedek közötti kapcsolatokat a táblázatokban előforduló azonos tartalmú oszlopok (tulajdonságok) teremtik meg.

Az adatbázis akkor válik értékessé, ha megadunk hozzá egy olyan szoftvert, amellyel az adatbázist kezelni tudjuk. Ezt a szoftvert adatbáziskezelő rendszernek nevezzük.

Az adatbázisokkal két fontos műveletet kell elvégezni: az adatbázis létrehozását (szerkezetének definiálását s feltöltését konkrét adatokkal) és módosítását, valamint az adatoknak megadott szempontok szerinti visszakeresését, lekérdezését az adatbázisból.

Az adatbáziskezelő rendszerek a fő funkciókon kívül több más feladatot is ellátnak: adatvédelemről, adatbiztonságról gondoskodnak, integritási feltételek teljesülését figyelik, valamint szinkronizációt tesznek lehetővé.

Az adatvédelem, adatbiztonság fogalmán azt értjük, hogy a felhasználók csak a jogosultságuknak megfelelő adatokhoz tudnak hozzáférni, tudják azokat módosítani, illetve lekérdezni. Véd az illetéktelen lekérdezések , s főleg az adatbázis-rongálók ellen.

Az adatbázis létrehozásakor a belső szerkezetével, konzisztenciájával, azaz a tárolásra kerülő adataival kapcsolatban úgynevezett integritási feltételek fogalmazhatók meg.

Ez a módszer biztosítja, hogy az ORACLE adatbázis táblázataira és a táblázatok oszlopaira megadott megszorítások, úgynevezett kényszerek előírásával az adatok tartalmilag megengedett egyedi, vagy előírt tartományokon belüli értékekkel rendelkezzenek. Ugyancsak határozhatunk arról is, hogy egy adott táblázat valamelyik oszlopadatának kitöltése kötelező-e, vagy sem.

Feltételként előírható az is, hogy valamely táblázat egy oszlopából, vagy több oszlopának összekapcsolásából előállított kulcsnak soronkénti értékei egymástól különbözzenek, azaz egyediek legyenek. Ezekkel az egyedi kulcsokkal a táblázat hozzájuk tartozó soraiban lévő adatokra más táblázatokból vagy oszlopokból egyértelműen és automatikusan hivatkozhatunk. Egyedi kulcsok definiálása esetén a táblázatban már meglévő kulcsú sor ismételt elhelyezésére nincs lehetőség, s a hiányzó kulcsú sorra való hivatkozás esetén is hibajelzést kapunk.

Az előbbiek szerinti ellenőrzésékkel kiszűrhetők azok az input adatok, amelyek nem tesznek eleget az előírt feltételeknek, azaz hibásak.

A szinkronizáció különösen nagy adatbázisoknál fontos, amikor egyidejűleg sok felhasználó fordulhat esetleg ugyanazon adatokhoz. Előfordulhat az is, hogy az egyik felhasználó éppen módosítani akarja, míg a másik éppen lekérdezni azokat. Ezeknek a holtpont helyzeteknek a megoldása igen bonyolult és nehéz programozói feladat, de lekezelésüket az adatbáziskezelő szoftvernek biztosítania kell.

Az adatbázis kezelésének eszközei

Az ORACLE relációs adatbáziskezelő rendszer lehetőséget teremt arra, hogy saját fejlesztő eszközeit felhasználva, úgynevezett alkalmazásokat hozzunk létre. Bár az alkalmazások fejlesztéséhez magas fokú adatbáziskezelési és programozói ismeretek szükségesek, ugyanakkor viszont a programsorok terjedelme és a fejlesztések ideje lerövidül, az elkészült szoftverek pedig a hagyományoshoz hasonlóan egyszerű módon futtathatók.

Az SQL*DBA fejlesztőrendszerrel történt a hitelesítéshez szükséges táblaterület kialakítása, a vizsgálatot végző felhasználó nevének, jelszavának, jogainak és táblaterületének definiálása.

Az SQL*PLUS fejlesztőrendszer segítségével valósult meg a vizsgálathoz szükséges táblázatok s ezek oszlopainak létrehozása, beleértve az adattípusok és az adatokra vonatkozó megszorítások definiálását is. Ugyancsak ezzel az eszközzel lehetett a táblázatok oszlopadatainak más táblázatok oszlopaira való hivatkozását előírni.

Az SQL*LOAD program az ASCII formátumban, táblázatos formában megadott adatok ORACLE adatbázisba való bevitelét teszi lehetővé a beolvasásra és ellenőrzésre kidolgozott paraméterezéssel. Ez a program a hibás, vagy a táblázatok megszorításainak nem megfelelő sorokat hibafájl-listákra teszi. A relációs adatbázisok egyik jellemzője, hogy az adattárolás független a feltöltés sorrendjétől, így a hibás adatsorok kijavítása után nem szükséges az elfogadott többi adat újra olvasása, csak a módosított adatsorok bevitelét kell megismételni.

A már adatbázisba került adatokra az SQL (Structured Query Language) és PL/SQL nyelven kifejlesztett ellenőrző és belső konzisztenciát vizsgáló programok, illetve lekérdezések szintén SQL*DBA vagy SQL*PLUS környezetben futtathatók. Szükséges megjegyezni, hogy az SQL nyelv - a nevével ellentétben - nemcsak lekérdezésre, hanem hagyományos programok fejlesztésére is eredményesen használható.

Az adatállományok ellenőrzése

A formátum ellenőrzése már a beolvasással párhuzamosan megtörténik, az adatok értelmezési tartományainak vizsgálatára pedig vagy a beolvasás során, vagy pedig utána szoftverrel kerülhet sor. Az előbbi előnye, hogy a nem megengedett értékű adatsorok nem kerülhetnek az adatbázisba, hátránya viszont, hogy a későbbiek során az ezekre hivatkozó, meglétüket feltételező táblasorok is hibásaknak minősülnek. Így a szoftverrel történő ellenőrzést látjuk célszerűbbnek a hibalisták fájlokban történő kigyűjtésével.

A belső konzisztencia keretében vizsgálandó relációk, geometriai és topológiai kritériumok a DAT1. Szabályzatban fogalmazódtak meg. Ezek a szoftverben SQL és PL/SQL nyelven megírt utasításokból, illetve eljárásokból állnak. Az SQL nyelv tömörsége, nemzetközi szabványjellege és közérthetősége miatt az algoritmusok más, bonyolultabb, nehezen felépíthető formában történő megadásától eltekinthetünk.

A belső konzisztencia vizsgálatának egyik része, hogy a különböző táblázatok különböző sorainak más táblázatok más soraira való relációs hivatkozások (szülő-gyermek öröklődési kapcsolatok) adatelemei oda és/vagy vissza irányban létezzenek (pl. egy vonal leírásához csak a már adatbázisba bevitt pontokat használhatjuk).

A belső konzisztenciák vizsgálatának másik része geometriai és topológiai jellegű. Ezek hibáit szoftverrel matematikailag megfogalmazott logikai feltételek és összetett adatbázis-lekérdezések kiértékelése útján határozzuk meg.

Az eljárások ellenőrzése tesztadatok felhasználásával valósult meg. Az adatbáziskezelő szoftver továbbfejlesztése - a gyakorlati használat során felmerülő igények szerint - folyamatosan történik.

Példa az adatbázis táblázatainak szerkezetére:

A táblázat neve: **T_PONT**

Állami alapadatokhoz és alapadatokhoz szükséges táblázat.

A táblázat adatai:

Adatmező			Megnevezés	Egyéb jellemzők
Neve	Típusa	Hossza		
pont_id	N	8	Azonosító	K,RO,R:1
pont_x	N	9	EOV x koordináta	K
pont_y	N	9	EOV y koordináta	K
pont_H	N	8	EOMA magasság	O^k
pontossági_oszt_V	N	2	Pontossági osztály vizszintes alappont vagy részletpont	K, R:2
pontossági_oszt_M	N	2	Pontossági osztály magassági alappont vagy mag. részletpont	K, R:2

Kulcs: pont_id

Értéktartomány: pont_id: 1-99 999 999, pontonként egyedi,
pont_x: 32 000.00 - 384 000.00 m,
pont_y: 384 000.00 - 960 000.00 m,
pont_H: 0.000 - 1014.000 m, vagy NULL
pontossági_oszt_V: T_KOZEPHIBA kódtáblázat szerinti kód,
pontossági_oszt_M: T_KOZEPHIBA kódtáblázat szerinti kód,

Hivatkozott táblázatok: R:1 = MEGSZ_DATUMG táblázat,
R:2 = T_KOZEPHIBA kódtáblázat,

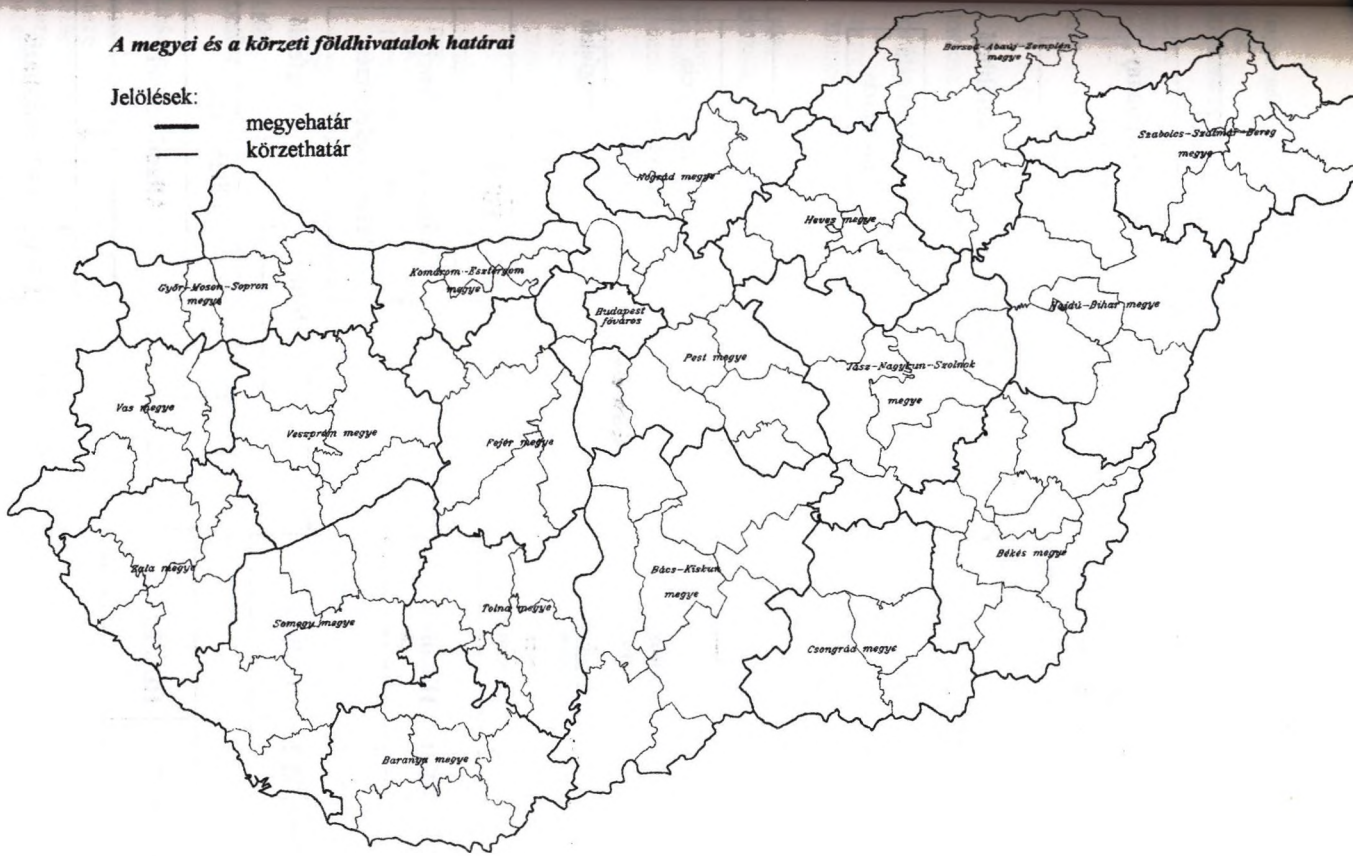
Hivatkozó táblázatok: RO = T_HATARVONAL, T_VONAL, T_IZOLALT, T_KOZB_CSPONT, T_VEG_CSPONT táblázatok, és minden T_OBJ_ATTRxx jelű táblázat.

A megyei és a körzeti földhivatalok határai

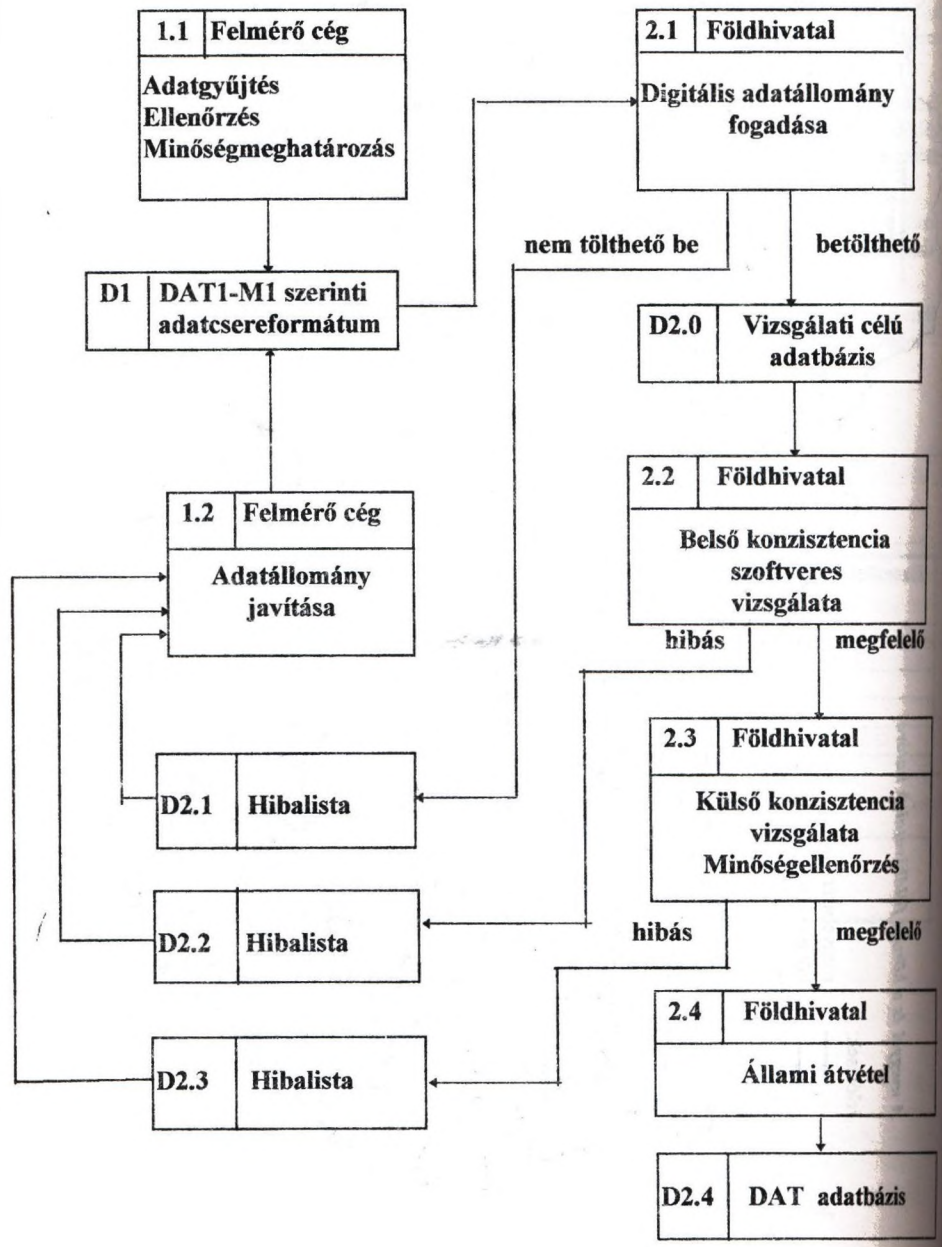
Jelölések:

megyehatár

körzethatár



DAT adatállományok átvételének folyamábrája (DFD)



A szoftver néhány részlete:

1.példa:

Az adatbázis pontokat tartalmazó táblázatának előállítása

```
CREATE TABLE T_PONT
(PONT_ID NUMBER(8)    PRIMARY KEY CONSTRAINT PONTID,
 PONT_X  NUMBER(9,2)  CHECK(PONT_X BETWEEN 32000.0 AND 384000.0)
                          CONSTRAINT PONTX,
 PONT_Y NUMBER(9,2)   CHECK(PONT_Y BETWEEN 384000.0 AND 960000.0)
                              CONSTRAINT PONTY,
 PONT_H NUMBER(8,3)   DEFAULT NULL
                      CHECK((PONT_H BETWEEN 0.0 AND 1014.0)
                           OR PONT_H IS NULL) CONSTRAINT PONTH,
 PONTOSSAGI_OSZT_V  NUMBER(2) REFERENCES T_KOZEPHIBA
                           (KOZEPHIBA_OSZT) CONSTRAINT PONTV,
 PONTOSSAGI_OSZT_M  NUMBER(2) REFERENCES T_KOZEPHIBA
                           (KOZEPHIBA_OSZT) CONSTRAINT PONTM,
 UNIQUE  (PONT_X, PONT_Y, PONT_H) CONSTRAINT PONTXYH )
 TABLESPACE DATADAT;
```

2.példa:

A pontokat tartalmazó adatállomány ellenőrzéssel történő beolvastatása ASCII formátumból

```
sqlload dat/teszt c:\datpar\t_pont.ctl    c:\dathiba\t_pont.log
sqlload dat/teszt c:\datpar\t_hatarv.ctl
c:\dathiba\t_hatarv.log

  LOAD DATA
  INFILE "c:\datadat\t_pont.dat"
  BADFILE "c:\dathiba\t_pont.bad"
  DISCARDFILE "c:\dathiba\t_pont.dsc"
  REPLACE

  INTO TABLE T_PONT
  FIELDS TERMINATED BY '*'
  TRAILING NULLCOLS
  (PONT_ID, PONT_X, PONT_Y,
   PONT_H CHAR NULLIF PONT_H=BLANKS,
   PONTOSSAGI_OSZT_V CHAR NULLIF PONT_H=BLANKS,
   PONTOSSAGI_OSZT_M CHAR NULLIF PONT_M=BLANKS)
```

3.példa:

A szoftver pontokra vonatkozó ellenőrzésének néhány SQL lekérdezése

```
DOC
Hibás pontok:
#
select pont_id, pont_x from t_pont
where pont_x not between 32000 and 384000;
select pont_id, pont_y from t_pont
where pont_y not between 384000 and 960000;
select pont_id, pont_h from t_pont
where pont_h not between 0 and 1014;
```

Az Objektum relációs adatbázis-kezelés előnyei a pénzpiaci alkalmazások területén

Sándor Gábor
S&OR Számítástechnikai Tanácsadó Bt.

Bevezetés

Napjaink nagy technológiai váltásaiban kiemelkedő szerepe lehet az Objektum relációs adatbázis-kezelő rendszerek megjelenésének. A tetszőleges komplexitású objektumok és a nagy hatékonyságú relációs adatbázis-kezelők összekapcsolásának természetes alkalmazási területei a Multimédia, Web, Intranet, Dokumentum-kezelés, 2D/3D, stb.

Ugyanakkor az Objektum relációs technológia rugalmassága, kiterjeszthetősége és nem utolsósorban hatékonysága révén, olyan hagyományosan RDBMS alapú területeken is hódít, mint a pénzpiaci, tőzsdei alkalmazások (pl. trendek elemzése, kockázat elemzés, portfolio kezelés, stb.). Ezt egy speciális kiterjesztés, az un. ***Idősorozatok*** ("Time series") modul teszi lehetővé. Az előadás erre az új technikára és az alkalmazásából fakadó előnyökre világít rá.

Az elmúlt években az Objektum relációs adatbázis-kezelés piacán az Illustra vezető szerepet vívott ki magának. A témának külön aktualitást ad az, hogy már kapható az Informix Universal Servere, amely az Illustra rendszerét, így a tetszőleges komplexitású, felhasználó által definiálható adattípusok kezelését is, kombinálja az Informix nagy hatékonyságú dinamikusan méretezhető adatbáziskezelőjével.

Objektum-relációs adatbázis-kezelés (ORDBMS)

Az Illustra ORDBMS 7 évi kutatás-fejlesztés (Postgres projekt, Berkeley egyetem) eredménye. Több mint 400 ügyfélnél helyezték üzembe. Az Illustra egy adatbázis kiszolgálóból és tetszőleges számú, DataBlade nevű osztálykönyvtárból (modulból) áll. Az adatbázis-motor a hagyományos RDBMS funkciók mellett az Illustra technológiája révén lehetőséget nyújt tetszőlegesen komplex adattípusok létrehozására és intelligens kezelésére. Egy DataBlade modul adatstruktúra gyűjteményt, a kapcsolódó adat manipulációs (szükség esetén optimalizált) függvényeket, illetve opcionálisan indexelési módszereket tartalmaz.

