TechNet MAGAZIN

2008. MÁRCIUS – ÁPRILIS

MÉLYVÍZ, CSAK ÚSZÓKNAK!

Microsoft® SQL Server™ 2008

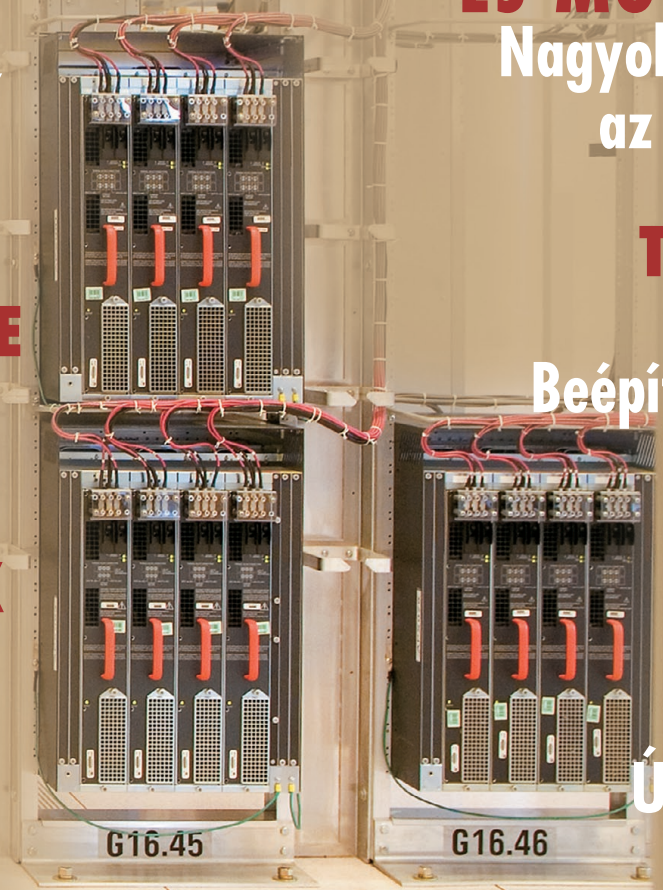
A DISTRIBUTED FILE SYSTEM
A cég növekszik, egyre több a felhasználó

AZ EGYSÉGBE ZÁRT ERŐ: ADO.NET ENTITY FRAMEWORK
Segítséget nyújtani az adatkezeléshez

TELJESÍTMÉNYHANGOLÁS ÉS MONITOROZÁS
Nagyobb teljesítmény az SQL 2008-ban

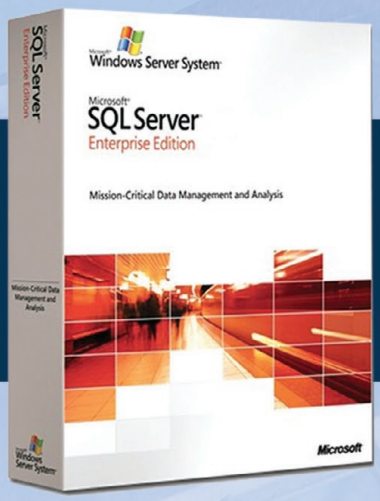
TÖMÖRÍTÉSI TECHNIKÁK
Beépített támogatás a tömörítéshez

TÚL A RELÁCIÓS VILÁGON
Új CLR- és natív típusok, új SQL-parancsok



Ára: 999 Ft
9 771586 518005 08002

Microsoft TechNet



Adatok bárhol, bármikor

Microsoft® SQL Server™ 2008

Ismerje meg Ön is tanfolyamunkon!

SQL Server 2008 üzemeltetési és fejlesztői újdonságok

2008. május 21-23.

A három napos, gyakorlatokra épülő tanfolyamot SQL Server 2005 (esetleg SQL Server 2000) adatbázis-adminisztrátorok, üzemeltetők, fejlesztők részére ajánljuk, akik szeretnék ismereteiket frissíteni SQL Server 2008-ra, és szeretnék elsajátítani az adatbázis üzemeltetési és fejlesztői újdonságait, az SQL 2008 eszközök és technológiák gyakorlati használatát.

Előadó

Árva Gábor (MCT), az SQL téma hazai szakértője, aki a márciusi, Redmondban tartott SQL Server 2008 konferencián is részt vett, így a legfrissebb információkkal, anyagokkal várja az érdeklődőket.

Kedvezményes résztvétel!

Korábbi SQL Server 2000 vagy SQL Server 2005 tanfolyamainkon résztvevők számára **50.000 Ft kedvezményt** biztosítunk a májusi időpontra. A képzésre Microsoft SA oktatási bónok felhasználhatók.

Csatlakozzon be most!

SZÁMALK Továbbképzés – Telefon: 203-0304/4122 mellék
www.szamalk.hu/tisza/sql2008

AZ ÖRÖK-MOZGÓ ADAT-BÁZIS-KEZELŐ

A Microsoft platformja az egyik legfontosabb szereplő a vállalatok informatikájában.

Nagy Levente
termékmenedzser, SQL Server 2008
Nagy.Levente@microsoft.com

A Microsoft a 90-es évek közepétől kezdve folyamatosan ostromolta a nagyvállalati informatika bástyáit és szervertermeit. A Windows NT megjelenésével egy olyan operációs rendszer alapjait tette le, amelyik egyesítette a Windows jól megszokott könnyű használatát és széles körben használt programozói interfészeit a vállalati informatikában elvárt stabilitással, skálázhatósággal.

Ezzel közel párhuzamosan a vállalati informatika egy másik nagy vonulatába, az adatbázisok piacára is belépett a Microsoft: megjelentette az első, teljesen Microsoft-színekben versenyző SQL Servert (4.2, 1994-ben). Ez a kiszolgáló számos kiadást megélt már azóta, tudása egyre csak bővült és bővült, magába olvasztva egyre újabb technológiákat, új lehetőségeket adva a fejlesztők és üzemeltetők kezébe. A hagyományos relációs adatbázis fokozatosan kiegészült egy erős OLAP-szerverrel, üzletiintelligencia-eszközökkel, beépített jelentéskészítéssel és magas rendelkezésre állást biztosító opciókkal – és mindezek azóta is a rendszer részét képezik: aki SQL Serverre bízza magát, az gyakorlatilag biztos lehet benne, hogy ezek a képességek nem drága opciók formájában, hanem alapszolgáltatásként járnak!

Idén ősszel megjelenik a legújabb SQL Server, amely továbbfejleszti a nagyvállalati funkciókört, amellett, hogy megtartja jól ismert skálázhatóságát is. Az ingyenes Express változattól a nagyvállalatok igényeit kielégítő Enterprise változatig számos újítást tartalmaz, amelyek a jelenkor adatbázis-kihívásaira válaszolnak. Új adattípusok, tömörítés, transzparens adattitkosítás, deklaratív adatbázis-menedzsment, erőforrás-priorizálás, új jelentéskészítő motor, Microsoft Office-ba integrált jelentéskészítés és számos más újítás erősíti a szoftvert.

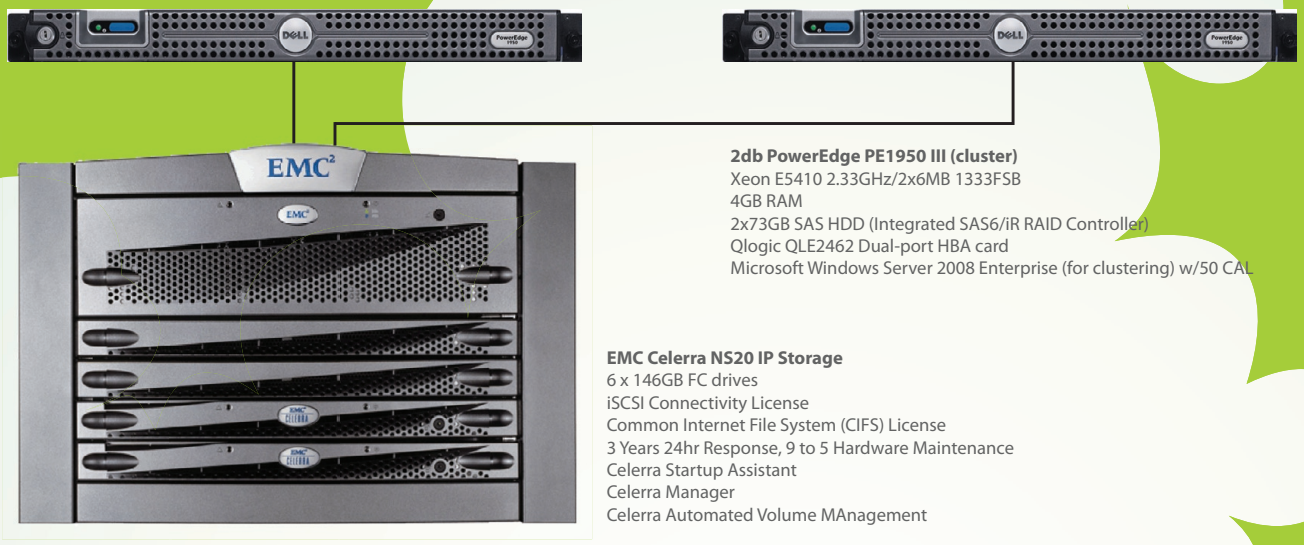
Az SQL Server 2008 próbaváltozata a Microsoft Connect honlapján – http://connect.microsoft.com/sql/ – már elérhető, érdemes kipróbálni. Ahhoz pedig, hogy segítsük a szoftver megismerését, egy külön TechNet Magazin számot szenteltünk a legfontosabb újdonságoknak.

Vágjunk hát bele!

High-Availability Cluster

DELL-EMC HA Cluster Bundle

Teljeskörű DELL-EMC megoldás kis- és középvállalatok informatikai kihívásaira



2db PowerEdge PE1950 III (cluster)
Xeon E5410 2.33GHz/2x6MB 1333FSB
4GB RAM
2x73GB SAS HDD (Integrated SAS6/iR RAID Controller)
Qlogic QLE2462 Dual-port HBA card
Microsoft Windows Server 2008 Enterprise (for clustering) w/50 CAL

EMC Celerra NS20 IP Storage
6 x 146GB FC drives
iSCSI Connectivity License
Common Internet File System (CIFS) License
3 Years 24hr Response, 9 to 5 Hardware Maintenance
Celerra Startup Assistant
Celerra Manager
Celerra Automated Volume MAnagement

Főbb előnyök

- magas megbízhatóságú hibatürő megoldás a kis- és középvállalatok részére (file kiszolgáló , levelező rendszer)
- bevizsgált és minősített megoldás
- magas megbízhatóságú komponensek
- egyszerű menedzselhetőség
- rugalmasan bővítető
- központi adattárolóba integrált Microsoft file kiszolgáló
- alap infrastruktúra a Microsoft külső kiszolgálói számára (MS Exchange 2007, MS SQL 2008…)

4.999.000 Ft

(bruttó: 5.998.800 Ft)*

Június 15-ig történő rendelés esetén 2 napos Windows 2008 Server Hands-on-Lab tréninget adunk ajándékba!

DELL EMC² where information lives Windows Server 2008

intel Xeon inside

HUMANSOFT INFORMATIKA

Microsoft GOLD CERTIFIED Partner

EMC² VELOCITY² PARTNER

Microsoft Gold és EMC Velocity Partner
– teljeskörű integrációs és támogatási szolgáltatások –

Dell | Authorised Service Provider

*Az árak 175 Ft/USD árfolyamig érvényesek és tájékoztató jellegűek, a változtatás jogát fenntartjuk! Az árak a hardver installációs szolgáltatásokat tartalmazzák, a további egyéni telepítés-igényével forduljon hozzánk bizalommal!

HUMANsoft Kft.
1037 Budapest,
Montevideo u. 8.
Tel.: (1) 270-7600
Fax: (1) 270-7679
www.humansoft.hu

Címlapon

Microsoft®
SQL Server®

Alkalmazásplatform

Infrastruktúra

TechNet MAGAZIN

SZERKESZTŐSÉG
Főszerkesztő
Sziebig Andrea – *asziebig@itbusiness.hu*
Szakmai lektor
Budai Péter – *i-pbudai@microsoft.com*
Vezető szerkesztő
Varga János – *jvarga@itbusiness.hu*
Nyomdai előkészítés
Graffaello Kft.
Korrektor
Matula Zsolt
Lapterv és címlap
Emotion Bt.

Szerkesztőség és kiadó címe
IT-Business Publishing Kft.
1072 Budapest, Rákóczi út 28.
Tel.: 577-7970, fax: 577-7995

KIADÓ
Kiadja a Microsoft Magyarország megbízásából az IT-Business Publishing Kft.

A kiadásért felel
Sziebig Andrea ügyvezető
asziebig@itbusiness.hu
Tel.: 577-7999, fax: 577-7995

Médiareferensek
Németh Krisztina – kneméth@itbusiness.hu, tel.: 577-7973
Oláh Bernadette – bolah@itbusiness.hu, tel.: 577-7972
Rátóti Sarolta – sratoti@itbusiness.hu, tel.: 577-7971

Fax: 577-7995

Terjesztés
Terjesztett példányszám: 4700

Előfizethető a kiadó ügyfélszolgálatán:
terjesztés@itbusiness.hu
Az éves előfizetés díja 4990 forint.
MCP-k számára ingyenes!

Nyomda:
Pauker Nyomdaipari Kft.
1047 Budapest, Baross utca 11-15.
Felelős vezető: Vértes Gábor ügyvezető igazgató
ISSN 1586-5185

SQL Server 2008: teljesítmény-hangolás és monitorozás

Hogy mi a különbség? Az egyik könnyebb, mint a másik...

Változnak az idők. Az ipari forradalom idején a munkások szétverték a gépeket, azzal a felkiáltással, hogy azok elveszik a munkájukat. 2008-ban az informatikusok azért ütik-verik a gépeket, mert olyan kegyetlen sok munkát adnak. De a jelek szerint a Microsoft szereti az embereket, és az SQL Server 2008-ba jó pár olyan eszközt pakolt, amelyek az azonos teljesítmény eléréséhez szükséges emberi munkamennyiség csökkentését célozzák meg. Magyarán mondva: kevesebbet kell dolgozni.

E cikk célja bemutatni azokat az eszközöket, amelyeket teljesítményhangolásra és monitorozásra használhatunk az SQL Server 2008-ban. Nem esik tehát szó azokról az elemekről, amelyek egyszerűen csak javítják a teljesítményt minden különösebb felhajtás nélkül. Így nem írok a MERGE statementről, az adattömörítésről (mind a backup, mind az adatbázis szintjén), az adattárházak életét megkönnyítő Change Data Capture-ről és a tuningolt Integration Services lookupról, abba pedig véletlenül sem megyek bele, hogy a mirroring is hatékonyabb lett. Szigorúan csak arról lesz szó, hogy miféle módokon lehet a pillanatnyinál nagyobb teljesítményt kicsalni az SQL 2008-ból, illetve arról, hogyan is tudjuk megmérni a pillanatnyi teljesítményt. Tekintettel a bőség zavarára, mindkét csoportból egy-egy új eszközt választottam ki részletesebb boncolgatásra, a többiekről pedig majd egy későbbi alkalommal esik részletesen szó.

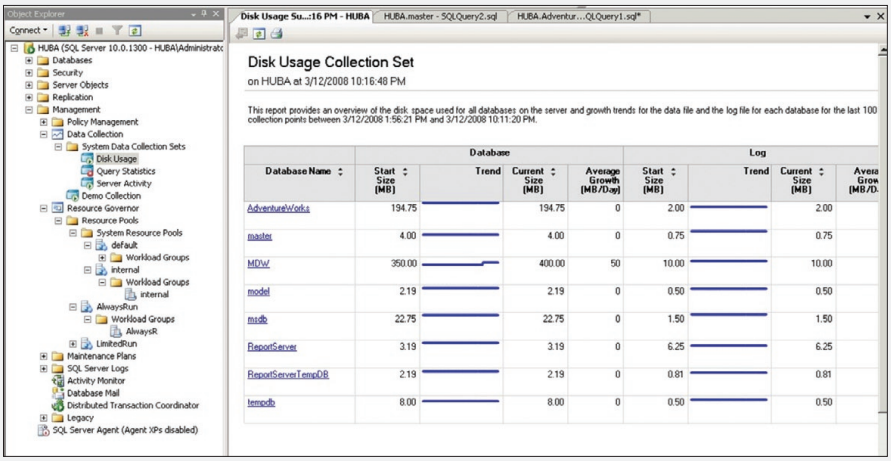
Kezdjük a monitorozással, hiszen a teljesítményhangolás célja a szervert egy nem annyira jó állapotból egy jobb állapotba eljuttatni, és megfelelő monitorozó eszközök híján úgy járunk, mint a törpök, akik folyton azt kérdezték: „Törp Papa, messze vagyunk még?”. Szóval ajánlatos valamilyen monitorozást végezni a hangolás előtt és után is, hogy lássuk, javítottunk-e a helyzeten.

Tudom, mennyi diszket használtál tavaly nyáron

A teljesítménymonitorozás sok ember fejében összekapcsolódik a teljesítményproblémával, ám itt is igaz a régi mondás: ha békét akarsz, készülj a háborúra. Azaz aki nyugodt, békés üzemeltetői pályafutást szeretne, készüljön fel arra, hogy nem lesz benne része. Esetünkben ez a baselining nevű varázslattal, azaz a rendszer normál működés során történő monitorozásával valósítható meg. Képzeljük el, hogy egy tavaszi vasárnapon jelzik: nagyon lassú az egyik adatbázis, megnézzük, és tényleg az, és kizárólag az ismert alkalmazás használja. Nekiállunk monitorozni, és azt látjuk, hogy a diszk ki van hajtva, és másodpercenként átlag 645 tranzakció zajlik az adatbázisban. Nagyszerű. Mi az összefüggés? Nehéz megmondani. Ha lenne egy baseline, amelyben látnánk, hogy mennyi volt békeidőben ugyanez a szám, akkor könnyebb dolgunk lenne, esetleg többórás izzadás helyett rögtön kiszúrnánk, hogy megtízszereződött a forgalom valamiféle új üzleti tevékenység miatt.

A teljesítménymonitorozás két régi pillére az *SQL Profiler* és a *Performance Monitor* az SQL Server teljesítményszámlálókkal. Az SQL Server 2005 pedig betömte a hiányzó lyukakat a *Dynamic Management View-k* (DMV) bevezetésével. Mit tudott még ehhez hozzátenni az SQL Server 2008? Még több DMV-t, még több performance countert, még jobb profilert, az *Extended Eventset* és a *Performance Studiót*.

A **Performance Studio** igen érdekes elem. Mindenekelőtt jó tudni, hogy ilyen nevű gombócot nem nagyon fogunk találni az SQL Server menedzsmenteszközei között. Ez ugyanis leginkább egy keretrendszer, ami a már ismert Profiler és Performance Monitor



1. ábra. A lemezfelhasználás vizsgálata

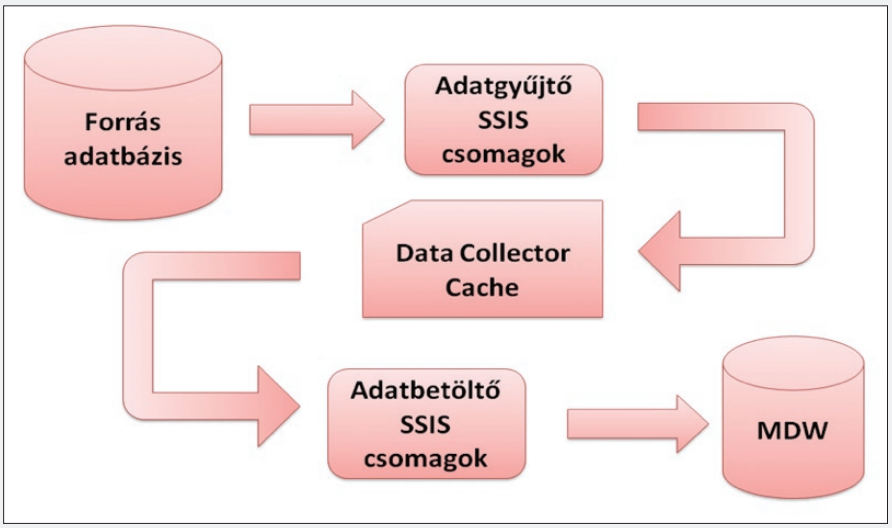
mellett a *Data Collection* nevű új eszközt (ezt hívják még *Performance Data Warehouse-nak* is) foglalja magában, plusz egy API-t a fejlesztők számára. De miért ilyen érdekes ez az eszköz? Igazából semmi újat nem tudunk meg a segítségével az SQL Server működéséről, de *ahogyan* megtudjuk a már megszokott dolgokat, az merőben újszerű. A monitorozott adatokat ugyanis elraktározza egy adatbázisban, amiből rögtön lehet csinos riportokat generálni, illetve meg is őrzi ezeket tetszőlegesen sokáig. Persze ilyet bárki tudott csinálni eddig is, de legyünk őszinték: kinek volt kedve/ideje hozzá? És mindezt, mondjuk, 19 szerveren? És nem utolsósorban: ilyen kicsi többlet-teljesítményigénnyel? Az SQL 2008-ban egy pár kattintással beállíthatjuk a Management Data Warehouse (MDW) adatbázisunkat, ahogyan ezt itt nevezik. Három beépített szerepkör van hozzá: az író (aki betölti az adatokat), az olvasó (aki riportokat készít) és az adminisztrátor (aki művelt, írni és olvasni is tud), minden monitorhoz tartozik egy alapriport, és lehet szabadon készíteni további riportokat a Reporting Services segítségével, tehát szinte egy kész, riportolásra képes alkalmazást kapunk, semmi szükség arra, hogy furfangos kódok írásával töltsük a drága időnket. Nézzük meg, hogyan is működik ez a kis kütyü *(1. ábra)*.

A Data Collector egy egységes adatgyűjtési megoldást használ különböző forrásokból származó adatok betöltésére. Ez amellett, hogy végre kényelmesen összevethetővé teszi a különböző forrásból származó eseményeket, magában hordozza azt a nagyszerű lehetőséget, hogy írhatunk hozzá saját bővítményeket is. Az adatfolyamban a következő új fogalmakkal találkozhatunk, miközben, mondjuk, a sys.dm_tran_locks dinamikus nézetből a várakozó és a teljesített lock requestek számát töltjük be a mini-adattárházunkba:

- **Data Provider.** Az adat forrása (esetünkben az SQL Server instance, ahol a lekérdezés fog futni).
- **Collector Type.** Az „egységesítő” réteg, ami az adatforrás-specifikus információt MDW-kompatibilis állapotba hozza. Jelenleg négy ilyen van: teljesítményszámlálókat, SQL Trace-adatokat, T-SQL-lekérdezéseket, illetve lekérdezésstatisztikákat gyűjthetünk. (Mi a T-SQL lekérdezést használjuk, hiszen egy lekérdezés a monitorozásunk központi eleme.)
- **Collection Item.** A Collector Type egy konkrét (és nevesített) példánya (például a Teljesitett Lock Keresek, illetve Varakozo Lock Keresek neveket adhatjuk a mi kis adatgyűjtésünknek).
- **Collection Set.** Egy vagy több Collection Itemet tartalmaz, amelyeket együttesen kezelhetünk (mi most két itemet tettünk bele a setbe). Ez az adatgyűjtési egység az adatgyűjtés során.

Data Collector. Az MDW adatbázis mellett az msdb adatbázist is igen intenzíven használja, mivel egyrészt mind az adatgyűjtést, mind az adatbetöltést apró kicsi SSIS-csomagok végzik, másrészt a saját tábláit is itt tárolja (a *syscollector* kezdetűek tartoznak hozzá).

Sokakban felvetődhet a kérdés: mennyibe kerül ez az eszköz, mennyi plusz-erőforrást igényel? Hiszen adatbázisa van, SSIS-csomagokat futtat, és elég sok adatot gyűjtöget össze. Körülbelül 5 százalék erőforrást tartsunk fenn neki, annyiból ki fog jönni (ez a szám napi 250-350 megabájt adat gyűjtésére vonatkozik, ami egyáltalán nem kevés). Természetesen érdemes csak azokat az adatokat gyűjteni, amelyek szükségesek, hiszen minél többet gyűjtünk, annál több erőforrás kell. A teljesítmény kapcsán esetleg emlékeinkből előszívároghat az ajánlás, miszerint például az SQL Trace-t diszkre kell menteni a legjobb teljesítmény érdekében, meg hasonlók. Szerencsére az SQL Server fejlesztői



2. ábra. A teljesítményadatok gyűjtésének folyamata

nemcsak írják, hanem olvassák is a saját ajánlásaikat, úgyhogy a begyűjtött adatok először egy könyvtárba kerülnek (ez a Data Collector Cache), majd onnan töltődnek be az MDW adatbázisba. Az igazán hatékony adatgyűj-

téshez az MDW adatbázist másik szerveren érdemes tartani, mint amit monitorozunk (természetesen akár több szerverhez is használhatjuk ugyanazt az adatbázist).

Az apró kényelmetlenség a Data Collector használatában az lehet, hogy nincsen GUI a collection setek készítéséhez, csak T-SQL-ben tudjuk létrehozni őket (jó pár paramétert tudunk később állítani a GUI-n keresztül). A meglévő mintákat azonban ki tudjuk scripteltetni, és azokra (és a Books Online-ra) támaszkodva egész könnyen legyárthatjuk saját collection setjeinket. A riportok készítése pedig tényleg könnyű, néhány óra alatt elkészíthetünk olyan bonyolult riportokat, éves kimutatásokat, amelyekre nyugodtan rámondhatjuk a főnöknek: egész héten ezen dolgoztam (én természetesen soha nem tennék ilyet, de a lehetőség adott).

A másik új szereplő, az **Extended Events** vagy **xEvents** egy kicsit más súlycsoport. Ez egy eseménykezelő infrastruktúra, ami az SQL Server processz futása során bekövetkező események teljes körű monitorozására hivatott. A monitorozási lehetőségek sok helyütt átfedésben vannak az SQL Profilerrel, de annál jóval mélyebben nézhetünk bele az eseményekbe.

A Profiler például mutatja a lock-eseményeket, az xEvents pedig még a latch-eseményeket is mutatja, azaz amihez létezik esemény az SQL Serverben mint fejlesztett alkalmazásban, azt megnézhetjük vele. Az elkapott eseményeket az Extended Events Engine továbbítja az általunk meghatározott eseménykezelőknek (targets), amelyek ennek hatására csinálnak valamit (action). Eseményfeldolgozó lehet például az Event Tracing for Windows (ETW) is, egy eseményfájl, vagy az Event Pairing, ami megadott szabályok alapján párosítja az eseményeket, és segítségével könnyen megtalálhatóak az egyedülálló események, például a lock, amit elfelejt elengedni egy processz.

Az Extended Events egy roppant érdekes terület azoknak, akik már jártasabbak az SQL Server lelkivilágában, de meg kell fizetni érte: ehhez az eszközhöz semmilyen grafikus segédlet nem létezik, kizárólag T-SQL-ben (EWT-hez plusz parancssorban) lehet kezelni. Aki szeretne jobban megismerkedni vele (remélem, vannak ilyenek), az kezdetnek megnézheti a gyűjthető eseményeket az alábbi lekérdezéssel:

```
select dxo.name, dxo.description, dxp.name
from sys.dm_xe_objects dxo
join sys.dm_xe_packages dxp
on dxo.package_guid = dxp.guid
where dxo.object_type = 'Event'
```

Mint említettem, az új eszközök mellett a régiekkel is foglalkoztak a fejlesztők. Az SQL Profilert nagyon nem kell bemutatni: az üzemeltetők és fejlesztők egyik legjobb barátja, mivel könnyen, kényelmesen és gyorsan lehet vele megtudni, hogy pontosan mit is csinál a szerverünk, milyen utasítások végrehajtásával van elfoglalva, mindezt kényelmesen össze is kattogtathatjuk. Nos, most már van benne környezetérzékeny integráció: Management Studióból az eddig megszokott Tools – Profiler – kézi beállítás helyett egy gombnyomásra (ikonkattintásra) nyílik az SSMS mellé egy új Profiler-ablak, ami már be van konfigurálva, hogy az éppen aktív SPID-re szűrjön. Nem nagy dolog, de megint nyerek vele egy kis időt.

Túrjunk bele!

A monitorozó eszközök után nézzük, miként lehet elérni, hogy a Performance Data Warehouse-unkból megnyugtató riportokat generáljunk. A triviális út az, hogy belejavítunk az adatokba, de próbáljuk meg a másik módszert: tuningoljuk az SQL-szervert egy kicsit. Már említettem, hogy elég sok teljesítményjavító újdonság van az SQL Server 2008-ban, ezek egy része azonnal rendelkezésünkre áll, mindennemű módosítás nélkül, egy részéhez pedig azért kell egy kicsit módosítanunk. Ezekre itt külön nem térek ki, kizárólag a klasszikus hangolásról lesz szó. Induljunk ki egy képzeletbeli esetből: van egy alkalmazásunk, mögötte egy adatbázisunk SQL Server 2008-on, és egy borús pénteki napon azzal talál meg minket a helpdesk, hogy az alkalmazás felhasználói bejelentések szerint botrányosan lassú, mi pedig megállapítjuk, hogy igazából az adatbázis a gyenge láncszem (erre a feladatra a Profiler és a Perfmon tökéletesen alkalmas). Itt kezdődik az igazi stratégiai játék, ugyanis egyrészt meg kell találnunk a Perfmonnal, hogy milyen hardvererőforrásnak van szűkében a szerver, másrészt meg kell találnunk a Profilerrel, hogy miféle query okozhat ilyen erőforráshiányt.

