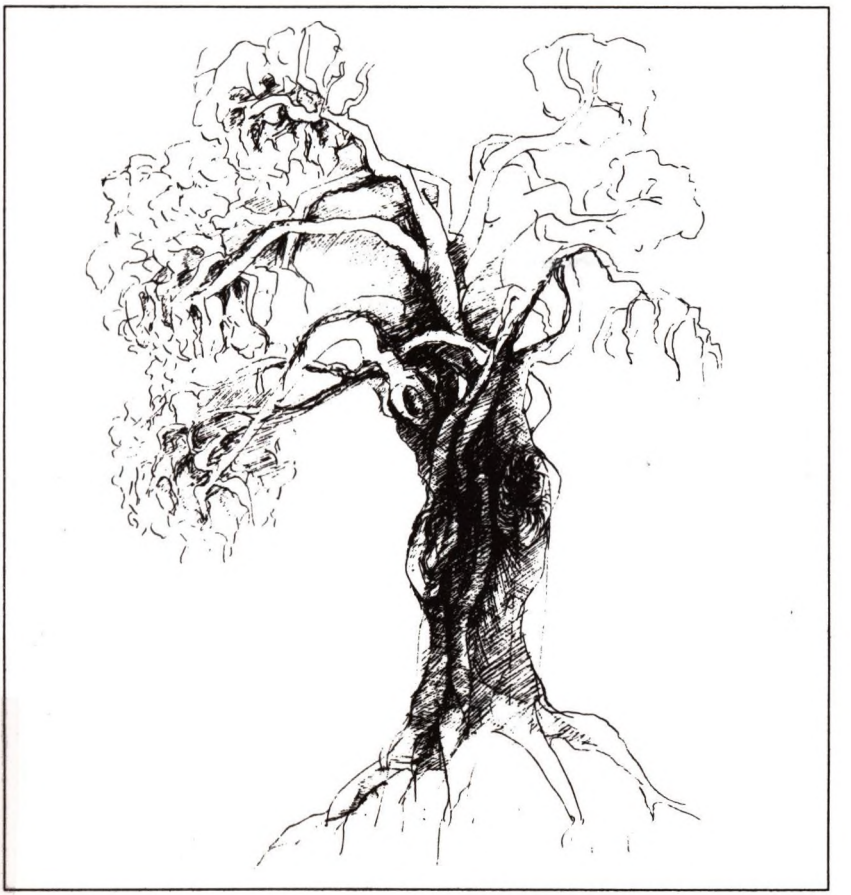
RELÁCIÓS ADATBÁZISKEZELŐK II. MAGYARORSZÁGI KONFERENCIÁJA

1995. január 25-27.



Szeretettel meghívjuk
az 1995. január 25 -27 - én
rendezendő

RELÁCIÓS ADATBÁZISKEZELŐK II. MAGYARORSZÁGI KONFERENCIÁJÁRA

FAIR Információs Rendszerek Kft.
Neumann János Számítógéptudományi Társaság

RELÁCIÓS ADATBÁZISKEZELŐK II: MAGYARORSZÁGI KONFERENCIÁJA 1995. január 25-27.

FAIR Információs Rendszerek
NJSZT

ÁFÉSZ Sokszorosító üzem VÁC

Tisztelt Résztvevő!

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

Ez a konferencia - éppúgy, mint az 1993-ban megrendezett előző - szolgálni kívánja a relációs adatbázis technológia hazai elterjedését azáltal, hogy szakmai fórumot teremt a tapasztalatok és vélemények cseréjének.

A konferencia előadásainak követését szeretné megkönnyíteni az az összeállítás, amit Ön most a kezében tart. Ebben a kötetben az elfogadott előadások egy részének írott anyaga található meg, a tervezett elhangzás sorrendjében.

Remélem, hogy a konferencián ismertetett gondolatok elérik céljukat és hozzájárulnak ahhoz, hogy az Ön munkája is könnyebb legyen.

Végezetül szeretnék köszönetet mondani mindenkinek, aki munkájával, ötleteivel és hangos vagy halk bíztatásával hozzájárult a konferencia megrendezéséhez.

1995. január 25.

Dr. Ecsedi-Tóth Péter
FAIR Információs Rendszerek

TARTALOMJEGYZÉK

Információs rendszerfejlesztés az OLAJTERV Rt.-ben
Fericsán Gábor (OLAJTERV Rt) **1. old.**

Ügyviteli (pénzügyi) rendszer fejlesztése az OLAJTERV Rt.-ben
Nagy Péter (OLAJTERV Rt.) **6. old.**

INFORMIX alapú kereskedelmi alkalmazások tapasztalatai (német nyelvterületen)
Halász János (KEG, Ausztria) **10. old.**

Tárolt eljárások szerepe üzleti alkalmazásokban
Németh Miklós (IQSOFT) **14. old.**

Hallgatói információs rendszer készítése ORACLE adatbáziskezelő felhasználásával
Gulyás János, Szabó István (Miskolci Egyetem) **19. old.**

Túl a relációs adatkezelésen
Balogh Judit, Juhász István (KLTE) **24. old.**

A beágyazott relációs adatmodell reprezentációja relációs adatbáziskezelő rendszerek felhasználásával
Kovács György (ELTE), Hajas Csilla (KLTE) **30. old.**

Fuzzy relációs adatmodell reprezentálása relációs adatbáziskezelő rendszerekben
Nikovits Tibor, dr. Kiss Attila (ELTE), Achs Ágnes (PMMF) **40. old.**

Disk és file menedzselés nagy méretű OSF/1 (UNIX), INGRES rendszereknél
Losonczi János (Orsz. Eü. Pénztár) **53. old.**

Második Generációs kliens - szerver architektúra gyakorlati alkalmazásának tapasztalatai Mo-n
Porkoláb Zoltán, Sziebig Ferenc (ALBACOMP C.G.) **62. old.**

IQDBA - az IQSoft interaktív adatbázisadminisztrátora
Gere Tamás (IQSOFT) **75. old.**

Nagyméretű adatbázisok kezelésével és adatkonverziókkal kapcsolatos tapasztalatok
Tornai Imre (Orsz. Eü. Pénztár) **79. old.**
CASE eszközök a 90 - es évek rendszerfejlesztésében
Bánné Varga Gabriella (CASE) **81. old**
Integráltság , nyitottság és testre szabhatóság Westmount I - CASE - ben
Balogh Kálmán (OPENSOFT) **90. old.**

Adatgyüjtés szervezés tervezése a KSH-ban ORACLE CASE segítségével
Györki Ildikó (KSH) **93. old.**
Alapnyilvántartások fejlődési tendenciái
Mezey Gyula **105. old.**
Indexelési stratégiák kialakítása
Mezei Gyula **113. old.**
A relációs adatbázisokban tárolt adatok felhasználása a COMSHARE vezetői inf. rsz.-ben
Durugy Gabriella, Bánkuti Zoltán (SzKI) **122. old.**
Ürlap és adatbáziskezelés JETFORM szoftverrel
Bíró Miklós, Hettessy Jenő, Remzső Tibor (SzTAKI) **126. old.**

A MEGA termékek jellemzése
Csillag Péter (QUANTUM) **132. old.**
A SAPIENS adatbáziskezelő és gyakorlati alkalmazása
Rada József (DUNAFERR) **138. old.**
VIOLA adatbázis lekérdező nyelv és prototipus készítő rendszer
Varga Kornél (STUDIDACT) **146. old.**
Az ESPERANT grafikus lekérdező és riport generátor
Vincze Attila (COMET) **153. old.**
Az ORACLE CASE oktatása a KLTE-en
Eperjesi Barnabás, Kellermann Lászlóné, Papp Ágnes (KLTE) **159. old.**

Adatmásolat készítés, az osztott adatkezelés egyik lehetősége INGRES környezetben
Molnár Imre (VT-SOFT) **165. old.**
Elosztott ORACLE adatbázisok
Mohay Tamás (ORACLE Hungary) **167. old.**
Az INFORMIX dinamikusan skálázható architektúrája
Sándor Gábor (OPENSOFT) **170. old.**

DB2 relációs adatbáziskezelő AS/400-on
Serfőző József (IBM Magyarorsyág) **173. old.**

A SYBASE relációs adatbáziskezelő rendszer
Czuprik Zoltán, Nagy Dániel (AXIS) **177. old.**
A QUICKOBJECT technológia adatbázis alkalmazásokban
Ertner Péter (IQSOFT) **194. old.**

A vállalati adatfeldolgozás eredményeinek konzisztenciája
Szász Eszter, Szenes Katalin (MATÁV) **197. old.**
Adatbázis-függetlenség és örökölt adatok megőrzése MAGIC-kel
Horváth Ferenc (ONYX) **198. old.**
Sémagenerálás és rekonstrukció SSADM-Engineer-rel
Horváth Katalin (IBIS) **200. old.**

Információs rendszer fejlesztése az OLAJTERV Rt-ben

Fericsán Gábor (OLAJTERV Rt.)

Bevezetés

Az OLAJTERV Rt. vezetése néhány évvel ezelőtt elhatározta, hogy a cég versenyképességének javítása érdekében mind a termelő tevékenységének, mind információs rendszerének technológiájában megpróbálja fokozatosan elérni az iparilag fejlett európai országok színvonalát. Műszaki tervező rendszerünket 1992-93 folyamán világviszonylatban is élenjáró három dimenziós létesítménytervező rendszer telepítésével, adaptálásával korszerűsítettük. Részben ez a rendszer kényszerítette ki, hogy munkamódszereinket is egyértelműen projekt-orientálttá tegyük. A vállalat vezetése a szervezetet is ennek megfelelően struktúrálta át. Mindezen elképzelések együttesen biztosíthatják annak a közvetlen célnak az elérését, hogy a vállalat 1995-ben nemzetközileg elfogadott minőségbiztosítási tanúsítványt szerezzen.

Az információs rendszer célja annak biztosítása, hogy minden vállalati vezető (felsővezetők, különböző szintű irodai/szervezeti vezetők, projekt vezetők) a megfelelő időben hozzájuthasson az általa irányított szervezet működésével kapcsolatos információkhoz, melyek munkájukhoz, döntéseikhez szükségesek. Másképpen fogalmazva, az információs rendszer kialakításának fő célja az, hogy a vállalat stratégiai elképzeléseihez, céljaihoz illeszkedve erőforrásként, eszközként segítse azt.

Az előadás témája az ezt kiszolgáló számítógépes információs rendszer kialakítása során szerzett legfontosabb tapasztalatok összefoglalása.

Előzmények

Az 1988. előtti időszakot az OLAJTERV ügyviteli területének számítógépes kiszolgálásában egy-egy önálló ügyviteli tevékenység támogatása, majd a korábban kidolgozott, illetve vásárolt rendszerek szintentartása jellemezte. Ezek a programok egymástól teljesen függetlenül, heterogén eszközparkon, különböző szoftverek felhasználásával készültek.

1989 és 1992 között fogalmazódott meg először egy egységes rendszer igénye, mely elképzelés középpontjában a rendszertechnikai környezet homogenizálása (UNIX operációs rendszer, ORACLE adatbáziskezelő) állt. Ekkor került megvalósításra, mintegy másfél éves belső fejlesztés eredményeképpen, az első ORACLE alkalmazás, a könyvelési rendszer. Röviddel később ezt követte az anyag- és eszköznyilvántartási rendszer kidolgozása, mely már közvetlen adatkapcsolatban állt a könyvelési rendszerrel.

1993-ben, a bevezetőben már említett stratégiai változások eredményeképpen a korábbiakhoz képest teljesen átdolgozott koncepció fogalmazódott meg. Ez már nem csupán az ügyviteli tevékenységek közvetlen támogatását célozza, hanem egy olyan

integrált rendszert, mely képes a projektjeink kézbentartásához, valamint a vállalat irányításához szükséges információk szolgáltatására is. Továbbá tájékoztatást ad a szervezetben zajló valóságos folyamatokról, mindenki számára lehetővé teszi saját feladatainak elvégzésére való koncentrálást, "kikényszeríti" a tevékenységek, folyamatok vállalati szintű átgondolását, megfogalmazását.

A rendszer felépítése, a megvalósítás ütemezése

Az 1993. májusban elfogadott rendszerfejlesztés stratégiai koncepció az információs rendszert három alrendszerre osztotta :

A legnagyobb jelentőséggel a ***vezetői tájékoztató rendszer*** bír, 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. Időben ez az ügyviteli rendszer bevezetését követően 1995. első félévében kerül kidolgozásra, eddig még ki nem választott lekérdező eszköz felhasználásával, vagy egy kész alkalmazás adaptálásával. A rendszer tervezett bevezetésének időpontja 1995. július 1.

A ***projekt menedzsment rendszer*** már szerződött, illetve előkészítés alatt álló megrendeléseink kézbentartását (határidőre, adott műszaki tartalommal való elvégzését) hivatott biztosítani, továbbá a társaság piaci környezetével áll informális kapcsolatban. Így olyan információkat kezel, amely segíti a már leszerződött munkáink határidőre, jobb minőségben történő kiszállítását, ugyanakkor tájékoztat szabad kapacitásainkról, illetve piaci lehetőségeinkről. Kész termék beszerzését választottuk, a rendszer, melynek eszközspecifikációját a következő fejezet tartalmazza, 1994. január óta üzemel

Az ***ügyviteli rendszer*** végeredményben pénzügyi helyzetünk, lehetőségeink ismeretében a társaság fizetőképességét hivatott biztosítani, elsősorban költségeink szigorú kézbentartásával. Az ügyviteli rendszer elsődleges célja ennek megfelelően az, hogy pontos, naprakész információt biztosítson a társaság tervezett és tényszerű költségfelhasználásról, folyamatosan változó pénzügyi helyzetéről. Fontos elvárás a projekt menedzsment rendszerrel való szoros kapcsolattartás, ugyanis az árbevételek és költségek elsősorban a projektek kapcsán jelennek meg. Az ügyviteli rendszer a korábbi tapasztalatokra épülő belső rendszertervezéssel, majd az implementációs fázisban külső partner cég (FAIR Információs Rendszerek Kft) bevonásával került kidolgozásra 1993. szeptember és 1994. december között. A rendszer 1995. január óta üzemel. Részletesebben lásd a későbbi fejezetekben.

A fejlesztési koncepcióban megfogalmazódott továbbá annak igénye, hogy a rendszer a későbbiekben legyen integrálható a vállalat tervezett dokumentumkezelő rendszeréhez, illetve illeszkedjen a kialakítandó irodaautomatizálási környezetbe.

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 (a munkahelyek száma 16) egy ALR POWERPRO alkalmazás szerver gépen futnak, SCO UNIX operációs rendszerfelügyelete alatt. A munkahelyek DEC VT510 terminálok, melyek az épület különböző részeiben felállított három db. terminál-szerver eszközön keresztül kapcsolódnak az ETHERNET hálózatra.

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 ***vezetői tájékoztató rendszer*** tervezett munkahelyei (kb. 16 munkahely) szintén MS-Windows alapú PC-k, melyek FTP PC/TCP szoftverrel kapcsolódnak a hálózatra.

Az ügyviteli fejlesztés közvetlen céljai, tapasztalatai

Főbb célkitűzések

A rendszerrel szemben megfogalmazott főbb célkitőzések a következők voltak :

- Egységes fogalmi rendszer

  Az egységes fogalmi rendszer kialakításának célja, hogy az adott rendszerben előforduló szakmai, környezeti fogalmakat minden érintett rendszertervező, fejlesztő, majd a későbbi aktív és potenciális felhasználó számára azonos módon, egyszerűen, közérthetően fogalmazza meg, definiálja.

- Egységes tranzakciókezelés

  Az egységes tranzakció-kezelési mechanizmus lényege, hogy minden adatmódosító tétel tárolásra kerüljön, annak jellemző adataival. Minden, ilyen értelemben vett tranzakció egyedileg azonosított, adatai szükség szerint mindaddig módosíthatók, ameddig más rendszerek felé feladásra nem kerültek, amely időpont után viszont már csak sztornózhatók, ennek az érintett alkalmazás felé történő közlésével.

- 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 kitűzött célok, az egységes rendszer megvalósítása gyakorlatilag nem lehetséges megbízható módszertani háttér nélkül. Különös tekintettel szükséges ez az alábbiak miatt :

- egységes gondolkodás a team minden résztvevője között,
- a feladatok egymásra épülő, teljeskörű, kontrollált elvégezhetősége,
- dokumentálás megfelelő színvonala.

Mindezek érdekében az ORACLE CASE*Method módszertanát, illetve az erre épülő CASE eszközeit használtuk fel a fejlesztés során. Ez, vagy ehhez hasonló módszertanok, eszközök alkalmazása rendkívül alapos felkészültséget, gyakorlatot igényel.

A módszertani kérdésekkel hozható összefüggésbe, az, hogyan oldjuk meg a rendszer dinamikájának kezelését. Több lehetőség adódik, melyek alkalmas kombinációja adja legjobb módszert. Ezek :

- paraméterek többszintű alkalmazása,
- dinamikus adatszerkezet támogatására épülő eljárások alkalmazása,
- a CASE eszközökben megtestesülő fejlesztői adatbázis aktualizálása, majd ismételt adatbázis és/vagy akalmazás generálás.

Projekt szervezés kérdései, tapasztalatai

A fejlesztési munka eredményességét alapvetően meghatározó tényező a munka szervezettsége, a feladatokhoz illeszkedő projekt szervezet kialakítása, a munkavégzés tárgyának pontos megfogalmazása, a munka előrehaladásának követése, ellenőrzése. Különösképpen a következők figyelembe vétele szükséges - még magasszintű CASE alkalmazás mellett is :

- konvenciók kialakítása, teljeskörű, tudatos alkalmazása,
- módszertani szakember alkalmazása, folyamatos egyeztetések,
- dokumentációs felelős kijelölése, egységes dokumentációs technikák kidolgozása, alkalmazása,
- jól elhatárolható szerepkörök kijelölése,
- rövid és hosszútávú feladatmeghatározások kijelölése, ellenőrzési, visszacsatolási módszerek alkalmazása.

Jövőbeli elképzelések

Az eredeti elképzeléseknek megfelelően hátralévő feladatok

- az ügyviteli és projekt menedzsment rendszerek integrációja,
- a vezetői tájékoztató rendszer kidolgozása.

Az ügyviteli rendszer vázolt fejlesztése során megfogalmazott, javasolt feladatok :

- a funkció-, illetve modul tervezés és implementálás a CASE eszközök felhasználásával,
- az ORACLE CDE technológia használatának teljeskörű bevezetése,
- az új CASE eszközök (CDE2/CASE) megismerése, alkalmazásba vétele, átdolgozása ennek megfelelően.

Ügyviteli (pénzügyi) rendszer fejlesztése az OLAJTERV Rt-ben

Nagy Péter (OLAJTERV Rt.)

Bevezetés

Az ügyviteli rendszer az RT. pénzügyi helyzetének, lehetőségeinek ismeretében a társaság fizetőképességét hivatott biztosítani, elsősorban költségeinek szigorú kézbentartásával. Az ügyviteli rendszer elsődleges **célja** ennek megfelelően az, hogy pontos, naprakész információt biztosítson a társaság tervezett és tényszerű költségfelhasználásról, folyamatosan változó pénzügyi helyzetéről.

A rendszer alrendszerei

Az ügyviteli rendszer alrendszerei két kategóriába sorolhatók.

- A tényszerű adatokat kezelő, feldolgozó rendszerekre:
  - **pénzügyi** alrendszer
  - **könyvelési** alrendszer
  - **személyügyi** alrendszer
  - **anyag-, eszköznyilvántartási** alrendszer.
- Ezek adatait vállalati szinten szintetizáló, vállalati szintű gazdálkodási funkciókat realizáló:
  - **controlling** alrendszer.

Ez utóbbi alrendszer az, amely közvetlen, folyamatos kapcsolatban áll mind a vállalkozási, mind a vezetői információrendszerrel.

A kialakítandó számítógépes ügyviteli rendszer mindenekelőtt abban különbözik a korábbi megoldásoktól, hogy ***egységes rendszertechnikai alapokra épül fel***. Ennek legfontosabb elemei az egységes fogalmi rendszer, dokumentum azonosítási-, kezelési eljárások, tranzakció-kezelési módszerek, az ezekhez tartozó igazolási, stornírozási eljárások, az alrendszerek közötti kommunikációs mechanizmus. Ezen módszertani alapelvek biztosítják az ügyviteli folyamatok, adatszolgáltatások jobb áttekinthetőségét, magasabb színvonalú dokumentáltságát, az információk hatékonyabb kinyerését, visszakeresését, sok esetben egyszerűsítik a folyamatokat.

Az azonos hardver és software platformon kívül a különböző alrendszerek kapcsolatai valósítják meg a célul kitűzött egységességet, redundancia mentességet és integráltságot. Ezek a kapcsolatok teszik lehetővé, hogy a különböző önálló alrendszerek halmazából **egyetlen** és **közös rendszer** álljon elő.

Az alrendszerek két módon kapcsolódhatnak egymáshoz:

- Egy adott adathalmazt (Pl. különböző törzsek) csak az egyik alrendszer megfelelő funkciója módosíthat, a többi alrendszer ezt csak lekérdezheti.
- Az alrendszerek egymásnak szolgáltatnak adatokat , olymódon, hogy az egyik a saját adataiból a másik megfelelő kérése alapján, - amely megmondja, hogy mely adatokat, hogy csoportosítva, milyen sorrendben kéri - kiválasztja és átadja (feladja) a másik alrendszer számára, amelyik ezt feldolgozza.

A fejlesztés tapasztalatai a Tervezési fázisban

A rendszerfejlesztés indulásakor a fejlesztő team szerencsés helyzetben volt, mivel egyrészt minden területen már működött valamilyen számítógépes alkalmazás, másrészt a már működő alkalmazások közül néhánynak a kifejlesztő szakemberei a team tagjai voltak. Ebből kifolyólag a helyzetfelmérési munka tulajdonképpen a már felhalmozódott ismeretanyagnak a számviteli törvényeknek és és az RT számviteli és üzletpolitikájának megfelelő aktualizálása volt.

A rendszer tervezését az ORACLE Case Designer és Case Dictionary programcsomagjainak segítségével kezdtük el. Talán ismeretes, hogy az ORACLE Case Method (amely módszertan alapján az említett programok dolgoznak) a rendszertervezésben elsődlegesnek az adatmodell megfelelő kialakítását tekinti, és erre építi fel a funkció modellt. Mi is - osztva ezt az elvet - ennek megfelelően láttunk a rendszer tervezéshez hozzá.

Minden alrendszernek a rendszerfejlesztők közül volt egy gazdája, aki az adott alrendszer tervezését elsősorban csinálja. A Case-ben egy-egy alrendszert egy-egy alkalmazásnak definiáltunk. Ezen kívül létrehoztunk egy adminisztrációs alkalmazást, amelyben a közös objektumokat (domének, entitások) határoztuk meg, majd a közös objektumokra az azokat használó más alkalmazásoknak hozzáférési jogokat (olvasás) adtunk. Ezzel elértük, hogy a közös objektumok módosítása rögtön a többi alkalmazás számára látható legyen. (Eleinte felvetődött az a megoldás,hogy nem definiálunk adminisztrációs alkalmazást, hanem a közös objektumokat az alrendszerek között szétosztjuk és minden alkalmazás jogokat ad a másiknak, de ezt elvetettük, mivel az első esetben volt gazdája a közös objektumoknak, a második esetben ennek elérése nem látszott tisztán.). Ilyen felallásban készítettük el a rendszer adatmodelljét.

A funkció modell tervezésekor maguknak az ügyviteli funkcióknak a definiálása viszonylag könnyen ment a fentebb ismertetett okok miatt, az ügyviteli helyzet ismeretében. A hierarchikus funkciómodell kialakításakor azonban gondot okozott, hogy milyen mélyen állapítsuk meg az elemi funkciók szintjét. Tulajdonképpen ez már a kivitelezési fázisra is hatással van , mivel ha az elemi funkció szintjét nagyon mélyen határozzuk meg, akkor a tervezési fázisban kell nagyon aprólékosan meghatározni

mindent és az implementáció lesz viszonylag rövid és egyszerű, ha nem tesszük mélyre az elemi funkciók szintjét a tervezés rövid és a megvalósítás bonyolult.

A két lehetőség közül az utóbbit választottuk a következő okok miatt :

- a funkció modellezésben nem volt még megfelelő tapasztalatunk,
- az adatbázis implementálásában (forms,menü,riport) készítésben viszont volt mind a fejlesztő, mind a megvalósító teamnek tapasztalata,
- egy Case Dictionary probléma , amelyet tapasztaltunk. Eszerint amikor a megtervezett funkció hierchiából modulokat generált a Case, akkor csak az elemi funkciókat vitte át modulnak, a magasabb szinteket nem és a hierarchiát is elfelejtette és saját menü hierarchiát (a modul fajtája -forms,report) szerint) állított elő.

A választásunk eredményeként a funkciókat egy-egy alkalmazás esetében olyan mélységig vittük, hogy az majd az SQL*Menu egy-egy iteme legyen, függetlenül a bonyolultságtól.

A fejlesztés előre haladtával magunk is tapasztaltuk az a tényt, hogy nem érdemes nagyon gyorsan haladni előre, bőven kell időt hagyni a tervezési fázisra mivel minél jobban vissza kellet nyúlni az időben módosításkor annál több munkát kellett végezni. Tehát inkább alaposan kell tervezni és lassabban, mint gyorsan és felületesen. (Természetesen ezt az igazságot még nem minden megbízó és felhasználó ismeri!)

A tervezés végső fázisának tekintettük , amikor az adatbázis objektumait generáló scripteket a CASE Dictionary segítségével előállítottuk. Problámánk csak az volt , hogy a CASE 6-os Oracle Rdbms-nek megfelelő scripteket állított elő, nekünk a rendszert 7-es Rdbms-ben kellett definiálni, de néhány konvertálással hárítottuk a problémát.

A fejlesztés tapasztalatai a Kivitelezési fázisban

A rendszer megvalósításához rendelkezésünkre állt az ORACLE Case Generátor programcsomag, amely segítségével az elemi funkciókból (illetve az ezekből keletkezett modulokból) kész formsokat, menü alkalmazásokat illetve riportokat lehet generálni. Ezt a lehetőséget azonban csak akkor lehet igazán hatékonyan használni, ha az elemi funkció szintet nagyon mélyen határozzuk meg. Ezért úgy döntöttünk, hogy nem vesszük igénybe ezt a lehetőséget, az elkészítendő objektumok (forms,report) definiálását a Case Dictionary megfelelő riportjai illetve külön készített egységes képernyő tervek és output képek alapján végeztük el.

A kivitelezéshez az SQL*Forms, SQL*Menu, SQL*Reportwriter és az SQL*Plus termékek álltak a rendelkezésünkre.

A választott út miatt a forms-jaink meglehetősen bonyolultaknak sikerültek, de felhasználva a 7-es Rdbms új lehetőségeit (adatbázis triggerek, lefordított procedurák és package-ek) nagy segítségünkre voltak. A Case-ben tervezett, majd az objektum kreálásban megjelent megszorítások (Primary key,unique key, checks és foreign key) forms-ban való automatikus beépülésének lehetőségét maximálisan igénybe vettük.

A riportokat SQL*Riportwriterrel készítettük. Tapasztalataink szerint nagyon kell érteni ehhez a termékhez, ha jól szeretnénk dolgozni és meglehetősen aprólékos programozási munkát követel. Azonban nagyon sok olyan lista és riport - főleg a bonyolultabbak - volt, amelyet csak az SQL*Reportwriter-rel tudtunk megoldani illetve csak ezzel tudtuk az egységes lista és dokumentum formákat biztosítani.

A rendszer bevezetésének tapasztalatai

A rendszert egy UNIX-os számítógépen fejlesztettük, az ORACLE-ban az összes objektumot egy fejlesztő userként (a rendszergazda) definiáltuk és annak tulajdonába kerültek.

A program tesztekhez az adatokat két lépcsőben szerettük volna betölteni:

- a törzsadatokat (szervezetek,partnerek,munkaszámok,dolgozók,termékek) a meglévő rendszerekből leválogatva egy meghatározott nem túl nagyszámú halmazt,
- az input formsok tesztelésekor már megkomponált minta szerint bevinni az adatokat.

Ez az elv jó lett volna, ha a riportokat a formsok után kellett volna tesztelni, mivel azonban ez párhuzamosan történt a riportokhoz sajnos kézzel kellett generálni forgalmi és tranzakciós adatokat.

A rendszertesztet úgy végeztük el, hogy a törzs- és kódadatok a program tesztek után már az éles helyzetnek megfelelően kialakítottuk (már meg voltak a karbantartó formsaink!), elkészítettük azokat a programokat, amelyek a meglévő rendszereinkből a szükséges adatokat konvertálják és áttöltik az új és integrált rendszerbe. Ezekkel a programokkal a rendszerteszthez csak egy megfelelő számú mintát vittünk át.

Itt kell megjegyezni, hogy a különböző hardver és szoftver platformon működő régi rendszereinket konvártáló programok megtervezése és megvalósítása felért egy új alkalmazás elkészítésével. A legbonyolultabb volt a Turbo Pascal-ban írt, vásárolt rendszerből ki csalogatni az adatokat, az RT-nél írt dBase alkalmazások migrálására az ORACLE már adott eszközöket. A legsimább út azoknál az alkalmazásoknál volt, amit már ORACLE 6-os Rdbmsben futottak, ahonnan az adatbázis linkek használatával könnyen át lehetett a kívánt adatokat tölteni.

Pillanatnyilag a rendszerteszt végén járunk, január közepén indul élesben az új rendszer, amelyet körülbelül február végéig párhuzamosan futtatunk a régi rendszerekkel és márciusban már csak az új és teljes ügyviteli rendszer létezne az OLAJTERV RT-nél.

INFORMIX alapú kereskedelmi alkalmazások tapasztalatai német nyelvterületen

Dr. HALÁSZ JÁNOS
(EDV Consultor KEG, Ausztria)

Alkalmazások 1988 -1994 között

Ausztria: - Kórházi páciensadminisztráció és elszámolás
- Díjbeszedő rendszer
(gáz, víz, csatorna, szemét - áram nélkül)
- Projekt elszámolás és követés
- Anyagrendelés távrögzítéssel
- Kábel - TV elszámolás (fejlesztés alatt)

Svájc: - Dolgozók óraelszámolása
- Modulok ALX -hez

NSZK: - Anyaggazdálkodás (részvétel nagyprojektben)
- Termelésirányítás (részvétel nagyprojektben)

INFORMIX : 2.xx - 4.xx

UNIX : AIX, SCO Unix, SINIX

Az alkalmazások felépítése

Törzsadatállomány kezelés

Forgalom rögzítés

Számlázás

Statisztika

Rendszerprogramok

Törzsadatok

Utcanév Tábla ← Helységnév Tábla

- Személyi adatok
- Vevőkör Adatok
- Bank Adatok
- Tarifák (egyes alkalmazásoknál követi az időrendi változásokat, gyakrabb viszont , hogy felülírják és az eredeti érték az elszámolás - rekordokban marad ...Nem szép relációs megoldás , de használható !)
- Alkalmazásspecifikus törzsadatok

Rendszerprogramok

- az alkalmazás paraméterei

- változtatható szövegek (Help , Menü , Felirat)

- jelszó , a munkatárs adatai

- security (melyik komponens ki által hívható)

A törzsadatkezelő programok felépítése

1) Gyors eljárás : SQL - alapú , szerkesztővel ,
FORMBUILD utasításokkal " feljavítva "

Csak egyszerü kezelésekhez
ajánlható (pl. kód - leírás - összeg).
5 - 10 perc alatt is elkészíthető !

2) Összetett : Több éves gyakorlatom után , a meglévő
szabványokat , könyvtárakat felhasz -
nálva 1 - 2 nap alatt előállítható .
Egy komplikált törzsadatkezelés is
megvalósítható 1 - 2 hét alatt .

Menüvezérlés

KERES	ELŐRE	HÁTRA	ÚJ	MÓDOSÍT	TÖRÖL	NYOMTAT
(Query	Next	Prev	Add	Update	Remove	Out put)

Tapasztalatok

KERES - a 4.10 verziótól kezdve a CONSTRUCT megszakítható és ezt nagyon ajánlom beépíteni .

téves (ESC) leütéséből adódó hosszú keresés elkerülhető a "field touched " funkció beépítésével .

- ha a keresés eredmény több rekord , úgy érdemes egy Fxx billentyűvel az összes rekord átvételét lehetővé tenni és köztük lapozni (ELŐRE - HÁTRA)

MÓDOSÍT - az SQL nem engedi a direkt keresést , hanem először Query - t kér . A programozott megoldásnál praktikus a direkt keresést beépíteni .

NYOMTAT a régi rendszerekben külön menü volt
kezeléshez beépíteni .

Rugalmas , ha a nyomtatás paraméterezhető .
Output : képernyőre
file - ba
Lista : - az egész törzsadatállomány
- részhalmaz választása

Számlázás

- Befizetési csekk

A legalsó sorban OCR - B típussal kerül nyomtatásra

- a befizetendő összeg
- számlaszám
- vevőkód
- a célintézet kódja
- ellenőrző kód

A bankban (Svájcban a postahivatalban) a befizetet csekkről az adatokat beolvassák , az eredményt összegezve floppy -n , vagy szalagon visszaküldik és onnan bekönyvelhető .

- Bankátutalással

szalagra kerül , a bank az adathordozót kapja meg. A befizetést szintén mágneses adathordozón küldik vissza , a vevőnek pedig kivonatot küldenek. A bekönyvelés a csekk - eljáráshoz hasonlóan történik .

Általános tapasztalatok

- Érdemes optimalizálni a komplikáltabb SELECT -eket .
VIGYÁZAT : az SQL és 4GL Select végrehajtása között
lehetnek eltérések
az Informix belső Optimizer-e másképp
működik kis - és nagy mennyiségű
adat esetén .
- A felhasználók hiba , vagy problémák felmerülése esetén sürgős hibaelhárítást várnak el .
MEGOLDÁS : Modem

Tárolt eljárások szerepe üzleti alkalmazásokban

Németh Miklós (IQSOFT RT)

A kliens-szerver architektúra és a tárolt eljárások

A ***többszintű*** (multi-tiered) rendszereknél az alkalmazás intelligenciáját nem egyetlen program testesíti meg hanem több, amelyek tipikus esetben külön számítógépeken működnek. Elég tágan értelmezhető, hogy mely architektúrájú rendszerek tekinthetők többszintűnek. Az XBASE (Clipper, stb.) alkalmazásokat - akár hálózaton működik akár nem - nem lehet többszintű rendszernek tekinteni, mivel az alkalmazás teljes intelligenciáját a hálózat munkaállomásain működő programok tartalmazzák. A kliens-szerver architetúrájú rendszerek természetüknél fogva többszintűek. A NetWare fájlszerveren működő Btrieve NLM (record manager) szervert használó alkalmazásokat tekinthetjük az egyik legegyeszerűbb többszintű rendszer esetének; a Btrieve NLM szerver egy SQL-szerverhez képest ugyan kevés szolgáltatást kínál de az adatbázismanipuláció legalapvetőbb szolgáltatásait már nyújtja (főleg egy XBASE rendszerhez képest). Jóval komplexebb szolgáltatásokat nyújtanak az SQL-szerverek (Oracle, Gupta SQLBase, Watcom SQL, Sybase/Microsoft SQLServer, INFORMIX OnLine, Ingres, IBM DB2, IBM DB2/2, IBM SQL/400, stb.). A kétszintű (two-tiered) alkalmazások legkorszerűbb megoldásai csak tárolt eljárásokat támogató adatbáziskezelő szerverekkel valósíthatók meg. A háromszintű (three-tiered) rendszerekben az adatbázisszerver és a kliens-oldali programok közé egy újabb komponens ékelődik, az alkalmazás-szerver, amely még komplexebb (heterogén) feladatokat is el tud végezni. Háromszintű rendszerek a tranzakciómonitorokkal (pl. Novell TUXEDO) vagy más hasonló rendeltetésű eszközökkel hozhatók létre. A többszintű adatbázisorientált üzleti alkalmazások szorosan kötődnek a kliens-szerver architektúrához.

A többszintű alkalmazások egyik legfőbb előnye, hogy a teljes alkalmazásrendszer teljesítményét viszonylag költségtakarékosan lehet növelni. Az adatbázis (alkalmazás/tranzakció) szerver(gép) teljesítményének növelése elegendő a teljes (hálózati) rendszer összteljesítményének növeléséhez. Mivel egy szakszerűen megtervezett többszintű rendszer eleve (azaz természeténél fogva) minimálisan terheli a hálózatot, általában nincs szükség a hálózati (kommunikációs) hardver bővítésére. A kliens gépek "csak" akkora teljesítményűek kell legyenek, hogy a szerverről érkező adatokat grafikusan vagy karakteres módon megjelenítsék. A grafikus megjelenítés és az ezzel gyakran párosuló vizuális, interaktív, integrált, interpretatív 4GL front-end fejlesztőrendszerek (SQLWindows, PowerBuilder, Smalltalk) erőteljes kiépítésű kliensgépeket (486/33 8Mb RAM) igényelnek. A kliens-gépeknek éppen elég az adatprezebntáció feladattával megbirkózni. Az ellenkező következtetésre is juthatunk, miszerint ha már úgyis "erős" a kliensgép, akkor már nem különösebben többletteher az alkalmazás feldolgozásainak elvégzése. Ebben az esetben is azonban megmarad az a probléma, hogy a feldolgozandó adatok nem a kliens-gépen lokálisan helyezkednek el, hanem a hálózaton keresztül át kell mozgatni a feldolgozáshoz, és ez az, ami végül is "megölné" a rendszert.

A gazdagép(host)-terminál architektúrájú rendszerekkel szemben technológiai szempontból kevesebb előnye van a kliens-szerver architektúrának mint a fájlszerveres rendszerekkel szemben. A túl sok grafikus terminál (X-Window) működése olyan teher a gazdagép számára, ami miatt a rendszer teljesítménye elmarad a Windows kliensgépekből és a gazdagéppel azonos kiépítettségű kliens-szerver architektúrájú rendszerek mögött. Megfelelően nagy X-terminálszám esetén a kliens-szerver teljesítménye nagyobb. Ha mégis lenne olyan gazdagép-grafikus terminál konfiguráció, amely nagy terminálszám esetén is jobb teljesítményű mint a Windows-os kliens-szerver rendszerek, akkor is fennmarad még az alábbiakban kifejtett probléma.

Napjainkra a PC-k térhódítása olyan mérvűvé vált, hogy sztenderd irodainformatika elképzelhetetlen ezen kategóriájú gépek nélkül, és az azokon futtatott Microsoft (MS) Windows nélkül. Az MS Windowsra kifejlesztett irodaautomatizálási szoftvercsomagok elterjedtsége és népszerűsége olyan fokú, hogy korszerű integrált (operatív) vállalati információs rendszer (VIR) nem képzelhető el úgy, hogy az MS Windows környezetben keletkező (OLE) dokumentumok ne kerüljenek be a VIRbe. Számos hiányossága ellenére az MS Windows vált a sztenderd felhasználói felületté, és elképzelhetetlen ma már az, hogy egy újonnan kifejlesztett korszerű VIR ne ezt a felületet támogassa.

Ha elfogadjuk ezt az értékelést, akkor arra a következtetésre juthatunk, hogy a kliens-szerver architektúra ugyan nem sokkal kedvezőbb technológiájú mint a gazdagép-terminálos rendszerek, de a PCken működő MS Windows-ok tömege miatt gyakorlatilag csak kliens-szerver architektúrájú VIReket célszerű fejleszteni.

A többszintű rendszerek lehetővé teszik az erőforrások optimális megosztását - minden hardver/szoftverkomponens a számára ideális feladatot végzi - és helyes technológia alkalmazása esetén minimális a komponensek közötti adatcsere is.

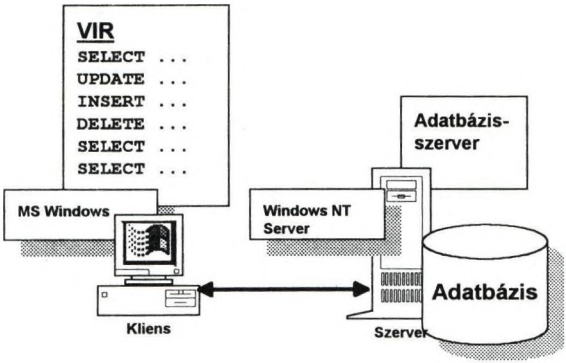
Az alkalmazás intelligenciájának több szintre való szétosztása javítja az alkalmazás strukturáltságát (rétegzettségét), és így sokkal inkább kézbentartható, hibatűrő rendszerhez juthatunk.

A többszintű rendszerekkel szemben a legjelentősebb kifogás az, hogy a gazdagép-terminálos rendszerekhez képest nagyobb a hálózati (a megjelenítés helyére való adateljuttatás) terhelése, illetve a rétegek (szintek) közötti kommunikációból adódó többletidő.

A fájlszerveres architektúrával szemben akkor van előnye a többszintű kliens-szerver architektúrának, ha az alkalmazás fejlesztése során számos szempontot állandóan szemelőtt tartunk. A kliens-szerver rendszerek fejlesztésében járatlan, legtöbbször az XBASE világból érkező fejlesztők gyakran rossz hatékonyságú, "bukásra itélt" rendszereket állítanak elő mivel ugyanazón technikákat - gyakran változtatás nélkül - ültetik át a kliens-szerver rendszerbe, amiket "korábbi életükben" megszoktak. A kliens-szerver rendszerekben a kliensoldali programok mindig csak annyi adatot válogassanak le a szerveren és töltsenek át az adatmegjelenítés (felhasználói input) helyére, amennyi feltétlenül szükséges. Az interaktív, felhasználóbarát felhasználói felületek gyakran alkalmazott eszközei a választólisták, amelyek kliens-szerver rendszerekben számos problémát vetnek fel. Ezeket az alkalmazás tervezőinek kezelni kell. Olyan szűkítő feltételt kell megadatni a felhasználóval, a választólista feltöltése előtt, amely már elegendően kevés adatsort eredményez, és így a felhasználó által is áttekinthető mennyiség. A leválogatást mindig az adatbázisszerverre kell bízni; mindenképpen kerüljük a kliens oldali keresést. Ha viszonylag statikus a választólista (kódlisták, törzsadatok), akkor célszerű a kliensgépen lokálisan tárolni a választólisták másolatait, és verziókövetéssel biztosítani a szinkronitás, amiben a triggerek nagy segítséget nyújthatnak. A képernyős adat- és a hozzájuk tartozó megnevezésmezőkbe való helyi választólistagörgetéssel jelentősen csökkenthető a listamegjelenítés - főleg a grafikus felületekre jellemző - többletideje. Az adatbázisszerverhez való fordulást a lehető legkevesebbre célszerű csökkenteni. Ha lehet egyetlen SELECT művelettel állítsuk elő a kívánt adathalmazt. Ha olyan komplex a SELECT művelet, hogy az adatbáziskezelőnk nem tudja hatékonyan optimalizálni, akkor mindenképpen használjunk tárolt eljárást. Az Oracle7, SQLBase 6 tárolt eljárásai kitűnően használhatók eljáráskurzorokként adathalmazok soronként vagy blokkonként való előállítására.

Vannak olyan feladatok, ahol az alkalmazásnak eredendően nagy adatmennyiségeket kellene átmozgatnia a hálózaton: nagyvolumenű nyomtatások, adatexport/adatimport. Ezen feladatok elvégzésére célszerű a szervergépen implementálni a feladatot, vagy ha az operációs rendszer lehetővé teszi (ilyen pl. a Windows NT) akkor magát a nyomtatási feladatot végző programot ott kell futtatni (egy MS Windows ablakban).

Tárolt eljárások alkalmazása nélküli rendszerek



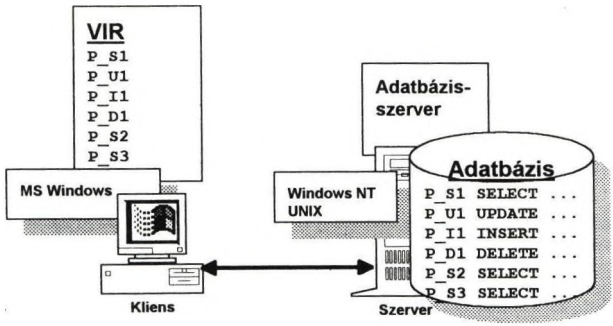
Az SQL parancsok szintaxisa gyakran (főleg a SELECT műveleteké) komplex és néha bosszantó mértékben nem hordozható egyik szerverről a másikra illetve egy bonyolultabb feladat (tranzakció) olyan sok SQL műveletet igényel, amitől annak végrehajtása komplexé válik.

Az SQL parancsok és az eredményhalmaz kliens és szerver közötti cseréjét az adott adatbáziskezelőhöz adott API-val (Application Programming Interface) lehet lebonyolítani. Minden adatbáziskezelőnek saját natív API-ja van (Oracle CI, Gupta SQLBase API, SQLServer DB-Library, stb.), amelyek működési elveiket tekintve nagymértékben de tényleges hívási szintakszisukat és szemantikájukat tekintve kevéssé hasonlítanak; az ODBC ezen a téren nagy előrelépés.

A tárolt eljárásokat nem alkalmazó alkalmazások esetén a kliens oldali (forrás)programok tartalmazzák az SQL parancsokat, és ezáltal bizonyos mértékben sértik az adatbázis szuverenitását (esetleg az integritását is), ugyanis egy komplex kapcsoló (join) SELECT-nél (UNION) pontosan kell ismerni az adatbázis belső szerkezetét.

Az igazán szuverén adatbázisok igyekszenek minél többet elfedni az adatbázis belső szerkezetéről, és csak jól definiált interfészen (VIEW-kon, tárolt parancsok/eljárások) keresztül láttatják az adatokat. A legszerencsésebb helyzet az, amikor az alkalmazások forrásprogramjai csak egyszerű SELECT, UPDATE, DELETE, INSERT ... SELECT műveleteket tartalmaznak. Az egyszerű SELECT műveletnek tekinthető a következő: SELECT ... FROM <1 db view> [WHERE ...] [ORDER BY ...].

Tárolt eljárásokat alkalmazó rendszerek



Az adatbázis szuverenitását és integritását nagymértékben megnövelhetjük tárolt eljárások és triggerek alkalmazásával. A triggereket nem támogató adatbáziskezelők esetén az adatbázis

adatintegritásának biztosítása nagyrészt a kliens-oldalon futó programok feladata, és az adatbázis csak korlátozott garanciát tud adni arra, hogy ezek a programok tényleg betartsák a szabályokat.

Tárolt eljárás

A tárolt eljárás (távoli) adatbázisban, az adott adatbáziskezelő saját (procedurális) nyelvén (Oracle PL/SQL, Sybase Transact SQL, Gupta SAL) írt, kompilált formában tárolt távoli eljárás, amit a kliens-oldali programok közvetlenül vagy közvetetten végrehajtathatnak az adatbázisszerverrel. A tárolt eljárások lehetnek statikusak vagy dinamikusak. A legtöbb adatbáziskezelő (Oracle7, Sybase) csak ***statikus tárolt eljárás***okat támogat, amelyek nem engedik meg, hogy az SQL műveleteket futásidőben állítsák össze; az SQL műveleteket tárolási fázisban már a végleges formára kell hozni. A ***dinamikus tárolt eljárás***ok (Gupta SQLBase 6) lehetővé teszik, hogy az eljárás által végrehajtott SQL parancsok (teljes egészében csak) futásidőben álljanak elő. Ezzel természetesen elvesztik azt az előnyt, amit a kompilálva való tárolás nyújt.

Trigger

A trigger olyan speciális tárolt eljárás, amelyet az adatbáziskezelő valamilyen definiált esemény hatására végrahajt. A kiváltó események lehetnek: INSERT/UPDATE/DELETE művelet előtt és/vagy után, adott időpont elérésekor vagy megadott időtartam letelte után. Az időzített triggereket némely adatbáziskezelő "event"-eknek nevezi.

A triggerek egyik leggyakoribb felhasználási területe, amikor az adatbázis adatintegritását biztosítják. A hivatkozás-integritást (referential integrity (RI)) leginkább deklaratív módon a PRIMARY KEY - FOREIGN KEY klauzulákkal célszerű megvalósítani. A WITH CHECK OPTION cikkellyel ellátott viewk SQL-sztenderd adatellenőrzést tesznek lehetővé. Az alaptáblákhoz rendelt CHECK klauzulák lehetőségei korlátozottak és a DELETE műveleteket sem lehet kontrollálni velük. Viewkkal csak oly módon lehet korlártozni a DELETE műveleteket, ha csak a törlési műveletek számára előkészített WITH CHECK OPTION-os viewkra engedélyezzük (grant-oljuk) a DELETE műveletek jogát. A triggerekben tetszőleges komplexitású ellenörzéseket helyezhetünk el, és hiba esetén egyedi hibakódot generáltathatunk. Ha egy adatbáziskezelő megfelelően támogatja a triggereket, akkor főleg ezen utóbbi ok miatt célszerű a RI előírásait kivéve valamennyi egyéb ellenörzést triggerekre bízni.

A triggerek másik fontos felhasználási területe az adatbázis teljesítménye (sebessége) növelésének céljából vagy más okból megkivánt redundáns, aggregált adatok képzése és tárolása. A VIRek vezetői információs alrendszerei különösen megkívánják a gyorsan előállított vezetői információkat, amelyek általában bővelkednek aggregált adatokban. A következő táblázat összefoglalja ezen feladatok megoldásának lehetőségeit.

Módszer leírása	***Karbantartás ideje alatti terhelés (többletidő)***	***Lekérdezés sebessége***	***Dina-mizmus***	***Elérhe-tőség***
[1] A kérés pillanatában előállított SELECT művelet, amely közvetlenül állítja elő az aggregált adatokat.	nincs	lassú	teljes	bizonyos időn belül *(1)*
[2] Időnként (éjszaka, hétvégén) a fenti módon előállított adatokat tároljuk.	nincs	gyors	részleges	nem azonnali *(2)*
[3] Triggerekkel minden tranzakció során karbantartjuk az érintett aggregátumokat.	van	gyors	részleges	azonnali

Megjegyzések:

(1) A bonyolult előállítású nagy adatbázisokra kiadott aggregált adatokat előállító SELECT-ek válaszideje gyakran igen nagy és így az adatok elérhetősége nem minden esetben gyors, gyakran több mint egy órát is kell rájuk várni.

(2) Az adatok csak a batch-ben lefuttatott aggregátumelőállító funkciók után közvetlenül, az első befolyásoló tranzakció COMMIT-álásáig aktuálisak.

A kliens-programok is megpróbálhatják a triggerek feladatát átvenni, de ez az eljárás sértené az adatbázis szuverenitását és megbizhatatlansága miatt kritikus operatív feladatok esetén feltétlenül kerülendő.

Mikor jöhetnek számításba az egyes megoldások?

[1] A módszer lassúsága miatt csak ritkán szükséges előre teljesen megjósolhatatlan (ad hoc) adatok előállítására használható. Igen fontos szempont tehát adatbáziskezelő választásnál, hogy az ad hoc lekérdezéseket milyen válaszidővel hajtja végre.

[2] Mivel a módszer fő jellemzője, hogy az előállított adatok jellemzően nem aktuálisak, napvégi, heti, havi összesítők készítésére használható.

[3] A módszer minden időpillanatban (on-line) aktuális aggregált adatokat garantál, így pl. on-line értékesítési funkciók készletszint korrekcióinak megvalósítására használható. Ha több árut akar a felhasználó diszponálni, mint ami a raktárban található akkor a tranzakciót nem engedi meg a trigger, vagy pl. egy oszlopban átadott trigger paraméter vezérlésével csak a rendelkezésreálló készletet allokálja.

Bármily meglepő egy (insert/update for each row) triggert lehet paraméterezni oly módon, hogy egy vagy több olyan oszlopot kell definiálni az alaptáblába, amelyet kifejezetten az alaptáblához kapcsolt triggerek paraméterezésére használunk, és az INSERT, UPDATE műveletek során ebbe az oszlopba tesszük az aktuális paramétert.

Hallgatói információs rendszer készítése Oracle adatbáziskezelő felhasználásával

Gulyás János , Szabó István
Miskolci Egyetem

1992 elején az egyetemen átszervezték a hallgatói nyilvántartás addigi ügyvitelét . Bizonyos feladatok a tanulmányi osztályról átkerültek a dékáni hivatalokhoz . A megnövekedett feladatok ellátásának megkönnyítésére a dékáni hivatalok segítséget kértek . Mivel létszám növelésére nem volt lehetőség , sürgős igényként merült fel a hallgatói nyilvántartás számítógépes támogatása .

Először kész rendszer vásárlásával próbálkoztunk , de nem találtunk olyan programot , amely megfelelő lett volna a dékáni hivatalok számára . Ekkor döntöttünk a saját rendszer fejlesztése mellett . Ez az elhatározás egybe esett az új adatbáziskezelési módszerekkel való megismerkedéssel . Kb egy félévvel előtte vettük meg az Oracle 6.0 adatbáziskezelőt és már létezett egy ethernet alapú hálózat , amelyben UNIX gápek voltak a szerverek . Az addigi tapasztalatok alapján is nyilvánvalóak voltak az új adatbáziskezelési technika előnyei :

- Lényegesen rövidebb fejlesztési idő
- Az új igények , módosítások gyorsabban átvezethetőek a rendszeren
- Megszüntethető ugyanazon adatok egyidejűleg több helyen történő tárolása , egységes egyetemi információs rendszer építhető ki .
- Egységes , operációs rendszertől független felület biztosítható a felhasználók számára
- Az Oracle biztonsági rendszerére alapozva felhasználóra szabott adatmódosítások és lekérdezések valósíthatóak meg

Az Oracle 6.0-ra alapozva kezdtük el a fejlesztést . Induláskor kliens - szerver konfiguráció alkalmazása mellett döntöttünk . A központi adatbázis 486-s PC-n futó SCO Unix operációs rendszer alatt működött , ezért a terminálos megoldás nagyon leterhelte volna a központi gépet .

A rendszer megvalósítása során igyekeztünk minél előbb a hivatalok rendelkezésére bocsátani használható modulokat , ezért az un. prototipus módszer alkalmazása mellett döntöttünk . Ezen cél megvalósításához készítettünk egy rendszervázlatot , ez alapján megterveztük az adatbázis strukturát és rögzítettük az ügyviteli folyamatok során végrehajtandó tevékenységeket . Ezen un. rendszerterv alapján kezdtük elkészíteni az egyes modulokat . A felhasználók tapasztalatai alapján a későbbiekben átdolgoztuk , pontosítottuk az egyes modulokat .

A rendszer használatba vétele során rögtön az összes hallgató adatát gépre akarták vitetni . Ez a tény nagymértékben lassította a rendszer használatba vételét . A másik nagy gondot az egyes karok eltérő sajátosságai jelentik .

A Hallgatói rendszer felépítése

Az adatbázis felépítése során elsődleges szempontunk volt a rugalmasság . Igyekeztünk olyan strukturát megtervezni , amellyel az összes kari változat megvalósítható a rendszer túlbonyolítása nélkül .

Egy hallgató egyedi azonosítása a következő két adat alapján lehetséges :

- azonosító , 6 jegyű szám , melynek első két számjegye jelenti a beiratkozás évét , az utolsó 4 pedig a rendszerbe felvétel sorszámát . Az utóbbi sorszám minden naptári évben ujraindul .

- a képzés sorszáma . Egy hallgató egyidejűleg több képzésben is részt vehet . Ezért az adatai két nagy logikai csoportra vannak bontva . Az adott képzéstől független , illetve a képzésre jellemző adatok . Ez a szám azt mutatja meg , hogy a hallgatónak az adott képzése milyen sorszámmal van nyilvántartva . A tanulmányi képzési sorszám szerint elkülönítve vannak eltárolva . A személyi és nyelvvizsga adatok egységesek, mert függetlenek a képzéstől .

Ez a két adat már elégséges a hallgató egyedi azonosításához . Azonban még nagyon lényeges adat a hallgató tanulmányi állapota . Ez mutatja meg , hogy a hallgató hogyan áll a tanulmányaival az adott pillanatban . Például a következő tanulmányi állapotokat lehet megkülönböztetni :

- 1 Felvételt nyert . Ezek a hallgatók felvételt nyertek ugyan , de nem iratkoztak még be egyetlen félévre sem .

- 2 Félévre beiratkozott . Ezen hallgatók egy adott félévre beiratkoztak és azt még nem zárták le .

- 3 Félévet ismétel sikertelen vizsgázás esetén . Mint ahogyan az előbbiekből kiderül ezzel az állapotkóddal azok a hallgatók rendelkeznek , akiknek vizsgáikat nem sikerült eredményesen letenni .

- 4 Félévét lezárta . Ezzel az állapotkóddal rendelkező hallgató félévét sikeresen befejezte .

- 6 Évkihagyás félév végzése után . Ha a hallgató a félév sikeres lezárása után évhalasztást kap ilyen állapotkóddal rendelkezik .

- 0 Évhalasztás . Felvett hallgatók kérhetnek évhalasztást tanulmányaik megkezdése előtt

- 7 Féléve érvénytelen . Ezen hallgatók tanulmányaikat felév közben abbahagyták , a vizsgákat vagy el sem kezdték , vagy pedig félbehagyták .

- 8 Feltételesen lezárva . Ezek a hallgatók előző félévi vizsgáikat a következő félév kezdetéig nem fejezték be és halasztást kaptak . Az aktuális félévre a 8-s kóddal rendelkezők beírathatóak , de ez a félévük nem zárható le 4-s kóddal mindaddig , amíg az előző félév nincs normálisan lezárva . Utolsó félév esetében ez a kód nem használható .

- 9 Tanulmányait befejezte . Ezek a hallgatók az utolsó tanulmányi félévet is sikeresen befejezték , de még nyilvántartásban vannak valamilyen okból . Pl csak abszolútóriummal rendelkeznek .

A Hallgatói nyilvántartási rendszerben a következő tevékenységcsoportok választhatók szét :

a) Hallgatók tanulmányaihoz kapcsolódó adatok nyilvántartása

b) Ösztöndíjak számfejtése

c) Tanszéki információs rendszer . (Jegyek felvitele , lekérdezése)

a) Hallgatók tanulmányi adatainak nyilvántartása

Miután a hallgató felvételt nyert adatai a hallgatók felvétele menüpont segítségével bekerülnek a nyilvántartási rendszerbe . A felvett hallgatókat két különböző csoportba sorolhatjuk . Többségében a hallgatóknak ez az első képzésük , amelyre felvételt nyertek . Ekkor a hallgató összes adatát be kell vinni a rendszerbe . A másik lényegesen kisebb csoportot azok a hallgatók alkotják , akiknek ez már nem az első szakjuk , amelyre felvételt nyertek . Ezeket a hallgatókat a párhuzamos képzésre felvétel menüpont segítségével kell felvenni a rendszerbe . Ekkor a hallgató már meglévő azonosítójával lesz továbbra is nyilvántartva a

rendszerben , de az újabb szakjához tartozó tanulmányi adatok az előzőektől különböző képzési sorszámmal lesznek nyilvántartva . A felvett hallgatók tanulmányi állapota 1-s , azaz 'felvételt nyert' -re állítódik be .

Az országos felvételi program adatait (ELTE számítógépes szolgálata által készített) a rendszer át tudja venni , ilyen módon a személyi adatok nagyrészét nem kell kézzel bevinni .