Az ORDBMS-ek tehát egyrészt relációs adatbázis-kezelőknek is tekinthetők, mivel támogatják az SQL használatát, másrészt objektum-orientáltak, a tetszőleges komplexitású adattípusok előállítása és kezelése szempontjából. Az ORDBMS-ek SQL felületének leírása, az SQL-3 kvázi szabványnak számít. Az SQL-3 az SQL-92 kiterjesztése felhasználó által definiált függvényekkel és operátorokkal. Ilyenek lehetnek például az adott objektumok közötti hasonlósági- illetve távolság függvények.

A megvalósított objektum-relációs tulajdonságok a következők:

- Absztrakt adattípusok használata (struktúra és viselkedés specifikálása),
- Halmazok, tömbök, absztrakt típusok, stb. tetszőleges mélységű egymásba ágyazása,
- Felhasználói függvények, operátorok definiálása adattípusokhoz (C vagy SQL szinten)
- Függvényekre is kiterjesztett költségalapú optimalizálás,
- Típus specifikus indexelési módszerek,
- Polimorfizmus; egy- és többszintű öröklődés,
- Objektumok verziókezelése ("időutazás"),
- Rule-ok, alert-ek használata.

Idősorozatok (Time Series DataBlade modul)

Az idővel kapcsolatos adatok kezelésére az élet számos területén szükség van. A pénzügyi felhasználások mellett megemlíthetők még a fejlesztési, gyártási, kereskedelmi célú alkalmazások időbeli vonatkozásai. Szintén jelentős a tudományos ill. kutatási eredmények, valamint a hírek, információk kezelése terén jelentkező igények kielégítése is .

Az Idősorozatok (TS modul) használatából fakadó előnyök

Bár a hagyományos RDBMS-ek képesek az időpontokat az adatokkal együtt tárolni, erre azonban a tábla egy-egy sorára szükség van. Nagy tömegű adatnál ez a tárolási mód hatékonytalan és a válaszidők a tábla növekedésével szintén nőnek. Ráadásul az alkalmazások fejlesztése és módosítása is körülményessé válik. Nehezíti a helyzetet, hogy az adatokhoz tartozó szemantikus információ az alkalmazásba kerül.

A TS modul használatával számos komplex adattípus (pl. naptár, idősorozat) rendelkezésre áll, amelyek az adatok tárolását és kezelését hatékonnyá teszi. A TS modul mintegy 40, az új adattípusokon értelmes függvényt és az adatok közvetlen elérését biztosító API-t is tartalmaz,

ami a fejlesztést jelentősen megkönnyíti. Igen fontos, hogy az adatokhoz tartozó szemantika szintén az adatbázisba kerülhet rule-ok, alert-ek formájában.

Eset tanulmány

Tekintsünk egy pénzügyi tanácsadó céget, amely intézményi és magánszemély befektetők részére részvény portfoliokat kezel. A cég különböző technikákat alkalmaz, hogy a tőzsdeindexnél jobb eredményt érjen el a befektetői részére. Ehhez a tevékenységhez nagy mennyiségű adatot kezelnek, amelyek tartalmazzák az egyes részvények időbeli viselkedéseit, karakterisztikáit, a részvényt kibocsátó cég forgalmi adatait, stb. Rendszeresen megkapják a magyar és külföldi tőzsdei adatokat is. Az adatbázisra alapuló elemzések révén tesz a cég javaslatot bizonyos részvények vásárlására, illetve eladására.

Tegyük fel, hogy a cég 3000 részvény adatait tárolja 20 évre visszamenőleg napi bontásban. Erre a hagyományos relációs adatbázis-kezelésnél hozzávetőlegesen 3000*20*300, azaz mintegy 18 millió sorra van szükség (az évi munkanapok számánál a tőzsdék eltérő nyitva tartásával is számoltunk). A TS modul használatával részvényenként egy (komplex adattípusokat is tartalmazó) sorra, azaz 3000 sorra van szükség.

Záró gondolatok

Az Idősorozatok (TS modul) az egyik "legnépszerűbb" DataBlade modul. Az alkalmazásával nyerhető előnyök a következők:

- Fejlesztés/Karbantartás

  A fejlesztő a feladatra koncentrálhat, a hatékony tárolás a TS modul feladata. Az adattípus módosítások lokálisan végezhetők, nincs szükség a rendszer átírására. A kezelt adatokat jól áttekinthető és hatékony struktúra szerint tároljuk.

- Performancia

  Valós méretű alkalmazásoknál mintegy 20-szoros sebességnövekedést lehetett elérni.

- Skálázhatóság

  Az adatbázis méretének növekedésével a válaszidők érdemben nem változtak.

Relációs adatbáziskezelő rendszerek alkalmazása a magyar állattenyésztési adatbázis fejlesztésekben és használata WWW -en

Dr. Herdon Miklós[1] - Kovács Zoltán[2] - Szegedi János[3]

[1]DATE, Mezőgazdaságtudományi Kar

[2]Hajdú-Bihar Megyei Területi Agrárkamara

[3]Wintech Kft.

Bevezetés

Az adatbázisfejlesztés az Információs Infrastruktúra Fejlesztési Program támogatásával 1992-ben kezdődött. A rendszer első verziója az IIF előírásait figyelembevéve elsősorban az X.25 hálózati szolgáltatásra készült. A SUN S10-es szerver és az INGRES adatbáziskezelő rendszerre alapozott fejlesztés 1993-ban készült el. Az X.25 hálózati szolgáltatás felhasználói interfészének karakterorientáltsága nem nyújthatta azt a minőségü szolgáltatást, melyet a második generáció elkészülte jelentett. A rendszer második generációjának kidolgozása az NIIF és a Földmüvelésügyi Minisztérium támogatásával készült az INTERNET WWW szolgáltatás előnyeit kihasználva. Közben platformváltásra is sor került. Az DEC AXP szerverünkön üzemelő WWW szerver mellett az adatbázis átkerült egy Windows NT SQL szerverre. A képeket, videofelvételeket, hanganyagot tartalmazó WWW szerveren müködő információs rendszerből közvetlen linkek alapján történhet a relációs adatbázis lekérdezés, melynek eredményei a WWW klienseken tekinthető meg. Az adatbázis tartalmazza a Magyarországon tenyésztett ló, szarvasmarha, sertés, juh tyúk, lúd, kacsa és pulyka fajok legfontosabb jellemzőit, a legfontosabb magyarországi törzstenyészetek és a régió egyes termelő üzemeinek paramétereit. Az egyes fajták esetén nyilvántartott információk: a fajtaleírás, létszámadatok, szaporulati mutatók, termelési paraméterek, a fajta jelenlegi helyzete, alkalmazott tenyésztési eljárások. Az adatbázis fajonként a tenyésztett fajtáktól függően 4-20 évre vonatkozó idősorokat tartalmaznak. Az adatbázis 8 faj körülbelül 250 fajta információs bázisát tartalmazza.

I. A rendszer első generációs változata

A rendszer három részből állt : *1. Adatgyüjtő és adatrögzítő rész, 2. Adatkonverziós rész, 3. Adatbázis rendszer*

Az adatbázis rendszer kifejlesztése SUN S10 Szerveren, SOLARIS 2.2 környezetben az INGRES adatbáziskezelő rendszerrel történt. A szolgáltatáshoz alapvetően a SUN solaris 2.2 rendszerkörnyezet, az INGRES 6.4 adatbáziskezelő rendszer futtató rendszere és felhasználói oldalon LAN ethernet vagy WAN X.25-ös hálózatból való eléréshez terminál emulátor program volt szükséges. A rendszer fejlesztése és tesztelése során erre a célra a KERMIT programot használtuk. A fejlesztéshez használt INGRES modulok és szoftverek a következők voltak : INGRES menü rendszeréből a Tables, Forms, INGRES interactive SQL, INGRES beágyazott SQL, Cygnus public domain C rendszer

Felhasználói interfészek

A rendszerben két lekérdezési lehetőség került beépítésre. Ezek a „Menürendszer" és az „INGRES SQL monitor"

A teljes rendszer C programozási nyelven beágyazott SQL-el (ESQLC) készült. Az animaldb felhasználói névvel történő bejelentkezés után .login eljárásfájl indította a futtatható rendszert. A bejelentkező kép után a menürendszert, vagy a funkcióbillentyűkre kötött menü sorban választhattuk az open SQL hívását.

II. A második generációs változat

A második generációs rendszerben a multimédia lehetőségeivel élve (kép, video, hanganyagok) az adatbázislekérdezés lehetősége is jelentősen változott az első generációs rendszerhez képest. Az alkalmazott szoftverek a következők voltak :Windows NT Server 4.0, Microsoft Internet Information Server 2.0, Microsoft SQL Server 6.0

A technológia leírása

Az adatok egy Microsoft SQL Serveren lévő adatbázisban kerültek tárolásra. Az adatbázisban paraméterezett tárolt eljárások segítségével valósítjuk meg az adatok lekérdezését, ezáltal elkerülhető, hogy a tábla vagy nézet szintű engedélyeket osszunk ki a felhasználóknak, azaz a felhasználó csak meghatározott tárolt eljárásokat érhet el, melyeket megfelelően

paraméterezve csak bizonyos előre meghatározott lekérdezésket végezhet el. Így biztosítható az Interneten keresztüli „anonymous" elérés esetén is a megfelelő adatvédelem. (Jelenleg az adatbáziskezelő és az Internet Information Server ugyanazon a gépen található ami csökkenti az adatvédelmi lehetőségeket.)

A felhasználó WWW felületen - CGI alkamlmazás segítségével - küldheti el kéréseit az Internet Information Servernek (IIS). Az IIS az Internet Database Connection (IDC) kiterjesztés használatával ODBC-n küldi el kérését az adatbáziskezelőnek. Az adatbáziskezelő által küldött választ az IDC egy sasblion fájl (HTX) felhasználásával alakítja át HTML formátumúvá.

Az alkalmazott megoldásban tehát a felhasználó kérését követően a WEB lapok mindig újragenerálódnak, az adatbázis aktuális állapotának megfelelően. Ez a megoldás valamelyest lassítja az információk elérését, viszont semmilyen további beavatkozásra - pl. periodikus frissítés - nincs szükség az adatbázis adatainak változása esetén.

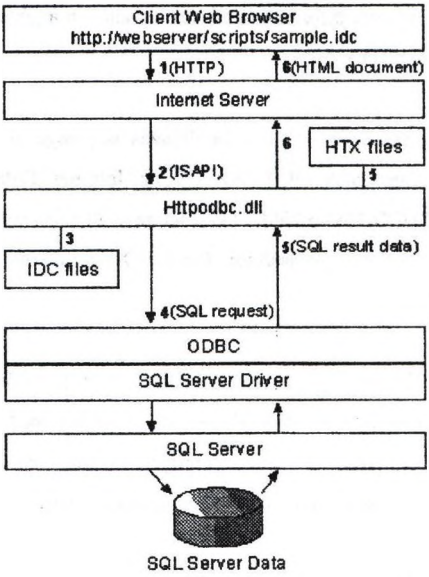
Az alkalmazott megoldás ma már nem tartozik a legkorszerűbbek közé, mivel nem tartalm sem kliens sem szerver oldali „aktív" elemeket (pl. Java vagy VB script), azonban a technológia egyszerűen továbbfejleszthető ebbe az irányba.

Az IIS 3.0 „Active Server Pages" megjelenésével lehetővé válik a technológia gyors továbbfejlesztése. Az adatbáziskezelő és Intenet Server közötti kommunikációt ebben az esteben már közvetlenül (speciális szerver kiterjesztés nélkül) OLE alkalmazásával végezhetjük el. Az adatbáziskezelővel történő közvetlen kapcsolattartásra teljesítménybeli hátránya ellenére is az ODBC-t tartjuk a legalkalmasabbnak, magas szintű támogatottsága, „ipari szabvánnyá" válása miatt.

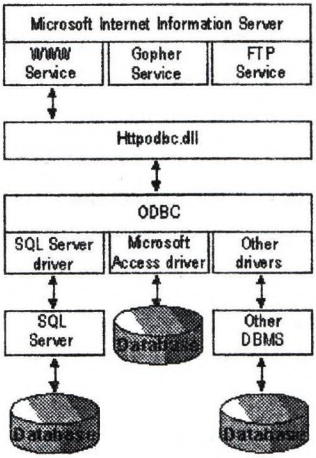
A CGI kiváltására a kliens oldali Java ill. VB srcipt alkalmazása, valamint az ActiveX technológia felhasználása tűnik járható útnak.

A következő ábrákon a megvalósitás alapjául szolgáló modell valamint a megvalósitott rendszer adatbázis lekérdezéssel kapcsolatos néhány WEB lapja látható.

A Microsoft Internet Information Server adatbázis-kezelő kiterjesztésének működési modellje



Az MS Internet Information Server kapcsolata adatbázisokkal

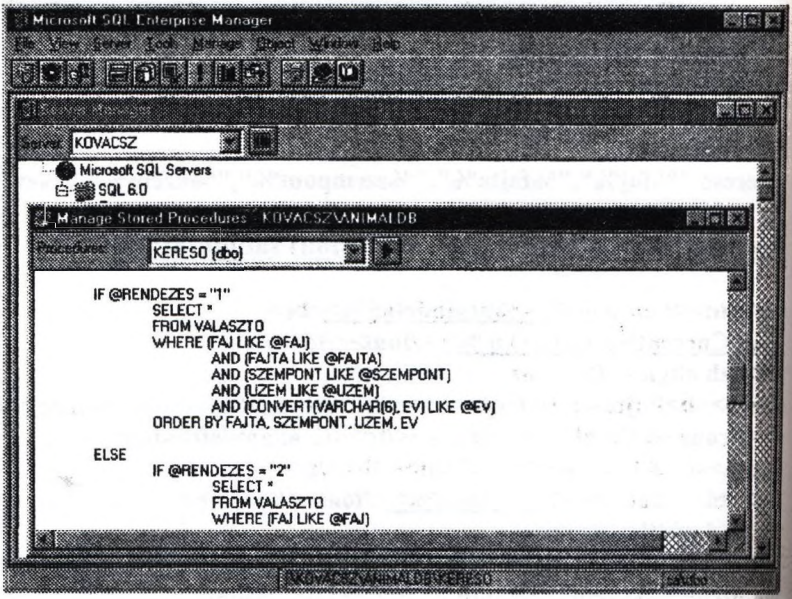


A kereso.idc (Internet Database Connector) fájl tartalma

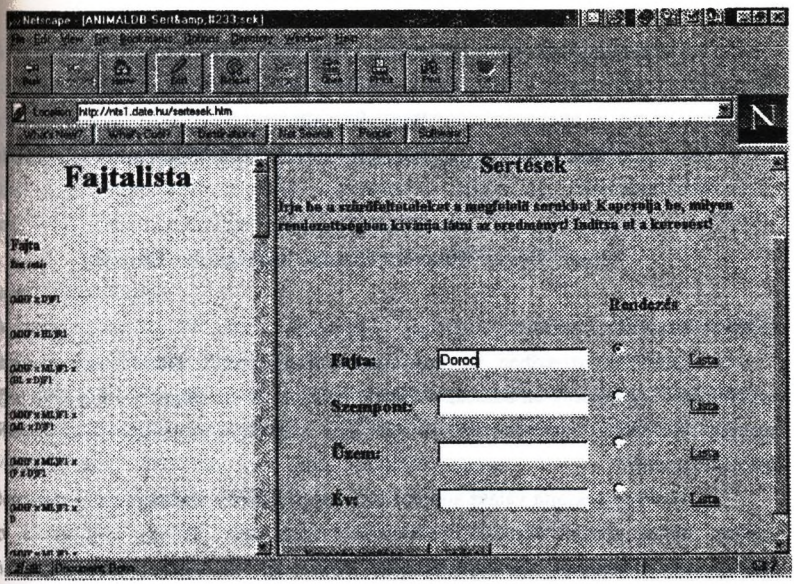
Datasource: ANIMALDB
Username: sa
Template: kereso.htx
SQLStatement:
+kereso "%faj%","%fajta%","%szempont%","%uzem%","%ev%"

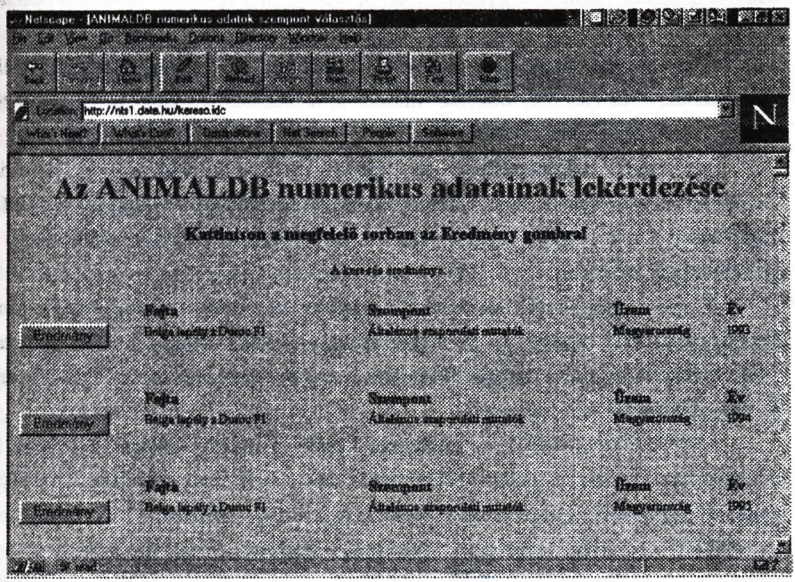
Részlet a kereso.htx (HTML extension) sablonfájlból

```
<tr><td><font size=2><%begindetail%><br>
<%if CurrentRecord EQ 0 %> </font></td></tr>
<tr><th align=left><font
size=3><b>Fajta</b></font></th><td><strong>Szempont</strong></td><t
d><strong>&Uuml;zem</strong></td><th align=left><font
size=3><b>&Eacute;v</b></font></th></tr>
<tr><td><font size=2><%endif%> </font></td></tr>
<tr><td width=80><a
href="kereso1.idc?fajta=<%fajta%>&amp;szempont=<%szempont%>&a
mp;uzem=<%uzem%>&amp;ev=<%ev%>"><font
size=2><%fajta%></font></a></td><td><font
size=2><%szempont%></font></td><td><font
size=2><%uzem%></font></td><td align=right><font
size=2><%ev%></font></td></tr>
<tr><td><font size=2><%enddetail%> </font></td></tr>
</table>
</center></div>
<p><font size=2><%if CurrentRecord EQ 0 %> <b>Nincs a
felt&eacute;teleknek megfelel&otilde; rekord.</b> <%else%> </font></p>
<hr>
<p><font size=2>Kattintson a fajt&aacute;ra az adatok
megjelen&iacute;t&eacute;s&eacute;hez.<%endif%> </font></p>
```


Microsoft SQL Enterprise Manager
Server: KOVACSZ
Microsoft SQL Servers
SQL 6.0
Manage Stored Procedures - KOVACSZ\ANIMALDB
KERESO (dbo)
IF @RENDEZES = "1"
SELECT *
FROM VALASZTO
WHERE (FAJ LIKE @FAJ)
AND (FAJTA LIKE @FAJTA)
AND (SZEMPONT LIKE @SZEMPONT)
AND (UZEM LIKE @UZEM)
AND (CONVERT(VARCHAR(6), EV) LIKE @EV)
ORDER BY FAJTA, SZEMPONT, UZEM, EV
ELSE
IF @RENDEZES = "2"
SELECT *
FROM VALASZTO
WHERE (FAJ LIKE @FAJ)


Netscape - [Magyar Állattenyésztési Adatbázis]
http://www.date.hu/agr/animaldb/index.htm
Magyar Állattenyésztési Adatbázis
Szarvasmarha - Baromfi - Ló - Juh - Sertés
Adatbázis lekérdezés - Rendszerismertető - WWW-linkek


Netscape - [ANIMALDB Sertések]
http://nts1.date.hu/sertesek.htm
Fajtalista
Sertések
Írja be a szűrőfeltételeket a megfelelő sorokba! Kapcsolja be, milyen rendezettségben kívánja látni az eredményt! Indítsa el a keresést!
Rendezés
Fajta:
Doroc
Szempont:
Üzem:
Év:
Lista


Netscape - [ANIMALDB numerikus adatok szempont választás]
http://nts1.date.hu/kereso.idc
Az ANIMALDB numerikus adatainak lekérdezése
Kattintson a megfelelő sorban az Eredmény gombra!
Eredmény
Fajta
Szempont
Üzem
Év

Tudásbázistranszformációk alkalmazása relációs adatbázisokra

Benczúr András
Eötvös Loránd Tudományegyetem, Általános Számítástudományi Tanszék

B. Novák Ágnes,
Bánki Donát Műszaki Főiskola, Informatika Tanszék

1. Bevezetés

Az utóbbi évtizedben számos cikkben foglalkoztak tudásbázisok módosításának problémájával. Nemcsak az adatbáziskezelés, hanem a mesterséges intelligencia területén is központi probléma, hogy miként módosítsuk a már tárolt ismereteket, ha új információ birtokába jutunk?

A kérdésre sokfajta válasz adható, attól függően, mi a viszonya az új információnak a régihez. Egyes szerzők konkrét operátorokat adnak meg [Dal88], míg mások bizonyos tulajdonságokat kielégítő operátorcsaládokat definiálnak Elsőként Alchourrón, Gardenfors és Makinson fogalmaztak meg 1985-ben írt cikkükben [AGM85] egy olyan axióma-rendszert, amelyet szerintük minden, a gyakorlatban jól használható elméletet módosító operátornak ki kell elégíteni.