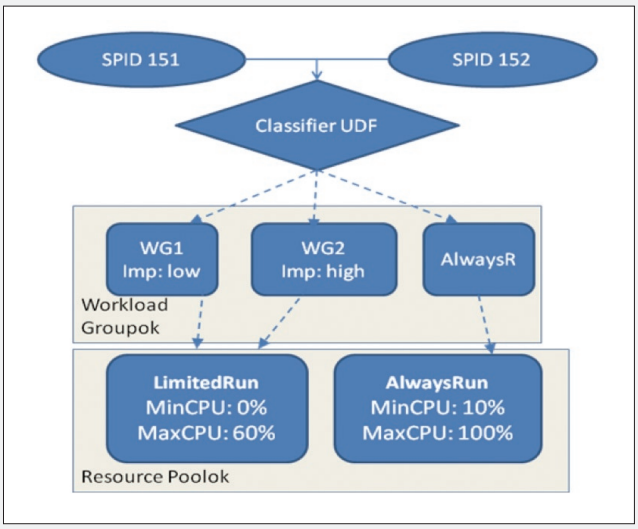
Itt elérkeztünk az első valódi teljesítményhangoló eszközhöz: bevethetjük a már jól ismert automata fegyvert, a **Database Engine Tuning Advisort** (DETA), amelynek odaadjuk a Profiler trace-t, és az majd kitalálja, hogy mit kell tenni az adatbázissal, hogy jobb legyen: indexeket, statisztikákat, particionálásokat ajánl, akár a meglévő mellé, akár ahelyett, ez csupán beállítás kérdése. Aki használta a DETA egyébként igen hasznos szolgáltatásait, az most kicsit kétkedve néz esetleg, hogy miért szeretném órákkal eltolni a megoldást, mert a DETA amellett, hogy jó, lassú is, ami végül is érthető, csak nem mindig elfogadható. Egyszerűbb megoldás a Management Studióban nekiállni a végrehajtási terveket bogarászni, ugyanakkor így adjuk meg magunknak az esélyt igazán nagy hibák elkövetésére, hiszen csak kiragadott részleteket próbálunk optimalizálni, esetleg a többiek rovására. Ez persze nem elrettentés akar lenni, csupán figyelmeztetés.

Vannak azonban olyan teljesítményproblémák, amelyeket nem igazán lehet optimalizálással megoldani. Például ha a lassulás oka az, hogy Robotka Robika riportokat készít a cég 15 éves történetéről napi és személy szerinti bontásban, akkor más eszközhöz kell nyúlnunk (nem Robika megfenyítésére gondolok). Ez a más eszköz pedig a **Resource Governor**, ami felszámolja az üzemeltető valós idejű erőforrás-zsonglőr tevékenységi körét. A Resource Governor az, amit a neve sugall: hatalom az erőforrások felett. Jó, igazából pontosítani kell, az erőforrások körébe egyelőre csak a processzor és a memória tartozik, ezeket lehet osztani, illetve elvenni, de az I/O-szabályozás is rajta van a fejlesztők listáján.

Az alapséma könnyen átlátható a *3. ábra* segítségével: definiálhatunk Resource Poolokat, amelyeknek kioszthatunk egy minimális és egy maximális százalékértéket memóriából és CPU-ból. Továbbá definiálhatunk Workload Groupokat, amelyekbe a processzek kerülnek majd, és ezeket a workload groupokat hozzárendelhetjük a resource poolokhoz, valamint adhatunk nekik prioritást is egy hármas skálán. Azt, hogy egy processz melyik workload groupba kerül, egy classification function fogja eldönteni. De nézzük meg mélyebben, hogyan is működik ez a konstrukció!

A resource poolhoz rendelt minimum-erőforrások azt határozzák meg, hogy mennyi

erőforrás lesz fenntartva mindenképpen az adott poolnak, a maximum pedig azt, hogy amennyiben az erőforrásért versengés folyik, legfeljebb mennyit kaphat a pool. Tehát egy adott resource pool lehetséges erőforrás-felhasználását meghatározza a többi pool minimumértékének az összege. A minimumok összege természetesen nem lehet 100 fölött, és amennyiben pont kijön a 100 százalék, gyakorlatilag előre le van osztva minden erőforrás. Érdemes tehát jól meggondolni, hogy kinek és miért akarunk minimumértéket beállítani, ez a minimum ugyanis kárba vész, ha nem használjuk. Ezek után senki sem fog meglepődni, ha a ResPool1 néha 80 százalék processzoridőt használ el – ha van szabad kapacitás, akkor azért nem nehezíti az életet az Erőforrás Kormányzó. Viszont 90 százaléknál többet sosem használhat, mivel 10 százalék mindig le van foglalva a ResPool2 számára. A workload group olyan csoport, amibe megadott szabályok alapján kerülnek az SQL-processzek. Ez a besorolás kapcsolódáskor történik meg, a processz szintjén, később semmilyen úton nem módosítható.



3. ábra. Finomhangolható teljesítménybeállítások

Viszont a workload groupokat egyik resource poolból a másikba átmozgathatjuk bármikor, illetve a workload groupok és resource poolok beállításait is tetszés szerint módosíthatjuk, amikor csak akarjuk.

Az SQL Server 2008-ban két beépített resource pool és két beépített workload group található: ezeket internalnak és defaultnak hívják. (A Data Collectionhöz kapcsolódó képen azt is láthatjuk, hogyan néz ki mindez a Management Studióban.) Az internal workload groupba sorolja az SQL a saját maga számára kritikus processzeket, és ezt az internal resource poolhoz rendeli. Ez nem szerkeszthető, nem lehet ide sorolni user processzeket, nem lehet korlátozni – és természetesen annyi erőforrást használ, amennyit akar, függetlenül a többi pool beállításától. A default workload group a „gyűjtő", azok a processzek kerülnek ide, amelyek nem illettek be máshova. A classification function, ami a session létrejöttekor lefut, egy stringet ad vissza, és ezt a szerver workload group névként értelmezi. Ha nincs olyan nevű workload group, vagy elbukik a classification function, vagy nincs classification function, akkor irány a default, ami a hasonló nevű resource poolban dolgozik. A default resource pool paraméterei állíthatók ugyan, de érdemesebb létrehozni egy saját poolt, szebb is, és nem kér enni, ha nem használjuk (hacsak nem adtunk meg minimumértékeket). Mindenképpen fontos tudni, hogy a classification function még a logon része, és ha nem fut le a timeouton belül, akkor a kliens a connection timeout élményében részesül. Ezért különösen fontos, hogy a classification function gyors legyen (még egy érv: a Resource Governor connection pooling esetén is minden sessionre meghívja a classification functiont).

A processzor- és memóriaelosztás mellett még van néhány hasznos paraméter, amit lehet állítani egy workload groupnál. A prioritást már említettem, ez akkor hasznos, ha egy resource poolban több workload group is van. Emellett megadhatjuk a párhuzamosan futó kérések maximális számát, ha ennél több érkezne, akkor azok várni fognak egy szintén meghatározható timeout ideig. Ha esetleg sikerült egy olyan konfigurációt összehozni, ami annyira rossz, hogy még a saját megváltoztatásában is megakadályoz, akkor sem kell kétségbe esni: a Dedicated Admin Console (DAC) az internal resource poolba tartozik, azon a kiskapun mindig bejuthatunk.

Ez eddig jól hangzik, de hogyan tudom megnézni, hogy valójában mennyi memóriát eszik a resource poolom, illetve jól működik-e a classification function, amit írtam? A válasz kézenfekvő: az új, illetve módosított menedzsmentnézetekben. Az új *sys.dm_resource_governor_resource_pools* nézetben találjuk a számokat, a *sys.dm_exec_sessions* és *sys.dm_exec_requests* nézetek pedig gazdagodtak egy group_id oszloppal. És el ne felejtsük: vannak új performance counterek is hozzá...

Ha rendelkezésre állnak olyan lekérdezéseink, amelyek néha jól futnak, aztán elromlanak, akkor a **Plan Guide/Plan Forcing** segítségével egy jó végrehajtási tervet megragadva azt használhatjuk, amíg boldogan élünk. Fel kell hívnom azonban a figyelmet arra, hogy ha rossz tervet mentünk el, akkor szemrebbenés nélkül rontja a teljesítményt az SQL-szerver. Természetesen ezeket a terveket bármikor eldobhatjuk, ha már nem kellenek.

Hab a tortán: **FORCESEEK**. Mindig nagy dilemma az optimizernek, hogy clustered index scant használjon vagy nonclustered index seeket, és aztán keresse meg a rekordokat drága bookmark lookup műveletekkel a WHERE feltétel feldolgozásakor. Eddig erre a covering index lehetett megoldás: felvettük azokat az oszlopokat is az indexbe, amelyeket le akartunk kérdezni, illetve ennek szebb változata az included oszlopok SQL 2005-ben. A FORCESEEK hint segítségével az optimizer dilemmáját szüntethetjük meg, hiszen arra utasítjuk, hogy a nonclustered index segítségével dolgozzon. A Plan Guide és FORCESEEK használatához természetesen tartsuk szem előtt az alapelvet: csak akkor vegyük el az optimizer munkáját, ha biztosan tudjuk, hogy mit teszünk.

Egy szó, mint száz: az SQL Server 2008 hangolhatósága igen sokoldalú: akár gyorstapaszra, akár időtálló megoldásra, akár csak erőforrás-elosztásra van szükségünk, mindent megtalálunk ebben az eszköztárban. És munkánk eredményességét minden hétfő reggel szebbnél szebb grafikonok is bizonyíthatják – ha beállítottuk a Performance Studióban.

Bitemo Erik
(erik.bitemo.hu)
MCDBA, MCTS, senior adatbázis-adminisztrátor
Walt Disney Magyarország

TÖMÖRÍTÉSI TECHNIKÁK

A Microsoft SQL Server 2008 beépítetten támogatja a tömörítést.

Elsőképp azt érdemes átnézni, hogy milyen lehetőség volt a korábbi verziókban az adatok kisebb helyen való tárolására. Az első, a Microsoft által támogatott adattömörítési technika az SQL Server 2005-ben jelent meg. Korábban az SQL Server 2000-ben a teljes adatbázist tömörített lemezre rakhattuk (írás és olvasás esetén is működött), de ez egy, a Microsoft által nem támogatott lehetőség volt. Ez a helyzetet változott meg az SQL Server 2005 megjelenésével, amikor csak olvasható (read only) adatbázis, vagy fájlcsoport esetén lehetett tömörített lemezen tárolni az adatainkat. De ez bármennyire jó öletnek tűnt, nem volt érdemes használni, mert megjósolhatatlan sebességcsökkenéssel járhatott a lekérdezésekre nézve.

Az SQL Server 2005 SP2-től kezdődően Enterprise és Developer verzió esetén a vardecimal adattárolási típust használhatjuk, amely a varchar, nvarchar, varbinary típusokhoz hasonló elven működik. Fontos, hogy kliensoldalon ebből semmit sem látunk, továbbra is ugyanolyan formában kapjuk meg a lekérdezett adatunkat, így nem kell módosítanunk a már meglévő kódunkon.

Ennek az adattárolási típusnak a decimal, numeric adattípusokhoz képest az az előnye, hogy ha nem használjuk ki az adott mezőnél a maximálisan rendelkezésre álló helyet, akkor az SQL-motor kisebb helyen fogja tárolni az adatainkat. Ahhoz, hogy a vardecimal tárolási módot használhassuk az adott adatbázis (sp_db_vardecimal_storage_format), illetve tábla szintjén (sp_tableoption) is engedélyezni kell:

```
USE master
EXEC sp_db_vardecimal_storage_format 'AdventureWorks','ON'
USE AdventureWorks;
EXEC sp_tableoption 'Production.WorkOrderRouting',
  'vardecimal storage format', 'ON';
```

Azt, hogy érdemes használnunk ezt a tárolási formát, az exec sp_estimated_rowsize_reduction_for_vardecimal 'táblaneve' futtatásával tudhatjuk meg. Például az adventureworks adatbázison lefuttatva a következő kódot.

```
exec sp_estimated_rowsize_reduction_for_vardecimal 'Purchasing.PurchaseOrderDetail'
```

	avg_rowlen_fixed_format	avg_rowlen_vardecimal_format	row_count
1	56.00	51.94	8788

Azt láthatjuk,hogy így az átlagos rekordhossz 56 bájtról 51.94 bájtra csökkenne. Még egy nagyon fontos dolog van a változó hosszúságú adatok tárolásával kapcsolatban, mégpedig az, hogy ha nem fix hosszú az adat, akkor minden egyes rekordban tárolni kell azt is, hogy az adott rekord adott mezőjében milyen hosszú az aktuális adat. Ezt az úgynevezett „column offset array" tartalmazza, mégpedig úgy, hogy megmutatja, hol található meg az adott adattartalom. Ezt két bájton tárolja, így könnyen belátható, hogy például egy char(2) hosszú mezőt nem érdemes varchar(2)-ként tárolni, mert az 1 karakter hosszú adat tárolásához is 3 bájt kell, szemben a char(2) 2 bájtjával. És ez így igaz a kisebb típusok esetében is, például az int (4 bájt), a smallint (2 bájt) és tinyint (1 bájt) tárolásához szükséges hely tovább már nem csökkenthető ezzel a módszerrel. Így el is érkeztünk az SQL Server 2008 egyik új adattárolási formájához, az úgynevezett rekordtömörítéshez (row compression), amelyet az Enterprise és Developer verziók támogatnak.

A rekordtömörítés megvalósításánál a következő szempontokat vették a figyelembe a fejlesztők:

- A numerikus adatokat (float, int, bigint, smallint) és azokat a típusokat, amelyek a numerikus adattárolási formán alapulnak (datetime, money) a vardecimal adattárolási módhoz hasonlóan lehessen tárolni.
- Fix hosszú mezőket, például char(100) változó hosszúságban tárolni úgy, hogy a feltöltő karaktereket nem tárolják.
- Az adatrekord leírásához szükséges hely csökkentése (mezők hossza, mezők kezdete és vége az adott rekordban). Ezzel a technikával az SQL Server a maximum 8 bájt helyet elfoglaló adattípusok – például bigint (8), int (4), smallint (2), char(8),

Adattípus	Leírás
smallint	Ha az adat 1 bájton elfér, akkor csak 1 bájton tárolja az adatot
int	Csak annyi bájtot használ, amennyire szükség van
bigint	Csak annyi bájtot használ, amennyire szükség van
decimal	Vardecimal tárolási forma
numeric	Vardecimal tárolási forma
bit	4 bit metaadat plusz
smallmoney	Intként tárolva (10 000-rel felszorozva)
money	Bigintként tárolva (10 000-rel felszorozva)
float,real	A nullákat nem tárolja. A real típusnál a mantissza nem tört részét tömöríti leginkább
datetime	Maximum 2 bájt megtakarítása
datetime2	A tárolt értéktől függő tömörítés (2-3 bájt megtakarítás)
datetimeoffset	Maximum 2 bájt megtakarítás
char	A kitöltő karakter levágva
nchar	A kitöltő karakter levágva
binary	A kitöltő karakter levágva (0)
timestamp/ rowversion	Intként tárolva

datetime(8) – esetén csak 4 extra bitet használ fel (az eddigi 2 bájttal szemben) arra, hogy változó hosszúságú adatként tárolja az adott mező értékét.

- A NULL, illetve 0 értékek ne foglaljanak el helyet.

Arról, hogy a rekordtömörítés mely adattípusokra van hatással az előző oldali táblázat szól. A rekordtömörítésen felül az adatlapokat is tömöríthetjük (page compression), ami a következő algoritmus szerint működik: hajtsuk végre a rekordtömörítést, majd szűrjük ki az adott adatlapon belül egyező értékeket. (Kétlépéses algoritmus, az úgynevezett „column-prefix", és „pagedictionary" lépésekből áll, ezeket később fejtem ki). Így ezekeket csak egyetlenegyszer tárolja el, attól függetlenül, hogy hányszor szerepel az adott adatlapon belül.

Nézzük meg a gyakorlatban egy teoretikus példán keresztül, hogy ez mit is jelent:

```
create database TestForCompress
go
use TestForCompress
create table Product(
ProductID int identity(1,1),
Name varchar(80) not null default '',
Status smallint not null default 0,
ProductCategory int not null default 0,
CONSTRAINT PK_ProductID PRIMARY KEY (ProductID)
)
GO
set nocount on
Insert into Product (Name) values (SYSDATETIME())
go 10000
select top 10 * from Product
```

Az eredményt a fenti ábra szemlélteti.

Nézzük meg, hogy milyen helymegtakarítással járna a rekordtömörítés alkalmazása.

```
exec sp_estimate_data_compression_savings 'dbo','Product',NULL,NULL,'ROW'
```

A lenti táblázatot így kell értelmezni: tábla neve, séma neve, index azonosítója (0 = heap, 1 = clustered index, 1> = nonclustered index), partíció száma (1, ha nincs partícionálva), a jelenlegi beállításokkal lefoglalt hely, illetve a lekérdezett tömörítési beállításokkal mennyi helyet foglal majd el. (Figyelem, nem minden mezőt jelenítettem meg!)

Ebből az következik, hogy a jelenlegi tábla a rekordtömörítés után kb. 20 százalékkal kevesebb helyet foglal el. Nézzük meg azt is, hogy a lapok tömörítése után mekkora lesz a megtakarítás:

```
exec sp_estimate_data_compression_savings 'dbo','Product',NULL,NULL,'PAGE'
```

Így már jóval jelentősebb a megtakarítás, hiszen kevesebb, mint 1/3 helyen lehet eltárolni ugyanazokat az adatokat. De ismerjük be, ez nem életszerű, mivel a nevek ilyen nagymértékben nem hasonlítanak egymáshoz. (Az adatlap-tömörítés az egyező értékeket csak egyszer tárolja, így a Name mezőben a „2008-03-07 10:12:36." részt és a 0 értékeket is.)

Results | Messages

	ProductID	Name	Status	ProductCategory
1	1	2008-03-07 10:12:36.7625024	0	0
2	2	2008-03-07 10:12:36.7625024	0	0
3	3	2008-03-07 10:12:36.7625024	0	0
4	4	2008-03-07 10:12:36.7625024	0	0
5	5	2008-03-07 10:12:36.7725168	0	0
6	6	2008-03-07 10:12:36.7725168	0	0
7	7	2008-03-07 10:12:36.7725168	0	0
8	8	2008-03-07 10:12:36.7725168	0	0
9	9	2008-03-07 10:12:36.7725168	0	0
10	10	2008-03-07 10:12:36.7725168	0	0

Hogy ne legyen ilyen sok egyezőség, futtassuk le az alábbi scriptet:

```
update product
set name=case when productid/2*2=productid then cast(productid as varchar(80)) else cast(newid() as varchar(80)) end,
            status = cast(productid/2*3 as smallint),
            ProductCategory=cast(ProductID/7*-RAND(100)*100 as int)

select top 10 * from product
```

Eredmény:

	ProductID	Name	Status	ProductCategory
1	1	8634DEFA-1363-488C-ADF5-AFBC6ABB25CF	0	0
2	2	2	3	0
3	3	7B2B7CD2-F69E-4D5C-9DAF-B3F2CA5B91E4	3	0
4	4	4	6	0
5	5	815FA3A0-88E3-448F-808E-FABD0FDFF319	6	0
6	6	6	9	0
7	7	15774E4E-69D8-444D-BA60-6054AB27D28B	9	71
8	8	8	12	71
9	9	26BF9194-3F09-4AFF-A3A5-E0125B1CA473	12	71
10	10	10	15	71

Tekintsük meg, hogy mennyi lenne a megtakarítás, ha tömörítenénk az adatlapokat:

```
exec sp_estimate_data_compression_savings 'dbo','Product',NULL,NULL,'PAGE'
```

A becsült helymegtakarítás 20 és 40 százalék közötti lehet az adattartalomtól függően.

Nézzük meg a valóságban is:

```
exec sp_spaceused 'product'
```

Kapcsoljuk be az adatlap-tömörítést. (Ez a rekordtömörítést is használja.)

```
ALTER TABLE product
REBUILD WITH (DATA_COMPRESSION = PAGE);
GO
exec sp_spaceused 'product'
```

A tömörítést nemcsak a táblára, hanem adott fájlcsoportra, indexre is be lehet kapcsolni, de ne felejtsük el, hogy a tömörítés használatakor további erőforrásokra (processzor) lesz szükség, és a tömörítés eredményessége mindig az adott adattartalomtól függ, így nem minden esetben éri meg a használata. Két eset, amikor szinte biztos, hogy érdemes használni a Page compressiont:

- Olyan fájlcsoportoknál, amelyek nem, vagy nem gyakran változnak (például read only).
- Azoknál az indexeknél, amelyeket csak ritkán használja a rendszer.

Azok az esetek, amikor nem érdemes használni a tömörítést:

- Amikor az eredeti tábla méretéhez képest túl kicsi a megtakarítás.
- Nagyon sok írás/olvasás történik az adott objektumra.
- Az objektum mérete az adott adatbázis méretéhez képest kicsi.

Azt, hogy mely objektumokat használják viszonylag keveset a lekérdezések az adott adatbázisban, az sys.dm_db_index _operational_stats DMV (Dynamic Management View) lekérdezésével tudhatjuk meg.

Nézzük meg, hogyan működik a „column prefix compression". Az SQL Server, a „column compression" használatakor oszlopon-

	object_name	schema_name	index_id	partition_number	size_with_current_compression_setting...	size_with_requested_compression_setting(KB)
1	Product	dbo	1	1	520	416

Helymegtakarítás a tömörítés használatával

	object_name	schema_name	index_id	partition_num...	size_with_current_compression_setting...	size_with_requested_compression_setting(KB)
1	Product	dbo	1	1	520	160

Helymegtakarítás a lapok tömörítése után

size_with_current_compression_setti...	size_with_requested_compression_setti...	sample_size_with_current_compression_setting(KB)	sample_size_with_requested_compression_setting(KB)
520	416	448	360

Helymegtakarítás az adatlapok tömörítése után

kénti mintát keres, mégpedig úgy, hogy olyan értékek után kutat az adott adatlapon, amelyek minden rekordban előfordulnak. Ahogy az elnevezésből is következik, csak kezdeti egyenlőségeket keres – például a következő két hexadecimális értékben csak a 0x11AA részt tekinti egyenlőnek: 0x11AABBCCDDAA, 0x11AACCCCDDAA. A „DDAA" részt pedig nem tekinti egyezőnek.

Hogy miért van ez így? Mintákat keresni az adatlapokon költséges művelet (főleg a processzort terheli), ezért a fejlesztők úgy döntöttek, hogy csak a kezdeti egyenlőségeket dolgozzák fel, a többit „veszni hagyják". Ebből következik, hogy a 0xAABBCC és 0x11BBCC értékekre nem tud tömöríteni az SQL Server.

Mielőtt tovább mennénk, fontos leszögezni, hogy bájtmintát keres az SQL Server, így típustól függetlenül minden adattípusra ugyanúgy működik a tömörítés.

Nézzünk egy gyakorlati példát. Képzeljük el a következő táblázat szerinti adatstruktúrát, amelynek egy adatlapját jelenítjük meg, az áttekinthetőség miatt szavakat használok, nem pedig hexadecimális számokat:

Fejléc-információ (header)			
1 adatlap (8K)	ID	Termék név	Státusz
1. rekord	A01	Asztal	X01
2. rekord	A02	Asztalra vissza	X03
3. rekord	A03	Asztalos	X02

Tömörített adathoz létrejön egy úgynevezett „anchor" rekord, amelynek szerkezete ugyanaz, mint a többi rekordnak:

Fejléc-információ (header)			
„Anchor" rekord	A03	Asztalra vissza	X02
1 adatlap (8K)	ID	Terméknév	Státusz
1. rekord	A01	Asztal	X01
2. rekord	A02	Asztalra vissza	X03
3. rekord	A03	Asztalos	X02

Az „ID" mezőből a rekordok közül az A03 értéket választotta ki az algoritmus, így az 1. rekordban a 21 azt jelenti, hogy az „anchor" mintából az első 2 bájtot használja, és ezután a maradványértéket hozzárakja. A 2. rekordban a szintén csak az első két bájtot használja, majd a maradványérték kerül hozzá, így lesz az értéke 22. A 3. rekordnál pedig a teljes minta használható, így „null" értéket tárol az SQL Server. A többi mező értékeit ugyanígy feldolgozva jön létre az alábbi táblázat:

Fejléc-információ (header)			
„Anchor" rekord:	A03	Asztalra vissza	X02
1 adatlap (8K)	ID	Termék név	Státusz
1. rekord	21	6	21
2. rekord	22	Null	23
3. rekord	Null	6os	Null

Mindezeket követően, ha az adatlap-tömörítés is be van kapcsolva, akkor a következő táblázathoz hasonlóan nézhet ki az eddig bemutatott adatlap:

Fejléc-információ (header)			
„Anchor" rekord:	A03	Asztalra vissza	X02
Adatlapszótár: 21			
1 adatlap (8K)	ID	Terméknév	Státusz
1. rekord	0	6	0
2. rekord	22	Null	23
3. rekord	null	6os	Null

A táblázatot a következőképpen kell értelmezni. Az SQL Server megkeresi azokat az értékeket az adott adatlapon, amelyek megegyeznek, és szótárként használja őket. Pirossal jelöltem meg a táblázatban a változásokat. Mivel a 21 érték két mezőben szerepelt, ezért az SQL kiválasztotta szótár értéknek, majd ezeket „0" értékkel helyettesítette be. Így nem magát az értéket tárolja, hanem csak azt, hogy a szótárban hányadik elemet foglalja el.

Backup tömörítési technika

A mentéseket tömöríteni a korábbi SQL Server-verziókban csak más gyártók termékeinek megvásárlásaival lehetett. Sajnos csak Enterprise és Developer verzió esetén használhatjuk a tömörítést, azt pedig, hogy mi legyen az alapértelmezett mentési forma, szerver- és adatbázisszinten adhatjuk meg. Az így beállított érték akkor jut érvényre, ha a BACKUP parancs futtatásakor nem adjuk meg, hogy tömörítse vagy ne tömörítse az adatokat.

```
USE master;
GO
EXEC sp_configure 'backup compression default', '1';
RECONFIGURE WITH OVERRIDE;
```

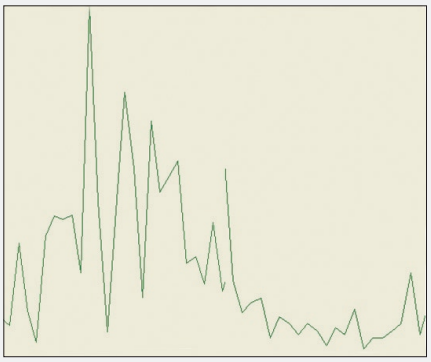
A mentésnél a WITH záradék egészült ki a COMPRESSION opcióval. Lefuttatva az alább scriptet a virtuális gépemen 15 másodperc alatt futott le a BACKUP a COMPRESSION opcióval, és 39 másodperc alatt a tömörítés nélküli. A fájlméret 39,4, illetve 172 megabájt. Látható, hogy a tömörítés gyorsabb volt, és kevesebb helyet foglalt el.

```
declare @datetime datetime=GETDATE()
BACKUP DATABASE Adventureworks
To DISK = 'C:\adventureworks.bck'
WITH INIT,COMPRESSION
select getdate()-@datetime

set @datetime=GETDATE()
BACKUP DATABASE Adventureworks
To DISK = 'C:\adventureworks1.bck'
WITH INIT
select getdate()-@datetime
```

Mint mindennek, a tömörítésnek is ára van. Az alábbi ábrán látható, hogy a tömörített mentés jóval nagyobb processzorterheltséggel jár.

A tömörített mentéseket bármelyik SQL Server-verzió vissza tudja állítani, azt, hogy az adott backup fájl tömörített formátumú



3. ábra. Processzorterheltség

	BackupName	BackupDescription	BackupType	ExpirationDate	Compressed
1	NULL	NULL	1	NULL	1

vagy sem, a RESTORE HEADERONLY paranccsal nézhetjük meg, ahol a Compressed mező 1 értéke jelenti azt, hogy a mentés tömörítve van.

Konklúzió

Bár a háttértároló rendszerek kapacitása egyre nő, és az árak is csökkennek, de az adatok tömörítésével a rendelkezésre álló hely tovább növelhető, és nem elhanyagolható tényezőként a kiírásra, olvasásra, mentésre és visszaállításra fordított idő a Microsoft SQL Server 2008-ban megjelent megoldásokkal jelentősen csökken.

Sáfár István
(isafar@cleware.hu), Architect

Declarative Management Framework

Hogyan és mivel lehet megfegyelmezni az SQL Server 2008-at.

Az SQL-szerver új verziója jelentősen átalakítja a menedzsmentfeladatok elvégzését. Eddigi eszközeinkkel – jobok, triggerek, profiler – korábbi SQL-verziókban követhettük a szerveren zajló változásokat és reagálhattunk rájuk. Jobokkal automatizálhattunk folyamatokat, sőt ellenőrizhettünk szerver- és adatbázis-opciókat, ha azok nem voltak megfelelőek, akkor változtathattunk rajtuk. Ellenőrizhettük, hogy az előre eltervezett beállításoknak megfelelően működik-e szerverünk, ha nem, esetleg egy DDL triggerben visszagördíthettük, egy jobban beállíthattuk, ha nem így történt. Ha egyszerre több szerveren akartunk valami hasonlót, akkor már bonyolult volt ezt megvalósítani, frissíteni.

Kialakíthattunk mindenféle névkonvenciót a táblák, triggerek, objektumok nevére, szerver-, illetve adatbázis-opciókra vonatkozóan. (Például minden tárolt eljárás neve proc_* alakú legyen, éles adatbázis recovery-modellje legyen FULL, mindig be legyen állítva az indexstatisztika készítése és frissítése, és még sorolhatnánk. Ahhoz, hogy ezeket az igényeinket kielégíthessük, saját magunknak kellett fejleszteni triggereket, jobokat stb.

Összefoglalva a jelenlegi verziókban a következő eszközöket használhatjuk erre a célra.

SQL Server Agent. Időzített jobokkal aszinkron módon riasztásokat válthatunk ki vagy reagálhatunk rájuk.

DDL triggerek/Event notification. Szinkron és aszinkron eseménykezelés az SQL Server 2005-től kezdődően.

SQL Server Best Practices Analyser. Külön letölthető ellenőrző eszköz több művelet, beállítás ellenőrzésére SQL 2000, 2005-ös verzióban.

SQL Server 2005 Surface Area Configuration Tool. Az SQL Server-példány egyes szolgáltatásainak beállítása/ellenőrzése.

Az SQL Server 2008-as verzióban a fent vázolt feladatok elvégzését segíti az az eszközrendszer, amelyet Declarative Management Frameworknek (DMF) hívunk. Mit takar ez a fogalom, hogyan használjuk, és miért is jó ez nekünk? Erre szeretnénk rávilágítani ebben a cikkben néhány példával fűszerezve. A fentebb felsorolt eszközök (job, trigger, profiler, database maintenance plan) természetesen továbbra is rendelkezésre állnak, a DMF kiegészíti, teljessé teszi a velük elvégezhető feladatokat, rendszert visz az adminisztrátori feladatokba.

Mi a DMF?

A Declarative Management Frameworkről a 2007. júliusi CTP-ben még Dynamic Management Framework fedőnéven lehetett olvasni, azóta ez persze a helyére került. Tulajdonképpen Policy Based Frameworknek is nevezhetnénk. Ugyanis az eddigi feladat (task) alapú adminisztrációt a házirend (Policy) alapú adminisztrációra cseréli.