A hallgatókat minden félév kezdetekor be kell iratni . Az új félévre történő beiratás a hallgató aktuális tanulmányi állapota alapján történik. A beiratás az adott félévre tanulmányi állapotuk alapján beirathatóak listája segítségével végezhető el. Az új félévre beiratáskor problémát okozhatnak azok a hallgatók akik az előző félévi vizsgák letételére halasztást kaptak és ezért még nincsenek lezárva, de már az új félévre is be kellene iratni őket . Ezen ellentmondás feloldására lett bevezetve a 8 **'Féléve feltételesen lezárva'** állapotkód . Ha ezen hallgatók előző félévét 8-s kóddal lezárjuk , az új félévre beirathatóak , de nyilvántartjuk azt is , hogy az előző félévük még nincs befejezve . Ha a hallgatónak előző félévét mégsem sikerül sikeresen befejeznie , akkor az előző félévét évismétléssel le kell zárni , majd az aktuális felévre vonatkozó beiratkozását törölni kell .

A beiratás során a hallgató féléve automatikusan meghatározott, igy ez a mezö normál beiratás során nem módosítható. De ha a felhasználó az **ADMINISZTRATOR** csoportba tartozik, akkor ő felülirhatja a hallgató aktuális félévét .

A beiratás után következő lépés a tantárgyak hozzárendelése a hallgatókhoz . Ezen lépés előtt azon karokon , ahol modulrendszerű képzés van , meg lehet adni a választott modulokat , ha az adott félévben indulnak a modulok a szakirányon . Ezután következhet a tantárgyak hallgatókhoz rendelése . Ezen összerendelés alapvetően két különböző módon végezhető el . Egyik lehetőség a tanterv szerinti tantárgy hozzárendelés . Ekkor az eltárolt tantervek alapján a hallgató szakja , szakiránya és választott moduljai segítségével kijelölhető , hogy valamely hallgató milyen tantárgyakat fog választani . A másik lehetőség a tantárgycsoportok alkalmazása .

A rendszer lehetőséget biztosít tantárgycsoportok definiálására . Az így létrehozott csoportokat hallgatókhoz hozzá lehet rendelni . Ez az összerendelés egyidejüleg több hallgató esetében is elvégezhető . A hallgatókhoz csoportosan hozzárendelt tantárgyak egyénenként módosíthatóak . Az egyedi módosítás során van lehetőség a tantárgyi felmentések beírására is . A felmentések két tipusát lehet megkülönböztetni . Jeggyel történő felmentés esetében a hallgató előző félévi jegye bele fog számítani az aktuális félévi átlagába . A másik lehetőség a jegy nélküli felmentés . Ekkor az adott tárgyból a hallgatónak nem lesz jegye .

A következő lépés az értékelések felvitele , karbantartása . A felvitel megkönnyítésére több különböző módszer rendelkezésre áll . Pl. fel lehet vinni tantárgyanként külön az aláírásokat , gyakorlati- és vizsgajegyeket , de lehetőség van az összes értékelés felvitelére egyidejűleg egy hallgató részére .

A jegyek felvitelét gyorsíthatja a tanszéki jegyfelvitel , amelynek segítségével a jegyek felvitelének hatalmas munkája megoszlik a tanszékek és a dékáni hivatalok között . Az utolsó mozzanat a hallgató félévének lezárása . A lezárás történhet átlagszámítással és átlag számítása nélkül . Átlagszámítással való lezárást azon hallgatók esetében célszerű választani , akiknek az állapotkódját tanulmányi eredményeik alapján akarjuk meghatározni . Az átlagszámítás nélküli félévzárást csak rendkívüli esetekben ajánlatos használni . Pl. félév érvénytelenítése stb . Külön menüpont segítségével lehetséges az évhalasztás , évkihagyás bejegyzése . Ekkor lehetséges a visszatérés félévének a bejegyzése is .

Új tanulmányi félév nyitása csak az adminisztrátor számára van engedélyezve . Ha az aktuális félévben még nincs minden hallgató lezárva valamilyen módon , akkor az új félév megnyitása nem lehetséges .

b) Ösztöndíjak számfejtése

Az ösztöndíj rendszerben a különböző juttatások , levonások alapvető jellemzői el vannak tárolva . Ezen juttatásokat csoportokba un ösztöndíj tipusokba , lehet sorolni . Az egyes ösztöndíj tipusokon belül tovább finomíthatóak az egyes jogcímek jellemzői .

A számfejtés megkezdéséhez először is ki kell alakítani a kódrendszert . Ez jelenti a juttatások illetve levonások felsorolását illetve a jellemzőik megadását . Pl Juttatás vagy levonás , előleg fizethető-e ezen a jogcímen , ha igen, mennyi az előleg havi rögzített összege , mennyi a rendszeres havi juttatás/levonás fix összege, ha nem változik hallgatónként . A juttatások beírása után lehetőség van ösztöndíj csoportok kialakítására . Mindenegyes csoporthoz a juttatások valamilyen körét lehet hozzárendelni és ki lehet jelölni egy olyan táblázatot , amelyből kell kikeresni a juttatás/levonás alapértékét szakonként . Az így kialakított csoportok lényegesen egyszerűsíthetik a megállapított ösztöndíjak hallgatókhoz rendelését . Az egyes juttatás/levonás kódokhoz tartozó alapértékek szakonként is megadhatóak , ha nem egységesek egyetemi szinten . Az egyes juttatások esetében először a program megnézi meg van-e adva alapérték szakonként , ha nincs akkor az egész egyetemre érvényes alapértéket veszi . Természetesen az alapértékek megadásának csak akkor van értelme , ha nem táblázatot használunk a juttatás/levonás értékének a meghatározására . Jelenleg csak a tanulmányi ösztöndíj értéke határozható meg táblázat segítségével . Itt ugyanis az összeg nem csak a szaktól függ , hanem a tanulmányi átlagtól is .

Ha a kódrendszer kialakítása megtörtént , kezdődhet a számfejtés előkészítése . Ez jelenti a megállapított ösztöndíj értékek hallgatókhoz rendelését . Ha egy hallgatót valamilyen ösztöndíjkategóriába besorolunk , az adott kategóriába tartozó juttatások illetve levonások automatikusan feljönnek alapértékeikkel együtt . Azok a juttatások , amelyeket ilyen módon a hallgatóhoz rendelünk , minden hónapban számfejtődnek , azaz ezek alkotják a rendszeres juttatások körét .

A számfejtés szempontjából négy esetet különböztethetünk meg .

- előleg fizetése . Csak olyan jogcímeken lehetséges kifizetés , amelyeknél a kódtáblában szerepel , hogy előleg fizethető . Az előleg fizetése megelőzheti a megállapított ösztöndíj értékek hallgatóhoz rendelését .
- előleg utáni ösztöndíj fizetése . Ha az előző hónapban előleg volt fizetve , akkor az adott hónapban számfejteni kell az előleg és a tényleges összeg különbségét . Az adott hónapra számfejtett összegek értéke rendszeres juttatás esetében csak a korrekció mezőbe írt értékkel módosítható . A rendszeres juttatásokon kívül még számfejthetőek eseti juttatások is .
- kétszeres ösztöndíj számfejtése . Ha nincs előleg fizetés , akkor lehet a kétszeres ösztöndíjat számfejteni ezen menüpont segítségével .
- normál ösztöndíj fizetése . Lényegében megegyezik az előző pontban leírtakkal , azzal a különbséggel , hogy nem kerül számfejtésre az előleg és a rendszeres juttatás különbsége .

A különböző jellegű ösztöndíjak számfejtését nem lehet keverni egy hónapon belül .

c) Tanszéki információs rendszer . (Jegyek felvitele , lekérdezése)

A tanszéki információs rendszer célja az , hogy tehermentesítse a dékáni hivatalokat a jegyek beírása alól . A tanszékek beírhatják azon hallgatók értékelését , akik a tanszék tárgyait hallgatják . Az előbbieken kívül a HG szolgáltatása a tanszékek felé , hogy lekérdezhetik a hozzájuk tartozó hallgatók tanulmányi jellegű adatait és ezekről listákat is készíthetnek .

Az elkészült rendszer eddigi tapasztalataink alapján megfelelően rugalmas . Az adatstruktura lényegi változtatására nem volt szükség . A rendszert eddigi tapasztalatok alapján az Oracle 7-re a következő hónapokban akarjuk átvinni , ami a táblaszintű adatok újragondolását , az adatok ellenőrzésének un . constraint -ekbe foglalását jelenti . Megfontoljuk az adatbázis szintű triggerek alkalmazását is .

Az adatok bevitelének , lekérdezésének egyszerűsítésére igyekszünk felhasználni az új változat megnövekedett képességeit . Pl az adatok lekérdezésének egyszerűsítésére újabb Oracle segédeszközöket használunk fel (Browser) .

Célunk egy olyan általános tanulmányi nyilvántartási rendszer kialakítására , amely megfelel lényeges módosítások nélkül a hazai egyetemeken , főiskolákon .

TÚL A RELÁCIÓS ADATKEZELÉSEN

Balogh Judit - Juhász István
(Kossuth Lajos Tudományegyetem)

A relációs adatbázis rendszerek napjainkban alapvető szerepet játszanak az adatbáziskezelés területén. Köszönhető ez elsősorban annak, hogy az adatokat ezek igen szemléletesen, a hétköznapi életben is használt táblázatokban kezelik. A relációs adatbáziskezelőket többek között a nagyfokú adatfüggetlenség, az on-line tranzakciókezelés, az automatikus kérdés optimalizálás, a megosztottan használható adatokon való sokrétű nézetdefiniálás, a magasszintű halmazorientált lekérdező nyelv, az SQL használata, több befogadó nyelv támogatása, negyedik generációs nyelvek beépítettsége jellemzi. Ezek az eszközök hatékony alkalmazások megírását teszik lehetővé.

Az elmúlt évtized azonban igen kemény kihívásokkal szembesítette a relációs adatbázis rendszereket. A "hagyományos" adattípusok (numerikus, szöveges, dátum, stb.) mellett egyre inkább előtérbe kerültek az új adattípusok (hang, kép, grafika, stb.), olyan alkalmazások esetén mint a mérnöki tervezés, multimédia, orvosi rendszerek, térinformatika. Ezen alkalmazások objektumai nagyméretűek, szerkezetük gyakran igen bonyolult, szemantikailag igen összetettek és a velük végezhető műveletek is komplexek. Ezen alkalmazásokhoz a relációs adatbáziskezelők eszközei már kevésnek bizonyultak. A probléma gyökere abban van, hogy a relációs adatmodell absztrakciós szintje alacsony. Gondoljunk arra, hogy a táblázatoknak harmadik normálformában kell lenniük és a táblázat elemei csak atomi értékek lehetnek.

Nagyjából szintén az elmúlt évtizedre esik az is, hogy az adatbáziskezelés területén is megjelent az objektumorientált filozófia, létrejöttek az első objektumorientált adatbázis rendszerek. Ezek igen magas absztrakciós szinttel rendelkeznek és hatékony alkalmazásfejlesztési eszközöket tartalmaznak. Az új igények hatására a relációs adatbáziskezelők (pl. ORACLE, INGRES, Sybase, stb.) új verzióiba olyan eszközök kerültek, amelyek határozottan objektumorientált jellemzőkkel rendelkeznek. Másrészt létrejött a relációs adatbáziskezelők egy új generációja (pl. POSTGRES), amely a relációs modellt egy magasabb absztrakciós szinten valósítják meg.

A továbbiakban az ISO és az ANSI SQL szabványa fejlesztés alatt álló verziójának, az úgynevezett SQL3-nak néhány olyan eszközéről és fogalmáról szólunk, melyek az objektumorientáltságot viszik be az SQL-be. Igyekeztünk a szabványtervezet terminológiájához igazodni és szabatosan, definíciószerűen a

fogalmakra (esetleg az újraértelmezett fogalmakra) kitérni. A szintaktikus elemek tárgyalása messze túlmutat jelenlegi lehetőségeinken.

Definíciók

- **absztrakt adattípus**: Hasonló objektumok - melyeket az adott absztrakt adattípus példányainak nevezünk - viselkedésének specifikálása.
  attribútum: Egy absztrakt adattípus egy tárolt komponense, amelynek saját neve és típusa van.
- **állapot**: Egy absztrakt adattípus egy példánya komponenseinek rendezett sorozata.
- **komponens** (absztrakt adattípusé): A reprezentáció egy eleme.
- **konstrukciós művelet**: Olyan művelet, amely létrehozza egy absztrakt adattípus egy példányát mint SQL-adatot.
- **lebontó (destructor) függvény**: Olyan függény, amely törli egy absztrakt adattípus egy példányát reprezentáló SQL-adatot.
- **bezárás**: Egy absztrakt adattípus bizonyos tagjainak láthatóságát szabályozó mechanizmus.
- **implementáció** (absztrakt adattípusé): Egy adott absztrakt adattípushoz kapcsolódó műveletek egy része olyan adatok segítségével realizálható, melyek SQL-adatokként vannak tárolva, más műveletek viszont végrehajtható kódként értelmezhetőek, mely kód egy objektumon végrehajtható manipulációkat implementál. Egy absztrakt adattípus implementációja tehát az adatstruktúra a tárolt adatokkal és a viselkedést leíró kód együttese.
- **interfész** (absztrakt adattípusé): Egy absztrakt adatípus viselkedésmódja, a be nem zárt attributumok és műveleti nevek együttese.
- **tag**: Egy abszrakt adattípus attributuma vagy művelete.
- **módosító (mutator) függvény**: Olyan függvény, amelynek megadva egy absztrakt adattípus példányt és egy értéket, megváltoztatja a példány állapotát.
- **lekérdező (observer) függvény**: Olyan függvény, amely nem változtatja meg egy absztrakt adattípus egy adott példányának az állapotát, hanem valamely attributumának értékével tér vissza.
- **reprezentáció** (absztrakt adattípusé): Egy absztrakt adattípus attributumainak rendezett sorozata.
- **sor típus**: Egy tábla egy sorának adattípusa, (oszlop név, adattípus) párok sorozata.
- **perzisztens**: Korlátlan ideig létezik, megszüntetéséről külön kell gondoskodni.

Adattípusok

Az adattípus ábrázolható értékek együttese. Egy érték logikai reprezentációja a literál. A fizikai reprezentáció implementációfüggő.

Az SQL3 kétféle adatípust támogat: **előredefiniált** (vagy **beépített**) és **absztrakt** adattípust. Az absztrakt adattípusnak nincs literál alakja, tehát csak fizikai reprezentációja létezik. Absztrakt adattípust definiálhat a szabvány, egy implementáció, egy alkalmazás.

Új primitív vagy absztrakt típus az ún. *distinct* típus segítségével hozható létre. Ez egy olyan új típus, amely megőrzi azon típus (az ún. *forrás* típus) reprezentációját, amelyből származik, viszont attól különbözik és nem hasonlítható vele össze. Kezelésére a felhasználó saját függvényeket definiál, ezek a függények csak ezen a típuson működnek, másra nem alkalmazhatók.

Az SQL3 az **összetett** típusokat is támogatja. Egy összetett típus elemei lehetnek primitív típusúak, absztrakt adattípus elemek, vagy összetett típusúak.

Az értékek **null** vagy **nem-null** értékek lehetnek. Egy null érték egy implementációfüggő speciális érték, amely különbözik az adott típus minden nem-null értékétől. Ténylegesen csak egyetlen null érték létezik, s ez az érték minden SQL típusnak tagja. Literál alakja nincs, a NULL kulcsszó jelzi, hogy az adott helyen null érték szerepel.

Előredefiniált típusok

Az SQL3 a következő primitív típusokat definiálja:

CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT, BINARY LARGE OBJECT, BIT, BIT VARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, ENUMERATED, FLOAT, REAL, DOUBLE PRECISION, BOOLEAN, DATE, TIME, TIME STAMP, INTERVAL.

Minden adattípus rendelkezik egy **típusleíró**val. Ez tartalmazza az adott típus azonosítását és minden olyan információt, amely a típus egy példányának jellemzéséhez szükséges.

A primitív típusok közül itt most csak egyet emelünk ki. A BINARY LARGE OBJECT az a típus, amely nagyméretű, strukturálatlan objektumokat tud kezelni táblaelemekként. Ezen típus elemi oktett sorozatokból állnak. Ezzel kezelhetünk például multimédiás adatokat.

Absztrakt adattípusok

Egy absztrakt adattípus definiálásánál meg kell adni az attributumait, melyek az értéket reprezentálják, meg kell adni azokat az operátorokat, melyek az értékek rendezési relációit határozzák meg és végül meg kell határozni azokat a műveleteket, melyek az absztrakt adattípus viselkedését implementálják.

Egy absztrakt adattípus definiálása mindig egy CREATE utasítás segítségével történik.

Egy Ti absztrakt adattípus értéke összehasonlítható egy Tj adattípus értékével, ha mindketten ugyanazon altípus családhoz tartoznak.

Egy absztrakt adattípus minden komponensére külön meg kell határozni a **bezárási szint**, amely lehet **public**, **private**, **protected**. A public komponensek alkotják az absztrakt adattípus interfész részét és láthatók minden, az adott absztrakt típushoz hozzáférési joggal rendelkező felhasználó számára. A private komponensek viszont csak az adott adattípus definíciójának belsejében láthatók, külvilág számára totálisan elrejtettek. A protected komponensek az adott absztrakt adattípus altípusai számára is láthatók.

A bezárás az absztrakt adattípus definíciójához kötődik és független attól, hogy a felhasználó milyen jogosultságokkal rendelkezik az absztrakt adattípusok létrehozásához illetve használatához.

Az absztrakt adattípus definiálásakor minden attributumhoz automatikusan létrejön egy lekérdező és egy módosító függvény, melyeknek a neve megegyezik az attributum nevével és bezárási szintjük is azonos az attributum bezárási szintjével.

A lekérdező függvényeknek egy argumentuma van, ennek típusa megegyezik az adott absztrakt adattípussal. A függvény visszaadja az attributum értékét.

A módosító függvénynek két argumentuma van, az első típusa megegyezik az adott absztrakt adattípussal, a másiké az adott attributuméval. A függvény az adott attributum értékét a második argumentum értékére változtatja és ezt az értéket adja vissza.

Egy absztrakt adattípus definiálásával megadhatunk **szupertípus**okat és akkor a definiált absztrakt adattípus ezek közvetlen **altípus**a lesz, az SQL3 tehát a többszörös öröklődés elvét valósítja meg. Az altípusban természetesen az attributum neveknek egyedieknek kell lenniük, ezért a névütközés feloldására az absztrakt altípus definiálásánál a szupertípusok attributumai átnevezhetők.

Az öröklődésnél a közvetlen szupertípus attributumainak reprezentációja változatlanul kerül át az altípusba. Az altípusban azonban tetszőleges függvények definiálhatók, akár ugyanazzal a névvel mint a szupertípusban. Tehát az SQL3 a módszer polimorfizmust valósítja meg.

Együttes (collection) típusok

Az **együttes** (az ISO definíciója szerint) nem más mint értékekből vagy objektumokból, mint elemekből álló multihalmaz. Az elemek ugyanazzal a típussal rendelkeznek.

Elemek egy együttese megadható egyrészt úgy, mint egy általános együttes, amelyet tovább nem specializálunk; másrészt megadható mint egy halmaz, egy multihalmaz vagy egy lista. Egy általános együttes nem példányosítható, de megadható vele bármely azonos típusú halmaz, multihalmaz, vagy lista eleme.

Az együttes egy érték.

Halmaz típus: A halmaz különböző elemek egy rendezetlen együttese.

Egy T típusú elemeket tartalmazó halmazt a SET(T) típussal adhatunk meg. Az üres halmaz típusa: SET( ).

Két halmazérték összehasonlítható, ha elemtípusuk azonos, vagy ha egyikük, vagy mindkettőjük típusa SET( ).

Két halmaz egyenlő, ha ugyanazokat az elemeket tartalmazzák és ezen elemek egyike sem nem-null érték. Két halmaz egyenlősége ismeretlen, ha null értékeket tartalmaznak és ezen null értékek megfelelő nem-null értékekkel való helyettesítése egyenlővé tenné a két halmazt. Egyébként a két halmaz nem egyenlő.

Bármely T típus esetén a T típusú elemek halmaza közvetlen altípusa a T típusú elemek általános együttesének.

Multihalmaz típus: A multihalmaz a halmazokhoz hasonló együttes típus, kivéve, hogy benne azonos elemek előfordulhatnak.

Egy T típusú multihalmazt tartalmazó multihalmazt a MULTISET(T) típussal adhatunk meg. Az üres multihalmaz típusa: MULTISET( ).

A halmaznál tett további megjegyzések a multihalmazra is érvényesek.

Lista típus: A lista egy olyan multihalmaz, amelyben az elemek rendezettek. Így, amikor egy új elemet viszünk a listába, akkor annak a helyét is meg kell adnunk.

A T típusú elemeket tartalmazó listát a LIST(T) típussal adhatunk meg. Az üres lista típusa: LIST( ).

A halmaznál tett további megjegyzések a listára is érvényesek.

Operátorok

Egy operátort egy vagy több értékre alkalmazva újabb értéket kapunk. Az operátorok vagy **előredefiniált standard operátor**ok vagy pedig **felhasználó által definiált operátor**ok. Ha egy előredefiniált standard operátort egy előredefiniált típusú értékre alkalmazunk, akkor mindig egy előredefiniált műveletet hajt végre. Ha egy előredefiniált standard operátort absztrakt adattípushoz tartozó értékre alkalmazunk, vagy egy felhasználó által definiált operátort primitív vagy absztrakt típusú értékre alkalmazunk, akkor implicit módon mindig egy rutinhívás következik be.

Oszlopok

Az oszlop értékeknek egy időben változó multihalmaza. Egy oszlop egy értéke a legkisebb olyan adategység, amelyet módosítani és leválogatni lehet. Ha a tábla egy absztrakt adattípus példányainak együttese, akkor a tábla egy oszlopa az adott absztrakt adattípus egy attributuma értékeinek multihalmaz és neve megegyezik az attributum nevével.

Táblák

Egy tábla soroknak az együttese. A sor egy sortípus egy példánya. A sor a legkisebb adategység, amelyet be lehet illeszteni egy táblába illetve ki lehet törölni belőle.

Mintahogy minden együttes vagy halmaz vagy multihalmaz, vagy lista, ugyanúgy minden tábla lehet halmaz tábla, vagy multihalmaz tábla, vagy lista tábla.

Egy halmaz tábla nem tartalmazhat megegyező sorokat, egy lista tábla sorai pedig rendezettek.

Egy tábla vagy **alaptábla**, vagy **származtatott tábla**.

A származtatott tábla közvetlenül vagy közvetve egy vagy több táblából keletkezik egy lekérdezés eredményeképpen.

A **nézettábla** egy külön definíciós utasításban megnevezett származtatott tábla.

Egy nézettábla lehet **állandó** és **ideiglenes** nézettábla.

Sémák

Az SQL-séma egy olyan perzisztens objektum, amelynek összetevői:

- név,
- tulajdonos,
- karakterkészlet,
- rutinok,
- komponensek (pl. típus, tábla, nézettábla, megszorítás, stb. definíciók).

Rutinok

A felhasználó akár a sémákhoz kapcsolódóan akár az absztrakt adattípusok viselkedésének realizálásához saját rutinokat hozhat létre. SQL utasítások segítségével, vagy valamely SQL3 által támogatott befogadó nyelven (Pascal, C, PLI, Ada, MUMPS, FORTRAN, COBOL). Egy rutin lehet eljárás vagy függvény. A paraméterek száma tetszőleges. A formális paraméter lista szintaktikája és szemantikája Ada-szerű.

Forrás:

ISO-ANSI Working Draft-Database Language SQL (SQL3) (August 1994).

A beágyazott relációs adatmodell reprezentálása relációs adatbáziskezelő rendszerek felhasználásával[1]

Kovács György
(ELTE, Általános Számítástudomány Tanszék
Budapest, Múzeum körút 6-8
email: gykovacs@ullman.elte.hu, kgy@ilab.sztaki.hu)

Hajas Csilla
(KLTE, Matematikai és Informatikai Intézet
Debrecen, Pf.12, 4010
email: hajas@palma1.elte.hu, hajas@math.klte.hu)

1. Bevezetés

Világszerte intenzív kutatás és fejlesztés folyik az összetettebb relációtipusokra épülő adatkezelés irányában. Jelenleg egy ELTE - SZTAKI közös OMFB pályázat keretében folyik kutatás-fejlesztési projekt, amelynek célja a relációs adatmodell három legígéretesebbnek bizonyuló kiterjesztésének (multi- nested- és fuzzy relációs adatmodell) megvalósítása létező relációs adatbáziskezelő rendszerek felhasználásával. A projekt végső célja olyan negyedik generációs fejlesztői környezet kialakítása, amely lehetővé teszi a kibővített adatmodellekre épülő hatékony alkalmazásfejlesztést.
A három kiterjesztett adatmodell megvalósítása egymástól függetlenül történik, a továbbiakban csak a beágyazott modellel kapcsolatos eddigi eredményeket, problémákat tárgyaljuk. A megvalósítás során az alábbi főbb feladatokat kell megoldani:

- A beágyazott relációs adatmodellben értelmezett relációk reprezentálása klasszikus értelemben vett relációk segítségével.
- A beágyazott relációs modellhez alkalmas lekérdező nyelv lerögzítése. Ez egyrészt relációs algebra és kalkulus megadását, másrészt az SQL relációs lekérdező nyelv megfelelő kiterjesztését jelenti.
- A kibővített SQL nyelvhez fordító készítése. A fordítás valamilyen hosztnyelvbe (pl. C) beágyazott SQL-re történik a megadott reprezentáció alapján.
- A negyedik generációs környezet kialakítása.

Egy ilyen rendszer implementálása lehetővé teszi, hogy adatbázisok tervezése során a fizikai adatmodellek fizikai szintre való leképezésekor a beágyazott relációs adatmodellt használjuk fizikai szintű adatmodellként (lásd [14]).
Jelen cikkünkben először a beágyazott relációs adatmodell lényegét foglaljuk össze röviden (2. fejezet). A modellhez tartozó matematikailag letisztázott elmélet megtalálható az ide vonatkozó irodalomban ([1]-[2], [4]-[9], [11]-[13]). A 3. fejezetben ismertetjük a beágyazott relációk közönséges relációkkal történő reprezentálását, majd a 4. fejezetben a beágyazott relációk kezelésére alkalmas SQL kiterjesztéssel (NF2-SQL) foglalkozunk. Megjegyezzük, hogy a definiált nyelv nem teljes, a már definiált elemek szintaxisa a függelékben található. A megvalósítás további feladataival ebben a cikkben nem foglalkozunk. Az 5. fejezetben ejtünk szót a fordítás menetéről.

[1] Ez a munka az OTKA-2149, az OTKA-T-014250, illetve az OTKA 2561 pályázatok támogatásával készült.

2. A beágyazott relációs adatmodellről

A beágyazott relációs adatmodell a klasszikus relációs adatmodell olyan kiterjesztése, amely nem-első normálformájú relációkkal foglalkozik, azaz feloldja az értékek atomi jellegére tett megszorítást.
A beágyazott modell relációit egymásba ágyazott táblákkal szemléltetjük. Egy tábla egy attribútuma lehet atomi vagy összetett (reláció-értékű, tábla-értékű). Egy atomi tulajdonság értékei a megengedett értéktartomány (domain) elemei, egy összetett tulajdonság értékei pedig relációk (táblák). A táblák egymásba való beágyazásával a beágyazott relációs adatmodell struktúrált adatok leírását teszi lehetővé és csökkenti a redundanciát.
Hasonlóan a relációs modellhez, ahol az adatbázis sémát a reláció sémák felsorolásával definiáljuk, a beágyazott relációs modellben is meg kell adnunk a beágyazott relációk sémáit, mégpedig a beágyazás minden szintjén. A séma megadása attribútumok (atomi vagy összetett) véges felsorolását jelenti. A séma nem lehet rekurzív, továbbá kikötjük, hogy egy sémán belül egy attribútum-név nem szerepelhet többször. A 2.1. ábrán mutat példát beágyazott relációra.

JELÖLTEK tábla:

NÉV	ÉRDEKLŐDÉS		
	TÉMAKÖR	KEDVENC	
		MEGNEVEZÉS	OSZTÁLYZAT
'Kiss P.'	'Film'	'W.Allen'	3
		'Belmondo'	1
	'Zene'	'Mozart'	4
		'Beatles'	5

2.1. ábra. Példa beágyazott relációra.

A relációs adatmodellhez tartozó absztrakt lekérdező rendszerek a relációs algebra és relációs kalkulusok. Ehhez hasonlóan léteznek a beágyazott modellhez is a megfelelően kiterjesztett algebrai műveletek és kalkulusok (lásd irodalomjegyzék). A leginkább említésre méltó újdonság, hogy a korábbi műveletek kiterjesztése mellett két struktúra átalakító művelet bevezetése is szükséges volt. Növelhetjük (nest) illetve csökkenthetjük (unnest) a beágyazás szintjét.

3. Beágyazott relációk reprezentálása

A beágyazott relációs adatmodell bevezetése lehetővé tette összetett adatok, objektumok ábrázolását a relációs modellben már bevált táblaszemlélet segítségével. Ahhoz, hogy az így kialakított logikai táblák által reprezentált adatokat fizikai szinten is el tudjuk érni és manipulálni tudjuk, első feladatként a modell elemeinek (séma, adatok) fizikai szintű ábrázolását kell megoldani. Mivel a készülő rendszer közönséges relációkat kezelő adatbázis-kezelőkre épül, ezért a beágyazott relációkat közönséges relációkkal fogjuk ábrázolni. A közönséges relációkkal való reprezentáció többféleképpen történhet. Rendszerünkben két különböző reprezentációt valósítunk meg (lásd 3.1 és 3.2 pontok).
A megvalósíthatóság szemponjából mindkét reprezentáció elfogadható, hatékonysági szempontból viszont jelentős különbség van a két megoldás között. Ugyanakkor nem lehet általánosan megmondani, hogy melyik a jobb reprezentáció. Bizonyos esetekben az egyik, máskor a másik reprezentáció előnyösebb, ez a beágyazott táblák tényleges tartalmától függ.

Mivel ilyen jellegű információkkal csak a felhasználó rendelkezik, a kívánt reprezentációt a táblák definiálásakor lehet majd megadni erre bevezetett opció segítségével.

3.1 Érték szerinti reprezentáció

A beágyazott relációk tábla-értékeit (belső tábláit) külön táblákként tároljuk. A befoglaló táblákban ezek után a belső táblák helyett csak azokra vonatkozó hivatkozásokat tárolunk (a belső tábla neve), amelyek már elemi (karakterlánc) típusból vett értékek. Így külső szinten a reláció-értékű attribútumokból karakterlánc típusú atomi attribútumokat csinálunk, melyeknek értékei táblanevek lesznek.
A nested modellben a beágyazás tetszőleges mélységű lehet, azaz a külön táblákként tárolt belső táblák maguk is tartalmazhatnak tábla-értékeket. A cél az, hogy csak közönséges tábláink legyenek, ezért a leírt módszert alkalmazzuk az összes olyan belső táblára, amely maga is tartalmaz beágyazást.
Ezzel a módszerrel tehát egy beágyazott táblát közönséges táblák halmazával ábrázoljuk. Egy közönséges tábla egy tábla-értéket reprezentál. Ennek a következménye, hogy az ábrázoló közönséges táblák száma nemcsak a sémától, hanem a beágyazott táblák aktuális tartalmától is függ. Dinamikusan kell tehát új táblákat kreálni illetve táblákat megszüntetni.

3.2 Oszlop szerinti reprezentáció

Egy beágyazott reláció belső tábláira igaz az, hogy az azonos oszlopban szereplő belső tábláknak ugyanaz a struktúrája. Ezt figyelembe véve, az előző pontban leírt reprezentációnak egy értelmes módosítása, hogy az egy oszlopba tartozó összetett értékeket ábrázoló közönséges táblákat összevonjuk. Így egyetlen közönséges táblával a beágyazott táblának nem egy tábla-értékét, hanem egy egész oszlopát ábrázoljuk.
Így azonban elveszne az az információ, hogy az összevont reprezentáló tábla egyes sorai a reprezentált oszlopon belül mely tábla-érték részei, és az a tábla-érték a beágyazott tábla mely sorához tartozott. Ezt a két egymással összefüggő kérdést úgy oldjuk meg, hogy a külső tábla minden sorához hozzáveszünk egy sorazonosítót (DOWN oszlop), valamint a belső táblákat ábrázoló közönséges táblák minden sorához hozzávesszük annak a (külső) sornak a sorazonosítóját, amelyikhez tartozik (UP oszlop). Ezek után a reprezentáló közönséges táblára vonatkozó, sorazonosító szerinti szelekciókkal kapjuk meg az összetartozó sorokat. A sorazonosítókat a rendszer automatikusan generálja, mégpedig úgy, hogy egy sorazonosító az egész adatbázison belül egyedi lesz.
Az összetett attribútumot a tartalmazó táblát reprezentáló közönséges táblában is megtartjuk ugyanolyan névvel, típusa LOGICAL lesz. Ha van belső tábla egy adott sorban, akkor a reprezentáló táblában TRUE szerepel, ha nincs, akkor NULL. Így tudjuk megkülönböztetni azt, hogy ha egy sorhoz tartozó belső táblát visszaadó szelekció (lásd fentebb) eredménye üres, akkor az azt jelenti-e, hogy ott nincs táblaérték (NULL eset), vagy pedig 0 sort tartalmazó belső tábla szerepel.
Ha a belső táblák maguk is tartalmaznak összetett attribútumokat, akkor azok reprezentálására ugyanezt az módszert alkalmazzuk. Az ilyen belső táblák ábrázolásakor a felfelé, vagyis a tartalmazó sorra vonatkozó hivatkozások mellett saját sorazonosítók hozzávételére is szükség van, amelyekre a tartalmazott belső táblák sorai hivatkoznak majd a reprezentáló táblában.
Az oszlop szerinti ábrázolási módban a reprezentációhoz szükséges közönséges táblák száma csak a beágyazott reláció sémájától függ, vagyis ez a reprezentáció statikus természetű. A táblák száma a sémában szereplő összetett attribútumok száma + 1.

3.3 Metaadatok tárolása

Mindkét reprezentáció esetén tárolni kell az eredeti beágyazott relációk sémáit, valamint a reprezentáló közönséges táblák közti kapcsolatokat. Tárolásuk külön relációs táblában történik, és a szabvány SQL-en keresztül kezeljük őket. A metaadatokat mindkét reprezentáció esetén ugyanolyan szerkezetű katalógustáblában tároljuk, a táblák tartalma lesz csak kissé különböző. A katalógustábla szerkezete a következő:

Oszlop	Típus	Leírás
TABLE_NAME	CHAR(32)	Beágyazott adatbázistáblának a neve, amelyben az egyes attribútumok szerepelnek. Ha az ATTR_NAME oszlopban táblanév áll, akkor itt NULL szerepel.
PARENT_ATTR	CHAR(32)	Minden (atomi vagy összetett) attribútumhoz annak az összetett attribútumnak a neve, amely őt a definiált séma szerint tartalmazza. Ez külső szintű attribútumok esetén a tábla neve lesz. Táblanév esetén NULL.
ATTR_NAME	CHAR(32)	Attribútumnevek. Itt szerepelnek mind az atomi, mind a összetett attribútumok, sőt még az adatbázistábla-nevek is.
TYPE_TNAME	CHAR(32)	Atomi attribútum esetén a megfelelő oszlop típusa. Összetett attribútum esetén itt vagy a TABLE kulcsszó áll (érték szerinti reprezentáció), vagy a reprezentáló közönséges tábla fizikai neve (oszlop szerinti reprezentáció).
COUNTER	INTEGER	Egy attribútum esetén ez a sorszám mondja meg azt, hogy az őt tartalmazó összetett attribútumon (PARENT_ATTR) belül hányadikként volt definiálva. Táblanév esetén NULL.

3.1. ábra. A katalógus (NESTED_TABLES) szerkezete.

A beágyazott táblákra vonatkozó műveletek során a katalógust gyakran kell lekérdezni bizonyos ellenőrzések elvégzése miatt, valamint számos művelet esetén módosítani is kell a katalógus tartalmát.

3.4 Egy példa a reprezentációra

Vegyük példának a 2.1. ábrán látható beágyazott táblát. Tegyük fel továbbá, hogy az ÉRDEKLŐDÉS tábla-értékű oszlopot oszlop szerint, míg a KEDVENC oszlopot érték szerint reprezentáljuk. A beágyazott táblát reprezentáló közönséges táblák halmazát a 3.2. és 3.3. ábra mutatja. A 3.2. ábra a tényleges adatokat reprezentáló táblákat tartalmazza, a 3.3. ábra pedig az ábrázolt beágyazott relációnak megfelelő bejegyzéseket mutatja a katalógusban.

JELÖLTEK tábla:

NÉV	ÉRDEKLŐDÉS	DOWN
'Kiss P.'	TRUE	'sor1'

ÉRDEKLŐDÉS tábla:

TÉMAKÖR	KEDVENC	UP
'Film'	'tábla_1'	'sor1'
'Zene'	'tábla_2'	'sor1'

tábla_1 tábla:

MEGNEVEZÉS	OSZTÁLYZAT
'W. Allen'	3
'Belmondo'	1

tábla_2 tábla:

MEGNEVEZÉS	OSZTÁLYZAT
'Mozart'	4
'Beatles'	5

3.2. ábra. Példa reprezentáció.

NESTED_TABLES tábla:

TABLE_NAME	PARENT_ATTR	ATTR_NAME	TYPE_TNAME	COUNTER
NULL	NULL	'JELÖLTEK'	'JELÖLTEK'	NULL
'JELÖLTEK'	'JELÖLTEK'	'NÉV'	'CHAR(20)'	1
'JELÖLTEK'	'JELÖLTEK'	'ÉRDEKLŐDÉS'	'ÉRDEKLŐDÉS'	2
'JELÖLTEK'	'ÉRDEKLŐDÉS'	'TÉMAKÖR'	'CHAR(10)'	1
'JELÖLTEK'	'ÉRDEKLŐDÉS'	'KEDVENC'	'TABLE'	2
'JELÖLTEK'	'KEDVENC'	'MEGNEVEZÉS'	'CHAR(10)'	1
'JELÖLTEK'	'KEDVENC'	'OSZTÁLYZAT'	'INTEGER'	2

3.3. ábra. Példa katalógus.

4. Az NF2-SQL nyelv

A relációs adatmodell legelterjedtebb lekérdező nyelve az SQL, mely a relációs algebrán és kalkuluson alapul. A meglévő szabvány SQL-t terjesztjük ki oly módon, hogy az beágyazott relációk kezelésére is alkalmas legyen és a szabvány SQL nyelv elemei a kiterjesztett nyelvben is kifejezhetőek legyenek. A kiterjesztett nyelvre NF2-SQL (Non First Normal Form SQL) néven fogunk hivatkozni.

A szabvány SQL-hez hasonlóan az NF2-SQL utasításai is három csoportba oszthatók: adatdefiníciós, adatmanipulációs és lekérdező utasítások. A továbbiakban csak az adatdefiníciós és adatmanipulációs utasításokkal foglalkozunk. Az egyes utasítások szintaxisát függelék tartalmazza.

3.1. Az adatdefiníciós nyelv (DDL).

Az NF2-SQL adatdefiníciós nyelve a beágyazott relációk sémáinak definiálására és megváltoztatására illetve megszüntetésére szolgáló utasításokat tartalmazza. Három utasítás tartozik ide, a CREATE TABLE, DROP TABLE ill. ALTER TABLE.

A CREATE TABLE utasítás megenged tetszőleges számú és mélységű beágyazást, azaz atomi értékű attribútumok mellett reláció-értékű attribútumok definiálását. Kétféle lehet, egyrészt definiálhatunk táblát az attribútumok és típusaik megadásával, másrészt már létező táblákra vonatkozó lekérdezéssel.
A beágyazási lehetőség miatt bevezetünk egy új típust, a tábla-értékek típusát. Az új típust a TABLE kulcsszó jelöli. Az ily módon definiált belső táblákat azonban ki kell fejteni, meg kell adni a belső sémákat is.
A 3. pontban már említettük, hogy a beágyazott relációk közönséges relációkkal való reprezentálása kétféleképpen történhet, egy tábla értékű oszlopot reprezentálhatunk érték szerint vagy oszlop szerint. Ez az oszlopok definiálásakor a felhasználó írhatja elő a TABLE BY VALUES illetve a TABLE [BY COLUMN] opciók segítségével. Alapértelmezés szerint az összetett oszlopok oszlop szerint vannak reprezentálva.(lásd Függelék).

A DROP TABLE a táblák törlésére szolgáló utasítás. Szintaktikailag megegyezik a szabvány SQL szintaktikával.

A táblák szerkezetének megváltoztatására szolgáló utasítás az ALTER TABLE. Különböző konkrét adatbáziskezelők táblák szerkezetének megváltoztatását különböző mértékben engedik meg. Azt mondjuk, hogy mi a konkrét megvalósítás szintjén döntjük mindig el, hogy az NF2-SQL ALTER TABLE utasítását milyen szintig valósítjuk meg. Legbővebb esetben táblához hozzáadhatunk új oszlopokat, törölhetünk oszlopokat, valamint megváltoztathatjuk oszlopok típusát.

4.2. Adatmanipulációs nyelv (DML)

A klasszikus relációs modellben nincsenek egy táblának belső táblái, ezért az adatmanipulációs műveletek mindig az adatbázis táblákra vonatkoznak. A beágyazott modellben az összetett értékek maguk is táblák, és mindhárom DML utasítás vonatkozhat belső táblára is. A cél, hogy a beágyazott résztáblákat a lehető legkönnyebben tudjuk manipulálni, hozzáadni (INSERT), módosítani (UPDATE) és törölni (DELETE) sorokat.
A nyelv szintjén ez azt jelenti, hogy az INSERT, UPDATE és DELETE utasításban a táblanév helyén reláció-értékű attribútum név is állhat. Nyilván ez a bővítés önmagában nem elég, hiszen a tábla-értékű attribútum megadása nem határozza meg, hogy melyik sorban lévő belső tábla(k)ra vonatkozik a művelet.
Még alapvetőbb probléma, hogy az sem derül ki, hogy mely adatbázis reláció belső tábláiról van szó. Ezek a problémák azonban megoldhatók anélkül, hogy további bővítéseket, bonyolításokat vezetnénk be nyelvünkben. Igaz ugyanis, hogy függetlenül attól, hogy milyen műveletet hajtunk végre egy belső táblán, az mindig egy módosítást jelent a belső táblát tartalmazó adatbázis táblán. Így a belső táblákra vonatkozó bármilyen műveletet (INSERT, UPDATE, DELETE) a tartalmazó adatbázis táblára vonatkozó UPDATE művelettel hajtjuk végre.
SQL-ben egy (atomi) attribútum változtatásakor új értékként tetszőleges megfelelő típusú kifejezés használató. NF2-SQL-ben az ennek megfelelő kiterjesztés abból áll, hogy ott ahol a

módosítandó attribútum egy tábla-érték, megengedünk olyan tetszőleges kifejezést, aminek éredménye egy megfelelő típusú reláció.
Tovább nem részletezzük ezt a részt, a három adatmanipulációs utasítás pontos szintaxisa megtalálható a függelékben.

4.3. Lekérdezések.

Az NF2-SQL lekérdező nyelve a kiterjesztett SELECT utasításból és két újonnan bevezetett utasításból, a NEST és UNNEST utasításokból fog állni. E két új utasítás a beágyazott táblák átstruktúrálására szolgál majd. Az NF2-SQL lekérdező nyelvét itt nem részletezzük, annak pontos kidolgozása folyamatban van. Megjegyezzük még, hogy az NF2-SQL már definiált elemei is változhatnak a lekérdező nyelv kidolgozása során.

5. Szabvány SQL-re fordító programok kidolgozása.

Miután a beágyazott relációk kezelésére definiáltuk az SQL-szerű nyelvet (NF2-SQL), a következő feladat a megadott nyelvhez fordító program készítése. A fordítás alapja a beágyazott adatmodell reprezentálásának módja. A reprezentáció mindkét esetben közönséges relációkkal történik, melyek kezelése SQL-en keresztül valósul meg, így a fordító az NF2-SQL programjait hoszt nyelvbe beágyazott SQL-re fordítja.
Egy NF2-SQL program NF2-SQL utasításokból áll, amelyek beágyazott relációkat definiálnak, lekérdeznek, módosítanak. Minden NF2-SQL utasítás a beágyazott modell szerint értelmezett relációkra vonatkozó műveletet jelent. A tényleges műveleteket azonban a reprezentáló közönséges táblákon kell végrehajtani SQL utasításokon keresztül. A fordító program feladata, hogy meghatározza azt, hogy egy NF2-SQL művelet mely reprezentáló táblákat érinti és hogyan, és gondoskodnia kell azok végrehajtásáról.
Ennek elvégzéséhez a fordítónak szüksége van azokra a belső adatokra (pl. az eredeti beágyazott séma), amelyeket az eredeti illetve a reprezentáló táblákról közönséges relációk formájában eltároltunk. A fordító program további feladata a szükséges ellenőrzések (pl. séma nem lehet rekurzív) elvégzése és hiba esetén hibaüzenet küldése. A fordítóprogram készítése a jól ismert módszereken alapján történik.
Most példákon keresztül szemléltetjük, hogy milyen SQL utasításokat eredményeznek egyes NF2-SQL utasítás végrehajtások. Az SQL utasítások hoszt nyelvbe való beágyazásától eltekintünk a példáinkban.

Vegyük a 2.1. ábrán található táblát. Hozzuk először létre a megfelelő üres táblát, majd töltsük fel az ábrán látható tartalommal. NF2-SQL-ben az alábbi utasításokat kell végrehajtani:

A kreáló utasítás NF2-SQL-ben:

```
CREATE TABLE JELÖLTEK
      ( NÉV                  CHAR(20),
        ÉRDEKLŐDÉS           TABLE
            ( TÉMAKÖR        CHAR(10),
              KEDVENC        TABLE BY VALUES
                  ( MEGNEVEZÉS      CHAR(10),
                    OSZTÁLYZAT      INTEGER ) ) );
```

A reprezentáció szintjén végrehajtandó SQL utasítások:

```
CREATE TABLE JELÖLTEK
        ( NÉV                   CHAR(20),
          ÉRDEKLŐDÉS            LOGICAL,
          DOWN                  BYTE(16) );
CREATE TABLE ÉRDEKLŐDÉS
                ( UP            BYTE(16)
                  TÉMAKÖR       CHAR(10),
                  KEDVENC       CHAR(32) );
INSERT INTO NESTED_TABLES
        VALUES (NULL, NULL, 'JELÖLTEK', 'JELÖLTEK', NULL);
INSERT INTO NESTED_TABLES
        VALUES ('JELÖLTEK', 'JELÖLTEK', 'NÉV', 'CHAR(20)', 1);
INSERT INTO NESTED_TABLES
        VALUES ('JELÖLTEK', 'JELÖLTEK', 'ÉRDEKLŐDÉS', 'ÉRDEKLŐDÉS', 2);
INSERT INTO NESTED_TABLES
        VALUES ('JELÖLTEK', 'ÉRDEKLŐDÉS', 'TÉMAKÖR', 'CHAR(10)', 1);
INSERT INTO NESTED_TABLES
        VALUES ('JELÖLTEK', 'ÉRDEKLŐDÉS', 'KEDVENC', 'TABLE', 2);
INSERT INTO NESTED_TABLES
        VALUES ('JELÖLTEK', 'KEDVENC', 'MEGNEVEZÉS', 'CHAR(10)', 1);
INSERT INTO NESTED_TABLES
        VALUES ('JELÖLTEK', 'KEDVENC', 'OSZTÁLYZAT', 'INTEGER', 2);
```

A táblát feltöltő NF2-SQL utasítás:

```
INSERT  INTO  JELÖLTEK
        VALUES ( ('Kiss P.', (  ('Film',     ( ('W.Allen', 3) ('Belmondo', 1) ))
                                ('Zene', ( ('Mozart', 4) ('Beatles', 5) ))))) ;
```

A reprezentáció szintjén végrehajtandó SQL utasítások:

```
CREATE TABLE tábla_1
        (MEGNEVEZÉS             CHAR(10),
         OSZTÁLYZAT             INTEGER);
INSERT INTO tábla_1
        VALUES ('W.Allen', 3);
INSERT INTO tábla_1
        VALUES ('Belmondo', 1);
CREATE TABLE tábla_2
        (MEGNEVEZÉS             CHAR(10),
         OSZTÁLYZAT INTEGER);
INSERT INTO tábla_2
        VALUES ('Mozart', 4);
INSERT INTO tábla_2
        VALUES 'Beatles', 5);
INSERT INTO JELÖLTEK
        VALUES ('Kiss P.', TRUE, 'sor1');
INSERT INTO ÉRDEKLŐDÉS (TÉMAKÖR, KEDVENC, UP)
        VALUES ('Film', 'tábla_1', 'sor1');
INSERT INTO ÉRDEKLŐDÉS (TÉMAKÖR, KEDVENC, UP)
        VALUES ('Zene', 'tábla_2', 'sor1');
```

A tábla_1, tábla_2 és sor1 a rendszer által automatikusan generált értékek.

Függelék. Az NF2-SQL BNF-szerű leírása

```
<NF2-SQL stmt> ::-    <ddl stmt> | <dml stmt> | <query stmt>
<query stmt> ::       /* Under developement */
<ddl stmt> ::-        <create table stmt> | <drop table stmt> | <alter table stmt>
<create table stmt> ::- CREATE TABLE <table name> <table def>
<table name> ::-      < valid  SQL table name >
<table def> ::-       <column defs> | <create query>
<column defs> ::-     (<column spec list>)
<column spec list> ::- <column spec> | <column spec> , <column spec list>
<column spec> ::-     <atomic spec> | <structured spec>
<atomic spec> ::-     <column name> <atomic type> [<null option>]
<column name> ::-     < valid  SQL column name >
<atomic type> ::-     integer | float | ... /* SQL data types */
<null option> ::-     WITH NULL | NOT NULL
<structured spec> ::- <column name> TABLE [<kind of repr>] [<null option>]
                          <column defs>
<kind of repr> ::-    BY VALUES | BY COLUMN
<create query> ::-    [<scheme>] AS <query stmt>
<scheme> ::-          (<column list>)
<column list> ::-     <column name> [<scheme>]
                      | <column name> [<scheme>] , <column list>]
<drop table stmt> ::- DROP TABLE <table name>
<alter table stmt> ::- ALTER TABLE <table name> <alter option>
<alter option> ::-    <add columns> | <drop columns> | <modify columns>
<add columns> ::-     ADD [TO <column name>] <column defs>
<drop columns> ::-    DROP (<column list>)
<column list> ::-     <column name> | <column name> , <column list>
<modify columns> ::-  MODIFY (<atomic spec list>)
<atomic spec list> ::- <atomic spec> | <atomic spec> , <atomic spec list>
<dml stmt> ::-        <insert stmt> | <delete stmt> | <update stmt>
<insert stmt> ::-     INSERT INTO <dml object> [<scheme>] <to be inserted>
<dml object> ::-      <table name> | <column name>
<to be inserted> ::-  <insert values> | <query stmt>
<insert values> ::-   VALUES { <table constant> | <sor> }
<table constant> ::-  (<sor list>)
<sor list> ::-        <sor> | <sor> <sor list>
<sor> ::-             (<query expr list>)
<query expr list> ::- <query expr> | <query expr> , <query expr list>
<query expr> ::-      < SQL constant expression > | <table name>
                      | <table constant> | (<query stmt>)
<delete stmt> ::-     DELETE FROM <dml object> [<alias>]
                          [WHERE <search condition>]
<search condition> ::- /* Under developement */
<update stmt> ::-     UPDATE <dml object> [<alias>]
                          [FROM <table list>]
                          SET <assigment list>
                          [WHERE <search condition>]
<table list> ::-      <table spec> | <table spec> ,<table list>
<table spec> ::-      <table name> [<alias>]
<alias> ::-           < valid SQL correlation name >
<assigment list> ::-  <assigment> | <assigment> ,<assigment list>
<assigment> ::-       <column name> = < dml expr>
<dml expr> ::-        <query expr> | (<dml stmt>)
```

Irodalomjegyzék

[1] BENCZÚR A., HAJAS C., KISS A., KOCSIS A., KOVÁCS G., MÁRKUS T., NIKOVITS T. Negyedik generációs szoftverfejlesztési környezet kidolgozása relációs adatbáziskezelő rendszerek felhasználói felületének kibővítésével. ELTE TTK Ált.Szám.Tud.Tanszék, 1994, három kutatás-fejlesztési jelentés:
1. Megvalósíthatósági tanulmány.
2. A multi-, nested és fuzzy relációs adatmodell sémaleíró nyelveinek kidolgozása.
3. A multi-, nested és fuzzy relációs adatmodell relációs algebra és relációs kalkulus kidolgozása.

[2] BENCZÚR, A., AMER, K., and HAJAS, C. An extension of Datalog rules for nested relations. Techn.Report 1993, 88.sz.

[3] DEPPISH, A., PAUL, H.B., and SCHEK, H.J. A storage system for complex objects. In Proceedings of the International Workshop on Object-Oriented Database Systems (Pacific Grove, Calif.), 1986, pp.183-195.

[4] GYSSENS, M., PAREDAENS, J., and VAN GUCHT, D. A uniform approach towards handlind atomic and structured information in the nested relational database. J.ACM Vol.36, No.4 (Oct.1989), pp.790-825.

[5] HAJAS, Cs. Informatikai kutatások a beágyazott adatbázis területén. Informatikai a felsőoktatásban országos konferencia, Debreceni Universitas, 1993.

[6] HERNANDEZ, H.J. Extended nesred relations. Acta Inform. Vol.30, 1993, pp.741-771.

[7] LEVENE, M. and LOIZOU, G. The nested universal relational data model, JCSS. Vol.49, No.3, 1994, pp.683-717.

[8] KORTH, H.F. and ROTH, M.A. Query langueges for nested relational databases. LNCS. No.361., pp.190-204.

[9] PAREDAENS, J. and VAN GUCHT, D. Converting nested algebra expressions into flat algebra expressions. ACM Trans.Datab.Syst.Vol.17, No.1 (March.1992), pp.65-93.

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

[11] ROTH, M.A., KORTH, H.F., and SILBERSCHATZ, A. Extended algebra and calculus for nested relational databases. ACM Trans.Datab.Syst.Vol.13, No.4 (Dec.1988), pp.389-417.

[12] SCHEK, H.J., SCHOLL, M.H. The relational model with relation-valued attributes Information Systems, Vol.11, No.2, 1986, pp.137-147.

[13] THOMAS, S.J., FISHER, P.C. Nested relational structures. The Theory of Databases. P.C.Kanellakis, Ed.JAI Press, Greenwich, 1986, pp.269-307.

[14] Van BOMMEL, P., KOVÁCS, Gy. and MICSIK, A. Transformation of database populations and operations from the conceptual to the internal level. Inform.Systems, Vol.19, No.2. pp175-191., 1994.

Fuzzy relációs adatmodell reprezentálása relációs adatbázis-kezelő rendszerek felhasználásával[1]

Nikovits Tibor
(ELTE Általános Számítástudományi Tanszék
Budapest, Múzeum krt. 6-8.
e-mail: nikovits@ullman.elte.hu)

Dr. Kiss Attila
(ELTE Általános Számítástudományi Tanszék
Budapest, Múzeum krt. 6-8.
e-mail: kiss@palma1.elte.hu)

Achs Ágnes
(Pollack Mihály Műszaki Főiskola
Pécs
e-mail: achs@pmmf.hu)

[1]Ez a munka az OMFB 94-97-67-054 illetve az OTKA-2149 pályázatok támogatásával készült.

1. Bevezetés

Világszerte intenzív kutatás folyik az összetettebb adatmodellekre épülő adatkezelés irányában. Jelenleg egy ELTE - SZTAKI közös OMFB pályázat keretében folyik kutatás és fejlesztés, amelynek célja a relációs adatmodell három legígéretesebbnek tűnő kiterjesztésének (multi- nested- és fuzzy relációs adatmodell) megvalósítása létező relációs adatbáziskezelő rendszerek felhasználásával. A három új adatmodellt egymástól függetlenül szeretnénk létrehozni. A projekt végső célja olyan negyedik generációs fejlesztői környezet kialakítása, amely lehetővé teszi a kibővített adatmodellekre épülő hatékony alkalmazásfejlesztést. Jelen cikk e munka eddigi eredményeit és további terveit írja le a fuzzy adatmodellre vonatkozóan.

2. Előzmények

Az utóbbi időben a szakértői rendszerek, mesterséges intelligencia jellegű alkalmazások használata során felmerült az igény, hogy a felhasználók a természetes nyelvekhez hasonló módon kommunikálhassanak a rendszerekkel. Ennek során a felhasználó olyan információkat is közölhet a rendszerrel, hogy "egy adott személy fiatal", és olyan kérdéseket is feltehet, hogy "kik laknak Budapest közelében". Az ehhez hasonló bizonytalan információk kezelésére az egyik legelterjedtebb elmélet a fuzzy halmazok elmélete (Zadeh [8]). A fuzzy halmaz a halmaz karakterisztikus függvényének az általánosítása, azaz az alaphalmazról a [0,1]-be képező függvény, amelynél a függvényérték egy x helyen azt fejezi ki, hogy az x mennyire tekinthető a halmazba tartozó pontnak. Ezeket a beletartozási értékeket a gyakorlatban szakértők határozhatják meg a tapasztalataik alapján, szubjektív módon, vagyis a megfelelő fuzzy halmazok kiválasztása a szakértő(k) tudását, véleményét tükrözi.
A fuzzy halmazok kutatása kiterjed a matematika szinte minden területére. Az információs rendszerek szempontjából döntő fontosságú az első rendű logika kiterjesztése. A fuzzy logika ma már jól kidolgozott része a fuzzy halmazok elméletének (Novak [5], Li-Liu [4]), így biztos alapot jelent különböző működő gyakorlati alkalmazások kialakításában, például berendezések, műszerek vezérlésében. A japán ipari technológiák egyre több helyen használják a fuzzy halmazok legfrissebb kutatási eredményeit (videokamera, robotok).
A bizonytalan információt kezelő rendszer két szempontból tekinthető a klasszikus információs rendszer kibővítésének:

bizonytalan, pontatlan, határozatlan vagy hiányos információ tárolására is képes,

nem pontosan feltett, bizonytalan, szubjektív fogalmakat használó kérdésekre is tud választ adni.

A fenti követelményeknek eleget tevő rendszer készítéséhez először az adatmodellt kell megválasztani a hozzátartozó lekérdező nyelvvel. A fuzzy halmazok tárolásával kapcsolatos problémák kutatása, különböző fuzzy adatbázis modellek vizsgálata a 80-as évek elején kezdődött (Buckles-Petry [1], Zemankova [9], Umano [6]). Ezekben a fuzzy adatmodellekben az a közös vonás, hogy valamilyen formában a gyakorlatban leginkább bevált Codd-féle klasszikus relációs adatmodellt általánosítják, vagyis a táblák, relációk maradnak az adatmodell alapelemei. Az általánosítás történhet az attribútumokhoz tartozó speciális adattípusok bevezetésével. Például a táblázatban szereplő értékek lehetnek halmaz típusúak (Buckles [1]), fuzzy halmaz típusúak (Vila [7]). Másik lehetőség, hogy a tábla soraiban szereplő értékek közti

kapcsolat szorosságát, bizonytalanságát jelezhetjük a sorhoz rendelt [0,1]-be eső számmal, vagy a [0,1] egy fuzzy halmazával. Az így kapott fuzzy relációk matematikai értelemben jól kezelhetők (Kiss [3]), de gyakorlati szerepük kisebb, mint az előbb említett modelleknek, mivel nehezebben értelmezhetők. Jelentőségüket az adja, hogy a lekérdezések során részeredményként [0,1]-beli számokkal súlyozott sorokat kaphatunk, amelyeket fuzzy relációnak tekinthetünk, és a fuzzy relációkra kidolgozott eszközöket használhatjuk a kezelésükre. Szokás még az adattípusok kiterjesztéséből és a sorok súlyozásából származtatott kevert modellt is vizsgálni (Umano [6]). Olyan kevert modellt is tanulmányoznak, ahol nem a sorokhoz rendelnek súlyokat, hanem minden egyes sorban szereplő érték kap egy kompatibilitási értéket (Vila [7]).