Katzuno és Mendelzon [KM91] leszűkítve a vizsgálat tárgyát nulladrendű tudásbázisokra, az AGM-axiómákból kiindulva, a tudásbázist megváltoztató operátorok két családját adták meg, axiomatikus úton.

Az egyik axiómarendszer az eredeti AGM-posztulátumok nulladrendre való leszűkítésével keletkezett. Ezen axiómarendszert kielégítő operátorokat nevezték el revíziós operátoroknak. A revíziós operátorokat a továbbiakban (tulajdonságaik miatt) kijavító operátoroknak nevezzük.

A gyakorlati igények kielégítésére azonban ezek az operátorok nem minden esetben feleltek meg. Ezért, néhány axióma megtartásával egy újabb axiómarendszert is megadtak. Az újabb rendszert kielégítő operátorokat update operátoroknak nevezték el. E cikkben ezeket felfrissítő operátoroknak nevezzük.

[KM91]-ben a tudásbázis nulladrendű nyelven megfogalmazott jólformált formulák konjunkciója, vagyis, egyetlen nulladrendű formulával reprezentálható. A tudásbázisok halmazát jelölje *TB*. A probléma a következő: adott a tárolt tudás a φ formulával. Az új információt egy másik tudásbázis, a μ formula képviseli. Ha a tudásbázist megváltoztató operátort a $\circ$ szimbólummal jelöljük, ahol $\circ$: $TB \rightarrow TB$ függvény, mi legyen az eredménye a $\varphi \circ \mu$ transzformációnak?

A nulladrendben használatos alapvető elveket megtartva az elsőrendű tudásbázis elsőrendű nyelven megfogalmazott jólformált formulák konjunkciójaként értelmezhető.

Az alábbiakban először azt a speciális elsőrendű nyelvet adjuk meg, amellyel a tudásbázist reprezentálni szeretnénk, majd ennek segítségével értelmezzük az elsőrendű tudásbázis fogalmát (2 fejezet). A nulladrendű tudásbázisokra ismeretes axiómarendszert, amelynek segítségével a felfrissítés illetve a kijavítás eredménye megadható, a 3. részben foglalmazzuk át az elsőrendű esetre. A 4. fejezetben a függőségi rendszerek kezelésének problémakörével foglalkozunk. Végül néhány nyitott kérdést ismertetünk az utolsó részben.

2. Elsőrendű tudásbázisok

A kiszámíthatósági problémák, valamint a végtelen modellek elkerülése érdekében olyan, függvényszimbólumokat nem tartalmazó elsőrendű nyelvet adunk meg, amely a biztonságos relációkalkulusnak felel meg. A biztonságosság úgy érhető el, hogy a formulában szereplő konstansok adják a formula aktív értéktartományát , $ADOM \subseteq C$. Minden változó csak $ADOM$-ból vehet fel értéket, így új érték nem jelenhet meg.

Az elsőrendű L nyelv a következő szimbólumokból áll:

Változók:	$X := \{ x_i \mid i \in N \}$
Konstansok:	$C := \{c_i \mid i \in N \}$
Predikátumok:	$R := \{ R_i \mid i \in N \}$
Zárójelek:	(,)
Logikai összekötők:	$\wedge$; $\vee$; $\neg$
Kvantor:	$\exists$
Egyenlőség:	=

ahol N jelöli a természetes számok halmazát.

Az R_i predikátum argumentumainak számát *arg* (i) jelöli. A változókat és konstansokat együttesen *term*eknek nevezzük. Ha *arg(i)* = n és t_1, t_2 , t_3 .....t_n termek, akkor $R(t_1, t_2 , t_3 .....t_n)$ és $t_k = t_l$ *atomok.* Ha a t_1, t_2 , t_3 .....t_n termek mindegyike konstans, akkor $R(t_1, t_2 , t_3 .....t_n)$ és $t_k = t_l$ *alapatomok.* A *jólformált formulák* a szokásos módon képezhetők a $\wedge$; $\vee$; $\neg$ bázison.

Elsőrendű tudásbázison (e fejezetben a továbbiakban tudásbázison) ezen elsőrendű nyelven megfogalmazott formulák konjunkcióját értjük. A tudásbázis modelljei az alább definiált, bizonyos tulajdonságokat kielégítő adatbázisok.

Az *ab adatbázis* relációk véges halmazából áll: $ab:=\{r_1 , r_2 , r_3 , ...r_n \}$ ahol $r_i \subseteq C^{arg(i)}$ minden i-re. Az r_i reláció elemeit r_i *sorainak* nevezzük, és $<c_1, c_2, c_3,...c_{arg(i)}>$ -vel jelöljük. Az *ab* adatbázis *sémája s* $(ab) := \{R_1 , R_2 , R_3 ,...R_l\}$.

A μ *tudásbázis sémája* a μ-ben előforduló prédikátumszimbólumok halmaza, jelölése: $s(\mu)$.

A μ tudásbázis *interpretációi* mindazok az *ab* adatbázisok, amelyekre $s(\mu) \subseteq s(ab)$. Az összes interpretácó halmaza $\Im$, az összes adatbázis halmaza *AB*.

A μ tudásbázis *modelljei* a μ azon *ab* interpretációi, amelyekre a következő tulajdonságok teljesülnek:

Ha a μ tudásbázis a következő formulák valamelyike

1. $c_k = c_l$ akkor $k = l$.
2. $R_i(c_1, c_2, c_3, \ldots c_n)$ akkor $<c_1, c_2, c_3, \ldots c_n> \in r_i$
3. $\nu \wedge \varphi$, akkor *ab* modellje ν-nek is és φ-nek is.
4. $\nu \vee \varphi$, akkor *ab* vagy ν-nek vagy φ-nek modellje
5. $\neg\nu$, akkor *ab* nem modellje ν-nek.
6. $\exists x\, \nu$, akkor *ab valamilyen* $c \in C$-*re* modellje a $\nu(x|c)$ formulának, ahol $\nu(x|c)$ a c konstans helyettesítését jelenti a ν formulában x minden szabad előfordulásába.

3. Elsőrendű tudásbázisok transzformációi

3.1 Elsőrendű tudásbázisok kijavítása

A könnyebb áttekinthetőség kedvéért ebben a fejezetben a (már tárolt) φ tudásbázist a modelljeit tartalmazó adatbázishalmazzal reprezentáljuk, és tb_φ-vel jelöljük.

Tudásbázisok kijavításának egy megoldása az adatbázisok halmazán definiált totális előrendezés segítségével adható meg [BN96]. Előrendezés alatt reflexív és tranzitív relációt értünk. Ez az előrendezés lényegében azt fejezi ki, hogy az egyes adatbázisok mennyire különböznek egy adott *ab* adatbázistól, hiszen az $A \oplus B := (A \backslash B) \cup (B \backslash A)$ szimmetrikus differencia éppen az A és B halmaz azon elemeiből áll, amelyek csak az egyik halmazban szerepelnek. Ha kevéssé különböznek tőle, akkor azt mondjuk, hogy *közelebb* állnak az adott *ab* adatbázishoz. Az operátort pedig úgy definiáljuk, hogy eredményül az *ab*-hez legközelebbi adatbázisokat tartalmazza.

A tudásbázisok távolságát a következőképpen értelmezhetjük. Az azonos sémájú r_i, r_j relációk távolságát *táv* $(r_i, r_j) := |r_i \oplus r_j|$ adja meg. Az azonos sémájú ab_m, ab_n adatbázisok távolságát

$$táv\ (ab_m, ab_n) := \sum kül\ (r_i^m, r_i^n), \text{ ahol } r_i^m \in ab_m,\ r_i^n \in ab_n$$

összefüggéssel definiáljuk. A tb_φ tudásbázis és az *ab* adatbázisok távolsága:

$$távolság\ (tb_\varphi, ab) := Min_{ab_k \in tb_\varphi}\{különbség\ (ab_k, ab)\}$$

Így $ab_m \leq_\varphi ab_n$ akkor és csak akkor, ha $távolság\ (tb_\varphi, ab_m) \leq távolság\ (tb_\varphi, ab_n)$ teljesül. A definícióból közvetlenül adódik az alábbi lemma:

Lemma: $távolság\ (tb_\varphi, ab) = 0$ akkor és csak akor, ha $r_i^m = r_i^n$, ahol $r_i^m \in ab_m$, $r_i^n \in ab_n$ minden i-re.

Definiáljuk a *k* kijavító operátort a következőképpen:

Definíció:
$k: TB \times TB \rightarrow TB$, $k(tb_\varphi, tb_\mu) := Min\{tb_\mu, \leq_\varphi\}$

Az elnevezés jogosságát tétel bizonyítja, melynek kimondása előtt ismertetjük a nulladrendű operátorokra vonatkozó (K1)-(K6) axiómákat:

(K1) $\varphi \circ \mu \Rightarrow \mu$
(K2) Ha $\varphi \wedge \mu$ kielégíthető, akkor $\varphi \circ \mu \Leftrightarrow \varphi \wedge \mu$
(K3) Ha μ kielégíthető, akkor $\varphi \circ \mu$ is kielégíthető
(K4) Ha $\varphi_1 \Leftrightarrow \varphi_2$ és $\mu_1 \Leftrightarrow \mu_2$ akkor $(\varphi_1 \circ \mu_1) \Leftrightarrow (\varphi_2 \circ \mu_2)$
(K5) $(\varphi \circ \mu) \wedge \nu \Rightarrow \varphi \circ (\mu \wedge \nu)$
(K6) Ha $(\varphi \circ \mu) \wedge \nu$ kielégíthető, akkor
$\varphi \circ (\mu \wedge \nu) \Rightarrow (\varphi \circ \mu) \wedge \nu$

A (K1) axióma szerint az új ismeret visszakapható a kijavított tudásbázisból. Ez az axióma az új μ tudásnak az eredeti φ tudásbázishoz képest feltételezett "igazabb" voltát (feltétel nélküli elfogadását) fejezi ki. Amennyiben φ és μ konzisztensek, akkor az eredmény az eredeti és az új tudásbázis közös modelljeire húzódva pontosabbá válik a (K2) axióma értelmében. A (K3) axióma a kijavítás igen lényeges tulajdonságát rögzíti, éspedig ha (az eredeti tudásbázis konzisztens voltától függetlenül) az új tudást hordozó μ formula kielégíthető, az eredményül kapott tudásbázis is az lesz. Ez tehát azt is biztosítja, hogy konzisztens tudásbázissal való kijavításkor nem keletkezhet inkonzisztencia. Ez is indokolja az operátor elnevezését. A (K4) axióma Dalal szintaxis-függetlenségi alapelvét fejezi ki. Eszerint a kijavítás eredményéül adódó tudásbázis jelentése független kell hogy legyen éppúgy az eredeti tudásbázis, mint maga a kijavítás szintaxisától. A tudásbázis minimális változtatásának elvét a (K5)-(K6) axiómák azt fejezik ki.

A fenti axiómarendszert az elsőrendű értelmezésnek megfelelően átírva kapjuk az alábbi (1)-(6) axiómákat:

(1) $k(tb_\varphi, tb_\mu) \subseteq tb_\mu$
(2) Ha $tb_\varphi \cap tb_\mu \neq \emptyset$, akkor $k(tb_\varphi, tb_\mu) = tb_\varphi \cap tb_\mu$
(3) Ha $tb_\mu \neq \emptyset$, akkor $k(tb_\varphi, tb_\mu) \neq \emptyset$
(4) Ha $tb_\varphi = tb_\mu$, akkor $k(tb, tb_\varphi) = k(tb, tb_\mu)$
(5) $k(tb_\varphi, tb_\mu) \cap tb_\gamma \subseteq k(tb_\varphi, tb_\mu \cap tb_\gamma)$
(6) Ha $k(tb_\varphi, tb_\mu) \cap tb_\gamma \neq \emptyset$, akkor $k(tb_\varphi, tb_\mu \cap tb_\gamma) \subseteq k(tb_\varphi, tb_\mu) \cap tb_\gamma$

Ezen axiómák teljesülését a következő tétel biztosítja:

Tétel: A k: $TB \times TB \rightarrow TB$, $k(tb_\varphi, tb_\mu) := Min\ \{ tb_\mu , \leq_\varphi \}$ összefüggéssel megadott operátor kielégíti az (1)-(6) axiómarendszert.

A tétel bizonyítása megtalálható [BN96b]-ben. Az axiómarendszer nemcsak a fent megadott $táv(r_i, r_j) := |r_i \oplus r_j|$ távolság segítségével megadott operátorral elégíthető ki. Tovabbi példák szintén [BN96b]-ben találhatók.

3.2 Elsőrendű tudásbázisok felfrissítése

A kijavításhoz hasonlóan itt is adatbázisok halmazán definiált előrendezés segítségével adható meg a transzformáció [GMR92], itt azonban az előrendezés **parciális**.

Az ab_m adatbázis közelebb van az ab adatbázishoz, mint az ab_n adatbázis, ha:

- 1. $s(ab_m) = s(ab_n)$ és $s(ab) \subseteq s(ab_m)$
- 2. $ab_m \leqq_{ab} ab_n$ ha minden $r_i^m \in ab_m$, $r_i^n \in ab_n$, $r_i \in ab$
  - a.) $r_i^m \oplus r_i \subseteq r_i^n \oplus r_i$ minden olyan relációra, amely az ab_m, ab_n, ab adatbázisok mindegyikében előfordul
  - b.) $r_i^m \oplus \emptyset \subseteq r_i^n \oplus \emptyset$ a többi relációra.

Könnyen látható, hogy a $\leqq_{ab}$ parciális előrendezés AB -n. Legyen f az adatbázisokon értelmezett függvény, amely minden ab adatbázishoz a fentieknek megfelelő $\leq_{ab}$ előrendezést rendeli hozzá:

f: $TB \times TB \rightarrow TB$, $f(\varphi, \mu) := \cup_{ab \in tb_\varphi} Min\ \{ tb_\mu , \leqq_{ab} \}$

[GMR92]-ben a szerzők bizonyítják, hogy ezen összefüggéssel megadott f operátor kielégíti a felfrissítő operátorra megadott axiómákat, amelyek nulladrendben megfogalmazva az alábbiak:

- (F1) $\varphi \Diamond \mu \Rightarrow \mu$
- (F2) Ha $\varphi \Rightarrow \mu$, akkor $\varphi \Diamond \mu \Leftrightarrow \varphi$
- (F3) Ha φ és μ mindegyike kielégíthető, akkor $\varphi \Diamond \mu$ is az
- (F4) Ha $\varphi_1 \Leftrightarrow \varphi_2$ és $\mu_1 \Leftrightarrow \mu_2$, akkor $\varphi_1 \Diamond \mu_1 \Leftrightarrow \varphi_2 \Diamond \mu_2$
- (F5) $(\varphi \Diamond \mu) \wedge \nu \Rightarrow \varphi \Diamond (\mu \wedge \nu)$
- (F6) $\varphi \Diamond \mu_1 \Rightarrow \mu_2$, és $\varphi \Diamond \mu_2 \Rightarrow \mu_1$, akkor $\varphi \Diamond \mu_1 \Leftrightarrow \varphi \Diamond \mu_2$
- (F7) Ha $|Mod(\varphi)|=1$, akkor $(\varphi \Diamond \mu_1) \wedge (\varphi \Diamond \mu_2) \Rightarrow \varphi \Diamond (\mu_1 \vee \mu_2)$
- (F8) $(\varphi_1 \vee \varphi_2) \Diamond \mu \Leftrightarrow (\varphi_1 \Diamond \mu) \vee (\varphi_2 \Diamond \mu)$

Az (F1), (F4), (F5) axiómák rendre az (K1), (K4), (K5) axiómák megfelelői. Az (F2) axióma azonban lényegesen különbözik (K2)-től. (F2) következményeképpen ha az eredeti tudásbázis inkonzisztens, akkor a felfrissített tudás is az. Tehát a tudásbázisba valamilyen módon bekerült ellentmondás felfrissítéssel nem hozható helyre. Ez összhangban van azzal az intuitív elvárással, hogy a felfrissítés a (φ által tükrözött) világ változásait viszi be a tudásbázisba.

4. Függőségi rendszerek kezelése

A tudásbázisok nemcsak a leírandó világ tényeinek megfelelő formulakból, hanem a világ leírását pontosító, és egyben az adatbázis helyességének ellenőrzését lehetővé tevő kényszereket leíró formulákból állnak. A kényszerek kezelésére sokfajta technika ismeretes. Pl. Reiter [Rei88] a tudásbázis reprezentációtól különböző modális nyelvet javasol. [KM91]-ben a kényszerek leírása, illetve kezelése a tudásbázis reprezentációval azonos nyelven történik. Mi is ezt követjük.

A kényszerek egy része a az adatbázis tervezése során kialakítandó függőségi rendszer. A 2. pontban megadott tudásbázis fogalmat finomíthatjuk oly módon, hogy az a függőségi rendszereket könnyebben kezelhető formát öltsön. E célból az elsőrendű φ tudásbázis két formula konjunkciójaként adott: $\varphi_1 \wedge \varphi_2$, ahol φ_1 a szokásos modellkijelölő formula, φ_2 pedig a függőségi rendszert leíró formulák konjunkciója. Csak egyenlőség-(mint például a funkcionális függőség), illetve teljes sorgeneráló (mint például többértékű) függőségek legyenek megengedettek. A φ_2 formula tehát rögzített.

A probléma most az, hogy hogyan értelmezzük a $\varphi \nabla \mu$ transzformációt, ha azt is megköveteljük, hogy $\varphi \nabla \mu \Rightarrow \varphi_2$ teljesüljön?

Többfajta megoldás képzelhető el pl. aszerint, hogy a φ_2 formulát melyik tudásbázishoz akarjuk hozzácsatolni, a transzformáció alkalmazása során.

A kijavítás esetében eléggé egyszerű megoldás adódik ha $\mu \wedge \varphi_2$ konzisztens. Ekkor ugyanis a φ_2 formulát az új tudással kezelhetjük együtt, hiszen a kijavítás eredményeként kapott tudásbázisra is megköveteljük φ_2 érvényességét. Ez esetben a $\bullet$, függőségi rendszert biztosító kijavító operátor a következőképpen adható meg:

$\varphi \bullet \mu \Leftrightarrow \varphi \circ (\mu \wedge \varphi_2)$. E definícióból könnyen látható, hogy a $\bullet$ operátorral a μ új tudás szerint kijavított tudásbázis kielégíti a φ_2-ben leírt függőségi rendszert. Az új, $\bullet$ operátor szintén kijavító operátor.

Nehezebb a megoldás, ha $\mu \wedge \varphi_2$ inkonzisztens. Az alábbi módszert ez esetben is érdemes alkalmazni. A kapott * operátor azonban nem tekinthető sem kijavító, sem felfrissítő operátornak. Először a $\varphi_1 \nabla \mu$ transzformáció hajtandó végre, ahol most ∇ jelenti a kijavító, illetve a felfrissítő operátorok valamelyikét, majd az eredményül kapott tudásbázisra "alkalmazzuk" a φ_2 függőségi rendszert. Formálisan pl. a következőképpen jelölhetjük:

$$\varphi * \mu \Leftrightarrow \text{Chase}\{(\varphi_1 \nabla \mu), \varphi_2\},$$

ahol Chase (α, β) jelenti a β formula "alkalmazását" az α tudásbázisra, pl. a következőképpen (A Chase név az ugyanilyen nevű eljárásra utal, a módszer egyezése miatt):

Amennyiben β egy részformulája egyenlőséggeneráló függőséget ír le, akkor az összes egyenlőséget megsértő szimbólumot egységesitjük α -ban, akár direkt módon α modelljeinek átírásával, akár a megfelelő elsőrendű formula hozzávételével.

Ha β-ban egy részformula teljes sorgeneráló függőséget ír le, úgy a modellben a megfelelő sorokat hozzávesszük.

5. Nyitott kérdések

A kijavításnál alkalmazott $\varphi \bullet \mu \Leftrightarrow \varphi \circ (\mu \wedge \varphi_2)$ transzformációhoz hasonlóan a felfrisssítés esetében is erdemes megvizsgálni az ehhez hasonló $\varphi \bullet \mu \Leftrightarrow \varphi \nabla (\mu \wedge \varphi_2)$ transzformációt.

A 4. pontban szereplő $\varphi \bullet \mu \Leftrightarrow \varphi \nabla (\mu \wedge \varphi_2)$ típusú, illetve a $\varphi * \mu \Leftrightarrow \text{Chase}\{(\varphi_1 \nabla \mu), \varphi_2\}$, típusú operátorok viszonya is tisztázásra szorul.

Irodalom:

[AGM85] C.E. Alchourrón, P. Gardenfors, D.Makinson: On the logic of theory change: partial meet contraction and revision functions. Journal of Symbolic Logic, (50), 510 -530, 1985.

[BN94] A. Benczúr, Á. B. Novák: On Knowledgebase Change Operators. Proceedings of International Scientific Conference, Section Artificial Intelligence, Herlany, Slovakia, 1994.

[BN95] A. Benczúr, Á. B. Novák, P. Z. Révész: Klasszikus és súlyozott tudásbázisok transzformációi, megjelenés alatt: Alkalmazott Matematikai Lapok, 17, 3 - 4 szám.