Dióhéjban: az adminisztrátor az SQL Server Management Studióban (SSMS) létrehoz házirendeket, amelyek leírják, milyen feltételeknek, kondícióknak kell teljesülniük az adatbázisban (például minden tábla neve a 'tbl' karaktersorozattal kezdődik, mindig be van kapcsolva az indexstatisztika-frissítés stb). Ezután hozzárendeljük, mely szerverekre, adatbázisokra jusson érvényre a házirend, amelyet egy központi konzolról ellenőrizhetünk. Ezek a házirendek az MSDB adatbázisban tárolódnak, exportálhatók, importálhatók. Nagy hasznát vehetjük nemcsak több szerveres üzemeltetés esetén, hanem tesztszerver kialakításakor is. Tehát házirendeket definiálunk feladatok (task) helyett.

Ezzel a házirend fogalma utat tört magának az SQL Serverben is. Eddig csak az Active Directory, Ras-szerver környékén tüsténkedők találkozhattak vele. Most már az adatbázis-adminisztrátorok sem kerülhetik el. Persze fontos megjegyezni, hogy a DMF házirendjei nem keverendők össze az előbb említettekkel. Az SQL Serverben a házirend az MSDB adatbázisban található.

Az előjáték után nézzük meg a szereplőket, tehát az eposzi enumeráció következik:

Három komponens alkotja a keretrendszert:

Házirend-menedzsment. Házirend létrehozása az SSMS segítségével.

Explicit adminisztráció. Kiválasztjuk a házirend célját, a szenvedő alanyt, vagyis a „pácienst”.

Automatikus adminisztráció. Mikor értékelődjön ki az a házirend?

És még egy kis terminológia, lássuk a DMF-hez tartozó objektumokat:

Facet. Logikai jellemzők csoportja egy adott objektumtípusra (például: tábla, adatbázis stb.), beépített. Házirendek, kondíciók létrehozásánál használjuk.

Kondíciók (Condition). Logikai feltételek, amelyeket mi hozunk létre. A megkívánt (elérendő) állapotot írjuk le vele.

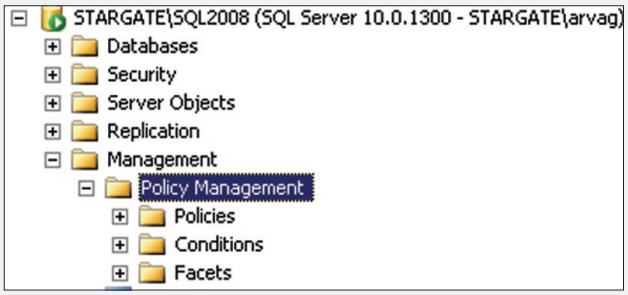
Házirend (Policy). Kondíciókat tartalmaz, és egy adott objektumra, szerverre, adatbázisra vonatkozik, meghatározza a végrehajtás módját.

Házirendcsoport (Policy Group). Házirendek csoportja. Egy házirend egy csoportnak lehet tagja. Könnyíti az adminisztrációt.

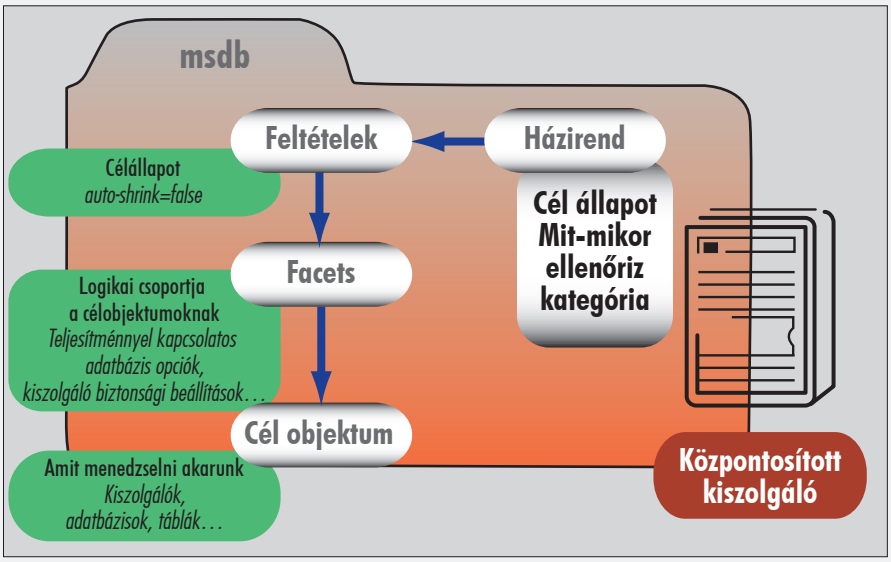
Kitaláltuk, hogy egy adatbázisban minden felhasználói táblának a ’tbl’ karaktersorozattal kell kezdődnie, és tetszőlegesen folytatódhat. Ha valaki nem ilyen kezdő karaktersorozattal hoz létre táblát, azt akadályozza is meg az SQL Server. Ezt kellene megvalósítani a DMF eszközeivel. DMF nélkül kellene írnunk egy DDL triggert, amely a create table eventre gerjed, és ha az első három bötű nem ’tbl’ vala, akkor rollback transaction.

Ez is megfelelő megoldás lenne, de minden esetben egy rövid kis kódot kellene írni hozzá. Hogyan lehet ezt „trendi” módon az SQL Server 2008 DMF segítségével kivitelezni?

Ehhez használjuk az SSMS-t mint eszközt.



2. ábra. Új objektumok a Management konténerben



1. ábra. A DMF architektúrája

Végrehajtási mód (Execution mode). Hogyan hajtódjon végre a házirend, azaz mikor, mi módon? A cikk későbbi részében részletesen kitérünk ezekre a lehetőségekre. Most kedvcsinálónak felsoroljuk a lehetőségeket:

- manuálisan futtatható;
- beállított időzítésnek megfelelően;
- csak logolja, mi történt a házirenddel kapcsolatban;
- végül, ha ellentétes a házirenddel, akkor akadályozza meg a változtatást.

Azoknak, akik képregényen nőttek fel, következik egy ábra *(1. ábra)*.

Ha a Management konténert kiböngésszük, akkor feltűnnek az új objektumok *(2. ábra)*.

A facets konténer tartalmazza objektumtípusonként a faceteket, amelyek az adott objektumtípus (tábla, schema stb..) ellenőrizhető, beállítható jellemzőit tartalmazza, csak ezekre vonatkozóan adhatunk meg feltételeket (Condition).

A Kondíciók konkrét logikai kifejezések, amelyek a facetekben található jellemzőkre vonatkoznak. Kötelező elemei a kondíciónak, ezáltal a házirendnek.

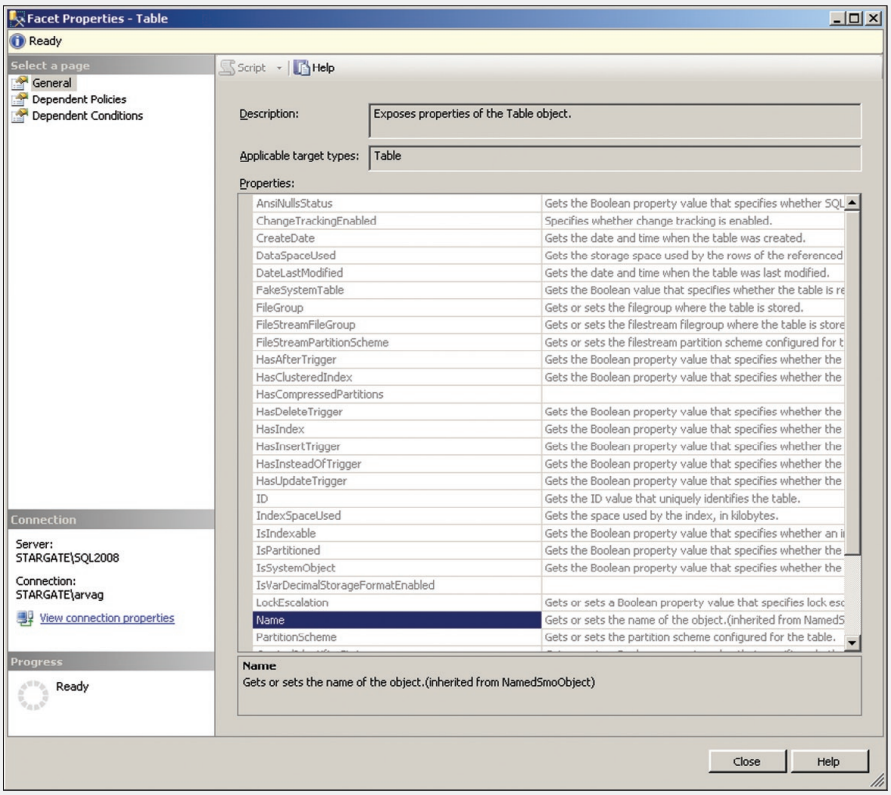
Például a Table facet jellemzői között látjuk a Name tulajdonságot. Ebben a nézetben csak szemlélődünk, mint a moziban. Ha be akarunk szállni a buliba, akkor a Kondíciók (Condition) konténerben egy határozott jobb klikk és új Kondíciókat választva *(4. ábra)* adhatunk nevet a kondíciónak. Kiválaszthatjuk a tábla objektumfeltétel-készletéből (table facet), hogy a név @name tulajdonságra akarunk feltételt adni, nevezetesen @name =’tbl%’, ezzel kész a feltétel. Már használtunk is két objektumot a DMF-ből. De ez a Kondíció még „csak lóg a levegőben”, kötni kellene még valahova.

```
Declare @condition_id int
EXEC msdb.dbo.sp_syspolicy_add_condition @name
=N'Tábla név',
@description=N'', @facet=N'Table',
 @expression=N'<Operator>
  <TypeClass>Bool</TypeClass>
  <OpType>LIKE</OpType>
  <Count>2</Count>
  <Attribute>
   <TypeClass>String</TypeClass>
   <Name>Name</Name>
  </Attribute>
  <Constant>
   <TypeClass>String</TypeClass>
   <ObjType>System.String</ObjType>
   <Value>tbl%</Value>
  </Constant>
</Operator>', @is_name_condition=2, @obj_name
=N'tbl%',
@condition_id=@condition_id OUTPUT
```

De előtte nézzük meg, hogyan néz ki ez a Kondíció „tudományosan” T-SQL ben, mert amit összekergetünk az egérrel, azt le is scriptelhetjük, és akkor bemutatókon el tudjuk ámítani a népet, hogy micsoda hard core fickók vagyunk *(3. ábra)*.

Ha megvan a Kondíció, akkor – ahogy korábban említettük – alkalmazni kellene. A feltétel, ugye, táblára vonatkozik, de nem mondtuk meg, melyik táblára, mikor értékelődjön ki, mi lesz, ha nem teljesül, és így tovább. Ehhez kell a házirend, ebben adjuk

meg, melyik kondíció milyen objektummal kerüljön kapcsolatba és hogyan, a hogyan lesz a végrehajtási mód (execution mode).



3. ábra. A kondíciónk beállításai grafikusan

A házirend létrehozásánál meg kell adnunk egy nevet a házirendnek, engedélyeznünk kell, és ami a legfontosabb, meg kell mondanunk, hogy melyik kondíció tartozik hozzá. Egyszerre csak egy kondíció tartozhat egy házirendhez. Ha többet szeretnénk, akkor jön jól a Házirendcsoport (Policy group). Meg kell adnunk, mely objektumokra jusson érvényre (jelen esetben minden táblára), és azt is meg kell mondanunk, melyik adatbázisban *(5. ábra)*. Persze az ábrán látható a csalafintaság, mert valójában több Kondíciót használtunk fel, hiszen a cél-objektumkör meghatározásához felhasználtunk egy szűrőfeltételt az adatbázisra vonatkozóan, ami természetesen újabb kondícióként jelenik meg. Tehát tovább pontosítva: egy kondícióban adhatjuk meg, mit kell ellenőrizni, betartatni, azt pedig, hogy hol, több kondícióban is leírhatjuk. Persze az egy kondíció nem olyan szigorú feltétel, mert mint ahogy azt a korábbi párbeszédpanelen láthatjuk (4. ábra), a kondíció létrehozásánál több jellemzőt különböző facetekből használhatunk fel, így hozva létre egy bonyolultabb kifejezést mint Kondíciót. Itt kell meghatározni a végrehajtási módot.

- **On demand.** Manuálisan futtatható az Evaluate opcióval (ez egyébként minden más opció választásakor is lehetséges).
- **On Schedule.** Időzítésnek megfelelően az SQL Agent Service ellenőrzi a feltételeket.
- **On Change – Log only.** Amikor a Kondíciókban leírtakkal ellentétes esemény történik, nem akadályozza meg, csak logolja az eseményt.
- **On Change – Prevent.** Amikor a Kondíciókban leírtakkal ellentétes esemény történik, nem engedi a változtatást, megakadályozza azt.

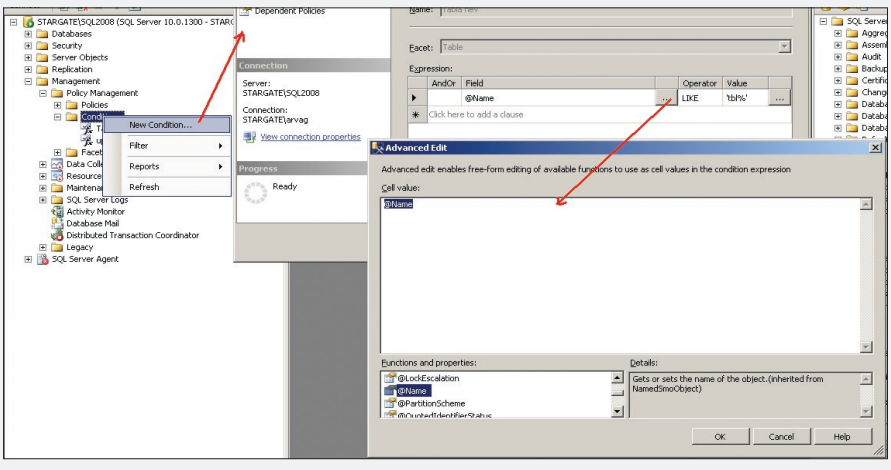
Ezenkívül szűrhetünk a szervertulajdonságra vonatkozóan is, a Server restriction mezőben megadhatunk egy kondíciót, ami korlátozhatja a futtatószerverek körét.

A description lapon opcionálisan megadhatunk egy kategóriát, (ha nincs, csinálhatunk egyet) ez segít kategorizálni a házirendeket, ha a házirendjeink száma lelkesedésünkből fakadóan a végtelenhez tart. A Description mezőben dokumentálhatunk, nem úgy, ahogyan az ábrán látszik. Ezenkívül megadhatunk egy üzenetet, ami a logban majd segít a tájékozódásban egy url kíséretében, amit persze tesztelhetünk is. (Erre az url-re kattinthat majd az a rendszergazda, aki szeretne bővebb infóhoz jutni, ez lehet egy oldal az intraneten, ami, mondjuk, a követendő adminisztrációs technikákat taglalja.)

Műveletek házirendekkel

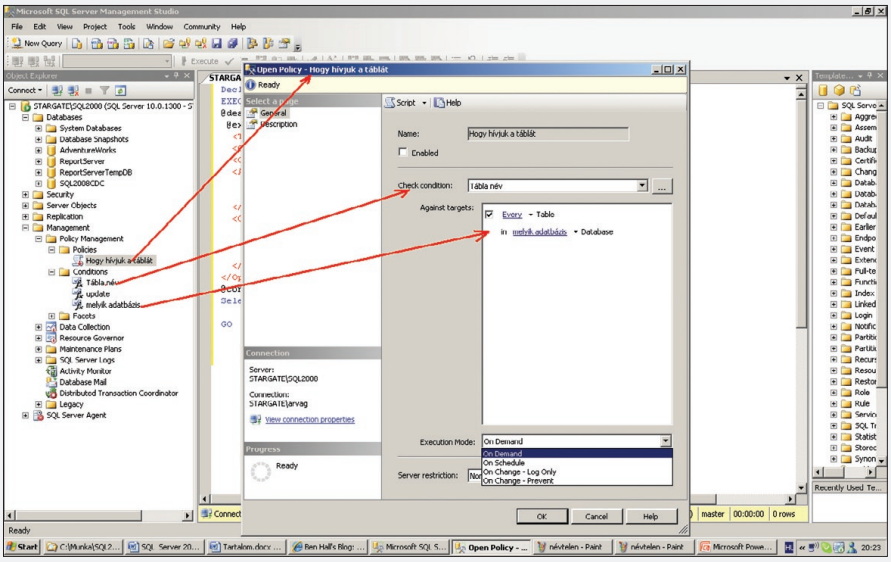
Ha a házirend végrehajtási módját on demand-ra állítottuk, akkor humán interface segítségével elindíthatjuk a kiértékelést (Evaluate), azaz rákattintunk. A fenti példa esetében valami hasonlót kapunk, mint ami a *7. ábrán* látható.

A komoly példaadatbázisban három egész



4. ábra. A kondíciónk beállításai a mélyben

darab táblánk van – így legalább áttekinthetjük –, ebből egy olyan user-tábla, ami a házirendünknek megfelel, és 'tbl ' betűsorozattal kezdődik a neve. A hibás soroknál megte-



5. ábra. Egy SQL házirend opciói

kinthetjük, hogy a kondíció melyik feltételével nem egyezik a tábla tulajdonsága.

Mi történik, ha olyan táblát akarunk létrehozni, ami távolról sem elégíti ki házirendben megtestesült finoman cizellált követelményrendszerünket? Semmi. A tábla gond nélkül, sikeresen létrejön. Ha legközelebb kiértékeljük a házirendet, akkor látjuk a tábla neve mellett, hogy ez bizony nem felel meg a házirend követelményeinek. Ezen kívül az SSMS-ből is tájékozódhatunk, mert megjelöli a problémás objektumokhoz vezető utat. Bejelöli a kritikus objektumokat, amelyeket az explorer detailsben is láthatunk a Policy Healt State oszlopban.

Ha azt látjuk, hogy ez így jó, akkor akár exportálhatjuk is a policyt xml-fájlba. Később ezt más szerveren importálhatjuk. Ezzel megvalósítható, hogy egy tesztrendszeren kikísérletezett házirendcsomagot az éles környezetbe vissza lehessen tölteni.

Időzítés

Fentebb utaltunk rá, hogy időzítést lehet hozzárendelni a házirend kiértékeléséhez. Készíthetünk szokásos időzítést, amit az SQL Agent fog lefuttatni (persze csak akkor, ha fut a szolgáltatás). Felhasználhatunk meglévő időzítéseket is, amelyek a Pick nyomógombra jelennek meg.

Ha ez megvan, akkor megnézhetjük, hogy létrejön egy új job, amelynek a neve a CHECK_házirendnév forma alapján képződik. Ha belekukkantunk, akkor egy lépése lesz: Evaluate Policy a típusa Power Shell (már ez is van az SQL Server 2008-ban) és meghívja a házirend kiértékelését.

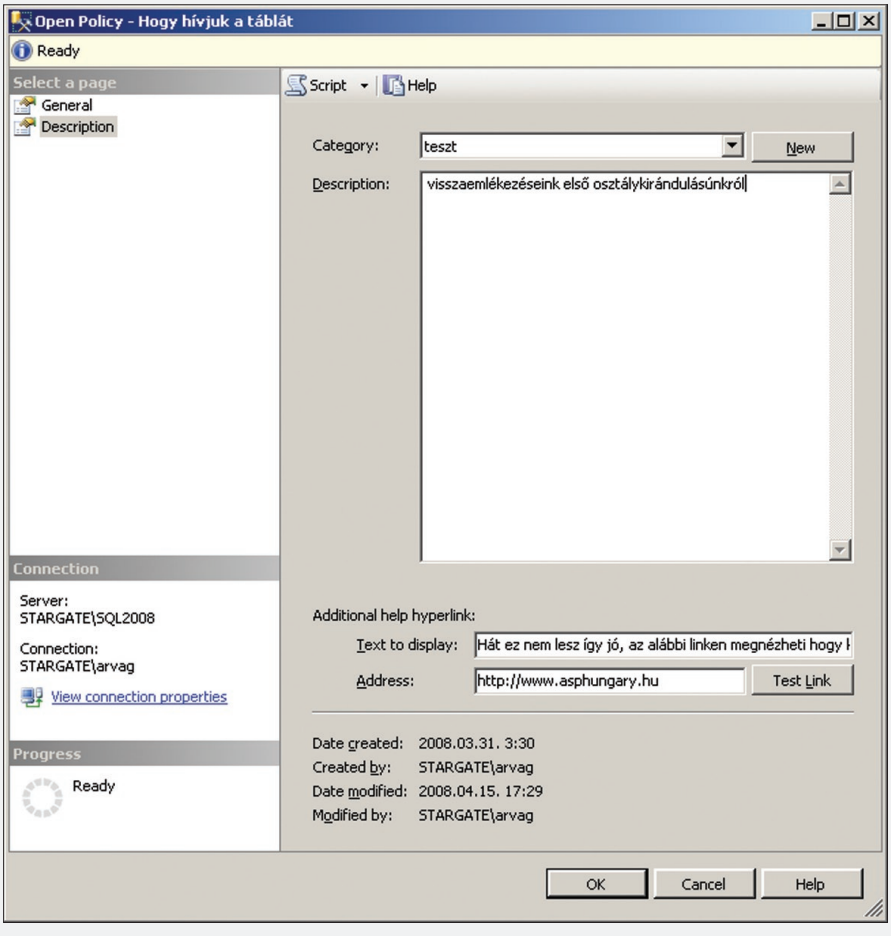
```
Evaluate-Policy -CheckSqlScriptAsProxy $true -ServerInstance
STARGATE\SQL2008 -Policy „Hogy hívjuk a táblát”
```

A job futásáról két forrásból is tudunk tájékozódni. Az egyik a „klasszikus” job history, ahol látjuk, hogy sikeresen lefutott-e. A másik a házirend-kiértékelés eredménye, a házirenden jobbklikk és history. Itt minden eseményt láthatunk a házirenddel kapcsolatban, akár kézzel értékeltettük ki, akár időzítve. Láthatjuk még az eredményt és benne azoknak az objektumoknak a listáját, amelyek nem feleltek meg a házirendben foglalt feltételeknek (Condition).

Ha ugyanannak a házirendnek a végrehajtási módját megváltoztatjuk On Change Log only módra, akkor a korábbi jobnak nyoma vész (törlődik).

Természetesen a Házirend Historyban látjuk a futási eredményeket. Ekkor már csak a logból olvashatjuk ki, ha valamely művelet nem felelt meg a házirendnek.

Ha On Change – Prevent a kiválasztott opció, akkor minden alkalommal, amikor olyan műveletet akarok végrehajtani, ami a házirendben foglaltakat nem elégíti ki, a művelet visszagördítődik. Az alábbi ábrán



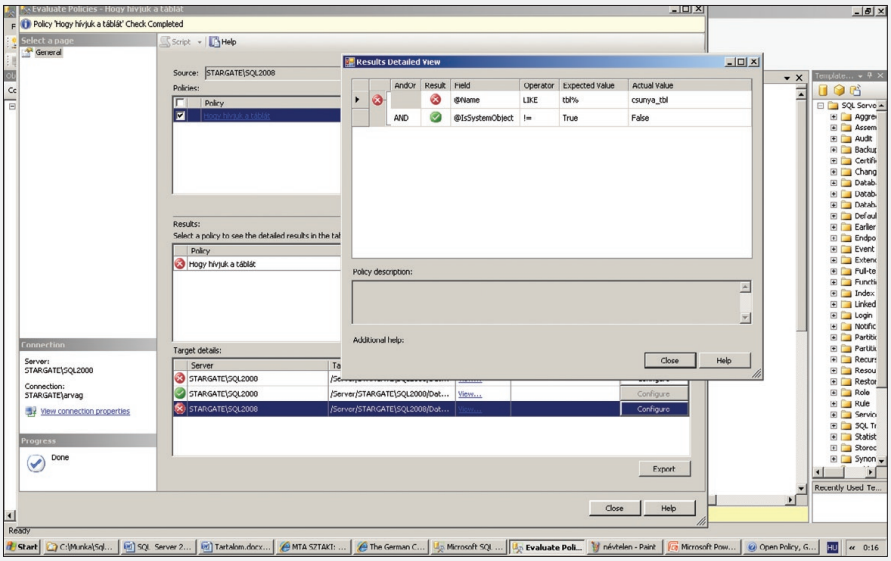
6. ábra. Segítség a házirendek kategorizálásában

szándékosan egy olyan táblát hoztam létre, aminek a neve nem tbl-lel kezdődik, ebben az esetben a mentés, azaz a CREATE TABLE utasítás hiúsult meg.

Melyik opció mikor jó?

Néhány gondolatébresztő tipp, bár ezek után sokakban számtalan ötlet fogalmazódhat meg. Talán azt lehetne mondani, hogy a kézi futtatás a tesztelés alatt álló házirendeknél hasznos, illetve előnye, hogy a log-nézetben akkor lehetünk könyörtelenek, és jöhet az On Change – Prevent opció.

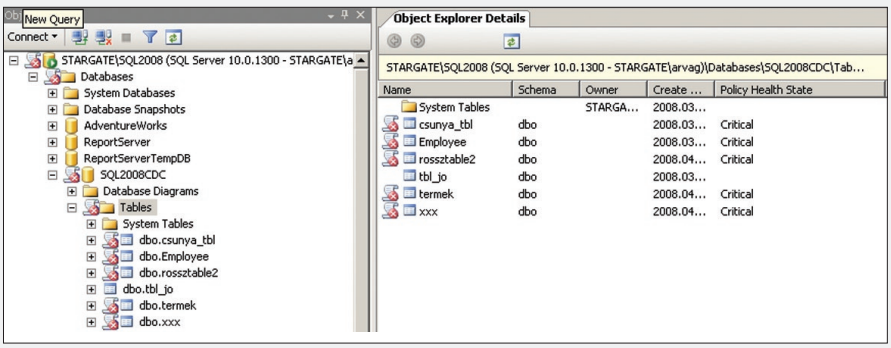
Egy nagyon egyszerű példán néztük meg a főbb elemeket. Rövid időn belül készíthetünk olyan házirendeket, amelyek a kívánt beállításokat ellenőrzik, a nem kívántakat pedig akár online megakadályozzák. Fejlesztők akár ellenőrizhetik, hogy minden táblán van-e elsődleges index, van-e a táblában olyan oszlop, amelyben azt rögzítjük, ki, mikor módosította a táblát, van-e neki



7. ábra. Egy házirend kiértékelésének eredménye

egy csomópont alatt áttekinthetően láthatjuk az összes objektumot, amelyek megfelelnek, illetve azokat, amelyek nem felelnek meg a kritériumoknak.

megfelelő default értéke. Adminisztrátorok ellenőrizhetik a már korábban említett indexstatisztikákra vonatkozó beállításokat, megakadályozhatják, hogy ezeket valaki meg-



8. ábra. Szépen látszik végig a fában, hogy hol van probléma

Az On Change Log only esetén időrendben láthatjuk, mi történt. Ez jó, ha a legutolsó hibákat akarjuk látni, illetve ha nem akarjuk megakadályozni a nem kívánt műveletet.

Ha szigorúak vagyunk, és már teszteltük a beállításokat egy éles, üzemelő rendszernél, változtassa a recovery-modellel együtt. Ha végiggörgetjük a facetek jellemzőit, akkor rengeteg ötletet összeszedhetünk arra, hogyan lehet hatékonyabban adminisztrálni az SQL Server 2008-at.

Ahogy a bevezetőben említettük, persze ezek jó részét eddig is megtehettük volna, de csak programozással.

Többszerveres adminisztráció

Fentebb utaltunk rá, hogy exportálni tudjuk a házirendeket xml-fájlba, és ezeket importálni is tudjuk más szerverre vagy szervercsoportba. A regisztrált szerverek alatt található egy Configuration Servers mappa, ez alá regisztrálhatunk egy szervert, a szerver alá több szervercsoportot, a szervercsoportok alá pedig egyéb szervereket. Ezzel csoportosíthatjuk az adminisztrálandó szervereket. Minden egyes szervernél lehetőség van házirendimportra, -kiértékelésre, de ezen kívül a Konfigurációs szerveren is megtehetjük ezt.

Mi van mögötte?

A DMF mögé DDL trigger van bújtatva, amely szerver-, illetve adatbázisszintű eseményekre figyel (hogy melyekre, az az általunk beállított házirendektől függ). Ha nincs beállítva házirend On Change kezdetű opcióval, akkor nem jön létre DDL trigger. Ha készítünk egy házirendet (policy), akkor a házirendhez tartozó kondícióból kiderül, melyik facetet használtuk fel. A facet tulajdonságából pedig kiderül, hogy melyik eseményre (eventre) kell figyelnie a DDL triggernek.

A DMF motor az SQLCLR-ben fut, akkor is, ha az adott példányban nincs engedélyezve az SQLCLR. Az SQLCLR-nek ugyanis két üzemmódja van: On és Off. Off módban a Microsoft által aláírt és feltelepített Assemblyk futhatnak, tehát működik a kedvenc DMF-ünk.

Kicsit részletesebben (lábvíz, mélyvíz helyett). A cikken végigvonuló nem túl bonyolult példa esetében, ha egy házirend létrehozásakor On Change Log only vagy On Change – Prevent opciót választottuk, akkor létrejön a Server objects\Triggers konténerben egy syspolicy_server_trigger.

Ha belepillantunk, akkor látjuk, hogy egy speciális felhasználó nevében fut, ez a '##MS_PolicyEventProcessingLogin##'. A Master és az MSDB adatbázisban található meg adatbázis-felhasználóként. Ez utóbbi az érdekes, itt tagja a PolicyAdministratorRole szerepkörnek – talán nem véletlenül. Ennek a szerepkörnek execute joga van a házirendeket kezelő sp_syspolicy_*** tárolt eljárásokon. Ezenkívül select joga van a syspolicy**** kezdetű nézeteken.