A fuzzy halmazok elméletében az általánosításokat mindig úgy kell definiálni, hogy közönséges halmazok esetén az eredeti fogalmat, tulajdonságot kapjuk vissza. Általános alapelvként használt a kiterjesztési elv (Novak [5]), amely azt mondja meg, hogy közönséges függvénybe fuzzy halmazt helyettesítve milyen fuzzy halmaz legyen a függvényérték. Ha X és Y fuzzy halmazok, akkor az XΘY elemi összehasonlítás értéke a kiterjesztési elv alapján a {hamis,igaz} ({0,1}) halmaznak fuzzy halmaza, vagyis az igaz és hamis érték egy-egy súlyt kap.
Ahhoz, hogy a műveletek ne vezessenek ki a modellből vagy azt a kevert modellt kell választani, ahol a sorokhoz a [0,1] egy fuzzy halmazát rendeljük, vagy a sorhoz tartozó fuzzy halmaz alapján kell eldönteni, hogy a sor beletartozik-e a kiválasztás eredményébe vagy nem. Ezt leggyakrabban úgy döntik el, hogy az "igaz"-hoz tartozó súly nagyobb-e egy előre megadott küszöbszintnél.

A különböző modellekben az elemi összehasonlító operátorokat ki szokták egészíteni logikai értékekre vagy [0,1]-be képező függvényekkel. Például bevezethetjük a körülbelül egyenlőség mérésére a következő függvényt:

$$kbegyenlő(x,y) = \begin{cases} 1 & , \text{ ha } |x-y| < 0.1 \\ 0.9 & , \text{ ha } 0.1 \le |x-y| < 0.2 \\ 0 & , \text{ különben} \end{cases}$$

Ha X és Y fuzzy halmazok a számegyenesen, akkor kbegyenlő(X,Y) a kiterjesztési elv alapján a {0, 0.9, 1} fuzzy halmaza lesz.

A logikai kifejezésekben gyakran módosítókat is használhatunk, úgymint "nagyon magas", "elég drága". A módosítók $[0,1] \rightarrow [0,1]$ típusú függvények.

Egy Fuzzy adatmodellen alapuló adatbázis-kezelő rendszer készítésénél természetes elvárás, hogy a rendszer egy relációs adatbázis-kezelő rendszerre épüljön, annak szolgáltatásait használja fel. Ez azt jelenti, hogy a fuzzy adatmodell kiválasztása után a fuzzy adatbázist és minden hozzá kapcsolódó információt közönséges táblákkal kell reprezentálnunk. Hasonlóan a fuzzy lekérdezéseket is közönséges lekérdező nyelven, esetleg programozási nyelvet is felhasználva valósítjuk meg.
A fentiekben vázolt lehetőségeket és problémákat figyelembe véve a fuzzy adatbázis-kezelő rendszer kidolgozása során a következő feladatokat kell megoldanunk.

1. Fuzzy adatmodell megadása.
2. Fuzzy adatmodell reprezentálása közönséges relációk segítségével.

3. Fuzzy lekérdező nyelv megadása.
4. Fuzzy lekérdezések megvalósítása szabvány SQL segítségével.
5. 4GL fejlesztői környezet kialakítása.

A továbbiakban az általunk választott fuzzy adatmodellt és annak reprezentálását ismertetjük.

3. Fuzzy relációs adatmodell

A bevezetésben leírt lehetőségek közül azt választottuk ki, amely elég komplex, de a felhasználók számára mégis könnyen érthető. Ez az a modell, amelyben a tábla soraihoz nem rendelünk értéket vagy fuzzy halmazt, viszont a sorokban szereplő értékek fuzzy halmazok is lehetnek.
Ez a fuzzy relációs adatmodell a klasszikus relációs adatmodellnek olyan kiterjesztése, amelyben a relációk oszlopainak típusa a megszokott típusokon kívül úgynevezett fuzzy típusú is lehet. Ezekben az oszlopokban tudunk bizonytalan információt tárolni. Az ilyen típusú oszlopokban szereplő értékek valamilyen előre megadott alakú fuzzy halmazok lehetnek. Mint látni fogjuk a megszokott egész, valós, dátum és karakteres adatok a fuzzy halmazok speciális esetének tekinthetők, így egy fuzzy típusú oszlopban egyszerre lehetnek közönséges értékek és fuzzy halmazok. A kezelhetőség érdekében csak speciális alakú - véges sok paraméterrel leírható - fuzzy halmazok előfordulása megengedett. A modellben lehetőség van névvel ellátott fuzzy konstansok továbbá olyan módosító operátorok és hasonlósági relációk definiálására, amelyek segítségével a fuzzy oszlopok és az új objektumok kezelhetők.

Egy relációban a megengedett adattípusok tehát a következők:

A szabvány SQL adattípusai:

(Valós, Egész, Dátum, Karakteres ...stb.)

Az SQL adattípusok fuzzy változatai:

(Fuzzy valós, Fuzzy egész, Fuzzy dátum, Fuzzy karakteres ...stb.)

A különböző típusú oszlopokban az alábbi értékek szerepelhetnek:

A nem fuzzy oszlopokban a szokásos értékek megengedettek.

A fuzzy oszlopokban az oszlop alaptípusától függően a következő speciális fuzzy halmazok megengedettek:

1. Folytonos alaphalmazon megadott trapéz alakú fuzzy halmaz.
2. Folytonos alaphalmazon megadott lineáris függvényekkel (poligonnal) megadható fuzzy halmaz.
3. Tetszőleges alaphalmazon megadott véges sok diszkrét értékkel megadható fuzzy halmaz.

Folytonos alaphalmaznak tekintjük az egész, valós és dátum típusú alaphalmazokat.

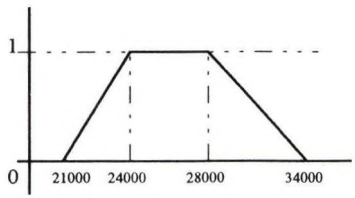
A trapéz alakú fuzzy halmazok a poligonok speciális esetének tekinthetők, hogy mégis külön típusként definiáljuk őket annak a következő az oka. Az alkalmazásokban legtöbbször elegendő a trapéz alakú fuzzy halmazok használata és mivel ezek előre rögzített számú paraméterrel leírhatóak (pontosan 4 paraméterrel) ezért lényegesen egyszerűbben és gyorsabban kezelhetők az adatbáziskezelő számára. A felhasználó így eldöntheti, hogy megelégszik a trapéz nyújtotta lehetőségekkel vagy egy speciálisabb fuzzy halmazt kíván-e megadni.

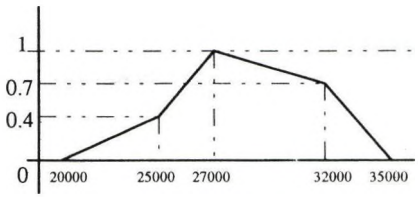
Példa:

Tároljuk egy DOLGOZÓK nevű táblában a dolgozók nevét fizetését életkorát és nyelvismeretét. A név oszlop típusa legyen karakteres, tehát itt semmiféle fuzzy információt nem fogunk tárolni. A fizetés oszlop legyen fuzzy egész, vagyis itt a következő értékek adhatók meg:

fizetés = 28000
fizetés = { 1/25000, 0.9/26000, 0.8/27000 }
fizetés = trapéz(21000, 24000, 28000, 34000)
fizetés = lineáris(0/20000, 0.4/25000, 1/27000, 0.7/32000, 0/35000)

Az utóbbi két értéket az alábbi ábrák szemléltetik:





Az életkor legyen szintén fuzzy egész, végül a nyelvismeret oszlop típusa legyen fuzzy karakteres.
Ebben az oszlopban megadható a következő információ:

nyelvismeret = { 1/"Angol", 0.6/"Német", 0.3/"Francia" }

A tábla egy lehetséges előfordulása:

Név	Fizetés	Kor	Nyelvismeret
Kovács Géza	28000	42	{1/Angol, 0.4/Svéd, 1/Japán}
Szabó Miklós	{ 1/25000, 0.9/26000, 0.8/27000 }	29	Német
Lakatos Pál	trapéz(21000, 24000, 28000, 34000)	34	{1/Orosz, 1/Spanyol}
Bagaméri	lineáris(0/20000, 0.4/25000, 1/27000, 0.7/32000, 0/35000)	56	Angol

Az új típusokon kívül a modellben megadhatunk további objektumokat, amelyek részben a fuzzy értékek megadására szolgálnak, részben pedig a lekérdezésekben használhatóak. Ezek az objektumok a fuzzy konstansok, a módosítók és a hasonlósági relációk.

Fuzzy konstansok

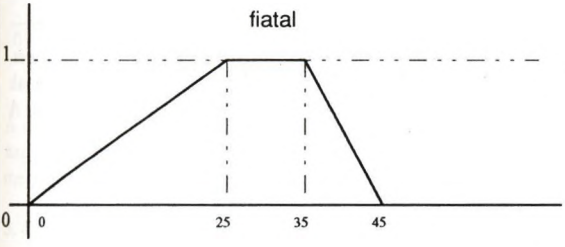
A fuzzy konstans egy olyan fuzzy halmaz, amelyet a felhasználó adhat meg, névvel láthat el és a későbbiekben a neve alapján teheti be egy sor megfelelő oszlopába. Amennyiben a fuzzy konstanst a felhasználó a későbbiekben módosítja, akkor a módosítás az összes olyan sort érinti, amelyikben a fuzzy konstans szerepel. A fuzzy konstans lehet diszkrét értékekkel megadott, trapéz alakú vagy lineáris szakaszokkal megadott. A fuzzy konstans megadásakor meg kell adni azt az alaphalmazt is, amely fölött a fuzzy konstansként megjelenő fuzzy halmaz értelmezve van. Trapéz alakú és lineáris fuzzy konstans megadható a korábban már említett, folytonos alaphalmazokon. Diszkrét értékekkel megadott fuzzy konstans alaphalmaza tetszőleges lehet. Egy fuzzy konstans csak olyan fuzzy típusú oszlopba tehető be értékként, amely oszlopnak az alaptípusa megegyezik a fuzzy konstans alaptípusával. Így tehát egy fuzzy egész oszlopba betehetünk egészek fölött értelmezett trapéz alakú, lineáris vagy diszkrét értékekkel megadott fuzzy konstanst, fuzzy valós oszlopba szintén a fenti három alakú, de valósak fölött értelmezett fuzzy konstanst, de például fuzzy karakteres oszlopba csak diszkrét értékekkel megadott fuzzy konstans tehető.

Példák fuzzy konstansra:

A következő fiatal nevű fuzzy konstans az egészek fölött van értelmezve és így a Kor oszlopba helyezhetjük el értékként.

fiatal (trapéz, egész, 0, 25, 35, 45)

A fuzzy konstanst az alábbi ábra szemlélteti:



Az alábbi tárgyalóképes nevű fuzzy konstans a karakteres alaphalmaz fölött van értelmezve és így a Nyelvismeret oszlopba tehető.

tárgyalóképes (diszkrét, karakteres, 1/"Angol", 1/"Német", 1/"Francia")

Hasonlósági relációk

A hasonlósági relációk segítségével azt adhatja meg a felhasználó, hogy az adatbázisban szereplő két érték mennyire tekinthető hasonlónak. A két érték típusának azonosnak kell

lennie, a hasonlóság mértéke pedig egy 0 és 1 közé eső számmal adható meg. A hasonlósági reláció létrehozásakor meg kell adni, hogy az milyen típusú értékek összehasonlítására lesz alkalmazható. Ez lesz az a típus, amely fölött a hasonlósági reláció értelmezve van. Ettől a típustól függően a hasonlósági reláció a következő módok valamelyikén adható meg.
Diszkrét értékek összehasonlítása esetén a hasonlósági reláció az elempárokhoz egy 0 és 1 közé eső számot rendel. A relációtól elvárjuk, hogy szimmetrikus legyen, vagyis az (A, B) elempárhoz és a (B, A) elempárhoz ugyanazt a számot rendelje. Elvárjuk továbbá, hogy a reláció reflexív legyen abban az értelemben, hogy minden A értékre az (A, A) elempárhoz 1-et rendeljen. Az egyes elempárok hasonlósági viszonyát a felhasználó adhatja meg a fenti korlátozások figyelembevételével úgy, hogy megadja a két értéket és a hozzájuk tartozó 0 és 1 közötti számot.
Egész, valós és dátum értékek összehasonlítását úgy adhatja meg a felhasználó, hogy a hasonlóság mértékét (vagyis egy 0 és 1 közötti számot) az értékek eltérésének függvényében adja meg. Ez a megadás egy lépcsős függvény segítségével történhet, ahol a felhasználó a lépcsős függvényt (*a*, *b*) párok formájában adhatja meg. Itt *a* jelöli az értékek eltérésének maximumát, *b* pedig megadja az ilyen eltéréssel rendelkező értékek hasonlóságának mértékét.

Példa:

A körülbelül egyidős embereket összehasonlíthatjuk egymással az életkoruk alapján. Azokat, akik között legfeljebb 3 év korkülönbség van hasonló korúaknak tekintünk 1 beletartozási értékkel. Ahogy nő a korkülönbség úgy egyre kevésbé mondjuk azt két emberről, hogy hasonló korú. Ezt az alábbi hasonlókorú nevű hasonlósági relációval fejezhetjük ki.

$$\text{hasonlókorú}(x,y)=\begin{cases}1 & \text{ha} |x-y| \le 3\\ 0.7 & \text{ha } 3<|x-y| \le 6\\ 0.4 & \text{ha } 6<|x-y| \le 8\\ 0 & \text{egyébként}\end{cases}$$

Diszkrét értékek összehasonlítására nézzük például a következő lehetőséget. Összehasonlíthatjuk a különböző nyelveket egymással aszerint, hogy mennyire hasonlítanak egymásra. Az alábbi nyelv_hasonló nevű hasonlósági reláció ezt az információt tartalmazza. A felhasználó a következőképpen adhatja meg a relációt:

nyelv_hasonló (diszkrét, karakteres, 0.9/{"Angol", "Német"},
0.2/{"Német", "Magyar" })

A fenti hasonlósági relációt táblázatos formában az alábbi ábra mutatja.

nyelv_hasonló	angol	német	magyar
angol	1	0.9	0
német	0.9	1	0.2
magyar	0	0.2	1

A hasonlósági reláció és a táblázat összehasonlítása után a következőket láthatjuk: Minden érték hasonló önmagához 1 beletartozási értékkel. Ha két érték hasonlóságát a felhasználó nem adta meg, akkor a két érték hasonlóságát 0-nak tekintjük.

A hasonlósági relációkat a lekérdezésekben fogjuk használni. Az imént megadott két hasonlósági relációt például az alábbi lekérdezések megválaszolásához tudjuk használni.

Kik azok akik az angolhoz hasonló nyelvet beszélnek adott a beletartozási értékkel?
Kik azok akik körülbelül 30 évesek adott a beletartozási értékkel?

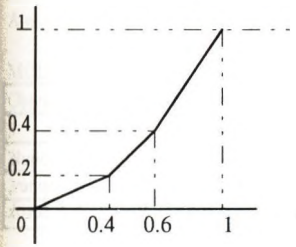
Módosítók

A módosítók [0,1] -ből [0,1] -be képező függvények, amelyek egy adott értéknek egy adott fuzzy halmazba való beletartozásának a mértékét módosíthatják. A véges reprezentálhatóság miatt csak olyan függvények adhatók meg, amelyek véges sok lineáris szakaszból állnak. (poligonok)

Példa:

Az alábbi ábrán látható "nagyon" nevű módosítót megadhatjuk a következő adatokkal:

nagyon = { 0/0, 0.2/0.4, 0.4/0.6, 1/1 }



A módosítókat szintén a lekérdezésekben használhatjuk. Lekérdezhetjük például, hogy kiknek nagyon magas a fizetése, vagy hogy kik nagyon fiatalok, és most is mindkét esetben megadhatunk egy küszöböt a beletartozási értékek számára.

3. Fuzzy adatmodell reprezentálása

A fentiekben megadott fuzzy adatmodellt a lekérdező nyelvével együtt egy közönséges relációs adatbázis-kezelőre építve valósítjuk meg. Ehhez a fuzzy adatbázis tábláit, a szereplő fuzzy halmaz definíciókat, a lekérdezésben használható módosítókat és hasonlósági relációkat közönséges táblák rendszerével fogjuk reprezentálni. Most először csak felsoroljuk azokat a táblákat, amelyekben a fuzzy objektumokra és a köztük fennálló kapcsolatokra vonatkozó információkat tárolni fogjuk, megadjuk a táblák sémáját, majd ezután részletesen leírjuk az egyes táblák szerepét. Ezeket a táblákat együttesen fuzzy meta-adatbázisnak fogjuk nevezni, a benne szereplő táblákat pedig meta-tábláknak. A meta-adatbázis táblái a következők:

FUZZY_COLUMNS

Tábla név	Oszlopnév	Oszlop_azonosító	Oszlop_típus

FUZZY_OBJECTS

Oszlop_azonosító	Objektumnév	Objektum_azonosító	Objektum_típus

MODIFIERS

Objektum_azonosító	Érték	Módosított érték

SIMILARITY_D

Objektum_azonosító	Objektum_1	Objektum_2	Érték

SIMILARITY_S

Objektum_azonosító	Különbség	Érték

TRAPEZOID

Objektum_azonosító	Érték_1	Érték_2	Érték_3	Érték_4

LINEAR

Objektum_azonosító	Érték	Beletartozás

DISCRETE

Objektum_azonosító	Érték	Beletartozás

FUZZY_COLUMNS tábla

Azt az információt, hogy mely táblák mely oszlopai fuzzy típusúak egy külön táblában tároljuk, amelynek a neve FUZZY_COLUMNS. A tábla oszlopai: tábla_név, oszlopnév, oszlop_azonosító és oszloptípus. A fenti DOLGOZÓ táblára vonatkozóan a következő sorok lesznek tárolva:

FUZZY_COLUMNS

Tábla név	Oszlopnév	Oszlop_azonosító	Oszlop_típus
DOLGOZÓ	fizetés	1	fuzzy egész
DOLGOZÓ	nyelvismeret	2	fuzzy karakteres
DOLGOZÓ	kor	3	fuzzy egész

Magában a DOLGOZÓ táblában a fuzzy oszlopokat karakteres típusú oszlopokként hozzuk létre és ott az egyes sorokban az ott értékként szereplő fuzzy objektumok azonosítóját tároljuk.

FUZZY_OBJECTS tábla

A fuzzy objektumokra vonatkozó információkat egy FUZZY_OBJECTS nevű táblában tároljuk, amelynek oszlopai: Oszlop_azonosító, Objektum_név, Objektum_azonosító, Objektum_típus.

Az oszlop_azonosító mondja meg, hogy az objektum mely oszlopokba tehető be értékként, illetve operátor esetén mely oszlopokra alkalmazható. Egy objektumhoz több oszlop is megadható. Az objektumtípus lehet trapéz, lineáris vagy diszkrét hasonlósági reláció, módosító és lineáris vagy diszkrét fuzzy halmaz.

A fenti példákban eddig használt fuzzy objektumok tárolása a táblában az alábbi:

FUZZY_OBJECTS

Oszlop_azonosító	Objektumnév	Objektum_azonosító	Objektum_típus
3	fiatal	1	trapéz
2	nyelvhasonló	2	hasonlóság_diszkrét
3	hasonlókorú	3	hasonlóság_lépcsős
3	nagyon	4	módosító
2	tárgyalóképes	5	diszkrét

MODIFIERS tábla

Ebben a táblában tároljuk a felhasználó által definiált módosítók adatait. A tábla oszlopai: Objektum_azonosító, Érték, Módosított_érték. Az érték és a módosított érték a poligonnal megadott módosító szögpontjainak a koordinátáit jelölik.

A nagyon nevű módosítót a következőképpen tároljuk:

MODIFIERS

Objektum_azonosító	Érték	Módosított érték
3	0	0
3	0.4	0.2
3	0.6	0.4
3	1	1

SIMILARITY_D tábla

Ebben a táblában tároljuk a felhasználó által definiált hasonlósági relációk adatait, amennyiben azok diszkrét értékek hasonlóságára vonatkoznak. Amikor a felhasználó létrehozza a hasonlósági relációt meg kell adnia annak típusát is, ami lehet diszkrét vagy lépcsős. Az így létrehozott diszkrét típusú hasonlósági relációk adatait tároljuk ebben a meta-táblában. A tábla oszlopai: Objektum_azonosító, Objektum_1, Objektum_2, Érték.
Az érték adja meg, hogy a két objektum mennyire hasonló egymáshoz.

A nyelv_hasonló nevű relációt a következőképpen tároljuk:

SIMILARITY_D

Objektum_azonosító	Objektum_1	Objektum_2	Érték
2	Angol	Német	0.9
2	Német	Magyar	0.2

A táblában nem szereplő objektum párok hasonlóságát 0-nak tekintjük. Egy objektum pár csak egyszer szerepelhet a táblában rögzített objektum_azonosító esetén.

SIMILARITY_S tábla

Ebben a táblában tároljuk a felhasználó által definiált hasonlósági relációk adatait, amennyiben azokat a felhasználó lépcsős függvénnyel (step function) adta meg. Ezt a felhasználó a hasonlósági reláció létrehozásakor adja meg a "lépcsős" típusmegjelöléssel. A tábla oszlopai: Objektum_azonosító, Különbség, Érték.
Két érték hasonlóságát a következőképpen kaphatjuk meg: Kiszámítjuk a két érték különbségének abszolút értékék, majd megkeressük azt a legkisebb értéket a Különbség oszlopban, amelyik ennél nagyobb vagy egyenlő. Ha találunk ilyet, akkor az adott sorban az Érték oszlopban található 0 és 1 közötti szám adja meg a két érték hasonlóságát. Ha nem találunk ilyet, akkor a hasonlóság 0 lesz. Amennyiben az összehasonlítandó értékek maguk is fuzzy halmazok, akkor a kiterjesztési elv alapján számoljuk ki a hasonlóságukat.

A hasonlókorú nevű relációt a következőképpen tároljuk:

SIMILARITY_S

Objektum_azonosító	Különbség	Érték
10	3	1
10	6	0.7
10	8	0.4

TRAPEZOID tábla

Ebben a táblában tároljuk a felhasználó által megadott trapéz alakú fuzzy halmazok 4 paraméterét. A tábla oszlopai: Objektum_azonosító, Érték_1, Érték_2, Érték_3, Érték_4. A speciális trapézokat, az S alakú fuzzy halmazokat úgy tároljuk a táblában, hogy az Érték_1 illetve Érték_4 mezőket üresen hagyjuk (NULL értéket teszünk bele)

A fiatal nevű fuzzy konstanst például a következőképpen tároljuk:

TRAPEZOID

Objektum_azonosító	Érték_1	Érték_2	Érték_3	Érték_4
1	0	25	35	45

LINEAR tábla

Ebben a táblában tároljuk a felhasználó által megadott lineáris függvényekkel leírható fuzzy halmazok paramétereit. A táblában a poligon szögpontjainak koordinátáit tároljuk hasonlóan a MODIFIERS táblához. A tábla oszlopai: Objektum_azonosító, Érték, Beletartozás.

Bagaméri úr fizetését a következő módon tároljuk:

LINEAR

Objektum_azonosító	Érték	Beletartozás
128	20000	0
128	25000	0.4
128	27000	1
128	32000	0.7
128	35000	1

Egy adott fuzzy objektumhoz tartozó Érték, Beletartozás párok ismeretében a legkisebb megadott értéknél kisebb értékekre ugyanazt a beletartozási értéket tekintjük érvényesnek, mint ami a legkisebb értékre vonatkozik, a legnagyobb megadott értéknél nagyobbakra pedig ugyanazt, mint ami a legnagyobb értékhez tartozik.

DISCRETE tábla

Ebben a táblában tároljuk a felhasználó által megadott diszkrét fuzzy halmazok paramétereit. A táblában a különböző értékeket és azok lehetőségét (beletartozási értékét) tároljuk. A tábla oszlopai: Objektum_azonosító, Érték, Beletartozás.

A tárgyalóképes nevű fuzzy halmazt a következő módon tároljuk:

DISCRETE

Objektum_azonosító	Érték	Beletartozás
129	Angol	1
129	Német	1
129	Francia	1

4. Az új modell lekérdező nyelve

A fuzzy adatmodellt relációs adatbázis-kezelő segítségével fogjuk megvalósítani, amelynek lekérdező nyelve az SQL. Ezért a lekérdező nyelvet is úgy adjuk meg, hogy az a szabványos SQL-nek olyan kiterjesztése legyen, amelyben lehetőség van az új adattípusok és új objektumok kezelésére, de benne a szabvány SQL nyelvi elemei is kifejezhetőek legyenek. A lekérdező nyelvet FSQL néven fogjuk nevezni. Az FSQL nyelv utasításai a szabvány SQL-hez hasonlóan adatdefiníciós és adatmanipulációs utasításokra oszthatók.
Az FSQL nyelv definíciója után fordító programot készítünk, amelyik az FSQL nyelvű programokat - a reprezentáció ismeretében - szabványos SQL utasításokat tartalmazó programokká alakítja. Mivel az egyes adatbázis-kezelő rendszerekben az SQL nyelv procedurális bővítései jelentősen eltérnek egymástól, ezért nem rögzítünk egyet sem ezek közül, hanem az FSQL utasításokat C-nyelvbe beágyazott SQL programra fordítjuk. Egy FSQL programból a fordító tehát egy C-be ágyazott SQL programot készít.
Az FSQL nyelv definiálása és a fordítóprogram készítése jelenleg is folyik, ezért ezeket itt most nem ismertetjük.

Irodalom

[1] Buckles B. P.-Petry F. E. (1989) A Domain Calculus for Fuzzy Relational Databases, Fuzzy Sets and Systems Vol. 29 pp. 327-340

[2] Kerre E.E.-Chen G-Vandenbulcke J (1991) A Step Towards the Theory of Fuzzy Relational Database Design, Proc. Conf. on Fuzzy Sets and Systems, Belgium

[3] Kiss A. (1991) λ-Decomposition of Fuzzy Relational Databases, Annales Univ Sci. Budapest Sect Comp. 12 pp. 133-142

[4] Li D.-Liu D. (1990) A Fuzzy PROLOG Database System, Research Study Press

[5] Novak V. (1986) Fuzzy Sets and their Applications, IOP Publishing Ltd, England

[6] Umano M. - Ezawa Y. (1991) Implementation of SQL-type Data Manipulation Language for Fuzzy Relational Databases, Proc. Conf. on Fuzzy Systems, Japan

[7] Vila A.-Medina J. M.-Pons O.-Cubero J. C.-Prados M.-Diaz J. (1993) A Knowledge Representation Model for Fuzzy Databases, Submitted to ACM Trans. on Database Dystems

[8] Zadeh L. A. (1978) Fuzzy Sets as a Basis for a Theory of Possibility, Fuzzy Sets and Systems Vol 1 pp. 3-28

[9] Zemankova-Leech M.-Kandel A. (1984) Fuzzy Relational Databases a key for Expert Systems Verla TUV Rheiland

Disk és file menedzselés nagy méretű OSF/1 (UNIX), INGRES rendszereknél

(INGRES system administrator információs utility)

Losonczi János, Országos Egészségügyi Pénztár

Az INGRES system-, és az adatbázis adminisztrátorok feladatait több INGRES utility segíti, pld. az accessdb, az infodb, az ipm, stb. Vannak azonban olyan jellegű problémák is, melyek megoldására az INGRES csak nagyon körülményesen képes. Egy sok diszkből, file system -ből, adatlokációból, adatbázisból, és adatbázis objektumból álló rendszer esetében az INGRES rendszerért és az adatbázisokért felelős INGRES és adatbázis adminisztrátoroknak komoly gondot jelent az adatlokációkon a hellyel való gazdálkodás, ill. az adatbázis objektumokat alkotó file -ok paramétereinek (pld. idő) nyilvántartása. Vegyünk két példát.

1.Példa

Tegyük fel, hogy egy file systemen van egy adatlokáció, két adatbázis, és adatbázisonként több tucat tábla és index. A helyzetet bonyolíthatja esetleg még az is, hogy az adatbázisok ún. extended adatbázisok, azaz átnyúlnak más file system -eken levő más adatlokációkra is. Tételezzük fel továbbá, hogy az idők folyamán a file system -en nő az állományok mérete, és

kell szervezni, vagy -allokálni valamely táblá(ka)t, vagy indexe(ke)t. Az elképzelés jó, csak a megvalósításig vezető út komplikált.

Ha ugyanis a file system -en elfogadható helyfoglalási viszonyokat akarunk teremteni, akkor nyilván tudjuk, hogy mennyi helyet akarunk felszabadítani. Az adott méretű hely felszabadításához természetesen tudnunk kell azt, hogy melyik tábla, vagy index mekkora helyet foglal az adott file system -en. És ettől a ponttól az INGRES már alig nyújt segítséget. Rendelkezésünkre bocsát néhány tucat rendszertáblát, melyekben majdnem szabadon „búvárkodhat" a szerencsétlen adminisztrátor. Míg egyetlen ilyen esetben azt megtalálja, hogy mely táblákat, vagy indexeket a legcélszerűbb mozgatni ahhoz, hogy a kívánt méretű terület felszabaduljon, addig eltelhet akár egy negyed óra is. Ezt egy „kellően" nagy rendszer gazdája nem engedheti meg magának, neki percek alatt meg kell hoznia az optimális döntést.

2.Példa

Tételezzük fel, hogy a felhasználók dolgoznak egy adatbázisban. Egyesek olvasnak, mások írnak bizonyos táblákat, ill. indexeket. Ekkor pld. egy hardver hiba miatt váratlanul leáll a számítógép. A hiba kijavítása után az adatbázis valószínűleg inkonzisztens állapotba kerül. Az inkonzisztencia megszüntetésének kikényszerítése után célszerű az egész adatbázist egy régebbi mentésről visszatölteni. De mi van akkor, ha az adatbázis mérete több GB, és a

visszatöltés több óráig tart? Érdemes egy esetleg néhány MB-os sérült állomány miatt ekkora munkába fogni?

Ha tudnánk, hogy melyek azok a táblák, indexek, amiket a leálláskor a felhasználók éppen írtak, akkor csak azokat kellene a mentésről visszatölteni.

Egy ilyen feladatnak a megoldása még az előbbinél is időigényesebb, mert az INGRES itt még a rendszertáblákban sem tárolja a megfelelő információt.

A problémákat már látjuk, de mi a megoldás? A megoldás egy lehetséges módját az OEP Számítóközpontja Szoftverfejlesztési Ágazata Adatbázis és Rendszerprogramozási osztályának munkatársai találták meg. Létrehoztak egy szoftvert és egy adatbázist, mely az INGRES által használt file system -ek, adatlokációk, adatbázisok, táblák és indexek, valamint a hozzájuk tartozó file -ok megfelelô paramétereit begyűjti, ill. tárolja, majd célszerűen csoportosítva bemutatja őket.

A utility szolgáltatásai:

A program az INGRES master adatbázisából kiolvassa, hogy az adott INGRES installációban milyen adatbázisok, ill. adatlokációk vannak. Ezután átfésüli azokat a file system -eket, ahol adatlokációk vannak, és begyűjti a következő adatokat:

- file system kapacitás,
- file system foglaltság,
- adatlokáció által elfoglalt hely,
- egy adatbázis szegmens által az adatlokáción elfoglalt hely,
- egy adatbázis szegmensben levő tábla, vagy index szegmens által elfoglalt hely,
- egy tábla, vagy index szegmens utolsó írási időpontja

A begyűjtött adatokból a következő információkat szolgáltatja:

1. File system-ek, lokációk és az alattuk lévő adatbázis részletek
2. Adatbázis részletek és a felettük lévő lokációk, file system-ek
3. A táblákat alkotó file -ok mérete
4. Táblák mérete
5. Adatbázisok mérete
6. Adatlokációk helyfoglalása, file system-ek kihasználtsága
7. Táblák utolsó írási időpontja

Az információk részletesebb formában a mellékletben láthatók.

Mellékletek

1.

File systemek, lok ciók ,s az alattuk levõ adatbázis részletek

Loc Name: r1_dat11b
Fs Name: /dat11b

Fs Total Mb: 1505 Fs Avail Mb: 645

Db Name	Db Mb
flaci	1
folyo	598
hfr	47
jaro	206

2.

Adatbázis részletek és a felettük levõ lokációk, file system -ek

Db Name	Dba Name	Db Mb	Fs Name	Loc Name	Fs Total Mb	Fs Avail Mb
fekvo	r1sp	600	/dat16c	r1_dat16c	1908	1202
fekvo	r1sp	641	/dat34c	r1_dat34c	2007	1269
fekvo	r1sp	542	/dat9c	r1_dat9c	1001	385
finan	r11j	75	/dat16c	r1_dat16c	1908	1202
finan	r11j	70	/dat34c	r1_dat34c	2007	1269
flaci	r1fl	1	/dat11b	r1_dat11b	1505	645
folyo	folyo	598	/dat11b	r1_dat11b	1505	645
folyo	folyo	880	/dat35b	r1_dat35b	1505	440
gyszer	gyszer	10	/dat34c	r1_dat34c	2007	1269
hfr	r11j	47	/dat11b	r1_dat11b	1505	645
hfr	r11j	51	/dat35b	r1_dat35b	1505	440
horvos	horvos	8	/dat34c	r1_dat34c	2007	1269
iidbdb	$ingres	1	/inginst	ii_database	484	254
jaro	r11j	206	/dat11b	r1_dat11b	1505	645
jaro	r11j	126	/dat35b	r1_dat35b	1505	440
pecset	pecset	0	/dat16c	r1_dat16c	1908	1202

3.

A táblákat alkotó file -ok mérete

Loc Name	Db Name	File Name	Ext	File Size	Table Name	Owner
rl_dat35b	folyo	aaaaaamj	t00	237189120	lap	folyo
rl_dat11b	folyo	aaaaabne	t00	227971072	l91_11m	rlfl
rl_dat35b	folyo	aaaaabhg	t00	143484928	nev_cim0	folyo
rl_dat35b	folyo	aaaaabkp	t00	132874240	nev_cim	folyo
rl_dat11b	folyo	aaaaabml	t00	107923456	level	folyo
rl_dat35b	folyo	aaaaabml	t01	107905024	level	folyo
rl_dat9c	fekvo	aaaaaaoi	t00	103645184	fe_be92	rlsp
rl_dat16c	fekvo	aaaaaaoi	t01	103645184	fe_be92	rlsp
rl_dat34c	fekvo	aaaaaaoi	t02	103636992	fe_be92	rlsp
rl_dat35b	folyo	aaaaaaon	t00	84062208	l05	folyo
rl_dat11b	folyo	aaaaaaon	t01	84049920	l05	folyo
rl_dat9c	fekvo	aaaaaaph	t00	80019456	fe_be94	rlsp
rl_dat16c	fekvo	aaaaaaph	t01	80011264	fe_be94	rlsp
rl_dat34c	fekvo	aaaaaaph	t02	79986688	fe_be94	rlsp
rl_dat11b	jaro	aaaaabah	t00	78594048	pont_1	rlfk
rl_dat9c	fekvo	aaaaaapb	t00	67176448	fe_be93	rlsp

4.

Táblák mérete

Db Name	Table Name	Table Kb	Owner
fekvo	fe_be92	310926	rlsp
fekvo	fe_be94	240016	rlsp
folyo	lap	237189	folyo
folyo	l91_11m	227971	rlfl
folyo	level	215828	folyo
fekvo	fe_be93	201524	rlsp
fekvo	fe_be94_uj	168844	rlsp
folyo	l05	168111	folyo
folyo	nev_cim0	143484	folyo
fekvo	fe_be_diag94	134217	rlsp
folyo	nev_cim	132874	folyo
fekvo	mano_imre	110438	rlfk
folyo	nev_cim_bp	98422	folyo
fekvo	raford_89	91663	rlsp
jaro	pont_93	90242	rllj
folyo	ger	83178	folyo

5.

Adatbázisok mérete

Db Name	Db Kb
fekvo	1869159
folyo	1550770
jaro	348334
finan	152887
hfr	103944
pecset	70358
sys_tem	22703
gyszer	10663
horvos	8457
iidbdb	1651
flaci	1228

6.

Adatlokációk helyfoglalása, file system -eik kihasználtsága

Loc Name	Fs Name	Fs Total Mb	Fs Avail Mb	Loc Mb
ii_database	/inginst	484	254	1
rl_dat11b	/dat11b	1505	645	852
rl_dat16c	/dat16c	1908	1202	696
rl_dat34c	/dat34c	2007	1269	729
rl_dat35b	/dat35b	1505	440	1057
rl_dat9c	/dat9c	1001	385	609

7.

Táblák utolsó írási időpontja

Db Name	Table Name	Owner	Last write time
fekvo	aa	rlsp	1994-11-23 16:28:00
fekvo	ajtai7_1	rlfk	1994-11-22 12:19:00
fekvo	c	rlfk	1994-11-21 18:50:00
fekvo	date	rlsp	1994-11-23 14:29:00
fekvo	dbf_tb_tmp	rlsp	1994-11-23 16:33:00
fekvo	diag1	rlsp	1994-11-01 15:35:00
fekvo	diag2	rlsp	1994-11-01 15:34:00
fekvo	drg	rlsp	1994-11-01 16:26:00
fekvo	drg93	rlsp	1994-11-10 15:03:00
fekvo	drg94	rlsp	1994-11-10 15:24:00
fekvo	f_agysuml	rlsp	1994-11-01 15:53:00
fekvo	fe_be92	rlsp	1994-11-01 16:26:00
fekvo	fe_be93	rlsp	1994-11-01 16:55:00
fekvo	fe_be94	rlsp	1994-11-01 17:47:00
fekvo	fe_be94_add	rlsp	1994-11-25 13:51:00
fekvo	fe_be94_conc	rlsp	1994-11-01 15:06:00

Második generációs kliens - szerver architektúra gyakorlati alkalmazásának tapasztalatai Magyarországon

Porkoláb Zoltán - Sziebig Ferenc
ALBACOMP Consulting Group
gsd@palma1.elte.hu

Nagyon sok eszköz található a piacon, amely kliens/szerver technológiát ajánl, de hogy mi a valódi funkcionalitásuk, az messze nem tiszta. A "kliens/szerver" manapság egy nagyon divatos bűvszó. A legtöbb eszköz, mely erre hivatkozik, valójában egy kliens oldali tool, mely lehetővé teszi gyönyörű színes képernyők létrehozását (általában csak MS-Windows alatt), gyors prototípus alkalmazások elkészítését. Az eredmény gyors és látványos.

A szomorú igazság az, hogy a legtöbb terméket, melyekkel kliens/szerver rendszereket építenek, nem erre a célra tervezték. Ez különösen igaz azokra a termékekre, melyek egy egyszerű SQL alapú kliens/szerver interfészre alapulnak (pl. SQL gateway, ODBC). Ezeket sohasem szabadna komoly tranzakció feldolgozó rendszerekhez használni. Minden esetben azt találjuk, hogy a válaszidők drasztikusan növekszenek a felhasználók számának növekedésével. Ennek magyarázata a következő: a tipikus kliens/szerver környezetben négy szintű overhead rakódik egy kérés kiszolgálására:

- a kommunikációs réteg feldolgozza a kérést,
- ezután a kérés lefordítódik a megfelelő kommunikációs protokollra,
- a kérés és paraméterei átvitelre kerülnek a hálózaton a szerver oldalra,
- a szerver oldali processz veszi a kérést és végrehajtatja azt (az adatbázis szerverrel).

Ezek után az egész eljárás megismétlődik visszafelé. Ez a folyamat lehet, hogy csak egy tized másodpercet vesz igénybe, de ha egy kliens oldali funkció végrehajtása csak tíz ilyen párbeszédet igényel, már egy másodperc a plusz rárakódó idő - és ez csak sokszorozódhat, amennyiben a kliens és a szerver oldal párbeszéde nem hatékony. Természetesen még ronthatja a helyzetet a szerver vagy a hálózat szűk keresztmetszeteinek elérése.
Valójában magát az SQL nyelvet sem a kliens/szerver architektúra kiszolgálására tervezték.

A fenti kérdésekre az egyik válasz a triggerek és tárolt eljárások használata. Ezek az automatikusan induló eljárások hasznosak lehetnek például, mikor egy termék rekordot kitörölve törlődnek vele együtt a hozzá tartozó raktárkészlet, discount leíró és más rekordok. Ilyen esetben a teljes adatbázis tevékenység kiváltható egyetlen kliens hívással. Szintén hasonló megoldás, amikor több SQL parancsból álló kötegeket küldhetünk át egyben a klines oldalról.

A fenti megoldásoknak azonban vannak hátrányai. Egyrészt megsértik azt az elvet, hogy a programozónak nem szabad tudnia az adatok fizikai szervezéséről:
- ahhoz, hogy a programozó optimalizálja az alkalmazását, figyelembe kell vennie az adatbázis műveletek fizikai szétosztását a hálózaton.

Másrészt a legtöbb 4GL rendszer nem használja ki a fenti lehetőségeket. Legjobb esetben fel tudom használni a szerveren található adatbáziskezelő triggereit és tárolt eljárásait, de ezeket nekem kell egy másik nyelven leprogramoznom. Ez a karbantartásnál fog gondot okozni, mivel az alkalmazás fejlesztését kettéválasztottam két teljesen különböző környezetre: egy 4GL alapú kliensre (alkalmazási logika 4GL-ben) és egy, az adatbáziskezelő típusától függő szerver oldalra (alkalmazási logika adatbázis katalógusokban).

A kliens/szerver rendszerek **második generációja** az alkalmazásfejlesztésen kívül az elkészült alkalmazások üzemeltetésének hatékonysági kérdéseire is választ ad. Ez a válasz a következő:

az alkalmazás rugalmas és transzparens módú partícionálása azért, hogy minimalizáljuk a kliens és a szerver közötti párbeszédet.

Ebben az esetben lehetséges az alkalmazást egy teljesen önálló rendszeren (pl. MS-Windows-os PC-n) kifejleszteni és utána a futását a 4GL kód módosítása nélkül dinamikusan szétosztani egy hálózaton.

A SuperNOVA teljes mértékben kielégíti a második generációs kliens/szerver rendszerek követelményeit. A következőkben ismertetjük azon tulajdonságait, melyek ezt lehetővé teszik.

A SuperNOVA elosztott feldolgozásának néhány jellemzője

1.) Elosztott feldolgozás az SQL-en túl

Alapvető, hogy a kliens és szerver eszközök ne csak SQL-üzeneteket váltsanak. Ha elosztott feldolgozásról van szó, nemcsak adatbázis-műveleteket várunk a szervertől.

Objektumokat és eljárásokat is meg akarunk osztani.

2.) Azonos nyelv a kliensen és a szerveren

Kliens és szerver magas-szintű kommunikációs protokollt igényel és olyan parancsokat, amelyek azonos módon futnak a hálózat részét alkotó rendszerek mindegyikén. Ekkor a kliens és szerver könnyen, közbülső paraméterezés stb. nélkül megállapodhat az alkalmazás egy darabjának helyi vagy a szerveren való futtatásában. A kód bárhol futhat.

3.) Az alkalmazás-kód teljes hordozhatósága

Az alkalmazás-kód azonos kell, hogy legyen minden platformon, hogy futási idejű, dinamikus objektum- és eljárás-elosztás lehetséges legyen. Fusson a kód minden rendszeren fordítás, linkelés nélkül. Ekkor lehetséges az, hogy a kliens előző szerepkiosztás nélkül, szabadon dönthesse el, egy eljárást helyben akar-e végrahajtani avagy egy szervernek akarja-e kiadni.

4.) Az eszköz teljessége

Nem elégséges egy képernyő-maszkokat felépítő eszköz. Az eszköznek a teljes alkalmazás-fejlesztést támogatnia kell, anélkül, hogy 3. generációs kódra lenne szükség. Szükséges ugyanakkor SQL transzparens használata vagy 3. generációs kód integrációja.

5.) Dinamikus kód-elosztás és szabályozható telepítés

Ha egy alkalmazás több klienses és szerveren fut, objektumok és eljárások meg lehetnek osztva és mező-értékek jöhetnek-mehetnek a hálózaton. Szükséges, hogy egy teljes alkalmazás egy szerveren (avagy egy kliensen) legyen tárolható, és a megosztott részek szükség szerint másolódjanak át oda, ahol szükség van rájuk.

Ez futás közbeni másolást jelent, de minthogy a kód kompakt, és egy hívásnál csak egy kis rész másolódik, nem megy a teljesítmény rovására. A másolás egyszer történik meg, és egy megosztott eljárás további használata esetén a cél-rendszeren már meglevő eljárás nem másolódik át újra.

Szükség lehet arra, hogy az alkalmazás bizonyos részei a hálózat más részén legyenek tárolva és dinamikus töltődjenek be. Ez az induláskor jelent több munkát, de megkönnyíti a dinamikus disztribúciót.

6.) A kód függetlensége a hálózat szerkezetétől

Az alkalmazás kódjának függetlennek kell lennie a hálózat szerkezetétől, a szerverek nevétől, a felhasznált protokolltól, stb. A kódot esetleg egy nem-megosztott rendszerre fejlesztették, és futnia kell minden változtatás, újrafordítás nélkül egy olyan hálózati környezetben, ahol sok más operációs rendszer, adatbáziskezelő és kommunikációs protokoll van. A megosztási sémának, az elérhető szerverekre, szolgáltatásokra stb. vonatkozó információnak az alkalmazáson kívül kell lennie, és megváltozásának nem szabad kihatnia az alkalmazásra.

7.) A hálózaton levő szerverek dinamikus, futási időbeli megváltozása

Előfordulhat, hogy futási időben a hálózaton levő néhány szervert kicserélnek vagy valamely objektum vagy eljárás egy másik szerveren hajtandó végre.

Szükséges, hogy az alkalmazás futása közben a disztribúciós séma megváltoztatható legyen úgy, hogy a konfiguráció e megváltozását az alkalmazás futása tükrözze, anélkül, hogy az alkalmazáson változtatni vagy azt akár csak újraindítani kellene.

8.) Szimmetrikus szolgáltatások

Minden szolgáltatásnak a kliens - szerver szerepek kiosztásától függetlenül elérhetőnek kell lennie. A legtöbb esetben MS-Windows-os PC-k a kliensek. Mindazonáltal ennek

nem szabad kötelezően előírtan így lennie. A hálózaton levő minden egyes rendszernek kliensként is és szerverként is működnie kell, szolgáltatásokat kell tudnia kérni és kapni.

9.) A kliens - szerver szerep felcserélhetősége (reverse disztribúció)

Fontos, hogy a hálózati tevékenységek optimalizálhatóak legyenek bonyolult beállítás és alkalmazás-hangolás nélkül. Az eljárások egyszerűsíthetők, ha a kliens - szerver szerep dinamikusan felcserélhető. Sok PC-s kliens használhat egy vagy több szerveren levő szoftvert. Ez a módszer lehetővé teszi az alkalmazás központi tárolását és olyan funkciók megosztását (azaz a PC-re való kiosztását), melyek a felhasználói felületet stb. jelentik.

10.) Privát szolgáltatások

Ennél a módszernél minden kliensnek saját (dedikált) processze van a szerver(ek)en, amely őt a kívánt szolgáltatással ellátja.

11.) Fordított privát szolgáltatások

Ilyenkor a kliens és a saját (dedikált) processz a szolgáltatás elindulása után szerepet cserél.

12.) Többszörös privát szolgáltatások

Egy klienshez több különböző szolgáltatás (szervíz) kapcsolódhat. E szervízek egy vagy több szerveren is futhatnak az alkalmazás megváltoztatása nélkül.

13.) Többszintű szolgáltatások

Bizonyos esetekben egy kliens olyan szolgáltatást kérhet egy szervertől, amit az nem tud nyújtani. Többszintű szolgáltatás esetén a szerver egy másiktól kéri azt, és adja a kliensnek, amelyik nem is tud arról, hogy kérését végül is egy másik szerver teljesítette: a többszintű szolgáltatások a kliens számára teljesen transzparensek.

14.) Megosztott avagy állapot nélküli szolgáltatások

Ekkor több kliens használhatja megosztottan ugyanazokat a szolgáltatásokat (szervízeket). A szervíz adója minden kérést az illető kliens számára fenntartott kapcsolatként kezel, és amikor ez véget ér, elöveszi a következőt és elfelejti előző tevékenységét. Ezt a módot sok tranzakció-orientált környezet használja.

15.) Titkosítás

A kliensek és a szerverek közötti kommunikáció bizalmas, és ezért titkosítani kell. Ekkor a felhasználó dönti el, hogy az átviendő adatok a biztonság érdekében titkosítandóak-e.

16.) Adattömörítés.

Ekkor a kliensek és a szerverek között átviendő adatok mennyisége csökkenthető. Ennek a némiképpen megnövekedő memóriafelhasználás az ára.

17.) Adatfolyamok átvitele

Ha nagyméretű adatblokkokat a kliensek és a szerverek között átvinni, ez a lehető leggyorsabb adatátviteli mód. Nagy tömegű adatot szolgáltató és feldolgozó rendszerekben használatos.

18.) Transzparens állományelérési szolgáltatások

Létezik olyan objektum, hogy "állomány-objektum", "file object". Ez ugyanúgy elérhető és feldolgozható, mint bármely más objektum. Így minden eljárás vagy objektum, amely egy állományt kíván elérni, ezt arra való tekintet nélkül teheti meg, hogy az a hálózaton hol helyezkedik el. Ez nem igényel hálózati állományelérési rendszert (NFS) és transzparens az alkalmazás számára.

19.) Nem-váró szolgáltatások és párhuzamos feldolgozás

Sok szolgáltatás lehetséges "váró" és "nem-váró" módban. Ez utóbbi esetén a kliens kéri a szolgáltatást és utána folytatja munkáját, anélkül, hogy a szolgáltatást megvárná. Így az alkalmazás párhuzamosan futhat több szerveren és igazi párhuzamos feldolgozást valósít meg. Ha a kliens szinkronizációt kér egy bizonyos szerverrel, ezt az aszinkron esemény-kezelő valósítja meg. Ez a fent leírtakkal valamint a dinamikus hálózati kód-elosztással együtt nagy párhuzamos rendszerek felépítését teszi lehetővé egyszerűen és hatékonyan.

20.) Tranzakció monitorokkal való összekapcsolhatóság

A Tuxedo illetve Top End tranzakció-kezelő rendszerekkel való összekapcsolhatóság e rendszerek minden szolgáltatását elérhetővé teszi a fejlesztő számára. Lehetséges mind kliens, mind szerver oldali kód generálása az említett tranzakcó monitorok számára.

Az alábbiakban ismertetünk egy tipikus SuperNOVA alapú alkalmazást, amely kihasználja a termék képességét több adatbázis- ill. filekezelő párhuzamos meghajtásával, multiplatformos és szép számmal alkalmaz tárolt eljárásokat ill. SuperNOVA daemon processeket. Az alkalmazás az Országos Munkaügyi Központ (OMK) kirendeltségi közvetítési Clipperes rendszerét felváltani hivatott és jelenleg fejlesztés alatt áll.

A jelenlegi kirendeltségi Clipperes közvetítő rendszert - amely többek között a munkanélküliek, állásajánlatok és kapcsolódó vállatok nyilvántartását, adminisztrációját végzi, valamint számos paraméter alapján megpróbál a munkanélkülinek állást, a munkáltatónak munkavállalót ajánlani - az ország területét teljesen lefedve több mint 100 kirendeltségi irodában telepítették. A közvetítő modul szoros kapcsolatban áll más (ügyiratkezelési, pénzügyi) modulokkal, és a központilag elkészített programokat számos helyen egészítik ki helyi fejlesztések.

Ennek a bonyolult rendszernek relációs adatbáziskezelőben (Oracle) történő teljes újraírása hosszú folyamat, és az átállás nem történhet egyetlen hétvége alatt. A rendszert modulonként kell újraírni, miközben folyamatos kompatibilitást kell biztosítani mind az új (Oracle), mint a meglévő (Clipper) programok felé. Ekkor lehetségessé válik egyrészt az egyes kirendeltségek külön-külön történő átállítása, másrészt egy kirendeltségen belül a funkciók részenkénti kiváltása - azaz egyes átírt új (Oracle ill. SuperNOVA) programok, valamint a régi Clipper programok egymás mellett élése. További feladat a jelenleg izolált kirendeltségek (amelyek esetenkénti filetransferrel kommunikálnak) közötti megbízható on-line (vagy közel on-line idejű) adatcsere megvalósítása.

A kirendeltségekhez SCO UNIX alapú szerver tartozik Oracle adatbáziskezelővel, Novell alapú fileszerver a kirendeltségi Clipper adatállományokkal, valamint számos AT386/486 alapú munkaállomás 4/8 MB memóriával. A munkaállomások és szerverek egy ethernet hálózatba kapcsoltak, elérik a Novell szervert és (TCP alapú kapcsolattal) a UNIX szervert. Ez a hálózat routeren keresztül on-line X.25 kapcsolatban áll az OMK központi szervereivel (Data General számítógépek, Clarion diszk rendszerrel).

Az új kirendeltségi közvetítő rendszer MS-Windows alapú SuperNOVA alkalmazás, amely az AT386/486 alapú munkaállomásokon fut. A program működése az SCO szerveren telepített Oracle instance adataira épül. Itt helyezkednek el a munkanélküliek, az állásajánlatok, stb. adatai. A program az egyes adatok karbantartásán kívül az állások és ügyfelek közötti közvetítés különböző feladatait is elvégzi. A hagyományos szerver-kliens felépítés mellett a hatékonyság növelése érdekében kihasználjuk a SuperNOVA néhány plusz szolgáltatását is: A kliens memóriájában tárolt - de a server Oracle tábláival megegyezően kezelt - un. tms (temporary memory storage) táblákat alkalmazunk. A munkanélküliek és állásajánlatok egymáshoz rendelését végző, user input/outputot nem igénylő feladatokat a szerveren tárolt és kliensről aktivizált SuperNOVA nyelven megírt tárolt eljárásokkal végezzük el. Ezek az eljárások syntaxisukban semmiben nem térnek el a "normál" SuperNOVA programrészletektől. Szerverre telepítésüket az elkészült program futtatása során, a hálózati forgalom - SuperNOVA eszközökkel történő - megfigyelése alapján dönthetjük el - anélkül, hogy a programon utólagosan változtatni kéne.

Az SCO UNIX/Oracle álományok bizonyos részhalmaza - mintegy tükörként - a Novell szerveren, Clipperben is létezik. Ezeket az adatokat használják a csatlakozó - még Clipper alapú - modulok. Valahányszor módosul az Oracle állomány a SuperNOVA elvégzi a párhuzamos update-ket a Clipper állományokon is. A Clipper programok - ameddig még szükség van rájuk - változtatás nélkül futnak ugyanazon munkaállomáson az új rendszerrel párhuzamosan - DOS ablakban. Paradox módon azok a Clipper programok, melyeket eddig cipőkanállal szorítottak be a 640Kb-ból megmaradt szabad memóriába - hála a Windows alatt magas memóriába tölthető hálózati drivereknek - most sokkal több hellyel gazdálkodhatnak.

Az új rendszer fontos eleme a kirendeltségek közötti kommunikáció, amely replikációs elven épül fel. Az ország minden egyes települése hozzátartozik valamelyik kirendeltséghez (egy időben pontosan egyhez); oda jelentik be az állásajánlatokat, ott adminisztrálják a munkanélkülit. Ugyanakkor egyes állásajánlatok vonzási körzete átnyúlhat más körzetekbe is. Ezeket a "dinamikus régiókat" település-település közötti relációk írják le, a munkavégzés helye és a potenciális munkavállalók lakhelye között. (Menetrendi és más okok miatt még az sem bizonyos, hogy ez a reláció szimmetrikus.) Nyilvánvaló, hogy ezeket az állásajánlatokat el kell juttatni mindazokra a kirendeltségekre, amelyeknek legalább egy települése beletartozik az állásajánlat dinamikus régiójába.

(A másik potenciális megoldás: olyan központi állás-adatbázis felépítése, amelyben többezer távoli munkaállomásról egyszerre végeznek bonyolult, on-line leválogatásokat, kevésbé hatékony lenne. A feladat jellemzője, hogy a közvetítés az ügyféllel végzett interjú közben, a szűrő feltételek folyamatos változtatás mellett több lépésben iterálva történik, míg az állásajánlatok felvitele, karbantartása jóval ritkább és kevésbé erőforrásigényes feladat. Ésszerű az előzőt helyben, a kirendeltségen végezni, és az utóbbi tevékenység információit utaztatni a kirendeltségek között. Amúgy sincsen jelentősége, ha egy állásajánlat pár perces késéssel jelenik meg a szomszédos iroda kínálatában, így az adatreplikáció aszinkron módon is történhet.)

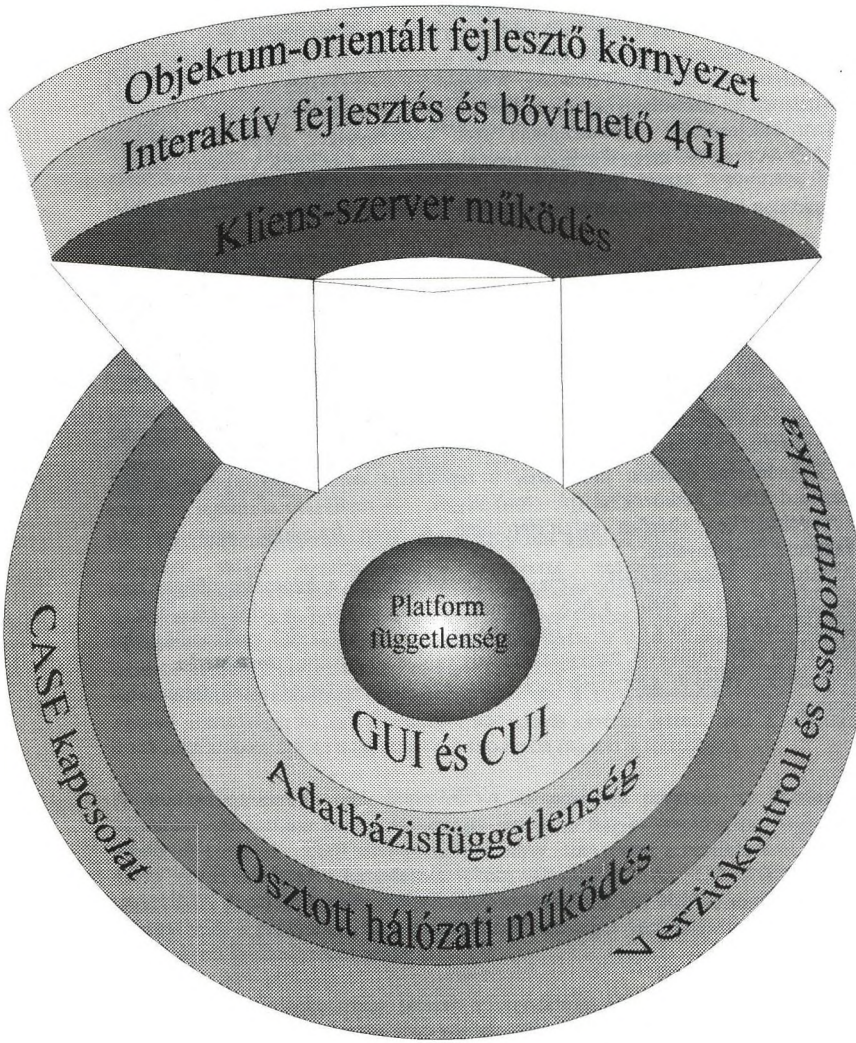
A munkaállomásokon futó SuperNOVA kirendeltségi program az állásajánlatok bármilyen módosítása esetén beállít egy ezt jelző flag-et (replikációs flag) a tábla módosult sorában. Ettől eltekintve a rendszer ezen része a replikációról "mit sem tudva" végzi feladatait a kirendeltség lokális hálózatában. Az SCO UNIX serveren - mint UNIX daemon process - időről időre aktivizálódik egy eljárás, amely elindítja azt a SuperNOVA programot, amely a replikáció tényleges véghezviteléért felelős. Ez a program két tevékenységet végez el.