[BN96] A. Benczúr, Á. B. Novák, P. Z. Révész: On Classical and Weighted Knowledgebase Transformations, Computers Math. Applic., Vol. 32, No. 5., 1996.

[BN96b] Benczúr A., B. Novák Á: Revíziós (kijavító) és update (felfrissítő) operátorok értelmezése relációs modellen alapuló tudásbázisokra. Informatika a Felsőoktatásban, 1996, Debrecen, Konferencia kiadvány.

[Dal88] M.Dalal: Updates in Propositional Databases, Technical Report DCS -TR - 222, Department of Computer Science. Rutgers University, New Brunswick,NJ 1988.

[GMR92] A. Grahne, O.Mendelzon, P. Z. Revesz: Knowledge Base Transformations, Proceedings of the Eleventh ACM-SIGACT-SIGART Symposium on Principles of Database Systems, 1992.

[KM91] H.Katsuno, A.O.Mendelzon: On the Difference between Updating a Knowledge Base and Revising it. Proceedings of the Second International Conferences on Principles of Knowledge Representation and Reasoning 1991.

[Rei88] R. Reiter: On integrity constrains, Proceedings Second Conference on Theoretical Aspects of Reasoning about Knowledge, Pacific Grove, 1988.

Sybase Interactive Warehouse

A Data Warehouse kiterjesztése

Gollnhofer Gábor
AXIS Számítástechnikai Kft.
1997.

A Data Warehousing következő generációja

Napjainkban egy teljes üzletciklus lezajlik kevesebb mint 18 hónap alatt, a verseny nemzetközivé válik és egyre élesedik, valamint a növekedési esélyek is csökkennek. Egy ilyen üzleti környezetben annak van előnye, aki a leggyorsabban tudja összeszedni és átlátni a különböző üzletrészek és termékvonalak adatait, információit, akár történeti, akár geográfiai szempontból. Ezek az információk megmutathatják, hogy hogyan reagáljunk a különböző üzleti lehetőségekre, gyártási problémákra, a versenytársak lépéseire, vagy akár politikai, gazdasági változásokra. A vállalatoknak most minden eddiginél jobban versenyképesnek kell lenniük, ezért jobban, hatékonyabban kell tudniuk kihasználni a rendelkezésre álló információkat. A data warehousing azért, hogy az üzletmenetről átfogó képet adjon, ezeket a problémákat célozza meg a különböző forrásból származó adatok rendszeres integrálásával és az interaktív analízisekhez szükséges technológiák biztosításával.

Noha a data warehousing nem új keletű koncepció, igazából csak az utóbbi években vált fontossá. Már a '80-as évek elején is kellettek elemzések az üzleti analízisekhez, de akkoriban a "Számítástechnikai Osztály"-tól kellett ezeket kérni és akár több napig is eltarthatott, mire megszületett a nyomtatott eredmény. Ezekből a listákból kiderült, hogy hogyan megy az üzlet (eladások, nyereségesség, vevők, termékek, stb.), mindazonáltal csak napok-hetek alatt lehetett egy megalapozott döntést meghozni.

Az adatok 100 százaléka a felhasználók 1 százalékának

Ahogy a technológia fejlődött, úgy újabb, hatékonyabb információgyűjtési módszerek váltak elérhetővé. A relációs adatbázisok és az új döntéstámogató (DSS) lekérdező eszközök kombinálásával a végfelhasználók már jóval gyorsabban juthattak az adatokhoz és jobban kézben tarthatták hogy mi is kerüljön a listákra. Ez azonban többnyire csak a kiváltságos kevesek privilégiuma volt, tekintettel a számítástechnikai erőforrások és a megfelelő teljesítményhez szükséges beruházások magas költségére. A 80-as évek végén, a kereskedelmi UNIX rendszerek és a kliens/szerver technológia előretörésével a marketing vagy pénzügyi osztályokon lehetővé vált olyan DSS rendszerek bevezetése melyek csökkentették a számítástechnikai költségeket, és viszonylagos függetlenséget nyújtottat az IT megkötések alól. Ahogy a lekérdező eszközök egyszerűsödtek, úgy akart mind több és több ember hozzáférni az adatokhoz. Ráadásul mindenki olyan adatokat akart látni, amelyek az ő igényeihez illeszkedtek. A hangsúly mindinkább eltolódott a tranzakció-feldolgozó rendszerektől a döntéstámogató (DSS) rendszerek felé. Ha több üzleti-elemző férhet az adatokhoz, gyorsabban, könnyebben, költség-hatékonyabban, akkor a döntéseket gyorsabban és jobb információkon alapulva lehet meghozni. Azok a vállalatok, melyek jobban kihasználják az adataikat komoly előnyre tehetnek szert a versenyben.

Az ígéretek és a realitás

A gyorsabb információ eléréssel és az adatok könnyebb kezelhetőségével az üzleti felhasználók megtudhatják, hogy kik a vevőik, miket vásárolnak és mikor, sőt akár még a versenytársakról is juthatnak némi információhoz. A nagyobb felhasználó- és üzlet-orientáltsággal együtt, a data warehousing lett az egyik új üzleti ígéret. De a szükségletekkel és a várható üzleti előnyökkel szemben figyelembe kell venni a valóságot is, azaz hogy ennek

az előnynek a megszerzése mennyibe is kerül munkában, erőforrásokban, költségekben. Sok számítástechnikai vezető szembesült már azzal a ténnyel, hogy a warehousing project nem érte el a célját, túl sokáig tartott és túl költséges volt.
Elemzők, mint például a Gartner Group, úgy becslik hogy a legtöbb data warehousing project körülbelül 3-5 millió USA dollárba kerül, hónapokig tart a fejlesztés, csak nagy költségekkel módosítható, és végül ha elkészül, gyakran nem is adja meg a kívánt információkat.

Változó követelmények

A data warehousing-nak meg kell változnia. A Meta Group becslése szerint Fortune 1000 cégeinek 70%-a számára már nem az a kérdés hogy kell-e data warehouse, hanem az hogy hogyan lehet leghatékonyabban, a legjobban kihasználható DW-t felépíteni. Amire szükség van az egy új megközelítésmód, amely a felhasználók mind jobb információkkal való ellátására koncentrál.
Az új data warehouse-nak:

- Kezelnie kell az egyre növekvő adatmennyiséget
- Ki kell szolgálnia a mind több felhasználót
- Lehetőséget kell adnia az adatokhoz való mind lazább hozzáférésre, ad-hoc lekérdezésekre
- Gyorsabban és olcsóbban megvalósíthatónak kell lennie

Az Interactive Warehouse: a Data Warehouse kiterjesztése

Hogy megfeleljen a data warehousing változó követelményeinek, a Sybase újradefiniálta a hagyományos data warehouse elképzelést, és azzal az "interactive warehouse"-al helyettesítette, amely segíti a felhasználókat az egész szervezeten belül az információk létrehozásában, integrálásában, továbbításában. Interaktív, mert arra fókuszál, hogy az információ bárhol elérhető legyen, hogy a felhasználók a saját igényeiknek megfelelő kérdéseket tehessék fel, és a saját speciális üzleti igényeiknek megfelelő válaszokat kapják. Az "Interactive warehousing" az adatok és információk körének kibővítését, új felhasználók elérését, és új kérdéstartományok kezelését jelenti.

A Sybase ezt az interakív warehouse-ot az alábbiakkal támogatja:

- A Warehouse WORKS rugalmas, bővíthető architektúrával
- Az interakív warehouse speciális lekérdezési igényeinek megfelelő szerverekkel
- Köztes elemekkel (middleware) az adat- és információforrások széles körének eléréséhez
- Az adatmozgatás és -disztribúció integrálásával és menedzselésével a vállalati warehouse környezetben
- A SAFE/DW warehousing szolgáltatásokkal, az interactive warehouse rendszerek sikeres kialakításához

Az Interactive Warehouse felépítése

A data warehouse nem egy készen kapható termék, ennek kialakítása egy folyamat. A sikeres data warehouse - az interaktív warehouse alapja egy architektúra, amely az elejétől a végéig tartó megoldás terveként használható. Ez architektúra az alapja az üzleti felhasználók használható, megbízható, pontos és aktuális adatokkal való ellátásának.
Azoknak, akik egy interaktív warehouse-ot akarnak kifejleszteni, alapvetően tisztában kell lenniük az üzleti céljaikkal, a meglévő számítástechnikai erőforrásaikkal, és ezek figyelembe vételével kell kialakítaniuk az ő egyedi data warehouse megoldásuknak a keretrendszerét. Tudniuk kell hogy:

- Mik a warehouse forrás adatai?
- Az üzleti felhasználóknak milyen adatokra van szükségük?
- Milyen eszközök kellenek, hogy a warehouse használható legyen?
- Hogyan lesznek mozgatva és terjesztve (disztributálva) az adatok?
- Hogyan lesznek tárolva az adatok a warehouse-ban?

Ezek az alapvetően megválaszolandó kérdések. Túl ezeken, figyelembe kell venni a meglévő üzleti szabályokat, modelleket, előírásokat, a termékek és vevők milyen definíciója felel meg a vállalatnak, hogyan és mikor jönnek létre az adatok, és mennyit kell a warehouse-nak tartalmaznia.
A Sybase azért indított a Warehouse WORKS programot, hogy leírja az interaktív warehouse azon architektúráját és elemeit - Sybase® adatbázisok és eszközök és első osztályú partner termékek - melyekkel a különböző vállalatok és iparágak egyedi és változó igényeinek megfelelő warehouse-okat lehet létrehozni.

Ahogy az 1. ábrán látszik, ez az architektúra és a alapelemei összekapcsolódva támogatják az interaktív warehouse-ot. A Sybase interaktív warehouse architektúrája olyan folyamatokból, termékekből és adatokból áll, melyek egy egységes, elejétől a végig tartó megoldást nyújtanak a különböző forrásokból származó adatok integrálására. Lehetővé teszi az üzleti szabályok és modellek felhasználását, transzformációs ill. konverziós logika használatát az összegyűjtött adatok könnyű felhasználhatóságához, automatizálja és menedzseli az adatok data warehouse-ba vagy data-mart-okba szétosztását, és az adatokat a szabványos lekérdező és jelentéskészítő eszközök, alkalmazások által elérhető formában tárolja.

A Warehouse elérhetőségének és alkalmazási területének bővítése

Az megfelelő data warehouse megoldás kiválasztása mindig az adott vállalat üzleti szempontjain kell hogy alapuljon, és nem az adatbázis gyártó szempontjain. Egy megoldás nem illik minden alkalmazásra. A különböző vállalatoknak különböző igényei lehetnek a nagy központi warehouse-tól a speciális data mart-ig, vagy akár a személyi adatraktárig a mobil felhasználóknak.

Egy nagy, központi vállalati warehouse lehet "az adatraktár", vagy egy többszintű data warehouse környezetben szolgálhat archiválási célokat, vagy a vállalati-szintű információk átmeneti tárakénti.

A nagy, központi vállalati warehouse-ok jellemzői:

- Rengeteg adat (több száz gigabájttól több terabájtig)
- Részletes adatok az egész vállalatról
- Szabványos jelentések és előkészített lekérdezések támogatása
- Korlátlan méretezhetőség (több adat, nagyobb feldolgozási teljesítmény)
- Központi adminisztráció

A data mart általában kisebb data warehouse, többnyire adott témakörhöz vagy üzleti egységhez kapcsolódóan. Tipikus jellemzői:

- Kisebb adatmennyiség (néhány száz gigabájt adat)
- Az adatok témakörhöz vagy üzletághoz kapcsoló szempontból nézve
- Gyors válaszok összetett ad-hoc lekérdezésekre interaktív analízisekhez
- Nagyobb IT autonómia és minimális adminisztráció

A személyes data mart, egy új elem, egy adott felhasználó igényeire szabott információkat tárolja, például eladás támogatási rendszerekben a távoli vagy mobil felhasználók számára. A személyes data mart (néha "mini mart"-ként nevezik) jellemzői:

- Kicsi, személyes adatmennyiség
- Véletlenszerű (változó) felhasználás
- Könnyű szinkronizáció a nagyobb vagy fő adatraktárral
- Ön-adminisztráció

Ahhoz hogy a warehouse-ból üzleti értéket (előnyt) kovácsoljunk létfontosságú a nagysebességű hozzáférés. A data warehouse-oknak gyors, hatékony adatszolgáltatást kell nyújtaniuk több felhasználó számára, akár tetszőleges lekérdező eszközökhöz is. Ehhez mind központi, mind lokális warehouse-okat támogatni kell, ezzel lehetővé téve a helyi igényekhez alakítást és a hálózati költségek csökkentését. Végül a warehouse-oknak többféle alkalmazást kell támogatniuk a nagy jelentésektől és adatösszegzésektől a nagysebességű, bonyolult, ad-hoc lekérdezésekig.

A mai felhasználók magas követelményeket állítanak a data warehouse elé. Egy Meta Group felmérés szerint (1993-ban és 1995-ben) a legtöbb warehouse-tól azt várják, hogy 12-18 hónap múlva akár 500 felhasználót is képes legyen kiszolgálni. Azonban a Sybase interactive warehouse bevezetéséig sok felhasználó, többféle követelménynek megfelelő támogatása nem volt megvalósítható és költséghatékony. A felhasználók számának növekedésével az adatbázisok mérete is 3-5-szörösére nőtt az elmúlt néhány év alatt.

A System 11 megoldás

A mai data warehouse-ok által megkövetelt rugalmasság és teljesítmény eléréséhez a Sybase System 11 adatbázis termékcsalád az alábbi adatkezelési megoldásokat nyújtja:

- SQL Server 11 - data warehousing és döntéstámogatás (DSS) alkalmazásokhoz -vegyes terheléshez teljesítmény optimalizáció és data warehouse-ok SMP hardveren
- Sybase MPP - terabájtok kezeléséhez - ideális központi warehouse, szinte korlátlanul méretezhető és párhuzamos, nagy sebességű adatkezelési műveletek
- Sybase IQ - az interaktív analízisekre optimalizált - nagy sebességű adatelérés, kitűnő query teljesítmény az interaktív data mart-okhoz
- SQL Anywhere - a személyi data mart-okhoz (analízisek bármikor, bárhol) - költséghatékony megoldás a személyi vagy mobil felhasználásokhoz

- OmniCONNECT - a rugalmas adatintegrációra optimalizált - transzparens adatelérés a felhasználók az adatok tényleges fizikai elhelyezkedésétől függetlenül elérhetik az adatokat

A Sybase SQL Server, az alap adatbázis technológia, a tökéletes platform az előkészített analízisekhez akár központi warehouse, akár data mart környezetben, jelentős adatbetöltési és lekérdezési teljesítménnyel, kiváló CPU és memória erőforrás-felhasználással. Az SQL Server 11 egyedülálló architektúrája mind az OLTP (tranzakció-feldolgozás) mind a DSS (döntéstámogatás) alkalmazásokat támogatja, és közel lineáris méretezhetőséget nyújt SMP hardvereken.

A Sybase MPP nagy teljesítményű, parallel feldolgozást tesz lehetővé shared-nothing architektúrában, maximalizálva a hasznos processzor teljesítményt, minimálisra csökkentve az erőforrás ütközéseket. Ez a felépítés kiváló teljesítményt és hatékonyságot nyújt. Nagy sebességű report készítés és összegzések, párhuzamos adatkarbantartás, párhuzamos adatbetöltés; ezek teszik a Sybase MPP-t a nagy vállalati szintű warehouse-ok ideális eszközévé.

A Sybase IQ a query teljesítményt növeli meg 10-100 szorosára a standard RDBMS rendszerekhez képest, azonos hardveren futva. Ez az új, korábbi korlátokat áttörő szerver termék kifejezetten a döntéstámogatás és összetett adatelemzések támogatására készült. Nagy sebessége a védett "Bit-Wise" lekérdező technológiának köszönhető:

- Oszlop-szintű adattárolás és - elérés kizárja a szükségtelen adat feldolgozásokat
- Teljesítményre hangolt indexek (bitmap, B-tree, stb.)
- Tömörített adattárolás a diszkfelhasználás és az I/O csökkentésére
- Párhozamos feldolgozás (SMP platformokon) - maximális query sebesség

A Sybase IQ okosabban dolgozza fel a komplex, interaktív lekérdezéseket, és így ideális megoldás a nagy teljesítményű, rugalmas, költség-hatékony, interaktív data mart-okhoz.

A SQL Anywhere nem csak osztály szintű, hanem akár felhasználó szintű adatelosztást is lehetővé tesz (pl. mobil felhasználók). Ez a hatékony desktop adatbáziskezelő kevesebb mint 1 MB memóriát igényel, s így ideális megoldás pl. az ügynökök hordozható PC-iben. Az SQL Anywhere-rel, a mobil felhasználóknak csak a szükséges információkat kell magukkal vinni, két irányú replikáció biztosítja hogy az információk oda-vissza áramolhassanak köztük és a központi data warehouse között.

Az OmniCONNECT-el, az összes adatforrás egységes egészként látszik, a felhasználók akár teljes írás/olvasás lehetőséget is kaphatnak, és lekérdezéseket futtathatnak a különböző adatbázisokon és rendszereken.

Metaadat használat

A sikeres data warehouse-ok másik kulcseleme a metadata - az információ a warehouse-ban tárolt adatokról. A metaadatoknak két típusa van. A relációs metaadat az adatbázist írja le (sorok, oszlopok, adattípusok); a warehouse metaadat a relációs metaadatokat használva a warehouse környezetet írja le (honnan és mikor jött az adat, hogyan volt kigyűjtve, hogyan és mikor lett létrehozva, stb.).

A metaadat kritikus mind a technikai adminisztrációhoz, mind az üzleti felhasználóknak. A metaadat ad információkat a reportokról, az adatok tárolásáról és feldogozásáról és vagy központilag tárolt vagy logikailag központosított a különböző eszköz specifikus tárolókból. A technikai fejlesztők és adminisztrátorok az adatok átalakításánál, az üzleti szabályok érvényesítésekor, és a különböző adatnézetek definiálásakor használják a metaadatokat. Az adatbázis és adatraktár adminisztrátorok a warehouse adatok definiálásakor használják (táblák, oszlopok), és az adatmozgatások töltések menedzselésénél ill. ellenőrzésekor. Az üzleti felhasználók az egyes adatelemek jelentését nézik meg a metaadatokból, hogy az adott adat honnan származik, mikor és hogyan lett kigyűjtve, stb.

Az adminisztrátorok a metaadatokat használják annak meghatározására, hogy:

- Milyen adatot tárolnak a warehouse-ban
- Melyik rendszerekből kell adatokat átvenni
- Hogyan kell transzformálni az adatokat
- Hogyan kell biztosítani az adatokat
- Hogyan kell módosítani a warehouse-t, ha az üzemi rendszer megváltozik
- Hogyan kell átvezetni az adatmodell változásokat
- Hogyan kell a metaadatokat az adatokkal szinkronban tartani
- Hogyan kell az üzleti felhasználóknak biztosítani az adatok üzleti nézetét

Az üzleti felhasználók azt szeretnék tudni, hogy:

- Milyen adat van warehouse-ban
- Honnan jön az adat
- Mit jelent az adat

- Hogyan jött létre az adat
- Milyen adatot kell használni egy adott kérdés megválaszolásához
- Hogyan lehet elérni az adatot
- Mikor jött létre az adat, és mikor került a warehouse-ba.

Ha az adatok konzisztensek az egyéb osztályok által használt adatokkal, akkor a metaadatok sokban segítik az IT osztály munkáját a meglévő rendszerek átvitelekor az új klien/szerver rendszerre. A metadatok lehetőséget adnak az új technológiák jobb integrálására is, mert a rendszerek és adatok logikai absztrakciójából származnak. A végfelhasználók számára a metaadatok megerősítik a bizalmat a kritikus üzleti analízisekhez használt adatok pontosságában.

A Sybase jelenleg fejleszti a teljes, integrált metaadat management rendszert. Ez a rendszer egy központi metaadattárat kezel majd, ahová a metaadatokat különböző CASE eszközökből lehet bevinni, mint pl. az S-Designor® vagy más harmadik gyártó eszközeiből, illetve a különböző adattranszformáló és mozgató- és lekérdező eszközökből.

A Sybase megoldás interaktív warehouse-okhoz

A Sybase Warehouse WORKS architektúrával több felhasználó férhet hozzá, több információhoz, és több kérdést tehet fel, hogy jobb, gyorsabb válaszokat kapjon a vállalata a versenyelőny megszerzéséhez és megtartásához. Az interaktív warehousing megoldással a felhasználók jobb, gyorsabban megtérülést érhetnek el a warehousing-ba befektetett erőforrásokon, és olyan méretezhető, rugalmas felépítésű rendszerhez juthatnak hozzá, amely képes lesz alkalmazkodni és integrálni az új termékeket és technológiákat, és befektetéseik jövőben is megőrizik értéküket.