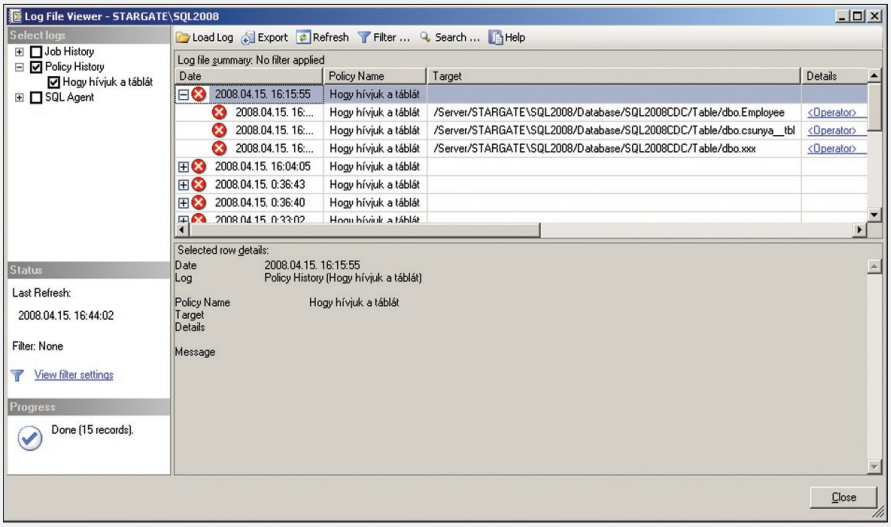
```
CREATE TRIGGER [syspolicy_server_trigger] ON ALL SERVER
WITH EXECUTE AS '##MS_PolicyEventProcessingLogin##'
FOR ALTER_TABLE,CREATE_TABLE,RENAME
AS
BEGIN
    DECLARE @event_data xml
    SELECT @event_data = EVENTDATA()
    EXEC [msdb].[dbo].[sp_syspolicy_dispatch_event]
@event_data = @event_data, @synchronous = 1
END
```

Ezeket a nézeteket használhatjuk arra is, hogy a meglévő házirendeket és kondíciókat scriptből lekérdezhessük:

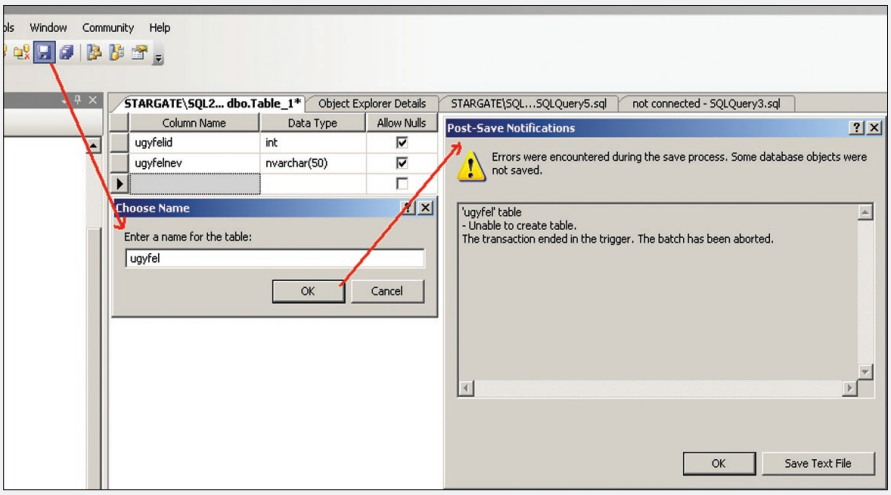
```
select * from syspolicy_policies
select * from syspolicy_conditions
```

De nézzük tovább a triggert! Látható, hogy egy jól nevelt DDL triggernek megfelelően ALTER_TABLE, CRAETE_TABLE, RENAME eseményekre gerjed. Azért csak ezekre, mert csak egy működő (engedélyezett) házirendünk van, és annak a kiértékeléséhez ezek az események elegendőek.

Aki már látott DDL triggert, annak ismerős lehet, hogy az EVENTDATA() függvény adja vissza xml-ben az esemény jellemzőit (milyen utasítás, mikor, process id stb.). Ezt átadja a sp_syspolicy_dispatch_event tárolt eljárásnak, azzal együtt, hogy ez szinkron vala. Az eljárást az MSDB adatbázisban találjuk. Most humanitárius okokból nem másolnám ide, bár kétségtelenül telne vele az oldal, és növelné a cikk tudományosságát. Ehelyett olcsó népszerűségből fakadóan csak felvázolom, hogy mit csinál; kibogarássza az xml-adatból az esemény típusát, az adatbázis nevét, az objektum nevét, típusát a különböző fent említett nézetekkel, táblákkal összefűzve beszúrja az msdb.dbo.syspolicy_execution_internal táblába. Naná, hogy ezen is van egy trigger, ami syspolicy_execution_trigger néven fut. Ez szépen előszedi a frissen beszúrt rekordokat az előző táblából egy jó kis cursorba, és átadja a sp_syspolicy_execute_policy tárolt eljárásnak. Persze, ha hiba van, akkor Raiserror. Ez az eljárás meghívja a sys.sp_execute_policy eljárást, ez ellenőrzi a házirend feltételeit, és ha 1-et ad vissza, akkor Rollback. Na végre! Már úgyis elvesztettem a fonalat.

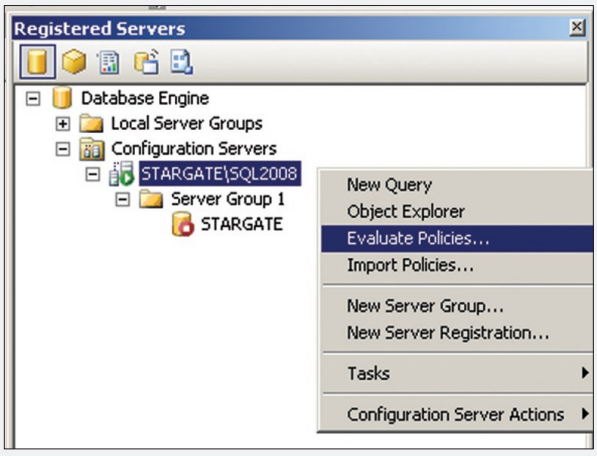


9. ábra. Részletes napló a megfelelőségről



10. ábra. A házirendnek nem megfelelő utasítás nem hajtódott végre

Tehát DDL Trigger -> sp_syspolicy_dispatch_event tárolt eljárás -> syspolicy_execution_internal tábla -> syspolicy_execution_trigger -> sp_syspolicy_execute_policy tárolt eljárás -> sys.sp_execute_policy.



11. ábra. A házirend a Konfigurációs szerveren is ellenőrizhető

Még egyszerűbben: a DDL jellegű triggerek karon fogva néhány tárolt eljárással az MSDB adatbázisból dolgozzák fel az eseményeket, felhasználva az MSDB adatbázisban tárolt házirend-definíciókat.

Van kedvünk ezeket inkább kézzel megírni? Ugye, nincs, mert hát lustaság a fejlődés motorja... Tehát megint nem vajákosságról van szó a háttérben, hanem a meglévő eszközök jól átgondolt hatékony alkalmazásáról. Az eredmény pedig? Egy hatékony adminisztrációs keretrendszer.

Árva Gábor
(Arva.Gabor@asphungary.hu)
MCSE, MCT, MCTS, Számalk

TÚL A RELÁCIÓS VILÁGON

Fejlesztői újdonságok és új adattípusok.

Az SQL Server 2008 folytatja az építkezést azon az alapon, amit az SQL Server 2005-ös verzió vezetett be, új CLR- és natív típusokkal, valamint új SQL-parancsokkal. Ezeket tekintjük át a cikkünkben.

Dátumtípusok

Mi a probléma a meglévő DATETIME és SMALLDATETIME típusokkal?

1. Kicsi az értékkészletük, 1753 előtt is volt már világ.
2. Kicsi a pontosságuk, a pontosabbnak, a DATETIME-nak is 3 ms a felbontása.
3. Nem kezelnek időzónát.
4. Nincs külön csak dátum- és csak időtároló típus. Eleve, sokszor csak az egyik kell, például napra kerekített dátumtárolás, és ilyenkor nemcsak könnyebb kezelni a külön tárolt darabokat, de hely sem kell neki annyi.

Mit kapunk hát az SQL Server 2008-ban?

1. DATE típus. 0001-01-01 és 9999-12-31 között működik, és csak 3 bájtot foglal el. Értelemszerűen nap a felbontása.

2. TIME típus. Maximum 100 ns felbontású, lehet szabályozni, mennyire legyen pontos. TIME(7) például 100 ns-os, és 5 bájtot igényel. TIME(0) csak 3 bájt, cserébe csak századmásodpercig pontos. Vagy a TIME(4) 4 bájt, és 3-4 digitig pontos (kb. ms-os felbontás).

3. DATETIME2. Az előző kettő hibridje, 6-8 bájt kell neki, nyilván az időtag pontosságától függően. Ez lehet egy jó DATETIME-alternatíva, ha nagyobb pontosságra és hosszabb időszakok kezelésére van szükségünk.

4. DATETIMEOFFSET típus. A DATETIME2 időzónával kiegészített változata. Szövegként így szoktuk leírni: „2007-05-08 12:35:29.1234567+12:15”, azaz a jelzett időponthoz képest plusz 12 óra 15 perc az időeltolódás.

Hogyan látszanak ezek a típusok ADO.NET-ből? Az SqlDbType-ba belekerült négy új érték:

```
SqlDbType.Date
SqlDbType.Time
SqlDbType.DateTime2
SqlDbType.DateTimeOffSet
```

Az SQL DATE és DATETIME2 a CLR megszokott DateTime típusára képződik le. A TIME a TimeSpanre, a DATETIMEOFFSET pedig egy új CLR-típusra a System.DateTimeOffsetté alakul át.

HierarchyID adattípus

Ő egy olyan típus, amely egy hierarchia, azaz egy fa egy adott pontját tudja megcímezni. Hogyan lehet relációs adatbázisban fát építeni? Például rekurzív, önhivatkozó táblával, mint a Northwind adatbázis Employees táblája, vagy az AdventureWorks adatbázis HumanResources. Employee táblája. Ez utóbbiban a ManagerID oszlop mutat a főnök EmployeeID-jára.

Az így felépített fa tetszőleges eleme jellemezhető egy úgynevezett OrdPath-szal. Ebben a gyermekelemeknek sorrendjük van, mint például az xml infosetben, így a gyerekek megcímezhetők a szülők alatti sorszámukkal. 1/2/4 például a gyökér-node 2. gyermekének a 4. gyerekét jelenti.

A HierarchyID egy olyan CLR-típus, amely egy OrdPath-t képes tárolni. Segítségével igen kompakt módon lehet tárolni egy hierarchia-node helyét egy fában. Normál esetben például rekurzív CTE-vel járhatunk be egy hierarchiát, hogy meghatározzuk az elérési útját egy node-nak.

Ez elég lassú persze, minden szinthez kell egy JOIN. Egy táblában HierarchyID oszlop segítségével minden egyes, a fa egy node-ját reprezentáló sorhoz tárolhatjuk a sornak mint fa-node-nak a hierarchiában elfoglalt helyét, így rekurzió nélkül is azonnal látható, hol foglal helyet a hierarchiában az adott sor (mint node).

A HierarchyID felfogható egyfajta denormalizálási technikának is, hisz a hierarchia tárolható a már említett relációs módon is. Akár egyszerre is lehet használni a kettőt, de külön-külön is. Vannak esetek, amikor az egyik hatékonyabb, van, amikor a másik.

Mire jó a HierarchyID? Vannak műveletek, amelyeket gyorsabban lehet végrehajtani a segítségével, mivel a node-ok elérési útja van enkódolva az idben, így a felindexelt id alapján egyes lekérdezések hatékonyak lehetnek.

Kiinduló adatként nézzük az alábbi táblát, amely az AdventureWorks adatbázis HumanResources.Employee táblájának ada-

taiból készült (a relációs adatok hierarchikussá alakításához lásd [1]).

```
SELECT OrgNode.ToString() AS LogicalNode, *
FROM HumanResources.NewOrg ORDER BY LogicalNode;
```

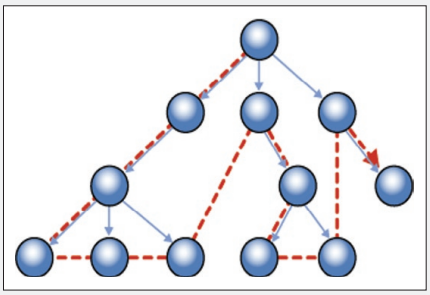
LogicalNode	OrgNode	EmpId	LoginID	ManagerID	Title
/	0x	109	ken0	NULL	Chief Executive Officer
/1/	0x58	6	david0	109	Marketing Manager
/1/1/	0x5AC0	2	kevin0	6	Marketing Assistant
...					
/1/7/	0x5CE0	271	john5	6	Marketing Specialist
/1/8/	0x5D10	272	mary2	6	Marketing Assistant
/2/	0x68	12	terri0	109	Vice President of Engineering
/2/1/	0x6AC0	3	roberto0	12	Engineering Manager
/2/1/1/	0x6AD6	4	rob0	3	Senior Tool Designer
/2/1/2/	0x6ADA	9	gail0	3	Design Engineer
/2/1/3/	0x6ADE	11	jossef0	3	Design Engineer
/2/1/4/	0x6AE1	158	dylan0	3	Research and Development Manager
/2/1/4/1/	0x6AE158	79	diane1	158	Research and Development Engineer
/2/1/4/2/	0x6AE168	114	gigi0	158	Research and Development Engineer

Nézzük meg például, hogyan keresnénk meg egy adott ember összes direkt vagy indirekt beosztottját? Azaz, az adott node alatti részfát szeretnénk kiválasztani.

```
declare @manager HierarchyID = (select OrgNode
from HumanResources.NewOrg
where LoginID = 'terri0')

select
cast(OrgNode as varchar(50)) as OrdPath,
EmployeeID, LoginID, ManagerID, Title
from HumanResources.NewOrg
where @manager.IsDescendant(OrgNode) = 1
order by OrdPath
```

Kikeressük terri0 HierarchyID-ját, majd az IsDescendant metódus segítségével leszűrjük az utódait. Gyerekek, unokák stb. A függvény magát a kiinduló node-ot is visszaadja, azaz a DescendantOrSelf talán precízebb név lenne.



1. ábra. A HieararchyId-n létrehozott index mélységi bejárás alapján rendezi a hierarchikus adatokat

A lekérdezéskimenet a fenti táblázat /2/1-gyel kezdődő soraiból áll.

Az IsDescedantra fel van készítve az optimalizáló, így ha HiearchyId-s oszlopon indexet hozunk létre, akkor a lekérdezést a leghatékonyabb módon, index seek-kel hajtja végre. Azaz részfa kikeresése esetén a HiearchyId sokszorosával gyorsabban szolgáltat eredményeket, mint a hagyományos, rekurzív JOIN-os eljárás.

Ez azonban nem mindig igaz, ha például csak a közvetlen beosztottakat szeretném lekérdezni, erre használhatom a HiearchyId-t és GetAncestort metódusát. De akkor már lassabb lesz a lekérdezés, mint az egyszerű JOIN-os relációmegoldás (mivel a részfából ki kell szűrnie a szervernek a nem direkt gyerekeket).

```
select
cast(OrgNode as varchar(50)) as OrdPath,
EmployeeID, LoginID, ManagerID, Title
from HumanResources.NewOrg
where OrgNode.GetAncestor(1) = @manager
```

De ezen is lehet azért javítani. A HiearchyId mélységi bejárás alapján rendezi az adatokat *(1. ábra)*, ezért jó részfa szűrésre.

Ami nekünk a direkt gyerekek hatékony szűréséhez kellene, az a szélességi (szintenkénti) bejárás alapján rendezett index *(2. ábra)*.

Ilyen index számított, indexelt oszloppal képezhető. Ehhez fel kell vennünk a táblába egy új, számított oszlopot, ami a node-ok szintjét számolja ki (a GetLevel metódus segítségével):

```
alter table HumanResources.NewOrg
add OrgLevel as OrgNode.GetLevel()
```

Ez így néz ki:

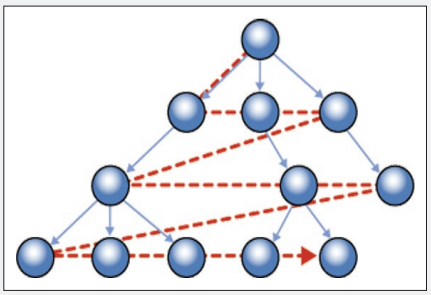
Path	OrgLevel	EmployeeId	ManagerID
/	0	109	NULL
/1/	1	6	109
/1/1/	2	2	6
/1/2/	2	46	6
...			
/1/8/	2	272	6
/2/	1	12	109
/2/1/	2	3	12
/2/1/1/	3	4	3
/2/1/2/	3	9	3
...			
/2/1/4/1/	4	79	158

És most jön az index a számított oszlopra:

```
create nonclustered index IDX_Org_Breadth_First
on HumanResources.NewOrg(OrgLevel, OrgNode)
include (EmployeeID, LoginID, ManagerID, Title);
```

Ettől a GetAncestoros lekérdezés index seekre vált, azaz nagyon hatékony lesz, az új indexnek köszönhetően.

A hierarchia módosításakor (példák: [2]) mindig frissíteni kell az érintett HierarchyID adatokat is, ez jelentős költséggel járhat akkor, ha a fa teteje felé kell egy node-ot új szülő alá helyezni, azaz például egy beosztott, akinek emellett sok beosztottja is van, új főnököt kap. Ekkor nemcsak a beosztott sorában kell a HierarchyID értékét módosítani, hanem az összes közvetlen és közvetett beosztottnál is.



2. ábra. Szélességi (szintenkénti) bejárás alapján rendezett index

Az inkonzisztens HierarchyID-jét elkerülendő érdemes integritásvédelmet berakni a HierarchyID-k kezelésébe, amely azt ellenőrzi, hogy mindenkinek van-e szülője. Ezt elég egyszerű megoldani:

Minden sorhoz képezzük a szülőt a GetAncestor(1) segítségével, majd egy foreign key contrainttel betartatjuk, hogy legyen ilyen szülő a primary key-k, azaz az HieararchyId-k között (feltételezzük a HierarchyID oszlopon van a primary key constraint).

```
alter table HumanResources.NewOrg
add ParentId AS OrgNode.GetAncestor(1) persisted
constraint FK_Parent
references HumanResources.NewOrg(OrgNode)
```

Térbeli adattípusok

Két térbeli (spatial) típust tartalmaz az SQL Server 2008: geometry és geography. A geometry hagyományos, euklidészi, derékszögű, sík koordinátarendszerben dolgozik, míg a geography elliptikus, a Földön elhelyezkedő, földrajzi koordinátákat modellező típus (szélesség, hosszúság stb.).

Koordinátákkal dolgozó programok natívan tudják tárolni az adataikat, és rengeteg műveletet (átfedik-e egymást alakzatok,

milyen közel vannak, stb.) értelmezhetnek rajtuk.

A típusok különböző szabványokra épülnek, ezek közül a Well Known Text formátum az, amellyel szövegként írhatunk le alakzatokat:

```
POINT(6 10)
LINESTRING(3 4,10 50,20 25)
POLYGON((1 1,5 1,5 5,1 5,1 1),(2 2, 3 2, 3 3, 2 3,2 2))
MULTIPOINT(3.5 5.6,4.8 10.5)
MULTILINESTRING((3 4,10 50,20 25),
        (-5 -8,-10 -8,-15 -4))
MULTIPOLYGON(((1 1,5 1,5 5,1 5,1 1),
        (2 2, 3 2, 3 3, 2 3,2 2)),((3 3,6 2,6 4,3 3)))
GEOMETRYCOLLECTION(POINT(4 6),
        LINESTRING(4 6,7 10))
```

Nézzünk egy-két példát a geometry típussal. Hozzunk létre egy pontot reprezentáló változót:

```
declare @g geometry;
set @g = geometry::STGeomFromText('POINT (3 4)',
0);
select @g.ToString()

POINT (3 4)
```

Az STGeomFromText a már hivatkozott WKT szövegből elemzi az alakzatot. Használhattuk volna a specializáltabb STPointFromText metódust is.

Rakjuk össze a pontokat egy halmazba:

```
(STUnion):declare @a geometry =
        geometry::STGeomFromText('POINT(0 0)', 0);
declare @b geometry =
        geometry::STGeomFromText('POINT(4 4)', 0);

select @a.STUnion(@b).ToString();

MULTIPOINT ((4 4), (0 0))
```

Készítsünk belőle vonalat:

```
select @a.STUnion(@b).STConvexHull().ToString();

LINESTRING (4 4, 0 0)
```

Nagyon sok műveletet implementálnak ezek a geometriai típusok. Természetesen az igazi felhasználása során általában térképészeti adatokat tartalmazó táblákon végzünk geometriai műveleteket. Például, ha kíváncsiak vagyunk arra, hogy mely városok vannak Magyarországon:

```
select CityName from City
where geom.STIntersects(
(select geom from Country
where CountryName = N'Hungary')) = 1
```

A Country táblából kiválasztjuk Magyarország körvonalát, és a városok közül azokat válogatjuk ki, amelyek metszik az országot. Nyilván persze ezt egyszerűen elővehetnénk egy táblázatból is, nem kellenek hozzá geometriai metódusok. De ha egy tetszőleges területet választunk ki, akkor már értelmes lehet a kérdés.

Mely városok vannak a Duna 10 kilométeres körzetében?

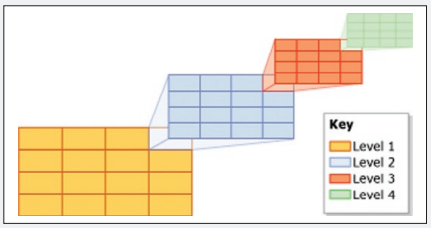
```
declare @danube geography
select @danube = select geom from River where
        NAME = N'Danube'
select CityName,
geom.ToString() 'A város koordinátái',
geom.STDistance(@danube) 'Távolság a folyótól'
from City
where geom.STDistance(@danube) < 10000
```

Az STDistance a minimális távolságot adja meg egy pont és egy alakzat között, azaz leszűrjük azokat a városokat, amelyek a Dunától mint sokszögtől való távolsága kevesebb, mint 10 000 egység. Esetünkben az egység méter, ami a geom nevű, geography típusú oszlop betöltéskor beállított vonatkoztatási rendszere miatt van.

A geometriai típusok CLR-típusként vannak implementálva, amit akár mi is megírhattunk volna. De itt nem állt meg a Microsoft. Sok geometriai adat kezelésekor ugyanis felmerül az igény, hogy indexekkel szeretnénk meggyorsítani a lekérdezéseket, ugyanakkor ezek az adatok messze nem olyan egyszerűen indexelhetők, mint mondjuk egy varchar vagy egy int oszlop.

A geometriai adatokat indexelő spatial index éppúgy B* fa, mint a közönséges indexeknél, csak kérdés, hogyan tudja a szerver az alakzatok adatait úgy átalakítani valamilyen bináris adathalmazzá, ami aztán meggyorsíthat bizonyos műveleteket, például távolságszámítást vagy metszet meghatározását?

A következő történik az index létrehozásakor. A síkot vagy féltekét felbontják cellákra, mint amikor „kockás” papírra rajzolunk. Az alakzatok ezekre a négyzetekre vannak feketve. Minden cellát további cellákra bonthatunk, és ott is meghatározhatjuk, mely alcellákban van még benne egy alakzat, és melyekben nincs. Ezt a dekompozíciót max. 4 szinten folytatják, így egyre finomabb felbontásban gyűjtenek információt az alakzatról *(3. ábra)*. Hogy ne szabaduljanak el az adatok, vannak olyan szabályok, amelyek korlátozzák az adatmennyiségeket.



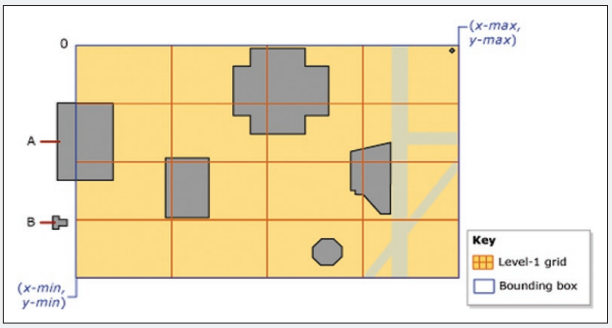
3. ábra. A spatial index egyre finomabb cellákra bontja fel a síkot

Érezhető: be kell határolni a „kockás papír” méretét, hogy véges méretű legyen az index *(4. ábra)*, erre az index létrehozásakor van lehetőség. Ez azonban csak a sík adatokkal dolgozó geometry típusra vonatkozik, a geographynál véges a területünk, hisz a Föld felszínéről van szó.

A geography típus ellipszoid adatainál még bonyolultabb a helyzet, azokat levetítik síkba, és úgy dolgoznak vele *(5. ábra)*.

A spatial index ezek után nagyon tömören az alakzatok által elfoglalt rácspontokat jegyzi le.

Mire jó ezek után egy spatial index? Megfelelő körülmények között az STIntersects(), STEquals(), és STDistance() metódusokat meg tudja támogatni egy spatial index.



4. ábra. Az alakzatokat ráillesztik az előző lépésben lefektetett rácsra

A teszteléshez a korábbi, Dunához közeli településeket kiválasztó példánkat futtattam, de sokkal több adatra, egy, az Amerikai Egyesült Államok városait tartalmazó táblán. Végrehajtási terve így néz ki spatial index nélkül *(6. ábra)*.

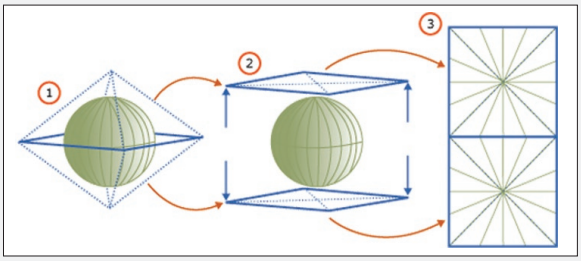
Látható, hogy a Filter operátor viszi el a költések zömét, ebben az alaptábla minden egyes sorára végrehajtják a szűrést. Ennek a Filternek kellene kiesnie vagy legalábbis a költségének csökkennie az index hatására. Index nélkül a végrehajtás átlagban 2800 ms-ot vett igénybe, ebből kb. 2000 ms a CPU-költség, azaz igen erősen processzorintenzív a szűrés. A lapolvasások száma 1850.

Alapbeállításokkal hozzunk létre egy spatial indexet a geography oszlopunkon:

```
create spatial index idx_USCity_Spatial_1 on USCity
(geom)
```

A végrehajtási idő leesik 160 ms-ra! A lapolvasások száma 1720 lett, azaz ebben nem mutatkozott jelentős különbség, de a CPU-idő leesett kb. 50 ms-ra, 2000-ről. Azért ez igen nagy nyereség.

A végrehajtási terv *(7. ábra)*. A jobb alsó Clustered Index Seek működik az spatial indexre építve. Azonban a spatial index csak közelítő index, hisz véges felbontású a rácsozat, így nem tud pontos eredményeket adni. Ezért látható a bal alsó részben egy Filter operátor, amely az index által kiszűrt durva adathalmazt véglegesíti. Az index által leválogatott adatok között ott van minden, a feltételre illeszkedő város, csak lehetnek benne olyan, a feltételhez közeli városok is, amelyekre nem teljesül a feltétel. Ezeket dobja ki a Filter. Példánkban a bal oldali Merge Joinból 20 sor potyog ki, valójában ennyit adott vissza az index, mivel durva a felbontása. A Filter és Nested Loop Join után már csak 10 sor maradt, ami már a helyes kimenet. Azaz igaz, hogy az index durva felbontása miatt jöttek be felesleges sorok, de nem baj, 20 sorból mégis csak könnyebb kiválasztani a jókat a lassú függvény (STDistance) tényleges végrehajtásával, mint az eredeti 34 000-ből.



5. ábra. A geography típus Föld féltekéit reprezentáló koordinátáit síkba vetítik le

Mivel az index felbontását, azaz, hogy az adott térrészt mekkora részekre bontjuk, szabályozhatjuk, így egyensúlyozhatunk a processzorterhelés és az IO-költség között. Hisz ha az index nagyfelbontású, akkor nagyobb lesz az IO-költség, mert nagyobb lesz az indexfa, de kevesebb fals sor lesz, így kevesebb processzorköltsége lesz a kevesebb soron végrehajtott tényleges spatial művelet (például STDistance) végrehajtásának.

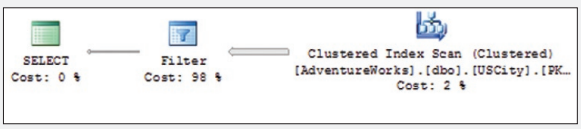
Ezt tesztelendő hozzuk létre a maximális felbontású indexet!

```
create spatial index idx_USCity_Spatial_2
on USCity(geom) using geography_grid
with (
   grids = (high, high, high, high)
);
```

A korábbi indexünk alapértelmezett medium felbontással jött létre minden szinten, ami azt jelenti, hogy minden szint 8×8-as rácsra bontja az előzőt. Ez az új indexünk a high miatt 16×16-os ráccsal dolgozik.

Ha az előző, medium index is rajta van a táblán, akkor a szerver nem használja ezt azt új indexet, mert nem éri meg neki. Ledobva az előzőt vagy index-hinttel ráerőszakolva erre, a végrehajtási idő 160 ms körül alakul, ami kb. azonos az előző indexelt megoldással. Azonban a lapolvasások száma az ottani 1700-ról felugrik 4800-ra, ami nem meglepő, hisz jóval több adatot tárol az indextábla. A végrehajtási tervben az index révén 12 sor jön vissza, azaz csak 2 potyasor jön ki az indexből, köszönhető a finom felbontású rácsnak.

Ellenpróbaként csináltam egy low, low, low, low durva felbontású indexet is, ebben szintenként 4×4-esek a rácsok. Ekkor az index 148 sort válogat le, ebből szűrik le a tényleges 10-et. Ennek végrehajtási teljesítménymutatói nagyon hasonlóak az első indexéhez.



6. ábra. Szűrés távolságra spatial index nélkül

Valószínűleg, ha sűrű elhelyezkedésű adatokról lenne szó (például nem városok, hanem városon belüli házak), akkor már inkább egy finomabb index érné meg, a durva miatt túl sok adatot kellene lassú módon szűrnie a szervernek.

Összegezve látható: szépen lehet játszani, hogy az adatok eloszlásának megfelelően milyen felbontású indexet hozunk létre a spatial adatainknak, így játszhatunk az IO- és a CPU-költség között valamiféle optimumra. Szép optimalizálási munka lehet ez.

A témában részletesebb példák a [3] címen találhatók.

Streaming-adatok tárolása

Nagytömegű adatokat, például filmeket, nagy dokumentumokat, képeket vagy egyéb nagytömegű, strukturálatlan adatokat tárolni akaró fejlesztők számára hasznos lehet ez az új szolgáltatás. A VARBINARY(MAX) oszlopokban, ha megjelöljük őket a FILESTREAM attribútummal, akkor az adatok NEM az adatbázisban fognak tárolódni, hanem a fájlrendszerben, sima fájlokként. Így még a 2 gigabájtos korlát is megszűnik! Nem kell most már azon se töprengni, hogy a nagy adataink adatbázisban legyenek-e vagy fájlrendszerben, aggódva a tranzakcionális konzisztencia miatt.