Elsőként azokat a rekordokat, melyek a kirendeltség állományában a replikációs flag kijelöl, elviszi OMK központi állás-adatbázisába és ott elvégzi az állomány szükséges módosítását (insert, update, mark-to-delete). A kezelt rekordban lévő "munkavégzés helye" információ (település), és az OMK központi szerveren tárolt Dinamikus régió tábla (település-település kapcsolat) és a Kirendeltségek települései tábla (település-kirendeltség) alapján eldönti, mely kirendeltségek számára kell a rekordok új állapotát közölni. Ezeknek a kirendeltségeknek "üzenetet hagy" a Replikációs FIFO-ban: az értesítendő kirendeltség (valójában a kirendeltségi daemon) azonosítója, illetve referencia a kezelt rekord(ok)ra (kulcs).

Második lépésként megnézi, hogy a Replikációs FIFO-ban saját maga számára van-e üzenet. Ha van, kikeresi a megfelelő rekordot a központi állás-adatbázisból, és elvégzi a szükséges módosításokat a saját kirendeltségi állományában. Végül törli a neki szóló, már feldolgozott üzenteket a FIFO-ból. A Replikációs FIFO tábla tehát tényleg FIFO-ként üzemel: az üzenethagyók appendálják, az üzenetfogadók csonkolják. Ezután a program befejeződik. Természetesen ezen tevékenységek során a replikációs SuperNOVA program maga is számos tárolt eljárást indít el az OMK központi gépén.

Az ismertetett aszinkron modell több előnnyel is jár.

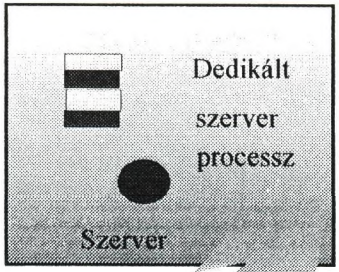
- A tényleges kirendeltségi program "semmit sem tud" a replikációról. Emiatt egyszerűbb, és gond nélkül fut az OMK központi erőforrások elérhetetlensége esetén is.
- A központi erőforrások nem munkaállomásonként, hanem kirendeltségenként fogadnak egy-egy replikációs session-t, ami legalább egy nagyságrend javulást jelent a központi server és a hálózati erőrorrások terheltségében.
- A központi erőforrások elérése mindig egy oldalról (alulról) kezdeményezett, így programozása egyszerűbb és a protokoll biztonságosabb.
- A kirendeltségi daemon processzek a replikációt emberi beavatkozás nélkül, standard UNIX eszközökkel (pl. cron) végzik. A replikáció sűrűsége az egyes kirendeltségeken eltérően paraméterezhető, a helyi forgalom, vagy hálózati kapacitás függvényében.
- A folyamat önszinkronizáló. Ha egy kirendeltséggel a kapcsolat ideiglenesen megszakad, akkor azalatt a helyi állományban szaporodnak a jelölt rekordok, a központban pedig nő a FIFO. A kapcsolat helyreállta után az első replikációs ciklus (bár kicsit több munkával, mint máskor) helyreállítja a normális helyzetet.
- A dinamikus régió ill. a kirendeltségek határainak átszervezése egyszerű eljárásokkal véghezvihető.



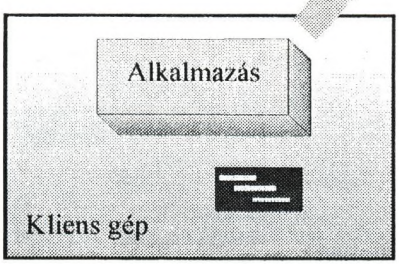
SuperNOVA

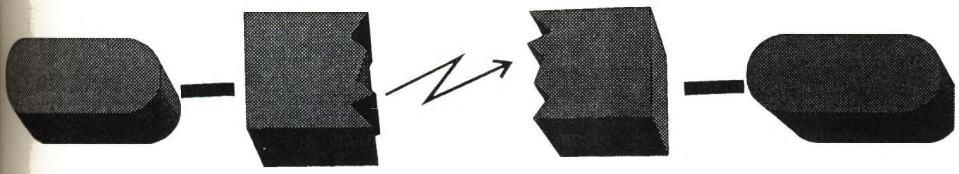
1. ábr

Protocol Overhead
Szerver oldali parancsértelmezés



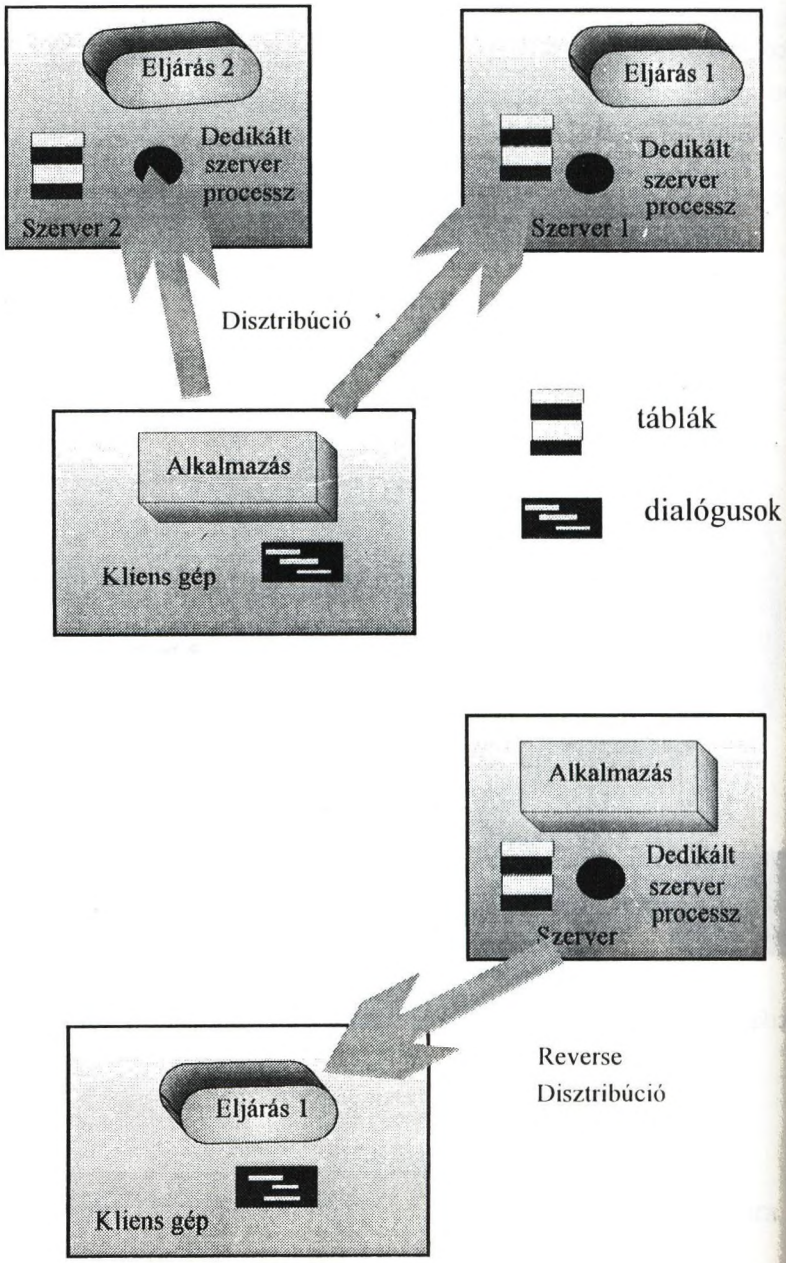
SQL szintű kommunikáció
Automatikus SQL generálás





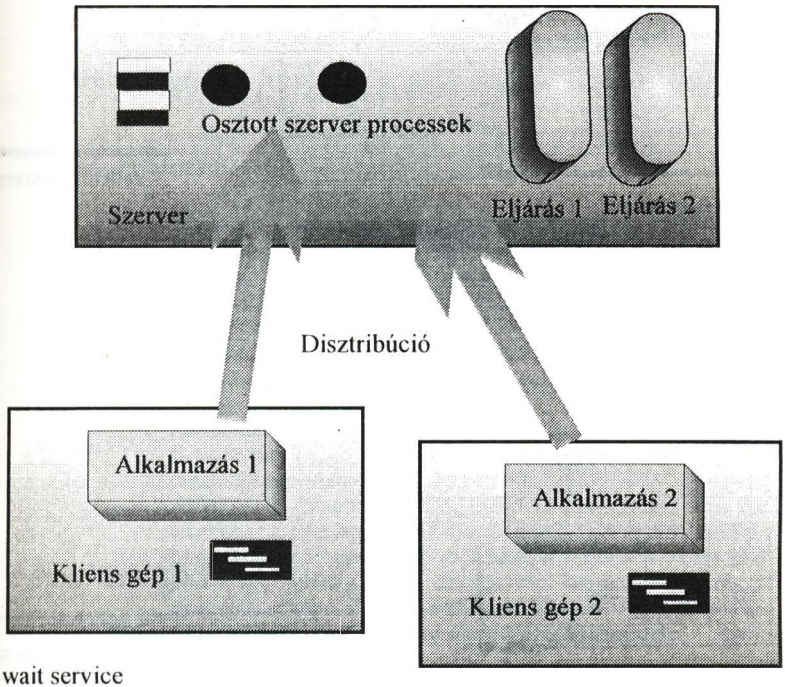
User interface | Applikációs logika | Applikációs logika | Adatbázis kezelés

Triggerek, tárolt eljárások vagy 4GL?



SuperNOVA

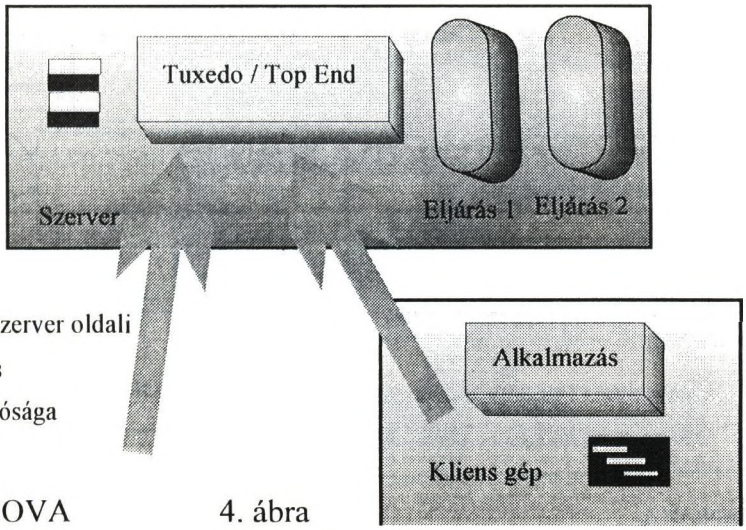
3. ábra



No wait service

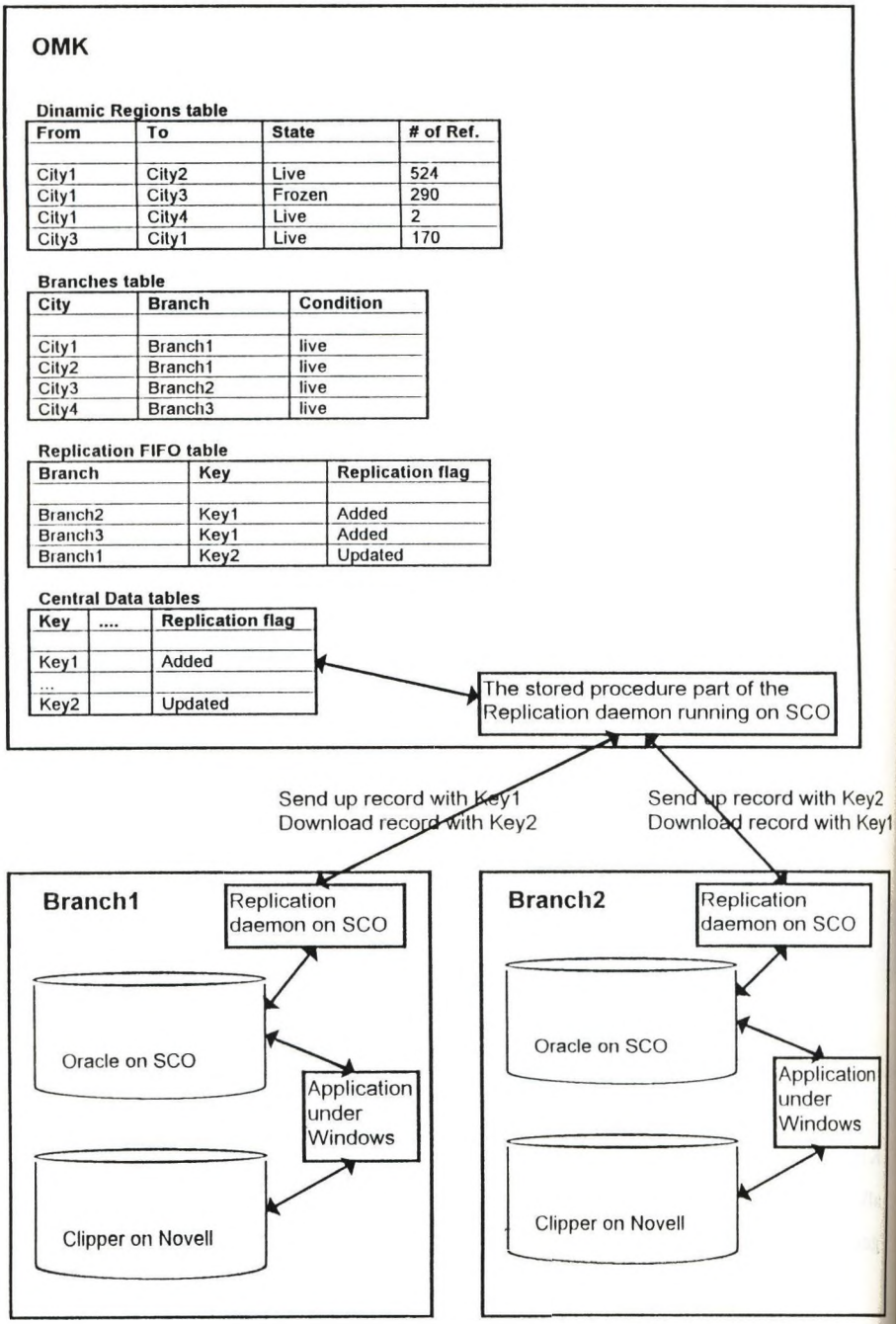
Szinkronizációs lehetőség

Parallel processzing



Kliens és szerver oldali alkalmazás generálhatósága

SuperNOVA 4. ábra


OMK
Dinamic Regions table
From
To
State
# of Ref.
City1
City2
Live
524
City1
City3
Frozen
290
City1
City4
Live
2
City3
City1
Live
170
Branches table
City
Branch
Condition
City1
Branch1
live
City2
Branch1
live
City3
Branch2
live
City4
Branch3
live
Replication FIFO table
Branch
Key
Replication flag
Branch2
Key1
Added
Branch3
Key1
Added
Branch1
Key2
Updated
Central Data tables
Key
....
Replication flag
Key1
Added
...
Key2
Updated
The stored procedure part of the Replication daemon running on SCO
Send up record with Key1
Download record with Key2
Send up record with Key2
Download record with Key1
Branch1
Replication daemon on SCO
Oracle on SCO
Application under Windows
Clipper on Novell
Branch2
Replication daemon on SCO
Oracle on SCO
Application under Windows
Clipper on Novell

IQ*DBA - az IQSoft interaktív adatbázisadminisztrátora

Gere Tamás - Németh Miklós (IQSOFT RT)

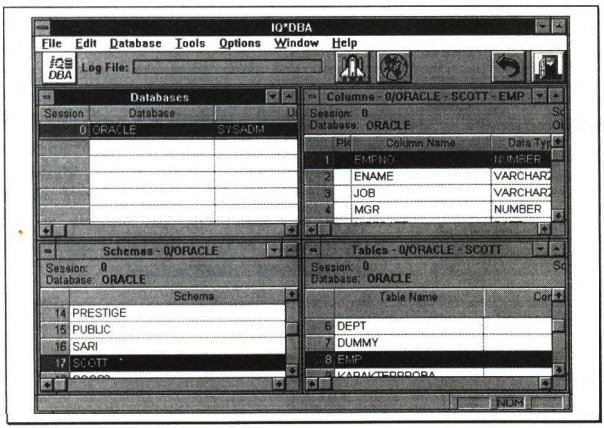
A programról röviden

A relációs adatbázis karbantartása, a felhasználói jogosultságok kiosztása, az adatbázisban tárolt objektumok és kapcsolataik feltárása mind olyan feladat, amivel egy adatbázis adminisztrátornak nap mint nap szembe kell néznie. Ez a munka sokszor nem könnyű, főleg, ha az adatbázis elég terjedelmes és bonyolult. Hasonló feladatokkal kell megküzdenie a fejlesztőknek is, amikor egy információs rendszeren dolgoznak.

Előfordulhat, hogy a végfelhasználóknál, tesztelőknél, fejlesztőknél stb. eltérő állapotú adatbázis van. Különböző lehet a tartalma vagy akár a szerkezete is. Nem tudni, hogy egy tábla, nézet (view), index vagy szinoníma létezik-e vagy sem. Az adatbázist a jelenlegi állapotában kellene kivinni az ügyfélhez. Megvannak-e a kapcsolatok az adatbázis megfelelő objektumai között (pl. primary key - foreign key kapcsolatok). Ezek a dilemmák sokszor okoznak fejtörést a fejlesztés egyes fázisaiban vagy egy használatban levő rendszer karbantartása során.

Az ilyen és ehhez hasonló problémák megoldásához használható az IQ*DBA. Segít megismerni az adatbázis szerkezetét, lehetővé teszi, hogy egyszerű, interaktív módon módosítsuk azt, megmutatja az adatbázisban tárolt adatokat és adatösszefüggéseket, stb.

Az IQ*DBA



Az IQ*DBA szolgáltatásai

Az IQ*DBA szolgáltatásai három fő területre csoportosíthatóak:

- Adatbázisszerkezet Bejáró
- Adatbázisszerkezet Manipuláló
- Adatmegjelenítő

Adatbázisszerkezet Bejáró

Az Adatbázisszerkezet Bejáró a kezelni vagy megtekinteni kívánt objektum egyszerű elérésére, megtalálására ad módot.
Az Adatbázisszerkezet Bejáró lehetővé teszi, hogy egyszerű "mutass és kattints" (point-and-click) technikát használva bejárjuk az adatbázis hierarchiát, felkutatva a számunkra fontos objektumokat, információkat. Pl.: Ha egy tábla felépítését akarjuk megnézni (hasonlóan az ábrához):

1. bejelentkezünk a megfelelő adatbázisba,
2. az adatbázist jelképező bejegyzésen kettőt kattintunk az egérgombbal, megjelenik az adatbázis összes sémája (schema),
3. megkeressük azt a sémát, amelyikhez a tábla tartozik, kijelöljük,
4. kettőt kattintunk, mire megjelenik a séma összes táblája,
5. megkeressük a táblát,
6. kettőt kattintunk, és megjelenik a tábla leírása: oszlopai neve, típusa, hossza, stb.

Séma definíció

Megjegyzés: **Sémának** nevezzük azon adatbázis objektumok együttesét, amelyek egy adott adatbázis felhasználó tulajdonában vannak. A séma tehát egyetlen adatbázis felhasználóhoz tartozik és neve a felhasználó nevével egyezik meg.

Adatbázisszerkezet Manipuláló

Az Adatbázisszerkezet Manipuláló az adatbázis használatával kapcsolatos karbantartási tevékenységek egyszerű elvégzését teszi lehetővé.
Az adatbázis szerkezetének bejárása közben minden ponton elérhetők az Adatbázisszerkezet Manipuláló környezetérzékeny szolgáltatásai. Ha pl. éppen a felhasználók jogosultságait vizsgáljuk, akkor adhatunk vagy elvonhatunk bizonyos jogokat a kiválasztott felhasználótól.

Az Adatbázisszerkezet Manipuláló többek közt lehetővé teszi, hogy

- objektumokat, jogosultságokat hozzunk létre, módosítsunk vagy töröljünk a hierarchia bármely pontján;
- az objektumokat (tábla, nézet, oszlop) megjegyzésekkel lássuk el;
- tevékenységünket naplózzuk;
- használjuk az SQL*Executort, melynek segítségével az általunk megadott SQL műveleteket végrehajthatjuk, illetve általa megnézhetőek (és esetleg módosíthatóak) azok az SQL parancsok, amelyek a program használata során az adatbázisban változást idézhetnek elő.

Adatmegjelenítő

Az Adatmegjelenítő lehetővé teszi az adatbázisban tárolt adatok és kapcsolataik ablakban való megjelenítését.
Ha az adatbázis hierarchia olyan szintjére érünk, ahol adattartalom van (tábla, nézet), akkor az adott objektum adattartalmát meg tudjuk nézni. Még hasznosabb, hogy megkereshetjük mindazokat a kapcsolatokat az adatbázisban, amelyek az objektum (csak tábla) kiválasztott sorára hivatkoznak.

Az IQ*DBA v1.0 által támogatott adatbáziskezelők

Az IQ*DBA v1.0 által jelenleg támogatott relációs adatbáziskezelők az alábbiak:

- SQLBase v5
- ORACLE v6
- ORACLE7

Megjegyzés: Tervezzük ezen kör bővítését az IQ*DBA későbbi verzióiban. Először az SQLBase v6 és a SyBase (SQLServer) adatbáziskezelőket fogjuk felvenni a palettára, majd a piaci igények függvényében döntünk a további fejlesztésekről.

Kiknek ajánljuk az IQ*DBA -t?

Az IQ*DBA -t olyan számítástechnikai szakembereknek ajánljuk, akik

- **adatbázis adminisztrátori** teendőket látnak el;
- **fejlesztőként** dolgoznak olyan információs rendszerek kidolgozásában, amelyek valamilyen relációs adatbázison alapulnak.

A program felhasználóiról feltételezzük, hogy

- felhasználói szinten ismerik a Windows 3.1-et;
- ismerik az (adatbáziskezelő specifikus) SQL nyelvet;
- ismerik (legalább alapjaiban) az általuk használt adatbáziskezelő rendszer belső szerkezetét;
- értik az angol nyelvet. Ez az utolsó megszorítás furcsa lehet a magyar piacon, de úgy érezzük, hogy egy számítástechnikai szakembernek ez a szakmai angol nem okozhat nehézséget (sőt, nem kell bajlódnia a már ismert angol fogalmak magyar megfelelőinek megtanulásával).

Nagyméretű adatbázisok kezelésével és adatkonverziókkal kapcsolatos tapasztalatok

A társadalombiztosítás területén nagyméretű adatállományok fordulnak elő tekintettel arra, hogy az egészségügy és társadalombiztosítás nagy ember és intézménytömeget érint.

Ilyen nagy állományok:

Egészségbiztosítási igazolvány állomány
Járulék és folyószámla állomány
Finanszírozási állomány
Fekvőbeteg állomány

Ezen állományok nagy része a Megyei Egészségbiztosítási Pénztáraknál keletkezik, amelyeket valamilyen úton feldolgozás céljából a nálunk levő DEC 10000-és központi feldolgozó számítógépbe kell betáplálni.

Az adatok floppykon, mágnesszalagokon, kazettákon, vagy hálózaton keresztül jutnak a rendeltetési helyükre. Az állományok TXT, DBF, pascal típusúak, vagy cobol programmal előállítottak, különböző platformokról jönnek, DOS, BS2000, UNIX, VMS operációs rendszerrel hozták azokat létre.

Az adatok ezen sokszínűségét kell valahogy közös nevezőre hozni.
Ez a közös nevező az, hogy minden adatot INGRES adatbáziskezelő segítségével adatbázisokba töltünk be.
Amíg ez a végső állapot létrejön, az állományokon sok munkát kell végezni. Munkatársainkkal sok konverziós programot, eljárást dolgoztunk ki ahhoz, hogy az adatállományok végső formájukba, az adattáblákba kerüljenek.
A következő probléma a meglevő adattáblák kezelése, ha az nagy méretet jelent.

Adattábláink viszonylag egyszerű szerkezetűek, de esetenként nagyon sok mezőt tartalmaznak. Természetesen a nagy méret úgy jön létre, hogy vagy nagyon sok a mező, vagy nagyon sok a sor (azaz a rekordszám), vagy mindkettő sok.

Az adattáblák betöltésével a következő dolgokra kell ügyelni.

— Megfelelő méretű és darabszámú lokáció kialakítása az adatbázis számára.
— Az adott tábla betöltésénél a lokációról soha nem szabad megfeledkezni. (melyik lokációt, vagy lokációkat használja fel a tábla)
— Megfelelő nagyságú „log" file-t kell kialakítani.
— Betöltésnél a tárolási strukrúra heap legyen!
(Ezek mindegyikét az előadásnál konkrét utasításokkal, példákkal illusztrálom)

Az adattáblák feldolgozásánál a következő dolgokra kell ügyelni:

— Megfelelő számú és méretű legyen a munkaterület (II_WORK DIR).
— Lehetőleg több adatbázis szerver működjön (Jó, ha adatbázisonként külön adatbázis szerver működik nagy adatbázisok esetében)
— Megfelelő tárolási struktúrát válasszunk a feldolgozástól függően.
— Szükség esetén másodlagos indexeket is használjunk.
— Nagy adattáblák összekapcsolása (join - olása) esetén futás előtt használjuk ki a QEP adta lehetőségeket.
— Alkalmazzuk az optimalizálást!
(Ezek mindegyikére utasításokkal együtt példákat mutatok)

Jó, ha ismerjük az IPM, II NAMU, II MONITOR INGRES utilityket, mert alkalmanként nagy segítséget nyújthatnak elakadás esetén.
Gyakorlatban állandóan elő fog fordulni az a probléma, hogy adattáblákat ki kell másolmi TXT típusú állományba, hogy a fiókintézetekben feldolgozzák.
Ez is felvet bizonyos problémákat különösen akkor, ha a fiókintézettől ugyanolyan adatokat kapunk időnként, mint amiket mi is küldünk nekik.

— Ügyelni kell az ékezetes konverzióra.
— A konkrét INGRES parancsot körültekintően kell megválasztani.
(Erre is konkrét alkalmazási példát mutatok)

Mindezek ellenére elő fognak fordulni hibák, váratlan lerobbanások, adatlokáció betelések.

Mi ekkor a teendő?

— Az adattáblák átszervezése, átlokálása.
— Új lokációk beiktatása.

(Konkrét példákkal illusztrálva ezeket.)

Budapest, 1995. január 9.

Tornai Imre

Bánné dr Varga Gabriella

CASE eszközök a 90-es évek rendszerfejlesztésében

Relációs adatbáziskezelők II. konferenciája
Budapest, 1995. január 25 - 27.

CASE jelentése

CASE (**C**omputer - **A**ided **S**ystems **E**ngineering),

amely jelenti

- a szoftverkészítési folyamat fázisokra bontását,
- a fázisokhoz különböző technikák használatát,
- dokumentációs szabványokat,
- minőségbiztosítást,

(Software Engineering)

valamint

- a fentiek számítógépes támogatását,
- prototípuskészítést,
- kódgenerálást.

(Computer Aided)

80-as évek változásai

- Különböző modellek, technológiák lényegi, közös vonásainak felismerése, módszerek "konvergálása"

- PC-k megjelenése, terjedése

- Fentivel kapcsolatos magas szintű grafikus lehetőségek

új lökést adott a CASE technológiák fejlődésének

(Egyesek innen származtatják a CASE technológiákat, legalábbis mai értelemben)

Leggyakrabban használt struktúrált módszertanol

I. de Marco féle struktúrált rendszerelemzés

II. Gane & Sarson struktúrált rendszerelemzés

III. Yourdon struktúrált rendszertervezés

IV. Martin féle információ alapú iskola

V. SSADM

VI. ORACLE CASE*Method

VII. Chen féle adatmodellezés

VIII. Jackson - féle struktúrált tervezés

IX. Ward - Mellor módszertan real-time rendszerek készítésére

Leggyakrabban használt technikák

1. Adat aspektus:

Egyed - kapcsolat modellezés:

- ORACLE ill
- Chen féle

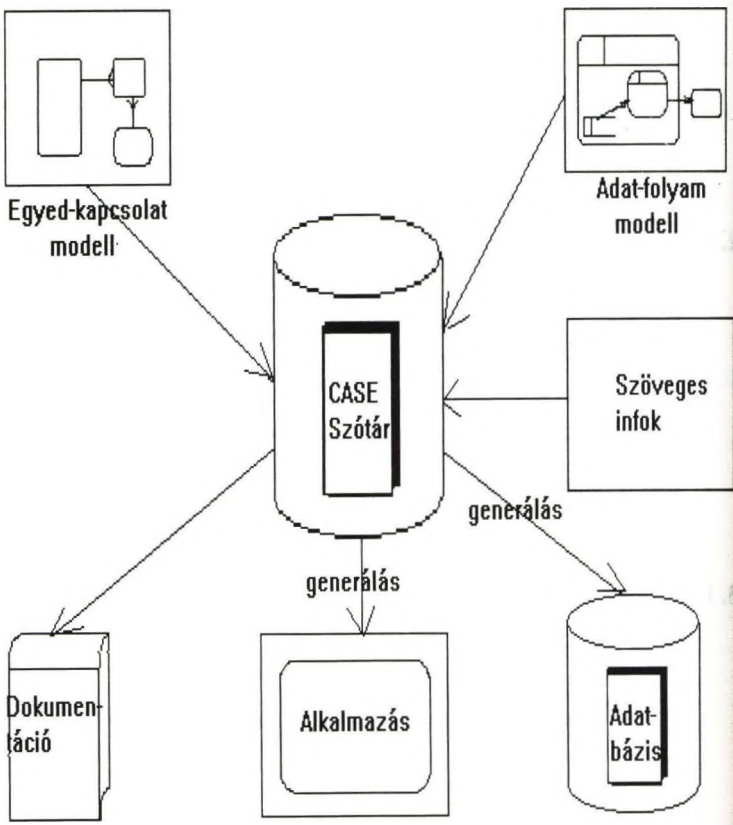
2. Funkció aspektus:

- DFD diagram / SSADM ill ORACLE CASE /
- de Marco féle DFD
- Yourdon struktúradiagrm
- Elemi funkcióleirás

3. Esemény aspektus:

- Egyedéletrajz
- Funkció "függési" diagram

CASE környezetek



Használat szintjei

1. Elemzési információk tárolása, dokumentáció készítés

2.Adatbázis generálás

3. Alkalmazás generálás

Mikor használjuk?

1 Ha teljes az **elkötelezettség**

- mind fejlesztői
- mind vezetői oldalon

2. Ha a **módszertani alapok stabilak**

3. Ha rászánunk az első projektre **1,5 -2 -ször több időt,** mint normál esetben

CASE fejlesztési esettanulmány

Táblaszám: 166

Oszlopszám: 1805

Átlagos oszlopszám: 11

Menük száma: 326 / csak formok, reportok nem !/

Alrendszerek:

	Form	Report
ALRENDSZER1	59	21
ALRENDSZER2	49	23
ALRENDSZER3	41	28
ALRENDSZER4	61	12
ALRENDSZER5	54	22
ALRENDSZER6	34	?
ALRENDSZER7	15	?

Stratégiai elemzés: 3 hó

Részletes elemzés átlag: 4 hó

Kivitelezés: 3 hó

CASE használat szintje: Kódgenerálás

Integráltság, nyitottság és testre szabhatóság Westmount I-CASE-ben

dr. Balogh Kálmán (OpenSoft Kft.)

1525 Budapest 114, Pf. 49.
Tel, fax: 160-0717; 169-9542.
E-mail (Internet): kbalogh@informix.opensoft.kfki.hu .

Szoftver eszközök értéke, használhatósága erősen függ a következő tulajdonságoktól

- az eszköz komponenseinek integráltsága
- az eszköz nyitottsága
- az eszköz testre szabhatósága.

Különösen érdekes a szoftver testre szabhatósága egy, a rendszerkészítés teljes folyamatát támogató CASE eszköznél, ahol fontos, hogy a konstruktív modellező-konkretizáló-generáló lépéseket a fejlesztendő alkalmazás speciális igényeinek megfelelően lehessen szabályozni.

A felsorolt tulajdonságok nem függetlenek egymástól. A Westmount CASE eszköze ezeket egymással összhangban, színvonalasan elégíti ki. A továbbiakban az eszköz tulajdonságait vázoljuk.

1 Integráltság

Az eszköz neve (Integrated CASE) arról tanúskodik, hogy a Westmount cég a gazdag eszközkészlet -pl. kb. 8-féle diagramkészítési technika - integrálását fontos célnak tekinti [1]. A diagramok a fejlesztendő alkalmazás egy-egy összetartozó, de még jól áttekinthető részét különböző szempontokból, nézetekből mutatják be. A kurrens objektum különböző nézetei között gyorsan váltani lehet. Az ellenörzési lehetőségek és az egységes repository a fejlesztési technikák összehangolt alkalmazását bitosítják.

2 Nyitottság [2]

Az I-CASE többféle tekintetben is nyílt rendszer:

- Összetett dokumentum készítésére szolgáló komponense összekapcsolható több DTP eszközzel (*Interleaf*, *Frame Maker* vagy *WordPerfect*).
- Más CASE eszközökkel való információcseréhez rendelkezik *CDIF* csatolóval (szabványhoz való alkalmazkodásban megnyilváló nyitottság).
- Változatokkal rendelkezik különböző fejlesztési módszerek (*SSADM*, *Yourdon Rumbaugh*, illetve *Ward/Mellor*) támogatására.
- RDBMS-ek szerint is rendelkezik változatokkal; külső adatbáziskezelőket (pl. az *Informix*ot) egyrészt saját maga megvalósításában használja fel hatékonyan, másrészt teljes és specifikus fejlesztési környezetet biztosít számukra.

Az I-CASE nyitottsága változataiban is megnyilvánul. Specifikus változatai a különböző módszerekkel és RDBMS-ekkel való szoros integrálást teszik lehetővé.

---

[1] A Westmount eszközök az *OVUM Ltd.* 1993. szeptemberi, 34 CASE eszközről szóló összehasonlítása szerint is jól integráltak, kategóriájukban a legkiválóbbak közé tartoznak.

[2] '94-ben a Westmount cég kapta *az X/Open szervezet díját*, (szoftverfejlesztő eszközök kategóriában) amelyet a szervezet évente ítél oda nyílt rendszerek készítőinek.

3 Testre szabhatóság

A nagybani fejlesztési folyamat testre szabható a *fejlesztési fázis - munkapad* összerendelés megváltoztatásával, illetve a munkapadok, menük változtatásával, bővítésével.

Erős, RDBMS-specifikus kódgenerátorai (SQL, 4GL, 3GL) az alkalmazás igényei szerint hangolhatók [3]. SQL constraint, integritási feltétel, kapcsolt táblák kezelési politikáját leíró szabály, tárolt procedúra és kivétel kezelő, valamint 4GL, illetve 3GL (C vagy C^{++} program beágyazott SQL utasításokkal) sémákra támaszkodhatunk. A diagramokat kiegészítve a kódolás magasabb szintjét a *PDL* (Program Design Language), alacsonyabb szintjét és a sémák módosítását, ill. elkészítését a *TCL* (a Berkeley University Tool Command Language-e) támogatja és teszi flexibilissé.

[3] A konkretizálási folyamat minőségét eszközspecifikus ellenőrzésekkel, teljesség ellenőrzéssel, keresztreferenciák készítésével biztosíthatjuk.

ADATGYŰJTÉS SZERVEZÉS TERVEZÉSE
a KSH-ban

ORACLE CASE segítségével

(Györki Ildikó - Központi Statisztikai Hivatal)

Bevezetés

A KSH 1992-ben vásárolta meg az ORACLE rendszert. A központot és a 20 területi igazgatóságot hálózatba kapcsolt egységes rendszerként kívánja felépíteni. A teljes rendszer tervezése, a régi adatbázisok és alkalmazások átállítása az új környezetre, új témák adatbázisba szervezése, alkalmazások készítése jelentős tervezési, programozási, dokumentálási munkát jelent.

Az egyik elsőként megoldandó feladat a gazdálkodó szervezetekre vonatkozó adatgyűjtések szervezése. A statisztikai tevékenység legmunkaigényesebb feladata az adatszolgáltatók kiválasztása, kérdőívek kiküldése, fogadása, az adatszolgáltatókkal való kapcsolattartás. Ezt fogja össze az un. adatgyűjtés szervezés alrendszer.

Az alrendszer tervezése az ORACLE CASE módszerével és eszközeivel (CASE* Dictionary, CASE* Designer, CASE* Generator for SQL* Forms and SQL* Reports) történik. A feladat nemcsak az adott alrendszer megtervezése, hanem annak megállapítása is, hogy milyen szerepet töltsön be a CASE a teljes statisztikai rendszer tervezésében, dokumentálásában, az alkalmazások készítésében.

Az előadás összefoglalja a megoldandó feladatot, a tervezés lépéseit, a megvalósítás módját. Beszámol a CASE alkalmazás tapasztalatairól

1 A feladat:

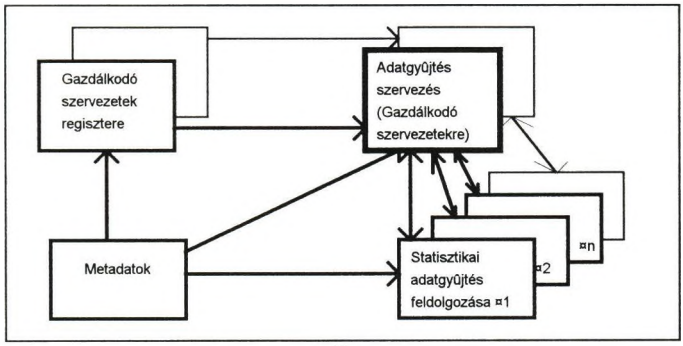
Az adatgyűjtések szervezését támogató rendszert először a gazdálkodó szervezetek körében végzett adatgyűjtésekre vonakozóan alakítjuk ki. A rendszernek a következő feladatokat kell ellátni:

- Álljon rendelkezésre
  - az adatgyűjtésekhez tartozó adatszolgáltatók kiválasztási módszere,
  - az adott adatgyűjtési időszakokban (hónap, negyedév, év) működő gazdálkodó szervezetekre, tevékenységükre vonatkozó információ.
- Jelölje ki az egyes adatszolgáltatók által teljesítendő adatgyűjtéseket.
- A kérdőívek postázásához
  - biztosítsa, hogy megfelelő példányszámú nyomtatvány álljon rendelkezésre,
  - listákkal, etikettel támogassal különböző adatgyűjtések egyidejű kiküldését az adatszolgáltatónak.
- A kérdőívek fogadásánál
  - kísérje figyelemmel a kérdőívek beérkezését,
  - vizsgálja, tárolja a nem teljesítés indokát.
- Szolgáltasson információt az adatszogáltatók leterheltségéről.
- Alapul szolgáljon a reprezentativ adatgyűjtések mintaválasztásához.
- A kérdőívek ellenőrzéséhez, feldolgozásához, a minták teljeskörűsítéséhez biztosítson adatokat az adaszolgáltatókról, jellemzőikről és adatszolgáltatásaikról.

Az adatszolgáltatók kijelölése, az infomációk elemzése a központ feladata. A kérdőívek kiküldése, a kérdőívek fogadása, az adatszolgáltatókkal való kapcsolattartás viszont a területi igazgatóságokon történik.

2. A tervezést megelőző kiinduló lépések

Az adatgyűjtés szervezés tervezésének első lépése a feladat elhelyezése, az alrendszer határainak megállapítása volt a teljes rendszeren belül. Mely funkciók képezik az adott alrendszer részét és melyek tartoznak más alrendszerhez, hol vannak közösen használt egyedek, funkciók.



1.ábra
Adatgyűjtés szervezés helye a rendszerben

A CASE használat szempontjából ennek különösen nagy jelentősége van, mivel itt történik a CASE alkalmazás alapegységének az un. **alkalmazásnak** a kijelölése. A helyes alkalmazás kiválasztás nagyban befolyásolja a munka sikerét.

- Egy adott alkalmazáson belül egyszerübb a munkavégzés, mint amikor más alkalmazás objektumait is használni kell.
- A több alkalmazás által is használt egyedeket a konzisztens kezelés érdekében viszont csak egyszer, a fő alkalmazásukban célszerű leírni.
- Ennek az a feltétele, hogy valamennyi alkalmazás fejlesztése párhuzamosan történjen egy adott szintig a CASE rendszerben.

Nehéz teljesíteni ezt a követelményt. Az utólagos módosítás viszont sok plusz munkát okoz.

Az ábra a statisztikai rendszer fő alrendszereit (alrendszer típusait), fő alkalmazásait mutatja. A nyilak az alrendszerek elemeinek közös használatát jelzik. Jelenleg a gazdálkodó szervezetek adatgyűjtés szervezése és a metaadatok alrendszerekhez tartozó alkalmazások tervezése indult el CASE-ben.

A tervezés induló fázisában kellett kidolgozni az alkalmazott **névkonvenciót** is a rendszerben illetve a CASE-ben használandó objektumokra.:

- felhasználókra,
- alkalmazásokra,
- egyedekre, táblákra,
- attributumokra, oszlopokra,
- funkciókra, modulokra

Az ajánlások ellenére nem beszédes nevekkel dolgozunk a rendszerben - mivel a statisztikában sok egymáshoz tartalmilag közel álló objektum van, aminek az egyértelmű azonosítása nehézkes -, hanem témacsoporton, témán belüli azonosítókkal, sorszámokkal. Ez a kényszerűség viszont nem eléggé olvasmányos diagramokat eredményez a CASE-ben.
Az ábrákhoz mindig mellékelni kell az egyedek leírását is.

3. A tervezés fázisai és az alkalmazott eszközök

A CASE módszer a
- stratégia
- elemzés
- tervezés
- implementálás

fázisokat különbözteti meg és ajánlásokat ad, hogy az egyes fázisokban milyen eszközöket, technikákat alkalmazzunk.

A **stratégia tervezés** a teljes alrendszert átfogta. Ennek során a CASE* Dictionary és Designer segítségével elkészült:
- átfogó Adatfolyam diagram és leírás (lásd 2. ábra)
- az Egyed-kapcsolat diagram. és leírás,
- a Funkció hierarchia diagram és leírás
- valamint a rendszer megvalósításának időbecslése (rendszer méret).

Az **elemzést** nem a teljes alrendszerre végeztük el, hanem csak a rövid távon megvalósítható funkciókhoz kapcsolódóan. Ehhez CASE* Dictionary-vel és Desginer-rel
- részletes Funkció hierarchia diagramot és funkció leírást
- részletes Egyed-kapcsolat diagramot és adatleírást
- Domain (értékkészlet) leírásokat

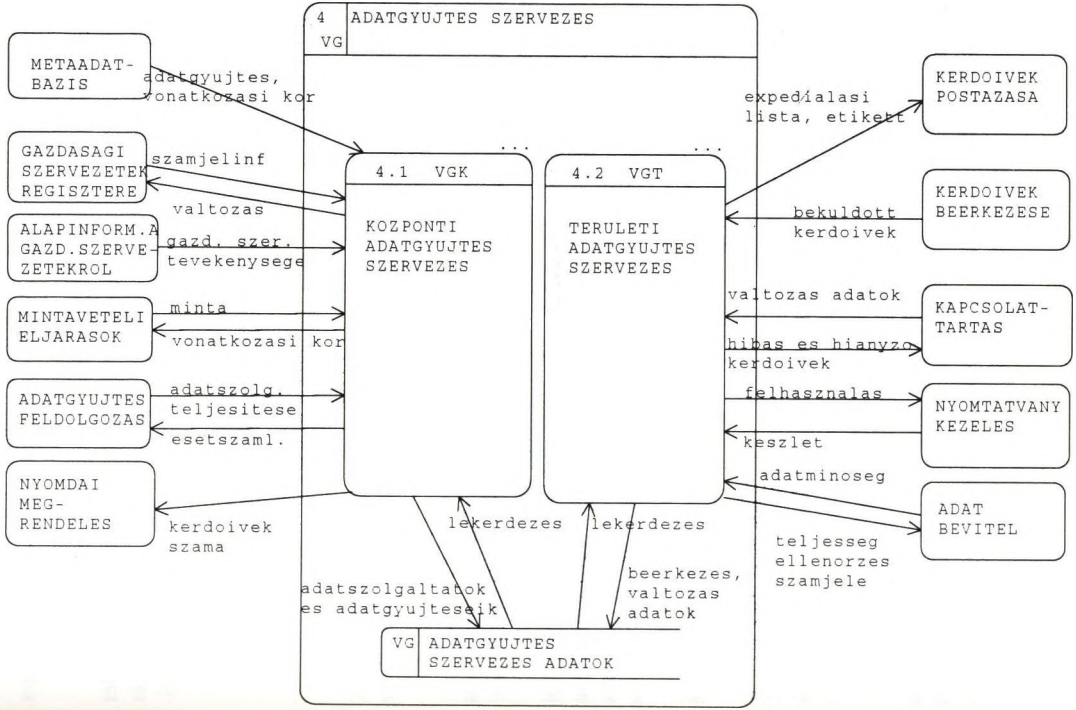
készítettünk.

A **tervezés** fázisában csak a Dictionary-t használtuk egy még szűkebb körre, csak azokhoz a funkciókhoz, amelynek a megvalósítása is napirenden van.
- Egyedek és kapcsolataik leírásából a rendszer default módon képezte a rendszer a táblákat, oszlopokat, egyedi és idegen kulcsokat, kommenteket.
- Definiálni kellett a view-kat, a kulcsokon kívüli indexeket.
- A funkcióleírásból képzett modulleírásokat tovább finomítottuk a megvalósítandó feladatnak megfelelően.

Az **implementálás** különböző eszközökre támaszkodva készült:

2. ábra

ADATARAMLASI DIAGRAM



- A tábla és view definíciókat, constraint leírásokat a CASE* Dictionary generálta.
- A képernyős alkalmazások (formok) részben vagy teljesen a CASE*Generator segítségével készültek.
- Menü az SQL*Menu-vel készült, CASE*Generator használatának a részleges megvalósítás miatt nem volt értelme.
- A riportok is CASE használat nélkül készültek az új ORACLE eszközökkel.
- Az adatbázis betöltése, aktualizálása első fázisban, a párhuzamos üzemelés feltételi között a régi adatbázisban (IBM-en MARKIV-gyel) karbantartott és onnan átvett állományokból történik. Ehhez SQL*Loadert, SQL Plus és PL\SQL eljárásokat használtunk.

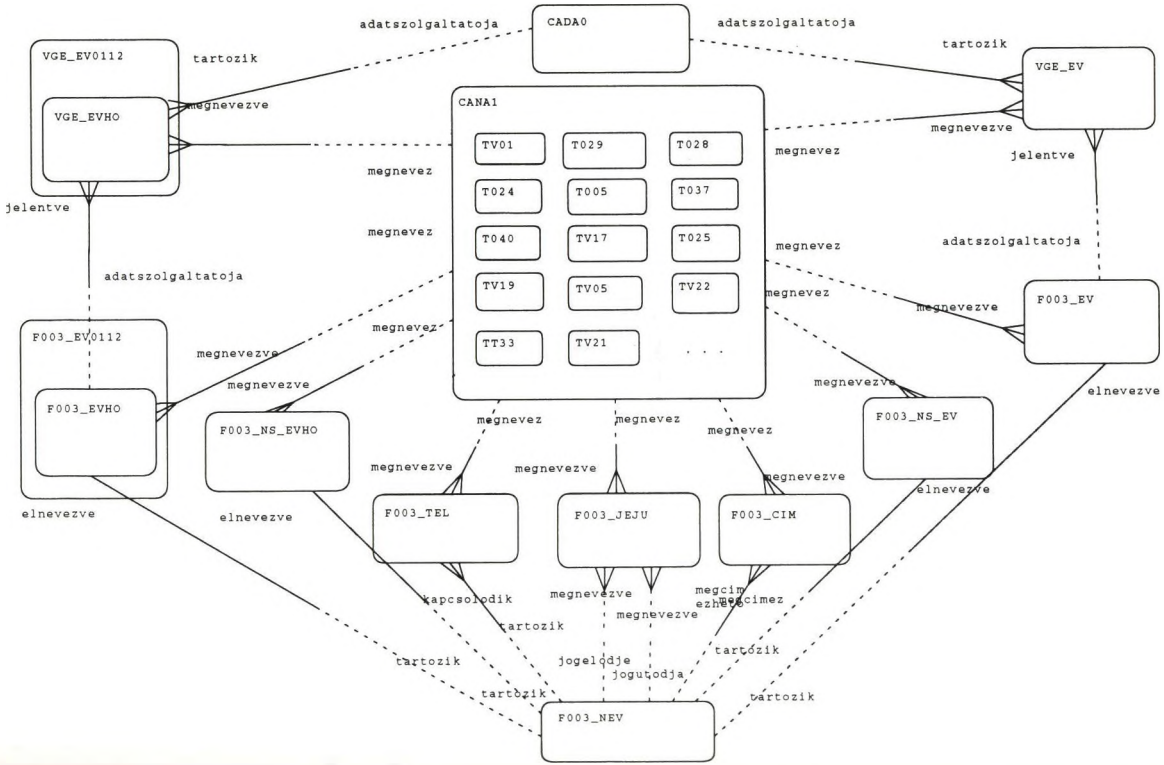
4. Tapasztalatok a rendszer tervezéséhez, CASE alkalmazáshoz kapcsolódóan

4.1 Az eszközök és környezetük

A CASE alkalmazás kezdeti fázisait az **installálás problémái** nehezítették. A HP UNIX serveren elhelyezett CASE eszközöket és adatbázist HP workstation-on keresztül illetve PC-s X terminálról terminál üzemmódban érjük el.

- Sok időt vett igénybe az egységes funkcióbillentyű kiosztás és alkalmazás megoldása a különböző eszközökre (SQL*Forms, Menu,Report, CASE*Dictionary és a CASE*Designerből hivható előbbi eszközökre).
- Nehezen törődtünk bele, hogy magyar karaktereket csak a szöveges mezöknél tudunk alkalmazni (leírások, magyarázatok, promptok). A diagramokban megjelenő szövegeket (reláció neveket, címeket, funkció leírásokat) ékezetes betük nélkül kell írni, mivel vagy a dictionary-vel való kapcsolat vagy a nyomtatás volt sikertelen.
- Különböző szélességű riportok írására nincs felkészülve a rendszer, így a riportok csak file-ból nyomtathatók.

Ide kell sorolni a **verzió problémák**at, a több lépéses lemeradást a CASE eszközök fejlesztése és az egyéb ORACLE eszközök fejlesztése között. Miközben a rendszerünk már 7-es verziójú, a CASE csak 6-os verziójú ORACLE környezetben müködött, önálló adatbázist igényelt. Igy amelett, hogy a szoftver dupla helyet igényelt, nehezítette a generált alkalmazások próbáját a másik adatbázisban levő próbaadatok elérése is. A generált utasítások (CREATE ill ALTER TABLE utasítások) is 6-os típusúak, ami a szintaktika változás miatt editálást igényel futtatás előtt. Meg sem kell említenem, hogy a CASE*Generátorok még nem készültek fel az új ORACLE eszközökre (4-es Forms, 2-es Report stb), tehát csak a régi Forms, Report generálható.


CADA0
adatszolgaltatoja
adatszolgaltatoja
VGE_EV0112
tartozik
VGE_EVHO
megnevezve
tartozik
VGE_EV
megnevezve
CANA1
TV01
T029
T028
T024
T005
T037
T040
TV17
T025
TV19
TV05
TV22
TT33
TV21
megnevez
megnevez
megnevez
megnevez
megnevez
jelentve
jelentve
adatszolgaltatoja
adatszolgaltatoja
F003_EV0112
F003_EVHO
F003_EV
megnevezve
megnevezve
megnevezve
megnevezve
F003_NS_EVHO
F003_NS_EV
megnevez
megnevez
megnevez
megnevezve
megnevezve
megnevezve
elnevezve
elnevezve
elnevezve
F003_TEL
F003_JEJU
F003_CIM
kapcsolodik
megnevezve
megcim
megcimez
megnevezve
tartozik
tartozik
tartozik
jogelodje
jogutodja
tartozik
tartozik
F003_NEV

Táblatípusok és leírásuk

Egyed Neve	Leírása
CADA0	Adatgyűjtések leírása
F003_CIM	Gazdálkodó szervezetek címadatai
F003_EV	Éves számjel állomány - gazdálkodó szervezetek leírása és ismérvei
F003_EVHO	Évközi számjel állomány - gazdálkodó szervezetek leírása és ismérvei (adott hónap)
F003_EV0112	Évközi számjel állomány - gazdálkodó szervezetek leírása és ismérvei (12 hónap)
F003_JEJU	Gazdálkodó szervezetek jogelődei illetve jogutódai
F003_NEV	Gazdálkodó szervezet megnevezése
F003_NS_EV	Gazdálkodó szervezetek éves számjele (teljes számjel)
F003_NS_EVHO	Gazdálkodó szervezetek adott havi számjele (teljes számjel aktuális hónap)
F003_TEL	Kapcsolattartás adatai
VGE_EV	Adatszolgáltatók adatgyűjtései
VGE_EVHO	Adatszolgáltatók adatgyűjtései (adott hónap)
VGE_EV0112	Adatszolgáltatók adatgyűjtései (12 hónap)

Megjegyzés :

- az EV HO a táblanevekben konkrét értéket kap pld EV helyett 94 vagy HO helyett 03 szerepel

- kivéve az un. NS állománynál ahol a HO mindig az aktuális hónapot helyettesíti,
  tehát a táblanévben pld 95HO fog szerepelni.

4.2 A tervezéssel kapcsolatos tapasztalatok

A környezet problémáin felülemelkedve a tervezést jól szolgálták a CASE eszközök. A tervezés egyes fázisaiban megismert információkkal egyre bővebb. mindig aktuális információ állt rendelkezésre. A változások, javítások konzekvensen végigvezethetők a rendszeren. Minden lényeges információ leírható a dictionary-ben.

Igy a már megtervezett rendszer teljes mélységében dokumentált.

- az egyedek és attributumaik;
- az egyedek kapcsolata;
- a domainek
- a táblák, oszlopaik, kulcsok, view-k;
- a funkciók, a funkciókban szereplő egyedek és attrubutumaik
- a funkciókhoz kapcsolódó modulok, a modulokban szereplő táblák, oszlopok, a táblák, oszlopok felhasználásának jellege és attributumai, stb.

A tervezés minőségének ellenőrzésére sokféle **riport** készíthető. A mátrix jellegű keresztreferenciákat nem használtuk, ez a rendszer enélkül is áttekinthető volt.

A több egyedben, táblában is használt attributumok, oszlopok jellemzőinek megadása a **domain**-ekben jól szolgálta a konzisztens és munkakímélő megoldást.

A domain-ekben leírhatók a lehetséges **értékkészletek** is. Sajnos azonban a jelenlegi verzióban csak CASE*Dictionary-ben történhet a bevitele és aktualizálása

- külső file-ból, táblából való feltöltése
- Dictionary-n kívüli alkalmazással való módosítása

nem lehetséges. A statisztikai rendszerben már korábban is létező értékkészletek aktualizálást nem CASE userek végzik. Újbóli rögzítésükre párhuzamos karbantartásukra nincs mód.

A funkciókhoz a tartalmi meghatározás mellett a megoldandó feladat méretére, a megoldás eszközére vonatkozóan is megadhatók információk. Korlátozott **projekttervezési** feladatok is elláthatók vele.

Kijelölhetők szervezeti egységek és a szervezeti egységekhez hozzárendelhetők a funkciók, illetve egyedek. Ez szolgál az **osztott feldolgozás** tervezésére. Bár ennek a leírása még nem történt meg a szótárban, úgy gondolom itt még vannak lehetőségek a CASE eszközök funkcionalitásának bővítésére.

4.3 Az alkalmazás generálással kapcsolatos tapasztalatok

Az adatszótárba vitt infomációkat

- egyrészt a **DDL utasítások** előállítására
- másrészt a képernyős Forms alkalmazások generálásához használtuk fel.

Az adatbázis táblák létrehozásához szükséges DDL utasításokat

- CREATE TABLE,
- COMMENT ON,
- ALTER TABLE ADD CONSTRAINT,
- CREATE INDEX,
- CREATE VIEW

kivétel nélkül az adatszótárból generáltuk. Editálást a 7-es verzióhoz szükséges szintaktika változás valamint a nem generált méret (storage) paraméterekkel való kiegészítés igényelt.

A **CASE*Generator for SQL*Forms/Menu** segítségével több alkalmazás generálására került sor. A generálásra kiválasztott formok lekérdező illetve csak kódellenőrzést tartalmazó beviteli formok voltak, mint pld:

- adatgyűjtés katalógusból a lekérdezendő adatgyűjtés és időszak kiválasztása,
- az adatgyűjtéshez tartozó adatszolgáltatók jellemző adatainak lekérdezése,
- kapcsolattartás adatainak karbantartása.

A forma képét, működését befolyásoló un. preferenciákkal gyakorlatilag tetszőleges alkalmazás generálható. Nagy időigényű viszont a preferenciák hatásának megismerése, alkalmazása.