Uniface Seven Független OPEN 4GL

/Wippelhauser Tamás UNISOFTWARE Rendszerház/

A Compuware bejelentése alapján ez év első harmadában várható a Uniface 4GL fejlesztő rendszer legújabb változatának a UNIFACE SEVEN-nek a megjelenése. A Compuware amerikai, széles ügyfélkörrel rendelkező korábban elsősorban nagy gépes rendszerekkel foglalkozó számítástechnikai cég. Tavalyi forgalma 500 millió dollár volt, amellyel a legnagyobb szoftver szállítók közé tartozik. A Uniface a közepes és nagy méretű kliens-szerver alkalmazások világszerte egyik legelterjedtebb fejlesztő eszköze. Elsősorban platform- és adatbázis függetlenségével, valamint széles funkció skálájával hódít. A jól tervezett és a konkurrenciához képest könnyen elsajátítható rendszer évek óta sikeresen alkalmazkodik a folyamatosan növekvő követelményekhez. Az új változatban maradt a modell vezérelt felépítés, azonban több jelentős funkcionalitással bővült a rendszer. A fejlesztőknek továbbra is a feladat pontos elvi megfogalmazására kell koncentrálniuk, a konkrét implementáció megvalósítása a Uniface feladata.

A modell a Uniface központi része. Itt lehet előírni a táblák, mezők, kulcsok, relációk globális tulajdonságait, az objektumok aktiválási pontjainak eljárásait. Más 4GL eszközökkel szemben a Uniface nem közvetlenül az adatbáziskezelőt programozza, hanem csak rutinfeladatokat helyez el az adatbázis-kezelőn belül. Az alkalmazás logikája viszont kívül, az interpretálandó alkalmazásban van leírva. A modellben globálisan leírt rutinok az egyes formokba átöröklődnek, de egyedi tartalommal felülírhatók.

A Uniface alkalmazások grafikus - Windows(16 és 32bit), Motif - és karakteres üzemben változtatás nélkül futnak. A fejlesztés során minden tulajdonság és minden eseménykezelő kód a megfelelő objektumon érhető el, megkönnyítve a kiterjedt rendszerekben való tájékozódást. A nem procedurális (kijelölés, beállítás, kiválasztás) és a procedurális eszközök használati ideje fejlesztés közben nagyjából egyensúlyban van, ami azt jelzi, hogy a rendszer maximális támogatást nyújt a részletproblémák megoldására.

A procedurális nyelv lehetőségei árnyaltak. Közel 50 féle beavatkozási ponton 100 különböző procedurális utasítás használható, és ugyanennyi függvény és állapotjelző segíti a munkát. A nyelv eltakarja előlünk az SQL-t, de a rendszer biztosít lehetőségeket a használatára. Az egyes triggerekben elhelyezett eljárások mindig a definíció eredeti helyén érhetők el, de a

globális keresés támogatására egységes forráskód is rendelkezésre áll. A rendszer mindkét irányban nyitott harmadik generációs nyelvek felé. A futtatórendszer is felhívható 3GL programból illetve 3GL rutinok is elérhetők a Uniface procedurákból.

A függetlenség több szempontból is alapeszméje a rendszernek. A platform függetlenség (VAX/VMS, szinte minden UNIX, VOS, OS/2 PM, DOS, MS-WINDOWS), hálózat függetienség (DECNET, TCP/IP, LAN Manager, Novell), és adatbázisfüggetlenség (Sybase, Oracle, Informix, Ingress, Progress, DB2, RDB, RMS, C-Isam, dBase 3 stb.) a maximális hordozhatóságot biztosítja. Mára azonban a technológia függetlenség mellett az architektúra függetlenség (több szintű kliens-szerver, WEB) is megfogalmazódott új követelményként.

A Compuware új Uniface stratégiája a 'UNIFACE in 3-D' jelmondattal foglalható össze. A 3D (development,deployment,delivery) lényege, hogy a fejlesztés és futtatás elemei mellett a szállítás és annak komponensei is nagy hangsúlyt kapnak.

A hetes változat fontosabb újdonságai a 3D struktúrában a következők :

DEVELOPMENT (fejlesztés):

- A fejlesztői környezet új formában jelenik meg. Az indító képernyő kicserélődik egy un. munkaterületre amely a fejlesztő környezet fő elemeit absztrakt formában jeleníti meg, így a fejlesztő könnyebben tud a kívánt szinten belépni. A különböző fejlesztési pontokról 'shortcut'-ok képezhetők, amelyek segítségével az aktuális fejlesztési pontok egyszerűen aktiválhatóak. A futási környezet aktiválását egy 'hotspot' segíti.

- Komponensek, Business Model-Driven Development.
  A Uniface Six-nek egy komponense volt a form, ez oldotta meg az interaktiv megjelenítést, a háttérben történő futást, a nyomtatást. A Uniface Seven éles különbséget tesz ezen komponensek közt és a form mellé bevezet két új komponenst a 'service'-t és a 'report'-ot. Ezen komponensek megjelenésével a Unifaca Seven világosan szétválasztja a megjelenítési és az applikációs logikát. A service segítségével valós üzleti szabályokat lehet egységesen leírni a model elemeinek (pl. táblák, mezők, relációk) és az ehhez kapcsolódó szabályoknak a felhasználásával. Ezt az objektumot több formból használhatjuk és tekinthetjük egy 'Business objektum'-nak, mely egy üzleti folyamat komplex és globális leírását foglalja magában. Az üzleti szabályok későbbi változását elég itt központilag változtatni és ez automatikusan érvényes lesz az egyes formokon. Ezzel az absztrakció egy magasabb fokát tudja a Uniface Seven megvalósítani, ami a tervezést, karbantartást nagymértékben segíti.

- Komponens templétek.

A modellben eddig meglevő templétek kiegészülnek egy új típussal a komponens templéttel. Ezekben a templétekben központilag lehet definiálni formok és service-k különböző fajtáit, amelyeket a fejlesztők egységesen használhatnak. Nagy alkalmazásoknál, több fejlesztő munkájának összehangolására, gyorsítására és egységes felhasználói felület kialakításánál mutatkozik meg elsősorban az előnye.

- Operations.

A komponensek aktiválására új módot vezet be a Uniface Seven az 'operation'-t. Ennek segítségével egy paramétertömb átadásával különböző módokon aktiválható egy komponens, a paraméterekben lokális változók is szerepelhetnek.

- Az új funkciók megjelenése számos új trigger, utasítás, függvény megjelenését hozta magával.

DEPLOYMENT (futtatás):

- A Uniface Seven futtató környezete továbbra is támogatja az eddigi futási környezeteket, karakteres futtató is lesz.

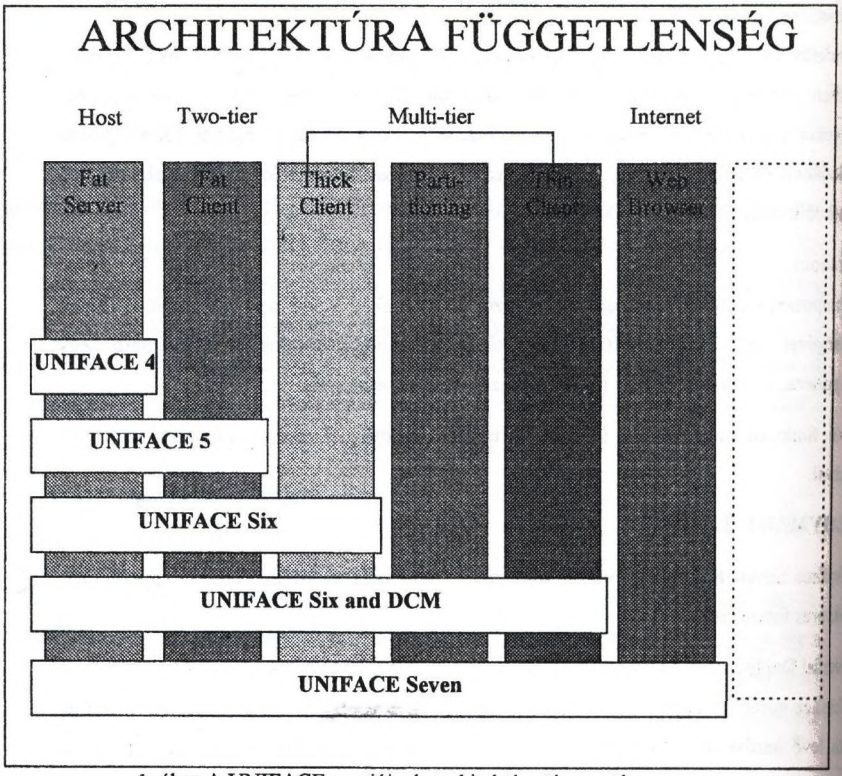
- Universal Deployment Architecture

A Uniface korábban technológia független volt. Ez azt jelenti, hogy eddig független volt az alatta levő hardvertől, operációs rendszertől, adatbázistól és hálózattól. A Uniface Seven teljesen architektúra független is, ami azt jelenti, hogy a technológia mellett az alatta levő architektúrától is független, úgy mint a host alapú, több rétegű kliens/szerver és internetes futtatás. Forráskód módosítása nélkül a futási időben dönthető el, hogy hol és milyen módon fusson egy komponens. Ezt az UDA segítségével lehet elérni. Az UDA három kulcstermékből áll:

-Uniface PolyServer az adatbázis és fájl eléréseket végzi

-Uniface ApplicationServer a service-k és reportok particionálását és futtatását végzi

-Uniface WebEnabler az alkalmazások internetes futtatását végzi



1. ábra A UNIFACE verzióinak architektúra támogatása

- Komponens alapú partícionálás.

A megjelenítési és applikációs logika éles szétválasztása, és az applikációs szerver megjelenése tette lehetővé a komponens alapú partícionálását. A felhasználói felület részei a formok, a menük mindig a kliens oldalon futnak. A service komponensek amelyek az applikációs logikát tartalmazzák futhatnak kliens oldalon, vagy szerver oldalon egy, vagy akár több szerveren is. A NameServer segítségével meghatározhatjuk, hogy alap esetben melyik komponens melyik szerveren fusson. Ezt futási időben a terheléstől függően módosíthatjuk. A reportok a másik két komponens a formok és a service-k tulajdonságainak ötvözete, mivel az adatbázis műveletek a szerver oldalon futnak le, az eredmény pedig a kliens oldalon képződik. A partícionálási lehetőség erősen csökkenti a hálózat forgalmát, és növeli a futási sebességet, mivel a nagy erőforrásokat igénylő programrészeket a nagyteljesítményű szervereken lehet futtatni. Ennek következtében kisebb teljesítményű 'thin-client'-en is futtatható az alkalmazás.

-Internet futtató rendszer.

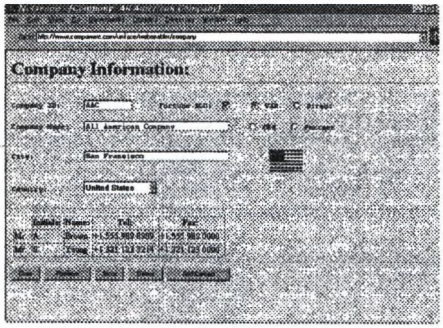
A **Uniface** Web stratégiája az architektúra függetlenségre való törekvés központi eleme. Lényege, hogy a már **meglévő**, vagy kifejlesztendő **alkalmazások változtatás nélkül futtathatók** legyenek akár kliens/szerver, akár **internet környezetben**, pusztán a futtató rendszer kiegészítésével. A koncepció szerver bázisú, azaz a WebEnabler elnevezésű futtató rendszert kell a Web szerveren telepíteni. Ezzel az alkalmazásunk az intraneten, vagy interneten elérhetővé válik a Netscape, és a Microsoft böngészők számára. A kialakított megoldás lehetővé teszi, hogy a **Uniface fejlesztőknek nem kell megtanulni és használni a HTML nyelvet** ahhoz, hogy Web alkalmazást készítsenek, a WebEnabler pedig biztonságosan elválasztja a HTTP szervertől az alkalmazás védendő adatbázisát. Ugyanazok a fejlesztési elvek érvényesek a kliens/szerver és a Web alkalmazásokban. A Uniface Seven a WebEnabler használata nélkül is képes egyes formokból különálló HTML dokumentumokat generálni.

-Non-modal formok.

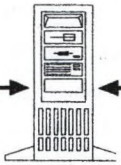
Az egyik jelentős változtatás a Uniface Seven-ben a non-modal formok bevezetése. A non-modal formok beveztésével a felhasználóknak lehetőségük lesz interaktiv módon több formot egyidejüleg kezelni. A korábbi Uniface verziókban a formok egy stack-ben helyezkedtek el, és csak az utoljára indított form volt változtatható. A non-modal formok nem egy stack-ben, hanem egy un. form pool-ban helyezkednek el és itt az adatok a különböző formokon párhuzamosan módosíthatók anélkül, hogy más formokból ki kéne lépni. Egy adott form egyidejűleg több példányban is elindítható. Az aktív ablakok szinkron/aszinkron üzenetekkel kommunikálhatnak egymással. Ugyanez az üzenő rendszer működik a helyi, vagy távoli alkalmazás különböző komponenseinek párbeszédében. A komponensek elindíthatók szinkron vagy aszinkron, attached vagy detached módban. Működik a paralell tranzakció kezelés is.

A következő ábra a UNIFACE internetes stratégiáját mutatja be. A központi futtató elemeit interneten és kliens-server felépítésben párhuzamosan érhetjük el.

WEB Browser
(' Thin ' - Client)



WEB Server



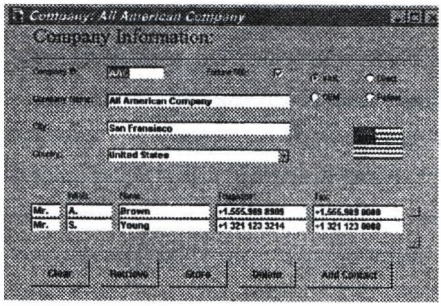
UNIFACE WebEnabler

UNIFACE Deployment

- DBMS
- Network
- TP Monitor
- Mainframe

- UNIFACE Forms
- Enterprise Data

UNIFACE Client
(' Fat ' - Client)



UNIFACE PolyServer

- OLE támogatás.

A Uniface Six esetében az OLE támogatás külső elemként beintegrálható a rendszerbe. Ez a Uniface Seven-nél alapszolgáltatásként beépül.

- A nyomtatási, riport készítési lehetőségek kibővülése. Arányos fontok nyomtatása skálázható méretben, WYSIWYG megjelenés. Ide sorolható a HTML nyomtatási kimenet lehetőségének megjelenése is.

- Erősebb Mainframe támogatás várható a Uniface Seven-nél, amely a Compuware orientáltságából következik. (MVS, DB2-IMS-VSAM)

DELIVERY (szállítás):

A Compuware a Uniface 4GL-lel nem csupán egy egyszerű fejlesztő eszközt kíván adni, hanem egy technológiát, egy feladat megoldási eljárást, amellyel komplex megbízható alkalmazások készíthetőek rövid idő alatt. Ennek elősegítését, gyors és megbízható megvalósítását kívánja elősegíteni a 3D stratégia harmadik eleme a 'delivery'. Fő elemei a következőek:

- Start-up csomag.

A Uniface Seven az új templéteket is magába foglaló példa programokat, megoldási javaslatokat ad, amelyek megkönnyítik az applikáció fejlesztés indítását, és jó minőségű előállítását.

- Standards & Guidelines.

-Dokumentáció: Standards & Guidelines kézikönyv. Ez előírásokat, ajánlásokat tartalmaz a kódolásra, az objektumok névválasztási módszerére, a felhsználói felület kialakítására, tranzakció kontrollra stb. Elsősorban több fejlesztő munkájának összehangolásánál, nagyobb alkalmazások írásánál van nagy jelentősége ezen tényezők alapos átgondolásának és következetes betartásának.

-Building Blocks: Ez olyan szoftver elemeket tartalmaz, amelyek közvetlenül beépíthetők a programokba. Ilyenek például: technikai kulcs generálás, batch ütemező, jogosultsági rendszerrel ellátott menü, applikációs példa modellek, globális objektumok.

A meglévő Uniface Six alkalmazásokat egy e célra szolgáló alkalmazás automatikusan konvertálja majd át a Uniface Seven alá.

DB2 Universal Database

Nyikes Tamás, IBM Magyarország

Mit is értünk univerzális szerver alatt?

Az utóbbi egy évben sok szó esett az univerzális adatbázisokról, univerzális szerverekről. Az egész úgy kezdődött, hogy az Informix bevezette a fogalmat a "Universal Information Server" nevű termékével. Röviddel ezután megvásárolták az Illustra Information Sytems nevű céget, mely biztosította számukra, hogy a már meghirdetett elvet meg is valósíthassák. Később az Oracle is csatlakozott azáltal, hogy az Oracle 7.3-at kinevezték az "Oracle Universal Server" relációs komponensének. Szintén ők jegyeztették be a "Universal Server" védjegyet.

"Ez idő alatt az IBM csendesen ül a marakodók felett, szállítva azt, amit a felhasználók és a szakértők a legfejlettebb 'univerzális ' adatbázisnak neveznek, a DB2 Version 2 formájában" **- Datamation, 1996 április 15, M. Halper**

"A legtöbb ember nincs tudatában annak, hogy a rengeteg bejelentés ellenére csupán egy cég, az IBM, szállít objektum kiterjesztéseket a relációs adatbázis-kezelőjéhez már fél éve, mondja Herb Edelstein az Euclid Associates nevű piackutató cégtől" **- Software Magazine, 1996 május**

Amint a fent olvasható idézetek mutatják, az IBM több mint egy éve szállítja az univerzális szerver minden lényeges elemét, magát az elnevezést leszámítva. A következőkben arról lesz szó, hogyan bővíti tovább az IBM ezeket a funkciókat a DB2 Universal Database V5 bevezetésével.

Tehát mi is az a "DB2 Universal Database"? Egyszerűen szólva egy biztonságos és kényelmes hely, ahová a fontos információinkat eltárolhatjuk. Ami miatt "universal" az az, hogy minden féle információt tárolhatunk benne. Nem csak hagyományos relációs adatokat, hanem tetszőleges strukturált, vagy nem strukturált bináris információt - dokumentumokat és szövegeket számos nyelven, grafikát, képeket, multimédiát (audiot és videót), vagy egy adott feldolgozás szempontjából fontos formátumú adatokat, mint például mérnöki rajzok, térképek, biztosítási űrlapok, vagy bármi más. Sőt, a DB2 UDB még ennél is tovább megy. Az adatbiztonságnak többféle módját nyújtja. Először is az előző verziókra épül, melyek már valódi ügyfeleknél bizonyították megbízhatóságukat. Másodszor lehetőség van a növekedésre, nem fordulhat elő, hogy további felhasználók bekapcsolásával a rendszer nem képes őket kiszolgálni. Végül bármikor lehetőség van az adatok egy korábbi állapotra történő visszaállítására, ami komoly segítséget jelent, ha valami váratlan történik a rendszerrel. Tehát nagy rendelkezésre állást biztosít akár huszonnégy órás működést is lehetővé téve. A biztonságon kívül fontos szempont a könnyű használhatóság. Ennek része a grafikus felületen keresztül történő adminisztráció, mely megkönnyíti a rendszergazda dolgát. Más IBM és nem IBM eszközökkel az ipari szabványoknak való megfelelés biztosítja a kapcsolatot. Könnyű

elérhetőséget a sokféle szerver és kliens platform támogatása és a beépített NC, illetve Internet támogatás nyújtja.

Ha tehát a DB2 UDB egy biztonságos és kényelmes adattárolási eszköz, akkor hogyan használhatjuk, és ez milyen előnyökkel jár? Először is a DB2 egyformán alkalmas üzleti intelligencia típusú és tranzakció feldolgozás típusú feladatok elvégzésére. Üzleti intelligencia alatt itt olyan technológiákat értünk mint a döntés támogatás, adatraktározás, adatbányászat és on-line analitikai feldolgozás. Tehát OLAP-tól az OLTP-ig, interaktív multimédiától a kötegelt feldolgozásig gyors és megbízható alkalmazásokat fejleszthetünk. Ez azt jelenti, hogy bármilyen felmerülő igényt kielégít egyetlen termék.

Csupán termékösszefésülés?

A DB2 UDB tulajdonképpen két nagyon sikeres termék összefésülése, funkciói a DB2 Parallel Edition V1.2 és a DB2 Common Server V2.1.2-ből tevődnek össze. Mindkét termék ugyanarról a pontról indult, ez volt a DB2/6000 Version 1, de szándékosan külön irányban párhuzamosan folyt másfél évig a fejlesztésük azért, hogy a speciális ügyfél igényeket minél hamarabb ki lehessen elégíteni. A DB2 PE elsősorban arra lett kifejlesztve, hogy nagyon nagy adatbázisokon (VLDB) lekérdezéseket futtathassunk masszív párhuzamos gépeket (MPP) felhasználva platformul. Elsősorban az IBM RS/6000 SP gépek képességit használta ki, melyre optimalizálta a lekérdezéseket. Az 1995 szeptemberi debütálása óta a Unix világ egyik legsikeresebb párhuzamos adatbázis motorjává vált. Egy nemrégiben történt felmérés alapján a legnagyobb és a harmadik legnagyobb Unix adatbázis is DB2 PE-ben van. Jelen pillanatban 1.2-es verziónál jár, és több mint ezer SP csomóponton fut szerte a világon, ami meghaladja bármely más párhuzamos adatbázis elterjedtségét az SP gépeken.