Végül is mikor érdemes használni a filestream store-t?

1. Ha a tárolandó objektumok átlagmérete 1 megabájt felett van.

2. Ha fontos, hogy nagyon gyorsan tudjuk kiolvasni őket.

3. Ha az alkalmazás rendelkezik középső réteggel (az adatok eléréséhez ugyanis nem elég a TSQL).

Kisebb adatokhoz a sima varbinary(max) oszlop jobb választás lehet.

A streaming-szolgáltatás használatához engedélyezni kell azt a szerverpéldányra:

```
EXEC sp_filestream_configure @enable_level = 3;
```

Az @enable_level azt szabályozza, hogy csak TSQL-ből, fájlrendszerből vagy megosztáson keresztül is láthatóak lesznek-e a streaming-adataink. Azaz, az adatokat közönséges fájlrendszeren és megosztáson keresztül is elérhetjük.

A streaming-adatokat külön file groupba kell terelni, ezért eleve így hozom létre az adatbázist:

```
CREATE DATABASE FSTeszt ON PRIMARY (
NAME = FSTesztData,
FILENAME = N'C:\test\FSTeszt.mdf' ),
FILEGROUP FileStreamGroup1 CONTAINS FILESTREAM (
NAME = FileStreamData,
FILENAME = N'C:\test\FileStreamData')
```

A CONTAINS FILESTREAM jelöli ki a speciális file groupunkat. A FILENAME valójában ebben az esetben nem egy fájlnév, mint megszokhattuk, hanem egy könyvár elérési útja, itt tárolódnak fájlokban az stream-adatok.

Hozzunk létre egy táblát, ami használ streaming-adatokat:

```
CREATE TABLE Kepek (
Id uniqueidentifier rowguidcol not null primary key
      default (newid()),
Name nvarchar(256) not null,
Photo varbinary(max) filestream null)
```

Mindenképpen kell egy rowguidcol-os oszlop, ez lehet PK is, vagy csak egy sima oszlop, de kell, ezzel tudja az adatbázis összehozni a tábla sorait a fájlokkal.

Az adatokat beküldhetjük SQL-parancsként is, a megszokott TDS-csatornát felhasználva. Azonban pont azért rakták ki fájlrendszerbe ezeket a nagy adatokat, hogy NE TDS-en keresztül, hanem SMB-vel, a Windows fájlmegosztásán keresztül érjük el őket. Ez kicsit szokatlan, hisz használunk ugyan SQL INSERT-et, de azért kell fájlkezelés is a megoldásban.

Nincs managed felület az adatok kezelésére, natív API van, azt lehet használni .NET-ből, interopon keresztül.

A megoldás menete vázlatosan a következőképpen alakul.

1. Hozzákapcsolódunk a szerverhez SqlConnectionnel, tranzakciót indítunk, ágyazzunk az adatoknak a fájlrendszerben. Enélkül, ha NULL maradna az oszlop értéke, nem jönne létre fájl a diszken, így a következő lépés se menne.

3. Visszaolvassuk a sorunkhoz tartozó stream mint fájl elérési útját és egy tranzakció-azonosítót, ez kell majd a következő lépésben. Mindkettőre van egy új függvény, illetve metódus (kiemelve).

```
select Photo.PathName() PathName,
get_filestream_transaction_context() TranCtx
from Kepek where Id = @Id
```

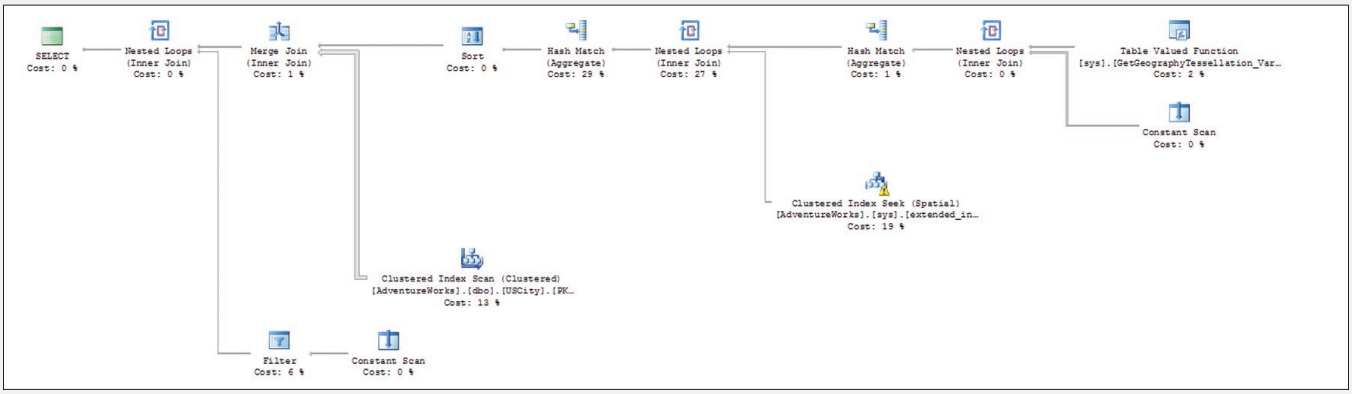
4. Kapunk egy natív függvényt, az OpenSqlFilestream-et, azzal lehet megnyitni tranzakcionálisan a streamünket mint fájlt. Mivel natív függvény, interopon keresztül érjük el:

```
[DllImport("sqlncli10.dll", SetLastError = true, CharSet
= CharSet.Unicode)]
public static extern SafeFileHandle OpenSqlFilestream(
   string FilestreamPath,
   UInt32 DesiredAccess,
   UInt32 OpenOptions,
   byte[] FilestreamTransactionContext,
   UInt32 FilestreamTransactionContextLength,
   LARGE_INTEGER_SQL AllocationSize
   );
```

A paraméterek értékét az előző pont select-je szolgáltatja, amit egy readerrel érek el:

```
byte[] tranCtx = (byte[])reader["TranCtx"];
SafeFileHandle h = OpenSqlFilestream(
(string)reader["PathName"],
DESIRED_ACCESS_WRITE,
0,
tranCtx,
(uint)tranCtx.Length,
new LARGE_INTEGER_SQL(0))
```

Kapunk egy Handle-t, amit az interop-réteg egyből be is csomagol egy SafeFileHandle-be.



7. ábra. Szűrés spatial indexszel

5. A Win32 handle-t egy FileStream objektumon keresztül érhetjük el egyszerűen:

```
FileStream fsWrite = new FileStream(h, FileAccess.Write)
```

6. Most már csak írni kell az fsWrite-ba. Éppen ez a lényege a streaming-elérésnek: nem összerakunk egy 34 gigabájtos objektumot, mondjuk bájt[]-öt, és odavágjuk a szervernek, hanem apránként lapátoljuk be az adatokat. Valahogy így például:

```
int readed;
do
{
   byte[] buff = new byte[4096];
   readed = fsRead.Read(buff, 0, buff.Length);
}
while (readed > 0);
```

Az fsRead a helyi képre van megnyitva, amit fel akarunk tölteni a szerverre.

Zárásul érdemes tudni, hogy az új, jelentősen gyorsított full-text index megy a filestream adatokra is.

A teljes példa a [4] címen érhető el.

Tábla típusú paraméterek

Nem skaláris, illetve sok, nem ismert számú skaláris adat átadása például egy tárolt eljárásnak elég körülményes volt eddig. Van, aki vesszővel elválasztott szövegbe rakta ösz-

sze a paramétereket, van, aki XML-ben vitte át a listát az SQL 2000-ben bevezetett OPENXML vagy az SQL Server 2005 XML-típusának segítségével. Mások átmeneti táblába szúrták be az adatokat, amit egy rákövetkező tárolt eljáráshívással dolgoztak fel.

Az SQL Server 2008-ban egyszerűen át lehet adni egy tábla típusú változót a hívott

Hivatkozások a cikkben

1. http://soci.hu/blog/index.php/2008/01/07/sql-server-2008-ujdonsagok-10-hierarchyid-adattipus-1/
2. http://soci.hu/blog/index.php/2008/01/19/sql-server-2008-ujdonsagok-13-hierarchyid-adattipus-5/
3. http://soci.hu/blog/index.php/2008/02/06/sql-server-2008-ujdonsagok-18-terbeli-adattipusok-5/
4. http://soci.hu/blog/index.php/2007/12/13/sql-server-2008-ujdonsagok-5-streaming-adatok-kezelese-kliens-oldalrol/

eljárásnak. Ehhez a CREATE TYPE-ot okosították fel, ami már nemcsak álneveket tud létrehozni (SQL Server 7), vagy CLR-típusokat (SQL Server 2005), hanem táblatípusokat is. Például:

```
CREATE TYPE LocationTableType AS TABLE (
  LocationName VARCHAR(50),
  CostRate INT
)
```

Aztán paraméterként így lehet használni tárolt eljárásban:

```
CREATE PROC InsertLoc (
  @par LocationTableType READONLY
)
```

A @par ezek után feldolgozható, mint egy sima lokális tábla típusú változó, lehet joinolni, insert-select-tel másik táblába beszúrni stb.

Az újítás legfőbb előnye, hogy egy hálózati körülfordulással, strukturált, típusos adatokat tudunk feljuttatni a szerverre.

Ügyféloldalon ADO.NET-ből nagyon egyszerű feltölteni a táblaparamétert, egyszerűen egy sima DataTable-t kell adatokkal feltölteni, ami már direktben mehet is be a szervernek.

Változó inicializálás deklarációkor és egyszerűsített értékadás

Az declare után azonnal értéket is lehet adni az új lokális változónak:

```
declare @a int = 5
```

A C nyelvekben már megszokott módon össze lehet vonni egyes operátorokat és az értékadást egy műveletbe:

```
set @a += 4; @b *= 4; ...
```

Jól jön ez például ciklusokban, amikor növelgetni kell egy változót.

Komponálható adatmódosító műveletek

Az OUTPUT kulcsszó már ismerős lehet az SQL Server 2005-ből, egy DML- (INSERT, UPDATE, DELETE) művelet által érintett sorokat lehetett kirakni tábla típusú változóba vagy lokális változókba.

A 2008-ban ezt tovább bővítették, így a kimenet bemenetként szolgálhat egy INSERT utasítás részére, azaz össze lehet csövezni mindenféle átmeneti tábla nélkül a DML-műveleteket. Innen a komponálható DML elnevezés. Lássunk egy egyszerű példát:

```
create table t1(col1 int);
create table t2(col1 int);

insert into t1 values (1),(2),(3);

insert into t2(col1) select col1 from
(update t1 set col1 = col1 + 1
output inserted.col1) as d;

select * from t2

col1
----
2
3
4
```

Egyszerűen nevet kell adni az output kimenetének, és máris táblaként kezelhetjük.

Lássunk egy összetettebb auditálást megvalósító példát:

```
CREATE PROC InsertLoc (
  @par LocationTableType READONLY
)
CREATE TABLE Stock (Stock VARCHAR(10) PRIMARY KEY,
Qty INT CHECK (Qty > 0));
CREATE TABLE Trades (Stock VARCHAR(10) PRIMARY KEY,
Delta INT CHECK (Delta <> 0));
CREATE TABLE AuditChanges (Action varchar(6), Stock
VARCHAR(6), Qty INT);
GO
INSERT Stock VALUES('MSFT', 10), ('BOEING', 5);
INSERT Trades VALUES('MSFT', 5), ('BOEING', -5), ('GE',
3);
GO
INSERT INTO AuditChanges
SELECT * FROM
(
MERGE Stock S
  USING Trades T
  ON S.Stock = T.Stock
  WHEN MATCHED AND (Qty + T.Delta = 0) THEN
    DELETE
  WHEN MATCHED THEN
    UPDATE SET Qty += T.Delta
  WHEN NOT MATCHED THEN
    INSERT VALUES(Stock, T.Delta)
  OUTPUT $action, T.Stock, inserted.Qty
) tab (action, stock, qty);
GO

select * from AuditChanges

Action Stock  Qty
------ ------ --
DELETE BOEING NULL
INSERT GE     3
UPDATE MSFT   15
```

Sorkonstruktor

A VALUES most már egyszerre több értéket is tud kezelni:

```
INSERT INTO @Movie (MovieRatingId, Title)
VALUES
(3, 'SQL the Movie'),
(4, 'SQL Massacre'),
(1, 'SQL for Everyone')
```

Ugyanez természetesen működik SELECT parancsokban is. Jól jöhet a VALUES által előállított virtuális tábla, ha skaláris értékekből, például paraméterekből kell röptében táblát előállítani.

Zárszó

Az SQL Server 2008 számos alkalmazásfejlesztési újdonsága révén egyrészt hatékonyabbá válik az eddig megszokott alkalmazások fejlesztése (HiearchyId, FileStream stb.), másrészt teljesen új, térinformatikai adatokkal operáló fejlett alkalmazások írására nyílik mód. Gondoljuk csak el, hogy a vektoros spatial adatok Windows Presentation Foundation alapú (vektoros) megjelenítési felülettel vagy Silverlight alapú webes felülettel kombinálva elképesztően attraktív és informatív programok készítését teszi lehetővé.

Soczó Zsolt
(http://soci.hu)
MCSD, MCDBA, ASP.NET MVP
Research Engineer, Qualification Development Kft.

Adatprofilozás az Integration Services 2008 segítségével

Nyolc módszer, amellyel villámgyorsan feltérképezhető egy vállalati rendszer adatminősége.

Az Integration Services 2008 legnagyobb újdonsága kétség kívül a most bemutatandó adatprofilozó eszköz lesz. Ha egy mondatban kellene megfogalmaznom, hogy mit is jelent az adatprofilozás, akkor azt a következőképpen tenném: az adatprofilozás az a folyamat, amelynek során megvizsgáljuk a forrásrendszerek adatait, azokról statisztikákat készítünk (milyen az eloszlásuk, kitöltöttségük, mennyi a minimumuk, maximumuk, átlaguk, ...) és információt gyűjtünk (milyen minőségűek, mennyire tiszták az adatok és integrálhatóak-e az adattárházba...).

Az adatprofilozás során találkozik először az adattárház-tervező az éles adatokkal. Ekkor kezd el kialakulni benne egy kép az adatminőségről, és a profilozás befejezésekor már elegendő információja lesz ahhoz, hogy a fizikai adatmodellt is el tudja készíteni. Minél pontosabban sikerül a profilozás, annál jobb képet kap a forrásrendszerekről, és annál pontosabban tervezheti meg az adatmodellt. Épp ezért fontos, hogy a profilozás során minél kifinomultabb képet kapjon a forrásrendszerek adatairól.

Ám a profilozást nemcsak a forrásrendszerek analizálására lehet használni, hanem az elkészült adattárház adatminőségének biztosítására és ellenőrzésére is. Képzelje el a cikket olvasó szakember, hogy neki kell átvennie a szállítók által elkészített adattárház több száz SSIS-betöltő csomagját. Hogyan fogna hozzá? Egyenként megnézni biztos nem lesz ideje, ám ha megprofilozza az adattárház adatait, akkor képet kaphat a betöltési folyamtok helyes vagy helytelen működéséről.

Szintén jó szolgálatot tehet nekünk az adatprofilozó taszk a betöltések előtti adathelyesség-ellenőrzésekre. Vajon az érkező új adatok minimuma és maximuma a tűréshatáron belül van? (Például: óvodások nem lehetnek 10 évesek.) Vajon ki vannak töltve az extraktumokban (érkező forrásadatokban) a mezők, vagy tele vannak NULL értékkel? Az adatprofilozó taszk segítségével mindezekre a kérdésekre választ kaphatunk, és már az extraktumok megérkezése után azonnal, jóval a betöltés megkezdése előtt képünk lehet az adathelyességről, és dönthetünk: elindítjuk a betöltést vagy kérünk új forrásadatot.

A forrásrendszerek profilozására eddig is léteztek eszközök, de most a Microsoft is kifejlesztett egyet, és ezt jó szokásához híven becsomagolta az SQL Server 2008 programcsomagba. (Pontosabban az Integration Services 2008-nak, az SQL Server 2008 mellé csomagolt ETL-eszköznek a részévé tette.)

Vegyük hát górcső alá az új adatprofilozó eszközt, és nézzük meg, hogyan támogatja az üzletiintelligencia- vagy adattárház-bevezetések folyamatát, és milyen segítséget nyújt a forrásrendszerek felméréséhez.

Az adatprofilozó taszk működésének áttekintése

Az adatprofilozó taszk

1. felolvassa az adatokat a forrástáblákból;
2. elvégzi az adatok profilozását;
3. az elemzés eredményét kiteszi egy XML-fájlba (vagy beteszi egy SSIS-változóba).

A profilozás eredményét megnézhetjük egy erre a célra külön kifejlesztett eszköz, a Data Profile Viewer segítségével.

(Megjegyzés: A Data Profile Viewer nem található meg a programfájlok között – legalábbis a CTP5-ös verzióban. Az alkalmazást a

„%programfiles%\Microsoft SQL Server\100\DTS\Binn\DataProfileViewer.exe” könyvtárból kell kézzel elindítani.)

Milyen vizsgálatokat végezhetünk az adatprofilozó taszkkal?

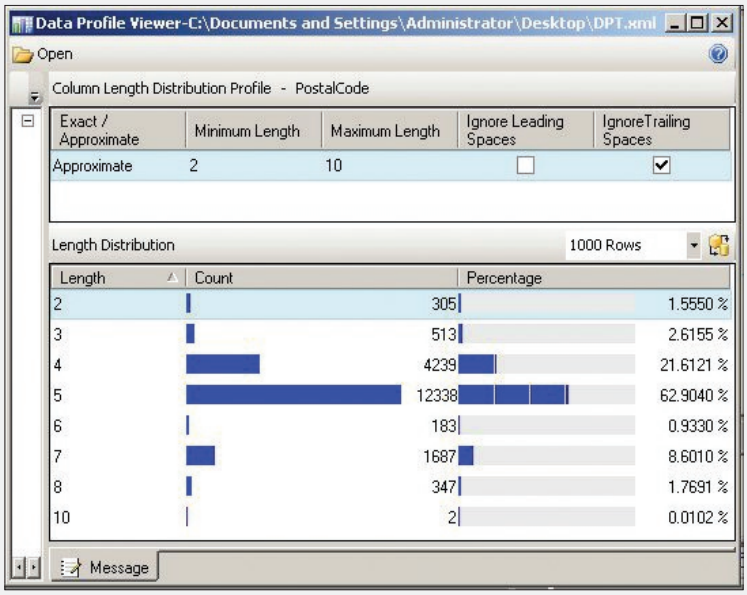
A most következő fejezetekben végigmegyünk az adatprofilozási módszereken, és megvizsgáljuk, hogy melyik módszert mire lehet használni a gyakorlatban.

Adathosszeloszlás-elemzés (Column Length Distribution)

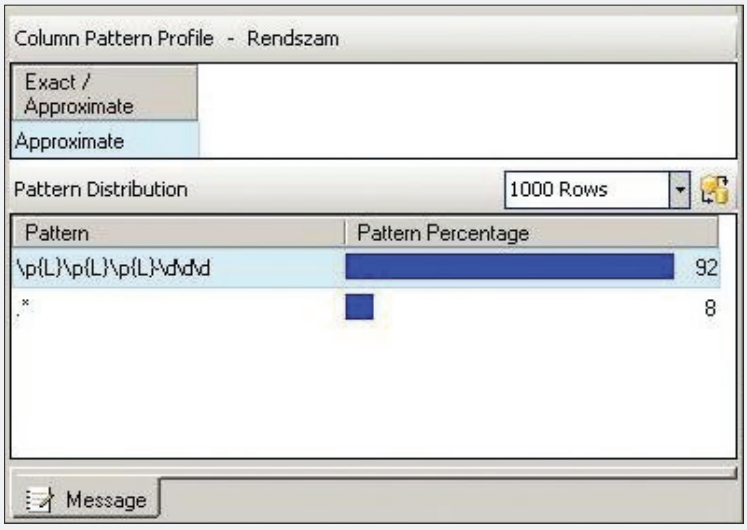
Az adathosszok vizsgálatával az a célunk, hogy megnézzük, milyen a forrásoszlopban tárolt adatok hosszának eloszlása. Ezt két okból tesszük:

1. ha nincs információnk a forrásrendszerek adattípusairól és hosszáról (mert, mondjuk, csak egy szövegfájlt kapunk), akkor csak ez alapján fogjuk tudni meghatározni, hogy az adatmodellben az adott mező milyen széles legyen;

2. így ellenőrizni tudjuk, hogy megfelelnek-e a feltételezett hossznak az adatbázisban tárolt értékek.



1. ábra. Adathosszok eloszlása (Column Length Distribution)



2. ábra. Patternanalízis

Mondok egy példát: ha az irányítószám-mezőben van 6 karakter hosszú és 3 karakter hosszú mező is, akkor biztos, hogy a forrásrendszer nem validálja az irányítószámokat, és valószínű, hogy az adattárházba töltés előtt tisztítani kell őket.

Korábban ezt Min és Max értékek leválogatásával ellenőriztük, de ezek a számok nem adtak felvilágosítást arról, hogy nem megfelelő az adatminőség, akkor mekkora a gond (azaz például az összes irányítószám hány százaléka 6 jegyű). Erre ad most megoldást az SSIS adatprofilozó taszkja.

Lefuttatva az AdventureWorks Address táblájának PostalCode oszlopára az adathossz-eloszlás vizsgálatát azt kapjuk, hogy az összes PostalCode 63 százaléka 5 számjegy hosszúságú (2-es minimum- és 10-es maximumértékekkel). Nem ismerem az amerikai irányítószámok rendszerét, de ha ez magyarországi példa lenne, akkor erősen el kéne gondolkodnunk a forrásadatok megbízhatóságán...

Kulcsképesség-elemzés (Candidate Key)

Az adatprofilozó taszkkal megvizsgálhatjuk, hogy egy kulcsnak gondolt oszlop tényleg alkalmas-e kulcsképzésre vagy sem. Ha ismétlődő elemeket tartalmaz, akkor saját maga nyilván nem lehet kulcs az adattárházban (illetve meg kell nézni, hogy hány ismétlődést tartalmaz, és ebből el lehet dönteni, hogy kulccsá tehető-e vagy sem).

Az eszköz lehetőséget biztosít összetett kulcsok kulcsképesség-vizsgálatára is.

Kitöltöttségvizsgálat (Column Null Ratio)

A kitöltöttségvizsgálat talán a leggyakrabban alkalmazott adatprofilozási módszer, melynek során megvizsgáljuk, hogy egy oszlop hány NULL értéket tartalmaz (az oszlop hány százaléka NULL).

A forrásrendszerek oszlopainak kitöltöttségét vizsgáljuk például akkor, amikor dimenziók attribútumait tervezzük.

Mondok egy példát: Tegyük fel, hogy a vevőtörzsnek a forrásrendszer dokumentációja szerint van olyan oszlopa, amely a vevőcsoportokat tartalmazza. Első ránézésre ebből érdemes lesz attribútumot építenünk, igaz? Sőt, a fejünkben már körvonalazódik egy vevő–vevőcsoport hierarchia is... Ám teljesen felesleges a vevőcsoportra attribútumot vagy hierarchiát építenünk, ha annak kitöltöttsége nem éri el a 10 százalékot.

A kitöltöttség vizsgálatakor azonban nemcsak azt kell vizsgálnunk, hogy az oszlop hány százaléka tartalmaz NULL értéket, hanem azt is, hogy hány százaléka tartalmaz üres sztringet. Sajnos ez utóbbi keresését az eszköz aktuális verziója nem támogatja.

Mintakeresés (Pattern Analysis)

Na ez egy érdekes cucc. Nem csinál mást, mint kiszedi az oszlopból a mintákat. Megnézi, hogy milyen mintát húzhat rá az adatokra, és ezeknek a mintáknak az eloszlását vizsgálja. Mindjárt mondok egy példát, hogy érthetőbb legyen.

Csináltam egy demótáblát, és feltöltöttem ABC-123 formátumú rendszámokkal (persze tettem bele egy-két hibát. Például: ABC123, ABC 123, ...). Megprofilozva ezt az oszlopot, a következőt kaptam:

Három betűből, kötőjelből, három számból áll a rendszám az oszlop összes elemének 92 százalékában. A maradék nyolc szálalék nem felel meg a szabálynak *(2. ábra)*.

A mintakeresés segítségével validálhatjuk például a telefonszámokat is. Hány százalékuk tartalmaz körzetszámot, hány nem, hány tartalmaz kötőjelet, szóközt, pluszjelet, hány nem stb.

Oszlopstatisztikák (Column statistics)

Megmutatja az oszlopról, hogy mennyi a

- minimuma;
- maximuma;
- átlaga;
- szórása.

Amikor adatprofilozásról beszélünk, a legtöbb embernek e négy mutató ugrik be, hiszen ezeket használjuk a leggyakrabban. A minimum- és maximumértékekből képet kaphatunk az oszlopban tárolt adatok tartalmáról. Az átlag és a szórás pedig segíteni fog nekünk az Analysis Services Measure-jeinek méretezéséhez.

Sajnos a Data profiling taszk jelenlegi (CTP5-ös verziója) nem tud minimumot és maximumot számolni szöveges oszlopok esetén. Ez azért baj, mert az első (min) és az utolsó (max) szövegek (szavak) is sok információt elárultak az adatbázisban tárolt adatokról (pláne akkor, amikor az oszlopoknak nem beszédesek a neveik, vagy amikor az oszlopban mást tárolnak, mint ami a rendeltetésük).

Értékeloszlás-analízis (Column Value Distribution)

Az értékeloszlás-analízissel megállapíthatjuk, hogy hány különböző érték található az oszlopban, és azoknak milyen az eloszlásuk. Mutatom:

Az AdventureWorks adatbázis [Sales] [vSalesPerson] nézetében 4 különböző job title található, és ezek eloszlása olvasható le a *3. ábrából.*

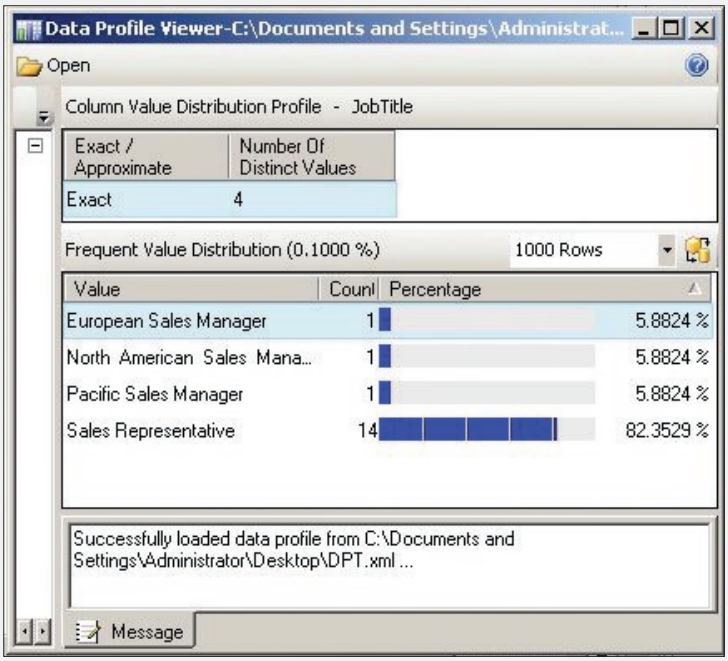
Összefüggés-elemzés (Functional Dependency)

Az összefüggés-elemzés elvégzésével arra a kérdésre kapunk választ, hogy egyik oszlop elemei mennyire függnek a másik oszlop elemeitől. Használhatjuk ezt a módszert hierarchiák keresésére vagy hierarchia-hipotézisek tesztelésére. Mondok egy példát: Egy munkatársnak csak egy főnöke van, vagy esetleg lehet több is? Ha csak egy van, akkor építhetünk belőle hierarchiát, ha több van, akkor nem.

Az összefüggés-elemzés támogatni fog minket az OLAP-adatmodell tervezésénél is. Ha két oszlop között látszólag hierarchikus kapcsolat van, de ezt az oszlopokban tárolt adatok nem támasztják alá, akkor felesleges rá hierarchiát tervezni. És ez jó, ha már a tervezési fázisban eldől, és nem akkor döbbenünk rá, amikor minden kész, már csak nem tudjuk felösszegezni a hierarchiát...

Részhalmazok keresése (Value Inclusion)

Az elemzéssel megállapíthatjuk, hogy az egyik oszlop adatai részhalmazai-e másik tábla adott oszlopának. Remekül használhatjuk ezt a módszert arra, hogy kapcsolatokat keressünk két tábla között, vagy kapcsolatok erősségét és irányát vizsgáljuk. Különösen akkor lehet nagy hasznunkra ez az elemzés, ha a forrásadatok szövegfájlban érkeznek, és a forrásrendszer dokumentációjából nem derül ki, hogy az adott szövegfájl melyik másikkal áll még relációban.



3. ábra. Értékeloszlás-analízis (Column Value Distribution)

Összefoglalás

Az SQL Server 2008 adatprofilozó taszkja jó kis eszköz. Sokat fog nekünk segíteni a forrásrendszerek felmérésekor, az adattárházak minőségbiztosításakor vagy akár a forrásadatok betöltés előtti validálásakor.

Mindazonáltal a jelenlegi (CTP 5-ös) verziónak még vannak olyan hiányosságai, amelyekről érdemes néhány szót ejteni.

Az első és talán legfontosabb, hogy az adatprofilozó taszk csak SQL Server 2000-es vagy újabb verziójú adatbázisokat képes profilozni. Szövegfájlokat, Oracle- vagy más adatbázisokat sajnos nem. Szintén sajnálatos, hogy a jelenlegi verziót még nem készítették fel arra, hogy az adatprofilozás során talált anomáliákat kommentezni, a profilozás eredményét valamilyen kulturált formában dokumentálni lehessen, és a döntéshozók elé lehessen tárni. Ehhez csak arra lenne szükség, hogy a profilozás eredményét exportálni lehessen Word- vagy Excel-dokumentumba.