A generálás során jelentkező problémák, módosítási igények két példája:

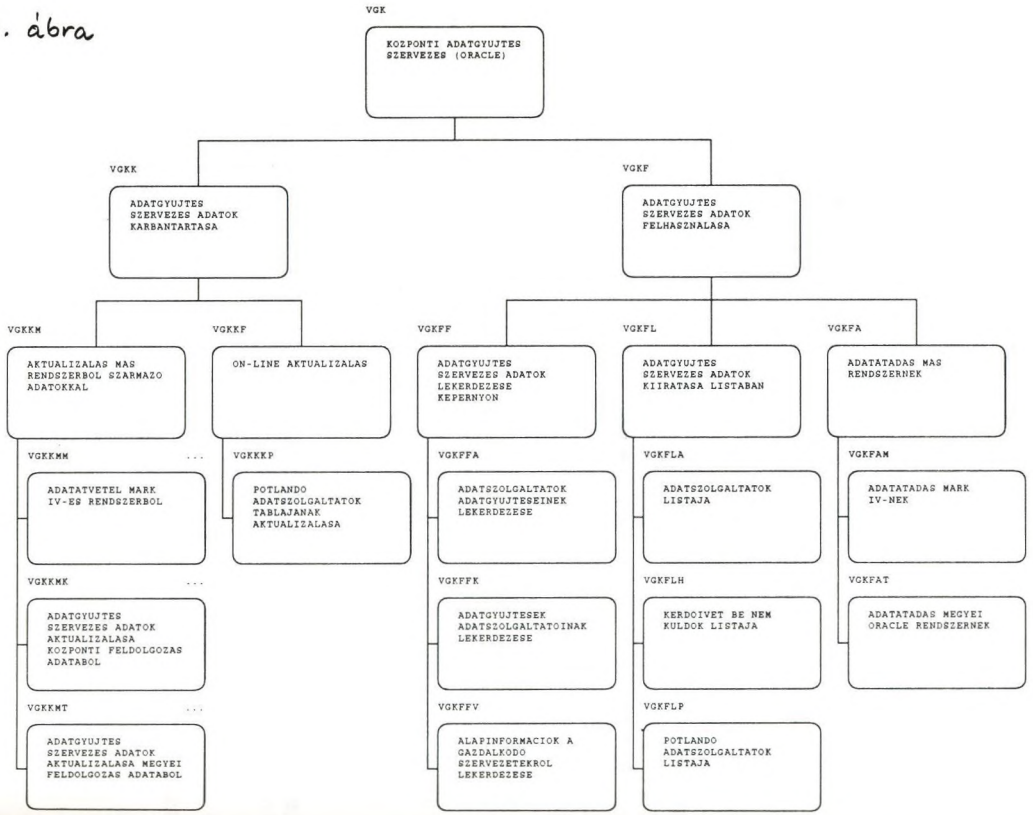
- az értékkészletek megjelenítése és
- a formok közötti értékátadás

A képernyők sok kódolt mezőt tartalmaznak, melyekhez a lehetséges kódok listáját illetve konkrét érték esetén a kódhoz tartozó megnevezést hívható ablakokban kell megjeleníteni. A generátor a domain-ek értékkészletéből képes az értéklistát megjelenítő ablakot generálni. Ezt a lehetőséget azonban nem tudtuk kihasználni az előző pontban említett probléma miatt (az értékkészletek adatszótárba vitele a mi rendszerünkben nem lehetséges). A listákat, kódok megnevezését megjelenítő ablakokkal, blokkokkal ki kellett egészíteni a formokat.

A Dictionary-vel modulhálók, modulok közötti kapcsolatok definiálhatók. Ennek alapján generálhatók értékátadással egymást hívó formok. Mi is kihasználtuk ezt a lehetőséget a kiválasztott adatgyűjtésekhez tartozó további információk kezelésére (1. form az adatgyűjtés és időszakának kiválasztása, 2. form az adatgyűjtés adatszolgáltatóit jellemző adatok kezelése). A generálás eredményeként kapott formok tartalmazzák a másik form hívásához, értékátadáshoz szükséges utasításokat, a generált globális változó neveket azonban módosítani kellett a helyes működéshez.

A generátor számos preferenciával□támogatja□mind a Dictionary-ben történt változtatás utáni részleges újragenerálást, mind az SQL*Forms-szal módosított form visszaírását a Dictionary-be, mi azonban ezt nem alkalmaztuk.

4. ábra


VGK
KOZPONTI ADATGYUJTES SZERVEZES (ORACLE)
VGKK
ADATGYUJTES SZERVEZES ADATOK KARBANTARTASA
VGKF
ADATGYUJTES SZERVEZES ADATOK FELHASZNALASA
VGKKM
AKTUALIZALAS MAS RENDSZERBOL SZARMAZO ADATOKKAL
VGKKF
ON-LINE AKTUALIZALAS
VGKFF
ADATGYUJTES SZERVEZES ADATOK LEKERDEZESE KEPERNYON
VGKFL
ADATGYUJTES SZERVEZES ADATOK KIIRATASA LISTABAN
VGKFA
ADATATADAS MAS RENDSZERNEK
VGKKMM ...
ADATATVETEL MARK IV-ES RENDSZERBOL
VGKKMK ...
ADATGYUJTES SZERVEZES ADATOK AKTUALIZALASA KOZPONTI FELDOLGOZAS ADATABOL
VGKKMT ...
ADATGYUJTES SZERVEZES ADATOK AKTUALIZALASA MEGYEI FELDOLGOZAS ADATABOL
VGKKKP
POTLANDO ADATSZOLGALTATOK TABLAJANAK AKTUALIZALASA
VGKFFA
ADATSZOLGALTATOK ADATGYUJTESEINEK LEKERDEZESE
VGKFFK
ADATGYUJTESEK ADATSZOLGALTATOINAK LEKERDEZESE
VGKFFV
ALAPINFORMACIOK A GAZDALKODO SZERVEZETEKROL LEKERDEZESE
VGKFLA
ADATSZOLGALTATOK LISTAJA
VGKFLH
KERDOIVET BE NEM KULDOK LISTAJA
VGKFLP
POTLANDO ADATSZOLGALTATOK LISTAJA
VGKFAM
ADATATADAS MARK IV-NEK
VGKFAT
ADATATADAS MEGYEI ORACLE RENDSZERNEK

4.4 A dokumentáció minősége

A CASE alkalmazással mindenkor naprakész dokumentációt lehet előállítani.

Az egyed-kapcsolat diagram egyszerű módot ad rá, hogy
- átfogó képet adjunk a teljes rendszerről,
- csak egyes alrendszereit ábrázoljuk,
- vagy az egyes funkciókhoz tartozó részdokumentációkat készítsünk.

Ezt az eszközt használtuk a legtöbbet (egy részdiagramot a 3. ábra mutat)

A funkciódiagrammal is kiemelhető a funkcióhierarchia szintek csoportja, de nem válogathatjuk ki olyan rugalmasan a kívánt funkciókat, mint az egyed-kapcsolat diagramnál. (4. ábra)

A Dictionary riportokból csak néhányat alkalmaztunk, mivel a többi túl részletes információt szolgáltatott. Probléma volt a riportok angol nyelvű címe és fejszövege. Dokumentációba illesztéskor ezért szerkeszteni kellett, ami ismételt futtatások esetén nem járható megoldás.

A Dictionary metamodeljére épülve tetszőleges riportok készíthetők, de ezt a munkát csak új CASE verzió installálása után érdemes elvégezni.

5. A CASE alkalmazás feltételei

A CASE alkalmazás nagy időigényű, sok ismeretet igénylő, ezért csak rendszeres használat esetén hatékony munkaeszköz. Fontos, hogy a befektetett munka arányban álljon az eredménnyel. Bevezetése megfontoltságot igényel.

A CASE módszertan valamennyi alkalmazás tervezésénél alkalmazható. Nagyobb feladatoknál azonban a módszer használatát az eszközök használatával is támogatni kell.

Valószínüleg egy egészséges kompromisszum adja a megoldást:
- valamennyi feladatnál a CASE módszertan alkalmazása,
- egyszerübb feladatoknál elsődlegesn a designer nyújtotta lehetőségek igénybe vétele,
- és összetettebb, átfogóbb projekteknél a teljes eszközkészlet alkalmazása.

A munka sikeréhez különböző, de valamilyen szinten CASE-ben járatos szakemberek értő együttműködése szükséges:
- a CASE-t alkalmazó tervezőn kívül,
- sok alkalmazás esetén CASE adminisztrátor,
- nagyobb alkalmazások esetén részletes modulspecifikációt készítő tervező,
- a CASE specifikációt értő programozó,
- a tervek elfogadásához a dokumentációt értő megrendelő.

Ez viszont csak széleskörű oktatással képzelhető el.

Az országos közigazgatási alapnyilvántartások működésének egyes sajátosságai

A közigazgatási országos alapnyilvántartások működését törvények szabályozzák. Általában a szabályozás kiterjed az olyan kérdésekre is, mint pl.:

- az input bizonylatok
- a szöveges és (tér)képi nyilvántartások/ aktuális és történeti adatbázisok
- a nyilvántartás egyedtipusainak azonosítói/ kódjai
- az output információ szolgáltatási szabályai/ naplózása
- a nyilvántartás input bizonylatok nélküli elérése, stb.

Megjegyzendő, hogy egyetlen alapnyilvántartásra is általában több különböző jogszabály hat (pl. földadózási, építési engedélyezési, stb. vonatkozású jogszabályok a telekkönyvre). Ki kell emelni azt is, hogy számos országban az önkormányzatok a saját illetékességi körükön belül hoznak határozatokat, rendeleteket, amelyekben az országos nyilvántartás egyedtípusainak általános értelmezésétől eltérő fogalmakat, esetleg más értelmezésű egyedtípusokat, tulajdonságtipusokat használnak (pl. az építés éve, stb.). Teljesen lehetetlen az ilyen fogalmi különbségeket úgy megszüntetni, hogy a közigazgatási információrendszerben előforduló minden tulajdonságtípusra kiterjedő összhangolási, szabványosítási tevékenységet végezzünk, hiszen rengeteg adatféleség és fogalom merülne fel. Azonban néhány alapvetően fontos, az ország közigazgatásában mindenhol ugyanúgy értelmezendő fogalomra -egészen annak egységes kódolásáig bezárólag - kötelező szabványos értelmezést, formát és használatot kell előírni.

A közigazgatási feladatok azon része (pl. valamely közmű javítása miatti útfelbontás, stb.) amely széleskörű koordinációt is magábafoglal, azt igényli, hogy a koordináció által érintett szervezetek folyamatosan egy közös ú.n. postaládába (mailbox) juttassanak olyan szöveges adatokat, mint pl. a munka fajtájának, megkezdésének, várható befejezésének az adatai. Egy megadott időpontra előretekintve a kérdéses útvonalszakasz vonatkozásában, így lehetségessé válik a közműépítési munkálatok olyan összhangolása (egy ütemezést optimalizáló programmal), hogy a közlekedést minél kevésbé akadályozza.

Az adatok vonatkozásában ki kell emelni, hogy a közigazgatási nyilvántartások adataira általában az jellemző, hogy nem egyetlen szervezet vagy szervezeti

egység tart rájuk igényt. Viszonylag ritka az olyan adat, amelyet csak egyetlen szervnél, vagy osztályon használnak. Ebből is következik, hogy akár az egész közigazgatást, akár csak egy szintjét, vagy szervét nézzük is (pl. önkormányzati információs rendszer) annak középpontjában mindig egy közös adatbázis áll. A közös adatbázis létrehozásának viszont lényegesen több előfeltétele van, (pl. hatékony adatcsere és koordinációs mechanizmusok szükségesek), mint egy nem közös használatú adatbázis esetén.

Az egyedtípusok vonatkozásában a szegmentáltság lesz jellemző. Egyrészt az osztott használat, másrészt az adatkarbantartási jogosultságok is abba az irányba hatnak, hogy egy "atomizált" adatmodell jöjjön létre. Azon objektumok részletes leírása, amelyek geokódoltak, ezt a tendenciát még jobban növeli. Mindez oda vezet, hogy az egyes szegmenstípusokat is egyedi azonosítótípusokkal lássuk el, amelyek segítségével a láncolás, stb., lehetséges. Összetett, új egyedtípusok hozhatók létre pl. úgy, hogy két egyedtípus egy-egy egyedaltípusának bizonyos előfordulásait láncoljuk össze. Ez a különböző alkalmazások szempontjából jelenthet célszerű megoldást

Minden egyes tulajdonságtípushoz (vagy szegmenstípushoz) az adattartalom pontos vagy becsült voltának jelzésére célszerű lehet felvenni a tűrésmezőre jellemző adatot is. Meglehet, hogy erre csak bizonyos alkalmazásoknak lesz szükségük. Úgyszintén jelölni kell az adatfelvétel/módosítás időpontját is.

Célszerű még az is, hogy az adatkarbantartás input tételeinek összehangolt kezelésére egy mailbox-rendszer gyűjtse be mindazon üzeneteket, amelyek a térbelileg közelálló objektumokra vonatkoznak. Ez segíti a felhasználót, hogy a közel elhelyezkedő objektumokra vonatkozó változásokat időben is rendszerezettebben át tudja tekinteni.

Az adatgyűjtő rendszer kialakításánál az adatgazdai hatásköröket, egyúttal a hozzáférési jogosultságokat is szabályozni kell.

A grafikus, különösen a térinformatikai adatok kezelése speciális problémákat vet fel, de ezekkel a jelen értekezés nem foglalkozik (v.ö. Mezey + Mezeyné 1995).

Az országos alapnyilvántartások nagy "iratüzemek", amelyek munkafolyama jól strukturált, szervezetileg pedig funkcionálisan, azon belül elsősorban a munkafolyamat (ügymenet) alapján csoporosított. Ennek ellenére azonban az ügymenetek sajnos általában nincsenek jól dokumentálva, ezek modellezésére szükség van.

2. Alapnyilvántartás adatgyűjtése és adattárolásának gépesítési irányai

Vizsgáljuk ezt a népességnyilvántartás példáján keresztül, bár megállapításaink más alapnyilvántartásokra is adaptállhatók alkalmazhatók.

Áttekintve a nemzetközi szakirodalmi forrásokat azt érzékelhetjük, hogy a népességnyilvántartás számos országban még nem, vagy alig gépesített. Ennek egyik oka az, hogy sok fejlődő országban még a népmozgalmi események (születés, halál, házasság stb.) észlelése és helyi regisztrálása is lényegében alig megoldott (pl. Afrika, Ázsia stb.). A lakcímváltozások bejelentése ugyancsak jónéhány országban nem kötelező. Egy másik ok a gépesítéshez, illetve a velejáró távközlési adatátviteli hálózat fejlesztéséhez szükséges jelentős beruházás, amelyet az állami költségvetésből a téma viszonylag kis prioritása miatt nehéz kiharcolni. Erre utal az a tapasztalat, hogy a népességnyilvántartás ott fejlődött aránylag erőteljesen, ahol közvetlenül a belügy (esetleg részben a pénzügyminiszter) alá tartozik. Ez világszerte az esetek kb. harmada, míg az egészségügy (kb. negyedrész) rosszabb, az igazságügy, illetve bíróságok vagy a statisztikai hivatal alá rendelés (hatodrész) még gyengébb fejlesztési lehetőségeket jelent általában. Maga a gépesítés lényegében véve négy fő irányt, technikai alternatívát jelent elméletileg:

1. A népességnyilvántartási alapiratok/bizonylatok papíreredetiben való tárolása mellett másolatok készítéséhez, küldéséhez (fax) szükséges gépesítés, esetleg kiegészítő jelleggel a visszakeresés számítógépes segítése.

2. Részben helymegtakarítás miatt, részben mert a mikrofilmet az adott országban az eredetivel azonos bizonyító erejű iratnak fogadják el, mikrofilmtekercs vagy mikrofilmlapos technika alkalmazása, gyakran kiegészítő jelleggel a visszakeresés számítógépes segítése is.

3. Részben a működési hatékonyság növelése miatt, részben mert az optikai lemezt (WORM-ot) az adott országban az eredetivel azonos bizonyítóerejű iratnak fogadják el, optikai tárolók alkalmazása, ahol kötelezően megjelenik a számítógépes visszakeresés.

4. Tekintettel arra, hogy tömeges visszakeresések, vagy a statisztikai feldolgozások céljára nem elegendő csupán az eredeti iratról felvételezéssel egy hasonmás kép előállítása, sok népességnyilvántartás gépesítése hagyományos adatfeldolgozási szemlélettel kezdődött és nem alkalmaz sem mikrofilmet, sem optikai lemezt.

A gyakorlatban sokszor a négy fő irány vegyülékei észlelhetőek. Például hazánkban évtizedeken át a népességnyilvántartás feladatát lényegében ellátta a

BM AH LKO, ahol az egy személlyel életciklusa során történt igazgatási események iratait (vagy arról értesítést) tárolták. Mivel innen vagy egyedi információt adtak, vagy hiteles iratmásolatot, elegendő volt az 1. technikai fő irány követése is, azonban számítógépesítés híján nem tudta szolgáltatni a statisztikához, tervezéshez szükséges adattömegeket. A KSH felügyelete alatt hozták ezért létre a 4. fő irányban (hagyományos adatfeldolgozás) gépesített ÁNH-t a 70-es években, amelynek a 90-es évektől a BM alá rendelése lehetővé teszi a két addig független szervezet, de mégis szorosan összekapcsolódó funkciók, és azonos alapadatok illetve bizonylatok összehangolt feldolgozását és egységes szolgáltatási felület létrehozását. Ez az eddig elkülönült folyamatok és eltérő technikai megoldások integrálását jelenti.

A két szervezet együttes működésének szoros koordinálása természetesen jelenti a jogi szabályozás, a szervezeti és működési szabályzatok, még alapvetőbben azonban a működés folyamatainak és az eltérő technikai feltételrendszernek az egybehangolását. Ez pedig az 1., a 4., de a hatékony működés miatt 3., a gazdaságos és biztonságosabb működés miatt pedig a 2. technikai fő irányok, megoldási módszerek legrészletesebb összehangolását igényli.

Ha az 1. irányban működő rendszerről akár a 2., akár a 3. vagy a 4. fő irányba akarunk áttérni, a fő gond a nagytömegű múltbeli irat tartalmának gépbe vitele lesz. Ha a régi iratokat hagyjuk és egy határnaptól csak az új iratok gépreviteléét határozzuk el, a fő probléma a régi papír és az új gépi rendszer együttműködtetése. Ez pl. azt jelenti, hogy a papíriratárban ülő személy egy terminál képernyőjén "belelát" a gépi nyilvántartásba, de a számítóközpontban ülő személy egyáltalán nem "láthat bele" az irattárba, ezért bár az adatszolgáltatáshoz szükséges információ egy része gépen hamar megvan, az ugyancsak szükséges másik részt csak ugyanolyan lassan lehet megkapni, mint ahogyan gépesítés nélkül régebben történt.

Cél az, hogy a két fenti nyilvántartás idővel minél szervesebben kapcsolódhasson úgy, hogy belőlük való szolgáltatás esetén az ügyfél számára ne tűnjék ki, hogy itt hány szervezet és nyilvántartás működik, hanem egyetlen és egységes szolgáltatási felületet érzékeljen. Cél az is, hogy az adatgyűjtési folyamatokat integráljuk, ennek főleg gazdaságossági okai vannak.

- A multbeli fejlődés trendjét az 2. pont összefoglalta.
- A jelenlegi hazai helyzetről áttekintést ad (Mezey - Mezeyné 1995.)
- A jövőbeli továbbfejlesztést meghatározó alapvető tényező egyszerüen megfogalmazható:
  a digitalizált, adatbázisban kezelt
  adattartalom bővülése szinte 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 az országos közigazgatási alapnyilvántartások már működő adatfeldolgozó rendszere szöveges adatbázisai mellé képi adatbázisok illesztése úgy, hogy azok a szöveges adatbázistól függetlenül is tudjanak működni, mégse alakuljon ki két egymással konkurráló, párhuzamos adatbázis (szöveges és képi), tehát az adatbázisokat integráljuk.

Várható, hogy a későbbiekben a képi adatbázisokon kívül (amelyek nem mozgó képeket tárolnak), egyrészt mozgó képanyagot, másrészt hangfelvételeket tároló multimédia adatbázisokat is kell a szöveges adatbázisokhoz illeszteni.
Erre azonban majd csak a későbbi jövőben kerül sor, hiszen előtérben ma az iratok korszerü kezelési lehetőségei állnak.

Amerikai felmérések szerint egy szervezet információinak átlagosan 95%-a papíron érkezik be. Azt már a szervezeten belül lehet megállapítani, hogy évente mennyit költenek fax-papírra, nyomtatványokra, fénymásolatokra, stb.
Magyarországon ez az arány valószínüleg 97-98% körüli lehet a közigazgatásban.

.2.1. Képi adatbázisok

Amikor egy már hosszabb ideje működő adatfeldolgozó rendszerhez egy olyan új (jelen esetben: grafikus) alrendszert kell illeszteni, amelynek képi adatbázisa papír-iratok digitalizált képeit tárolja, képfelvételre és továbbításra alkalmas eszközöket, adathordozókat, berendezéseket kell beszerezni és a digitalizált grafikus adatok (a továbbiakban: képek) gyüjtésére, átvitelére szolgáló információs csatornát nyitni. Ugyanakkor azonban a gazdaságosság azt követeli, hogy ezt az új iratgyűjtő rendszert a régi adatgyüjtő rendszerrel integráljuk.

Hasonlóan az adat- és az iratgyüjtés egybe: irat-és adatgyűjtő rendszerbe integrálásához, valamint a szöveges és a képi adatbázis integrálásához, célszerü a két adatbázisból igényelt információkat kifelé egyetlen szolgáltatási felületet képezve szolgáltatni, azaz a kétféle szolgáltatást nyujtó működést integrálni.

Egy országos alapnyilvántartás továbbfejlődésének egyik fő iránya a hagyományos adatfeldolgozó rendszeréhez iratkezelő rendszer (DIP= document image processing) illesztése, azaz integrált adat- és iratgyüjtő rendszer, integrált szöveges és képi adatbázis, és egységes szolgáltatási felület kialakítása. Ez a fő irány a multimédia rendszerek irányába mutat majd később tovább.

2.2. Történeti adatbázisok

A közigazgatási országos alapnyilvántartások továbbfejlődésének másik fő iránya a számítógépes történeti adatbázisok és temporális adatbázisok kialakítása. Ismertek a közigazgatás nagy hagyományu statisztikai vagy pénzügyi idősorokat kezelő adatbázisai, ezek azonban általában a rendszeresen végrehajtott ú.n. szinkronizált adatgyűjtések adatait tartják nyilván. Ugyanakkor a közigazgatási alapnyilvántartások a nyilvántartott objektumról mindeddig csak a legutolsó ismert állapotát kifejező adatrekordját tudták kezelni. Az egyedre vonatkozó régebbi állapotait kifejező adatok hatékony kezelését (a terjedelmes történeti adatállományok miatt) általában nem tudták számítógéppel végezni,hanem egészen hagyományos módon működő okmány- és okirattárak látták el.

Meg kell jegyezni, ezek nem- szinkronizált változásjelentések iratait tárolják. Időszerüvé most válik az alapnyilvántartásokra annyira jellemző, aszinkron működésü, eseményvezérelt, történetiséget kifejező adatbázisok szervezése.

Ezeknek az előzőként említett fő irányvonalhoz szoros közük van két szálon is:

- a képi adatbázishoz a multbanyúló alapbizonylatok, papíriratok is rendelkezésre állnak,

- a képi adatbázis katalógusaként célszerüen a működő szöveges adatbázis rendelkezésre áll.

.2.3. Elosztott adatbázisok

A közigazgatási országos alapnyilvántartások továbbfejlesztésének harmadik fő iránya a fokozatos dekoncentrálás, távlatilag pedig decentralizálás.

A közigazgatásnak (Mezey 1980, és Mezey 1979) az eszközöket és adatokat a centrumba koncentráló országos alapnyilvántartások helyett olyan osztott adatbázisokra van szüksége, amelynek decentrumai a polgármesteri hivatalok, a helyben szükséges adatokat ott tárolják és az országos központban néhány alapvető adat mellett olyan információk vannak csupán, hogy pl. mely adatot hol lehet megtalálni. Az adat-ellenőrzéshez szükséges etalonok jelentős része szintén ott áll majd rendelkezésre. Ez illeszkedik az önkormányzati rendszer sajátosságaihoz.
A különböző profilu alapnyilvántartások pedig egymástól lehetőleg függetlenül, az összekapcsolódás kényszerét nélkülözve tudjanak működni. Erre világosan utal a személyi szám alkalmazásának egy szűk körre behatárolása, külön TB- és külön adóazonosítójel bevezetése is.

Ami a történeti adattárakat, archivumokat illeti azok hagyományosan központi elhelyezése is részben oldódhat és decentralizálásuk később lehetséges.

Az eddig felsorolt 3 fő irány követése számos következményt is kivált, igy pl.:
- az információs rendszerek korszerü tervezési és dokumentálási módszereit. Magának az adatminőségirányításnak a tervezése, módszere ugyancsak ide tartozik.

- a csoportmunka hatékony szervezéséhez, minőségbiztosításához, és a vezetői tevékenységhez, egyaránt szükséges munkafolyamvezérlést, stb.

Az egyre bonyolultabbá váló, elosztottságuk mellett is mind összetettebb és mind több adatot átfogó alapnyilvántartások áttekintése, egységes dokumentálása, tervezése azért is nélkülözhetetlen, hogy pl. elkerüljük adatok többszörös tárolását, ismételt kezelését.

.2.4. Munkafolyam vezérlése

A munkafolyam vezérlés az adatminőség, a működésellenőrzés, a vezetői belső információs rendszer szinvonalának emelése irányába végezhető fejlesztés, amely gyakran párosul a szolgáltatási felület és az adatvédelem (data protection) hatékony kialakításával, sőt a csoportmunka és a szervezet újraszervezésére irányuló racionalizálással. Ez az irány az ügyviteli/ügyintézési folyamatok szervezésére, a csoportmunka szervezésére szolgáló groupware felhasználásával (Chroust - Benczúr ed. 1994) válik járhatóvá.

A működési folyamatok jobb megszervezését célzó legújabb módszertanok:

- Process Reengineering (működés újraszervezése) (Hammer 1990)
- Process Evolution (folyamatok továbbfejlesztése) (Davenport 1993),
- Continuous Quality Improvement (TQM+QWL = Quality of Working Life (Keen))

A munkafolyam és a munkafolyamvezérlés alapfogalmainak definicióját (Ellis 1994) felsorolja és leírja a munkafolyam statikus modellezésére alkalmazott módszereket: - Adatmodellezés
- Tevékenység modellezés
- Folyamat modellezés
- Célok, tervek és korlátozó feltételek által orientált modell
- Szervezeti, társadalmi oldal modellezése

A modell dinamikájának leírására alkalmazott módszerek:
- Petri hálók
- ICN (Information Control Net)
- Egyed/tranzakció modell
- Speech/Act modell
- Team automata

Az azonosítókkal szemben néhány általános alapkövetelményt állíthatunk:

a/ Egyértelmü és kölcsönös megfelelőség az azonosítandó objektumok halmazának elemei és a hozzájuk rendelt azonosítóik halmazának elemei között. Ekkor egyedi az azonosító.

b/ Az objektumot az életciklusa kezdetén kapott azonosító életciklusa végig kíséri(azonosítja), tehát az azonosító nem függhet az objektumot életútján ért változásoktól, hanem csak az objektum élete(léte) során végig változatlan adatok jellemezhetik.

c/ Az alkalmazása közbeni adatátvitelre és konverziókra tekintettel legyen hibafelismerő (illetve lehetőleg hibajavító) tulajdonságu kód.

Tervezési módszer

Bizonylatok esetére a bizonylatazonosító tipusának, kiosztása időzítésének és helyének (centrum-decentrum), sőt a felviteléhez szükséges technológiai variánsnak a költségeket is számbavevő kiválasztására tervezési módszert dolgoztunk ki.

Az a célunk, hogy kijelöljük egyrészt azt, hogy melyek legyenek a bizonylatazonosító tulajdonságai, információtartalma, másrészt azt, hogy megmondjuk, hogy az iratgyűjtő rendszerben, vagy azon kívül (pl. a nyomdában) és a bizonylat életciklusának mely eseménye alkalmával, továbbá centralizáltan vagy decentralizáltan osszuk-e ki a bizonylatazonosítót. Ehhez az alábbi tervezési módszert követjük

1. Határozzuk meg, hogy mely információ(ka)t hordozzon célszerüen maga a bizonylatazonosító, hogy az iratfeldolgozó rendszerben az adatminőség szempontjából fontos funkciók ellátását hatékonyá tegye.

Az azonosítókkal szemben néhány általános alapkövetelményt állíthatunk:

a/ Egyértelmü és kölcsönös megfelelőség az azonosítandó objektumok halmazának elemei és a hozzájuk rendelt azonosítóik halmazának elemei között. Ekkor egyedi az azonosító.

b/ Az objektumot az életciklusa kezdetén kapott azonosító életciklusa végig kíséri(azonosítja), tehát az azonosító nem függhet az objektumot életútján ért változásoktól, hanem csak az objektum élete(léte) során végig változatlan adatok jellemezhetik.

c/ Az alkalmazása közbeni adatátvitelre és konverziókra tekintettel legyen hibafelismerő (illetve lehetőleg hibajavító) tulajdonságu kód.

Tervezési módszer

Bizonylatok esetére a bizonylatazonosító tipusának, kiosztása időzítésének és helyének (centrum-decentrum), sőt a felviteléhez szükséges technológiai variánsnak a költségeket is számbavevő kiválasztására tervezési módszert dolgoztunk ki.

Az a célunk, hogy kijelöljük egyrészt azt, hogy melyek legyenek a bizonylatazonosító tulajdonságai, információtartalma, másrészt azt, hogy megmondjuk, hogy az iratgyűjtő rendszerben, vagy azon kívül (pl. a nyomdában) és a bizonylat életciklusának mely eseménye alkalmával, továbbá centralizáltan vagy decentralizáltan osszuk-e ki a bizonylatazonosítót. Ehhez az alábbi tervezési módszert követjük

1. Határozzuk meg, hogy mely információ(ka)t hordozzon célszerüen maga a bizonylatazonosító, hogy az iratfeldolgozó rendszerben az adatminőség szempontjából fontos funkciók ellátását hatékonnyá tegye.

A jelen anyagban természetesen az adatminőség szempontjai dominálnak, de külön fel kívánjuk hívni arra a figyelmet, hogy ezeket kiegészíthetjük (sőt felcserélhetnénk) más célszerüen megválasztott (így pl. adatvédelmi, stb.) szempontokkal, ha az egy másik információendszer szervezésénél fontosabb kritérium. A tervezési módszer tehát rugalmasan fogadhat be eltérő kiinduló helyzeteket.

Jelen esetben azért vettük fel éppen az alábbi 3 szempontot, mivel az első kettő az iratbiztonság, a harmadik pedig az adatbiztonság és az adathelyesség az adatminőséget közvetlenül befolyásoló tényezők. Természetesen még több kiegészítő szempontot is rögzíthettünk volna ezesetben is, de a tervezési módszer bemutatásához - úgy gondoljuk - ennyi elegendő.

Figyelembevéve tehát, hogy egy bizonylatazonosító rendelkezik-e az alábbi tulajdonságokkal:

- utal a bizonylatköteget a központba felküldő decentrumra,
- segíti a bizonylatköteg teljességének ellenőrzését,
- utal a bizonylat egyedi tartalmára,

a bizonylatazonosítók 8 tipusa különböztethető meg. Közülük az alábbi 3-nak nagyobb gyakorlati jelentősége van (a többi öttel a jelen értekezés nem foglalkozik):

Az első ("A"-tipus) a központi (pl.nyomdai) előállításkor (és nem az adat- és iratgyüjtő rendszerbeni iratforgalomban) igen gazdaságosan vihető fel a bizonylatra. Ez fontos gyakorlati szempont.

Egy másik esetben "A" tipusu bizonylatazonosító a bizonylatnak a központi okmánytárba való beérkezésekor szintén központilag vihető fel az alapiratra.

A második ("B"-tipus) az iratbiztonságtól elválasztva egymaga képviseli csak az adatbiztonságot és adathelyességet kifejező szempontot.
A közigazgatás információs rendszerében több irat ír(hat)ja le ugyanazt az eseményt. Gyakori, hogy ezen iratok más tipusu nyomtatványok, amelyeket egymástól eltérő iratazonosítókkal látnak el.

Ugyanakkor azonban (legalább részben) ugyanarról az eseményről rögzítenek adatokat. Van arra felhasználói igény egyébként is, hogy pl. egyazon eseményt különböző oldalról tükröző iratokat összevessenek, de egy szervezeten és adatgyüjtő rendszeren belül az adatbiztonság és adathelyesség szempontjából ez jó keresztellenőrzési lehetőség. A szükséges iratokat úgy lehet legkönnyebben összegyűjteni, ha a bennük tartalmilag közös elemeket már az iratazonosító tartalmazza.

A harmadik("C"-tipus) az adatbiztonságtól elválasztva egymaga képviseli csak az iratbiztonságot kifejező szempontot. Ott, ahol szükséges az iratforgalom "könyvelése", hagyományosan alkalmazzák ezt a tipust, mint a legegyszerübb megoldást.

"A"-tipusu bizonylatazonosító jellemzői:

- a bizonylat egyedi tartalmára nem utal,
- a bizonylatköteg teljességének ellenőrzését közvetlenül nem segíti
- a bizonylatköteget a központba felküldő decentrumra nem utal

"B"-tipusu bizonylatazonosító jellemzői:

- utal a bizonylat egyedi tartalmára
- a bizonylatköteg teljességének ellenőrzését közvetlenül nem segíti

"C"-tipusu bizonylatazonosító jellemzői:

- a bizonylatköteg teljességének ellenőrzését közvetlenül támogatja
- a bizonylatköteget a központba felküldő decentrumra utal,
- a bizonylat egyedi tartalmára nem utal.

Határozzuk meg, hogy egy bizonylat (papírirat) teljes életciklusát tipikusan mely jelentősebb események jelölik meg. Ezek az alábbiak:

1. Nyomtatvány ürlap nyomdai előállítása.

2. Elárusitóhelyekre, decentrumokba kiszállítása.

3. Decentrumokban a polgár az ürlapot kitölti és a bizonylatot leadja.

4. Decentrumban a bizonylat kódolása, adatrögzítése.

5. Bizonylatköteg továbbítása területi (megyei) szintre.

6. Bizonylatköteg továbbítása központi szintre.

7. Bizonylatok központi irattárba (okmánytárban) beérkeztetése, lerakása.

Iratazonosítót vagy az 1. (A tipusu iratazonosító),
vagy a 4.("B"- vagy "C"- tipusu)
vagy a 7. ("A"- tipusu iratazonosító)
események alkalmával lehet kiosztani praktikusan.

Az iratazonosító kiosztása alapvetően vagy központilag, (A tipus) vagy (B és C tipus) decentralizáltan történhet.

Központilag két alapeset van:

a) Még a nyomtatványoknak a decentrumokba való kiküldése előtt azonosítóval látjuk el az iratot.

b) Csak az iratoknak a decentrumokból való beérkezése után látjuk el azonosítóval az iratot.

Ezeken kívül létezik a központi és decentralizált megoldás elemeit ötvöző alábbi két változat is:

c) A központilag kinyomtatott iratazonosítókat (pl. etiketten) a decentrumoknak előre kiosztják és ott ragasztják fel az iratokra.

d) Létezik végül az iratazonosítók decentralizált kiosztása.

Az alapesetek alesetei (lásd a 1. táblázatot):

a) két aleset létezik:
aa) az iratazonosító a nyomdában kerül rá a nyomtatványürlapra, így azt teljes életciklusán elkiséri,
ab) az iratazonosítót etikettre központilag kinyomtatják és ugyancsak központilag ragasztják fel, még az így felszerelt nyomtatványürlapoknak a decentrumokba való szétküldése előtt.

b) két aleset van itt is:
ba) az iratazonosítót az iratra rápecsételjük vagy printerrel rányomtatjuk
bb) az iratazonosító etikettjét kinyomtatjuk és felragasztjuk az irat beérkezése után

c) ez az ab) és bb) alesetek kombinációja.

d) ugyancsak két aleset létezik:
da) az iratazonosítót az iratra rápecsételjük, vagy kézzel írjuk rá.
db) az iratazonosító etikettjét kinyomtatjuk és felrasztjuk az iratra még a centrumba való felküldése előtt.

1. táblázat

Bizonylatazonosító kiosztás lehetőségei

	Centralizáltan					Decentralizáltan	
	a			b	c	d	
	aa	ab	ba	bb		da	db
A	+	+	+	+	+	-	-
B	-	-	+	+	+	+	+
C	-	-	+	+	+	+	+

3. Hozzuk létre a 2. táblázatot

Megállapítható, hogy a bizonylatot azonosítóval ellátni az 1., a 4., és a 7. események alkalmával lehet. Ha megkülönböztetjük még azt, hogy bizonylatazonosító kiosztása (a nyomtatványra felvitele) központilag vagy decentrumokban történik, és figyelembe vesszük azt, hogy a bizonylatazonosító "A", "B", vagy "C" tipusu, akkor a 2. táblázatot kapjuk. E táblázatból kitünik, hogy "A" tipusu bizonylatazonosítót csak központilag, míg "B" és "C" tipusu bizonylatazonosítót decentrumokban, de központilag is lehet kiosztani.

Bizonylatazonosító- kiosztás lehetőségei

	Centralizáltan				Decentralizáltan		
	a		b		c		d
	aa	ab	ba	bb		da	db
A	1	1	7	7	1	-	-
B	-	-	7	7	3	3	4
C	-	-	7	7	4	3	4

2. táblázat

Összevonva a 2. táblázat egyes sorait ill. oszlopait, kapjuk:

	a	b	c	d
A	1	7	1	-
B-C	-	7	4	4

3. táblázat

4. Haladjunk végig a fenti 3. táblázat b, c, és d alapesetein szisztematikusan. Ehhez más adat- és iratfeldolgozó rendszerben felhasználható módon, kidolgoztuk további ~~két~~ táblázatok egységes strukturáját:

Decentralizált iratazonosító kiosztás C típus

A decentralizált bizonylatazonosító kiosztás (adatelőkészítés, kódolás) valamint az azt követő manuális illetve automatizált adatrögzítés lehetőségei és összefüggései az adatelőkészítés és kódolás eszközeivel:

3. ~~két~~ táblázat

C típusú	A bizonylatazonosítóval ellátás az adatrögzítéshez képest időben					
Bizonylatazonosító kiosztás	I. Előre történő		II. Utólag történő		III. Adatrögzítés közbeni	
Adatelőkészítés, kódolás eszközei / Adatrögzítés ütemezése	Manuális	Automatikus	Manuális	Automatikus	Manuális	Automatikus
	a d a t r ö g z í t é s					
Vonalkódnyomtatás / számítógéppel						
Numerikus karakterek nyomtatása / számítógéppel						
Kézi sorszámozógép / pecsételés						
Numerikus karakterek / kézírás						

A legelőnyösebb megoldás kiválasztása előtt egyrészt a bizonylatazonosító-kiosztás műveletét az adatrögzítés (amelynek manuális és automatikus változatait egyaránt figyelembevesszük) művelete előtti, közbeni, vagy utáni bizonylatazonosító-kiosztásra javasolt alternatívákra szervezhető folyamatok teljes körét áttekintjük (lásd a táblázatok oszlopait). Másrészt az adatelőkészítés és kódolás eszközeinek figyelembevétele megnöveli a technológiai változatok számát (lásd a táblázatok sorait).

5. Határozzuk meg a fenti táblázatokban az egyes folyamatváltozatokhoz és technológiai alternatívákhoz szükséges üzemeltetési és beruházási költségeket, majd rendeljük azt az egyes táblázatok rovataihoz. Az adatminőség szempontjából releváns szempontokat szövegesen értékeljük külön.

6. Amennyiben csak az adat- és iratgyüjtő rendszerben áramló iratok bizonylatazonosítóját és kiosztását kívánjuk megtervezni, akkor válasszuk ki a gazdaságosság és az adatminőség szempontjait mérlegelve a legalkalmasabb változatot. Ezzel egyben az iratazonosító kiosztásának valamennyi aspektusát is meghatároztuk.

7. Amennyiben azonban nem csupán a gyüjtőrendszer, hanem a változásjelentés nyomán a központi nyilvántartásának megváltozása következtében egy választ (vagy szolgáltatást) kell visszaküldeni a változást jelentő decentrumba, akkor a tervezési eljárást tovább folytatjuk úgy, hogy azt kiterjesztjük az információrendszer szolgáltatást (vagy választ) küldő szakaszában áramló iratazonosítókra is.

A relációs adatbázisokban tárolt adatok felhasználása a COMSHARE vezetői információs rendszerben

Durugy Gabriella / Bánkuti Zoltán

(SzKI Kft.)

Az előadás célja megmutatni, hogy miként illeszkedik a COMSHARE[1] vezetői információs rendszere a vállalatok által használt pénzügyi, munkaügyi, stb. célrendszerekhez. Hogyan kerül a csizma az asztalra? Ezek a célalkalmazások, melyek a vállalatok adatait tartják karban, jórészt valamilyen *relációs adatbáziskezelőt* használnak, azokra írodtak; itt kapcsolódik tehát ez a rövid dolgozat a konferencia témájához.

A vállalati információs rendszerek felépítésüket tekintve sok hasonlóságot mutatnak. Általánosságban elmondható, hogy a vállalati információs rendszerek az adatok mennyisége és irányultsága szempontjából elképzelhetők úgy is, mint egy piramis.

1. A piramis alsó része a legnagyobb mennyiségű adatok metszete, ami a vállalatok esetében a legszélesebb körű nyilvántartás, adatfeldolgozás, tranzakciók szintje. Nemritkán az itt működő rendszerek különböző gyártók termékei, esetleg különböző platformokon futnak, az általuk használt adatbázisok pedig csak a legritkábban hasonlítanak egymásra.

Egy a COMSHARE-hez hasonlatos rendszer használata nélkül itt a piramis véget is ér, hiszen ekkor a vezetők számára a jelentéseket közvetlenül az adatbázisok lekérdezésével, SQL eszközökkel készítik pl. MS Excelből. Akkor lehet ez hátrányos, ha a vezetők lekérdezései miatt az egész rendszer lelassul. Egy relációs adatbáziskezelőre épülő modern OLTP[2] rendszernek azonban vannak olyan előnyei, amelyek miatt gyakran nélkülözhetetlen a vállalati adatfeldolgozás szempontjából. Ezek:

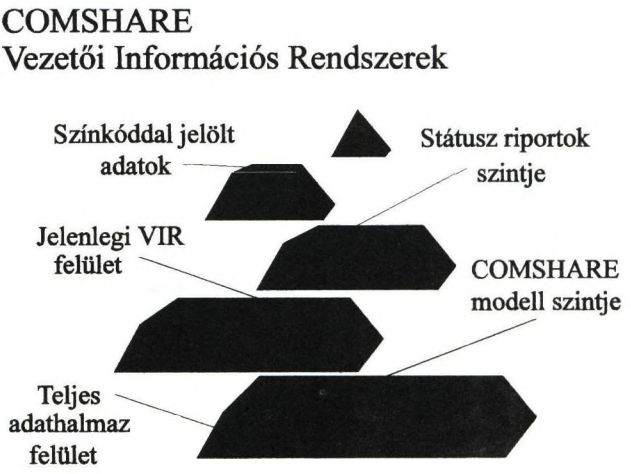
- nagy volumenű adatok kezelése;
- válalati szintű rendszerek;
- többfelhasználós, on line hozzáférés;
- SQL.

[1] A COMSHARE amerikai szoftvergyártó cég, mely több, mint 25 éve foglalkozik EIS (Executive Information Systems - vezetői információs rendszerek) fejlesztésével, és mára a világpiac egyik meghatározó cége ezen a területen.

[2] OLTP - On Line Transaction Processing (körülbelül On Line Ügyvitel)

Vannak azonban olyan tulajdonságai is, amelyek miatt bizonyos feladatokra nem a legalkalmasabb. Mivel kétdimenziós táblázatokban tárolja az adatokat, az adatmegtekintésre mindig ugyanazt a síkot kínálja fel. Tehát nem lehet az adatokat tetszőleges szempontok szerint "szeletelni" vagy "tömbökbe vágni". Ugyanakkor, nem támogatja az adatkonszolidációt, vagyis az adatok között nincs hierarchia, tehát nem lehet alábontani, göngyölni, összegezni egy struktúrán belül.

A jelentések ill. riportok általában SQL szekvenciák futtatásával készülnek. A vezetői információ-igény kiszolgálása, elsősorban az ad-hoc analízisek esetén, így sokszor nehézkessé válik, hiszen mindannyiszor újra meg újra meg kell ezeket írni a változások figyelmbe vételével.



Adatkonszolidáció a VIR-ben

2. A COMSHARE vezetői információs rendszerek a fent említettek kiküszöbölésére ezért általában egy multidimenzionális adatstruktúrában tárolják el azokat az adatokat, melyekre előreláthatólag a vezetőknek szükségük lesz. Hiszen a felső szintű vezetők úgysem az egész adattömeget "átlátva" akarnak dönteni; mindenképpen csak a számukra legfontosabb adatokat szeretnék számítógépük monitorán ill. papíron megjeleníteni. Az adatmodellt tetszőleges sűrűséggel lehet frissíteni, a frissítési periódust az alsó rendszerek lehetőségei szabják meg.

Ezeknek az adatbázisoknak a "válaszideje" nagyon rövid, az adatok gyorsan elérhetők, az adatok különböző szempontok szerint rendezhetők, analizálhatók (slice and dice, drilling).

A COMSHARE termékskáláján az igényektől függően választható modellezőket találunk, melyek között elsősorban a teljesítményben (és természetesen ebből adódóan árban is) jelentősek a különbségek. Windows környezetben a PRISM, OS/2 ill. Windows NT szervereken az ESSBASE a megfelelő modellező eszközök, amelyeken kívül mainframe megoldások is léteznek természetesen.

3. A kulcsadatok megjelenítése a felsőszintű vezetők gépein a legkülönbözőbb formákban lehetséges: relációs adatbáziskezelőkkel készített táblázatok, riportok, valamint ezek grafikonszerű ábrázolása, de ugyanakkor what-if analizísek és célkeresések is végezhetők. A vezetői munkahelyek Windows, OS/2 vagy Mac kliensek lehetnek, a COMMADER 4.0 megfelelő verzióját futtatva, mely kliens szerver megoldást kínál óriási hardver igény nélkül.

Itt külön érdemes kiemelni az Execu View nevű terméket, mely a Q+E Database Library integrálásával lehetővé teszi, hogy a relációs adatbázisokban tárolt adattáblákat multidimenzionális adatmodellként "lássuk". Ezáltal a már többször emlegetett "slice and dice" és "drill down" funkciók is megvalósíthatók ezeken az adatbázisokon, jelentősen megkönnyítve ezzel az adatok vizsgálatát és nem kevés SQL kódolástól kímélve meg a programozókat.

Végezetül elmondható, hogy a COMSHARE szoftvercsalád olyan technológiákat alkalmaz, melyekkel az eddiginél szélesebb vezetői kör számára tehetők elérhetővé kulcsfontosságú információk. Mindig egy testreszabott vezetői információs rendszer valósítható meg a vállalatnál, melynek a teljesítményéhez képest nincs nagy harver igénye.

Űrlap- és adatbáziskezelés JETFORM® szoftverrel

Biró Miklós, Hetthéssy Jenő, Remzső Tibor
MTA SZTAKI
1111 Budapest, Lágymányosi u. 11.
Tel: 269-8270
Fax: 269-8269

1. Elektronikus űrlapkezelés

A JetForm rendszerelemek az információtechnológiai rendszerekben fellelhető problémák egyik legjellegzetesebb fajtáját oldják meg igen korszerűen és hatékonyan.

A szokásos IT rendszerekben nincsenek igazán jó megoldások az űrlapok megjelenítésére, pedig az űrlapok a szokásos - nem számítógépes megjelenítési formák - legjellegzetesebb output formátumai. A jelenlegi IT rendszerek igen hatékony megoldásokat adnak arra, hogy hogyan tároljunk adatokat, hogyan és milyen eljárásokkal keressük azokat vissza, milyen képernyőket hozzunk létre. Általában kevés lehetőséget biztosítanak azonban arra, hogy professzionális formátumokat, űrlapokat nyomtathassunk.

A napjainkban igen időszerű EDI alkalmazások szintén felszínre hozták a professzionális megjelenítéssel kapcsolatos kérdéseket. Ezek lényege az, hogy az EDI rendszerek alkalmasak arra, hogy nagy biztonsággal elektronikusan továbbíthassunk elektronikus dokumentumokat (számlákat, szállítóleveleket, vámokmányokat, banki átutalásokat.). Arra nem képesek azonban, hogy a továbbított dokumentumokat professzionális módon megjelenítsék a fogadó oldalon, azokat a dokumentálásra alkalmas formára hozzák.

A korszerű form management rendszerek alkalmasak arra, hogy a fenti problémákat látványosan és hatékonyan megoldják. Ezek egyik legjobb tagja a JetForm cég által forgalmazott készlet, a JetForm Design, a JetForm Filler, a JetForm Server és a JetForm E-Mail rendszerkészlet.

A valóság az, hogy a korábban sokak által kitűzött papír nélküli iroda nem valósítható meg maradéktalanul. Nem elég az, hogy a dokumentumok adatait rögzítjük, szükség van arra is, hogy a dokumentumokat előállítsuk és azokat kezeljük, rendezzük, tároljuk. Amennyiben az űrlapok kezelésére egy elektronikus eljárást tudunk alkalmazni, a munka hatékonysága nagymértékben megnő.

A megoldás lényege az, hogy azokon a helyeken, ahol szükség van az igazi papírokra, ott papíralapú információkat biztosítunk, ahol azonban gazdaságosabb elektronikus megoldásokat alkalmazni, ott azokat alkalmazzuk. Lehetőséget kell biztosítanunk arra, hogy a dokumentumokat a keletkezés helyéről elektronikusan továbbítsuk a felhasználás helyére, majd ott rendelkezésre bocsássuk.

Erre bizonyos korlátozásokkal a telafax is lehetőséget nyújtana. Ez azonban nem elég. A telefax továbbítása jó minőségben nem olcsó, ugyanakkor a megkapott dokumentum minősége sem kielégítő.

Sokkal hatékonyabb megoldás, ha a küldő és a fogadó oldalon is rendelkezésre áll a számítógépeken egy-egy számítógéppel kitölthető űrlapformátum.

A küldő oldalon megfelelő módon kitölthetjük az űrlapot, majd a keletkezett adatokat - és csak azokat - a számítógépes hálózaton elküldjük azokra a helyekre, ahol azokat fel akarják használni. Ez a hely lehet egy épületen belül és távolabbi helyeken is (local area network és wide area network alkalmazások). A cél helyeken azután lehetőség van arra, hogy a megfelelő űrlapokat megjelenítsék, azokba beilleszthetőjenek a küldött adatok és jó minőségben kinyomtatódhassanak a végső formátumú űrlapok.

A lényeg az tehát, hogy a papíralapú dokumentumok mozgatását csökkentjük azzal együtt, hogy nem korlátozzuk az információk áramlását.

A megoldás látható előnye az, hogy csak a lényeges információkat mozgatjuk, a nagytömegű papírmozgásokat kiküszöböljük. Másik előny az, hogy amennyiben egy űrlapformátum változik, akkor elegendő a megváltozott űrlapot tartalmazó file-t átküldeni a fogadó állomásokra (ezek mérete néhányszor 10 kByte), azután a kinyomtatott űrlapok mind ebben a megváltozott formátumban jelennek meg. Ezzel szükségtelenné válik az ilyen esetekben szokásos selejtezés.

Komoly előnyt jelent az is, hogy - természetesen a jogi környezet figyelembevétele mellett - elegendő az, hogy csak az adatokat archiváljuk, nincs szükség arra, hogy minden papírlapot külön archiváljunk. Amennyiben elő kell keresni egy dokumentumot, akkor nem kell azt egy kartotéktengerben tenni, hanem elegendő egy jól strukturált adatállományt végignézni. A kívánt dokumentum ezután a megfelelő formátumban akár a képernyőn, akár nyomtatott formában rendelkezésre áll.

2. Az űrlapok működtetési jellegzetességei

Az elektronikus űrlapkezelő rendszerek tehát három szinten működhetnek:

- Automatikus űrlapkitöltést tesznek lehetővé és lehetőséget biztosítanak azok kinyomtatására.

  Előnyök:

  - szükségtelenné teszik az előrenyomtatott űrlapok használatát
  - az űrlapok mindig rendelkezésre állnak a készletektől függetlenül
  - az adatok pontossága javul
  - az adatkitöltéssel kapcsolatban egy sor intelligens lehetőséget biztosítanak (adatellenőrzések, automatikus formulakezelés, stb.)

- A megváltozott űrlapok a szervezetben elektronikus levélként terjeszthetők.

  Előnyök:

  - az űrlapok megváltoztatása után egyszerűen biztosíthatjuk azok terítését a szervezeten belül
  - szükségtelenné teszi az újabb nyomtatványok elkészítését, az új formátum az elküldés után azonnal használatra kész

- Adatbázis alkalmazásokat tesz lehetővé, lehetőséget biztosít arra, hogy az űrlapok mezőinek adatbázis mezőket feleltessünk meg, azokat az adatbázisokból automatikusan tölthessünk ki, valamint a beírt adatokat automatikusan beillesszük az adatbázisokba.

  Előnyök:

  - az információkezelést nagymértékben felgyorsítja és hatékonyabbá teszi
  - lehetőséget biztosít az adatokkal kapcsolatos emberi ellenőrzések követésére és naplózására (automatikus aláírásellenőrzés funkció)

Az űrlapokban mezőkkel (karakteres, numerikus, grafikus adatokat tartalmazó mezőkkel), grafikákkal, szövegekkel találkozhatunk.A mezők lehetnek:

- adatbeviteli (input) mezők
- származtatott mezők (input mezőkből matematikai műveletekkel származtatott mezők)
- kiszámított mezők

A mezőkkel kapcsolatos műveletek a JetForm rendszerben előre definiáltak, illetve tetszőleges Windows függvényt is használhatunk műveletként (DLL hívásra is lehetőség van.)

3. JetForm rendszerelemek

A JetForm rendszer alapvető részei:

- JetForm Design, az űrlapok grafikus tervezésére Windows környezetben.
- JetForm Filler, az űrlapok kitöltésére Windows, DOS, vagy Macintosh környezetben.
- JetForm Server, hálózati szerver rendszer. A PC lekötése nélkül teszi lehetővé nagy mennyiségű űrlap automatikus nyomtatását, elosztását és feltöltését adatbázisokból.
- JetForm E-Mail rendszer, kapcsolatot teremt a helyben használatos e-mail rendszerrel. Lehetővé teszi az elektronikusan elküldött űrlap további sorsának nyomonkövetését.

4. Adatbázis interfészek

A JetForm képes arra, hogy ANSI formátumban megadott adatokkal dolgozzon. Ezzel az ún. extended karakterek kezelésére alkalmas. Az *ODBC Manager* lehetővé teszi az *OEM to ANSI* konverzió végrehajtását bizonyos adatkezelő rendszerekben.

A fentiek mellett bizonyos adatkezelő rendszerekhez (pl. ORACLE, INGRES) speciális DLL-ekre van szükség. Ezeket a programcsomag tartalmazza.

A következő rendszerekhez létezik a JetForm 4.0 rendszerben ODBC driver:

- ALLBASE
- Btrieve 5, 6x
- Clipper
- DB2
- DB2/2
- DB2/6000
- dBase II, III, IV
- Excel .XLS file-ok
- FoxPro 1, 2.5
- Gupta SQL BAse
- IMAGE/SQL
- INFORMIX 4
- INFORMIX 5
- INGRES
- Microsoft SQL Server
- NetWare SQL 2.11, 3.x
- ORACLE 6 & 7
- PARADOX 3, 3.5, 4.0, .5
- Progress 6
- SQL/400
- SQL/DS
- Sybase System 10
- Sybase SQL Server 4
- Teradata
- Text files
- XDB 2.41, 3.0

A rendszer jelenleg a következő gateway alkalmazásokat támogatja:

- IBM DDCS/2
- IBM DDCS/6000
- Micro Decisionware
- Sybase - 4 NetGateway
- Sybase - 10 OmniNetGateway

A Mega termékek jellemzése

Csillagh Péter, Quantum Informatikai Kft.

Kulcsszavak:
UNIX alapú fejlesztés, nyitott felépítés, integrált, moduláris, hardver független, adatbázis interface, szabványok, paraméterezettség, folyamatos támogatás, több (párhuzamos) nyelvi felület.

A Quantum GmbH (Dortmund) software-ház magyarországi fejlesztő és forgalmazó csapata képviseletében néhány adattal jellemezzük a Mega termékcsoportot. A Mega software-k vállalatok ügyviteli feladatainak ellátását célozzák. A UNIX operációs rendszer alatti, C-nyelvű, mintegy 10 éves multú fejlesztés eredménye a közel 2000 installálás, amely a felhasználók igen széles körét öleli fel. A fejlesztési koncepció természetesen kihasználja a UNIX lehetőségeit (párhuzamos processek kezelése, kiterjedt paraméterezhetőség, portabilitás, modularitás).

A jellemző és párhuzamosan támogatott hardver eszközök áttekintését a következő táblázat foglalja össze.

Hardver típus	Operációs rendszer	Verzio	Support kezdete	Tovább fejleszt.	Support vége
BULL DPX2/x	*B.O.S.*	*összes*		*1993.12.*	*1994.12.*
DIGITAL	*Ultrix*	*4.3.*	*1993.01.*		
HP 9000/8	*HP-UX*	*9.0.*	*1993.07.*		
IBM RS6000	*AIX*	*3.2.*	*1992.10.*		
Kienzle	*System V*	*összes*		*1993.12.*	*1994.12.*
Motorola 36	*System V*	*összes*		*1993.12.*	*1994.12.*
Motorola RISC	*UNIX V/88*	*R 4.0 V 4.1.*	*1993.12.*		
NCR Tower	*SYSTEM V.3*	*összes*		*1993.12.*	*1994.12.*
PC 386	*SCO UNIX*	*3.2.4.*	*1992.07.*		
PC 485					
I-Pentium					
SNI MX/NSC	*SINIX-ATT*	*5.24.*		*1992.12.*	*1995.12.*
SNI MX/Intel	*SINIX*	*5.41.*	*1993.01.*		
SNI/RM	*SINIX*	*5.41.*	*1992.10.*		
SUN Sparc	*SUN-OS*	*4.1.3.*	*1993.01.*		
	SOLARIS	*2.1.*		*1993.12.*	*1994.03.*
		2.2.		*1993.12.*	*1994.03.*
		2.3.	*1994.01.*		
UNISYS 6000 (Mod. 30-65)	*UNIX V/386*	*R 4.0 V 4.2.*	*1993.12.*		
UNISYS 6000 (Mod. 70-85)	*összes*	*összes*		*1993.12.*	*1994.12.*

A Mega termékek fejlesztői folyamatosan követik az EK előírásokat, amelyek a support részét képező folyamatos verzióváltások során az ujabb termékekben megjelennek.

Verzióváltásra folyó gazdasági év közben is felkészült a rendszer, az adatok konvertálása egyszerűen, szabványos eszközökkel automatikusan történik.

Néhány termék jellemző funkcióit a következő felsorolás tartalmazza.

Mega/An - Tárgyi eszközök nyilvántartása, könyvelése

Törzsdatok (eszközök, országok, befektetések, biztosítások, garanciák)

Számlák (folyószámlák, dologi számlák, költséghelyek, költségviselők,biztosítások)

Kódtáblák (ÉCS elszámolási csoportok, ÉCS elszámolási formulák, eszközcsoportok, mérlegpoziciók, ÉCS százalék tábla, kedvezmények, ár-index tábla, vagyonjogi minimumok, kalkulált kamatok, biztosítási kódok)

Könyvelés (bejövő, kimenő, részmozgás, átkönyvelés, növekmény/pótlék, bonus/jóváírás, átírás, maradványérték-leírás, szolgáltatás könyvelés, leírt-eszköz törlés, garancia visszaigazolás)

Törzsadatlisták (számlarend, címlisták, eszköztörzs, biztosítási adatok, befektetési adatok, garanciális adatok, kézi ÉCS listája)

Tábla listák. Könyvelési fájl listák. Kiértékelések. Zárások.