OLTP és OLAP

A DB2 Common Server célja az volt, hogy az általános SQL piac igényeinek megfeleljen mind a Unix, mind az OS/2 és a Windows NT platformon. A funkciói nagyon gazdagok beleértve az objektum-relációs képességeket is, amely az "universal" jelzőt igazolja. Mind az OLTP teljesítménye nagyon jó, mind az optimalizálónak köszönhető lekérdezési sebessége kiváló. Második verziónál jár az 1993 -as bejelentését követően. A piacon bizonyította képességeit: több mint 2.000.000 felhasználót szolgál ki több mint 200.000 szerver. A DB2 CS rendelkezik a Unix, OS/2 és NT piac legnagyobb növekedési rátájával, mely a Unix esetben például 200 %-ot is meghalad szemben a piaci átlag 25 %-kal (IDC forrás). A soha véget nem érő teljesítmény harcban folyamatosan legyőzte a versenytársakat a TPC-C eredményeket tekintve (persze mindenki tudja, hogy ezek az eredmények folyamatosan változnak, így szinte mindig más az aktuális vezető).

A két termék egyesítése nemcsak azt jelenti, hogy a PE eljut a 2-es verzió funkcionalitási szintjére, és a 2-es verzió rendelkezni fog a PE párhuzamos képességeivel, hanem számos újdonság is található az új termékben. Ilyenek a további OLAP képességek, további világháló integráció, ipari szabványok támogatásának bővítése (DCE) további integrált eszközök, mint az adatreplikáció, kormányzó és job ütemező.

Párhuzamos feldolgozás

A DB2 UDB képes különböző szintű hardverelemek támogatására a laptop gépektől egészen a masszív párhuzamos rendszerekig, melyek többszáz csomópontot tartalmazhatnak, csomópontonként akár nyolc processzorral is. Természetesen a két véglet között található SMP és SMP cluster rendszerek is támogatottak. A DB2 UDB mind a párhuzamos tranzakciókat, mind a párhuzamos lekérdezéseket lehetővé teszi. Párhuzamos tranzakció alatt azt értjük, hogy minden SQL tranzakció más processzor által van végrehajtva. Párhuzamos lekérdezés pedig azt jelenti, hogy egy adott SQL utasítást a rendszer feldarabol és az több processzoron hajtódik végre. A cluster és MPP konfiguráción a "shared-nothing" architektúrát használhatjuk ki, mely arra utal, hogy egy adott csomóponthoz tartozó adatok csak hozzá tartoznak, nincsenek másokkal megosztva. Ez jóval hatékonyabb működést eredményez ezen rendszerek esetében. A DB2 hardver támogatottsága és párhuzamos funkciói lehetővé teszik, hogy a felhasználók számát, vagy a tranzakciók számát úgy növelhessük, hogy ne csökkenjen a válaszidő. Ezt skálázható OLTP-nek hívják. Növelhetjük egy lekérdezés által használt adatmennyiséget vagy ugyanazon adatmennyiség esetében a válaszidőt csökkenthetjük. Végül az is elképzelhető, hogy a felhasználók számát szeretnénk növelni úgy, hogy a lekérdezés válaszideje ne nőjön meg. Ezeket a mutatókat lekérdezés felskálázásának, lekérdezés felgyorsításának és skálázható lekérdezésnek hívjuk. Nagyon nehéz lenne a DB2 korlátait és kapacitását túlnőni.

Nagy megbízhatóság

A bevezetőben már említettük a DB2 UDB nagy megbízhatóságát. Első ránézésre triviálisnak tűnhet, hogy egy relációs adatbázis-kezelőnek nyújtania kell ilyen szolgáltatásokat. A tapasztalat viszont azt mutatja, hogy számos felhasználó, ügyfél keres jobb adatbázis megoldást éppen a megbízhatósági problémák miatt. A DB2 ez a jobb megoldás. A DB2 UDB biztosítja a nem jogosult elérések kiszűrését, sőt ezt a hálózati és operációs rendszer szolgáltatások integrálásával valósítja meg. Ezzel lecsökken a biztonság fenntartásához szükséges adminisztratív munka mennyisége. Ami sokszor még fontosabb az az, hogy a DB2 UDB védi az adatokat az integritás sérülésből eredő károk ellen, melyek a konkurens elérésnél lépnek fel. Például nincs szükség integritás ellenőrzésre mentés előtt, a DB2 biztosítja az integritást a rendes működés folyamán. Amikor hiba lép fel, például az operációs rendszer összeomlik, vagy meghibásodik egy lemezegység, akkor a DB2 biztosítja a visszaállítást. A visszaállítási opciók rugalmasak és fokozatosak, így például egy táblát visszaállíthatunk egy régebbi időpontra amikor elkezdődött egy hibás alkalmazás futtatása ami rosszul frissítette az adatokat.

Fontos szempont az operációs rendszerben meglévő nagy rendelkezésre állási funkciók maximális kihasználása. Ilyen például a HACMP, mely biztosítja a működést azáltal, hogy a hibás gép kiesésekor automatikusan átveszi a többi gép annak feladatait. On-line segédprogramok mint a mentés, visszatöltés, újraszervezés, stb. biztosítják, hogy a tervezett rendszerkiesést minimális mértéken tarthassuk. Párhuzamos működés csökkenti az időt, amit ezek a műveletek igényelnek. Ezek a funkciók létfontosságúak, amikor a rendszerünk elér egy bizonyos méretet és a rendszerkiesési ablak nem növelhető tovább (pl. csak egy hétvége áll rendelkezésre, mely korlátos). Manapság számos rendszernél követelmény a 24x7x365-ös rendelkezésre állás, melyet a DB2 UDB biztosítani tud.

Skálázhatóság

A DB2 UDB egyfelhasználós verziója kiválóan alkalmas olyan feladatokra, mint például a mobil alkalmazások, melyek igénylik a DB2 összes funkcióját és a megváltozott adatok replikációjával kapcsolódhatnak a nagyobb rendszerekhez. Kevésbé igényes alkalmazások a Lotus Approach szoftvert használhatják a kliens adatbázis kezeléséhez, mely benne van a DB2 csomagban. A DB2 UDB kiválóan alkalmas a hagyományos alkalmazásaink újratervezéséhez, vagy áthelyezéséhez is. Ugyanazt az SQL dialektus használja mint a többi DB2 adatbázis (DB2 for MVS, VM, VSE és OS/400), így ezekről a platformokról könnyen elérhetők a benne tárolt adatok és viszont. Sőt, ezeken a platformokon tárolt adatok az IBM átjáró szoftvereivel, a DDCS és a DataJoiner-rel transzparens módon érhető el. Az is előfordulhat persze, hogy éppen egy olyan adatbázis platformról szeretnénk továbblépni, mely nem biztosított megfelelő skálázhatóságot. A DB2 UDB erre is egy megfelelő megoldás. Az IBM heterogén rendszerekhez kifejlesztett adatbázis szoftverével, a DataJoiner-rel elérhet
adatok mintha csak DB2-ben lennének, akkor is ha azok Oracle, Sybase, Informix, MS SQL Server vagy VSAM rendszerekben lettek tárolva. Elérhetőségen túl a megoldás replikációs lehetőséget is nyújt. Ezenkívül természetesen sok más migrációs eszköz szolgáltatás áll rendelkezésre az IBM-től, és annak partner cégeitől. Ha már a partner cégekről beszélünk megemlítem, hogy több mint 300 DB2 fejlesztési partner több mint 2500 alkalmazása segíti a DB2 felhasználók táborát. Az Interneten elérhető a teljes "DB2 Solutions Directory" , itt csak a termelés irányítási rendszerek közül említem meg az SAP, Baan és J.D. Edwards-ot.

Üzleti intelligencia

A bevezetésben már elmondtam mit értünk üzleti intelligencia alatt. Ezek a technológiák nagy jelentőségűek abban, hogy cégük előnyre tegyen szert versenytársaikkal szemben. A DB2 Uviversal Database egy nagyszerű platform számításigényes feladatok ellátására. A lekérdezésen belüli párhuzamos feldolgozás SMP és MPP rendszereknél egy jó példa a képességek bizonyítására. További új funkciók mint a bittérképes indexelés, csillagkapcsolás (star join) támogatás , ROLLUP és CUBE függvények a multidimenzionális analízist és OLAP technológiát támogatják. Ezeket a funkciókat a már eddig is meglévő költségalapú optimalizáló is kihasználja. Az üzleti intelligencia egyik új felhasználási területe, hogy grafikus, kliens alapú termékeket használnak melyekkel komplex lekérdezéseket intéznek az adatbázis felé. Sajnos azonban gyakran ezek a lekérdezések nem oly módon lettek megírva, mely ideális a céladatbázis szempontjából. A DB2 optimalizálónak van egy rendkívüli képessége, mellyel e lekérdezések automatikusan újraírhatók, így a teljesítmény növelhető.

További IBM termékek segítik a DB2 adatbázis erősségeinek kihasználását. Ilyenek az Intelligent Miner és az Intelligent Decision Server, melyek az adatbányászat és döntéstámogatás területéhez tartoznak. Az IBM Visual Warehouse megoldása egy adatáruház megoldás, tehát adatraktárak esetében kisebb, osztály vagy kirendeltség szintű feladatokat old meg.

A hálózati számítástechnika napjaink új, rendkívüli gyorsasággal fejlődő területe. A web-et kihasználó alkalmazások lehetnek publikus Internet vagy belső Intranet alapúak. A DB2 UDB beépített web alapja a Net.Data és Java támogatást jelenti. A Net.Data web szerverekről teszi elérhetővé a DB2 adatait, Java pedig hordozható alkalmazások fejlesztéséhez nyújt segítséget a JDBC API-kon keresztül. Java alkalmazások futtathatók az Intraneten DB2 kliensekről, vagy egy tetszőleges web böngészőről Java appletek formájában. Az utóbbi nagyon hasznos az Internetes alkalmazásoknál, hiszen nincs szükség speciális DB2 kódra a kliens oldalon. A Java környezet alkalmas arra is, hogy tárolt eljárásokat, vagy felhasználó által definiált függvényeket (UDF) írjunk. A DDCS és a DataJoiner felhasználásával a DB2 UDB-n keresztül tetszőleges adatot elérhetünk az Internetről vagy Intranetről a DB2 UDB-n keresztül.

Integrált eszközök

Az adatbázis adminisztrátor szempontjából fontos eszközök a mentési és visszatöltési segédprogramok, újraszervezési program, import/export és nagysebességű betöltő mind része az alapszoftvernek és grafikus felületen keresztül érhető el. Grafikus teljesítmény monitorral mérhetjük az adatbázis-kezelő teljesítményét és hangolhatjuk az adatbázist. Egy ún. kormányzó figyeli a felhasználók tevékenységét és meggátolja a lekérdezések "túlfutását". Job ütemező biztosítja, hogy beállíthassuk a szerverünk háttérkapacitásának jó kihasználását.

Fejlesztési szempontból fontos rész a Software Developer's Kit, mely része a szerver szoftvernek, szintén tartalmazza a DB2 Extenders kiterjesztéseket, melyekkel dokumentum, kép, audio és video alkalmazások készíthetők. Szintén az SDK része a VisualAge for Basic egy példánya mellyel kliens oldali alkalmazásokat és/vagy szerver tárolt eljárásokat, UDF-eket készíthetünk Basic nyelven. A DB2 Extenders kiterjesztések és a VisualAge Basic része a külön vásárolt SDK-nak is.

A felhasználó részéről fontos a Lotus Approach, mely szintén a szerver kód részeként kerül árusításra, s ráadásul ki lett egészítve, hogy ideális grafikus front-end legyen a DB2 UDB-hez. A DataPropagator Relational nemrég még külön megvásárolható termék volt, most ez is a csomag része, mely a replikáció mindkét oldalát biztosítja. Mindezek a kiegészítésekkel az adatbázis-kezelő a PC-s világban megszokott árszinten egy termékként, a DB2 Universal Database-ként vásárolható meg.

Entitás-Relációs modellen alapuló alkalmazások fejlesztése Zim fejlesztőeszköz segítségével

Előadó:
Ballonyi Gyula (ballonyi@irf.hu)
IRF Szoftverház Kft
1056 Molnár u. 21
Tel/Fax: 266-78-68

A bemutatásra kerülő Zim fejlesztőeszköz számos olyan egyedi megoldással és technikával rendelkezik, melyek hatékonnyá és kézben tarthatóvá teszik az alkalmazás fejlesztést.
Az alkalmazás tervezés eszköze az Architect, a fejlesztés a Zim fejlesztő-környezettel történik.

Architect

Az Architect az üzleti és informatikai elemzés és dokumentáció készítés igényeit támogatja. Automatizálja a modellezés folyamatát, eszközöket és az ezekből alkotott eszköztárral működő, kezelői környezetet biztosít a módszerek, modell objektumai és összefüggéseik kezelésére.
Egyik alkotórésze a Customizer, mely segítségével a meglévő technikák szabályai szabadon módosíthatók, illetve új technikák hozhatók létre, lehetővé téve standard (pl. SSADM) vagy saját testre szabott módszertanok kialakítását. Az egyedi igényekhez való igazítás leíró jellegű és információs modelleket foglal magába osztályozott objektumok formájában, valamint a köztük fennálló összefüggéseket. Az egyedi igényekhez igazítás meghatározza:

- a modellekben az objektumok osztályait,
- az ezen objektumok kapcsolódását meghatározó relációkat,
- modellek diagramm ábráinak típusait (pl. adatfolyam diagramm, entitás-relációs diagramm)
- a modell összetevőinek megjelenését és tartalmát meghatározó szabályokat és jelzésrendszereket,
- a mátrix modell ábrák típusait (pl. üzleti elemzés, adatelemzés),
- a mátrixokat irányító jelrendszert,
- a képernyő elrendezését és a modellábra típusait,
- az adatok integritási szabályait.

A definiált technikák és modellek alkotóelemei, az objektumok és az azokat összekapcsoló relációk egy közös adatállományban (Repository) vannak letárolva. Minden objektum egyedi és csak egyszer definiált. Az objektum egy vagy több nézeten keresztül jelenhet meg. Ez azt jelenti, hogy egy objektum bármely nézetén végrehajtott változás azonnal tükröződik valamennyi nézeten.

A Repository dokumentumokat is tartalmazhat. Egy dokumentum fejezetekre osztott tartalomjegyzékből és objektumokra való hivatkozásokból áll, azaz nem más, mint a Repository adatainak egy speciális szervezése. Segítségükkel kényelmesen lehet riportokat, dokumentációkat (pl. megvalósítási tanulmány) készíteni.
A Zim fejlesztőeszközzel export és import funkciókon keresztül tart kapcsolatot. Az import lehetővé teszi már meglévő alkalmazások adatainak beolvasását újra-tervezés vagy elemzés céljából (re-engineering), pl. megrajzoltatható annak entitás-relációs diagrammja. Az export segítségével az esetleges megtervezett képernyők és tábladefiníciók áttölthetők a fejlesztőrendszerbe. Az Architect szoros kapcsolata a Zim fejlesztőrendszerrel nem kizárólagos. Lehetőség nyílik ún. "export driver" készítésre, mely segítségével tetszőleges fejlesztőrendszer számára képes megfelelő formátumú adatokat exportálni.

Zim

A Zim termékcsalád egy 4GL fejlesztőrendszert és adatbázismotort is jelent. A legfontosabb paraméterek:

- A kifejlesztett alkalmazások számos operációs rendszerrel ill. hálózati környezetben használhatók. A felhasználó a hardver és az alapszoftver eszközeinek fejlődésével egyidejűleg az alkalmazást is hordozhatja jelentősebb változtatás nélkül.
- Az alkalmazások futtathatók mind terminál mind kliens-szerver üzemmódban, az adatbázis részei lehetnek különböző fizikai helyeken (SQL szerverek, Zim adatbázisok).
- Az adatbázismotor kétirányú B* indexelést valósít meg. Külön létezik "char" adattípus, ahol a kis- és nagybetűk különbözőek ill. az "alpha" adattípus ahol egyezőnek számítanak.
- Egy adattábla mezőiből tetszőleges definíció (!) szerint pszeudo mezők hozhatók létre. Ezek fizikailag nem tárolódnak, értéküket a hivatkozáskor áll elő. Speciálisan rendelkeznek azzal tulajdonsággal, hogy indexelhetők. Ez azt jelenti, hogy csak index adatok tárolódnak az adatbázisban (adattartalom nem).
- Speciális az entitástáblák esetén a "peruser" tulajdonság. A peruser entitásokból minden felhasználó rendelkezik egy példánnyal. Annak sorai csak az adott felhasználó számára láthatóak. Használatukkal jelentős erőforrás tartalékok szabadíthatók fel a hálózat és foglaltság kezelés területén.
- A Zim alkalmazások valamennyi adata (objetkumok definíciói) egy ún. adatszótárban vannak eltárolva. Ezt azt eredményezi, hogy az alkalmazás (egy új környezetben) reprodukálható ezen adatok duplikálásával.
- Valamennyi entitástábla és objektum (így az adatszótár is) az entitás-relációs adatmodellnek megfelelően kerül kialakításra. Az adatmanipuláció sajátos módja az ún. "Set-processing", mely SQL-re fordítható, de annál hatékonyabb módszer (lásd később).
- Az alkalmazások a Zim saját procedurális nyelvén íródnak, ezen forrásprogramok hivatkoznak entitástáblákra, képernyőkre és más objektumokra. Az alkalmazás generátorok szintén objektumokat és az

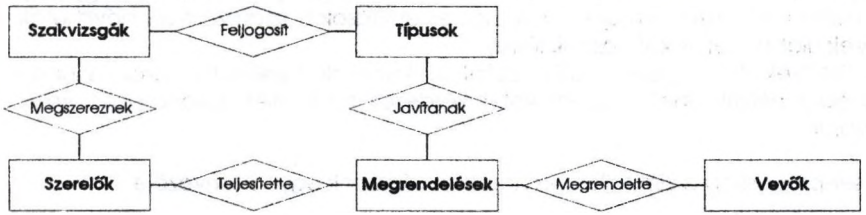
azokat használó programokat hoznak létre. Mind az objektumok, mind a programok utólagosan módosíthatók. Végrehajtásuk történhet compiler és interpreter módon is.

- Az eszköz egyaránt támogatja a modális és az esemény-vezérelt alkalmazások fejlesztését. Ez utóbbinál ún. processz szemlélet érvényesül, mely ablakokat köt össze az azokban generálódott események lekezelésére (lásd később).
- További lehetőség a forrásprogram egyszerűsítésére az állapot/nézet kezelés. Segítségével a program egyes állapotait belső állapotokká lehet nyilvánítani, és definiálni lehet közöttük átvezető engedélyezett eseményeket.
- A fejlesztőrendszer számos alkalmazás generátort bocsát rendelkezésre. Ezen generátorok testre szabhatók, de a meglévő programokból könnyűszerrel generátorok készíthetők. A generátorok készítése és alkalmazása hatékonnyá teszi a fejlesztést, de nem elrendelt út, használatuk mellőzhető.

Az Entitás-Relációs adatmodell

Az entitás-relációs adatmodell (ER modell) a relációs adatmodellnek egyfajta kiegészítése. A relációs kapcsolatok már ez utóbbi modellben is léteznek, azonban az entitás-relációs modellben ezen kapcsolatok önálló és saját névvel rendelkező objektummá válnak, a kapcsolati feltétel pusztán attribútuma az adott objektumnak. Az előbbi meghatározásból ki kell emelni az "önálló" jelzőt. A Zim-beli relációk nem pusztán nevesített feltételek, hanem olyan objektumok, melyek akár adatot is tartalmazhatnak.
A Zim-ben a reláció fogalma ki van terjesztve oly módon, hogy rajta két vagy több tetszőleges típusú objektum kapcsolatát kell érteni. Így elképzelhető egy képernyő és egy adattábla relációja is. A relációk egy speciális fajtája a reflexív reláció, melyben egy entitástábla saját magához kapcsolódik.

Az entitás-relációs modellnek a grafikus megfelelője az adatbázis egy logikai szintű - szemléletes - nézetét adják. Az entitásokat téglalapok a relációkat rombuszok jelölik (ld. alább).



Amennyiben az egyes objektumok nevei megfelelően vannak megválasztva, úgy az ábra magában elegendő információt nyújt az adatbázis szerkezetéről és az entitástáblák kapcsolatáról. Ez az adatbázis logikai képe, a Zim "set-

processing" adatmanipulációkban, a relációkra ezekkel a logikai nevekkel történik a hivatkozás.

A Set-processing

A set-processing műveletek (függetlenül a művelet típusától) minden esetben az ER modell, egy összefüggő tartományára vonatkoznak. Az alábbi példa összehasonlításképpen hozza egy lekérdezés SQL-el ill. set-processing műveletekkel való megoldását.