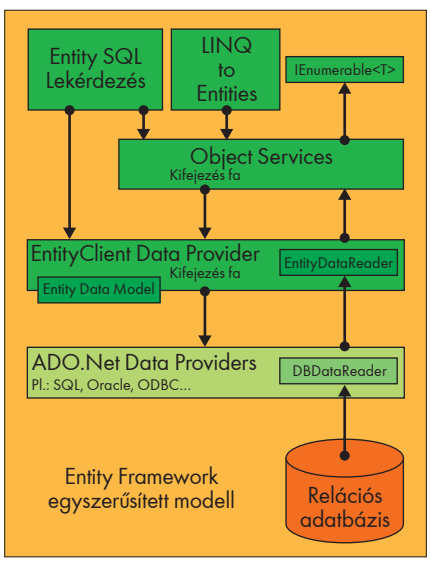
A jelenleg bemutatott adatprofilozó azonban még bétaverzióban van, így bízhatunk abban, hogy a végleges termékben ezeket a hiányosságokat is sikerül pótolni. Addig is érdemes elkezdeni az ismerkedést az új adatprofilozó taszkkal, hogy mire elkészül a végleges Integration Services 2008 verzió, már megismerjük annyira az eszközt, hogy fel lehessen mérni az adott vállalatnál az adatvagyon minőségét, és kialakuljon egy átfogó kép a forrásrendszerek helyes vagy helytelen működéséről.

Kővári Attila
(www.biprojekt.hu)
BI-bevezetési tanácsadó, SQL Server MVP

Az egységbe zárt erő: ADO.Net Entity Framework

Minden fejlesztőkörnyezetnek segítséget kell nyújtania az adatkezeléshez. A .Net Frameworknek a kezdetek óta megvan a maga átgondolt adatelérő technológiája. Ez a feladat nem is olyan egyszerű, nem csoda hát, hogy sokféle megoldás látott már napvilágot. Az ADO.Net is folyamatosan fejlődik, és most eljött az ideje, hogy megismerkedjünk az Entity Frameworkkel.

Az alapproblémát az adatbázisok és a programnyelvek szemléletkülönbsége okozza. A két technológia között keletkezett szakadék áthidalását megkísérlő konstrukciókat szemléletmódjuk alapján három csoportra tudjuk osztani *(Transaction Script, Table Module és Domain Model)*.[1] Az első csoportba a legegyszerűbb megoldások kerülnek, amikor adatelérés közben elvégezzük a megfelelő műveleteket, ellenőrzéseket az adatainkkal. A másik csoportba azokat sorolhatjuk, amelyek a relációs adatbázisok táblázataira támaszkodva táblaszerű objektumhalmazokkal dolgoznak. Ilyen például a Dataset megoldása, ami úgy néz ki a memóriában, mint egy forró vizes mosásban alaposan összement relációs adatbázis. A harmadik csoportba a kevésbé megalkuvó, üzleti objektum központú megoldások sorolhatók. Ez utóbbinak elvitathatatlan előnye, hogy észrevétlenül belesimul az OOP alapelveit elegánsan használó alkalmazások kódjába. Ezt célozza meg a .Net újabb verzióinak egyre igényesebb adatelérő technológiája.



Hogy kinek melyik megoldást kell választania, azt (jó esetben) nemcsak a fejlesztő hozzáértése, szokásai alapján kell meghatározni, hanem a fejlesztői környezetünk, eszközeink által támogatott módszerek is jelentősen befolyásolják a döntést. Életünk mindezidáig az ADO.Net és a Visual Studio által támogatott Dataset jegyében telt (Table Module). Az ebből való kitörést a támogatások megszüntetésével büntette fejlesztői környezetünk, bár a Domain Model elkötelezett híveinek volt lehetőségük alternatív eszközökkel enyhíteni ezt a fájdalmat.

Mostantól azonban házon belül is választhatunk. A változás már régóta a levegőben van (lehet, hogy Dániában is ezt érezték?), és hogy ennyit kellett rá várni, talán azt jelenti majd, hogy alaposan átgondolt, könnyen használható eszközöket és tiszta, áttekinthető környezetet kapunk.

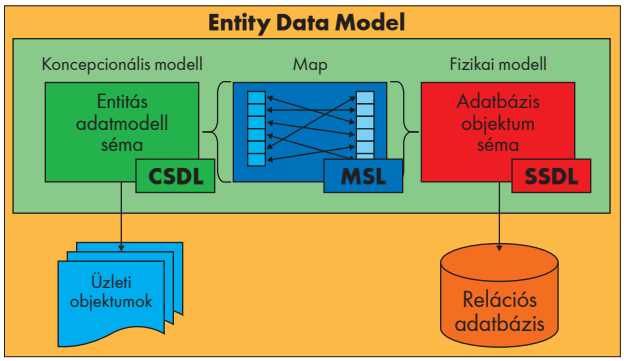
Ez a környezet az ADO.Net Entity framework[2], ami a réteges építkezés jegyében rákhúz még pár emeletet az eddigi ADO.Net-infrastruktúrára. Mint minden jó eszköz, az Entity Framework is problémák megoldásában segít. Ezeken a problémákon végigverekedve magunkat igyekszünk megismerkedni

az SQL Server 2008 hozta új adatelérő keretrendszerrel.

A sárga köves út

A korrekt adateléréshez hat fontos lépésen keresztül vezet az út, ezeket tekintjük át a továbbiakban.

Ha két különböző struktúra között szeretnénk adatot átvinni, az első probléma, amit meg kell oldanunk: meg kell határoznunk, hogy az egyik struktúrából hogyan jutunk a másikba, ez a **mapping**. A legegyszerűbb esetben a két oldal felépítése hasonló, legfeljebb az adatok nevei, sorrendjük, esetleg mennyiségük különbözik. Az igazi kihívást jelentő esetekben azonban az adatok struktúrája, absztrakciós szintje is teljesen eltérő lehet, miközben az információtartalom persze azonos. Erre a problémakörre nyújt elegáns megoldást az EF a háromrétegű (koncepcionális modell – megfeleltető réteg – fizikai modell) leírással.



Ha már tudjuk, hogyan feleltethetjük meg az adatforrásunk elemeit üzleti objektumaink tulajdonságainak, a következő kérdés az lesz, hogyan tudjuk ez alapján az információ alapján éppen a minket érdeklő halmazt megszerezni a háttértárból. Ez a **lekérdezés**. SQL alapú adatbázisoknál erre való a SELECT. Nem minden adatforrás SQL alapú, és ha azok is, köztük is van a félreértésekre elég okot adó különbség. Mégis ezt a lekérdezési nyelvet ismerik a legtöbben, és megfelelő alapot nyújthat egy olyan általánosításnak, amely könnyen alakítható az egyes konkrét adatforrások igényeihez. Az EF saját SQL nyelvet használ üzleti objektumainak válogatásához. Mivel ez nem adatbázisban, hanem egy objektumorientált környezetben történik, számos olyan elemet vittek az Entity SQL-be, amely ezeket a sajátosságokat (például öröklődés, objektumtulajdonságok) támogatja. Bár ezt az SQL-t a relációs SQL-ekhez hasonlóan stringként is adagolhatjuk, szerencsére nincs rá szükségünk, hiszen a LINQ pont ebben lesz segítségünkre.

Adatokat nemcsak lekérdezni szeretnénk, hanem létrehozni, módosítani is (különben nem lenne mit lekérdezni). A harmadik kérdés az, hogyan tudjuk a memóriában ténfergő objektumainkról eldönteni, hogy azokat vajon érdemes-e átadni a háttértárnak. Sőt, azok a fránya háttértárak többnyire azt sem maguktól döntik el, hogy amit kaptak, az vajon új adat-e vagy csak valami módosítás. Ezt tudjuk megoldani úgy, hogy a beolvasott adatokat megjelöljük valami gyorsan kopó festékkel, így látszódik rajtuk a változás. Amelyiken meg egyáltalán nincs festék, az új. Ehhez a **változáskövetéshez** – hogy elég finoman hangolt legyen – az EF a WPF adatkapcsolásánál megismert PropertyChanging-PropertyChanged párost lovagolja meg.

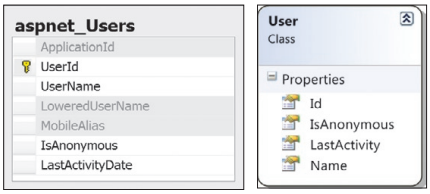
Ha idáig eljutunk a gátfutásban, már belátjuk a vidéket. Hétköznapi kérdések merülnek fel. Hogyan tudjuk rávenni adatelérő rétegünket a **tárolt eljárások használatára**? A tárolt eljárásokat elsőre talán a teljesítményoptimalizáció eszközének tekinthetjük. Nem kizárólag azok. Fontos szerepük van például az adatbázis-szerkezetünk, adatelérő logikánk függetlenítésében, továbbá biztonsági kérdések megnyugtató rendezésében. Az EF mind a lekérdezések, mind az adatmódosítások során lehetőséget ad tárolt eljárások használatára. Meg kell azonban barátkoznunk a gondolattal, hogy ebben az esetben kódoldalon leszünk kénytelenek némi flexibilitási veszteséget elkönyvelni.

Persze a mai alkalmazások olyanok, mint egy ogre, mert azok meg olyanok, mint a hagyma – többrétegűek. Ha az üzleti objektumainkat ki kell szakítanunk megszokott környezetükből, utaztatásuk céljából sorosítanunk kell őket. A túloldalra érve teljesen elvesztik a fonalat, és nem lesz olyan egyértelmű a változáskövetés megvalósítása. Ennek megoldása lesz az ötödik problémánk.

A végén felmerül a kérdés, hogy ez vajon tényleg képes-e bármilyen adatforrással együttműködni. De mire idáig érünk, szinte kizárólag visszatekintéssel megfejthetjük az **adatforrás-függetlenség** titkát.

A problémák közül (mapping, lekérdezés, változáskövetés, tárolt eljárások, többrétegű alkalmazások, adatforrás-függetlenség) nézzük meg az elsőt. Nem csupán azért, mert ez az első a sorban, hanem azért is, mert ez egyúttal – az Entity Data Model megismerése közben – a megoldás lényegét is feltárja számunkra.

Az első probléma: a két világ közötti tolmácsgép (ORM*)



Az objektumorientált és a relációs világ közötti adattranszformációnak ideális esetben az alkalmazásunk irányából teljesen el kell tüntetnie az adatbázisok szerkezeti sajátosságait. Nézzünk rá egy példát.

Kiindulási adatbázisunk legyen az ASP.Net autentikációs adatbázisa. Ezt könnyen előállíthatjuk magunknak a Visual Studio Command Promptban az `aspnet_regsql` parancs segítségével. Kezdjük az [aspnet_Users] táblával a munkát! Alkalmazásunk oldalán pedig egy osztályra van szükségünk, amit Usernek nevezünk el. Mint látjuk, a két oldal máris különbözik, bár a mezők számának különbözőségétől egyelőre eltekintünk. Van itt elnevezési különbség, és a számszerűségük sem passzol, hiszen a táblázatban sok rekord van, de a User objektumban csak egynek az adatait tárolhatjuk. Ha az alkalmazásunkban ilyen osztályokat szeretnénk használni,

Egyszerűsítve

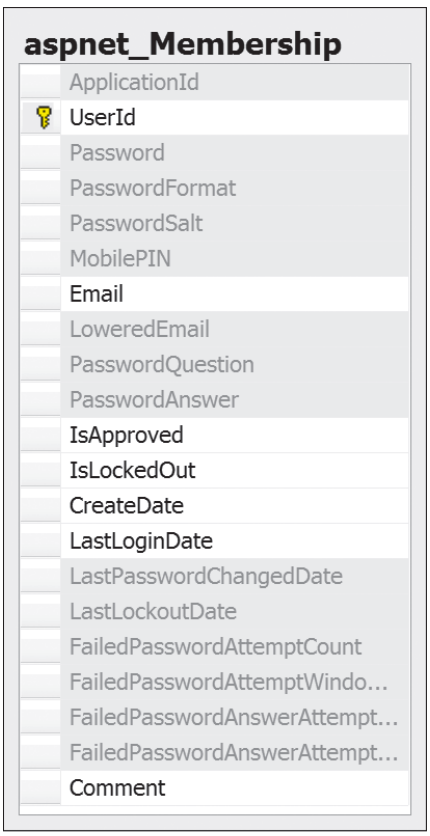
A cikkben szereplő kód- illetve XML-részletek nem teljesek, néha le vannak egyszerűsítve az átláthatóság és könnyebb megérthetőség érdekében. A cikkben szereplő modell egy VS projectben letölthető a NetAcademia honlapjáról. Ugyanott található egy nagyobb ábra, amely a három XML alkotóelemeinek összefüggéseit tartalmazza.

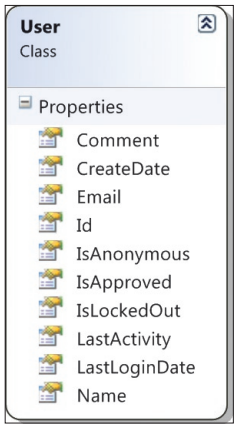
* Object-Relational Mapping

használatuk előtt gondoskodnunk kell valahogyan az adatok betöltéséről.

```
List<User> users = new List<User>();
DbProviderFactory dbFactory = DbProviderFactories.GetFactory("System.Data.SqlClient");
using (DbConnection connection = dbFactory.CreateConnection())
{
  connection.ConnectionString = @"Data Source=.;
                  Initial Catalog=EFSample;
                  Integrated Security=true";
  using (DbCommand command = connection.CreateCommand())
  {
    command.CommandText = @"SELECT
                UserId, UserName, IsAnonymous, LastActivityDate
                FROM dbo.aspnet_Users";
    connection.Open();
    DbDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
      users.Add(new User {
        Id=reader.GetGuid(0),
        Name=reader.GetString(1),
        IsAnonymous=reader.GetBoolean(2),
        LastActivity=reader.GetDateTime(3)
      });
    }
  }
}
```

A fenti kódrészlet a teljes táblázatot betölti, éppen ezért sok User objektumra van szüksége (List<User>).





Ez a feladat annyira gépies, hogy akár rá is bízhatnánk a gépre. Ha valami rövid deklaratív módot keresünk erre a célra, akkor hamar rátalálunk a LINQ to SQL attribútumos módszerére. Az a módszer azonban közvetlenül az adatbázis egy táblájához társítja osztálydefiníciónkat. Ennek a hátránya hamar megmutatkozik.

Bővítsük ki User osztályunkat pár mezővel. Vegyük észre, hogy az ezekhez a mezőkhöz tartozó adatokat az adatbázisban már egy másik tábla, az [aspnet_Membership] tartalmazza. Az üzleti modellünkben azonban fölösleges, sőt zavaró lenne, ha két osztályt használnánk erre a célra.

Így tehát nem elegendő osztályonként meghatározni a forrástáblázatot, hanem minden egyes property esetében tudni kellene, hogy melyik táblázat melyik oszlopából jut adathoz. Ezt persze még mindig meg lehetne valósítani attribútumokkal, de annál nagyobb rugalmasságot biztosító megoldást is találhatunk, ha ezeket az adatleképezési mintákat egy külön állományban tároljuk. A formátum persze nem lehet kétséges, hiszen az XML-nek van egy olyan előnye, hogy egyszerű hozzá látványos grafikus szerkesztőt készíteni, ami adathierarchiák esetében nagyon jól fog jönni. Szóval írjuk le mindezt XML formátumban.

Az EF rögtön három állományt is hadrendbe állított (igaz ugyan, hogy mostani állapotában fejlesztés közben már egy összevont fájlban található). Mi „természetesen" a középsővel (MSL) kezdjük.

Az adatmegfeleltetés leírása (MSL*)

Adatbázisunkban táblázatok vannak, ezek sorait tudjuk az üzleti (koncepcionális) modellünkben lévő objektumpéldányoknak megfeleltetni. Ezek az objektumok lesznek a mi egységeink, entitásaink, az entitások csoportjait pedig EntitySetnek nevezzük, amelyek így aztán a táblázatokkal állnak rokonságban. Egy ilyen EntitySet neve legyen Users. Tehát az adatok összerendelését kezdjük így.

```
<EntitySetMapping Name="Users">
</EntitySetMapping>
```

Ha most a legegyszerűbb leképzést szeretnénk ábrázolni, akkor a következőképpen írhatnánk azt le.

```
<EntitySetMapping Name="Users">
...
  <ScalarProperty Name="Id" ColumnName="UserId"
        />
  <ScalarProperty Name="Name"
        ColumnName="UserName" />
  <ScalarProperty Name="IsAnonymous"
        ColumnName="IsAnonymous" />
  <ScalarProperty Name="LastActivityDate"
        ColumnName="LastActivityDate" />
...
</EntitySetMapping>
```

A kipontozott részek a további finomítások helyeit jelzik.

Ebből az látszik, hogy az üzleti objektumunk Id property-je (Name=) az adatforrásunk táblájának UserId oszlopából (ColumnName=) táplálkozhat. Egyelőre csak épp az nem látszik, hogy melyik az a tábla. Hát írjuk oda azt is!

```
<EntitySetMapping Name="Users">
...
 <MappingFragment StoreEntitySet="aspnet_Users">
  <ScalarProperty Name="Id" ColumnName="UserId" />
  <ScalarProperty Name="Name"
        ColumnName="UserName" />
  <ScalarProperty Name="IsAnonymous"
        ColumnName="IsAnonymous" />
  <ScalarProperty Name="LastActivityDate"
        ColumnName="LastActivityDate" />
 </MappingFragment>
...
</EntitySetMapping>
```

* Mapping Schema Language, ami leírja a koncepcionális és a fizikai rétegek kapcsolatát (C-S mapping)

Máris nyilvánvaló, hogy amelyik listát az üzleti modellben én Usersnek hívom, az a háttértáron egy aspnet_Users nevű táblázat. Így azzal, hogy a táblázatot nem az oszlopoknál jelöljük, egyrészt az ismétlődéseket iktattuk ki az XML-ből, másrészt összeszedetté tettük a formátumot, ami a következő lépésben látható is lesz. Most ugyanis az következik, hogy a kibővített User osztályunk adatmezőihez adunk útmutatást a forrásadatokat illetően.

```
<EntitySetMapping Name="Users">
...
 <MappingFragment StoreEntitySet="aspnet_Users">
  <ScalarProperty Name="Id" ColumnName="UserId" />
  <ScalarProperty Name="Name"
          ColumnName="UserName" />
  <ScalarProperty Name="IsAnonymous"
          ColumnName="IsAnonymous" />
  <ScalarProperty Name="LastActivityDate"
          ColumnName="LastActivityDate" />
 </MappingFragment>
 <MappingFragment
          StoreEntitySet="aspnet_Membership">
  <ScalarProperty Name="Id" ColumnName="UserId"
          />
  <ScalarProperty Name="Comment"
          ColumnName="Comment" />
  <ScalarProperty Name="LastLoginDate"
          ColumnName="LastLoginDate" />
  <ScalarProperty Name="CreateDate"
          ColumnName="CreateDate" />
  <ScalarProperty Name="IsLockedOut"
          ColumnName="IsLockedOut" />
  <ScalarProperty Name="IsApproved"
          ColumnName="IsApproved" />
  <ScalarProperty Name="Email"
          ColumnName="Email" />
 </MappingFragment>
...
</EntitySetMapping>
```

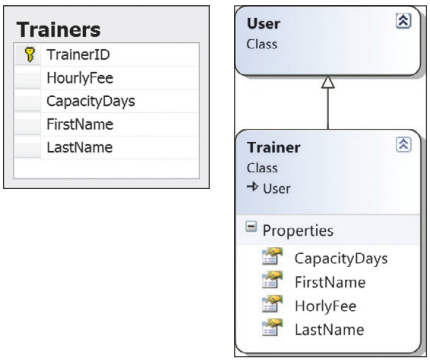
Így már tisztává vált a MappingFragment elnevezés is, hiszen ezek között az elemek között a teljes adatkapcsolásunknak csak egy része kap helyet. Alaposabb szemlélődés közben feltűnik, hogy az üzleti modell oldalán az Id kétszer is értéket kapott! Mindkét mapping fragmentben egyszer. Ez szükségszerű, hiszen ha meg szeretnénk írni azt a kódot, lekérdezést, amelyik megszerzi az adatbázisból a szükséges adatokat, minden üzleti objektumhoz egyértelműen meg kell találnunk a hozzá tartozó rekordot. Az üzleti objektumunk egyedi azonosítója az Id, ezért mindkét táblázatban ez alapján tudunk tájékozódni. Ez a gondolatmenet szerencsésen passzol is egy jólfésült 1–1 adatbázis-kapcsolathoz, ahol mindkét táblának ugyanaz az elsődleges kulcsa, és nem mellesleg az idegenkulcs-kapcsolatért is azok az oszlopok felelnek.

Nagyvonalúan beszélek én itt arról, hogy User osztály, meg hogy üzleti objektum, ugyanakkor az eddig XML-be foglaltakból csak az összesség (Users EntitySet) látszik, de az egyed (User) nem. Segítsünk hát ezen is!

```
<EntitySetMapping Name="Users">
 <EntityTypeMapping TypeName=
          "IsTypeOf(EFSampleModel.User)">
  <MappingFragment StoreEntitySet="aspnet_Users">
   <ScalarProperty Name="Id"
          ColumnName="UserId" />
   <ScalarProperty Name="Name"
          ColumnName="UserName" />
   ...
   <ScalarProperty Name="IsApproved"
          ColumnName="IsApproved" />
   <ScalarProperty Name="Email"
          ColumnName="Email" />
  </MappingFragment>
 </EntityTypeMapping>
</EntitySetMapping>
```

Ezzel az új elemmel, amivel bekereteztük mindkét MappingFragment részt, meghatároztuk, hogy az itt leírt property-sorozat tulajdonosa az EFSampleModel.User nevű osztályunk. Ilyen osztályok sokasága alkothat aztán egy Users nevű listát.

Remélem, hogy ezek alapján az információk alapján már mindenki tudna írni valami egyszerű, többé-kevésbé általános adatbetöltő kódot, vagy legalább belátja, hogy lehet ilyet készíteni.



De miért van vajon EntitySetMapping és EntityTypeMapping elemünk is? Talán feleslegesnek tűnik, hiszen a Users listában persze, hogy User elemek lesznek. Kivéve persze, ha a Usernek vannak leszármazottai.

Gondoskodjunk róla, hogy legyen. Készítsünk magunknak egy Trainers táblát. Lesznek az adatbázisunkban felhasználók, akik közül néhányan oktatók. Csak az oktatóknak tárolunk egy külön táblában oktatóspecifikus adatokat. Tehát egy oktató minden User-adattal rendelkezik, és még ezekkel a speciális adatokkal is. Ez is egy 1–1 kapcsolat, tehát a Trainers.TrainerID az tulajdonképpen az aspnet_Users.UserID. Ha az eddigi leírást a meglévő EntityTypeMapping elemen belül egészítenénk ki egy újabb MappingFragment elemmel, minden User-objektumunknak lennének oktatói tulajdonságaik. De nem ez a cél. Hanem az, hogy legyen egy Trainer osztályunk is. Ehhez lesz szükségünk egy újabb EntityTypeMapping szekcióra.

```
<EntitySetMapping Name="Users">
 <EntityTypeMapping TypeName=
          "IsTypeOf(EFSampleModel.User)">
  <MappingFragment StoreEntitySet="aspnet_Users">
   <ScalarProperty Name="Id"
          ColumnName="UserId" />
   <ScalarProperty Name="Name"
          ColumnName="UserName" />
   ...
   <ScalarProperty Name="IsApproved"
          ColumnName="IsApproved" />
   <ScalarProperty Name="Email"
          ColumnName="Email" />
  </MappingFragment>
 </EntityTypeMapping>
 <EntityTypeMapping TypeName=
          "IsTypeOf(EFSampleModel.Trainer)">
  <ScalarProperty Name="Id"
          ColumnName="TrainerId" />
  <ScalarProperty Name="CapacityDays"
          ColumnName="CapacityDays" />
  <ScalarProperty Name="FirstName"
          ColumnName="FirstName" />
  <ScalarProperty Name="LastName"
          ColumnName=" LastName" />
  <ScalarProperty Name="HourlyFee"
          ColumnName=" HourlyFee" />
 </EntityTypeMapping>
</EntitySetMapping>
```

Az kiderül, hogy ez az újabb típus az előző rokona (mindkettőt tartalmazhatja a Users EntitySet). De vajon melyik a másik ősosztálya? Úgy látszik, tovább már nem halaszthatjuk a koncepcionális modellt leíró részt.

A koncepcionális modell leírása (CSDL*)

Mint azt már sejtetni engedtem, az eddig felépített XML egy háromrétegű építmény köze-

* Conceptual Schema Definition Language, ami az üzleti objektumaink (entitásaink) adattartalmának és ezen objektumok kapcsolatainak leírását tartalmazza

pe. Ahogy az első ábrán is látszik, ez a középső réteg a két szélsőt köti össze mindenféle furfangos módon. Üzleti objektumainkat (legalábbis azok adatrészét) is le kell írnunk XML formátumban. Ezt hívjuk koncepcionális modellnek. Ez alapesetben egy egyszerű osztálydefiníció. De mi máris túlbonyolítottuk, hiszen leszármaztattunk egy új osztályt. Nézzük meg, hogy néz ez ki a CSDL keretein belül.

Induljunk ki abból, hogy szükségünk lesz egy User osztályra, amelyet most nem a class kulcsszóval fogunk definiálni, hanem XML-ben, és hogy ne is keverjük össze egy akármilyen típusdefinícióval, ezt entitás típusnak hívjuk.

```
<EntityType Name="User">
</EntityType>
```

Ilyen User objektumból az adatbázisban lévő tábla sokat tartalmaz, ezért a tábla szintjét is definiáljuk.

```
<EntityContainer Name="EFSampleEntities">
  <EntitySet Name="Users"
             EntityType="EFSampleModel.User" />
  ...
</EntityContainer>
```

Itt jelent meg az EntitySet elnevezés, ami az MSL EntitySetMapping eleménél köszönt vissza. Ez itt csak egyetlen sor, és egy olyan kollekciót definiál, amelynek elemei User típusúak lehetnek. A típus meghatározása az MSL EntityTypeMapping elemében használt típusmeghatározáshoz hasonlóan egy névtérrel együtt használja a típus nevét. Ezt a névteret a Schema elemben definiáljuk, és a C# namespace {} konstrukciójához hasonlóan minden beletartozik, amit a kezdő és vége elem között definiálunk.

```
<Schema Namespace="EFSampleModel"
        Alias="Self" xmlns="...">
  <EntityType Name="User">
  ...
  </EntityType>
</Schema>
```

Mivel ez az XML az adatok és adatmezők megfeleltetéséről szól, itt csak az entitásunk adattagjai érdekelnek bennünket. Tehát a definíciónk is ehhez méltón egyszerű lesz.

```
<EntityType Name="User">
  <Property Name="Id" Type="Guid" Nullable="false"
            />
  <Property Name="Name" Type="String"
            Nullable="false" MaxLength="256" />
  <Property Name="IsAnonymous" Type="Boolean"
            Nullable="false" />
  <Property Name="LastActivityDate" Type="DateTime"
            Nullable="false" />
  ...
</EntityType>
```

A pontok a többi adatmező definícióját jelzik, a megjelenítettekhez hasonló módon.

A CSDL tehát definiálja az üzleti entitásokat. Az MSL pedig valójában nem a C# vagy akármilyen nyelven megfogalmazott típusainkat köti össze az adatforrás adataival, hanem a CSDL-ben definiált entitásokat a majd hamarosan sorra kerülő adatszerkezet-leírással.

Szépen alakul az entitásunk, de egy fontos dolog még hiányzik belőle. Adathalmazokkal való munka során mindig tudnunk kell egyértelműen azonosítani egy adatcsomagot (rekordot, objektumot, entitást). Erre természetesen egy egyedi kulcs szolgál, ami SQL esetében az elsődleges kulcs. Ennek megfelelően itt is kulcsnak hívják.

```
<Schema Namespace="EFSampleModel"
        Alias="Self" xmlns="...">
  <EntityType Name="User">
    <Key>
      <PropertyRef Name="Id" />
    </Key>
    <Property Name="Id" Type="Guid" Nullable="false" />
    ...
  </EntityType>
</Schema>
```

Mint látjuk, a kulcs jelölése egyszerűen egy (vagy több) property-hivatkozást tartalmaz.

És mivel egyszerű entitásunk leírása így már teljesnek mondható, most jött el a pillanat a leszármaztatott Trainer osztály definiálására.

```
<EntityType Name="Trainer" BaseType="EFSampleModel.
            User">
  <Property Name="HourlyFee" Type="Decimal"
            Nullable="false" Precision="18" Scale="0"
            />
  <Property Name="CapacityDays" Type="Byte"
            Nullable="false" />
  <Property Name="FirstName" Type="String"
            Nullable="false" MaxLength="50" />
  <Property Name="LastName" Type="String"
            Nullable="false" MaxLength="50" />
</EntityType>
```

Ebben a definícióban két jellegzetesség látszik. Először is: van egy BaseType attribútum, amely egyértelművé teszi, hogy ez egy leszármaztatás. Másodszor pedig az, ami C#-ban is hasonlóan működik: csak az új adattagokat kell leírni.

Ha több táblázatból állítjuk össze a mi kis csomagunkat (de akár egy jó széles táblázat esetén is), előfordulhat, hogy a User osztályunknak rengeteg tulajdonsága lesz, amelyek között sokkal egyszerűbb lenne eligazodni, ha valami alapján csoportosítanánk azokat. Ez tulajdonképpen kényelmi szolgáltatás, de ezért ne gondoljuk, hogy nincs jelentősége. Ez is, mint a többi kényelmi funkció, növeli munkánk hatékonyságát. A definiálása pedig roppant egyszerű.

A csoportosítást (megint C#-gondolkodás szerint) úgy tudjuk megoldani, hogy definiálunk egy struktúrát, amelyben össze tudjuk csomagolni az együvé tartozó tulajdonságokat. Az entitásunknak pedig ezek helyett a tulajdonságok helyett csak egy tulajdonsága lesz, ami azonban az újonnan definiált struktúránkat tartalmazza.

Nézzük a CSDL-oldalt. Először a csoportosítótípusunk definíciója látható.

```
<ComplexType Name="ManagementType">
  <Property Name="LastActivityDate" Type="DateTime"
            Nullable="false" />
  <Property Name="IsAnonymous" Type="Boolean"
            Nullable="false" />
  <Property Name="IsApproved" Type="Boolean"
            Nullable="false" Default="true" />
  <Property Name="IsLockedOut" Type="Boolean"
            Nullable="false" Default="false" />
  <Property Name="CreateDate" Type="DateTime"
            Nullable="false" />
  <Property Name="LastLoginDate" Type="DateTime"
            Nullable="false" />
</ComplexType>
```

Ami után az entitás definíciója már egyértelmű, hiszen csak a kigyűjtött adatokat kell egy ManagementType típusúval helyettesíteni.

```
<EntityType Name="User">
  <Key>
    <PropertyRef Name="Id" />
  </Key>
  <Property Name="Id" Type="Guid" Nullable="false" />
  <Property Name="Management" Type=
      "Self.ManagementType" Nullable="false" />
  <Property Name="Name" Type="String" Nullable="false"
      MaxLength="256" />
  <Property Name="Email" Type="String" Nullable="true"
      MaxLength="256" />
  <Property Name="Comment" Type="String"
      Nullable="true" MaxLength="3000" />
</EntityType>
```

Azért figyeljük meg a Type attribútumot! A típusunk neve névtérrel van feltüntetve. Ez a névtér elvileg az, amit a Schema elemben definiáltunk – EFSampleModel. Használhatnánk azt is, de ha visszanézzük a Schema definíciót, akkor látunk ott egy Alias attribútumot, ami pont azt a célt szolgálja, hogy házon belül egyszerűbben írhassuk ki a névteret. Így került oda a Self.