Egyedi funkciók (konfigurálás, lekérdezések, szerep felosztások)

Mega/F - Pénzügy, könyvelés

On-line könyvelés (számlakönyvelés, kifizetések, dologi számlák, főkönyvi számlák, nyitott tételek teendői, számlalekérdezés, költség lekérdezés, vegyes átkönyvelések)

Kötegelt könyvelés (köteg kezelési műveletek, információ, határidős köteg, köteg betekintés)

Fizetési forgalom (kifizetési lista kezelése, nyomtatása, könyvelése, gyüjtőlista)

Törzsadatok (adósok, hitelezők, dologi számlák, ügynöki adatok)

Mandátum adatok (cégadatok, különleges számlák, nyelvi kódok, ÁFÁ-k, felszólítási szintek, szöveg elemek, banki kapcsolatok, fizetési formulák, csekk formulák, könyvelési kulcs, bizonylatszám csoportok)

Törzsadat listák (számlarend, zárolt számlák, számla törzsadatok, címlisták, periodus felosztás, költs. felosztási kulcs, kalkulált költségnem listák)

Kiértékelések (naplók, forgalmi áttekintés, főzárási áttekintés, likviditási terv, saldolisták, számlakivonatok, nyitott tétel lista, forgalmi statisztika, dispoz. lista, táblák)

Egyedi kiértékelések (nyitott tétel kiértékelések, kiegyenlítések, forgalmi jelentés, kiegyenlített adósok)

Felszólítások (listajavaslat képzés és nyomtatás, felsz. levél ellenőrzés, nyomtatás, késedelem könyvelés, késedelmi statisztika)

Zárások (éves előzetes, éves, pénzügyi peridus, költség periodus, napló törlés, periodus kezelés, kifutt napló törlése)

Mega/K - Költség elszámolás

Kiértékelések (költségelszámolási egyenlegek, költségkiértékelés)
Táblák (létrehozás, szabvány összköltség elj., szabvány szaldoegyenleg, ÜGK-kód, táblák ÜGK-kód szerint)
Törzsadatok (költséghely, költségviselő, kalkulált költségnemek, fix felosztások)
Költségátkönyvelések.

Mega/WWS - Anyag áruforgalom

Törzsadatok (cikktörzs, romlandó árú kezelés, ügyféltörzs, szállítótörzs, számlakezelés, egyéb törzsdatok, szövegelemek, információ)
Táblázatok (raktár, raktárhelyek, csoportosítások, munkatársak, ÁFÁ-k, országok, árlisták, szezonvezérlés, szállítmányozás, általános feltételek)
Zárások (árak aktualizálása, toplista, vételár átszervezés, eladásiár átszervezés, változó minimál köszlet, havi zárás, éves zárás, hitelkorlát aktualizálás, nyomtatás)
Vétel (érdeklődések, rendelések, javaslatok, reklamációk, számlaellenőrzés, vételárak, rendeléstár, nyomtatás, információ)
Eladás (ajánlatok, megbízások, reklamációk, eladás árak, megbízástár, nyomtatás, információ)
Anyaggazdálkodás (raktárkezelés, leltár, árubemenet, áruvisszaigazolás, készletkorrekcio, átvétel feldolgozás, visszáruzás, komisszionálás, nyomtatás, információ)
Kiegészítő modulok
Szerződések (poziciok, számlák, számla törlés, nyomtatás).
Költséghely/költségviselők.
Devizakezelés.
Szakaszos ügyletek.
Számlázás.

Mega/PZE - Munkaidőnyilvántartás

Törzsadatok (cégtörzs, éves naptár, időmodell, csoport definiciok, időszámlák, időszámlakezelés, számlacsoportok, határok és kerekítések, könyvelésnemek, személyi adatok, bérnemek határai)
Törzsadat listák.
Kiértékelések (időszámla listák, napi elszámolások, jelenléti lista, távozási lista, hiányzási lista, könyvelési lista, hiányzási statisztika, átkönyvelési napló)
Mozgási adatok (idő rögzítés, köteg kezelés, kézi átkönyvelés, hiányzás rögzítése)
Napi zárás.
Havi zárás.
Éves zárás.
Visszaszámlázás.

A Mega termékek hardver környezete a felhasználó igényei szerint alakítható. Az alap követelmény a saját adatbáziskezelő (QDBMS) használata és alfanumerikus monokrom monitorok alkalmazásával már biztosított. Beintegrálhatók a rendszerbe ETHERNET-re telepített diszkmentes 286-os PC eszközök. A terminál emuláció WINDOWS alatt is bitosított.

A WINDOWS cliens termék fejlesztése alfa tesztelés fázisába érkezett. Az adatmennyiség természetesen az alkalmazás függvénye. Az egyes termékek futtató kódja 100-200 MB tárigényű. A felhasználók számának növekedésével a CPU igény (a gyors válaszidőket biztosítandó) felhsználónként kb. 4MB-al növekszik. A rendszer teljesítőképessége meggyőzően napi többszáz számla és többezer felhasználó esetén mutatkozik meg.

A Mega termékek adataikat mandátum rendszerben kezelik. Egy mandátum teljesen önálló adategység, tulajdonképpen egy önálló gazdasági egységet (pénznemet, anyagtörzset, számlarendet, gazdasági évet, és valamennyi egyedi paraméter beállítást) jelent.

Az egyes felhasználók tevékenységi körét és a teljes rendszernek a rendszergazda által kijelölt részhalmazának használatára jogosító feltételeit minden esetben az alkalmazás függvényében paraméterezni lehet és kell.

Az egyes termékek funkcionális paraméterezését menü rendszerű eszközök biztosítják. A kontext szenzitív funkciós gombok (10 + 10 db) a működéshez logikailag szorosan kapcsolódó más menüágak tevékenységeinek - egymás fölé helyezett ablakok formájában történő - behívását teszik lehetővé.

A többszintű help funkció legalsó szintjén a teljes dokumentáció on-line help formájában jelenik meg.

A termék verziok érvényes főbb adatait az alábbi táblázat foglalja össze.

TERMÉK	Verzio	Továbbfejl. támogatás	Fejl. vége	Támogatás változatlan	Megjegyzések
Mega/AN	4.1. 4.2.	1993.08. 1993.12.			
Mega/F,K	3.9. 4.0.	1993.10. 1994.01.			
Mega/L	5.2. 5.3.	1994.01. 1995.01.		1995.07.	*Továbbiakban* csak test install.
Mega/ PZE	3.1. 3.3.	 1994.01.	1993.03.	1994.07.	
Mega/ WWS	4.3. 4.4. 4.5. 5.0. 5.1.	 1993.12. 1994.05.	1993.12. 1994.01. 1994.01.	1994.06. 1994.07. 1994.07.	Továbbiakban csak test install.

A termékek önálló (QDBMS) adatbáziskezelő rendszerrel rendelkeznek, de igény esetén a rendelkezésre álló interface eszközök módot adnak a jelentősebb relációs adatbázis-kezelő eszközök csatlakoztatásra. A rendelkezésre álló relációs adatbáziskezelő (RDBMS) intefrace eszközök felsorolását a termékek verzióinak és a támogatás aktuális adatainak feltüntetésével a következő táblázat foglalja össze.

Adatbázis Interface	**verz**	***Oper. neve***	**sys. ver.**	***Support kezdete***
Quantum	*4.3.*	*összes*	*össz.*	1994.01.
bas. sys.	4.4.	összes	össz.	1994.01.
INFORMIX	*4.1.x*	*AIX*	3.2x	1993.12.
		HP-UX	9.0x	1993.07.
Standard		SCO-UNIX	3.2.4.	1993.04.
Engine		SINIX-ATT	5.24.	1993.12.
(C-ISAM)		SINIX-Intel	5.41.	1993.07.
		SINIX-Risc	5.41.	1993.04.
		SUN-OS	4.1.3.	1993.04.
		ULTRIX	4.3.	1993.10.
	5.0x	AIX	3.2.	1993.12.
		SOLARIS	2.3.	1994.03.
	5.1x	AIX	3.2.	1993.12.
		SOLARIS	2.3.	1994.03.
INFORMIX	4.1.	AIX	3.2.	1993.12.
OnLine		HP-UX	9.0.	1993.10.
		SINIX-Intel	5.41.	1993.10.
		SINIX-Risc	5.41.	1993.09.
		SUN-OS	5.1.3.	1993.09.
	5.0.	AIX	3.2.	1993.12.
		HP-UX	9.0.	1993.12.
		SINIX-Intel	5.41.	1993.12.
		SINIX-Risc	5.41.	1993.12.
		SUN-OS	5.1.3.	1993.12.
	5.1.	*AIX*	*3.2.*	1993.12.
		HP-UX	*9.0.*	1993.12.
		SINIX-Intel	*5.41.*	1993.12.
		SINIX-Risc	*5.41.*	1993.12.
		SUN-OS	*5.1.3.*	1993.12.
	6.0.			1995
ORACLE	6.0. 31.	AIX	3.2.	1993.10.
	6.0. 32.	HP-UX	9.0.	1993.10.
	6.0. 36.	SUN-OS	4.1.3.	1993.10.
	6.0. 37.	SOLARIS	2.3.	1993.03.
	7.0.	AIX	3.2.	1994.02.
		HP-UX	9.0.	1994.02.
		SUN-OS	4.1.3.	1994.02.
		SOLARIS	2.3.	1994.03.

Az alapmodulok felsorolásával vázlatos képet kivántunk adni a termékek funkcionalitásáról. A kiegészítő modulok fejlesztése az egyedi alkalmazások tapasztalatai alapján folyamatosan történik. A jelenlegi készlet felsorolásától most eltekintünk. Részletesebb érdeklődés esetén termékbemutatóval egybekötve állunk rendelkezésére.

Az előadásban a hangsúlyt a termékek felépítésére, belső szabványaira, építőelemeire, a könyvtárak típusaira, a többnyelvű megoldásra, és a bővítési lehetőségekre kívánjuk helyezni.

Itt csupán megemlítjük, hogy jelenleg **német, angol, cseh, észt, magyar, lengyel,** nyelvű termékek szállíthatók.

Az egyes termékek igény szerint többnyelvű változatban is haszálhatók. Az egyik hazai felhasználó egyidejüleg német és magyar felülettet használ. Ebben az esetben egyazon adatrendszeren mindkét nyelv működik. A menük és a kinyomtatott bizonylatok nyelvét a bejelentkezés, illetve a jogosultság kijelölések során határozhatjuk meg.

A SAPIENS adatbáziskezelő-rendszer

és gyakorlati alkalmazása

Rada József

DUNAFERR Rt
Számítástechnikai és Szolgáltató Intézet

A Dunaferr Rt-nél hosszú évek óta használjuk a számítástechnikát és szinte pontosan ennyi ideje kutatunk olyan nagygépes adatbáziskezelő-rendszer után, amely segít minket a felhasználók egyre fokozódó igényeinek kielégítésében. Ennek folyamán tanulmányoztuk az IDMS, a DB2 és más adatbázis-kezelő rendszerek lehetőségeit, végül 1992. őszén lehetőségünk nyílt a SAPIENS objektum-orientált szabályalapú adatbáziskezelő-rendszer megismerésére, melynek megvásárlására két éves tesztelési időszak után került sor.

Mi az objektum-orientált programozás?

Az objektum-orientált programozás objektumok felismerésén alapul, melyek osztályozással különböző csoportokba kerülnek. Ezek meghatározásán kívül az osztálybasoroló eljárás és az egyes osztályok tulajdonságai - melyek meghatározzák az objektum viselkedését - szintén definiálásra kerülnek. Minden egyes eljáráshoz ún. üzenet kapcsolható, mely az osztály objektumainak küldhető. Egy eljárás küldhet üzenetet ugyanannak, vagy akár más objektumnak is. Egy adott osztályhoz tartozó üzenetek összességét osztály protokollnak nevezzük. Alapvető része még az objektum-orientált programozási technológiának az osztályba sorolt objektumok alosztályokba való rendezése.

A következő alapfogalom, mely az objektum-orientált programozás fontos jellemzője az öröklődés fogalma. Ez azt jelenti, hogy az egyes alosztályok - alapértelmezésben - öröklik a "szülő" osztály attributumait, eljárásait, üzeneteit (természetesen ezeken kívül saját jellemzőkkel is bírhatnak).

A harmadik, alapvetően új szemléletmódot tükröző kifejezés a többrétűség. Ez azon az elven alapul, hogy az ugyanzaz az üzenet elküldhető az egy "szülő" osztályhoz tartozó valamennyi alosztály objektumainak. Minden alosztály képes fogadni és értelmezni az üzenetet oly módon, ahogyan saját tulajdonságai és paraméterei megengedik.

Ehhez a fogalomkörhöz tartozik még az elkülönítés vagy egyszerűsítés fogalmának bevezetése is. Ellentétben az egyes elemek között létező kapcsolatok labirintusának vizsgálatával, az objektum-orientáltság az egyes objektumok elkülönítésén és a külvilágtól való elzárásában nyilvánul meg. Minden objektum különálló egységként értelmezhető, melyet nem befolyásol más objektumok viselkedése, tulajdonságai.

Ezek a lehetőségek (osztálybasorolás, öröklődés, többrétűség, elkülönítés) nagymértékben megkönnyítik az objektum-orientált programozási rendszerben fejlesztett rendszerek modularitását és segítenek a korszerűsítésben, karbantartásban.

Mi a SAPIENS?

Röviden a SAPIENS egy objektum-orientált szabály alapú adatbáziskezelő alkalmazásfejlesztő és karbantartó rendszer, mely lehetőséget ad különböző feladatok rugalmasan és gyorsan történő megoldására.

Ez pontosabban a következőket jelenti:

- létezik egy aktív tudásbázis, mely segíti az alkalmazások fejlesztését és azok tökéletes működését;

- létezik egy objektum orientált váz, mely áttekinthetővé teszi az adatbáziskezelő-rendszert;

- létezik egy szabály alapú objektum orientált technológia, mely a hagyományos programozást egyszerű szabályok alkotásával váltja ki;

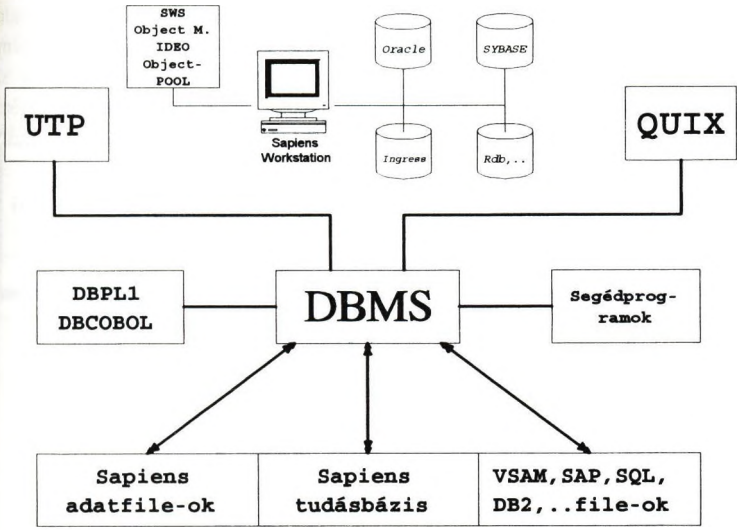
- megvalósítja a pozitív gondolkodás koncepcióját, melynek eredményeképpen az alkalmazások mérete jelentősen csökken;

- nyíltságot más platformok felé;

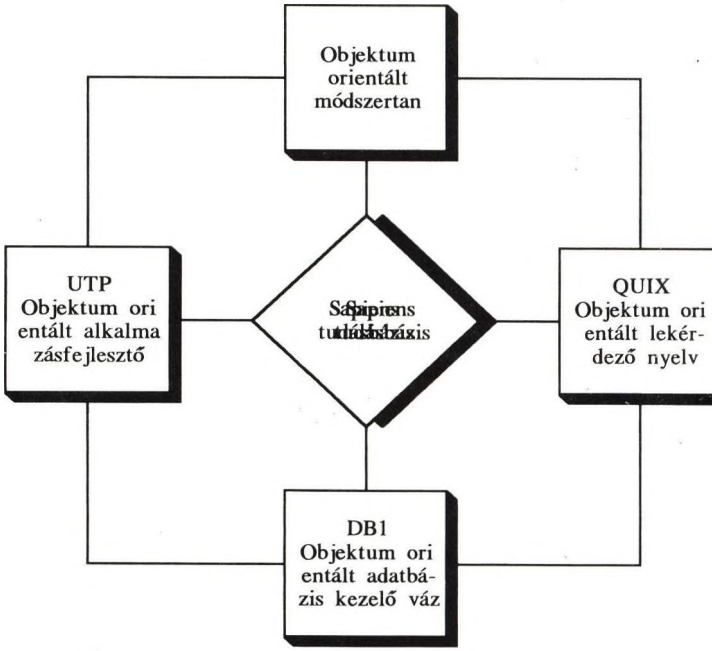
- a rendszerek integrálásának nagymértékű támogatását;

A SAPIENS fő komponensei:

A SAPIENS komponenseit a következő ábra mutatja :



1. ábra

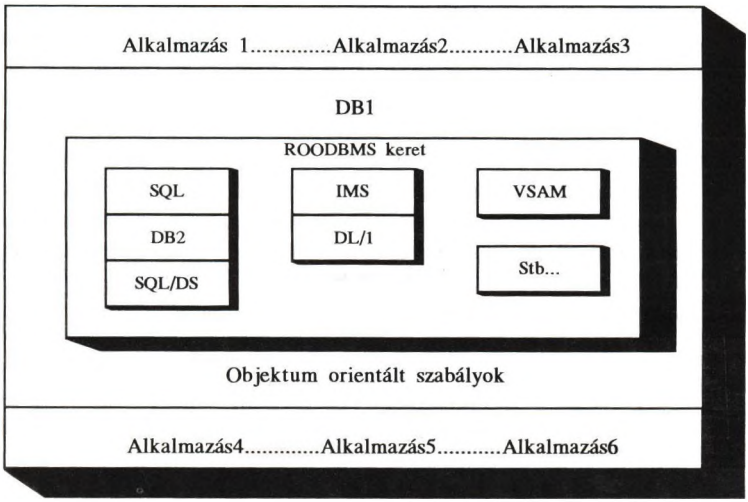

Objektum orientált módszertan
UTP Objektum ori entált alkalma zásfejlesztő
QUIX Objektum ori entált lekér- dező nyelv
DB1 Objektum ori entált adatbá- zis kezelő váz

2. ábra

DB1 - Az objektum orientált adatbáziskezelő rendszer

A SAPIENS integráltan összekapcsolt egy adatbáziskezelő rendszerrel, mely a DB1 nevet kapta. A rendszer alapvető szolgáltatásai közé tartozik azonban, hogy lehetőséget ad más adatbáziskezelőkkel létrehozott állományok kezelésére is.

Ez gyakorlatilag azt jelenti, hogy létezik valamiféle "keret", melynek segítségével a már korábban fejlesztett rendszerek könnyen átalakithatók SAPIENS alkalmazássá. A következő ábrán ez a bizonyos keret látható :



3. ábra

UTP - A Sapiens alkalmazásfejlesztő modulja

E fejezet elején már említésre került, hogy a SAPIENS alkalmazásásával a hagyományos értelemben vett programozás elkerülhető. A felhasználói rendszerek fejlesztésének rugalmas és egyszerű módszere nagyrészt az alkalmazásgenerátornak köszönhető, mely rendelkezik egy "alapmetódusok könyvtárával". Az itt található módszerek kivétel nélkül minden objektumra vonatkoznak. Az alkalmazásfejlesztő irányítását a tudásbázis végzi, ugyanis - többek között - az objektumok is itt kerülnek tárolásra. Mivel az alkalmazások fejlesztése és fenntartása elsősorban tranzakciók segítségével történik,

ezért az alkalmazásfejlesztő az *Univerzális Tranzakció Vezérlő* (Universal Transaction Processor) nevet kapta.

A programozás nélküli fejlesztésnek két fontos meghatározója van:

1. A SAPIENS rendelkezik a már fent említett alapmetódusok bázisával, mely minden objektumra érvényes. A fejlesztett rendszerben minden osztály (alosztály) örökli ezeket az alapmetódusokat.
2. Az alkalmazások a következő két részre bonthatók :

- Megjelenítés : mely ún. formákon (form) alapul. A forma egy meghatározott megjelenítési objektum, mely definiálja az adatképernyőket, folyamatokat és a menüket. A megjelenítést vezérlő rendszer tartalmazza az egyes formáknál megkövetelt funkciókat is. Ezek a formában megjelenő attributumoktól függenek, melyek blokkokba (block) vannak szervezve. Egy blokk segítségül hívhat másik formát többszintű megjelenítés céljából.

Egy-egy forma definiálása történhet automatikusan a SAPIENS által, de a felhasználó módosíthatja azt, vagy akár új formát is létrehozhat saját elképzeléseinek megfelelően.

- Feldolgozás : lényege abban áll, hogy az egyes objektumok adatmanipulációit szabályok határozzák meg. Ez azt jelenti, hogy a felhasználó által deklarált szabályok azok, amelyek elsődlegesen meghatározzák az objektum viselkedését.

A szabályok automatikusan végrehajtásra kerülnek melyet egy "kiváltó mechanizmus" vezérel.

A fent említettekből következik, hogy a SAPIENS tulajdonképpen egy alkalmazásgenerátor, melynek segítségével bármilyen adatfeldolgozási probléma könnyen megoldható.

A tudásbázis (Knowledgebase)

A SAPIENS rendelkezik egy aktiv tudásbázissal, mely tartalmaz egy hagyományos értelemben vett adatszótárt (Data Dictionary) és egy ezt kiegészítő alkalmazásszótárt (Application Dictionary) valamint egy "esetszótárt" (Case Dictionary) is. Az alkalmazásszótár írja le például a képernyőket, folyamatokat, menüket, míg az "esetszótár" az egyes entitások, kapcsolatok jellemzőit tartalmazza. A tudásbázis az a része a SAPIENS-nek, melynek segítségével a különböző alkalmazások végrehajthatók.

QUIX - A SAPIENS lekérdező nyelve

A lekérdező nyelv is az objektum orientált jellegre összpontosul. Képes automatikusan összekapcsolni objektumokat (Join), melynek következtében a felhasználó elkerülheti a bonyolult, összetett utasítások alkalmazását még a nem triviális esetekben is.

Rendelkezik ezen kívül szerkesztő és a megjelenési formát befolyásoló utasításokkal, melyek hasznosak az egyes listák, kimutatások elkészítésénél.

VIOLA

Adatbázis lekérdező nyelv és prototípus készítő rendszer

Varga Kornél és Kovács László

STUDIDACT BT.
1118 Budapest, Villányi út 55. I. I/1. Tel/fax: 165-2058

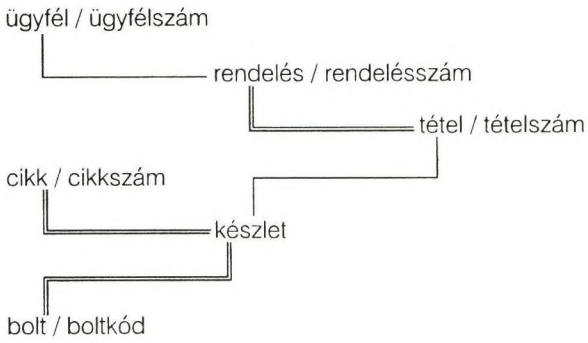
A Visual data Organizing Language (VIOLA) magyarul vizuális adatkezelő nyelvet jelent. Nevét onnan kapta, hogy mind az adatok egymással való kapcsolata, mind az adatbevitel és adatlekérdezés megfogalmazása jól áttekinthető, grafikus formában történik.

A VIOLA nyelv lényege és újdonsága az, hogy az adatlekérdezést a természetes nyelvhez közeli formában írja le, mintegy vezetve a felhasználót, hogy a kérdését a számítógép által elvárt matematikai precízséggel tudja megfogalmazni.

Az elvi háttérről

Szemléltetésül nézzük a következő példát: egy kereskedelmi vállalat több boltja forgalmazza a termékeket, központilag tartják nyilván a készleteket és az ügyfelek rendeléseit.

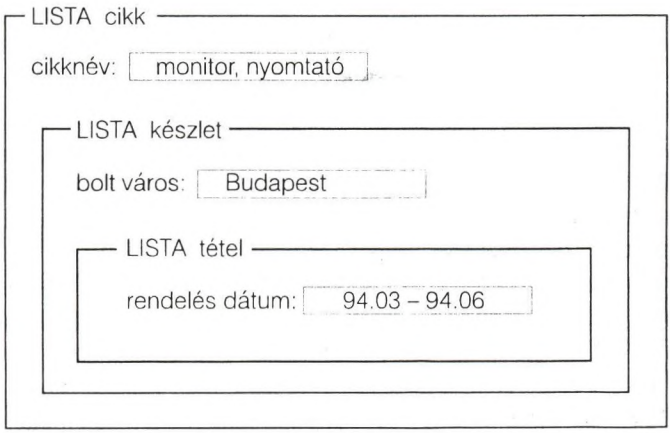
Az alábbi séma az adattáblák kapcsolati rendszerét ábrázolja:



A táblázat mögött (/ jellel elválasztva) a táblázat (elsődleges) kulcsa van feltüntetve. Az összekötő vonalak a kulcsok átvételét mutatják balról jobbra. A dupla vonal azt jelenti, hogy a bal oldali táblázat kulcsa a hozzá kapcsolódó (tőle jobbra lévő) táblázatban – annak saját kulcsa mellett – szintén elsődleges kulcs lesz. Így az ábráról leolvashatók az adatbázis szerkezetére vonatkozó legfontosabb információk. (A rajz szerint pl. a rendelés az ügyfélszámon keresztül kapcsolódik az ügyfélhez, de a rendelés táblának csak a rendelésszám a kulcsa, míg a tétel táblázatban a tételszám a rendeltetésszámmal együtt alkot kulcsot.)

A VIOLA nyelv lényege, hogy szervesen összefonódik az adatsémával. Kihasználja a táblázatok közötti (akár áttételes) kapcsolatokat, ami nagyon lerövidíti és áttekinthetővé teszi a programot.

Egy lekérdező program írható olyan formában is, hogy a felhasználó egyszerű képernyőtervet lásson, amelyben csak a felkínált mezők tartalmát írhatja át. *Pl. bizonyos cikkeknek a budapesti boltokban található készletét és egy időszakra vonatkozó rendelésállományát az alábbi program jeleníti meg:*



A fenti programnak van nyelvi megfelelője is, a VIOLA programok akár közönséges szövegszerkesztővel is szerkeszthetők. *A budapesti ügyfelek idei rendelésállományának összértékét pl. az alábbi program számítja ki:*

```
ügyfél.rendelés_érték [város = 'Budapest']:=
    ÖSSZEG tétel [
        rendelés.dátum>94.01.01
        mennyiség * cikk.ár ];
```

Az interaktív editor ugyanakkor – a programnak a sémával való állandó összevetése révén – sok támogatást nyújt a programok írásához. Az editor egy helyzetérzékeny menü segítségével mindig azokat az adatokat kínálja fel a felhasználónak, amik ott éppen logikusan elérhetők.

A VIOLA nyelv központi fogalma az un. **fókusz**. Azt jelenti, hogy egy program vagy kifejezés írása során mindig van egy "aktuális" táblázat, amelynek éppen egyik sorára pozicionálunk. Azt mondjuk, hogy ez a táblázat (ill. annak egy sora) van a fókuszban. Egy táblázat pozicionálása (azaz fókuszálása) logikus kihatással van a többi táblázatra, ami jól követhető a sémán: egy táblázattól jobbra helyezkednek el azok a táblázatok, amelyek vele 1:N kapcsolatban vannak, pl. egy ügyfél több rendelést is feladhat. Jobbról balra ugyanakkor hivatkozni lehet minden kapcsolódó táblázat mezőire, pl. az ügyfelek rendelési tételeinek kiírásakor (vagyis amikor a tétel van a fókuszban) a tételben szereplő cikk további adatai ugyanúgy "láthatók", mintha a tételben lennének tárolva (SQL terminológiával ez egy implicit view-képzést jelent).

A tapasztalat szerint a fenti séma segítségével felgyorsítható az adatbázis tervezése, továbbá könnyebben el lehet magyarázni a felhasználónak, hogy milyen adatokat tartalmaz az adatbázis, és azok milyen összefüggésben kereshetők vissza.

A fenti egyszerű feladat természetesen még szinte bármilyen módszerrel kezelhető. A VIOLA séma és a fókusz mechanizmus akkor válik igazán hasznossá, amikor bonyolultabb feladatról van szó. Ez részben sok táblázatot jelent, részben összetettebb kapcsolatrendszert. Pl. ha a fenti sémát kiegészítenénk egy város táblázattal, akkor ehhez több táblázat is kapcsolódna: az ügyfél a lakcíme révén, ill. a bolt a címe révén. Egy vásárlási tételre pozicionálva ekkor egyszerre két város is "látszik", amelyekre külön-külön is kell tudni hivatkozni: az a város, ahol a bolt van, ill. a tételt vásárló ügyfél lakhelye. A VIOLA nyelvben ennek a kezelése is jól követhető módon van megoldva.

A lekérdezés során valójában az adatsémán "navigálunk", mindig szem előtt tartva, mi is van éppen a fókuszban, melyik - a sémán jobbra lévő - táblázatra akarunk következő lépésként fókuszálni, ill. mely - a sémán balra lévő - táblázatok mezőire lehet ugyanúgy közvetlenül hivatkozni, mintha az aktuális táblázatban lennének.

Példaként válasszuk ki azon budapesti boltokat, melynek volt legalább két olyan ügyfele, aki tavaly minden cikkből vásárolt!

```
LISTA bolt [
    város = 'Budapest'
    2 <= ELEMSZÁM ügyfél [
        MINDEN cikk [
            VAN tétel [ year(rendelés.dátum) = 1994 ]
  ]
 ]
]
```

Mint látható, a VIOLA lekérdezés nagyon jól illeszkedik a természetes nyelvű megfogalmazáshoz, mondhatni szinte annak precíz átfogalmazása. Ez azért lehetséges, mert a fókusz a természetes nyelvű mondatoknak is jellemzője, és a kapcsolatok "beleértése" szintén fontos jellemzője az emberi gondolkodásnak és nyelvnek. (A fenti példában a "tavaly minden cikkből vásárolt" kifejezés értelemszerűen arra a boltra és azon ügyfélre vonatkozik, akire a mondat első felében "fókuszáltunk". A séma alapján az is "természetes", hogy a tétel mögötti zárójelben arra a rendelésre gondolunk, amiben az a tétel szerepel.)

Összehasonlításul nézzük meg ugyanennek a feladatnak SQL-beli megoldását!

```
SELECT * FROM bolt
WHERE város = 'Budapest' AND 2 <=
 ( SELECT COUNT(*) FROM ugyfel
  WHERE NOT EXIST
   ( SELECT * FROM cikk
    WHERE NOT EXIST
     ( SELECT * FROM tetel, rendelés
      WHERE tétel.rendelésszám = rendelés.rendelésszám AND
      rendelés.ügyfélszám = ügyfél.ügyfélszám AND
      tétel.cikkszám = cikk.cikkszám AND
      tétel.boltkód = bolt.boltkód AND
      year(rendelés.dátum) = 1994
     )
   )
 )
```

Mint látható, a táblázatok kapcsolatát minden alkalommal explicit módon ki kell írni, ami azon túl, hogy felesleges írásmunkával terheli a felhasználót, a kulcs kapcsolatok pontos ismeretét is feltételezi.

A relációs elmélet független egyedhalmazokkal végzett matematikai műveletekkel operál, amelyek algebraként jól használhatók, de a természetes nyelvtől távolabb esnek (nehezebb kifejezni pl. a hétköznapi nyelv "minden" operátorát).

Az SQL ősének (SEQUEL) egyik kulcsgondolata éppen a beágyzások struktúrált kezelése volt. Ugyanakkor az SQL-ben a beágyazások felhasználása igazán csak az egzisztenciális kvantifikálás esetében egyszerű. A VIOLA beágyazásai viszont felhasználják a sémakapcsolatokat, így az aggregát műveletek (ill. logikai kvantorok) természetesebb és szélesebb skáláját tudják nyújtani.

A VIOLA előnyei ténylegesen a több táblát érintő, a sémában definiált kapcsolatokat felhasználó és beágyazást tartalmazó lekérdezésekben mutatkoznak meg. Igaz, hogy egy teljes rendszerben nem ezek a leggyakoribb lekérdezések, de az ad-hoc lekérdezésekben pontosan ezekre van szükség. A mai rendszerekben éppen azért kevés az ilyen kérdés, mert az SQL átlagfelhasználói nincsenek "képesítve" a fenti lekérdezés megfogalmazására.

A VIOLA felület tetszőleges adatbáziskezelő felett megvalósítható, ehhez mindössze két dolog szükséges. Egyrészt az eredeti adatbázisnak megfelelő VIOLA lekérdező sémát kell elkészíteni, másrészt a VIOLA lekérdezéseket az eredeti adatbázison kell interpretálni.

Ha az adatbázis megfelelő tervezési dokumentumokkal rendelkezik, akkor a lekérdező séma gyakorlatilag megegyezik az adott felhasználóhoz tartozó external view-val. Ilyen információk hiányában a VIOLA séma tábláinak az eredeti adatbázis relációt lehet megfeleltetni, a kapcsolatokat pedig a referenciális integritási constraint-ek (hálós rendszerben a set definiciók) alapján képezhetők.

A VIOLA lekérdezések intepretálása relációs rendszerben célszerűen SQL-re fordítással történik. Ennek során a VIOLA indirekt hivatkozásai illesztésekkel, a beágyazások pedig SQL subselect-ekkel helyettesítődnek, amelyek kiegészülnek a kapcsolódást biztosító korlátozásokkal. (A fenti példa SQL lekérdezése egyben a VIOLA lekérdezés "fordításának" tekinthető.)

Hálós rendszerben a feldolgozás rekordonkénti ciklusban történik. A hivatkozások a "szülő" felé történő navigálást, a beágyazások a "gyerek" felé navigálva indított belső ciklust jelentenek. Ez a módszer indexek használatával bármely, csak rekordműveleteket biztosító rendszerre (pl. dBase) is kiterjeszthető.

Érdemes még megemlíteni, hogy a VIOLA lekérdezések könnyebben - jobban - optimalizálhatók, mint a hagyományos relációs lekérdezések. Erre vonatkozóan konkrét eredményeket értünk el, melyek közül talán a legjelentősebb, hogy a beágyazások kidolgozásánál az előre definiált 1-N kapcsolatok, és az azokhoz tartozó gyors hozzáférési utak - pl. index - közvetlenül felismerhetők és szelektivitásuk pontosabban számítható.

A VIOLA nyelv implementációja

A fentiek alapján a VIOLA nyelv egyik fő alkalmazása az adatlekérdezések felhasználóbarát megfogalmazása, ami az adott programkörnyezetnek megfelelően (leginkább SQL-re fordítva) interpretálható. A VIOLA nyelvnek elkészült azonban egy önálló implementációja is, ami - saját adatbázis-kezelőre támaszkodva - önmagában is használható alkalmazások készítésére.

Mire használható a VIOLA?

- Meglévő **adatbázisok lekérdezésére**. dBASE fájlokhoz közvetlen hozzáférést biztosít, más szerkezetű adatbázisokhoz pedig könnyen előállítható karakteres bemenetet.
- Adatbázis feladatok **tervezésére**, és egy **működő prototípus** gyors előállítására.
- Egyszerűbb **adatnyilvántartások készítésére**.
- Rendkívül jól használható a VIOLA az **oktatásban**, mind felhasználók alapszintű adatfeldolgozási képzésében, mind pedig professzionális szervezők és programozók oktatásánál, akár az SQL képzés előkészítéseként is.

A VIOLA különösen hatékony eszköz nagyobb adatbázisok egy-egy kisebb részhalmazának átvételére, majd ennek a szűkített adatbázisnak a további elemzésére, beleértve **kiegészítő számítások és riportok** készítését is.

Kinek készült a VIOLA?

Az adatbázis-kezelés oktatásánál és az alkalmazásfejlesztési gyakorlatban szerzett többéves tapasztalat alapján állítható, hogy a felhasználók gyorsan meg tudják szokni a VIOLA logikáját és egyszerű, kifejező kezelői felületét. A szoftverben jól elkülönülnek az egyes ügyfélköröknek szánt részek:

- A **kezdő végfelhasználó** számára a programozó pontosan körülhatárolt funkcionalitást és képernyő tervet tud előállítani.

- A **gyakorlottabb felhasználó** önállóan is meg tud oldani egyszerűbb feladatokat, és nagyobb szabadságfokot kaphat az adatok lekérdezésénél is.
- A **programozó** számára elsősorban a tervezési fázisban, egy működő prototípus előállításánál használható a VIOLA. Meglévő alkalmazáshoz illesztve pedig az adatok lekérdezésében nyújt magasszintű szolgáltatást a szokásos lehetőségekhez képest.

A VIOLA néhány további jellemzője

- Maga az adatmegjelenítő a programhoz hasonlóan egymásba ágyazott ablakokban mutatja az összetartozó adatokat.
- Az ablakok és mezők elrendezése futás közben is megváltoztatható.
- Bármelyik mezőre kérhető rendezés vagy csoportosítás.
- A számítás és rendezés meglepően gyors - a VIOLA speciális adattárolási technikájának köszönhetően.
- A VIOLA futtatórendszere világos keretbe foglalja a feladatmegoldás különböző fázisait: az adatséma összeállítását, adatok átvételét (pl. dBASE fájlokból) vagy az adatlekérdezést, ill. kiegészítő számításokat.
- Prototípus készítésekor, ill. oktatásnál hasznos segédeszköz a nyelvbe épített tesztadat generátor.

Technikai adatok

- A program C nyelven íródott, saját futtatórendszerrel és adatbázis-kezelővel rendelkezik.
- A program jelenleg DOS környezetben fut, minimális konfiguráció IBM AT 1 Mbyte memóriával.

* * *

Az Esperant grafikus lekérdező és riport generátor

Szerző: Vincze Attila

1995. január

COMET Kereskedelmi és Számítástechnikai Kft.
H-1037 Budapest, Mikoviny u. 2-4.
Tel.: (36-1) 250-5067 Fax: (36-1) 168-9540

Az Esperant grafikus lekérdező és riport generátor

Információ. Minden döntéshozónak, és az általuk használt kifinomult alkalmazásoknak egyre többre van szüksége belőle. De mindezek az információk értéktelenek, ha a szükségesnél többen, illetve nem a megfelelően kiértékelt formában állnak rendelkezésre. Ezért kerültek kifejlesztésre a munkaállomások felhasználóbarát grafikus környezetében futtatható - nem előre programozott végeredményeket produkáló - lekérdezők, melyek minimális végfelhasználói ismereteket követelnek. Ide tartozik a Software Ag. fejlesztésében elkészült Esperant is.

A számítástechnikai környezet felépítésének változása

A hajdan tradícionális mainframe környezetek, melyek nagyteljesítményű relációs adatbáziskezelőkön mint az ADABAS és a DB2 alapultak, mára osztott architekturás rendszerekké fejlődtek ki. Helyi hálózatokon keresztül OS2, UNIX stb. adatbázis szervereket elérő Windows alapú munkaállomások kapcsolódnak a mainfarme-es környezethez. Ezeken az adatbázis szervereken az eddig említett adabáziskezelők mellett, egy és ugyanazon rendszerben még számtalan más adatbáziskezelőt megtalálhatunk pl. az Oracle-t, Sybase-t stb. Ezek az osztott és heterogén környezetek sokkal bonylultabbá tették az adatelérések mecanizmusát, elsősorban a végfelhasználók számára. Szerencsére az SQL 'de facto' ipari szabvánnyá fejlődött ki az adatbázisok elérésre. Virtuálisan minden vezető relációs adatbáziskezelő támogatja az SQL szerinti lekérdezéseket, ide értve az ADABAS-t és az új ADABAS SQL szervert is. Az Esperant számára a

háttérben a fizikális kapcsolatot az ADABAS-val vagy más gyártók SQL adatbáziskezelőivel a Software Ag. ETIRE NET-WORK és az ENTIRE ACCESS termékei teremtik meg egy osztott környezetben.

SQL alapú grafikus lekérdező eszközök

A fizikailag élő kapcsolat önmagában még kevés, hiszen az SQL túl komplex és nehezen elsajátítható a végfelhasználük számára. Munkaállomások grafikus környezetében futó SQL lekérdezők tucatjai vannak a piacon, melyek generálják a szükséges SQL utasításokat a készítendő riportok számára. Mindezek a termékek egyszerű összesítések végeredményei, és egyszerű adatbázis struktúrák esetén tökéletesen működnek. De hány ilyen adatbázis, vagy információ igény van a termelésben. A legtöbb alkalmazás adattáblák százaival és nagyon bonyolult struktúrával dolgozik. A probléma akkor vetődikfel először amikor a népszerű grafikus lekérdezőket ilyen valós nagyméretű és komplex alkalmazások adatbázisain futtatják. Bár a legtöbb termék képes előállítani a megfelelő bonyolultságú SQL lekérdezést, van egy sokkal kritikusabb kérdés. A riportokban kimutatott eredmény nem korrekt. A legtöbb ilyen SQL lekérdező eszközzel igen könnyen készíthet a felhasználó félrevezető vagy inkorrekt eredményeket.

A mai végfelhasználói eszközök korlátai

A legtöbb lekérdező eszköz az SQL kifejezéseket közvetlenül a felhasználó kijelöléseiből generálja, a felhasználü minden egyes választása

közvetlen korrelációban van a generált SQL lekérdezéssel. Ezért, ahhoz, hogy elkerülje a félrevezető vagy inkorrekt eredményeket, a felhasználónak ismernie kell az SQL elméletét is. Értenie kell a relációs táblák kapcsolásának szabályait, mint például a köztes táblák kapcsolását és mit kell tegyen, ha több út is létezik a táblák kapcsolására; a 'GROUP BY' és a 'HAVING' kifejezések használatának szabályait; valamint ha subquery szükséges egy COUNT létrehozásához. Az adtabázis tervezésben való jártasságra is szükség van. A felhasználónak meg kell találnia minden egyes önálló logikai egységet jelentő adatot, amely gyakran szét van tördelve fizikailag különálló táblákba.
Például egy megrendelés adatai gyakran egy 'megrendelés fejléc' és egy 'megrendelés részletek' táblába van szeparálva. Ugyennennél a példánál maradva, a felhasználónak tudnia kell, hogyan használja a 'vevő', a 'termék' stb. táblákat, hogy a megrendeléshez tartozó egyéb információkat is fellelje.

A lehetséges új standard a végfelhasználói hozzáféréshez

A Software Ag. ESPERANT terméke megoldja ezt a problémát. Speciálisan az adabázis ismeretekkel nem rendelkező, üzleti szakemberek számára fejlesztették ki, és nem hasonlít a többi ilyen termékhez. Hatékony szakértői rendszer biztosítja, hogy félrevezető és hibás eredmények ne jöhessenek létre, és teljesen szükségtelenné válik az SQL és az alatta lévő adatbázis struktúra ismerete. Az ESPERANT Query Assistant az eddigi rendszerekhez hasonló módon a felhasználó számára könnyedén kezelhető egér kattintásokkal nyerheti ki, rendezheti, összegezhati stb. a kért információkat, miközben az SQL Expert a háttérben nyomon követi a

kéréseket és egyszerüen letiltja (nem kiválaszthatóvá teszi) a további választások közül azokat melyek félrevezető vagy hibás eredményt produkálhatnak.

Az ESPERANT nyelv

Az ESPERANT lekérdezéseket egyszerű, összetett angol monadatokban specifikálhatóak mint pl. az alábbi

'Show Coustomer name and customer city for custumers that have orders for every product'

A kezdők a Query Assistant segítségével, a gyakorlottabbak közvetlen bevitellel, vagy meglévő lekérdezések módosításával közvetlenül kérhetik az eredményeket, de bérki is írta a lekeérdezést mindenki el tudja olvasni és megérti azt. Ezért az előre definiált vagy elmentett lekérdezések értelemszerű elühívás és módosítása igen egyszerű feladatttá válik.

Hatékony SQL generálás

Az ESPERANT extrém méretű SQL is generálhat, így a lehető legszélesebb tartományát fedi le az üzleti jellegű lekérdezésekeknek. Létre tud hozni subquery-ket, teljes külső kapcsolásokat, UNION müveleteket, és dinamikus create wiew/drop view műveleteket végez szükség szerint.
Adatálja az éppen alatta futó adatbázis kezelőnek megfelelő SQL dialektust, ellentétben sok 'egy méret mindenkire' ODBC termékkel.

Nyitott és flexibilis adatbázis elérés

Az ESPERANT a felhasználó Windowsos PC-jén, vagy file serveren helyezkedik el, ahoonan a hálózati megoldások széles skáláján képes elérni bármely platformon futó bármely elterjedt relációs adatbázist (Oracle, Sybase, stb.). Mivel az ESPERANT teljesen Microsoft ODBC azonos ezért a legtöbb az ODBC támogatott adatforrást eléri.

AZ ORACLE CASE OKTATÁSA A KOSSUTH LAJOS TUDOMÁNYEGYETEMEN

EPERJESI BARNABÁS
KELLERMANN LÁSZLÓNÉ DR
PAPP ÁGNES

(Kossuth Lajos Tudományegyetem Informatikai és Számító Központ, Debrecen)

ELŐZMÉNYEK

A Kossuth Lajos Tudományegyetemen a programtervező matematikusok és informatika tanár szakosok képzésében jelentős szerephez jut az információs rendszerek kialakításával kapcsolatos tárgyak oktatása. A Matematikai és Informatikai Intézet és az Informatikai és Számító Központ munkatársai 1993-ban egy Adatbázis-kezelés sávot indítottak be az informatikát tanulók számára. A sáv egy szakterület mélyebb megismerését szolgáló előadások és gyakorlatok sorozata, átlagosan heti 4 órában, 4 féléven keresztül. Az adatbázis-kezelés sáv célja az adatmodellezés és adatbázis-kezelés elméleti és gyakorlati vonatkozásainak szélesebb körű tárgyalása, az alapképzés óráinak folytatásaként. A sáv tantervében a hangsúly a gyakorlaton van. A hallgatók konkrét adatbázis-kezelő rendszerekkel és negyedik generációs nyelven alapuló alkalmazásfejlesztő szoftverekkel ismerkednek meg tanulmányaik során. E szoftverek default lehetőségeik és beépített intelligenciájuk révén gyors fejlesztést tesznek lehetővé. A gyorsan elérhető eredmény ígérete azonban átgondolatlan, rendszerbe nem illeszthető, nehezen karbantartható, egyszóval rossz minőségű szoftver készítésére csábíthat. Nem elegendő tehát csak a konkrét szoftverek lehetőségeinek bemutatására szorítkozni az órákon, hanem foglalkozni kell azzal a kérdéssel is, hogy hogyan lehet a megismert eszközökkel jó minőségű rendszert készíteni.

Figyelmet kell szentelni az információs rendszerek és általában az informatikai alkalmazások fejlesztésénél a napjainkban még inkább előtérbe kerülő kérdésekre, mint például:

- Hogyan lehet megoldani az egyre nagyobb komplexitású rendszerek sikeres fejlesztését, fokozni a team-munka produktivitását, megbirkózni a szűk határidőkkel?

- Hogyan tegyünk eleget a rendszerrel szemben támasztott szigorú minőségi követelményeknek?

A fent vázolt problémák megoldásában támpontot adhatnak a szoftverminőség biztosításával foglalkozó szabványok és ajánlások, különösen az 1994. április 1-től Magyar Szabványként is elfogadott ISO 9000-3 nemzetközi szabvány.

Az előadás a T4033 sz. OTKA pályázat támogatásával készült.

A SZOFTVERMINŐSÉG ÉS BIZTOSÍTÁSA

Minőségbiztosítással, ezen belül számítógépes szoftverek minőségével kapcsolatban számos szabvány létezik (IEEE, MIL, DoD, EN, ISO, stb.). Különösen nagy figyelmet kell fordítanunk az ISO 9000 sorozathoz tartozó szabványokra, mivel ezeket a világon kb. 100 ország, közöttük Magyarország is elfogadta és tette nemzeti szabvánnyá. Az ISO 9000-3 szabvány foglalkozik speciálisan a szoftverfejlesztés során alkalmazandó minőségirányítási és minőségbiztosítási irányelvekkel és tevékenységekkel.

A minőség fogalmának megközelítési módja szoftver esetében eltér a minőség termékalapú definíciójától, helyette implicit módon egy folyamatalapú fogalom-meghatározást ad, tehát nem magának a szoftvernek jellemzőiről, hanem a létrehozására irányuló tevékenységről rendelkezik. A szabvány szerint a szoftverfejlesztési folyamat fázisokra bomlik, a fázisok eredményét ellenőrizni kell, az elkészült szoftvert pedig be kell vizsgálni.

A szabvány rendelkezik a minőségügyi rendszer kialakításával és működtetésével kapcsolatos tevékenységekről. A minőségügyi rendszer a minőségirányítás megvalósításához szükséges szervezeti felépítés, feladatkörök, eljárások, folyamatok és erőforrások összessége. A szoftvert előállítónak dokumentált minőségügyi rendszert kell létrehoznia és fenntartania. A minőségügyi rendszer legyen a teljes életciklusón átívelő, integrált minőségbiztosítási folyamat forrása, amely azt biztosítja, hogy a minőséget a fejlesztés előrehaladtával szisztematikusan beépítik a termékbe. Jobb a problémákat megelőzni, mint megjelenésük után korrigálásukkal foglalkozni.

Kitér a szabvány a minőségügyi rendszer dokumentálására, különös tekintettel a minőségügyi terv készítésére, ami egy adott termékre, projektre vagy szerződésre vonatkozó, a konkrét minőségügyi eljárásokat, erőforrásokat és a teendők sorrendjét rögzítő dokumentum.

A szoftverfejlesztés műszaki tervét valamilyen életciklusmodell alapján kell kidolgozni. A szabvány nem ír elő konkrét modellt, de maga is meghatároz fázisokat, amelyekkel részletesen foglalkozik:

- A szerződés átvizsgálása
- A megrendelő követelményeinek előírása
- A fejlesztés tervezése
- Minőségtervezés
- Programszerkesztés és programírás
- Tesztelés és bevizsgálás (érvényesítés, validálás)
- Átvétel
- Másolatkészítés, szállítás és üzembe helyezés (telepítés)
- Karbantartás

A szabvány foglalkozik olyan támogató tevékenységekkel, amelyek nem kötődnek konkrét fázishoz, a fejlesztés egész időtartama alatt figyelmet kell fordítani rájuk:

- Konfigurációkezelés (és nyilvántartás)
- A dokumentumok ellenőrzése
- Minőségdokumentáció
- Mérés

- Szabályok, gyakorlati módszerek és szokások (konvenciók)
- Eszközök és műszaki eljárások (technikák)
- Beszerzés
- Beszállított szoftvertermék
- Képzés

A szabványban megfogalmazott követelmények egy jelentős részének eleget tehetünk, ha az információs rendszerek kialakításánál egy megfelelő projektvezetési módszertanon túl egy strukturált rendszerfejlesztési módszertant is adaptálunk.

SRUKTURÁLT MÓDSZERTANOK

Általános jellemzők:

- A rendszerkészítési folyamatot fázisokra, részfeladatokra bontják, strukturálják.
- A feladatok megoldásának, elvégzésének módját megadják, ehhez előírásokat, ajánlásokat nyújtanak, ellenőrzési pontokat határoznak meg.
- Az egyes fázisokban jól definiált (diagramra és nem diagramra épülő) technikákat használnak.
- Dokumentációs szabványokat írnak elő, a dokumentálás folyamatos és nem utólagos.
- A rendszerfejlesztést technológiaként fogják fel, termékszemléletűek.
- A minőségbiztosítás kifejezett hangsúlyt kap.

Alkalmazásukkal a problémák nem oldódnak meg automatikusan, mindegyik átfogóbb mint ami egy konkrét rendszer kifejlesztéséhez szükséges, a kiválasztott módszertant a konkrét körülményekhez ésszerűen alkalmazva (adaptálva a projektre) érdemes felhasználni. A módszertanok elméletileg papíron-ceruzával is követhetők, de dokumentációigényességük miatt célszerű számítógépes eszközöket (CASE szoftvereket) alkalmazni.

A hallgatók az alapképzés során megismerkednek rendszertervezési módszertanokkal, de tapasztalataink szerint ezek tényleges alkalmazás hiányában elméleti ismeretek maradnak. Ezen megfontolások alapján döntöttünk úgy, hogy a sáv tematikájába beillesztjük egy strukturált módszertan és az annak folyamatát minél teljesebb körben támogató CASE szoftver oktatását is. Az 1993/94-es tanévben nyílt lehetőségünk először arra, hogy az ORACLE CASE rendszerfejlesztési módszertanát és az azt támogató eszközeinek használatát tanítsuk hallgatóinknak.

AZ ORACLE CASE

Az ORACLE CASE olyan technológia, mely módszertanával, modellezési technikáival és számítógépes rendszerfejlesztést támogató eszközeivel a bonyolult alkalmazások tervezését, fejlesztését a célok megfogalmazásától a megvalósításukig támogatja. Az ORACLE CASE rendszer középpontjában az ORACLE relációs adatbázis-kezelővel megvalósított adatbázis áll, ahol a leendő alkalmazás minden specifikált fejlesztési adata tárolódik.

A CASE*Method strukturált módszertan definiálja az életciklust, minden fázishoz feladatokat rendel, a feladatokhoz meghatározza a feltételeket, a bemenő és kimenő adatokat, a szükséges

tevékenységsort, erőforrásokat, technikákat, eszközöket és a minőségbiztosítással kapcsolatos teendőket. Az életciklus fázisai:

- Stratégia
- Analízis
- Tervezés
- Implementálás, felhasználói dokumentáció készítése
- Bevezetés
- Üzemeltetés

Támogatott modellezési technikák:

- Egyed-kapcsolat modell
- Funkcióhierarchia modell
- Adatfolyam modell
- Mátrix (keresztreferencia) technika
- Egyéb, nem diagramra épülő technikák

A fejlesztési adatbázis lehetővé teszi a leendő rendszerre vonatkozó, az egyes fejlesztési fázisokban specifikált adatok tárolását, így lehetőség nyílik az egyik fejlesztési fázis adatainak a következő fázisba vagy más számítógépes környezetbe való automatikus - de preferenciák megadásával szabályozható - transzformálására, ahol azt tovább lehet finomítani. Ez folytatható mindaddig, míg az adatbázis-leírás, a modulok, a komplett alkalmazások generálhatók. Ugyancsak a fejlesztési adatbázis teremt biztonságos keretet a csoportmunkához és alapját képezi a rendszer változatok, verziók kezelésének. Az ORACLE CASE támogatja a gyakorlatban megvalósuló iterativ fejlesztésnél elkerülhetetlenül adódó visszalépést, újratervezést és a különböző szintek között visszafejtési funkciókat is biztosít.

A fejlesztési adatbázist kezelő eszközök:

- CASE*Dictionary
  Form alapú, menüvezérelt karakteres tervezői interfész, mely az életciklus minden szakaszában használható.

- CASE*Designer
  Grafikus tervezői interfész, mely az alapvető modellezési technikákat támogatja.

- CASE*Generator for SQL*Forms
  SQL*ReportWriter/SQL*Plus
  SQL*Menu
  Alkalmazásgenerátorok, amelyek a fejlesztési adatbázis alapján adatbázis definíciókat és futtatható alkalmazásokat generálnak. A kifinomult generáló-képesség révén a prototípus jellegű alkalmazásoknál fejlettebb, akár végleges alkalmazások is készíthetők. A prototípusok a megfelelő ORACLE 4GL eszközzel tovább-fejleszthetők.

- CASE*Exchange
  Ezek az eszközök más gyártók CASE eszközeihez egy- vagy kétirányú kapcsolódást tesznek lehetővé a CASE adatbázisok egymásra való leképezhetősége függvényében.

OKTATOTT MÓDSZERTAN, ESZKÖZÖK, SZÁMÍTÓGÉPES KÖRNYEZET

Módszertan:

- CASE*Method

A rendelkezésre álló szoftver eszközök:

- CASE*Dictionary
- CASE*Designer
- CASE*Generator for SQL*Forms
  SQL*ReportWriter/SQL*Plus
  SQL*Menu

Környezet:

VAX 6510, VMS operációs rendszer

OKTATÁSI TAPASZTALATOK

A módszertannal és az eszközökkel egy féléven keresztül, heti 4 órában foglalkoztunk, amelyből 2 óra előadás, 2 óra gyakorlat volt. Az ORACLE CASE eszköztára rendkívül gazdag, ezért az oktatás eredményességét átgondolt oktatási módszerrel és segédanyagokkal igyekeztünk fokozni. Így sikerült a hallgatókat egy esettanulmány kapcsán a szoftverkészítés teljes folyamatán végigvezetni, sőt a biztonságos csoportmunka lehetőségeit is kipróbálni. Az előrehaladást segítette a hallgatók szakmai előismerete, ugyanis az előző félévekben már megismerték az ORACLE adatbázis-kezelőrendszert és az ORACLE 4GL eszközeit. Az esettanulmány kapcsán nyilván nem tudtuk bemutatni a rendszer minden lehetőségét, sőt az egyes szakaszokban még soknak is tűnt a megadandó illetve megadható információ mennyisége egy ilyen kis feladatnál. Valójában nagy méretű alkalmazások fejlesztése során bizonyosodhatna be az eszközök hatékonysága de ilyen volumenű rendszer elkészítésére, valóban önálló alkotómunkára a tantárgy időkeretei nem adnak lehetőséget.

Friss diplomás programozók, programtervezők hamar olyan munkakörbe kerülhetnek, ahol feladatuk lesz rendszerek üzemeltetése, kész rendszerek beszerzése, adaptálása vagy rendszerek fejlesztése során csoportmunkában való részvétel.

Világszínvonalú technológia megismerésével véleményünk szerint piacképes tudás birtokába jutnak a hallgatók. A CASE módszertanok által sugallt szemléletmód kialakítása napjainkban nagy jelentőséggel bír. Az alkalmas megközelítési mód sikeressé teheti az információs rendszerekkel kapcsolatos feladatok megoldását, akár fejlesztésről, üzemeltetésről vagy továbbfejlesztésről van szó.