SQL join:

```
select distinct VEVOK.Kod, VEVOK.Nev \
from        SZERELOK, MEGRENDELESEK, VEVOK \
where       SZERELOK.Kod = MEGRENDELESEK.SzereloKod and\
            VEVOK.Kod    = MEGRENDELESEK.VevoKod and\
            SZERELOK.Nev ='Kovács' and FelvDat=vDatum
```

Zim Set-processing:

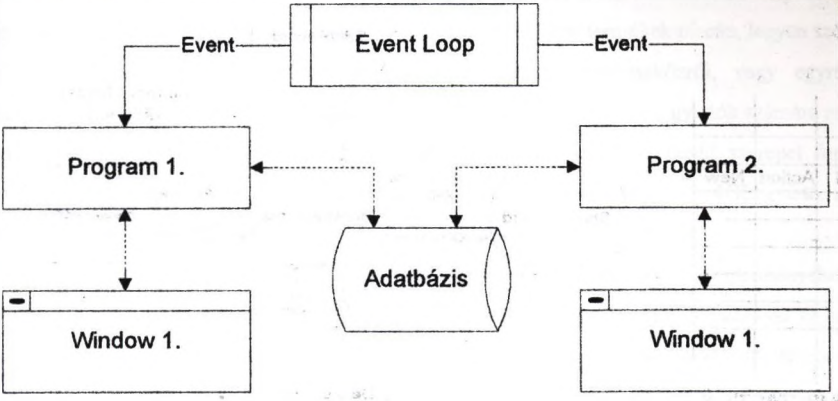
```
list all    SZERELOK Teljesitette MEGRENDELESEK Megrendelte VEVOK\
where       SZERELOK.Nev='Kovács' and FelvDat=vDatum \
keep        VEVOK
```

A megjelölt részek mutatják, hogy az SQL join mely részei helyett szerepelnek a set-processing utasításban relációk. Előnye nem pusztán a forrásprogram áttekinthetőségében jelentkezik, hanem emellett lehetővé teszi hogy az ER modellnek megfelelő logikai nézettel lehessen dolgozni.
A Zim-ben léteznek dinamikusan létrejövő objektumok ún. "set"-ek. Ezek az egyes adatmanipulációs műveletek eredményeként leválogatott rekordokat jelentik. A "set"-ek szintén önálló logikai névvel rendelkeznek. Nem adattartalmak, hanem referenciák (pointerek) kerülnek leválogatásra. A leválogatás eredménye nem csak egyetlen tábla bizonyos rekordjait jelentheti, hanem több tábla megfelelő feltételek alapján kapcsolódó rekord n-eseit (sorrendezett adatbázis-pointer vektor). Ezek a "set"-ek a későbbiekben tetszőleges célra felhasználhatók (pl. riport készítés) vagy egy további lekérdezésben újabb relációk és entitások hozzáfűzésével bővíthetők (ezek újabb "set"-eket hoznak létre).
A "set"-ek ténylegesen halmazként is képesek viselkedni, azaz minden további nélkül lehet képezni két lekérdezés metszetét, különbségét vagy unióját.

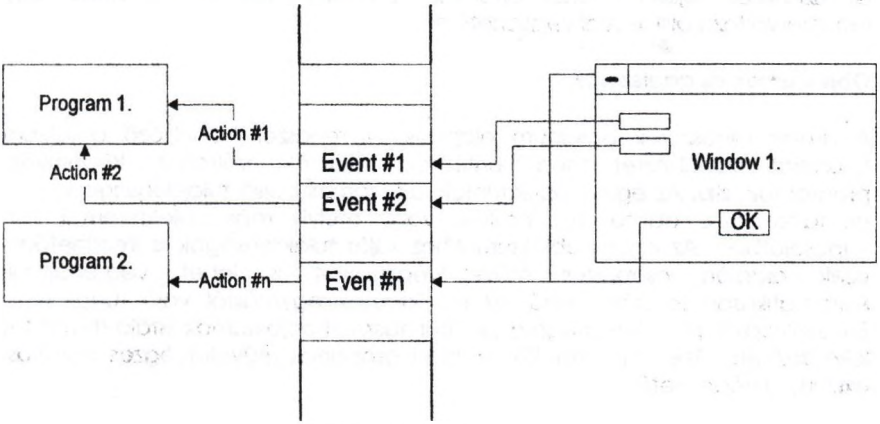
A set-processing a Zim fejlesztés hatékonyságának egy fő tényezője.

Esemény-vezérelt modell

Az esemény vezérelt Zim alkalmazások a processz szemléleten alapulnak. A felhasználónak lehetősége van egyszerre több ablakkal is kommunikálni. A rendszer nyugalmi állapotban egy úgynevezett várakozó hurokban várakozik *(event-loop)* mindaddig, amíg a felhasználó a képernyőn valamilyen beavatkozást nem végez (billentyű lenyomása, egér gomb kattintás). Erről az eseményről értesül a process-manager, mely eldönti, hogy mi a tennivaló, ill. melyik programot kell értesíteni az esemény bekövetkezéséről.



Az esemény lekezelés módja lehetőséget add arra, hogy egyes események "callback" függvényeit adatbázisban tároljuk. Egy adott ablak adott eseményére rögzíthető, hogy mely funkció hívandó meg. Ez dinamikus programstruktúra kialakítására ad lehetőséget.
Amennyiben a bekövetkezett eseményhez nincs adatbázisban tárolt funkció, úgy azt a process manager átadja a megfelelő programnak lekezelésre.



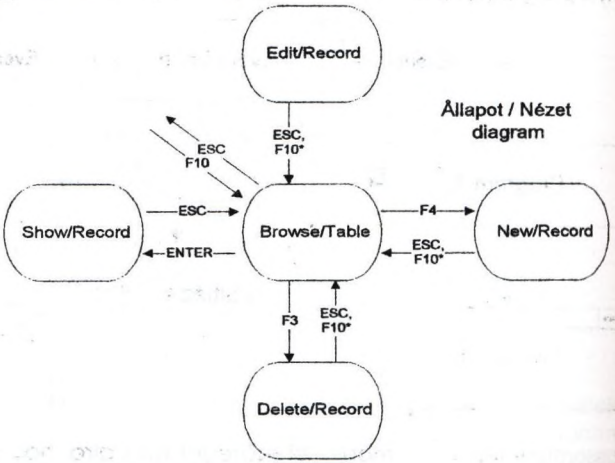
EVENTACTION

Állapotgép

A Zim lehetőséget ad arra, hogy a program egyes állapotait belső állapotokká nyilvánítsuk és ezen állapotok között az átmeneteket bizonyos eseményekre korlátozzuk. Ez módot ad arra, hogy a programot mint egy "véges automatát" lehessen kezelni. Az állapotátmenetek szabályai adatbázisban vannak letárolva. Ez egy összerendelő táblázat, mely megmutatja hogy az egyes belső állapotokból mely események mely belső állapotokba visznek át.

Old	Action	New

OBJECTACTION



Többnyelvű applikáció

Létezik két adatállomány, ahol az applikáció megfelelő nyelvű üzeneteit és az üzenetek fejléceit lehet eltárolni. A nyelvezetek közötti váltás egy rendszerváltozó állításával végezhető el.

Objektumok és adatszótár

A Zim-es rendszerek objektum alapúak. A rendszer különböző objektum típusokat különböztet meg: entitások, relációk, változók, képernyők, programok, stb. Az egyes objektumoknak számos belső tulajdonsága is van, pl. tartalmaz-e adatot módosult-e, vagy melyik más objektummal van kapcsolatban. Az egyes objektumokhoz saját tulajdonságok is illeszthetőek, ezek alapján csoportos műveletvégzéseket is lehet végrehajtani. Automatikusan tárolásra kerül az objektumok egymástól való függése is. Ennek alapján pl. adott programok által használt objektumok listája (hívási fa) lekérdezhető. Ezen információk alapján csoportos műveletvégzés (fordítás, export,...) végezhető.

WebSpeed: biztonságos PROGRESS alkalmazások az Interneten

Oláh András - ONLINE Kft.

A számítástechnika egyik legdinamikusabban növekvő piaca az Internethez kapcsoló termékek piaca. Ez az állítás különösen igaz az internetes szoftvertermékek piacán, legyen szó szörfözést biztosító alapszoftverről, WWW szerverről, fejlesztőeszközről, vagy egyre jellemzőbben Interneten működő alkalmazásokról. A "hagyományos" 4GL gyártók számára az Internet egy új platformot jelent. Természetesen ez a platform kiemelkedő szerepet fog betölteni a jövőben, ezért érdemes megvizsgálni, hogy egy " hagyományos" 4GL gyártó, a PROGRESS Software milyen módon támogatja az új platformot, az Internetet.

Előadásomban először a 4GL eszközök megjelenését és elterjedését kiváltó követelményeket elemzem, érintve a főbb fejlődési állomásokat. Ezt követően az Internet alkalmazásának és az internetes alkalmazások fejlődési szakaszai kerülnek sorra. Végül - technikai ismertetéssel együtt - a PROGRESS Software WebSpeed termékének példájára támaszkodva, próbálok választ adni arra a kérdésre, hogy milyen többletet nyújt egy 4GL eszköz alkalmazása az Interneten.

UNIX-os kezdetek

A 4GL-ek és a relációs adatbáziskezelők születése egybeesett a UNIX alapú rendszerek elterjedésével. A UNIX egyrészt robosztus alapot nyújtott nagy mennyiségű adat hatékony kezelésére, másrészt gyors terjeszkedését nem tudta követni a UNIX lehetőségeit kihasználó alkalmazások megjelenése. Az abban az időben rendelkezésre álló 3. generációs eszközökkel (C, PASCAL, stb.) megvalósuló fejlesztések nem biztosították azt a hatékonyságot, amit a bővülő piac megkövetelt. Ennek megfelelően a UNIX-os platform relatív alkalmazás hiánya új, hatékony fejlesztőeszközök számára nyitott utat. A megjelenő 4GL eszközök különböző formában és hangsúlyokkal automatizálták a szoftverműködés alsó szintjét, elfedték a technológiai infrastruktúrát, így biztosítva a fejlesztők számára a fizikai programozástól a magasabb szintű logikai programozás irányába történő elmozdulást.

Példánknál maradva a PROGRESS 4GL első verziói az alábbi versenyelőnyöket biztosították használóinak:

- RDBMS és Adatkönyvtár -> logikai adatkezelés a fizikai interfész ismerete nélkül,
- Tranzakciókezelés -> a biztonságos alkalmazásműködés alapja,
- Magas szintű utasítások és default működés -> fejlesztési hatékonyság növelése,
- Platformfüggetlenség -> a fizikai technológia elfedése,
- Végfelhasználói lekérdező eszközök -> a karbantartási igény csökkentése.

Természetesen verzióról verzióra a mai napig bővül a fenti lista, de a lényeg változatlan: hatékony, platformfüggetlen fejlesztés biztosítása. Az egyes 4GL gyártók ezen túl más és más tulajdonságok megvalósításával pozícionálták saját terméküket. A PROGRESS számára elsősorban a biztonságos "mission critical" működés megvalósítása jelentette azt a többletet, ami piaci sikerének alapjává vált. Ennek köszönhető, hogy a PROGRESS 2300 szoftver partnere ma már évente 1,5 milliárd USD értékű alkalmazást értékesít a világpiacon.

MS-Windows, a nagy kihívás

A 90-es évek elejétől a kliens-szerver technológia terjedése vált a szoftverpiac alapvető mozgatórugójává. Bár elsősorban a kliens-szerver vált "kulcs" kifejezéssé, de a kliens-szerver terjedését kísérő grafikus alkalmazások iránti igény szintén döntő változásokat hozott a 4GL piacon. Érezhetőbbé vált a szegmentálódás a szerver (RDBMS), illetve fejlesztő (4GL) központú fejlesztőeszközök között. Önmagában a kliens-szerver technológia biztosítása és tökéletesítése nem jelentett nagy megrázkódtatást (a PROGRESS már a 90-es évek előtti verzióiban biztosította a kliens-szerver működést), a "nagy ugrást" a UNIX hostok karakteres világából a Windows világába történő átlépés jelentette. Ennek megfelelően az RDBMS központú eszközök a (kliens-)szerver technológiára koncentrálva viszonylag egyenletesen tudtak lépést tartani a fejlődéssel. A 4GL központú termékeknek (a PROGRESS ide sorolható) végre kellett hajtani a nagy ugrást a többablakos világba, mindezt az újonnan megjelenő "MS-Windows only" fejlesztőeszközökkel versenyben. A kihívásra adott válasz alapvető eleme továbbra is a technológiai infrastruktúra elfedése. A PROGRESS olyan fejlesztőkörnyezettel jelent meg a piacon, ami egységes kódot biztosít karakter és Windows környezetben. Ezzel egyrészt jelentősen megkönnyítette az átmenetet VAR partnerei számára, másrészt versenyelőnyt biztosított magának a komplex, nemzetközi alkalmazások piacán.

Vissza a jövőbe, avagy az Internet mint új platform megjelenése

A kliens-szerver és a GUI programozás folyamatos térnyerése mellett a mai számítástechnikában az Internet vált a "legforróbb" területté. Meggyőződésem szerint a 4GL

eszközök számára az Internet alapvetően egy új platformként értékelendő, ahol ezek a termékek ugyanazokat a versenyelőnyöket biztosítják, mint tették eddigi történetük során. Az RDBMS alkalmazások szempontjából vizsgálva az új platformot - az új jellegzetességek mellett- két "hagyományos" tulajdonságot kell kiemelnünk:

- Grafikus felhasználói felület
- Szerver központú működés

A grafikus felhasználói felületet nem szükséges magyarázni, a Web elterjedése óta kizárólag ilyen alkalmazások születnek az Interneten. A szerver központú működés szintén a Web technológiának köszönhető és azt jelenti, hogy az alkalmazás minden része (az adatok és a felhasználói interakciók kezelése, a teljes programlogika) egy (vagy több) szerveren fut. A Web alkalmazások gyors terjedésének egyik záloga pontosan az a tulajdonság, hogy a felhasználó egyetlen szoftvert (browser) vásárol, és ettől kezdve bármilyen adatbázisban, bármilyen eszközzel fejlesztett alkalmazást futtathat otthoni gépéről.

Érdekes módon a fenti két tulajdonság, külön-külön már megvalósításra került a "hagyományos" 4GL-ek történetében, ahogy az előzőekben érintettük. Kivételt jelentenek azok az eszközök, amelyek a kliens-szerver technológiával elsősorban MS-Windows platformra koncentrálva jelentek meg a piacon. Számukra az Interneten történő megjelenés jelentős erőfeszítéseket kíván. Mielőtt azonban részletesebben tárgyalnám, hogy milyen változásokat hoz egy 4GL termék működésében és használatában az Internet architektúrája, illetve maga a 4GL technológia mit nyújt az internetes fejlesztők számára, érdemes áttekinteni az Internet és elsősorban a Web technológia terjedésének újkori fázisait.

Statikus Web oldalak megjelenése

Az Internet kereskedelmi felhasználásában, az első időszakban az Internet reklámhordozó médiumként történő használata dominált. A Web-et használó cégek publikálják termékeiket, szolgáltatásaikat, gazdasági adataikat egyszóval ugyanazokat az információkat, amelyeket a hagyományos médiumokban is közölnek. A Web oldalak megtervezett, statikus szöveges és képi információkat tartalmaznak, kezelésük a PR vezető kezében van. A Web használatának legfontosabb jellemzői ebben az esetben:

- kommunikációs eszköz,
- statikus tartalom,
- minimális költségek,

- "laza" technikai követelmények.

Interaktív Web oldalak megjelenése

A statikus Web oldalak után hamarosan megjelentek az első interaktív oldalak is. Ezek egyrészt az addig statikus Web helyek kiegészítéseként felhasználói visszacsatolást tettek lehetővé, másrészt új üzleti szolgáltatások beindítását biztosították. Ma már az interaktív Web helyek széles választéka üzemel az Interneten, a virtuális CD üzletektől a használt autók adás-vételéig. Ezek a Web oldalak már eszközöket nyújtanak a Web szerveren elhelyezkedő adatok lekérdezésére, adatok felvitelére egyszerű programlogika alkalmazásával. Az interaktív Web legfontosabb jellemzői :

- üzleti eszköz,
- adatbázis hozzáférés,
- "érzékelhető" beruházási igény,
- technikai követelmények teljesítmény oldalról.

Tranzakció központú alkalmazások az Interneten

Az interaktív Web alkalmazásokban rejlő lehetőségek felismerésével, a vállalatok egyre nagyobb része ismeri fel azt a versenyelőnyt, amit az Internet használata biztosít. Ennek megfelelően új igényként jelentkezik adatbázis és tranzakció-központú alkalmazások publikálása az új technológia segítségével. Tulajdonképpen a vállalati információs rendszerek "publikálása" kezdődik meg a partnerek és a vásárlók számára, lehetőséget nyújtva az önkiszolgáló attitűd kialakítására. Az ügyfél komplex tranzakciókat képes lebonyolítani, miközben állandó hozzáférési joga van a raktárkészlet adataitól kezdve, a szállítási információkon keresztül, a számla adatok ellenőrzéséig. Ezek a Web alkalmazások már komplex üzleti logika és adatbázis elérés hatékony megvalósítását igénylik, tehát fejlesztői és működtetői oldalról ugyanazok a kihívások jelentkeznek, amelyek a 4GL eszközök megjelenését hívták életre. Különösen egyértelmű az Internet alapú 4GL iránti igény ott, ahol a publikálni kívánt vállalati információs rendszer 4GL segítségével készült. Az elmondottaknak megfelelően a tranzakció központú Web alkalmazások legfontosabb jellemzői a következők:

- versenyelőnyt biztosító üzleti alkalmazás
- adatbázis tranzakciók

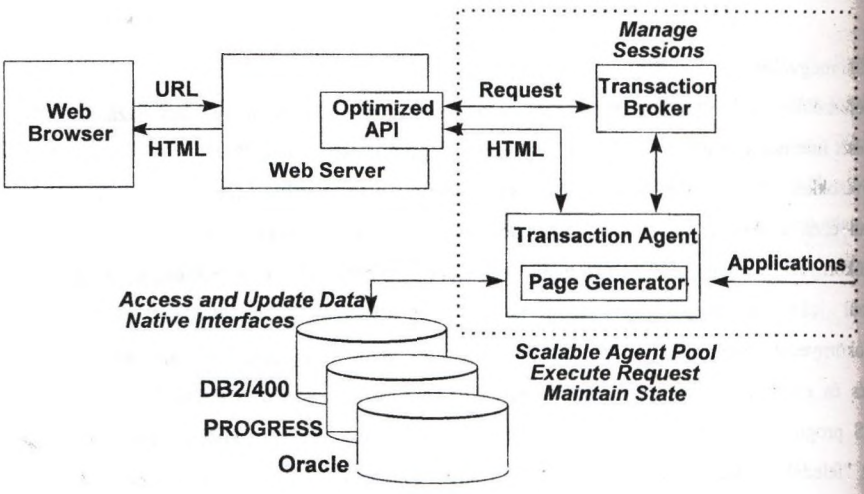
- erőteljes beruházási igény
- dinamikus skálázhatóság

Technikai megvalósítás

A következőkben technikai oldalról ismertetem a 4GL és RDBMS alapú alkalmazások működését Internet környezetben. Ennek segítségével pontosabban megérthető, hogy milyen új kihívásokkal kell szembenéznie a 4GL eszközöknek illetve milyen előnyöket hordoz magában ezen eszközök alkalmazása az Interneten. Példánk továbbra is a PROGRESS fejlesztőkörnyezet, amelynek Internet alapú változata a WebSpeed. A WebSpeed-et nem véletlenül jelentette meg különálló termékként a PROGRESS Software. Bár a fejlesztőkörnyezet alapvetően a PROGRESS legújabb verziójának adaptált változata, a működés és ezen keresztül a programfejlesztés eltérő elveken nyugszik a hagyományos RDBMS programozáshoz képest. A legfontosabb új jellemző ebben a környezetben, az Internet "feledékenysége". Amikor a Webszerver letölt egy oldalt a browser számára, a művelet befejezése után a szerver "elfelejt" minden információt, ami azon az oldalon szerepelt. Az adatbázis tranzakciók során viszont alapkövetelmény a változók és az adatbázis tartalmának megőrzése a tranzakció lezárásáig. A felhasználó egyrészt bonyolult - több oldal letöltését igénylő - műveleteket hajthat végre, ami igényli a változók értékének fenntartását és a használt adatbázisrekordok zárolását, másrészt a tranzakció során biztosítani kell a kiindulási állapot visszaállíthatóságát. A WebSpeed ennek biztosítására vezette be a State Aware technológiát és az ehhez kapcsolódó ügynök (agent) fogalmát. State Aware program működésekor egy felhasználóhoz egy exkluzív ügynök kapcsolódik a szerveren és ez az ügynök tárolja a tranzakcióban definiált változók és adatrekordok értékét, függetlenül attól, hogy éppen melyik Web oldal került letöltésre.

A WebSpeed felépítése

Az előzőekben elmondottaknak megfelelően a WebSpeed felépítését alapvetően meghatározza az ITP (Internet Transaction Processing) technológia biztosításának szükségessége. A termék működését a következő ábra szemlélteti:

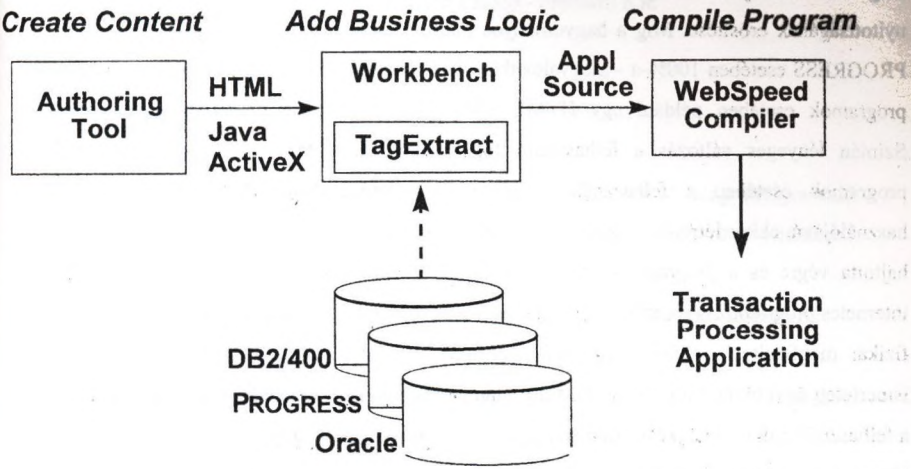


A WebSpeed központi eleme a Tranzakció Bróker. A Tranzakció Bróker felügyeli a már említett ügynököket (Transaction Agent), amennyiben szükséges, újat indít, vagy leállítja a kihasználatlanul működőket. Amikor egy felhasználó tranzakciót indít, a kérés a Webszerver közvetítésével jut el a Tranzakció Brókerhez. A Tranzakció Bróker a State Aware állapot biztosításához kijelöl egy ügynököt, amelyik exkluzívan kiszolgálja a felhasználót a tranzakció lezárásáig. A kijelölt ügynök tárolja a tranzakcióhoz tartozó változók értékeit, biztosítja az adatbázis elérést és a visszaállíthatóságot. A tranzakció lezárásakor az ügynök felszabadul és alkalmassá válik újabb tranzakciók kiszolgálására ugyanazon vagy más felhasználó igényének megfelelően.