Az ehhez tartozó MSL-oldal már egy kicsit trükkösebb, mert két táblázatból fogtunk össze adatokat.

```
<EntityTypeMapping
      TypeName="IsTypeOf(EFSampleModel.User)">
 <MappingFragment StoreEntitySet="aspnet_Users">
   <ScalarProperty Name="Id" ColumnName="UserId"
         />
   <ScalarProperty Name="Name"
         ColumnName="UserName" />
    <ComplexProperty Name="Management"
         TypeName="EFSampleModel.
         ManagementType">
      <ScalarProperty Name="LastActivityDate"
            ColumnName="LastActivityDate"/>
      <ScalarProperty Name="IsAnonymous"
            ColumnName="IsAnonymous"/>
    </ComplexProperty>
 </MappingFragment>
 <MappingFragment StoreEntitySet=
            "aspnet_Membership">
   <ScalarProperty Name="Id" ColumnName="UserId"
            />
   ...
    <ComplexProperty Name="Management"
            TypeName="EFSampleModel.
            ManagementType">
      <ScalarProperty Name="IsApproved"
            ColumnName="IsApproved"/>
      <ScalarProperty Name="IsLockedOut"
            ColumnName="IsLockedOut"/>
      <ScalarProperty Name="CreateDate"
            ColumnName="CreateDate" />
      <ScalarProperty Name="LastLoginDate"
            ColumnName="LastLoginDate" />
    </ComplexProperty>
 </MappingFragment>
</EntityTypeMapping>
```

Ennek eredménye például a CreateDate elérésének módjában valahogy így mutatkozik majd meg:

```
EFSampleModel.EFSampleEntities entities = new
EFSampleModel.EFSampleEntities();

foreach (EFSampleModel.User u in entities.Users)
{
   Console.WriteLine(u.Management.CreateDate);
}
```

Kapcsolatok

Természetesen az adatbázisban a tábláink, üzleti modellünkben pedig az entitásaink nincsenek elszigetelve egymástól. Sőt, talán az egyik legfontosabb tulajdonsága mindkét oldalnak, hogy kapcsolatot tud teremteni adathalmazaink közt, szép, logikus adatstruktúrákat teremtve ezzel. Ezeket a kapcsolatokat adatbázisszinten az idegenkulcsok segítségével hozzuk létre. Objektumok esetében viszont referenciákkal. A két technika erős különbségeket mutat még nedves félhomályban, ködlámpa nélkül is. Mindjárt a legszembetűnőbb, hogy adatbázisnál a gyerekek hivatkoznak a szülőtáblákra, objektumok esetében ez inkább fordítva van. És mivel egy gyermekre így több szülő is hivatkozhat, roppant egyszerűen előállhat a több–több kapcsolat. Relációs adatbázisoknál ez a fajta kapcsolat nem tudja hiányolni a kapcsolótáblát. Mivel ez sarkalatos pont egy ilyen rendszer használhatóságának megítélésekor, az ismerkedéshez nézzük meg erre az EF megoldását a CSDL-oldallal kezdve.

```
<Association Name="UsersInRoles">
 <End Role="RolesEnd" Type="EFSampleModel.Role"
       Multiplicity="*" />
 <End Role="UsersEnd" Type="EFSampleModel.User"
       Multiplicity="*" />
</Association>
```

Ezzel az egy Association elemmel meghatároztunk egy több–több kapcsolatot, ami a két végpont számosságából (Multiplicity=) látszik. A kapcsolatban részt vevő két végpontnak adtunk nevet (Role=), és meghatároztuk az adott oldalon helyet foglaló entitás típusát (Type=). Azzal, hogy a két entitásunk közötti kapcsolatot definiáltuk, még nem tudjuk kódból elérni ezt a kapcsolatot. Ehhez szükségünk van mindkét típusban a másik típussal feltölthető listatulajdonságra. Ezt valósíthatjuk meg a NavigationProperty elem segítségével az entitásaink definíciójában.

```
<EntityType Name="User">
 ...
 <NavigationProperty Name="Roles"
       Relationship="EFSampleModel.UsersInRoles"
            FromRole="UsersEnd" ToRole="RolesEnd"
/>
</EntityType>
```

Hasonló módon a Role entitásban:

```
<EntityType Name="Role">
 ...
 <NavigationProperty Name="Users" Relationship=
       "EFSampleModel.UsersInRoles"
            FromRole="RolesEnd"
                 ToRole="UsersEnd" />
</EntityType>
```

Két nevet adtunk meg, az egyik a property neve (Name=), amelyen keresztül elérhetjük a kapcsolat túloldalán lévő objektumokat, a másik pedig a hivatkozott kapcsolat neve (Relationship=). Mivel mindkét oldalon ugyanazt a kapcsolatot használjuk, meg kell határoznunk az irányt. Erre szolgál a FromRole és a ToRole attribútum, amelyek a kapcsolatdefinícióban meghatározott végpontneveket kaphatják értékül.

Ezeknek a definícióknak az eredményeképpen fogunk tudni a következő módon navigálni a kapcsolatainkon.

```
EFSampleModel.User user;
EFSampleModel.Role role;

...

foreach (EFSampleModel.Role userrole in user.Roles)
{
   Console.WriteLine(userrole.RoleName);
}

foreach (EFSampleModel.User roleuser in role.Users)
{
   Console.WriteLine(roleuser.Name);
}
```

Sajnos azonban ez a kapcsolat jelen állapotában még eléggé légből kapott.

Először is egy Association-példány csak két entitáspéldányt köt össze, tehát az entitáslistáinkhoz hasonlóan definiálni kell a kapcsolatlistát is, a már ismert helyen, az EntityContainer elem alatt.

```
<EntityContainer Name="EFSampleEntities">
 <EntitySet Name="Roles" EntityType=
       "EFSampleModel.Role" />
 <EntitySet Name="Users" EntityType=
       "EFSampleModel.User" />
 <AssociationSet Name="UsersInRolesSet"
       Association="EFSampleModel.UsersInRoles">
  <End Role="RolesEnd" EntitySet="Roles" />
  <End Role="UsersEnd" EntitySet="Users" />
 </AssociationSet>
</EntityContainer>
```

Itt a végpontok nem entitások, hanem entitásliták.

Nem határoztuk meg továbbá, hogy a kapcsolatot hogyan lehet kiolvasni az adatbázisból. Hiányzik tehát a középső réteg. Nézzük meg, mi a teendőnk az MSL-ben.

```
<AssociationSetMapping Name="UsersInRolesSet"
TypeName="EFSampleModel.UsersInRoles"
StoreEntitySet="aspnet_UsersInRoles">
 <EndProperty Name="RolesEnd">
  <ScalarProperty Name="Id" ColumnName="RoleId" />
 </EndProperty>
 <EndProperty Name="UsersEnd">
  <ScalarProperty Name="Id" ColumnName="UserId" />
 </EndProperty>
</AssociationSetMapping>
```

A párosítás most is setekre vonatkozik. Meghatározzuk, hogy melyik kapcsolattípust (TypeName=) melyik adatbázistáblából szeretnénk kiolvasni. Az adatbázisban a kapcsolatok idegenkulcs-oszlop formájában jelennek meg. Egy kapcsolat beolvasása tehát ennek az idegenkulcs-adatnak a megszerzését jelenti. Több–több kapcsolat esetén ezek az idegenkulcsok ki vannak emelve egy kapcsolótáblába. Ezt a kapcsolótáblát mutatjuk meg a StoreEntitySet attribútummal. Az első EndProperty az AssociationSet RolesEnd végén meghatározott Roles nevű EntitySet Id oszlopához rendeli a fizikai rétegben (StoreEntitySet=) található aspnet_UsersInRoles tábla RoleId oszlopát. A következő EndProperty hasonló módon a Users nevű EntitySethez kapcsolja a tábla UserId oszlopát.

Fizikai modell (SSDL*)

Az említett XML-hármasból eddig csak kettőbe sikerült bepillantást nyernünk. A harmadik sem kevésbé fontos, mégis csak a végére hagytam egy kis összefoglalót róla. Az eddigiek alapján viszonylag könnyen követhető a felépítése. A sémában definiált névtéren belül találhatók az adatlistáink (EntitySet) definíciói, amelyeket relációs adatbázis esetében egyszerűen a táblázatoknak feleltethetünk meg (Name=).

```
<Schema Namespace="EFSampleModel.Store"
        Alias="Self" xmlns="...">
<EntityContainer Name="dbo">
 <EntitySet Name="aspnet_Roles"
        EntityType="EFSampleModel.Store.sRoles" />
 <EntitySet Name="aspnet_Users"
        EntityType="EFSampleModel.Store.sUsers" />
 <EntitySet Name="aspnet_UsersInRoles"
        EntityType="EFSampleModel.Store.sUsersInRoles" />
 ...
```

Másrészt itt találhatók a kapcsolatvégpont-definíciók.

```
<AssociationSet Name="FK_UiR_RolesSet"
      Association="EFSampleModel.Store.FK_UiR_Roles">
 <End Role="FKRolesEnd" EntitySet="aspnet_Roles" />
 <End Role="FKUsersRolesEnd"
       EntitySet="aspnet_UsersInRoles" />
</AssociationSet>
```

Aztán az EntityContainer lezárása után maguk a táblasémák, vagy ha úgy tetszik, a tábláink egyedeit alkotó típusok definíciói, amelyekre a fenti EntitySet-elemek EntityType attribútumai hivatkoznak.

```
<EntityType Name="sUsersInRoles">
 <Key>
  <PropertyRef Name="UserId" />
  <PropertyRef Name="RoleId" />
 </Key>
 <Property Name="UserId" Type="uniqueidentifier"
       Nullable="false" />
 <Property Name="RoleId" Type="uniqueidentifier"
       Nullable="false" />
</EntityType>
```

No és persze a kapcsolatok definíciói, amelyek az SSDL-ben az adatbázis idegenkulcsainak megfelelői. A koncepcionális modellben található Association-elemek nem ezekre, hanem közvetlenül a tábla oszlopaira fordítódnak le. Erre itt az adatbázissal való kommunikáció miatt van szükség.

```
<Association Name="FK_UiR_Roles">
 <End Role="FKRolesEnd" Type=
       "EFSampleModel.Store.sRoles" Multiplicity="1" />
 <End Role="FKUsersRolesEnd" Type=
       "EFSampleModel.Store.sUsersInRoles"
       Multiplicity="*" />
 <ReferentialConstraint>
  <Principal Role="FKRolesEnd">
   <PropertyRef Name="RoleId" />
  </Principal>
  <Dependent Role="FKUsersInRolesEnd">
   <PropertyRef Name="RoleId" />
  </Dependent>
 </ReferentialConstraint>
</Association>
```

Ezzel a harmadik XML-lel teljessé vált a kép. A gyakorlatban ezek egységet alkotnak, és egy fájlban is találhatók (némi designer információval együtt). Ez az EDMX file.

Infrastruktúrátlanítás (vagy fordítva?)

Ezt a rengeteg XML-t nemcsak az írás örömért szerkesztettük (a Visual Studióval), ez az, amiből a kódgenerátor az alkalmazásunkban könnyedén felhasználható kódot, osztálydefiníciókat generál. Ezek is, mint minden generált osztály, partial classként kerülnek a kódba. Ezek fogják tartalmazni adatainkat, és ezeket tudjuk használni LINQ to Entities lekérdezéseinkben.

A generált kód specializál továbbá pár infrastrukturális osztályt, hogy a saját környezetünkben kényelmesen és biztonságosan használhassuk azokat. Ezek az osztályok számos társukkal alkotják az Object Servicest. Implementációik a System.Data.Objects és a System.Data.Objects.DataClasses névterekben találhatók. Közülük a legfontosabb, központi osztályunk az ObjectContext. Ez felelős az adatforráshoz való kapcsolódásért, a metaadatokért, melyek leírják modelljeinket, továbbá a változáskövetésért.

A designer támogatása

Természetesen létezik grafikus felület is az EDMX előállítására. S az is, hogy egy ilyen grafikus felület nem használja ki az összes lehetőséget – főleg, hogy még csak CTP változatban érhető el[3] – de a leggyakoribb helyzetek megoldásában mindenképpen segítségünkre van. Pakolhatnék ide szép képeket a varázslóból, de nem teszem. Azt remélem, hogy sikerült elég részletes leírást adnom az alapelvekről ahhoz, hogy a varázsló feliratai közt könnyedén elnavigáljunk. Vész esetén pedig jobbklikk az edmx-en és Open With...

Tóth László
(tocsi@netacademia.net)
NetAcademia

* Store Schema Definition Language

[1] Martin Fowler – Patterns of Enterprise Application Architecture

[2] http://www.microsoft.com/downloads/details.aspx?FamilyId=15DB9989-1621-444D-9B18-D1A04A21B519&displaylang=en

[3] http://www.microsoft.com/downloads/details.aspx?FamilyId=D8AE4404-8E05-41FC-94C8-C73D9E238F82&displaylang=en

MENTSÜK, AMI MENTHETŐ!

Minden cikknek, így ennek is van egy kis előélete. A cikk megírása előtt villámgyors adatgyűjtésbe kezdtem, és már az első félóra után látszott, hogy nemcsak a rendelkezésre álló néhány oldalt, hanem akár egy egész lapszámot meg lehetne tölteni a mentés-helyreállítás témaköréhez tartozó cikkekkel.

A fent leírtak még akkor is igazak, ha csak az SQL-adatbázisok és a SharePoint-farmok mentésére koncentrálok. A cikk megírása előtt villámgyors adatgyűjtésbe kezdtem. Így – kompromisszumokat keresve – kötöttem ki az alább következő két témánál.

SQL backup – for dummies

Elnézést kérek a szerverkatasztrófákon és helyreállításokon edződött adatbázis- és SharePoint-gazda kollégáktól. Az első fejezet nem nekik szól, kérem jó szándékú megértésüket. Inkább azok számára adnék – igazából csak futó – áttekintést az SQL- és a SharePoint-mentés szépségeiről, akik éppen most kezdenek ismerkedni a technológiával, és a legtöbb esetben szingli rendszergazdaként az összes többi rendszer mellett kell ezeket a nem kis komplexitású, ugyanakkor az üzletmenet szempontjából egyre inkább kritikus rendszereket egészségesen tartaniuk. Az első fejezet tehát azoknak szól, akik az alábbi kérdések közül legalább az egyikre nemmel válaszolnak. (A válaszadás önkéntes, nem reprezentatív, és senki sem ellenőrzi, tessék tehát őszintének lenni.)

- Pontosan tudja-e, hogy mit, milyen rendszerességgel és hogyan kell mentenie egy SQL-adatbázis vagy egy SharePoint-kiszolgáló(farm) esetén?
- Van-e kidolgozott módszere a felhasználói hibából eredő helyreállítási igények kezelésére (mint például dokumentumok törlése egy SharePoint-dokumentumtárból)?

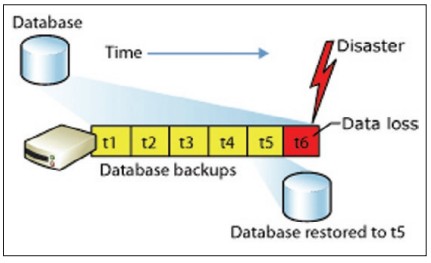
A jó hír az, hogy – a forradalmi változások előjelei ellenére – adataink továbbra is fájlok formájában öltenek testet a merevlemezen, és nagyrészt ebben a formájukban menthetők is. Igaz ez az adatbázisokkal dolgozó rendszerekre is, mint az Exchange, az SQL Server és a SharePoint, de e rendszerek esetén sokféle megfontolás miatt kerülőutat kell tennünk, ha megbízható mentést szeretnénk készíteni. Hogy csak a két legnyomósabb érvet említsem: az adatkonzisztencia biztosítása és a felhasználóknak nyújtott teljesítmény fenntartása. Az Exchange egy külön világ – bár mentési szempontból sok hasonlóságot mutat az SQL Server mentésével – így koncentráljunk most az SQL-adatbázisokra. Mint látni fogjuk, a SharePoint annyival árnyalja majd tovább a képet, hogy ott adatbázisok és fájlok sajátos együttese alkotja a konzisztens állapotot, amely alkalmas lehet egy katasztrófahelyzetből történő visszaállításra.

Az adatbázismentésekkel kapcsolatban először a mentési-visszaállítási stratégia és a recovery modell fogalmával kell megismerkednünk, mert ezek döntően meghatározzák lehetőségeinket. A mentési-visszaállítási stratégia egy elvi (és pénzügyi) döntés, amellyel igyekszünk maximalizálni az adatok rendelkezésre állását és minimalizálni az adatvesztés kockázatát. Egyszerűen belátható, hogy egy bizonyos ponton túl az adatok rendelkezésre állása nem növelhető tovább az adatvesztés kockázata nélkül (például a mentési ablak nem rövidíthető a mentés lefutásához technológiailag szükséges idő alá). A képlet már így sem egyszerű, de zavaró tényezőként ide társul még harmadikként a költség, és ekkor már komoly mérlegelésre van szükség: mi a helyes arány az adatbázisrendszer költségei (beleértve a mentési rendszer költségét), az adatok rendelkezésre állása és az adatvesztés kockázata között. Itt a döntő szót minden bizonnyal az üzleti vezetők fogják kimondani, de igyekezzünk döntésüket megfelelő technológiai adatokkal és érvekkel előkészíteni, mert a kialakított rendszert a mi feladatunk lesz üzemeltetni, és rajtunk fogják számon kérni a vállalt paramétereket.

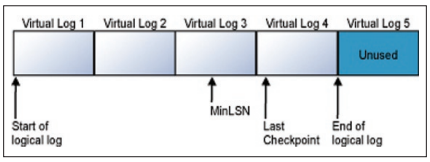
Visszatérve a technológiához: az SQL

Server adatbázisai esetén háromféle helyreállítási (recovery) modell közül választhatunk, ezek a simple, a full és a bulk logged. A simple modell alkalmazása esetén a mentés csak az adatbázisfájlt tartalmazza, a hozzá tartozó tranzakciós logokat nem, sőt ezek a logok időről időre (minden checkpointnál) csonkolódnak, hogy a logfájl ne töltse tele a merevlemezt. Így a visszatöltés lehetősége az utolsó mentett állapotra korlátozódik, a mentés utáni változások a tranzakciós logok csonkolása miatt elvesznek, és a visszatöltés után nem reprodukálhatók. Praktikusan tehát kisméretű és/vagy ritkábban változó adatbázisokon alkalmazhatjuk ezt a modellt, ezeket az adatbázisokat kellő gyakorisággal tudjuk lementeni ahhoz, hogy az adatvesztés kockázata ne nőjön magasra. Teljes és differenciális mentések váltogatásával és a gyakoriság jó beállításával már a simple modell alkalmazása esetén is az üzlet számára elfogadható mentési-visszaállítási stratégiát készíthetünk.

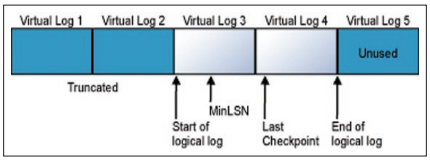
A full és bulk logged helyreállítási modellnél a tranzakciós log mentése előfeltétele annak, hogy a checkpoint felszabadítsa a már feldolgozott tranzakciók által elfoglalt helyet a logfájlban. A folyamat pontosabb megértéséhez nézzünk bele egy kicsit a logfájlba! A tranzakciós log az adatbázis szerves részeként, azzal egy időben jön létre, és az adatbázis konfigurációjától függően vele együtt növekedhet. A tranzakciós log belül kisebb, úgynevezett virtuális logfájlokra oszlik, ezeket a rendszer automatikusan kezeli: szükség esetén újabbakat hoz létre, illetve megfelelő feltételek esetén felszabadítja és újrahasznosítja őket. A tranzakciók sorában előrehaladva újabb és újabb virtuális logfájlok jönnek létre. Ha ezek a virtuális logok felemésztik a tranzakciós logfájl által előre lefoglalt lemezterületet, akkor (ha az adatbázis konfigurációja ezt megengedi) a tranzakciós log újabb területet foglal le a lemezen, amennyiben ez nem sikerül, akkor az adatbázist a konzisztencia megőrzése érdekében leállítja (akárcsak az Exchange).



1. ábra. Adatvesztés a simple recovery modell esetén



2. ábra. Tranzakciós log mentés előtt...



3. ábra. ...és után

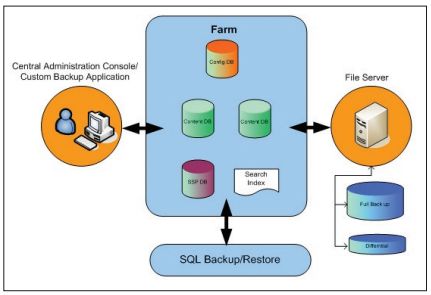
Vegyünk itt észre két pontot, ami üzemeltetési szempontból is fontos:

1. a tranzakciós logok által használt merevlemez szabad kapacitása – az adatbázist tartalmazó lemezekhez hasonlóan – szigorúan monitorozandó;

2. a tranzakciós logok töredezettekké válhatnak, ami a teljesítmény romlását okozhatja, a töredezettségmentesítést tehát tervszerűen el kell végezni!

Amikor egy full modell szerint működő adatbázis tranzakciós logját mentjük, akkor felhatalmazzuk a rendszert, hogy a következő checkpointra ráfutva mindazokat a virtuális logokat, amelyek csak sikeresen feldolgozott tranzakciókat tartalmaznak, és a mentésben szerepelnek, szabadítsa fel, és szükség esetén írja felül. Üzemeltetési szempontból azt nyerjük tehát a tranzakciós logok mentésével, hogy a logfájl által foglalt lemezterület kordában tarthatóvá válik. A mentések szempontjából viszont nagyon fontos, hogy az adatbázismentések és a logmentések konzisztensek legyenek. Csak akkor élvezhetjük ugyanis a modell előnyeit, és minimalizálhatjuk az adatvesztést, ha az adatbázismentésünk és valamennyi (!) logmentésünk sikeres, és ebből a tranzakciós logok visszaállításával egy folytonos tranzakció-sorozatot tudunk képezni. A logfájlokban ugyanis minden tranzakció rendelkezik egy azonosítóval (Log Sequence Number), és ha az LSN-számok folytonossága megszakad, akkor helyreállításkor a tranzakció továbbgörgetése már nem lehetséges (hiszen azt kockáztatjuk, hogy a hiányzó tranzakciók miatt szétesik az adatbázis egysége).

A bulk logged modellről most elegendő annyit megjegyeznünk, hogy használatát csak átmeneti időszakokra javasolják, olyankor, amikor az adatbázisokba nagy mennyiségű adatot töltenek fel, és a részletes logolás mellőzésével akarják javítani a feldolgozás sebességét. Ebben a módban csak a hagyományos tranzakciók logolása történik meg, a tömeges feldolgozás eseményei csak minimális mértékben kerülnek bele a logfájlba. Ebből következően az ilyen állapot visszaállítása rendkívül körülményes. A napi gyakorlatban jobban tesszük, ha a full modellt használjuk, és csak a tömeges feltöltés idejére változtatunk a recovery modellen (ha az feltétlenül szükséges).

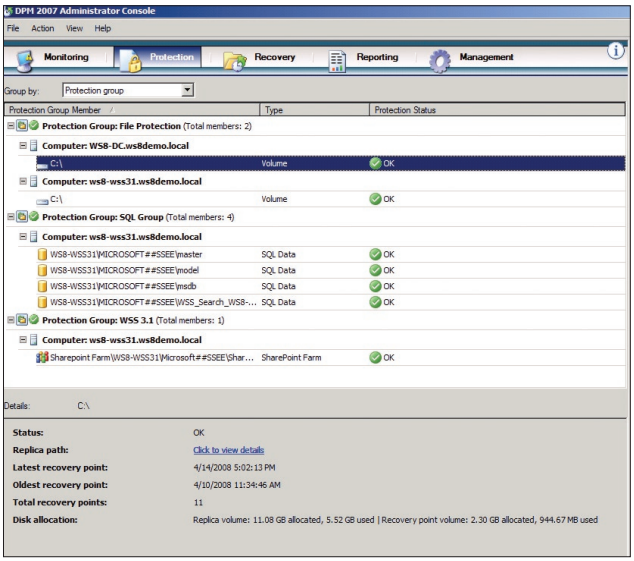


4. ábra. A mentendő SharePoint-komponensek

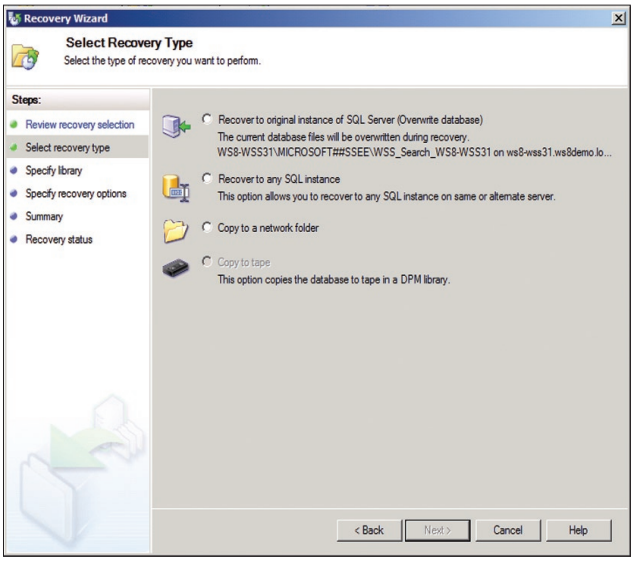
Elérési út	Tartalom
Program Files\Common Files\Microsoft Shared\ Web server extensions\12	Általánosan frissített fájlok, egyedi assemblyk, egyedi sablonok, egyedi site-definíciók.
Inetpub	IIS virtuális könyvtárak.
C:\WINNT\assembly	Global assembly cache (GAC). Egy védett operációsrendszer-terület, ahol a .NET Framework code assemblyk találhatók teljes rendszerjogosultságokkal.

Mint az előzőekből kiderült, a mentésekre három módunk kínálkozik, ami a simple modell esetén csak kettő: a teljes adatbázis mentése, az adatbázisról készített differenciá-

lis mentés és a tranzakciós logok mentése (ez utóbbi nem lehetséges a simple modell alkalmazásakor). A teljes és a differenciális mentés metódusa teljesen analóg a fájlmentéseknél alkalmazott azonos nevű eljárásokkal, így ezeket részletesen nem fejtem ki. A mentések



5. ábra. Amikor minden rendben van a mentésekkel



6. ábra. Bőséges lehetőség adatbázisok visszaállítására

tényleges megvalósítására pedig részletes segítséget találunk itt: Backing up and restoring databases in SQL Server http://go.microsoft.com/fwlink/?LinkID=102629&clcid=0x409.

Ha most visszatérünk az első kérdésünkhöz, akkor a hogyanról már lehet némi elképzelésünk, nézzük a gyakoriságot és a tartalmat. Ha csak az SQL-t nézzük, akkor az adatbázisainkat két kategóriába sorolhatjuk: vagy rendszeradatbázis vagy felhasználói adatbázis. Az előbbieket a rendszer a telepítéskor automatikusan létrehozza, és nagyrészt önállóan kezeli is. Az utóbbiakat az alkalmazásaink vagy alkalmazásgazdáink (sokszor mi magunk) hozzák létre, és mivel ezek hordozzák a cégünk szempontjából értéket képező adatokat, nagyobb figyelemmel kell foglalkoznunk velük. A felhasználói adatbázisok mentési gyakoriságát alapvetően az üzleti igények határozzák meg. Lehetségesek ritkán változó, tehát viszonylag statikus adatbázisok (például egy ügyféladatbázis), amelyeket elegendő akár hetente menteni (és mondjuk, naponként csak a tranzakciós logokat), míg egy termelési rendszer mögött állhat olyan adatbázis, ahol a logokat 15 percenként, az adatbázist magát pedig minden műszak végén menteni kell. Hogy ne veszítsük el a mentések fonalát a harmadik napon, járható út lehet, ha készítünk két-három mentési sablont az adatbázisaink mentésére, ami egységes módon határozza meg a teljes, a differenciális és logmentések váltakozását kis, közepes és nagy terhelésű adatbázisokra, igazodva az üzleti igényekhez. És ezzel el is érkeztünk ahhoz, amit az SQL terminológia Maintenance Plannek nevez, és használatáról bőven olvashatunk a fentebb már idézett URL-en.

A rendszeradatbázisokról is essék azért még néhány szó, mert teljes szerverkatasztrófa esetén bizony ezekre is szükség van a visszaálláshoz. A három legfontosabb adatbázis a master, a model és az msdb adatbázis. A master, mint neve is sugallja, az adatbázisok adatbázisa, egyszerűen szólva ez tartalmazza az összes kiszolgálószintű beállítást, mentése tehát elengedhetetlen. Ez az adatbázis simple recovery modellt követ, és csak teljes mentést támogat. Mindenképpen készítsünk róla rendszeres mentéseket, sőt minden alkalommal mentsük, ha rendszerbeállításokat módosítunk, vagy adatbázisaink száma változik.