IRODALOM

ISO 9000-3:1991 (MSZ ISO 9000-3:1994) Minőségirányítási és minőségbiztosítási szabványok, 3. rész: Irányelvek az ISO 9001 szabvány alkalmazásához a szoftverfejlesztés, -szállítás és -karbantartás területén

K. Balla, Attempts to introduce software quality management, Austrian-Hungarian Seminar on Software Engineering Proceedings, Klagenfurt, 7th and 8th April, 1994, pg. 6.1-6.10

Richard Barker: CASE*METHOD Tasks and Deliverables, Addison-Wesley Publishing Company, Wokingham, 1991

Richard Barker: CASE*METHOD Entity Relationship Modelling, Addison-Wesley Publishing Company, Wokingham, 1991

Richard Barker, Cliff Longman: CASE*METHOD Function and Process Modelling, Addison-Wesley Publishing Company, Wokingham, 1992

CASE*Dictionary Reference Guide V5.0

CASE*Designer User's Guide and Tutorial V1.1

CASE*Generator for SQL*Forms/SQL*Menu Tutorial and Reference V2.0

CASE*Generator for SQL*ReportWriter/SQL*Plus Tutorial and Reference V1.0

VT-SOFT

Adatmásolatkészítés

Az osztott adatkezelés egyik lehetősége INGRES környezetben

Mivel napjainkban egy vállalaton, intézményen belül az adatok egyre fokozottabb mértékben decentralizáltan oszlanak el és különféle adatbázisokban tárolódnak, az informatikai szervezetek igen hamar eljutnak arra a felismerésre, hogy stratégiájuknak ki kell terjednie ezeknek a meglehetősen elosztott adatoknak az ellenőrzésére és a vállalat, intézmény igényeinek megfelelő összehangolására. Az osztott adatokat tartalmazó hálózatok tipikusan heterogén hardver és szoftver elemekből épülnek fel, a manapság korszerű kliens-szerver technológia pedig valós idejű adathozzáférést tesz lehetővé és követel meg. Valós idejű tranzakció-orientált és más alkalmazások versenyeznek tehát egymással, ezért az adatbiztonság, az adat elérhetőség és a rendszer áteresztő képessége kiemelt jelentőségű az osztott környezetben. Az osztott adatbázis integritásának megőrzése pedig igen nagy kihívás a rendszergazdák számára.

Az integritás biztosítására szolgáló leginkább ismert eljárás a "két-fázisú ellenőrzés" (two-phase commit) az osztott adatkezelést szinkronizálja a résztranzakciók eredményének ellenőrzésével. A következő tranzakció végrehajtása (illetve az adott tranzakció sikeres befejezése) csak valamennyi résztranzakció sikeres lezárása után lehetséges. Ezen eljárás nagy hátránya, hogy a tranzakcióban résztvevő egyik gép sem lehet üzemen kívül, mert akkor az egész művelet függőben marad. Az integritás biztosításának másik eszköze, az adatmásolatkészítés megoldja ezt a problémát, mert aszinkron frissítést tesz lehetővé, hibatűrő módszert kínálva az adatbázis integritásának megőrzésére anélkül, hogy a felhasználó erőforrásainak 100%-os rendelkezésre állását követelné meg. A szakma széles körben elfogadta a másolatkészítést, mint olyan technikát, amellyel a hibatűrő osztott adatbázis funkcionalitás megvalósítható.

A CA-INGRES/Replicator, az INGRES rendszer adatmásolatkészítő modulja jelentős mértékben hozzájárul, hogy a fent felsorolt elvárások teljesüljenek úgy, hogy a helyi adatbázisból a kijelölt adatokat távoli adatbázisokba átmásolja. A CA-INGRES/Replicator különböző forrású adatbázisokat is képes bevonni a másolatkészítési sémába. A CA-INGRES kétirányú, nyílt másolatkészítési technikája támogatja az IMS, DB2, Oracle, IDMS és Datacom adatállományokról oda-vissza való adatállománymásolást is.

Egy vállalati szintű osztott hálózatban az INGRES/Replicator

- növeli a termelékenységet és áteresztő képességet úgy, hogy azoknak teszi lehetővé az adatok elérését, akiknek szükségük van rá és akkor, amikor szükség van rá
- adatbázis konzisztenciát és integritást biztosít
- rugalmasan konfigurálható és egyszerűen használható.

A CA-INGRES/Replicator automatikusan átmásolja a kijelölt adatokat a helyi adatbázisból egy vagy több távoli adatbázisba. A rendszer úgy konfigurálható, hogy a legkülönfélébb igények kielégítése mellett támogassa a heterogén hálózatok széles választékát. A rendszer például

- helyi adat-hozzáférést tehet lehetővé távoli adatbázisban, mely jelentősen csökkenti a válaszidőket és a hálózati adatforgalmat,
- hibatűrő működésmódot valósíthat meg, amikor a központi adatbázisból folyamatosan másolja át az adatokat egy háttér adatbázisba.

A CA-INGRES/Replicator úgy növeli a vállalati szintű adatkezelés hatékonyságát, hogy az adatokat oda másolja, ahol azt a legtöbben használják. Pl: központilag tárolt adatokat át lehet másolni az önálló fiók üzletekbe, ahol az adatok tényleges feldolgozása történik; vagy ellenkezőleg, a távoli helyen képződött adatokat egy adatközpontba lehet másolni, s így egy integrált adatbázist létrehozni.

VT-SOFT Kft.
1033 Budapest, Vörösvári út 103-105. ✉ 1300 Budapest 3. Pf.: 138.
Telefon: 250-0744, 250-1558, 250-1582, 250-0750 Telefax: 250-0750

A CA-INGRES/Replicator a felhasználói programok módosítása, azaz programozási munka nélkül teszi lehetővé az adattöbbszörözés előnyeinek használatát. Azt, hogy az aszinkron adatfrissítés milyen módon történjék, a rendszeradminisztrátor tetszőleges módon, az alkalmazói környezet igényei szerint állíthatja be: az adatmásolatkészítést kezdeményezheti egy tranzakció; egy trigger (mint rekordok száma, tranzakciók száma, bizonyos időpont, időtartam, stb.); vagy kézi beavatkozás.

Egy adatbázis-tábla egyszerű átmásolása nem biztosítja az adatbázis integritását, mert a tábla "snapshot" nem biztos, hogy az összes adatot átmásolja, amit egy tranzakció módosított. Pl. egy banki tranzakció módosíthatja az ügyfél megtakarított pénzének tábláját és a bank összes tőke tábláját. Ha a két tábla átmásolása nem történik azonos időben, akkor a háttér adatbázis táblái nem lesznek szinkronban a fő adatbázis tábláival, tehát a rendszer adatintegritása sérül.

A CA-INGRES/Replicator esetében egy tranzakció által módosított adat mindaddig nem másolható át egy távoli adatbázisba, amíg a tranzakció nem zárult le a helyi adatbázisban. Ha a tranzakció több tábla adatait is módosítja, akkor az összes módosítás egy tranzakcióként kerül át a távoli adatbázisba. Ez a tranzakció konzisztencia biztosítva van akár rendszerhiba esetén is, mely magába foglalhatja a fő-, illetve háttér adatbázisok hibáit, vagy akár a kommunikációs rendszer hibáit is.

A CA-INGRES/Replicator egyik legfontosabb jellemzője a felhasználó által beállítható konfliktus feloldás, mely a módosítások ütközését kezeli és így biztosítja az adatok konzisztenciáját.
Mikor az adat a háttér adatbásisokban eltér a bejövő másolattól, akkor a CA-INGRES/Replicator konfliktus feloldója észreveszi az ütközést, jelentést készít róla és a felhasználó által választhatóan elvégzi az alábbi opciók szerinti tevékenységet:

- Végrehajtja az első módosítást a háttér adatbázisban, "roll back"-eli a konfliktust és folytatja az adatmásolást. A felhasználó meghatározhatja, hogy a teljes tranzakciót, vagy csak azokat a sorokat "roll back"-elje, amelyek a konfliktust okozták.

- Módosítja a háttér adatbázist a legutolsó adatváltozással és folytatja a másolást.

- Feloldja a konfliktust egy felhasználó által definiált prioritás alapján és folytatja a másolást. Prioritásokat lehet hozzárendelni adatbázisokhoz, adattáblákhoz, tábla részekhez.

- Leállít minden adatmásolást, amíg a konfliktust kézzel megszüntetik.

Minden esetben a módosítási konfliktust okozó adat - módosítása előtti és utáni - értéke eltárolásra kerül. A konfliktus feloldó különösen akkor hasznos, mikor szinkronizálni kell két adattáblát egy rendszerhiba megszüntetése után.

A CA-INGRES/Replicator nagyszámú funkció-halmaza és konfigurációs rugalmassága révén jól használható minden osztott környezetben, kezdve az egyszerű master-slave kapcsolattól a bonyolult "peer-to-peer", kaszkád és gateway kapcsolatokig.

Az amerikai Forrester Research, Inc. független szakértő cég 1994. októberi "The Software Strategy Report, Volume Five, Number Seven: Weaving The Dataweb" tanulmánya szerint, mely az osztott adatkezelés lehetőségeinek kérdéseit elemzi, a CA-INGRES/Replicator jelenleg a legszélesebb szolgáltatásokkal rendelkező eszköz heterogén környezetű osztott adatkezelés megvalósítására és leginkább ajánlható azon számítástechnikai vezetők számára, akik hosszú távon is értékőrző beruházáshoz keresnek korszerű műszaki megoldásokat.

Elosztott ORACLE adatbázisok

dr Mohay Tamás
ORACLE

1. Abstract

Az előadás vázolja az elosztott adatbázisok néhány lehetséges alkalmazását, és bemutatja azokat az eszközöket amelyeket az ORACLE7 adatbáziskezelő (V7.0, V7.1, V7.2) ehhez nyújt.

2. Felhasználási lehetőségek.

Nem ritka manapság, hogy egy munkahelyről több adatbázis adatait is el lehet érni. Már a gépek közötti hálózatok önmagukban is lehetővé teszik, hogy a cég más városba települt egységeinek adatait tanulmányozzuk, összehasonlíthassuk. Minőségileg más lehetőségeink vannak azonban, ha maga az adatbáziskezelő is támogatja az adatok elosztását az egyes adatbázisok között. Az egész cég egyetlen adatbázist használhat, ahol az adatok földrajzi elosztása optimalizálási kérdés csupán. A különböző célokra más-más technikát lehet használni. Vegyünk egy példát:

- Az egyes egységek független könyvelést vezetnek, a vállalati vezetés számára szükséges összesített adatokat elosztott lekérdezéssel (distributed query) gyűjtik egybe.
- Az egységek közötti átutalásokat elosztott tranzakcióként végzik, amiben a kétfázisú véglegesítés (two-phase commit) biztosítja, hogy az átutalás vagy minden egységnél megtörténik vagy egyiknél sem.
- Az árlistát a központban készítik és hetente egy csak olvasható másolatát (read only snapshot) másolják oda minden egységnek.
- A vevőket is a központban tartják nyilván, de minden nap lemásolják az egységeknek a lista azon elmeit, amely vevők felbukkanása az adott egységnél várható. Ha egy új vevő bemegy az egyik egységhez, ott nyilvántartásba veszik, majd este az új vevő bekerül a központi nyilvántartásba, ahonnan lemásolódik mindenhova ahol szükség lehet rá (updatable snapshot).

- A cég egységes raktárt kezel, bár az egyes tételek más - más város raktárában vannak. Az összes egységben ugyanaz a (teljes) raktárkészlet van nyilvántartva, csak meg van adva, hogy melyik raktárban van az áru. Ha valaki egy rendelést fogad el, akkor kivételezi a raktárból az árut a saját nyilvántartásából, majd pár percen belül ez a módosítás megjelenik az összes egység raktárnyilvántartásában (N-way replication).

3. Az elosztott adatbázis kezelés technikái

Az elosztott adatbázis jóval több, mint a kliens - szerver (elosztott) rendszerek, mert itt adatbázis szerverek állnak kapcsolatban egymással, míg a kliens - szerver rendszerekben csak az alkalmazás logikája van elosztva az adatokat szolgáltató szerver és a kezelői felületet adó kliens gép között. A szerverek közötti műveletek a következők lehetnek:

- elosztott lekérdezés (distributed query)
- szinkron módosító művelet (two-phase commit, syncron remote procedure call)
- aszinkron módosító művelet (snapshot, asyncron remote procedure call, updateable snapshot, N-way replication).

3.1. Elosztott lekérdezés

Egy lekérdezésben (select) lévő táblák különböző gépeken lehetnek. A lekérdezés optimalizálásakor ugyanazok az információk állnak rendelkezésre a távoli táblákról, mint a helyiekről.

3.2. Két fázisú véglegesítés

Egy tranzakcióban több adatbázis felé is kiadunk adatmódosító utasításokat (insert, update, delete) és lekérdezéseket. Amikor a tranzakció lezárására az alkalmazás kiadja a commit utasítást, akkor elkezdődik egy párbeszéd a tranzakcióban érintett szerverek között, melyet az a szerver koordinál, ahova a tranzakciót végző alkalmazás be van jelentkezve (Global Coordinator). Meghatározódik az a szerver, ahol legelőször fog végrehajtódni a véglegesítés (a Commit Point Site). Ez a két szerver a protokol végrehajtása során egymást ellenőrzi. A folyamat lényege: Az első fázisban a Global Coordinator minden a tranzakcióban érintett szerverhez egy felhívást intéz (Prepare), azok három választ adhatnak, 'felkészültem, a szükséges erőforrások rendelkezésre állnak (Prepared)','Nem tudom véglegesíteni a módosításokat (Abort)','tőlem csak adatot kértek, nem érint a véglegesítés (read-only)'. Ha minden szerver adott időn belül jelezte, hogy felkészült vagy nem érintett, akkor a második fázis beindul, a koordinátor utasítja a Commit Point Site-ot a véglegesítésre. Ellenkező esetben a Rollback-re. Az összes szerver köteles ugyanazt a műveletet végezni, mint a Commit Point Site. Erre tett igéretett a 'felkészültem' üzenettel. A művelet renkívül robosztus, fel van készülve minden lehetséges hibára. Ha egy szerver kiesik (leszakad a hálózatról vagy leáll), amikor újra bekapcsolódik a munkába először is tisztázza mit tettek a többiek a függőben maradt tranzakciókkal, és ugyanazt teszi ő is.

3.3. Szinkronizált távoli eljárás hívás. (Syncron Remote Procedure Call)

Az A szerveren futó eljárás végrehajthat egy eljárást a B szerveren, majd annak befejeződése után tovább fut az A szerveren. Az A és B adatbázison végrehajtott műveletek mind egy tranzakció részei, erről a kétfázisú véglegesítés gondoskodik. Ez biztosítja az adatok konzisztenciáját, de a B szerver hibája, vagy a hálózat hibája esetén nem tud az A szerveren futó program sem lefutni.

3.4. Aszinkron távoli eljárás hívás.

Az A szerveren futó program el akar indítani egy eljárást a B szerveren, de nem kívánja megvárni annak lefutását. Beteszi a program indítást az A szerveren lévő sorba (job queue) és folytatja a feladatát, lezárja a saját tranzakcióját. Az A szerver gondoskodik a sorban álló programok lefuttatásáról a megadott (B) szerveren egy másik tranzakció keretében. A sorkezelő mechanizmus garantálja, hogy az egy szerver felé egy tranzakcióból indított programok az adott szerveren egy tranzakción belül és a hívás sorrendjében fussanak le. A külön tranzakcióból indított programok is megőrzik a hívási sorrendjüket. Minden hívás pontosan egyszer fut le a célként megjelőlt szerveren. Ha a B szerver nem érhető el, amikor az A szerver sorkezelő mechanizmusa programot akar rajta indítani, akkor a B-n indítandó programok kimaradnak, és egy későbbi időpontban az A szerver újra megpróbálja lefuttatni őket. Így a lokális tranzakció abban a tudatban fejezi be a futását, hogy a meghívott programok előbb vagy utóbb végrehajtódnak. Így vannak olyan időpontok amikor az elosztott adatbázis nem konzisztens, de ez bizonyos alkalmazásoknál megengedhető, és a hálózat egy részének hibája nem bénítja le a többi szerver működését.

3.5. Csak olvasható pillanatkép (Snapshot).

Az A szerveren levő mester tábláról, vagy annak egy részéről pillanatképeket tárolhatunk a B szerveren. A pillanatkép létrejöttekor fel is töltődik azokkal az adatokkal amiket a definiáló lekérdezés válogat le a mestertáblából. A későbbiek során a pillanatkép kétféleképpen frissítődhet: kitörli a pillanatképet majd újra letölti az összes adatot (teljes frissítés), vagy csak a mester táblában azóta változtatott sorokat vizsgálja meg, és tölti le. Ehhez létre kell hozni egy napló állományt a mester tábla mellett, amiben feljegyződik, hogy mely sorokat módosították. Ez a frissítési módszer csak azoknál a pillanatképeknél alkalmazható, amelyek definiáló lekérdezése csak egy táblára hivatkozik és nem tartalmaz összevonó műveletet (group by). Frissítéskor egy időfüggvény kiértékelésével kapott időpont határozza meg a következő frissítés időpontját.
A táblák között hivatkozási kapcsolatok lehetnek (referential integrity constraint), amelyek nem teljesülnének, ha az egyes mestertáblákról különböző időpontban készülnének frissítések. Ezért vezették be a pillanatkép csoportokat. Az egy csoportba tartozó táblákról egyszerre, egy tranzakción belül frissítődnek, ezáltal a pillanatképek között is fennmarad a hivatkozási integritás.

3.6. Változó mestertábla (dynamic ownership)

A mester tábla lemondhat az elsődlegességéről. Időben változtatható, hogy a másolatok közül melyik legyen az eredeti, a mester tábla, de egy adott pillanatban továbbra is csak egy tábla módosítható.

3.7. Módosítható pillanatkép (Updatable Snapshot)

A csak olvasható pillanatképekhez hasonló szerkezet, de megengedett a pillanatképek módosítása. A módosításokat egy aszinkron távoli eljáráshívás vezeti át a mester táblába, ahonnan a megszokott módon kerül el a többi másolatba (esetleg csak bizonyos másolatokba, amelyek definiáló lekérdezése ezt lehetővé teszi). Egy új probléma forrása, hogy ugyanannak a táblának egy adott sorát egyszerre több helyen is módosítják. Az eltérő módosítások a mestertábla módosításakor ütköznek össze. A tervező felelőssége, hogy megfelelő ütközésfeloldásról gondoskodjon. Írhat saját algoritmust, vagy választhat egyet az előre elkészített megoldásokból:

- a legutolsó módosítás jut érvényre,
- a nagyobbik értéket fogadja el,
- összeadja a változtatásokat.

Be kell látnunk, hogy egy elvi probléma terhét vállaljuk magunkra ezzel a módszerel, amivel a hálózatok lassúsága és megbízhatatlansága ellen kívánunk védekezni.

3.8. Egyenrangú másolatok (N-way replication)

A csoport minden tagja egyenrangú és szándéka szerint pontosan ugyanazt tartalmazza. Az új adat, vagy meglévő adatok módosítása aszinkron távoli eljáráshívásokkal kerül át a többi tagra. Akárcsak a módosítható pillanatképeknél itt is egy adatot több helyen is lehet egyszerre módosítani. Ugyanazokkal az ütközésfeloldó algoritmusokkal oldhatjuk fel az ellentmondást.

4. Összefoglalás

Mint láthatjuk az ORACLE7 adatbáziskezelő sokféle eszközt bocsát a felhasználók rendelkezésére, amivel egységes elosztott adatbázist lehet készíteni. Az egyes lehetőségek közötti választás a tervezők számára egy új kihívás.

Az Informix Dinamikusan Skálázható Architektúrája

Sándor Gábor (OpenSoft Kft.)

1525 Budapest 114, Pf. 49.
Tel, fax: 160-0717; 169-9542.
E-mail (Internet): gsandor@informix.opensoft.kfki.hu

Az 1994 elején megjelent INFORMIX-OnLine Dynamic Server V6.0 (majd az év folyamán a V7.0 és V7.1) a teljesen új *dinamikusan skálázható architektúrá*ra (DSA) épül. A belső párhuzamosságot és a többszálúságot alkalmazó termék megjelenésével új távlatok nyíltak meg a rendelkezésre álló hardver erőforrások maximális kihasználása felé. A rendszer - a dinamikus beavatkozási lehetőségekkel - új dimenzióba helyezi az adatbázis adminisztrátori tevékenységet is. Szintén újszerű az a megoldás, hogy az egy- és többprocesszoros (SMP, klaszter, és nagy mértékben párhuzamos) hardvereket egyazon adatbázis szerver tudja kiszolgálni. Ezáltal az alkalmazás elkészítésekor nem kell a hardver specialitásaira ügyelni, a program többprocesszoros környezetben is hatékony lesz a DSA alapú szerver jóvoltából.

Az Informix új szervereinek tervezésénél a jelenlegi adatbáziskezelőknél tapasztalt szűk keresztmetszetek felszámolása volt a fő cél. Az elemzések rávilágítottak az adatbázis-krízis tényére: a hagyományos adatbáziskezelők nem képesek az új, egyre nagyobb teljesítményű multiprocesszoros hardverek előnyeit kihasználni, ugyanakkor a világ egyre nagyobb adatbázisokat használ és nem tolerálja a lassú válaszidőket.

A kiutat az Informix cég - figyelemre méltó "rightsizing" megoldásként - az új dinamikusan skálázható architektúra megvalósításában és alkalmazásában jelölte meg.

A Dinamikusan Skálázható (méretezhető) Architektúra (DSA)

A szerver *méretezhetősége* részben a rendelkezésre álló erőforrásokkal (processzorok, diszkek, mágnesszalag egységek, stb.) való optimális gazdálkodásra, részben ezek tetszés szerinti bővítésére vonatkozik (pl. felhasználók számának, vagy a megoldandó feladatok bonyolultságának növekedése esetén).

A szerver *dinamikussága* részben a rendszer automatikus alkalmazkodóképességét, részben a menetközbeni emberi beavatkozások lehetőségét jelenti. Még a gyors és váratlan igények is kezelhetővé válnak. Az erőforrások futás közben gyorsan átcsoportosíthatók, a futó feladatok prioritásai változtathatók, ezáltal az éles (pl. OLTP) alkalmazások válaszideje nem romlik.

A DSA *architektúra* technológiai alapját a *belső párhuzamosság*, a *konkurrens többszálúság* és a *virtuális processzorok* jelentik.

- belső párhuzamosság
  Az adatbázis szervert a többszálúság és a párhuzamos feldolgozás maximális kihasználása céljából teljesen újonnan fejlesztették ki. Ezáltal lehetővé vált, hogy a párhuzamosság a rendszer mélyebb szintjein valósuljon meg.

- konkurrens többszálúság (multiple concurrent threads)
  A felhasználói kérések lekezelésénél, a processzeket kiváltó technika. A thread (egy szál) egy egyszerű végrehajtási utasítás folyam (szekvencia), amely egy diszkrét taszkként jelenik meg az operációs rendszerben. Ilyen szálak keletkeznek a felhasználó adatbázis műveletei során, amelyeket a DSA épít fel és állít sorba, majd a virtuális processzorok révén párhuzamosan, konkurrens módon hajtódnak végre.

- virtuális processzorok

  A virtuális processzorok (nagy hatékonyságú adatbázis szerver processzek) segítségével több felhasználó kérését össze lehet fogni, így kevesebb operációs rendszer processzre van szükség. A virtuális processzorok alkalmazásának további előnye, hogy amíg egy thread egy erőforrásra vár, addig a virtuális processzor egy másik kéréssel is tud foglalkozni. A virtuális processzorok száma dinamikusan változtatható az optimális végrehajtás eléréséhez.

A DSA alkalmazásából fakadó előnyök

A DSA architektúra alkalmazásával számos új lehetőség adódik a feldolgozás hatékonyságának növelésére. A legfontosabb jellemzőket és megoldásokat foglaljuk össze:

- Párhuzamos adat lekérdezés (Parallel Data Query)
- Tábla szintű particionálás
- Párhuzamos index készítés
- Párhuzamos betöltés/mentés (load/unload)
- Dinamikus shared memória
- Aszinkron I/O, előre olvasás

DB2 relációs adatbáziskezelő AS/400-on

Serfőző József, IBM Magyarországi Kft.

A hazai piacon is ismert, népszerű relációs adatbáziskezelőt igyekszik az előadás bemutatni, különös tekintettel az AS/400 platformra.

A DB2 adatbáziskező az IBM egyik legsikeresebb terméke, valamennyi IBM platformon megtalálható szinte minden IBM ügyfélnél. Történetének kezdete egybeesik a mainframe-ek megjelenésével. Ekkor fejlesztette ki az IBM azt az adatbáziskezelő technológiát, amit ma relációs adatbáziskezelésnek nevezünk. Hosszú fejlődésen ment keresztül a DB2, mire elérte mai formáját, elérte azt az állapotot, amikor meg lehetett alkotni különböző hardvereken működő implementációit, amivel egy adatbáziskezelő családdá vált. A nagygépes környezetet követően először a személyi számítógépeken futó OS/2 operációs rendszer alatt működő DB2/2 változat került forgalomba, melyet a AIX rendszer alatt működő DB2/6000 követett, majd az AS/400 relációs adatbáziskezelője is csatlakozott a családhoz.

A DB2/400 múltja, jelene

1980-ban jelent meg az IBM a System/38 rendszerrel a piacon. Legfontosabb jellemzője - amely újdonságnak számított a számítógépek körében - az, hogy az operációs rendszer integráltan tartalmazta az adatbáziskezelőt. Ekkoriban név nélkül, csak úgy a rendszer részeként létezett. 1988-ban a Sytem/38 utódjaként megjelent az AS/400 melynek operációs rendszere szintén magába foglalta az adatbáziskezelőt. SQL/400 terméknéven vált ismertté az a programcsomag amely ugyanennek az adatbázisnak az SQL szabvány szerinti elérést biztosítja. Az 1994-es év hozta az áttörést, amikor az adatbáziskezelőt felruházták mindazon tulajdonságokkal ami által beilleszkedett a DB2 családba.

Nyitottság

Sokféle szempont alapján lehet nyíltnak definiálni egy rendszert. Nyíltnak tekintem azokat, melyek különböző hardverplatformokkal, szoftverekkel, kommunikációs protokollokkal képesek együttműködni. Ezen követelményeknek a DB2/400 is megfelel. A DB2 különböző platformokon megvalósított változatai a Distributed Relational Database Arhitecture (DRDA) segítségével kapcsolhatók egy logikai adatbázissá. Az adatbáziskezelő a legismertebb 4GL fejlesztő eszközökkel is együttműködik. Egy rendhagyó megoldásról szeretnék itt most szót ejteni melyet egyedül a Progress 4GL kínál. A többi konkurens termékhez hasonló ismérvekkel rendelkezik, szintén az AS/400 DB2 adatbázikezelőjét használja, eltérés van viszont abban, hogy a kliens-szerver üzemmód mellett nativ terimálokról is futtathatók Progress alkalmazások.

Hálózatok

Az osztott adatbáziskezelés alapja az AS/400 fejlett hálózatkezelése. Napjainkban egyre gyakoribb a heterogén számítógépes környezet. Ilyen esetben elképzelhető, hogy többféle protokol (TCP/IP, Netware, stb) is aktív egyidejűleg. Ilyenkor az IBM AnyNet kommunikációs termék családja egyesíti egységes felületbe a protokollokat a programozó és a végfelhasználó számára.

Heterogenitás

A hálózatok esetében már esett szó a heterogenitásról, a probléma IBM eszközökkel történő megoldásáról. Heterogenitás nemcsak különböző hálózati protokollok jelenlétéből adódhat, hanem különböző adatbázisok egyidejű használatából is, például ha Oracle, Magic, stb. környezettel kívánjuk meglévő alkalmazásainkat kiegészíteni. Ebben az esetben az AS/400 és a DB2/400 adatbázisszerverként üzemel. Ennek az üzemmódnak a támogatása is beépül a számítógép architektúrájába.

Osztott adatbáziskezelés

Az előző pontokban már erről is esett szó. A különböző környezetekben elhelyezkedő adatok akár különböző formátumúak is lehetnek. Míg a DRDA az osztott adatbázis on-line elérését biztosítja, a DataPropagator az adatbázisok szinkronizálásának automatizált módszerét kínálja. Ez nagy eredmény, mert komoly feladatot róna a programozóra a különböző adatbázisokban tárolt adattáblák konzisztenciájának biztosítása.

Kliens/szever alkalmazás

Mivel az AS/400 (legfontosabb ismérvei alapján) adatbázis szervernek is tekinthető, magától érthető ilyen körű használhatósága. Az üzemmód további jelentősége a kliensoldali erőforrások kihasználásában rejlik. Ilyen eset lehet például multimédia alkalmazások megvalósítása, ahol a munkaállomás biztosítja a grafikus felületet, hangszintetizálást, stb.

Adatbiztonság

Adatbiztonság terén az AS/400, OS/400 és így az adatbáziskezelő is kimagasló teljesítményt nyújt. Ennek egyik alapeleme az adatok elérésének szabályozása. Ez több lépcsőben valósul meg, a rendszerbe való belépéstől kezdve, egyének, csoportok jogainak definiálásáig. Az objektumok védelme mellett az adatbiztosítás file, rekord, mező szinten is megvalósítható. A megvalósíott technika a legmagasabb felhasználói igényeket, a C2-s szabványt is kielégíti. Másik igen fontos tényező, amiről beszélni kell az AS/400 naplózási (journal) módszere. Ennek segítségével egyrészt lehetőség adódik sérült adatkörnyezet helyreállítására, másrészt megvizsgálhatók az ügyvitel szempontjából rendellenes események bekövetkezésének körülményei is (auditálás).

Integráltság

A DB2/400 nemcsak az operációs rendszerbe de a speciális (hardverközeli) mikrokódrétegbe is beépült. Ennek eredménye, hogy az adatbáziskezelőt nem kell külön installálni, egységes adatforrást biztosít, minden jogosult program hozzáférhet az adatokhoz, ami jó hatásfokot eredményez. Fontos még megemlíteni, hogy a DB2/400 is az AS/400 biztonsági és üzenetküldő rendszerére épül, ezáltal egységesen védi adatainkat, azonos módon küld mindenről értesítést, ugyanazt a programkezelői felületet biztosítja minden alkalmazás számára.

Magyar nyelvi támogatás

Itt is nagy szerepe van az adatbáziskezelő működését támogató környezetnek. Ma már teljesen természetes a nemzeti karakterek használata, az adatbázisban ABC szerinti rendezésre is. Az AS/400 ezenkívül támogatja a többnyelvű rendszereket és a különböző kódkészletek közötti automatikus konverziót is.

4GL vonások

- referenciális integritás

Új lehetőség az alkalmazás készítő számára, hogy indexek segítségével úgy rendel össze adattáblákat, hogy adott rekord törlése esetén más táblák kapcsolódó rekordjainak automatikus törlését is kérhetjük, illetve nem engedi meg azon rekordok törlését, amelyekre más táblák hivatkoznak.

- triggerek

A triggerek eseményvezérelt programozást tesznek lehetővé, adattáblához kötődnek. Ha az adatbázis valamelyik rekordjával meghatározott esemény történik, a triggerprogram definiciótól függően a változás előtt vagy után automatikusan lefut anélkül, hogy a programkészítőnek ezzel külön foglalkoznia kellene.

- tárolt procedúrák

Lehetővé teszi a munkaállomás által kezdeményezett programindítást távoli rendszereken is.

- 2 phase commit

Ez a technika az adatok integritását hivatott biztosítani osztott környezetben. A fentebb leírt funkciók feltételezik a tranzakció kezelést, vagyis adat csak akkor kerül be a táblákba, ha a kijelölt tranzakció erdményesen végrehajtódott, ellenkező esetben visszaáll az eredeti állapot.

A DB2/400 jövője

A DB2/400 jövőjének tárgyalásánál célszerű különválasztani a hardver illetve szoftver lehetőségeket.

Hardver

Az AS/400 fejlődésének iránya a PowerPC alapú 64 bites RISC technológia, ami az IBM POWER architektúrájának továbbfejlesztése. Jogosan merül fel a felhasználók, érdeklődők részéről, hogy az OS/400, a DB2/400 és a már meglévő alkalmazások hogyan fogják kihasználni a merőben új technológiát. Erre biztosíték az, hogy az AS/400 már ma egy 128 bites architektúrát valósít meg, tehát már ma készen áll a következő évek technológiájának befogadására és teljes körű hasznosítására.

Szoftver

A fenti ismertetésből egyértelműen kitűnik, hogy a DB2/400 összemérhető a piacon megtalálhtó negyedik generációs eszközökkel. Kiemelendő, hogy a DB2/400 új verziója támogatni fogja a felhasználó által definiált adattípust és utasításokat. Működését egy jól átgondolt hardver architektúra is eredményesen támogatja. Az OS/400 legfrissebb verziója az objektum alapú, C++ nyelvre épül, biztosítva az utat szabványos objektumorientált programozás felé.

SYBASE relációs adatbáziskezelő rendszer

az osztott rendszerek új generációja

Czuprik Zoltán / Nagy Dániel - Axis Számítástechnikai Kft.

BEVEZETÉS

A Sybase cég és a SYBASE RDBMS

Az 1984-ben alapított kaliforniai Sybase cég a kliens/szerver architektúrájú rendszerek vezető képviselőjeként jelentős szerepet tölt be az RDBMS technológia megújításában. 1987-ben jelent meg ***SYBASE®*** nevű termékével, amely kliens/szerver felépítésű, kifejezetten on-line alkalmazásokra tervezett RDBMS. A folyamatos fejlődés néhány további állomása:

- 1989: Secure Server
  Az első kliens/szerver API
- 1990: SMP támogatás
  IBM/MVS integráció
- 1992: Nemzetközi verziók, multimédia
- 1993: System 10
- 1994: Workgroup szerverek

A Sybase széleskörű kapcsolatrendszert épített ki az élvonalbeli hardver gyártókkal és szoftver cégekkel (SUN, HP, DEC, IBM, NCR, Microsoft, Novell stb.) közös fejlesztési programokkal biztosítva a legfejlettebb technológiák alkalmazását és a csúcsminőséget. A jelenleg is folyó együttműködési programok közül példaként említhető a SYBASE adaptálása a DEC Alpha 64 bites architektúrájához, párhuzamos adatbázis szerver fejlesztése az NCR-rel, a HP 9000 gépcsalád stratégiai fejlesztési platformként való alkalmazása a kliens/szerver architektúrájú rendszerekhez, vagy akár a SYBASE termékek integrálása a Windows NT környezetbe. A Sybase cég aktív tagja a nyílt rendszerekre és az RDBMS technológiára vonatkozó szabványok kidolgozásán munkálkodó testületeknek.

A SYBASE népszerűségét a ***SYBASE SQL Server™*** alapozta meg, de a cég nagy erőket fordított a front-end eszközök kínálatának fokozatos bővítésére is. Élvonalbeli front-end gyártó cégek beolvasztásának (SQL Solutions, Deft, Gain Technologies, Powersoft) is nagy szerepe volt abban, hogy az ***SYBASE SQL Lifecycle Tools™*** termékcsalád ma már a fejlesztői és végfelhasználói eszközök minden igényt kielégítő választékát kínálja, a CASE eszközöktől kezdve az objektum-orientált multimédia fejlesztőeszközökön keresztül az üzemeltetést osztott adatbázis környezetben támogató grafikus felületű eszközökig. Mindezt jól kiegészíti a ***SYBASE Open Interoperability Products*** termékcsalád, melyen keresztül nagyszámú független szoftvergyártó cég illesztette termékeit SYBASE környezetben való működésre. A vállalati szintű,

osztott, heterogén adatbázis környezet központosított felügyeletét és adminisztrációját a rendszerfelügyeleti eszközök (*Control Servers*) teszik lehetővé.

Az 1992 novemberében bejelentett legújabb termékverzió, a ***SYBASE System 10*** olyan kritikus területeken ad új megközelítésű megoldásokat, mint például a földrajzilag osztott heterogén adatbázisok, hibatűrő osztott adatbáziskezelés vagy a kifejezetten nagy adattömegek kezelése. A *SYBASE System 10* olyan technológiákat vonultat fel, melyek hosszú távon meghatározhatják az RDBMS rendszerek fejlődésének irányvonalát.

A Sybase a 90-es évek óta fokozatosan terjeszkedik az USA-n kívüli piacokon, ma már a világ több mint 40 országában rendelkezik képviselettel, 1994-ben jött létre a Sybase Kelet-Európa.

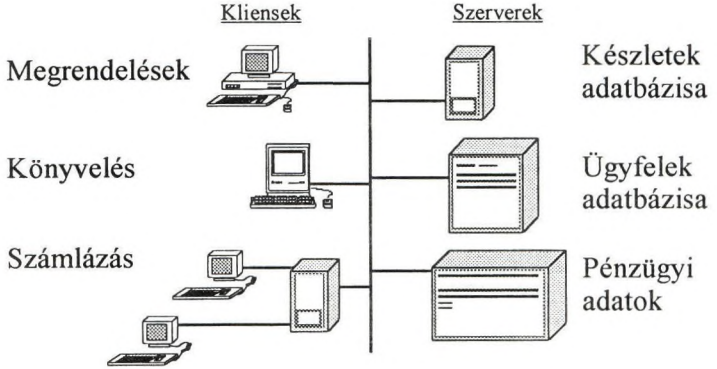
A SYBASE termékek értékesítési volumene dinamikusan nő. Az árbevétel 1993-ban 61%-kal, 1994-ben 67%-kal haladta meg az előző évit, és '94-ben elérte a 617 millió $-t. A Sybase és a Powersoft együttes árbevétele 730 millió $, ezzel a Sybase a világ 7. legnagyobb szoftver cége lett. Az első 10 szoftver cég közül a Sybase növekedési üteme a legmagasabb.

Kliens/szerver architektúra az "on-line vállalat" megvalósításához

A számítógépes adatfeldolgozás területén erősödő tendencia a kliens/szerver architektúrájú, osztott adatbáziskezelési képességekkel rendelkező rendszerek fokozatos térhódítása. Ezek a korszerű rendszerek részben a központi, nagygépre alapozott feldolgozás költségtakarékosabb és hatékonyabb alternatívái, részben pedig az elterjedt lokális hálózati rendszerek kereteit kinövő felhasználók számára biztosítják a továbblépés lehetőségét. Az igény ma a teljes vállalatot átfogó számítógépes információs rendszer, mely gyors és pontos adatszolgáltatásra képes nagyszámú on-line felhasználó számára, elosztott a vállalat területileg szétszórt egységei között és integrálja a meglévő eszközöket a PC-s lokális hálózatoktól a mainframe kategóriájú gépekig, a meglévő adatok és alkalmazások megtartásával.

Kliens/szerver = munkamegosztás

Az alkalmazások és információk megosztása - önálló rendszerek között



A SYBASE kliens/szerver architektúráját nagyfokú rugalmasság, modularitás és méretezhetőség jellemzi. A SYBASE akár többszáz felhasználó és multi-gigabájtos adatbázisok mellett is biztosítja az on-line tranzakció-feldolgozáshoz szükséges teljesítményt, védi az adatok integritását, folyamatos rendelkezésre állást tesz lehetővé, támogatja az adatok és alkalmazások földrajzi megosztását és integrálja a meglévő alkalmazásokat, - azaz biztosítja mindazon feltételeket, amelyek az "on-line vállalat" megvalósításához szükségesek.

A SYBASE kliens/szerver felépítése új határvonalat húz az alkalmazások és az adatbázis között. Ebben az architektúrában kulcsszerepet játszik a programozható SYBASE SQL Server, amely lehetővé teszi, hogy a tranzakciós logikát, vagyis az alkalmazásra jellemző általános adatkezelési ("üzleti") szabályokat közvetlenül az adatbázis szerveren tároljuk. Az ANSI szabvány szerinti SQL bővített változatával, Transact-SQL nyelven megírt, lefordított eljárások az adatbázis részeként tárolhatók és a kliens alkalmazások számára hozzáférhetők. Így, a hagyományos relációs adatbáziskezelő rendszerektől eltérően, az adatkezelési szabályokat nem kell minden egyes alkalmazói programba beépíteni. A kliens alkalmazások szerepe a felhasználói interfészeknek és a feladatvégrehajtás logikájának kezelésére korlátozódik.

A kliens/szerver architektúrájú rendszereknél a tárolt eljárások alkalmazása többféle előnnyel jár. Nő a rendszer teljesítménye, mivel a lefordított eljárások végrehajtása lényegesen gyorsabb, mint az egyedi SQL parancsoké; csökken a hálózat terhelése, mivel egy eljáráshívás összetett SQL parancssorozatot helyettesít; a szabályok központosított érvényesítésével nő az alkalmazás megbízhatósága; megnövekedik a fejlesztői produktivitás és egyszerűsödik a rendszer követhetősége is.

Sybase szoftvertermékek

A SYBASE termékek négy termékcsoportba sorolhatók:

(1) ***SQL Servers*** (relációs adatbázis kiszolgálók);

(2) A magas integráltsági fokú ***SQL Lifecycle Tools*** (életciklus-támogató eszközök) az alkalmazások tervezéséhez, fejlesztéséhez, működtetéséhez és karbantartásához;

(3) ***Control Servers*** (rendszerfelügyeleti eszközök), amelyek a vállalati szintű, osztott, heterogén adatbázis környezet központosított felügyeletét és adminisztrációját teszik lehetővé;

(4) Az ***Open Interfaces and Gateways*** (nyílt kapcsolódási felületek), amelyek heterogén adatforrások, alkalmazások és szolgáltatások rendszerbe integrálását teszik lehetővé.

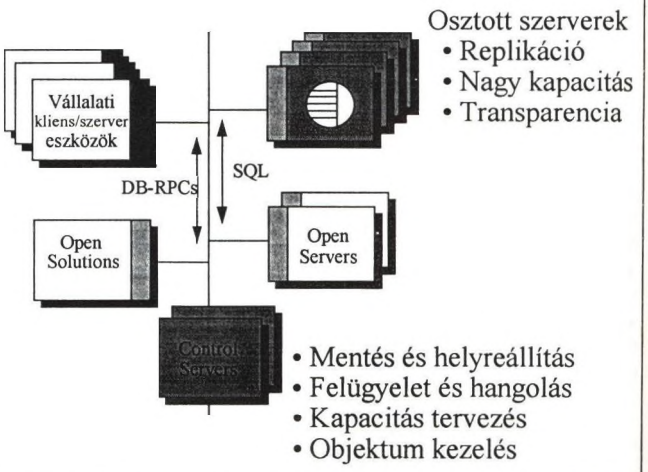
Néhány előzetes mondat a SYBASE négy termékcsaládjáról.

(1) Az SQL Server egyedülálló szinten biztosítja azt a teljesítményt és funkcionalitást, ami nélkül nagy, többfelhasználós környezetben igazi on-line tranzakció-feldolgozás csak korlátozottan valósítható meg. Jellemzői közül érdemes kiemelni a következőket.

- Méretezhető magas teljesítmény: az SQL szerver egységnyi idő alatt magas tranzakciószámot és alacsony válaszidőt garantál meglepően alacsony hardver költségvonzatok mellett. Emellett az adott alkalmazáshoz kiválasztható az optimális teljesítményű hardver- és szoftver konfiguráció. Az igények növekedésével a működő rendszer rugalmasan bővíthető, jelentse ez akár egy adott gép erőforrásainak növelését (pl. újabb processzorok hozzáadása), akár újabb szerverek integrálását, vagy egy nagyobb teljesítményű gépre való áttérést. A támogatott hardver-platformok széles skálája biztosítja az alkalmazásnak legjobban megfelelő hardvereszközök és operációs rendszerek között

való mindenkori szabad választást. A szerver-gép esetleges bővítése (vagy cseréje) esetén a SYBASE SQL Server megfelelő upgrade verziója installálható az adatok és a kész alkalmazások módosítása nélkül. A szimmetrikus multiprocesszoros gépeken a ***SYBASE Virtual Server Architecture*** biztosítja az optimális működést.

A SYBASE kliens/szerver felépítése



- Integritás és adatbiztonság: az adatkezelés eme elsődleges fontosságú szempontjai a szerver funkcióiként, az alkalmazói programoktól függetlenül érvényesíthetők. (Sem programozói, sem kezelői tévedéssel nem lehet kárt tenni az adatbázisban, illetve az alkalmazói programok tehermentesíthetők a logikai adatösszefüggések kezelésétől.) A szerver az ANSI SQL bővített változatával programozható ("intelligens szerver"). A szerveren tárolt lefordított programok (szabályok, triggerek, eljárások) biztosítják a logikai adatösszefüggések és előírások következetes betartását az alkalmazói programoktól független módon. A kiemelkedően magas biztonsági követelményű rendszerekhez a ***SYBASE Secure SQL Server*** változat áll rendelkezésre, mely a B1-es biztonsági szintnek felel meg.
- Magas rendelkezésre állási mutatók: a szerver napi 24 órás folyamatos üzemre képes. Leállás még a hagyományosan "off line" funkciók esetén sem szükséges. Az adatbázis karbantartása, mentése, diagnosztika, sőt az adatbázis struktúrájának módosítása is éles üzemben, az alkalmazói rendszer folyamatos működtetése mellett történhet. Emellett gyors hibafelderítési és javítási módszerekre, valamint szoftver alapú hibatűrő megoldásokra támaszkodhatunk.
- Nyílt osztott adatbáziskezelés: az SQL szerver támogatja az alkalmazások és adatbázisok elosztását olyan hálózatban, amelyek különféle munkaállomásokra és processzorokra épülnek. Egy alkalmazói program egyidőben több adatbázis-szerver adataihoz

férhet hozzá. Az SQL szerver által közvetlenül támogatott módszerek a kétfázisú érvényesítés (two-phase commit), mely több-szerveres tranzakciók esetén is garantálja az osztott adatbázisok konzisztenciáját, és a távoli eljáráshívás (Remote Procedure Call), mely kliensek és szerverek közötti, valamint a közvetlen szerver-szerver kommunikáció rugalmas eszköze. Az osztott adatkezelés támogatására az SQL szerverrel együttműködő további eszközök állnak rendelkezésre. A ***Replication Server*** az adatmásolatok automatikus szinkronizálását teszi lehetővé a különböző helyszínek között. Az ***OmniSQL Gateway*** teljes transzparenciát biztosít a vezető relációs adatbáziskezelő rendszerekre.

(2) Az életciklust támogató termékek a SYBASE SQL Server, illetve annak az intelligens adatszótára köré épülnek. Ez egységes és teljes átfogást biztosít az alkalmazások teljes életciklusára, nevezetesen az alábbi négy szakaszra:

- Tervezés, elemzés és konstrukció;
- Prototipizálás és fejlesztés;
- Tesztelés és belövés;
- Üzemeltetés és karbantartás.

(3) Rendszerfelügyelet, rendszeradminisztráció

A vállalati szintű, kliens/szerver architektúrájú rendszereknél elengedhetetlenül szükséges, hogy a felhasználók átfogó, hatékony eszközökkel rendelkezzenek a teljes osztott rendszerkörnyezetet központi felügyeletéhez. A SYBASE Control Serverek egységes interfészt biztosítanak rendszeradminisztrációs rutinfeledatok elvégzéséhez heterogén adatbázis-, hálózati- és operációs rendszer környezetben.

(4) Nyílt interfészek és felületek

A SYBASE a nyíltságot a szó igazi jelentése szerint értelmezi és kezeli: a nyíltság azt jelenti, hogy minden külső eszköz, alkalmazás vagy adatforrás együttműködhessen és használhasson bármilyen adatforrást, beleértve természetesen a SYBASE SQL Server-t is. A Sybase tehát inkább kooperál mintsem versenyez, ami a felhasználók szemszögéből lényegesen nagyobb szabadságot jelent: nem az RDBMS szállítója diktál nekik, hanem maguk választhatják ki a számukra leginkább szimpatikus és célszerű megoldást.

A SYBASE kliens/szerver architektúra egyedülálló együttműködést biztosít más relációs adatbáziskezelők felé. Ennek eszközei a nyílt kliens és szerver oldali interfészek és gateway-k. A SYBASE Open Server egységes hozzáférést kínál bármilyen adatforráshoz (IBM nagygépes fájlok, alkalmazások és adatbázisok; real-time folyamatvezérlők; ügyviteli rendszerek, multimédia, E-mail stb.). Ez a függetlenség egy teljes nagyvállalat egységes számítási környezetének az alapja.

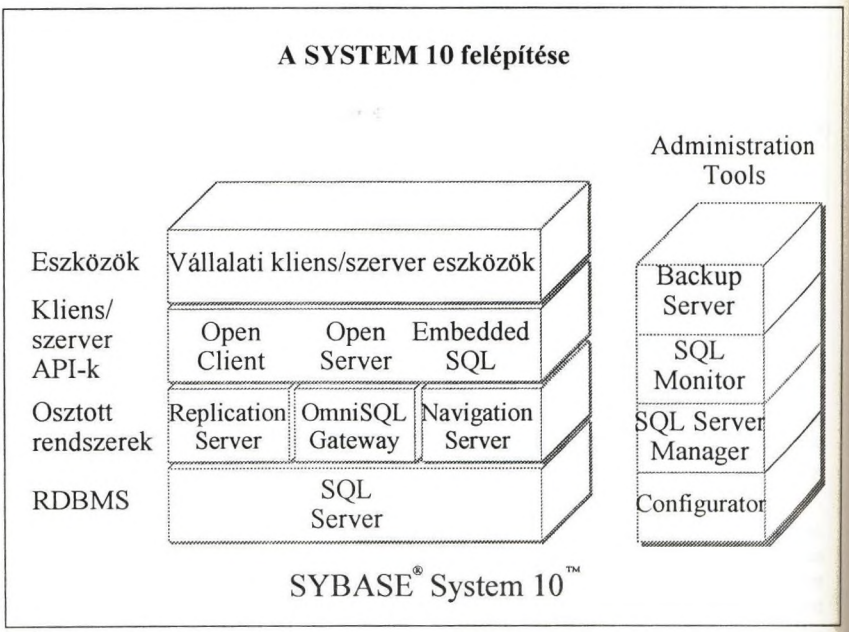
A SYBASE TERMÉKKOMPONENSEI

1. SQL SZERVEREK

1.1. A SYBASE SQL Server

A ***multi-threaded*** (többfonalas) architektúra eredményeként a SYBASE SQL Server hálózati környezetben, több száz vagy ezer felhasználó kezelése esetén is kimagasló teljesítményt nyújt. A folyamatos rendelkezésre állás és a hibatűrő megoldások az alkalmazások maximális szintű kiszolgálását teszik lehetővé.

Az adatbázis integritásának megőrzését a szerver több szinten támogatja: az adatmezők szintjén alapértelmezett értékek (default) és megengedett értékhalmaz (domain) írható elő; a táblák közötti kapcsolatokra vonatkozó szabályok (referenciális integritás) a tábladefiniálás fázisában rögzíthetők. A refereciális integritás összetettebb esetei az adatmódosító műveletek hatására automatikusan elinduló szerver programokkal, a triggerekkel kezelhetők. Többszerveres tranzakciók esetén a programozható kétfázisú érvényesítési módszerrel (two-phase commit) biztosítható az adatok konzisztenciájának fenntartása.



A SYBASE SQL Server az ANSI SQL bővített változatával, a ***Transact-SQL*** nyelven programozható, lehetővé téve az adatok, a tranzakciók és a vállalat általános érvényű szabályainak központosított kezelését. A Transact-SQL használatának előnyei különösen olyan on-line alkalmazások esetén jelentkeznek, amelyek előre definiált parametrizált lekérdezéseket és

tranzakciókat tartalmaznak. A Transact-SQL utasításokat lefordított formában az adatbázis aktív adatszótárában lehet tárolni. A Transact-SQL pluszként kínált lehetőségei kiterjednek a programvezérlési logikára, változók deklarálására, időzítéshez köthető akciók végrehajtására stb.

A SYBASE SQL Server legújabb termékverziója a ***SYBASE SQL Server 10*** tovább növeli a korábbi verziók eddig is magas teljesítményét. Az SQL Server 10 olyan adatbázis-kezelő, amely az ANSI SQL-89-es szabványon túlmutatva megfelel az ANSI SQL-92 alapszintjének (entry level) is. További jellemzői még a kimondottan nagyméretű adatbázisok támogatása és a C2-es szabványnak megfelelő adatvédelem (az U.S. Government Trusted Computer System Evaluation Criteria szerinti besorolás alapján).

A Sybase cég nyílt politikája és piaci szövetségei révén a ***SYBASE SQL Server*** a hardver/szoftver platformok széles skáláján elérhető. Az összes jelentősebb UNIX platformot és VAX/VMS környezetet támogatja. A PC-s platformokon futtatható költségkímélőbb SQL Server változatot a ***SYBASE SQL Server 10 for NetWare,*** a ***SYBASE SQL Server 10 for Windows NT***, valamint a ***SYBASE SQL Server 10 for SCO UNIX*** képviseli. A ***Microsoft SQL Server*** a SYBASE SQL Server 4.2. verziójávak az OS/2 ill. a Windows NT operációs rendszer alá portolt változata. A különböző környezetekben fejlesztett alkalmazások egymás között hordozhatók.

1.2. SYBASE SQL Server/CFT - a hibatűrő szerver

A SYBASE SQL Server/CFT (Cluster/VMS Fault Tolerance) DEC/VAX környezetben a VAXcluster lehetőségeire építve fokozza az adatfeldolgozás biztonságát. Az elsődleges SQL szerver egy készenléti változatáról van szó, amely a VAXcluster egy vagy több gépére telepítve azonnal és automatikusan átveszi az elsődleges SQL szerver funkcióját, ha az működésképtelenné válik.

1.3. SYBASE Secure SQL Server - a biztonsági szerver

A SYBASE Secure SQL Server megoldást kínál azokra az adatkezelési problémákra, amelyek többszintű biztonsági kontrollt igényelnek. Úgy tervezték, hogy kielégítse a B1-es megbízhatósági követelményeket, az U.S. National Computer Security Center operációs rendszerekre vonatkozó előírásai alapján. Emellett - fejlesztési célokra - elérhető a "SYBASE Secure SQL Toolkit" is, amely a hasonló nevű normál eszközkészlet biztonsági értelemben bővített változata.

1.4. A SYBASE Navigation Server 10

A Navigation Server 10 a mainframe kategóriájú gépek teljesítőképességét elérő rendszer kialakítását teszi lehetővé szabványos kliens/szerver eszközök felhasználásával. A párhuzamos feldolgozás módszerét alkalmazva támogatja a több Terabájt (=1000 GB) méretű adatbázisok kezelését többezer felhasználó egyidejű kiszolgálása mellett.

A Navigation Server a nagy párhuzamosságú (MPP) rendszereken, vagy szimmetrikus multiprocesszoros (SMP) hardver környezetben több *SYBASE SQL Server* működését hangolja össze, melyek az adatokat megosztva tárolják. Mind a tranzakció-feldolgozás, mind a lekérdezések végrehajtása az *SQL Server*-eken párhuzamosan történik. Az egyes lekérdezések felbontását, az egyidejűleg végezhető feladatok kiosztását és az eredmények összegzését a

Navigation Server végzi. Mindez az alkalmazások szempontjából transzparens módon történik, az alkalmazói programok csak egyetlen szerverrel, a *Navigation Server*-rel kommunikálnak. A *Navigation Server* nagy mértékben megnöveli a rendszer áteresztőképességét, túllépve a másodpercenkénti 1000 tranzakciós határt.

1.5. A SYBASE Replication Server 10

A Replication Server 10 az osztott adatkezelést új megközelítésben támogatja, az *adatbázis RPC*-k és a *kétfázisú érvényesítés* módszere mellett. A távoli szervereken tárolt adatmásolatok (replikátumok) egyezőségének automatikus fenntartása révén az on-line tranzakciófeldolgozás követelményeit kielégítő *hibatűrő osztott adatbázis-kezelést* valósít meg.

A *SYBASE Replication Server* "elsődleges" és "másolat" állományokat kezel, és a tranzakciók automatikus átvezetésével transzparens módon gondoskodik az állományok szikronizálásáról. Ha az elsődleges állomány nem elérhető, szerepét ideiglenes jelleggel egy másolat automatikusan átveheti. A hálózat vagy bármelyik helyi rendszer meghibásodása esetén a megfelelő helyszínre vonatkozó módosítások várakozási sorba kerülnek, majd a hiba elhárítását követően automatikusan végrehajtódnak.

A másolatok kezelése osztott adatfeldolgozásnál csökkenti az adatkezelő műveletek válaszidejét, mivel minden szükséges adat "helyben" elérhető, és feleslegessé válik a hálózati környezetre alkalmazható bonyolult optimalizálási módszerek kidolgozása is. Ugyanakkor a valós alkalmazói környezetben nagyobb rugalmasságot biztosít a kétfázisú érvényesítési módszerhez képest, mely működésképtelen, ha valamelyik érintett helyszín átmenetileg meghibásodik.

2. SYBASE SQL LIFECYCLE TOOLS - AZ ÉLETCIKLUST TÁMOGATÓ ESZKÖZÖK

A SYBASE környezet az alkalmazói rendszerek teljes életciklusához hatékony és magas integráltsági fokú eszközöket biztosít.

2.1. CASE Toolset - számítógéppel támogatott rendszertervezés

Annak eredményeként, hogy a Sybase cég a fejlesztői partnerkapcsolatokat stratégiai kérdésként kezeli, a SYBASE alapú alkalmazások fejlesztésére számos független gyártó CASE eszköze áll rendelkezésre. A Sybase saját CASE eszköze a "Deft".

2.1.1. A Deft

A Deft többfelhasználós szoftverfejlesztő segédeszköz az alkalmazások tervezéséhez és analíziséhez. Eszközei leegyszerűsítik és automatizálják az elemzés és tervezés folyatát heterogén adatbázis környezetű on-line alkalmazásokál. A Deft gördülékennyé teszi az alkalmazásfejlesztés kezdeti szakaszát, az esetleges hibák a fejlesztés korai fázisában kiszűrhetők, amikor még nincs jelentősebb költségkihatásuk.
A Deft editorokból és sémagenerátorokból áll. A ***Deft CASE Editor***-ok az összes ismertebb CASE módszert támogatják: Chen/Bachman, Martin, IRM, Yourdon, és Gane and Sarson. Interaktív módon szerkeszthetők egyedtípus-kapcsolat diagramok (ERD), adatfolyam diagramok (DFD), programstruktúra diagramok (PSD), és képernyőformátumok. Mind a négy editor egyetlen központi adatszótárt használ, amely a diagramokat és az adatdefiníciókat tárolja, ezál-

tal garantálva a tervezés integritását és konzisztenciáját. A ***Deft Schema Generator***-ok a CASE editorokkal előállított digramokból elkészítik a cél adatbáziskezelőnek megfelelő SQL adatdefiníciós leírást (DDL). A Deft által támogatott adatbáziskezelők: SYBASE, ORACLE, INGRES és Rdb.
A Deft eszközei lehetőséget adnak a működő adatbázisok utólagos elemzésére is. Az adatbázis séma egyszerű módon kinyerhető, előállíthatók az adatszótárak és a egyedtípus-kapcsolat diagramok. Ezek a CASE editorokkal az új kívánalmaknak megfelelően módosíthatók, majd újragenerálható az adatbázis.

A Deft hatékony grafikai eszközeire támaszkodva a különféle jelentések és programdokumentációk prezentációs minőségben állíthatók elő.

2.2. SQL Toolset - 4GL alkalmazásfejlesztő eszközök

A SYBASE SQL Toolset komplex on-line alkalmazások fejlesztéséhez biztosít hatékony eszközöket. Az alábbi termékeket foglalja magába:

- Data Workbench
- APT Workbench
- SYBASE APT-Library
- SYBASE APT Deployment Toolkit
- SYBASE Secure SQL Toolset
- SYBASE Embedded SQL
- SQR Workbench

2.2.1. SYBASE Data Workbench - adatkezelő és adatadminisztrációs programcsomag

A "SYBASE Data Workbench" vizuális (ablak technikán alapuló) eszközöknek egy integrált halmaza, amely adatbázis-adminisztrációs, jelentésgeneráló és döntéstámogató célzattal készült. Karakteres terminálokon és grafikus munkaállomásokon egyaránt működik. A következő részekből tevődik össze:

- **VQL (Visual Query Language - vizuális lekérdező nyelv):**

  az SQL-ben nem járatos felhasználók számára egy vizuális interfészt biztosít, amely végül is összetett SQL lekérdezések megkomponálását és futtatását eredményezi. Többtáblás szelekciókat és vetítéseket (project) is támogat, miközben javaslatot tesz az egyes táblák közötti valószínűsíthető kapcsolatokra (join). Ideális eszköz ad-hoc lekérdezések és döntéstámogató elemzések kivitelezéséhez.