Az ábrának megfelelően a felhasználó interakciója után a browser kérést küld a Webszervernek, a szerver továbbítja az alkalmazáshoz kapcsolódó adatokat a Tranzakció Brókernek. A bróker kijelöli a kiszolgáló ügynököt, amely ezután közvetlenül végzi a felhasználó kiszolgálását a tranzakció ideje alatt. Az ügynök biztosítja az adatbázis(ok) elérését, futtatja a programlogikát és feltölti változó értékekkel vagy - programtól függően - generálja az igényelt HTML oldalt. Az aktuális HTML oldalt végül a Webszerver továbbítja a felhasználóhoz.

Fejlesztés WebSpeed környezetben

A működés alapjainak megismerése után röviden WebSpeed környezetben történő fejlesztés fontosabb mozzanatait tartalmazza a következő ábra :



A kész alkalmazás futtatásakor a felhasználó HTML oldalakat tölt le, amelyek az alkalmazástól függően két alapvető csoportba sorolhatóak. A generált HTML oldalakat a szerveren működő program hozza létre teljes egészében, tipikusan ilyen egy lekérdezés elküldése utáni adatbázisriport. A hozzárendelt (mapped) HTML oldalakat a fejlesztés során hozzák létre tetszőleges HTML editor segítségével. Ezek az oldalak is tartalmazhatnak adatbázis vagy változó adatokat, de meg kell teremteni a kapcsolatot a HTML mezők és a változók illetve adatbázis adatok között. Ennek megfelelően a fejlesztés során először a hozzárendelt HTML oldalak létrehozása történik meg, majd az előzőekben leírt összerendelés. A WebSpeed által biztosított TagExtract modul biztosítja, hogy az összerendelés automatizáltan a képernyő tervezésekor megvalósulhasson. A hozzárendelés megvalósítása után a fejlesztő már ugyanabban a 4GL környezetben dolgozik, mint bármelyik hagyományos platformon, a WebSpeed integrált környezetben biztosítja a felhasználói interakciók kezelésének, az üzleti logika megvalósításának 4GL szintű programozását.

Különbségek és előnyök

Áttekintve a 4GL eszközök fejlődési fázisait, egyértelművé válik, milyen területeken követeli meg az Internet új technológia alkalmazását. Az egyik ilyen követelmény a fejlesztőkörnyezet nyitottságának erősítése. Míg a hagyományos platformokon a fejlesztés legnagyobb része - PROGRESS esetében 100%-a - megvalósítható volt az adott környezeten belül, az internetes programok esetében például egy HTML editor használata kikerülhetetlen követelmény. Szintén lényeges változás a felhasználó fogalmának átalakulása. A hagyományos 4GL programok esetében a felhasználó a programba történő belépéskor vált a rendszer használójává, ekkor létrejött a számára dedikált futtató környezet, amelyben tranzakciók sorát hajtotta végre és a programból történő kilépésig rendelkezésére állt saját környezete. Az internetes programozás során a felhasználó környezete az egyes tranzakciókhoz kötődik, nincs fizikai megfeleltetés a szerveren futó procedúra és a felhasználó között. A fentiekben ismertetett ügynökök a tranzakció lezárása után felszabadulnak. Lehetséges, hogy ugyanannak a felhasználónak minden egyes tranzakcióját más és más ügynök hajtja végre a szerveren.

Ugyanakkor a jelentős technológiai különbségek mellett is észrevehető, a 4GL környezet az Interneten is magában hordozza ugyanazokat a jelentős versenyelőnyöket, amelyek elterjedtségét indokolják. A hatékony fejlesztés biztosítása, a hagyományosan biztonságos és teljesítmény oldalról skálázható többfelhasználós működés, a tranzakció kezelés és adatintegritás megvalósítása azon a tulajdonságok közé tartoznak, amelyek lehetővé teszik komplex "mission critical" alkalmazások születését az Interneten is. Úgy tűnik, a piaci igény megjelenésével egyidőben rendelkezésre állnak a megfelelő eszközök is. A PROGRESS Software a WebSpeed kibocsátásával mindenesetre teljesítette "kötelességét": kipróbált környezetet biztosít minden fejlesztő számára, Internet alapú biztonságos RDBMS alkalmazások létrehozásához.

Az ORACLE Designer/2000-re épülő alkalmazásfejlesztések tapasztalatai

Fericsán Gábor, Szécsi László - FreeSoft Kft.

Bevezetés

Napjainkban az informatikai rendszerek kialakítását mindinkább az integráltságra való törekvés jellemzi. Mit is jelent az „integráltság" kifejezés ebben az értelemben? Elsősorban

- a lehetőleg egy helyen történő, felesleges redundanciáktól mentes adattárolást,
- a struktúrált és struktúrálatlan adatok együttes kezelhetőségét,
- az alkalmazások jelentős mértékben bővülő körének megfelelő teljesítménnyel történő kiszolgálását,
- egyszerű, jól áttekinthető használhatóságát,
- az alkalmazások telepítésének és üzemeltetésének biztonságos, ugyanakkor egyszerű és olcsó megoldhatóságát,
- a dinamikusan fejlődő technológiákkal való lépéstartás képességét,

röviden egy homogén, egységes fejlesztési és alkalmazási környezetet.

Aki már részt vett ezen céloknak megfelelő alkalmazói rendszerek kifejlesztésében, vagy akárcsak egy „kész" rendszer adaptálásában, meglévő alkalmazásokhoz való illesztésében, az tudhatja, milyen összetett feladatról van szó; kezdve az igények pontos megfogalmazásától egészen a kialakított rendszer üzemszerű átadásáig, karbantartásáig.

Az ORACLE Designer/2000 eszközei a mögöttük álló módszertani háttérrel együttesen alkalmasak a fentieknek megfelelő alkalmazói rendszerek teljes fejlesztési életciklusának támogatására, illetve kész alkalmazások előállítására, dokumentálására. Ahhoz azonban, hogy egy konkrét projekt sikeresen legyen, számos szervezési feltételnek kell teljesülnie, ideértve a szakembergárda felállítását és kiképzését is.

A Designer/2000-rel készített alkalmazásaink

Többéves ORACLE CASE rendszertervezési-fejlesztési tapasztalattal a hátunk mögött, 1996 elejétől foglalkoztunk konkrétan a Designer/2000 részletes megismerésével.

Feladatunk egy általános célú keretrendszer kialakítása volt, mely minden, a későbbiekben megvalósításra kerülő, Designer/2000, Developer/2000 alkalmazás mellé, változatlan formában továbbadható: Az alkalmazás az alábbi fő funkciókat valósítja meg :

- egy alkalmazással kapcsolatos telepítési információk rendszerezése (modul-munkehely összerendelés)
- általános kódszótár építés, karbantartás,
- paraméter beállítás (modul, modul/felhaszáló, modul/munkahely mélységben),
- a Developer/2000 Forms 4.5 által biztosított menühasználatnál kifinomultabb, dinamikus szerep-modul, szerep-kódcsoport, szerep-paraméter, illetve felhasználó-szerep összerendelés, (a szerepek kezelésének adatbázis-objektumokat érintő része adatbázis-szerepeken keresztül valósul meg)
- speciális, a fentiekkel összefüggésben álló adatszótár lekérdezések

A fejlesztésnek további célja volt, hogy a tervezett alkalmazás megvalósításával párhuzamosan egy Designer/2000-Developer/2000 *belső fejlesztési szabvány* jöjjön létre, mely kiterjed az ezen eszközökkel a későbbiekben végzendő fejlesztések minden fontos problémájára (módszertani lépések pontos specifikálása, a fogalmak egységes értelmezése, konvenciók, domain szabványok kidolgozása, stílusbeli ajánlások, a jelentkező kérdések, problémák egységes kezelésének módszertani kérdései, stb.).

Az elkészült alkalmazás 22 táblát, 15 nézetet, 27 modult tartalmaz. Elkészítési ideje 16 emberhónap (a szabványok kialakításával együtt) volt..

A Néprajzi Múzeum megbízásából augusztustól került megvalósításra a múzeum műtárgy-nyilvántartó- és publikációs rendszere. Fő jellemzői az alábbiak :

- 7 műtárgykategória (nagyszámú attributum : kategóriánként átlagosan 40, ebből kb. 10 egy-több kapcsolatból származó attributum),
- multimédia anyagok nagymennyiségű tárolása, lejátszása,
- a multimédia anyagokra is kiterjedő WWW publikációs lehetőség,
- tezauruszok felépítése, összerendelése műtárgyakkal, visszakeresés tezaurusz alapján (a tezaurusz olyan, előre definiált értékkészlet, melynek elemei között dinamikus felépíthető, többszörösen polihierarchikus kapcsolatok definiálhatók, illetve egy-egy elemére, annak nevén kívűl szinonimákkal is lehet hivatkozni - és az alapján visszakeresni).

Az elkészült alkalmazás kb. 30 táblát és ugyanannyi nézetet, és kb. 35 modult tartalmaz. Elkészítési ideje 11 emberhónap volt.

A Designer/2000-rel kapcsolatos tapasztalataink

Az eszközről általában ...

Legfőbb előnye, hogy valóban képes egy nagyvolumenű rendszerfejlesztés átfogó, minden fázisára kiterjedő támogatására, kész alkalmazások generálás útján történő előállítására, dokumentálására - függetlenül a projekt méreteitől, megközelítési módjától („zöld mezős" fejlesztés, adat-, folyamat orientált fejlesztés, gyors alkalmazás fejlesztés, meglévő rendszerekre épülő fejlesztés). Eszköztára rendkívül gazdag. A rendszer központi eleme a fejlesztői adatbázis (repository). Jól struktúrált szerkezete, hatékony elérése, dinamikus bővíthetősége gyakorlatilag bármilyen típusú projekt kezelését lehetővé teszi. A rendszerfejlesztés résztvevőinek kommunikációját igen hatékonyan segíti.

A közvetlen modulgenerálási lehetőségek fejlődtek a legszembetűnőbben. Már nem csupán prototípus szintű, annak egyszerűsített funkcionalitásával bíró modulok generálhatók, hanem - a sablon (template) állományok finomításával, preferenciák megfelelő beállításával - kb. 80-90%-os készültségű felhasználói modulok is. A generátorok választéka jelentős mértékben kibővült - jelen pillanatban 7 generátor áll rendelkezésre. A generátokkal kapcsolatosan külön kiemlítést érdemel a regenerálás lehetősége, valamint a meglévő alkalmazások adatbázisobjektumainak és bizonyos moduljainak visszafejtési (reverse engineering) lehetősége.

A fenti előnyök átfogóan elsősorban ORACLE RDBMS-re és a Developer/2000 fejlesztő eszközökre épülő fejlesztések esetében érvényesülnek.

Hibáit, hátrányait is sokáig lehetne sorolni. Mindenelőtt a kellő mélységű megismerésére fordítandó igen jelentős időt kell megemlíteni. Még magas szintű előképzettség esetében sem elegendő néhány hét arra, hogy valaki a „gyors alkalmazás fejlesztés" módszereit követhesse. Az megismerés nehézségeit mindenekelőtt a hiányos dokumentációk (ide értve a súgóban található leírásokat is) okozzák. Sok esetben csak hosszas próbálkozásokkal lehet rájönni, hogy egy bizonyos fogalomnak mi a pontos definíciója, vagy az elérni kívánt célt pontosan milyen úton, módon lehet megvalósítani.

Említést kell tenni a szoftver hibáiról is. Ezek súlyossága, mennyisége tekintetében az elmúlt 1-2 évben jelentős javulás figyelhető meg, mégis sok esetben igen komoly kellemetlenséget, fejtörést okozhatnak ezek

A tervező, fejlesztő által elkövetett, a rendszer által észlelt hibákról való tájékoztatás, javasolt korrekció színvonala az esetek jelentős hányadában igen szegényes, még akkor, ha egyértelmű, hogy a bővebb tájékoztatáshoz szükséges információ a rendszer rendelkezésére áll..

Jól szervezett csapatmunka azonban a fenti hibák, hiányosságok elhárításában, megoldásában is nagyon sokat segíthet.

Összeségében elmondható, hogy a szoftver beszerzésére fordított nem kis összeg megfelelő felhasználás esetében - hibáit, hiányosságait is figyelembe véve - akár egyetlen projekt során megtérülhet.

Házi szabványainkról ...

A Designer/2000 eszközeinek a tervező, fejlesztő szakemberek általi felhasználási szabadsági foka - a konkrét módszertani, szoftverhasználati kötöttségek ellenére - igen nagy. Ez egy-egy különböző összetételű fejlesztési projekt esetében egyértelműen az egységesség rovására megy. Ennek elkerülésére dolgoztunk ki egy olyan általánosan használható szabványosítási sémát (tervezési segédletet), melynek elemei gyakorlatilag minden projektben változtatás nélkül felhasználható. Ezen szabványok által érintett főbb területek :

- általános domain készlet,
- a tervezői, fejlesztői környezet minden lényeges objektumára kiterjedő névkonvenció-rendszer,
- a nem világosan definiált adatok értelmezésének pontosítása,
- a szabadon alkalmazható, TEXT típusú adatok egy körének egzakt értelmezése,
- a leíró adatok stílusbeli meghatározásai,
- a rendszeresen alkalmazott riportok körének, a felhasználás pontos módszertani specifikálása,
- a fejlesztői és végfelhasználói alkalmazások könyvtárstruktúrái,
- típusmegoldások felhasználási körének, módjának, dokumentálásának meghatározása,

- a véghasználói dokumentációk szerkezetének pontos meghatározása, az ezzel kapcsolatos MS Word template-k, makrók, valamint egy speciális generáló program alkalmazása,
- egyedi esetek kezelésének módszerei.

A fejlesztői team összetételéről ...

Az ORACLE CASE módszertana, valamint az ezzel foglakozó szakkönyvek pontosan meghatározzák azokat a szerepeket, szereplőket, amelyek, illetve akik egy-egy projekt során fontos szerephez jutnak. Ezen szerepek viszonylag nagy számossága miatt (az irodalmi adatok kb. 15-25, részben átfedésben álló szerepkörről tesznek említést) ezeket itt nem részleteznénk. Kiemelnénk azonban egy az irodalom által nem egyértelműen definiált, ugyanakkor tapasztalataink szerint a projekt végeredménye szempontjából rendkívül fontos feladatkört : a technológiai szakértő, vagy egyszerűen technológus feladatkörét.

Véleményünk szerint a technológiai szakértő konkrét tervezési-fejlesztési feladatokkal egyáltalán nem foglalkozik, viszont

- módszertani és eszközhasználati kérdésekben előkészíti a team munkáját,
- mind módszertani, mind Designer/2000 eszközhasználati tekintetben folyamatosan kontrollálja a fejlesztői team tagjainak munkáját,
- érvényesíti a belső szabványokban foglaltak maradéktalan betartását,
- részt vesz, vagy maga végzi az egyedi problémák megoldását, lehetőség szerinti általánosítását,
- a tapasztalatok alapján fejleszti a belső szabványokat és oktatja azokat,
- folyamatosan nyomonköveti a technológiai változásokat és azokat a szükséges mértékig átvezeti a projekten.

Természetesen egy nagyobb, több alkalmazást is magában foglaló projekt esetében több technológiai szakértő alkalmazása is indokolt lehet. Ekkor azonban gondoskodni tevékenységük megfelelő szintű összefogásáról, összehangolásáról is.

Az informatikai projektek egy lényeges sikertényezője

Egy rendszerfejlesztési projekt sikerének a korábban érintett technológiai, technikai jellemzők szükséges, de nem elégséges feltételei. Az is elmondható, hogy az ilyen projektek általában nem ezeken a feltételeken buknak el. A fejlesztés stratégiai

fázisának alapvetően szervezési kérdései azok, melyek a fejlesztés egészét tekintve a legkritikusabbak :

- van-e a cégnek a fejlesztendő terület működésére vonatkozó hosszútávú stratégiája,
- elkötelezett-e a cég vezetése, és főképpen első számú vezetője a tervezett fejlesztés mellett (szán-e, tud-e szánni elegendő időt a munkában való aktív közreműködésre)
- alkalmas-e a cég jelenlegi működése a tervezett rendszer befogadására (kidolgozattak-e a jelenlegi munkafolyamatok olyan mélységben, hogy ez a kérdés megválaszolható legyen),
- felvállalja-e a cég vezetése munkafolyamatainak - esetleg jelentős mértékű - megváltoztatását (és ebből adódóan szervezeti struktúrájának akár a felsővezetést is érintő hatásait).

Ha a fenti kényes kérdések megválaszolására nem is, de a mögöttük rejlő, nem kevésbé fontos munkafolyamat-szervezési kérdések elemzésére és újratervezésére kínál egy rendkívül hasznos és hatékony eszközt a Designer/2000 : a Process Modeller-t. Ez az eszköz alkalmas arra, hogy a jelenlegi munkafolyamatokat feltárva, elemezve, a tervező (a kulcsemberekkel egyetemben) egy jobb munkafolyamat-modellt hozzon létre. Mindezt úgy, hogy egyszerű eszközökkel kimutatja a jelenlegi folyamatoknak az idö és/vagy költségek szempontjából kritikus pontjait, és rendkívül jól szemléltetve - egyszerű, akár kész, akár egyedileg rögzített multimédia anyagok, egyszerű animációk beillesztésével - a tervezett munkafolyamatokat. Fő előnye, hogy a felhasználóval szorosan együttműködve a közös eredmény gyakorlatilag azonnal látható, kontrollálható.

Az ilyen módon előállított folyamatmodell elemei közvetlenül felhasználhatók a logikai tervezés adat-, funkció- és adatfolyam modelljeiben.

Jövőbeli elképzeléseink

Az ORACLE legfőbb erényének azt a technológa-együttest tartjuk, melynek fő komponensei :

- Universal Server (hagyományos struktúrált adatok, struktúrálatlan adatok, multimédia anyagok, adatfolyamok kezelése),
- kommunikációs technológiák (hagyományos ügyfél-kiszolgáló, valamint Internet alapú architektúrák, NCA)

- az alkalmazás-tervező és fejlesztő eszközök (elsősorban a Designer/2000 és Developer/2000, valamint Programmers/2000, Power Object),
- általános célú alkalmazói-, fejlesztői- és integrációs rendszerek (Interoffice, OLAP és adattárház eszközök),
- valamint a speciális célú alkalmazói rendszerek (pl. Financials).

Ezen technológiákra építve, a közös adatbázis háttér, valamint az egyes komponensekben definiált API interfészek révén előállíthatók a bevezetőben jelzett integrált alkalmazások.

A felhasználó szemével nézve a fentiek közül külön kiemelést érdemel az a három komponens, melyről megállapítható, hogy olyan tevékenységeket fednek le, melyek gyakorlatilag minden területen, minden cégnél előfordulnak és nagy jelentőséggel bírnak :

- dokumentációk kezelése (dokumentáció szervezés : Interoffice, dokumentumokban való keresés : Context opció),
- üzenetközvetítés, munkafolyamatok kezelése, dokumentációáramlás (Interoffice),
- vezetés (döntés) támogató rendszerek (ma használatos fogalommal élve - on-line elemző rendszerek : többdimenziós adattárolás, lekérdezés fejlesztői- és alkalmazói eszközei - Express Server, Object).

Ezek olyan fejlesztői területek, melyekre önálló Designer/2000 alkalmazások is készíthetők, illetve a Designer/2000 jól használható kész alkalmazásoknak a fenti eszközökkel való integrálására is. Ezévi elképzeléseink között a meglévő eredményeink konkrét projekteken keresztül történő bővítése, fejlesztése mellett ilyen irányú Designer/2000 szabványok, konvenciók kidolgozása is szerepel mind hagyományos ügyfél-kiszolgáló, mind pedig -kiemelt területként - WEB környezetre (tranzakciós Intranet, Internet alkalmazások).

Ezúton is szeretnénk kiemelni Jurányi Jánosné kolleganőnk munkáját, aki a fent említett fejlesztések résztvevőjeként, szabványosítási törekvéseink alkalmazója-, tesztelőjeként hasznos észrevételeivel, tanácsaival nagyban hozzájárult az eddig elért eredményeinkhez.

TARTALOMJEGYZÉK

FAIR Információs Rendszerek Kft.
NJSZT