- **Report Workbench (jelentéskészítés):**

  jelentések definiálására, tárolására és futtatására szolgáló hatékony vizuális eszköz. A jelentés támaszkodhat akár az SQL (beleértve a teljes Transact-SQL-t), akár a VQL segítségével kiválasztott adatokra. A jelentés definiálása egy alapformátum felajánlásával indul, ami az igényeknek megfelelően testre szabható.

- **Data Entry (adatbevitel):**

  automatikusan generált teljes képernyős formátumokat szolgáltat az adatbázisban való kereséshez, módosításhoz, felvételhez és törléshez.

- **Utilities (segédprogramok):**

  adatok betöltésre, kimentésére és az adatbázis konzisztenciájának ellenőrzésére szolgálnak. Megjeleníthető az adatbázis szerkezete, a korábban létrehozott SQL parancsok visszakereshetők, és lehetőség van az idő- és dátumformátum ill. a karakteres és numerikus adatok megjelenési formátumának definiálására.

- **Interactive SQL:**

  segédeszköz Transact-SQL utasítások interaktív szerkesztésére és végrehajtására.

- **Report Execute:**

  futtatórendszer a Report Workbench segítségével kifejlesztett jelentésdefiníciók alapján történő listázáshoz.

2.2.2. SYBASE APT Workbench - alkalmazásfejlesztő programcsomag

A "SYBASE APT Workbench" (APT = Application Productivity Tools) egy komplett környezet formátum-alapú alkalmazások prototipizálására és kifejlesztésére. A SYBASE APT Workbench egyaránt működik karakteres UNIX/VMS terminálokon és grafikus munkaállomásokon. A kifejlesztett alkalmazások változtatás nélkül átvihetők DOS PC-kre is. A SYBASE APT Workbench a következő részekből áll:

- **APT-Edit:**

  egy vizuális formátumszerkesztő, amellyel a formátumok és menük elrendezése és jellemzői adhatók meg, mint pl. a méret, a képernyőn való elhelyezkedés, mezőcsoportok és alcsoportok, illetve mezőérték gyanánt szolgáló halmazok.

- **APT-SQL:**

  a Sybase hatékony, könnyen elsajátítható negyedik generációs nyelve (4GL), amely az APT-Edit segítségével létrehozott formátumok logikájának és feldolgozásának a definiálására szolgál. Szintaxisa úgy tekinthető, mint a Transact-SQL-nek a formátumfeldolgozás irányában ható bővítése. Nyílt alkalmazási architektúrája révén 3GL programok is elérhetők, és ezek hívásokat tartalmazhatnak az APT-Library és az Open Client könyvtárak felé.

- **APT-Build:**

  egy prototípusgyártó és automatikus alkalmazásgeneráló eszköz többtáblás alkalmazások programozás nélküli létrehozására. Kiindulásként egy alapformátumot és APT-SQL kódot hoz létre, amely tetszés szerint bővíthető ill. testre szabható.

- **APT-Exec:**

  futtatórendszer az APT Workbench segítségével létrehozott 4GL ill. kevert 3GL/4GL alkalmazások végrehajtására.

2.2.3. A SYBASE APT-Library

Ez a könyvtár egy alkalmazásprogramozói interfész, mely lehetővé teszi, hogy az APT-Edit segítségével definiált formátumok 3GL-ből elérhetők legyenek. Lehetőség van egy formátum kezdőértékekkel való feltöltésére, a bevitt adatok visszakeresésére, a formátumnak a visszakeresett adatokkal való módosítására, valamint a formátum és a tartalmazott mezők attribútumainak a módosítására. Az APT-Library több programnyelvhez is rendelkezésre áll.

2.2.3. *A SYBASE APT Deployment Toolkit*

Az APT Workbench egyszerűsített változata. Lehetőséget nyújt a kész APT Worbench alkalmazások igény szerinti utólagos módosítására, a hibakeresésre, illetve az alkalmazások futtatására.

2.2.4. *A SYBASE Secure SQL Toolset - biztonsági SQL eszközök*

A SYBASE Secure Toolset komponensei a SYBASE Secure SQL Server-rel kompatibilisek. A fejlesztő- és futtatóeszközök "biztonsági" változatáról van szó, ilyenek a Secure Data Workbench, a Secure APT Workbench stb. A bővítések között szerepel a többszintű login és a hálózati támogatás, melynek segítségével akár közönséges, akár biztonsági szerverekkel lehet kommunikálni.

2.2.5. *A SYBASE Embedded SQL - előfordítók*

- szabványos interfész SQL parancsok beültetésére a legelterjedtebb 3. generációs programnyelvekbe (C, ADA, COBOL);
- az ANSI SQL támogatása hatékony SYBASE kiterjesztésekkel.

A SYBASE Embedded SQL előfordító révén lehetővé válik, hogy a programozó Transact-SQL parancsokat ültessen be egy harmadik generációs nyelven írt program szövegébe (mint pl. C vagy COBOL). Az előfeldolgozás során a Transact-SQL utasítások olyan kódra fordulnak le, amelyet a natív 3GL fordítója már fel tud dolgozni (a szerkesztéshez az Open Client/C-t használva). Az előfordító az ANSI szabványokhoz igazodik.

2.2.6. *SQR Workbench - jelentéskészítő programcsomag*

- **SQR - listagenerátor**

  A nem-procedurális SQL-t önálló procedurális nyelvvel kiegészítő hatékony jelentéskészítő eszköz. Segítségével a programozók csaknem minden feladatot megoldhatnak, ami egy 3GL keretein belül lehetséges, de kétszer olyan gyorsan, és félannyi programsorral. Szerkezetében egyedülálló megoldásokat támogat (pl. IF THEN ELSE és DO WHILE típusú procedurális utasításoknak közvetlenül SQL lekérdezésekbe való beágyazását).

- **Easy SQR - egyszerűsített listagenerátor**

  Az "Easy SQR" egy ablaktechnikát alkalmazó lekérdezési segédlet, amely az alkalmi és mélyebb szakértelmet nélkülöző felhasználókat segíti a jelentéskészítésben anélkül, hogy egyetlen sor programot le kellene hozzá írni. Alapformátumok széles választéka áll rendelkezésre táblázatok, formázott levelek, címlisták (és borítékcímkék) készítéséhez és adatexporthoz. Programozók is előszeretettel használják alkalmazások prototípusainak létrehozására, mivel módosítható SQR kódot generál.

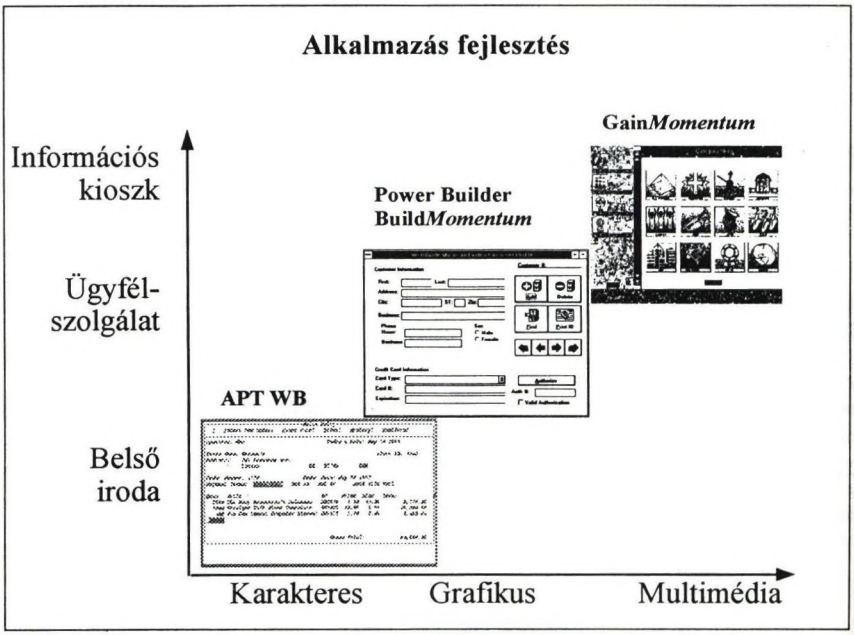
- **SQR Execute**

  Futtatórendszer az SQR listadefiníciók alapján való listakészítéshez.

2.3. *SYBASE Momentum - objektumorientált, grafikus fejlesztőeszközök*

A SYBASE Momentum termékcsalád komplex, vállalati szintű kliens/szerver alkalmazások fejlesztéséhez ad hatékony támogatást. A legmodernebb technológiákra építve jelentősen felgyorsítja a fejlesztést. Az alábbi termékeket foglalja magába:

- Build *Momentum*,
- Gain *Momentum*



2.3.1. *Build Momentum - GUI alkalmazásfejleszés*

A Build *Momentum* ojetum-orientált, grafikus (GUI) alkalmazásfejlesztő eszköz Windows, UNIX/Motif, és Macintosh környezetre. Jellemzői többek között a magas teljesítmény, a nagy fejlesztői hatékonyság, a csoportos fejlesztés támogatása. A SYBASE SQL Server-rel való teljes integráltsága mellett a SYBASE interfész eszközein keresztül más adatbáziskezelőket is támogat.

A Build *Momentum* a többfonalas felépítésű, mely lehetővé teszi a feladatok párhuzamos futtatását egyfelhasználós környezetben is (Windows 3.1, Macintosh). Így például a felhasználó egy hosszú futásidejű lekérdezés elindítása után egy másik feladattal folytathatja a munkát. A Build *Momentum* futásidőben optimalizálja a teljesítményt. Felderíti a szűk keresztmetszeteket és a leggyakrabban használt programodulokat hatékony gépi kódra fordítja.

A Build *Momentum* objektum-orientált fejlesztői környezetet biztosít a programkód és a grafikus elemek újrafelhasználhatóságának minden előnyével. A procedúrális kódolást "mutass-rá-

és-kattints" műveletekkel helyettesítő fejlesztési módszer használata az objektum-orientált technikákban kevésbé jártas fejlesztő számára is könnyen elsajátítható. A gyakorlott felhasználók teljeskörű objektum-orientált környezetben dolgozhatnak. A Build *Momentum* része egy objektum-orientált 4GL, emellett közvetlenül támogatja az APT-SQL nyelvet is, egyszerűvé téve az APT Workbench alkalmazások áttételét az új, grafikus környezetre.

A csoportmunkát verziókövetéssel, projekt kezelő eszközökkel, hozzáférési jogok kezelésével stb támogatja. Nemzetközi fejlesztői team kialakításához is segítséget ad több nyelv egyidejű használatának biztosításával.

2.3.2. Gain Momentum - integrált multimédia fejlesztői környezet

A GainMomentum általános célú, integrált fejlesztői környezet multimédia alkalmazások fejlesztésére. Olyan technológiákat egyesít, mint a hálózatkezelés, a grafikus felhasználói interfész, hypertext, az objektumorientált és vizuális programozás, SQL jellegű relációs és objektum-orientált adatbázisok kezelése.

A fejlesztők a multimédia alkalmazások prototípusának elkészítését, az alkalmazások kifejlesztését és tesztelését a hagyományos fejlesztési módszereknél szükséges idő töredéke alatt végezhetik.

A GainMomentum az adatok tárolására egy objektum-orientált adatbázist használ (Objectivity/DB). Az adatbázisban tárolt, különböző multimédia objektumok egy hypermédia réteg segítségével kezelhetők. Emellett a GainMomentum alkalmazások transzparens módon férhetnek hozzá a különböző relációs adatbázis-kezelők (SYBASE, ORACLE, INFORMIX) adataihoz is.

2.4. SQL Debug - tesztelés és belövés

Az SQL Debug interaktív forrásszintű SQL-debugger, amely egyedülálló szinten biztosítja, hogy a minőségi és a teljesítményre vonatkozó célkitűzések az alkalmazás életciklusának már korai szakaszában elérhetők. A programozók feltételes nyomkövetést és változóellenőrzést használhatnak a logikai és a névelírási hibák kiszűrésére. Lehetőség van töréspontok definiálására, ezáltal komplex procedúrák leállítására, és azon utasítások azonosítására, amelyek nem teljesítik az előírt teljesítmény-küszöbszintet. Az SQL Debug hozzáfér a SYBASE SQL Server áteresztőképességét mutató kulcs-indikátorokhoz, mint pl. a lekérdezés végrehajtási ideje vagy a fizikai I/O által igénybevett idő.

3. CONTROL SERVERS - RENDSZERFELÜGYELET

A SYBASE Control Server-ek az adminisztratív és rendszerfelügyeleti funkciókat központosítottan megvalósító termékek, melyek nélkülönözhetetlenek kiterjedt, osztott, heterogén adatbázis környezetben. Az adatbázisok mentése, a kliens/szerver környezet sebességproblémáinak diagnosztizálása, az osztott adatbázisok konfigurálása és a különböző típusú szerverek adminisztrációja a vállalati szintre kiterjedően, központilag történhet.

3.1. Backup Server 10 - adatbázis mentés/visszatöltés

A *Backup Server* feladata az adatbázisok és a tranzakció naplók nagysebességű mentése és visszatöltése, mely remote funkcióként is végezhető. Tehermentesíti az *SQL Server*-t ezen feladatok alól, amely így mentés és visszatöltés közben is változatlan, magas teljesítménnyel működhet. A Backup Server több SQL Servert képes központilag kezelni. A backup funkciók operátori beavatkozás nélkül is végrehajthatók, az automatikus backup indítását a felhasználó által definiált küszöbértékek vezérlik. Párhuzamosan 32 dump periféria kezelhető.

3.2. SQL Monitor - rendszerfelügyelet

A szerverek teljesítményének mérése az adatbázisok méretének és a felhasználók számának növekedésével fokozott jelentőségűvé válik. Az *SQL Monitor* grafikus felhasználói felülettel rendelkező eszköz a hálózat különböző pontjain lévő *SQL Server*ek teljesítményének nyomonkövetésére és hangolására. Az SQL Monitor nem terheli direkt lekérdezésekkel a szervereket, hanem az osztott memórián keresztül, real-time módon fér hozzá a szerverek által generált statisztikákhoz. A statisztikák kiterjednek a memória allokációra, a hálózati terhelésre, a CPU időfelhasználásra, a zárolási (lock) adatokra, az adat- és eljárás cache felhasználásra, a diszk I/O mennyiségére, a meghajtónkénti átlagos végrehajtási időre és a tranzakciós áteresztőképességre.

Az adatbázis-adminisztrátor meghatározhatja a kívánt statisztikákat, részletezettségük szintjét és a grafikusan is megjelenítendő adatokat. Az SQL Monitor Motif grafikus felhasználói interfésszel rendelkezik, így a statisztikák egyszerre több ablakban követhetők nyomon.

3.3. SQL Server Manager - rendszeradminisztráció

Az SQL Server Manager az SQL Serverek installálásának, a háttértár területek kezelésének és a szerverekhez való hozzáférések adminisztrálásának hatékony eszköze.

Az SQL Server Manager ablaktechnikán alapuló környezetet nyújt a rendszeradminisztráció minden nagyobb funkciójához. Segítségével komplex SQL szerver hálózatok kezelhetők anélkül, hogy a felhasználóknak valamilyen SYBASE-specifikus szintaxist meg kellene tanulniuk.

Az SQL Server Manager a kezelési funkciókat öt kategóriába sorolja: szerverek, eszközök, felhasználók, adatbázisok és jelentések. A "szerver" opció a szerverek felkapcsolására, installálására, konfigurálására, vizsgálatára, újrakreálására vonatkozik. Az "eszközök" opcióval a háttértárak listázása, hozzáadása, megszüntetése és tükrözése végezhető, a "felhasználók" a jelszavakra, felhasználókra ill. csoportokra nézve kínálnak megjelenítési, felvételi vagy törlési funkciókat. Az "adatbázis" címszó alatt listázhatunk, kreálhatunk, törölhetünk, módosíthatunk adatbázisokat ill. hozzájuk kapcsolódó objektumokat. A "jelentések" funkció a legkülönfélébb rendszerobjektumok állapotára kiterjedő parametrizált jelentések készítésére ad lehetőséget.

4. NYÍLT KLIENS/SZERVER KAPCSOLÓDÁSI FELÜLETEK

A SYBASE alkalmazásprogramozói interfészeket (API) biztosít más gyártók termékeivel való együttműködéshez vagy meglévő alkalmazásoknak az integrálásához a SYBASE kliens/szerver architektúrába. Az Open Client termékek egységes, konzisztens interfészt biztosítanak a SYBASE SQL Server-rel vagy a SYBASE Open Server-rel fejlesztett alkalmazásokkal történő kommunikációhoz. Az Open Server termékek pedig lehetővé teszik, hogy külső adatforrások, szolgáltatások és alkalmazások tudjanak SYBASE szerverként viselkedni a kliensek irányában.

A SYBASE más alkalmazásokból ODBC-n keresztül is elérhető.

4.1. A SYBASE Open Client - nyílt kliens interfész

A nyílt kliens-oldali felület lehetővé teszi, hogy a független szoftvergyártók termékei az SQL szerverrel együttműködjenek. Néhány ismertebb név a hazai viszonylatban is forgalmazott eszközök közül: QBE Vision, PowerBuilder SQLWindows, Supernova, FOCUS, MAGIC, Microsoft Access, EXCEL, Paradox, UNIFACE, harmadik generációs programnyelvek (és még hosszan sorolhatnánk, hiszen csak PC-s környezetben több, mint 150 ilyen termék található). Az ezen eszközökkel fejlesztett kész alkalmazások változtatás nélkül vagy minimális módosítással a SYBASE szerverhez illeszthetők, tetemes költségmegtakarítást garantálva egy nulláról induló fejlesztéshez képest.

A nyílt kliens programinterfészen keresztül tehát felhasználói programok, SYBASE-hez tartozó programeszközök, valamint nem-SYBASE termékek egyformán kommunikálhatnak az SQL Server-rel vagy az Open Server-rel kifejlesztett (adatforrásként viselkedő) applikációkkal. Tartalmazza mindazt a hálózatkezelési támogatást is, melynek révén az SQL Toolset összes komponense ill. az alkalmazások kliens gépeken futtatva transzparens módon kommunikálhatnak a szerverrel a hálózaton keresztül. Az Open Client több programnyelvi változatban is elérhető.

4.2. A SYBASE Open Server - nyílt szerver

A nyílt szerver-oldali felület a nem-SYBASE adatforrások integrálásának és illesztésének eszköze.

A SYBASE Open Server egy alkalmazásprogramozói interfész, amely a programozók kezébe adja olyan programok létrehozásának a lehetőségét, amelyek távoli eljáráshívásokat és SQL kéréseket fogadnak, és az adatokat visszaadják a hívónak (ami lehet egy kliens oldali program vagy egy másik SQL szerver). A SYBASE Open Server funkcióira alapozva - a már eleve meglévőkön kívül is - teljes értékű csatlakozó felületeket alakíthatunk ki a legkülönfélébb adatbáziskezelők, fájlkezelők illetve általános alkalmazói funkciók és szolgáltatások felé olymódon, hogy a SYBASE kliensek (ill. a többi SQL szerver) ezeket minden tekintetben szabványos SQL szerverként érzékelik.

A kulcsrakész gateway eszközök szintén erre a felületre épülnek. A hagyományos gateway megoldásoktól eltérően a SYBASE az adatkezelési funkciók teljes skáláját támogatja - magas teljesítmény mellett.

4.3. Kulcsrakész gateway eszközök

A SYBASE nyílt felületeire építve kész megoldások állnak rendelkezésre, amelyek biztosítják a nagygépes valamint az elterjedt RDBMS rendszerekkel való integrációt.

4.3.1. Az IBM nagygépek integrálásának eszközei

A Sybase Open Server for CICS, Net-Gateway és Open Gateway for DB2 termékei rugalmasan bővíthető megoldást kínálnak az IBM mainframe gépek kliens/szerver adatfeldolgozási környezetbe való integrálására.

Az ***Open Server/CICS*** a SYBASE környezetből lehetővé teszi egy IBM MVS nagyszámítógép CICS alkalmazásaihoz való transzparens hozzáférést. Ezen belül lehetséges bármely CICS erőforrás elérése, beleértve a DB2 statikus hívásait, DL/1 hívásokat, TS és TD sor hívásokat vagy VSAM hívásokat. Fogadja a CICS erőforrásoktól származó eredményeket, végrehajtja a szükséges adatkonverziókat és visszaadja azokat a kérelmező applikációnak.

A ***SYBASE Net Gateway*** egy belépési pontot biztosít egy IBM nagyszámítógéphez való teljes értékű hozzáféréshez. A Net Gateway a meglévő LAN installációk keretein belül teremti meg annak módját, hogy munkaállomások és PC-k elérjék a nagygép adatait. A Net Gateway egy LAN-on keresztül fogadja az SQL szervertől vagy az Open Client alkalmazásoktól származó kéréseket, majd azokat újra feladja a nagygépnek a LU 6.2-n keresztül. Eközben biztonsági és adminisztrációs kontrollt is ellát, tehát a SYBASE applikációk az IBM nagygépes környezetben megfelelően fognak viselkedni.

A ***SYBASE Open Gateway/DB2*** transzparens hozzáférést biztosít a CICS alatt futó DB2 adataihoz az SQL szerver és nyílt kliens alkalmazások részéről. Magába foglalja az Open Server/CICS-t, ennélfogva támogatja mind az SQL mind az RPC (távoli eljáráshívás) lehetőséget a CICS tranzakciók felé. Minden szükséges átalakítást elvégez a DB2 adatainak eléréséhez és aktualizálásához.

4.3.2. OmniSQL Gateway 10 - heterogén, osztott adatbázisok integrációja

A SYBASE OmniSQL Gateway 10 heterogén, osztott adatbázis környezet integrációjához ad támogatást. Lehetővé teszi, hogy az olyan vállalatoknál, ahol többféle adatbázis-kezelő rendszert használnak ill. az adatok több adatbázisban szétszórtan helyezkednek el, a fejlesztők és felhasználók látszólag egyetlen, összefüggő adatbázissal kommunikáljanak, és ne kelljen törődniük az adatok tényleges tárolási helyével, az egyes adatbázis-kezelők és SQL nyelvjárások sajátosságaival. Ennek megvalósításához többszintű transzparenciát biztosít:

Adatbázis transzparencia - A felhasználók azonos módon férhetnek hozzá a különböző adatbázis-kezelő rendszerek és adatforrások adataihoz. A jelenleg támogatott rendszerek: a SYBASE SQL Server, DB2, ORACLE, Rdb, INFORMIX, RMS és az ISAM.

A tárolási hely transzparenciája - Az adatok tényleges tárolási helyét egy globális adatszótár tartja nyilván, mely kezeli az adatforrások közötti különbségeket is, ilyenek például az adattípusok vagy az egyes objektumok elnevezésére vonatkozó konvenciók. Továbbá az adatszótárban találhatók az adathozzáférésre jogosult távoli felhasználók nevei és jelszavai is.

Nyelvi transzparencia - Az adatkezelés egységes módját a Transact-SQL nyelv biztosítja, mely a tárolt eljárások programozási nyelveként de facto ipari szabvánnyá vált. A Transact-SQL

közös fejlesztési és nyomkövetési környezetet biztosít többféle adatbázist magába foglaló alkalmazói rendszereknél. Lehetővé teszi "globális tárolt eljárások" megfogalmazását még azoknál a rendszereknél is, amelyek eredetileg nem támogatják a tárolt eljárásokat. Az OmniSQL Gateway a Transact-SQL utasításokat a célrendszer megfelelő SQL utasításaira fordítja át.

Heterogén, osztott join támogatás - A különböző adatforrások adatainak egyesítése (join) ugyanúgy végezhető, mintha egyetlen rendszerről lenne szó.

Írási/olvasási transzparencia - Az adatmódosítás és lekérdezés tetszőleges adatforrásra végrehajtható.

Központosított adminisztráció - Az adatbázis-adminisztrátori feladatok központilag elláthatók.

A támogatott adatbázis-kezelők a megfelelő *OmniSQL Access Module* segítségével kapcsolódnak az OmniSQL Gateway adatbázis motorjához (az OmiSQL Serverhez). Ezek a modulok önállóan, gateway eszközként is használhatók.

4.3.3. Önálló gateway eszközök

Az *OmniSQL Access Module* megfelelő változatai mellett további adatbázis-kezelőkhöz is készült gateway támogatás. A SYBASE Open Server alkalmazásként megvalósított gateway eszközök segítségével az INFORMIX, INGRES vagy Rdb adatbázisokban lévő adatok transzparens módon érhetők el a kliens-alkalmazásokból, ugyanúgy, mintha SQL szerver környezetben lévő adatbázisról lenne szó. A gateway eszközök automatikusan kezelik a kliens/szerver kapcsolatokat, a szükséges adattípus konverziókat, az SQL bővítmények közti leképezéseket és a hibaüzeneteket. Eme lehetőség birtokában egy szervezet a SYBASE-re való átállásnál megőrizheti a meglévő adatbázisokba, ill. adatbáziskezelőkbe való befektetéseit.

5. TÁMOGATOTT HARDVER/SZOFTVER PLATFORMOK

A SYBASE RDBMS működik minden ***UNIX*** és ***VAX*** platformon. Az olcsóbb kategóriát a ***Novell NetWare***, a ***Windows NT***, az ***SCO UNIX*** és az ***OS/2*** alatti változatok képviselik. A NetWare változatot külön is hangsúlyozzuk, mivel viszonylag alacsony árfekvés mellett biztosítja a nagy teljesítményű relációs adatbáziskezelők szolgáltatásait úgy, hogy az esetleg már meglévő Novell hálózat hatékonyan felhasználható. Kliensgépként már '286-os PC-k is felhasználhatók DOS vagy Windows alatt.

A SYBASE nyitottsága révén az alkalmazásfejlesztési eszközök széles skálája áll rendelkezésre, ez különösen igaz a PC-s fejlesztési környezetre (3GL, QBE Vision, PowerBuilder, MAGIC, FOCUS, Uniface, Smalltalk, SQLWindows, Supernova, Microsoft Access ...). Természetesen a hordozhatóság ez esetben az adott eszköz lehetőségeinek a függvénye.

A QuickObject technológia adatbázis-alkalmazásokban

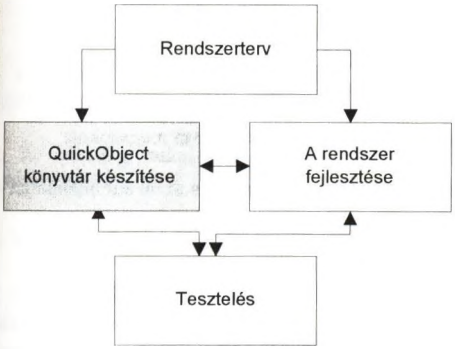
Ertner Péter (IQSOFT Rt)

Az SQLWindows a GUPTA cég (USA) fejlesztőeszköze, amely a világ egyik legjelentősebb és legtöbbször alkalmazott eszköze a kifejezetten kliens-szerver architektúrájú, Windows felületű közepes vagy nagyméretű szoftverprojektek implementálására. Az IQSoft két éve forgalmazza és alkalmazza az SQLWindows-t. Az SQLWindows 4-es verziója már tartalmazta az objektum-orientált fejlesztést támogató eszközöket. Az 1994 végén megjelent SQLWindows 5 központi eleme a QuickObject technológia, amely olyan objektum-osztályok kifejlesztését teszi lehetővé, amelyek ***fejlesztési*** időben, interaktiv módon vesznek részt az alkalmazás-fejlesztésben. A QuickObject technológia segítségével saját magunk készíthetünk generátorokat, amelyek valamely külső információ (pl. adatbáziskatalógus) alapján néhány interaktiv módon megadott paraméter és instrukció segítségével képernyőket vagy egész alkalmazásrészeket hoz létre. A QuickObject osztályokat részban vagy egészben C/C++-ban is el lehet készíteni. Minden QuickObject osztályhoz megadható egy olyan SQLWindows-ban vagy C/C++-ban írt alkalmazás (*.APP, *.EXE) - a QuickObject Design-time Interactive Editor (QDIE) -, amelyet az SQLWindows automatikusan meghív, amikor a fejlesztő egy QuickObject objektumot helyez el a tervezett képernyőn, vagy amikor később az atribútumokat változtatni szeretné. A QDIE az SQLWindows outline API-ján keresztül a forrásprogramot manipulálhatja. Az integráció hatékonyságának további foka, hogy a tool-paletta meghatározott részét is kezelhetik a QuickObject-ek, és így fejlesztési időben kommunikálhatnak egymással illetve a tervezőmérnökkel.

A QuickObject technológia egyedülállóan nyílt környezetté teszi az SQLWindows-t, amely nyomán számos cég kereskedelmi célú, felhasználásra kész QuickObject könyvtárat fog gyártani.

A QuickObjectek különösen az adatbázis-alkalmazásokban használhatók nagy hatékonysággal. Gyakori feladat, hogy egy viszonylag nagyméretű (50-nél több TABLE) adatbázisra alapuló olyan objektum-rendszert alkossunk, amely jelentősen leegyszerűsíti a tömeges tipusfeladatok implementálását. A feladatspecifikus QuickObject könyvtárak létrehozása után a programozók könnyedén implementálhatják az alkalmazást. A nem objektum-orientált, kész gyorsfejlesztő-eszközök (Oracle Forms, Magic, Progress, stb.) csak a fejlesztő cég fogalmai szerinti tipusfeladatokra hatékonyak, mivel a magasan integrált rendszerek ((post)4GL) nam adnak lehetőséget a nem default helyzetek hatékony kezelésére. Az SQLWindows fő erőssége abban van, hogy elegendően "alacsonyszintű" rendszerhozzáférést biztosit, és ugyanakkor magasszintű építőköveket kínál, hogy a QuickObject technológiában csúcsosodó objektum-orientált módszerekkel a fent említett eszközöknél sokkal hatékonyabb és ugyanakkor teljesen kézbentartott fejlesztőkörnyezetet hozzunk létre. Ezen elv arra alapszik, hogy minden valamirevaló fejlesztőcsapatban van egy-két nagyon kiváló képességű fejlesztőmérnök, akik képesek az SQLWindows építőköveiből (vagy más forrásból származó objektumokból) saját feladatorientált osztályokat létrehozni. A "tömegtermelés" ezt követően az így létrehozott

építőelemekre alapul, és nem igényel nagy mennyiségű magasan kvalifikált csúcsszakembert.
A egy cégre, vagy egy rendszerre jellemző szabványok meghatározása így viszonylag kevés, jó képességü szakember kezében tartható. A kész alkalmazás későbbi karbantartása is sokkal kisebb feladatokat jelent.
Az egyes alkalmazásokban ismétlődő objektumokat ki lehet emelni egy az egész fejlesztő cég számára hozzáférhető objektum könyvtárba.



A QuickObject technológia főbb jellemzői:

- fejlesztés idejű interaktív objektum-osztályok
  - Az alkalmazás fejlesztési fázisában is kényelmes dialógus dobozokkal vezérelt környezetben dolgozhatunk.
  - Bizonyos attribútumok beállítását kötelezővé tehetjük a fejlesztő számára.
  - Az objektumok nem megfelelő használatából adódó hibalehetőségeket a minimálisra csökkenthetjük.
- forrásprogram-manipuláló képesség
  - A QDIE-ből új objektumokat vehetünk fel a forrásprogramba, így készíthetünk olyan objektum osztályokat, amelyek az alkalmazás bizonyos részeit önállóan generálják. (pl. Az adatbázis struktúrájának megfelelően)
  - Meglevő objektumok jellemzőit lekérdezhetjük (pl. objektum-név). Ezzel a technikával könnyen készíthetünk egymással intelligensen együttműködő objektumosztályokat.
- az eszközpalettán keresztüli sztenderd fejlesztésidejű protokoll a QuickObject-ek számára
  - Az SQLWindows-ban az általunk legyártott objektumosztályok azonnal megjelennek a vizuális tervező palettán.
  - A QuickObject objektumok manipulálhatják az eszközpalettát is. Új

információkat jeleníthetünk meg, illetve lehetőségeket kínálhatunk fel a fejlesztőnek.

- részben vagy teljes egészében C/C++-ban implementálhatók (Class DLL)
  - A CDK (Component Development Kit) segítségével saját QuickObject osztályainkat elkészíthetjük C/C++-ban is. Az eszközpaletta és a forrásprogram manipulálását egy sztenderd API-n keresztül végezhetjük el.
  - Az így készült osztályok teljesen egyenrangúak az SQLWindows osztályokkal.

A GUPTA cég szállít az SQLWindows-al egy sztenderd QuickObject könyvtárat, amelynek részei:

- DataSource-ok (QuickDatabase, QuickTable)
  - Elsősorban DLL-ben implementált osztályok.
  - Az adatbázis és az alkalmazás között teremtik meg a kapcsolatot.
- Visualiser-ek
  - Editmező, rádiógomb, listadoboz és hasonló a Windows-ban megszokott elemek.
  - Ezek mindig valamilyen DataSource-hoz kapcsolódnak, és azzal autómatikusan együtt működnek.
  - QuickGraph: beépített üzleti grafikai osztály.
- Commander-ek:
  - Különböző funkciók kezdeményezésére szolgáló gombok (pl. Apply, Delete, stb.).
  - Ezek is mindig valamelyik DataSource-hoz vannak kapcsolva.
- Speciális lehetőségek:
  - Mail támogatás. A megfelelő osztályok önállóan kommunikálnak a megfelelő mail API-val.
  - Lotus Notes QuickObjectek.

Az IQSoft-ban elkészült egy magyar nyelvű QuickObject library . Ennek alapja a hagyományos objektumorientált alapokon nyugvó IQ20 Class library volt. Ennek jellemzői:

- teljes form kezelés
- intelligens táblázatok
- magyar nyelvű implementáció

Szász Eszter, Szenes Katalin
MATÁV Rt. Informatikai Intézet

A vállalati adatfeldolgozás eredményeinek konzisztenciája.
Prezentálásuk vezetői szinten
(Vázlat)

1. A vállalati információrendszerek korszerű szemléletű számítástechnikai vonatkozásai

2. A jelenleg alkalmazott adatfeldolgozási módszerek, lehetőségeik és hiányosságaik

3. Vezetői információ gyűjtése és megjelenítése a mai lehetőségekkel

4. Az új, ill. Magyarországon újnak számító
- alkalmazói rendszer-
- vezetői információs rendszer
fejlesztését támogató eszközkészletek felhasználásától várható eredmények

(A relációs, egyed - kapcsolat világszemléleten alapuló eszközkészletekhez, ill. az újra előretörő objektum-orientált filozófiához fűzött remények.)

5. Felkészülés az újfajta osztott adatfeldolgozási lehetőségekre

(Nemcsak az adatok, hanem a feldolgozó rendszer egyes részei is dinamikusan megoszthatóak, akár futásidő alatt is, mind a szerverek, mind a felhasználók között.)

Adatbázis-függetlenség és örökölt adatok megőrzése Magic-kel

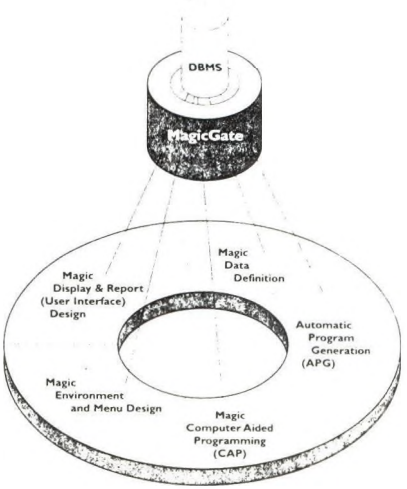
Onyx Szoftverház Kft.

A Magic, mint alkalmazásfejlesztő eszköz nem korlátozza a fejlesztőt és a felhasználót egyetlen adatbáziskezelő vagy fájlkezelő használatára és ezzel lehetővé teszi, hogy a nyílt rendszerek lehetőségeit maximálisan kihasználják.

A MagicGate termékek jelentik azt a hidat, amivel a fejlesztők és a felhasználók elérhetik a különböző adatbázisokat. Ezzel az új megoldással az információk a szervezetben közvetlenül és egyidejűleg elérhetők, legyenek azok (szinte) bármilyen platformon, illetve bármilyen formában. Mindez a fejlesztő és a felhasználó számára tökéletesen átlátszó módon történik. Azzal, hogy a Magic a legkülönbözőbb adatbáziskezelő szállítók termékeihez használható fejlesztő-eszközként, a Magic segítségével meg lehet őrizni a korábban létrehozott adatbázisokba eszközölt befektetéseket.

Lehetőség van arra is, hogy ugyanazt az adatbázis táblázatot különböző alkalmazások érjék el egyidejűleg akár helyi, akár távoli gépekről. A Magic támogatja a konkurens fejlesztést is különböző adatbáziskezelők egyidejű használata mellett.

Az adatbázis-elérések megkönnyítésére a Magic speciális segédeszközöket tartalmaz, amelyek segítségével alkalmazkodni lehet az egyes adatbáziskezelők egyedi struktúrájához. A Magic adatszótára értelmezi a használni kívánt adatbázis tárolási jellemzőit és annak megfelelően végzi el az adatbázis műveleteket. A MagicGate eszközök speciális mezőket kínálnak fel az adatbáziskezelő-specifikus jellemzők leírására. A Magic ki is bővíti az egyes adatbáziskezelők lehetőségeit oly módon, hogy megvalósít az adatbáziskezelő által nem támogatott funkciókat. Például egyes adatkezelők nem támogatják a kétirányú (előre és hátra történő) adatelérést, de ezt a MagicGate segítségével implementálni lehet.



A Magic támogatja mind az explicit, mind az implicit SQL hozzáféréseket. Mivel számos adatbáziskezelő túllép az ANSI szabvány előírásain, a MagicGate átjárók ezeket az eltéréseket az explicit és implicit SQL támogatással is kezelik:

- Az implicit SQL támogatás lehetővé teszi, hogy úgy fejlesszenek SQL adatbázisokat használó alkalmazásokat, hogy egyetlen sornyi SQL kódot se kelljen leírni. Az SQL parancsokat a Magic motor állítja elő automatikusan és teljesen átlátszó módon. Az így megírt alkalmazások teljesen adatbáziskezelő-függetlenek és így maximálisan hordozhatóak lesznek.
- Az explicit SQL támogatás lehetővé teszi SQL utasítások használatát a Magic-ből, beleértve a tárolt eljárások használatát is, a Magic-be a szükséges helyen beágyazott SQL utasítások segítségével. Az SQL-segítő támogatja a még SQL-ben kezdő programozók munkáját.

A Magic által támogatott ISAM adatkezelők: Btrieve, xBase, c-tree, Informix C-ISAM, DEC RMS, MicroFocus Cobol

A Magic által támogatott adatbáziskezelők: Oracle V6&V7, Informix, Sybase, Rdb, IBM db2

Sémagenerálás és -rekonstrukció SSADM Engineer-rel

Horváth Katalin
(KFKI-IBIS Informatikai Kft.
tel: 1695-874, fax: 1553-376)

1. Bevezetés

1.1 A séma funkcionalitása

A legkorszerűbb relációs adatbáziskezelő rendszereknél a séma ma már nem csak a létrehozandó adatbázis szerkezetét írja le egyre nagyobb részletességgel. E témakörben elsőként az adatbázis integritását biztosító automatikus ellenőrzések megadását kell megemlíteni, amelynek segítségével a sémában **deklaratívan** lehet olyan adatbázis-jellemzőket leírni, mint hivatkozási integritás, adattartalom-ellenőrzés, táblaellenőrzés.

Növekvő szerepet kapnak az olyan folyamatleíró kontsrukciók is (pl. triggerek, tárolt eljárások), amelyek segítségével a kliens-szerver architektúrájú alkalmazási rendszereknél az adatok mellett bizonyos **procedurális** részeket is központosítani lehet a szervergépen. E konstrukciók egyrészt az adatok változásához szorosan kötődő feldolgozások (ú.n. triggerek) másrészt a gyakran előforduló és több, különböző kliensgépről használandó (ú.n. tárolt) eljárások megadását teszik lehetővé. Míg az alkalmazások technikai szintjén ezek a hálózati forgalom csökkentését valamint az adatbázis integritásának megőrzését segítik elő, addig szervezési szinten a szervezeti működés alkalmazás által támogatott szabályainak egységes megfogalmazását és betartását hivatottak biztosítani.

1.2 A séma függetlensége

Az adatbáziskezelő rendszerek (ABKR) piacán lezajlott ill. várható átstruktúrálódások különösképpen előtérbe állítják azt az - egyébként is jogos - felhasználói igényt, hogy az alkalmazás legyen minél inkább független a megvalósításhoz és működtetéshez használt eszközöktől. Ez további kihívást jelent a séma készítése folyamán: olyan reprezentációját igényli az adatbázisnak, amely még független az egyes ABKR-ek egyedi jellemzőitől és nyitva hagyja az utat a különböző ABKR-ek felé.

1.3 A séma szerepe

A szerveradatbázisok ilyen módon megnövekedő jelentőségével és szerepével párhuzamosan az adatbázis szerkezetét és a kapcsolódó procedurális részeket is magába foglaló séma létrehozása kulcsfontosságú elemévé válik a teljes alkalmazásfejlesztésnek. Habár a séma adatszerkezeti alapkonstrukcióinak automatikus létrehozása már régebb óta jól megértett folyamat, amelyhez számos CASE-eszköz biztosít technológiát, a korszerű

sémák kibővült, teljes funkcionalitására ezek közül azonban már csak jóval kevesebb képes támogatást adni.

2. Az SSADM Engineer lehetőségei

2.1 Támogatott eszközök

Azok közül a technológiák közül, amelyek lehetővé teszik

- az adatbázisséma teljes funkcionalitásának eszközfüggetlen módon való reprezentálását,
- a séma automatikus létrehozását ill.
- már meglévő alkalmazási rendszerek ABKR-specifikus sémáinak eszközfüggetlen formába való alakítását,

a legkorszerűbbet kínálja az SSADM Engineer, a Magyarországon forgalmazott egyik legkedveltebb CASE-eszköz.

A sémagenerálás és -rekonstrukció csak egyik komponense (Server Builder) az LBMS angol-amerikai cég CASE-eszközének, amely hatékony, ugyanakkor eszközfüggetlen megoldást kínál a követelmények megfogalmazásától a rendszertervezésen keresztül egészen a programozásig. A kifejezetten kliens-szerver rendszerek készítését támogató eszköz a programozás területén mind a kliensoldalon (PowerBuilder, SQLWindows, Visual Basic), mind a szerveroldalon piacvezető fejlesztő/futtató eszközökkel áll szoros kapcsolatban.

A Server Builder tehát különböző szerveradatbázis-kezelőkre automatizálja a séma létrehozásának és rekonstruálásának folyamatát. A támogatás két szinten valósul meg:

- **alapszint**, amely az adatbázis adatszerkezetének definiálását jelenti (táblák, mezők, stb.)
- **emelt szint**, amely a korszerű adatbáziskezelők sémájában lehetséges összes konstrukciót jelenti (alapszint + virtuális táblák, felhasználói csoportok, hozzáférési jogok, triggerek, tárolt eljárások, integritásellenőrzés stb.)

Míg az alapszinten támogatott eszközöknél azok **száma** a leginkább kiemelendő: Informix, Ingres, Microsoft Access (1.1), Rdb (4.0 és 6.0), SQLBase, DB/2, SQL/400, xdb, WATCOM; addig az emelt szintnél a támogatás **teljessége** és **kétirányúsága** a meghatározó jegyek: ORACLE (V6 és V7), Sybase (4.2-4.9 és 10), Microsoft SQL Server.

2.2 A séma létrehozása

2.2.1 Logikai adatmodell készítése

A séma létrehozása az SSADM Engineer-ben már a fejlesztés legelején (követelmény- és rendszerelemzés) elkezdődik az ú.n. logikai adatmodell elkészítésével, amely ugyan már leírja a majdani adatbázis alapszerkezetét, de még nem tartalmazza azokat a részletinformációkat, amelyek a sémakód generálásához szükségesek.

2.2.2 Fizikai adatmodell készítése

A rendszertervezés folyamán a logikai adatmodell átalakul olyanná, amely a relációs adatmodellel már összhangban áll és relációs ABKR-re a sémagenerálást lehetővé teszi. Ennek során többek között megtörténik a szükséges kapcsoló táblák és kulcsmezők felvétele, elkészülnek a procedurális részek leírásai (triggerek, tárolt eljárások), megadásra kerülnek az adatobjektumokra a hozzáférési lehetőségek.

2.2.3 Fizikai nevek generálása

Míg a fejlesztés elején célszerű és hasznos, ha a sémaobjektumok nevei "beszédesek", azaz kimondhatóak és a szóhasználathoz közeliek, addig a rendszertervezés és különösen programozás során már a tömör, hivatkozást segítő alak a kívánatos. Az áttérés a logikai elnevezésről a fizikai többnyire jelentős erőfeszítést és körültekintést igényel. Ezt elősegítendő a Server Builder automatikusan létrehozza a cél-ABKR által használandó neveket, és a generálás közben figyelembe veszi a fejlesztő által közvetlenül megadott neveket és elnevezési szabályokat.

2.2.4 Tárigények számítása

Ez a lépés nem szükséges közvetlenül a séma generálhatóságához, de hasznos tájékoztatást ad már a megvalósítás korai fázisaiban a később létrehozandó adatbázis méretéről. A számítás felhasználja az adatállományokra és azok változására megadott összes mennyiségi információt (pl. az adatbázislap kitöltöttségi faktoránál), valamint természetesen a mezők típusát és hosszát. Fontos kiemelni, hogy a számítás különböző célkörnyezetekre is elvégezhető (ABKR + UNIX, Windows NT vagy OS/2), továbbá hogy a számított tárigény nemcsak az alkalmazási adatokra terjed ki, hanem a különböző indexekre, cluster-ekre és naplóállományokra is).

2.2.5 Sémaellenőrzés

Mielőtt a tényleges sémagenerálás megtörténne célszerű ellenőrizni a séma eszközfüggetlen reprezentációját teljesség és konzisztencia szempontjából. A tapasztalatok szerint számtalan olyan hiba és hiányosság küszöbölhető ki ilyen módon az adatbázistervből, amelyek a szokásos megközelítésben csak a programozás ill. tesztelés folyamán derül fény. Az ellenőrzés szelektíven - azaz a sémaobjektumok különböző csoportjaira (pl. táblák, mezők, eljárások stb.) egyenként - végezhető el. Ennek

segítségével fokozatosan és időben lehet a séma generálását előkészíteni, a létrejövő séma minőségét garantálni.

2.2.6 Sémagenerálás

Az előzőekben részletezett előkészítés után a generálás már meglehetősen egyértelmű tevékenység. Az ellenőrzéshez hasonlóan ez is végrehajtható szelektíven a sémakonstrukciók egyes csoportjai szerint (pl. táblák, eljárások, hozzáférési jogok stb.) valamint az iteratív sémafejlesztés támogatására a Server Builder lehetőséget ad a már adatbázisba került konstrukciók előzetes törlésére is.

2.3 A séma rekonstruálása

A sémarekonstrukció teljesen automatikus folyamat. Valamilyen megadott, létező adatbázishoz fordulva (ODBC-driveren keresztül) olvassa ki az információkat közvetlenül az adatbázis rendszerkatalógusából. Ennek eredményeképpen egy fizikai adatmodellt hoz létre (lásd 2.2.2), amelyhez felveszi az adatelemeket, procedurális modulokat, felhasználói szerepköröket és hozzáférési jogokat. Az adatbázis fizikai tárolására vonatkozó információk kivételével minden más információt rekonstruál.

3. Összefoglalás

Az SSADM Engineer és annak Server Builder komponense növeli az adatbázis-alapú információs rendszerek fejlesztésének **hatékonyságát** azzal, hogy az adatbázis sémaleírásának esetleges hibáira nem a programozás folyamán derül fény; hogy nincs szükség a sémában levő információk ismételt felvitelére a különböző ABKR-ek eltérő lehetőségei és elvárásai miatt; és hogy a séma manuális elkészítését a séma generálásával teljesen ki is lehet küszöbölni.

Segíti a meglévő rendszerek **bővítését** az aktuális sémák rekonstruálásával és az olyan, új funkcionalitás megadásával, amelyek az ABKR legkorszerűbb lehetőségeit használják ki, mint pl. tárolt eljárások és a hivatkozási integritás.

Elősegíti a rendszerek **karbantartását** a kliens- és a szerverkód valamint a fizikai adatbázis-leírás szoros integráltságának biztosításával és ezáltal az alkalmazás megfelelő szintű dokumentáltságának fenntartásával. Emellett lehetővé teszi a séma és alkotóelemei változásának követését (verzió- és konfigurációelenőrzés).

Mivel **eszközfüggetlenül** reprezentálja a fizikai adatbázisokat, így gyakorlatilag is lehetővé teszi, hogy egy szervezeten belül több, különböző ABKR-en alapuló alkalmazás is azonos szinten és módon legyen támogatva, valamint hogy az esetleges áttérés közöttük minél gördülékenyebben valósulhasson meg.

Sémagenerálás és -rekonstrukció SSADM Engineer-rel

Horváth Katalin
(KFKI-IBIS Informatikai Kft.
tel: 1695-874, fax: 1553-376)

1. Bevezetés

1.1 A séma funkcionalitása

A legkorszerűbb relációs adatbáziskezelő rendszereknél a séma ma már nem csak a létrehozandó adatbázis szerkezetét írja le egyre nagyobb részletességgel. E témakörben elsőként az adatbázis integritását biztosító automatikus ellenőrzések megadását kell megemlíteni, amelynek segítségével a sémában **deklaratívan** lehet olyan adatbázis-jellemzőket leírni, mint hivatkozási integritás, adattartalom-ellenőrzés, táblaellenőrzés.

Növekvő szerepet kapnak az olyan folyamatleíró kontsrukciók is (pl. triggerek, tárolt eljárások), amelyek segítségével a kliens-szerver architektúrájú alkalmazási rendszereknél az adatok mellett bizonyos **procedurális** részeket is központosítani lehet a szervergépen. E konstrukciók egyrészt az adatok változásához szorosan kötődő feldolgozások (ú.n. triggerek) másrészt a gyakran előforduló és több, különböző kliensgépről használandó (ú.n. tárolt) eljárások megadását teszik lehetővé. Míg az alkalmazások technikai szintjén ezek a hálózati forgalom csökkentését valamint az adatbázis integritásának megőrzését segítik elő, addig szervezési szinten a szervezeti működés alkalmazás által támogatott szabályainak egységes megfogalmazását és betartását hivatottak biztosítani.

1.2 A séma függetlensége

Az adatbáziskezelő rendszerek (ABKR) piacán lezajlott ill. várható átstruktúrálódások különösképpen előtérbe állítják azt az - egyébként is jogos - felhasználói igényt, hogy az alkalmazás legyen minél inkább független a megvalósításhoz és működtetéshez használt eszközöktől. Ez további kihívást jelent a séma készítése folyamán: olyan reprezentációját igényli az adatbázisnak, amely még független az egyes ABKR-ek egyedi jellemzőitől és nyitva hagyja az utat a különböző ABKR-ek felé.

1.3 A séma szerepe

A szerveradatbázisok ilyen módon megnövekedő jelentőségével és szerepével párhuzamosan az adatbázis szerkezetét és a kapcsolódó procedurális részeket is magába foglaló séma létrehozása kulcsfontosságú elemévé válik a teljes alkalmazásfejlesztésnek. Habár a séma adatszerkezeti alapkonstrukcióinak automatikus létrehozása már régebb óta jól megértett folyamat, amelyhez számos CASE-eszköz biztosít technológiát, a korszerű

sémák kibővült, teljes funkcionalitására ezek közül azonban már csak jóval kevesebb képes támogatást adni.

2. Az SSADM Engineer lehetőségei

2.1 Támogatott eszközök

Azok közül a technológiák közül, amelyek lehetővé teszik

- az adatbázisséma teljes funkcionalitásának eszközfüggetlen módon való reprezentálását,
- a séma automatikus létrehozását ill.
- már meglévő alkalmazási rendszerek ABKR-specifikus sémáinak eszközfüggetlen formába való alakítását,

a legkorszerűbbet kínálja az SSADM Engineer, a Magyarországon forgalmazott egyik legkedveltebb CASE-eszköz.

A sémagenerálás és -rekonstrukció csak egyik komponense (Server Builder) az LBMS angol-amerikai cég CASE-eszközének, amely hatékony, ugyanakkor eszközfüggetlen megoldást kínál a követelmények megfogalmazásától a rendszertervezésen keresztül egészen a programozásig. A kifejezetten kliens-szerver rendszerek készítését támogató eszköz a programozás területén mind a kliensoldalon (PowerBuilder, SQLWindows, Visual Basic), mind a szerveroldalon piacvezető fejlesztő/futtató eszközökkel áll szoros kapcsolatban.

A Server Builder tehát különböző szerveradatbázis-kezelőkre automatizálja a séma létrehozásának és rekonstruálásának folyamatát. A támogatás két szinten valósul meg:

- **alapszint**, amely az adatbázis adatszerkezetének definiálását jelenti (táblák, mezők, stb.)
- **emelt szint**, amely a korszerű adatbáziskezelők sémájában lehetséges összes konstrukciót jelenti (alapszint + virtuális táblák, felhasználói csoportok, hozzáférési jogok, triggerek, tárolt eljárások, integritásellenőrzés stb.)

Míg az alapszinten támogatott eszközöknél azok **száma** a leginkább kiemelendő: Informix, Ingres, Microsoft Access (1.1), Rdb (4.0 és 6.0), SQLBase, DB/2, SQL/400, xdb, WATCOM; addig az emelt szintnél a támogatás **teljessége** és **kétirányúsága** a meghatározó jegyek: ORACLE (V6 és V7), Sybase (4.2-4.9 és 10), Microsoft SQL Server.

2.2 A séma létrehozása

2.2.1 Logikai adatmodell készítése

A séma létrehozása az SSADM Engineer-ben már a fejlesztés legelején (követelmény- és rendszerelemzés) elkezdődik az ú.n. logikai adatmodell elkészítésével, amely ugyan már leírja a majdani adatbázis alapszerkezetét, de még nem tartalmazza azokat a részletinformációkat, amelyek a sémakód generálásához szükségesek.

2.2.2 Fizikai adatmodell készítése

A rendszertervezés folyamán a logikai adatmodell átalakul olyanná, amely a relációs adatmodellel már összhangban áll és relációs ABKR-re a sémagenerálást lehetővé teszi. Ennek során többek között megtörténik a szükséges kapcsoló táblák és kulcsmezők felvétele, elkészülnek a procedurális részek leírásai (triggerek, tárolt eljárások), megadásra kerülnek az adatobjektumokra a hozzáférési lehetőségek.

2.2.3 Fizikai nevek generálása

Míg a fejlesztés elején célszerű és hasznos, ha a sémaobjektumok nevei "beszédesek", azaz kimondhatóak és a szóhasználathoz közeliek, addig a rendszertervezés és különösen programozás során már a tömör, hivatkozást segítő alak a kívánatos. Az áttérés a logikai elnevezésről a fizikai többnyire jelentős erőfeszítést és körültekintést igényel. Ezt elősegítendő a Server Builder automatikusan létrehozza a cél-ABKR által használandó neveket, és a generálás közben figyelembe veszi a fejlesztő által közvetlenül megadott neveket és elnevezési szabályokat.

2.2.4 Tárigények számítása

Ez a lépés nem szükséges közvetlenül a séma generálhatóságához, de hasznos tájékoztatást ad már a megvalósítás korai fázisaiban a később létrehozandó adatbázis méretéről. A számítás felhasználja az adatállományokra és azok változására megadott összes mennyiségi információt (pl. az adatbázislap kitöltöttségi faktoránál), valamint természetesen a mezők típusát és hosszát. Fontos kiemelni, hogy a számítás különböző célkörnyezetekre is elvégezhető (ABKR + UNIX, Windows NT vagy OS/2), továbbá hogy a számított tárigény nemcsak az alkalmazási adatokra terjed ki, hanem a különböző indexekre, cluster-ekre és naplóállományokra is).

2.2.5 Sémaellenőrzés

Mielőtt a tényleges sémagenerálás megtörténne célszerű ellenőrizni a séma eszközfüggetlen reprezentációját teljesség és konzisztencia szempontjából. A tapasztalatok szerint számtalan olyan hiba és hiányosság küszöbölhető ki ilyen módon az adatbázistervből, amelyek a szokásos megközelítésben csak a programozás ill. tesztelés folyamán derül fény. Az ellenőrzés szelektíven - azaz a sémaobjektumok különböző csoportjaira (pl. táblák, mezők, eljárások stb.) egyenként - végezhető el. Ennek

segítségével fokozatosan és időben lehet a séma generálását előkészíteni, a létrejövő séma minőségét garantálni.

2.2.6 Sémagenerálás

Az előzőekben részletezett előkészítés után a generálás már meglehetősen egyértelmű tevékenység. Az ellenőrzéshez hasonlóan ez is végrehajtható szelektíven a sémakonstrukciók egyes csoportjai szerint (pl. táblák, eljárások, hozzáférési jogok stb.) valamint az iteratív sémafejlesztés támogatására a Server Builder lehetőséget ad a már adatbázisba került konstrukciók előzetes törlésére is.

2.3 A séma rekonstruálása

A sémarekonstrukció teljesen automatikus folyamat. Valamilyen megadott, létező adatbázishoz fordulva (ODBC-driveren keresztül) olvassa ki az információkat közvetlenül az adatbázis rendszerkatalógusából. Ennek eredményeképpen egy fizikai adatmodellt hoz létre (lásd 2.2.2), amelyhez felveszi az adatelemeket, procedurális modulokat, felhasználói szerepköröket és hozzáférési jogokat. Az adatbázis fizikai tárolására vonatkozó információk kivételével minden más információt rekonstruál.

3. Összefoglalás

Az SSADM Engineer és annak Server Builder komponense növeli az adatbázis-alapú információs rendszerek fejlesztésének **hatékonyságát** azzal, hogy az adatbázis sémaleírásának esetleges hibáira nem a programozás folyamán derül fény; hogy nincs szükség a sémában levő információk ismételt felvitelére a különböző ABKR-ek eltérő lehetőségei és elvárásai miatt; és hogy a séma manuális elkészítését a séma generálásával teljesen ki is lehet küszöbölni.

Segíti a meglévő rendszerek **bővítését** az aktuális sémák rekonstruálásával és az olyan, új funkcionalitás megadásával, amelyek az ABKR legkorszerűbb lehetőségeit használják ki, mint pl. tárolt eljárások és a hivatkozási integritás.

Elősegíti a rendszerek **karbantartását** a kliens- és a szerverkód valamint a fizikai adatbázis-leírás szoros integráltságának biztosításával és ezáltal az alkalmazás megfelelő szintű dokumentáltságának fenntartásával. Emellett lehetővé teszi a séma és alkotóelemei változásának követését (verzió- és konfigurációelenőrzés).

Mivel **eszközfüggetlenül** reprezentálja a fizikai adatbázisokat, így gyakorlatilag is lehetővé teszi, hogy egy szervezeten belül több, különböző ABKR-en alapuló alkalmazás is azonos szinten és módon legyen támogatva, valamint hogy az esetleges áttérés közöttük minél gördülékenyebben valósulhasson meg.

FAIR Információs Rendszerek
NJSZT