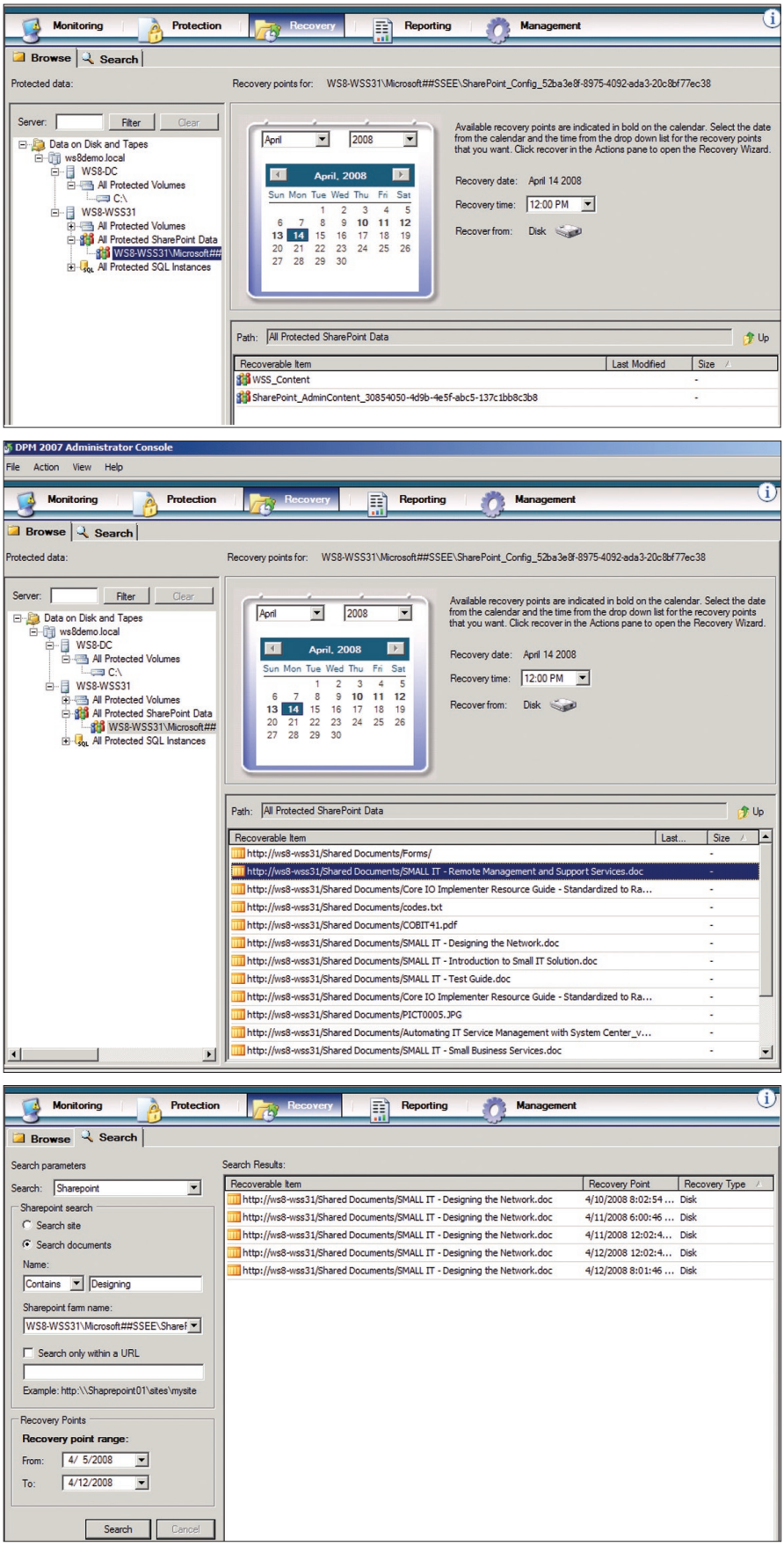
Hasonlóképpen járjunk el a model és msdb adatbázisokkal: az előbbi tartalmazza az alapértelmezett adatbázissémákat (és például hotfixek, szervizcsomagok is frissíthetik), az utóbbi a SQL Server Agent tevékenységeinek tárháza (például itt őrződik a mentések története). A recovery modell konfigurálható, de nem vétünk nagyot, ha a simple modellnél maradunk, csak végezzük rendszeresen a mentéseket.

Nem szükséges mentenünk a Resource és tempdb adatbázisokat, ezek nem hordoznak pótolhatatlan információkat. Ha létezik, akkor mindenképpen mentsük a distribution adatbázist (menthetjük a többi rendszeradatbázissal egy csomagban), de ez az adatbázis csak akkor létezik, ha a kiszolgálónk disztribútorként vesz részt egy adatbázis-replikációs folyamatban (tehát ne rémüljünk meg, ha nem találjuk).

Mennyiben más a SharePoint mentése?

Sajnos elég sokban, de megfelelő tervszerűséggel a dolog még kezelhető. Amit nagyon fontos látnunk, hogy sokkal szerteágazóbb információkra van szükségünk, mint egy „egyszerű” adatbázismentésnél. Amint az a *4. ábrán* látható, a SharePoint rendszerünkben (legyen az egy egykiszolgálós SharePoint Services vagy nagyvállalati MOSS 2007 farm) az adatokat több helyről kell összegyűjtenünk a mentéshez: a konfigurációs adatokat a ConfigDB, a feltöltött fájlokat a tartalom-adatbázisok (ContentDB) tartalmazzák, van ezen felül Shared Services Provider (SSP) adatbázisunk és keresési adatbázisunk; fontos, hogy mentsük a SharePoint binárisait (fájlszinten), a testre szabott tartalmakat és az IIS beállításait – ebben teszi könnyebbé a dolgunkat az IIS7, ahol már XML-fájlok tartalmazzák a szükséges adatokat.

Az SQL-adatbázisokkal viszonylag egyszerű dolgunk van, mentsünk minden tartalom-adatbázist és az SSP-adatbázist. Mentsük



7. ábra. SharePoint-tartalmak helyreállítási lehetőségei

a konfigurációs adatbázist is, de jó, ha tudjuk, hogy ezt csak akkor lehet helyreállításra használni, ha teljes kiszolgáló-katasztrófából állunk talpra és a kiszolgáló minden paramétere pontosan megegyezik az eredetiével (név, website-ok konfigurációja, adatbázis-instance, adatbázisnév, elérési utak) – betűről betűre minden. A keresési adatbázis közös életet él a hozzátartozó indexfájllal, így konzisztens módon csak a SharePoint belső eszközeivel menthető. Emiatt két megoldást választhatunk: vagy a belső eszközökkel mentjük, vagy lementjük adatbázisként, és helyreállítás esetén számolunk azzal, hogy az indexfájl teljes egészében újraépül.

Fontos, hogy ne csak mentésekkel rendelkezzünk erről a rengetegféle dologról, hanem legyen írásos konfigurációs leírásunk is (erősen javasolt tehát a szigorú változáskezelés), mert katasztrófa esetén sok olyan beállítás van, amit manuálisan, a dokumentációból kell megadnunk, és a helyreállítás sikere múlik rajta.

Milyen eszközeink lehetnek a SharePoint mentésére? Először is van egy beépített grafikus eszköz, amelyet a központi adminisztrációs felületen keresztül érhetünk el. Ezzel a teljes farmunkat, annak tetszőleges site-ját és bármelyik adatbázisát menthetjük. Ez az eszköz képes konzisztens mentést készíteni a keresési adatbázisról is, szinkronizálva azt az indexfájllal. Nagy hátránya viszont, hogy nem időzíthető, és az adminisztrátor közvetlen beavatkozását igényli. Az időzítést a parancssori stsadm.exe használatával tudjuk megvalósítani. Funkcionalitásában ez az eszköz mindent tud, amit a grafikus változat (sőt adminisztratív feladatokban többet is), a paraméterezése viszont lényegesen nehezebb. Mindenképpen szükségünk lesz fájlszintű mentésekre is, ezért semmiképpen ne hagyjuk ki a 38. oldalon lévő táblázatban található mappákat a rendszeres mentésekből.

Remélem, mostanra már mindenkiben motoszkál a kézenfekvő kérdés.

Lehetne ezt egyszerűbben?

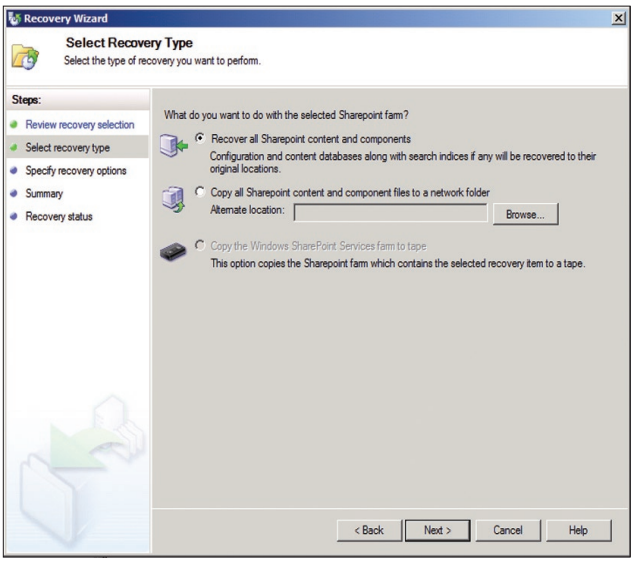
A válasz természetesen: igen. A System Center Data Protection Manager 2007 az alapvető fájlszintű mentéseken túl összetett alkalmazásszintű mentéseket is képes készíteni Exchange rendszerekről, SQL-kiszolgálók adatbázisairól, SharePoint-farmokról és Virtual Server 2005-kiszolgálókról, mindezt

pedig egyszerűen kezelhető felületen keresztül, pilótavizsga nélkül.

Mielőtt az egyszerűségre mutatnék példákat, nézzünk egy kicsit mélyebbre a folyamatokban, hogy megértsük, mi is zajlik a háttérben. Az alkalmazásszintű mentések minden esetben az árnyékmásolat-technológiára épülnek (Volume Shadow Copy). Ahhoz, hogy használni tudjuk a DPM-et az SQL- és SharePoint-kiszolgálók mentésére, engedélyeznünk kell az SQL VSS Writer és SharePoint VSS Writer szolgáltatásokat. (Ezek alapértelmezésben kézi indításra vannak állítva, tegyük tehát őket automatikusan indulókká!) A megfelelő mentési szabályok (Protection Group) létrehozásakor az adatbázisokról egy teljes másolat készül a DPM-kiszolgálón és – néhány különleges esetet kivéve – nem is készül több, hanem egy fájlszűrő segítségével a DPM innentől kezdve követi, hogy az adott adatbázisok mely diszkblokkokat foglalják el, és ezek közül melyik változott. A változásokat egy Express Full Backup nevű eljárás keretében veszi át az adatbázis-kiszolgálóról és rögzíti a saját diszkjein. (Érdemes tehát az Express Full Backup gyakoriságát az adatbázisunk változási gyakoriságához igazítani.) Ezenfelül az általunk megadott időközönként a tranzakciós logokról is készül egy mentés, itt a legrövidebb megadható időpont 15 perc.

A két metódus együtt alkalmas arra, hogy, az adatbázis bármelyik köztes állapotát előállítsuk, és akár egy megosztáson elhelyezzük, vagy egy másik adatbázis-kiszolgálón felcsatoljuk. Ha tudjuk, hogy maximum 512 árnyékmásolatunk lehet (tehát ennyi diszkblokk-változáscsomagot kezel a rendszer), akkor az ezekhez tartozó 15 percenkénti logmentéssel akár 344 ezer helyreállítási pontot is képezhetünk. (Óránként 4 logmentés × 24 óra × 7 nap × 512) És ez még csak a diszk alapú mentés, ami a közvetlen helyreállítást segíti. A DPM-kiszolgálóhoz csatlakozó szalagos eszközre az általunk megadott időközönként készíthetünk szintetikus mentést (amikor a DPM összedolgozza az adatbázisok aktuális állapotát egy menthető, konzisztens állapotá), amit aztán hosszabb ideig tárolhatunk, letétbe tehetünk, attól függően, hogy mi a mentésekre vonatkozó üzleti, jogi elvárás.

Lássunk akkor néhány képet, hogy ráérezzünk a dolog egyszerűségére! Az első képernyőn *(5. ábra)* azt a pillanatot láthatjuk, amikor valamennyi mentendő adatunkról – legyen az fájl vagy alkalmazás – rendelkezésre áll egy konzisztens (értsd: visszatölthető) állapot.



8. ábra. A SharePoint-adatok visszaállítása sem ördöngösség

A következő képen *(6. ábra)* már egy megkezdett SQL-adatbázis-visszaállítás látható. A négy választási lehetőség azt mutatja, milyen egyszerűen helyreállíthatjuk az adatbázist: az eredeti vagy egy alternatív kiszolgálón, vagy egy megosztáson, vagy akár közvetlenül szalagra. Az utóbbi opció nem képzavar, remekül használható például akkor, ha egy könyvelési adatbázist a péntek déli zárást követő állapotában kell archiválni. (A képen csak szalagos egység hiányában szürke az opció.)

A következő képsor *(7. ábra)* a SharePoint-tartalmak helyreállítási lehetőségeit mutatja: az első képen még csak azt látjuk, milyen tartalmak állnak rendelkezésre, a másodikon már egy konkrét dokumentumot választhatunk ki egy dokumentumtárban, míg a harmadik kép azt mutatja be, hogy adott dokumentumra akár rá is kereshetünk, és láthatjuk, hogy egyáltalán milyen mentett példányokkal rendelkezünk belőle.

A SharePoint esetén is rendelkezésünkre állnak alternatívák a visszatöltésre, noha a *8. ábra* éppen nem az összes lehetőséget mutatja. Hiányzik róla az alternatív visszatöltés egy másik farmba (ami történetesen éppen a visszaállítást szolgálja, és mondjuk csak virtuális kiszolgálón fut), mert a tesztrendszer nem lát másik SharePoint-kiszolgálót (tehát egyetlen másik gépen sem fut SharePoint VSS Writer). Ha lenne ilyenünk, akár az egész farm tartalmát reprodukálhatnánk a másik kiszolgálón.

A téma irodalma

- Backing up and restoring databases in SQL Server http://go.microsoft.com/fwlink/?LinkID=102629&clcid=0x409
- SQL Server 2008 Books Online http://msdn2.microsoft.com/en-us/library/bb543165(sql.100).aspx
- Balássy György blogja: SharePoint backup Powershellel http://www.msdnkk.hu/Article.aspx?Id=a62397dd-ce5f-dc11-8db3-0007e9ef0d89
- Plan for backup and recovery (Office SharePoint Server) (http://go.microsoft.com/fwlink/?LinkId=102799&clcid=0x409)
- Administering backup and recovery for Office SharePoint Server 2007 (http://go.microsoft.com/fwlink/?LinkID=102627&clcid=0x409)
- Using the Microsoft SharePoint Server 2007 backup tool (http://searchexchange.techtarget.com/general/0,295582,sid43_gci1275045,00.html)
- Data protection and recovery for Microsoft Office SharePoint Server 2007 in small to medium-sized deployments http://office.microsoft.com/download/afile.aspx?AssetID=AM102447701033
- System Center Data Protection Manager 2007 TechCenter (http://go.microsoft.com/fwlink/?LinkId=102807&clcid=0x409).
- How to Recover a Windows SharePoint Services Item (http://go.microsoft.com/fwlink/?LinkId=102815&clcid=0x409)
- How to Recover a Windows SharePoint Services Site (http://go.microsoft.com/fwlink/?LinkId=102826&clcid=0x409)
- How to Recover a Windows SharePoint Services Farm (http://go.microsoft.com/fwlink/?LinkId=102831&clcid=0x409)

Somogyi Csaba
(Csaba.Somogyi@microsoft.com)
Microsoft Magyarország

Te jó ég!

Csak nem egy újabb parancssori környezet?

Sok informatikus első találkozása a PowerShell-lel az Exchange Server 2007 kapcsán történt meg, nálam is ez a helyzet. Hiszen míg a Windows XP, Windows Server 2003, Windows Vista, sőt még a Windows Server 2008 esetében is ez a komponens opcionális, nem kötelező alkalmazni, addig az Exchange Server 2007 üzemeltetése során nem lehet megkerülni a használatát.

Ennek a cikknek nem az a célja, hogy elmélyedjen az Exchange Server 2007 részleteiben, így azok is bátran elolvashatják, akik azt nem ismerik. A célom inkább a PowerShell bővítési lehetőségeinek bemutatása az Exchange példáján keresztül, illetve azoknak az eltéréseknek, újdonságoknak a kiemelése az „alap” PowerShellhez képest, amelyek – véleményem szerint – képesek előrevetíteni a PowerShell továbbfejlesztésének irányait is.

Első ránézésre

Ha az Exchange Server 2007 programcsoportban meglátjuk az Exchange Management Shellt, az elnevezés alapján megijedhetünk, hogy ez egy újabb parancssori környezet, de szerencsére nem erről van szó, ez is a PowerShell.

1. ábra. Az Exchange Management Shell valójában egy testre szabott PowerShell

Ha rákattintunk az ikonra, akkor egy PowerShell-ablakot kapunk, de nem teljesen ugyanolyat, mint az „igazi” PowerShell esetében. És hogy milyen ez az ablak?

Ez az ablak alaphelyzetben kicsit „csúnya”, fekete a háttere, ellentétben a szép mélykék „alap” PowerShell-lel, nincs bekapcsolva a Quick Edit üzemmód, pici az ablak puffermérete, így csak kevés sort őriz meg. De természetesen ezeket a hiányosságokat nyugodtan orvosolhatjuk az ablak tulajdonságainak átállításával.

A másik különbség az „igazi” PowerShell-ablakhoz képest, hogy az ablak fejlécében az ablakot futtató gép neve és az Active Directory valamely ágának megnevezése látható, valamint alaphelyzetben be van töltve az Exchange snap-in, amit le is kérdezhetünk (a nem erre vonatkozó részeket helytakarékossági okokból kivágtam, és csak pontozással jelöltem):

```
[PS] C:\>Get-PSSnapin

...
Name        : Microsoft.Exchange.Management.PowerShell.Admin
Description : Admin Tasks for the Exchange Server PSVersion   : 1.0
```

Minden más tekintetben ez egy „normális” PowerShell-ablak, tehát mindaz alkalmazható, használható benne, amit a PowerShellben amúgy megszoktunk. Minek köszönhetők ezek a változások az alap PowerShell-konzolhoz képest? Ha megnézzük a fenti parancsikon mögötti parancssort, akkor ezt láthatjuk:

```
C:\WINDOWS\system32\windowspowershell\v1.0\
powershell.exe -PSConsoleFile „E:\Program Files\
Microsoft\Exchange Server\bin\exshell.psc1” -noexit
-command „. 'E:\Program Files\Microsoft\Exchange
Server\bin\Exchange.ps1'”
```

Nem egy rövid parancs, a lényege egy konzolfájl és egy szkript közvetlen meghívása a PowerShell-környezet indítása során. Nézzük meg ezt a két állományt! Elsőként a konzolfájlt:

```
<?xml version=”1.0” encoding=”utf-8”?>
<PSConsoleFile ConsoleSchemaVersion=”1.0”>
  <PSVersion>1.0</PSVersion>
  <PSSnapIns>
    <PSSnapIn Name=”Microsoft.Exchange.
Management.PowerShell.Admin” />
  </PSSnapIns>
</PSConsoleFile>
```

Itt töltődik be az Exchange Server rendszerkezeléssel kapcsolatos cmdleteket tartalmazó beépülő modul, vagy más néven snap-in. Nézzük a szkriptet (csak a fontosabb részeit):

```
# Copyright (c) Microsoft Corporation. All rights reserved.

...

## PROMPT ###############
## PowerShell can support very rich prompts, this
##simple one prints the current working directory and
## updates the console window title to show the
##machine name and directory.

function prompt
{
    $cwd = (get-location).Path
    $scope = „View Entire Forest”
    if (!$AdminSessionADSettings.ViewEntireForest)
    {
      $scope = $AdminSessionADSettings.DefaultScope
    }
    $host.UI.RawUI.WindowTitle = „Machine: „ +
$(hostname) + „ | Scope: „ + $scope
    $host.UI.Write(„Yellow”, $host.UI.RawUI.
BackGroundColor, „[PS]”)
    „ $cwd>”
}

## FUNCTIONs #############
## returns all defined functions
...
## only returns exchange commands function get-
excommand
{
    if ($args[0] -eq $null)
    {
      get-command -pssnapin Microsoft.Exchange*
    }
    else
    {
      get-command $args[0] | where { $_.psSnapin
-ilike 'Microsoft.Exchange*' }
    }
}
...
```

Ebben a szkriptben láthatjuk néhány változó definiálását, a prompt és az ablak fejlécének beállítását, néhány függvény definiálását. Érdekes módon az Exchange snap-in által definiált cmdletek kilistázását végző get-excommand nem cmdlet, hanem függvény! Ebből következően esetében például nem működik a tab kiegészítés, és a get-help-pel sem lehet segítséget kérni a használatával kapcsolatban.

Az Exchange-cmdletek feltérképezése

Nem egyszerű az ilyen Exchange snap-in által biztosított új cmdletek áttekintése. Elsőként nézzük meg, hány új cmdletünk van:

```
[PS] C:\>(get-command -PSSnapin „Microsoft.Exchange.
Management.PowerShell.Admin"
).Count
394
```

Nem kevés, 394 darab új cmdlet, az alap 129-cel szemben! Hogyan lehet ezeket áttekinteni? Szerencsére kapunk segítséget egyrészt a get-command, másrészt a get-help cmdlettől is. A get-command esetében használhatjuk a verb és a noun paramétert:

```
[PS] C:\>get-command -noun mailbox

CommandType   Name              Definition
-----------   ----              ----------
Cmdlet        Connect-Mailbox   Connect-Mailbox [-Identity]
...
Cmdlet        Disable-Mailbox   Disable-Mailbox [-Identity]
...
...

[PS] C:\>get-command -verb export

CommandType Name                Definition
----------- ----                ----------
Cmdlet      Export-ActiveSyncLog Export-ActiveSyncLog -
Filena...
Cmdlet      Export-Alias        Export-Alias [-Path]
<String...
...
```

Ha még ez sem lenne kellő segítség, akkor a get-help esetében használhatjuk a role, component és functionality paramétereket. Például a mailbox szerepű kiszolgálókon használható, szervezetszintű jogosultságok beállításával kapcsolatos cmdletek listáját az alábbi módon kérhetjük le:

```
[PS] C:\>get-help -role *mail* -component *permission* -
functionality global

Name                  Category  Synopsis
----                  --------  --------
Get-ADPermission      Cmdlet    Use the Get-ADPermissi...
Add-ADPermission      Cmdlet    Use the Add-ADPermissi...
Remove-ADPermission   Cmdlet    Use the Remove-ADPermi...
```

Felhasználó- és csoportkezelés

Az Exchange-rendszergazda tevékenységei közé a felhasználók és csoportok kezelése is beletartozik, így az Exchange PowerShell snap-in tartalmaz ezzel kapcsolatos cmdleteket is. Az alap PowerShellben az [ADSI] típusjelölővel lehet hivatkozni AD-objektumokra, az Exchange-környezetben még egyszerűbb a helyzet. Nem teljes értékű ugyan az AD felhasználómenedzsmentre szolgáló parancskészlete, hanem kifejezetten csak az Exchange Server rendszergazdáinak igényeire szabott, de azért így is sokat profitálhatunk belőle. Nézzük például a get-user cmdletet:

```
[PS] C:\>get-user brian

Name          RecipientType
----          -------------
Brian Cox     UserMailbox
```

Nézzük meg kicsit részletesebben Briant:

```
[PS] C:\>get-user brian | fl

IsSecurityPrincipal   : True
SamAccountName        : Brian
Sid                   : S-1-5-21-150787130-2833054149-
3883060369-1132
SidHistory            : {}
UserPrincipalName     : Brian@Adatum.com
...
DisplayName           : Brian Cox
...
Title                 :
...
DistinguishedName     : CN=Brian Cox,OU=Legal,DC=Adatum,
DC=com
Identity              : Adatum.com/Legal/Brian Cox
Guid                  : debe1f52-f0ea-4da0-b049-
8f22c2d45db2
ObjectCategory        : Adatum.com/Configuration/Schema/
Person
ObjectClass           : {top, person, organizationalPerson, user}
WhenChanged           : 2007.01.03. 19:50:42
WhenCreated           : 2007.01.03. 19:17:26
```

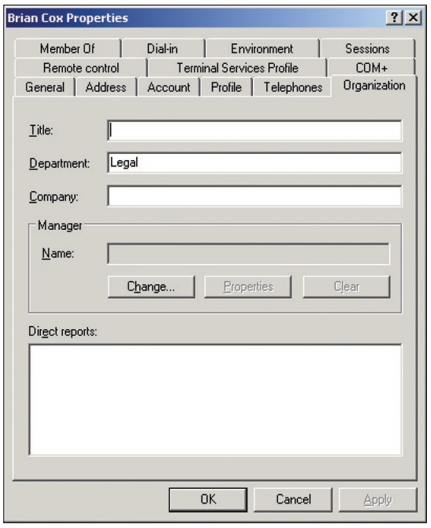
Nagyon sok tulajdonság lekérdezhető a felhasználói fiókokkal kapcsolatban. A további, Exchange-dzsel kapcsolatos attribútumot a get-mailbox cmdlettel lehet lekérdezni.

Get és Set

A fenti táblázatban a Title (beosztás) paraméter üres. Próbáljuk ezt feltölteni:

```
[PS] C:\>$u = Get-User brian
[PS] C:\>$u.title = 'Portás'
[PS] C:\>$u.title
Portás
```

Látszólag sikerült beállítani Brian beosztását, mint ahogy az utolsó, ellenőrző sorban látható, de ha az Active Directory Users and Computer eszközzel megnézzük Briant, akkor nem látjuk ezt a paramétert kitöltve:



2. ábra. Nem történt meg Brian adatainak módosítása

Mi történt akkor? Az AD-vel kapcsolatos Get-... kezdetű cmdletek nem direktben dolgoznak a címtárban, hanem a címtár-információkat betöltik a memóriába. Így az értékadás utasításom is a memóriában hajtódott végre és nem ténylegesen a felhasználói fiókon. Ezért van az AD-vel kapcsolatos Get-... cmdleteknek egy Set-... párjuk is, mert ez már ténylegesen beírja az értékeket a címtár adatbázisába. Nézzük meg, hogyan lehet ténylegesen címtárparamétert megváltoztatni:

```
[PS] C:\>Set-User brian -Title „Portás"
[PS] C:\>Get-User brian | fl Name, Title

Name : Brian Cox
Title : Portás
```

Vigyázni kell, hogy nem teljesen szimmetrikusak a Get-... és a Set-... cmdlet-párok. Jól mutatja ezt a csoportok kezelését végző Get-Group és a Set-Group páros:

```
[PS] C:\>get-group sales | fl

...
SamAccountName      : Sales
...
Members             : {Jeff Hay, Don Hall, Judy Lew}
...
```

A fenti példából látszik, hogy a csoport tagjai (Members) lekérdezhetők a get-group-pal. Nézzük meg a Set-Group szintaxisát:

```
 Set-Group
-Identity <GroupIdParameter>
[-Confirm [<SwitchParameter>]]
[-DisplayName <String>]
[-DomainController <Fqdn>]
[-IgnoreDefaultScope <SwitchParameter>]
[-ManagedBy <GeneralRecipientIdParameter>]
[-Name <String>]
[-Notes <String>]
[-PhoneticDisplayName <String>]
[-SimpleDisplayName <String>]
[-WhatIf [<SwitchParameter>]]
[-WindowsEmailAddress <SmtpAddress>]
[<CommonParameters>]
```

Itt viszont nem látunk a Members attribútumra vonatkozó paramétert. Mindez azt bizonyítja, hogy az Exchange-cmdletek nem próbálják meg magukra vállalni a teljes Active Directory kezelését, arra vagy az [ADSI] típushivatkozást, vagy külső gyártó snap-injét használhatjuk.

Új paraméterfajta: Identity

Az előző példákban elég nagyvonalúan hivatkoztam Brianre, semmi *distinguished name*, semmi *canonical name* nem szerepelt a hivatkozásban, holott például az [ADSI] használatánál teljes, pontos distinguished name kitöltésére van szükség. Mi teszi lehetővé ezt az egyszerű, kényelmes használatot? Nézzük meg a get-user súgóját:

```
[PS] C:\>get-help Get-User

...
SYNTAX
   get-User [-Identity <UserIdParameter>] [-Credential
<PSCredential>] [-DomainController <Fqdn>] [-Ignore
DefaultScope <SwitchParameter>] [-OrganizationalUnit <O
rganizationalUnitIdParameter>] [-ReadFromDomainController
<SwitchParameter>] [-RecipientTypeDetails <RecipientType
Details[]>] [-ResultSize <Unlimited>] [-SortBy <String>]
[<CommonParameters>]

   get-User [-Credential <PSCredential>] [-DomainController
<Fqdn>] [-Filter <String>] [-IgnoreDefaultScope <Switch
Parameter>] [-OrganizationalUnit <Organizational
UnitIdParameter>] [-ReadFromDomainController <Switch
Parameter>] [-RecipientTypeDetails <RecipientType
Details[]>] [-ResultSize <Unlimited>] [-SortBy <String>]
[<CommonParameters>]

   get-User [-Anr <String>] [-Credential <PSCredential>]
[-DomainController <Fqdn>] [-IgnoreDefaultScope <Switch
Parameter>] [-OrganizationalUnit <Organizational
UnitIdParameter>] [-ReadFromDomainController <Switch
Parameter>] [-RecipientTypeDetails <RecipientType
Details[]>] [-ResultSize <Unlimited>] [-SortBy <String>]
[<CommonParameters>]
...
PARAMETERS
   -Anr <String>
      The Anr parameter indicates that the argument will be
resolved using ambiguous name resolution (ANR).
...
   -Identity <UserIdParameter>
      The Identity parameter takes one of the following values:
      * GUID
      * Distinguished name (DN)
      * Domain\Account
      * User principal name (UPN)
      * Legacy Exchange DN
      * Simple Mail Transfer Protocol (SMTP) address
      * Alias
...
```

Látszik, hogy a többfajta szintaxisa alapján az első paramétert vagy ANR (Ambiguous Name Resolution) névfeloldásra, vagy – ehhez nagyon hasonló – Identity típusú névfeloldásra használja. Az Identity paraméter magyarázatát a fenti help-részletben láthatjuk is. Azt, hogy most melyiket használta, nem tudjuk, hiszen mindkettő jó eredményre vezet:

```
[PS] C:\>Get-User -anr brian

Name                  RecipientType
----                  -------------
Brian Cox             UserMailbox


[PS] C:\>Get-User -identity brian

Name                  RecipientType
----                  -------------
Brian Cox             UserMailbox
```

Mi ebből a tanulság? Hogy az Exchange-cmdletek kialakításánál törekedtek arra a fejlesztők, hogy a lehető legegyszerűbben lehessen hivatkozni az AD-objektumokra. Elég hamar elmenne a kedvünk a szkriptek írásától, ha minden esetben csak a teljes *distinguished name* kiírásával lehetne hivatkozni az AD-objektumokra, így azonban az Identity (és a felhasználókkal kapcsolatban az ANR) paraméterrel elég csak olyan mértékig utalni a keresett objektumokra, amikor már egyértelmű a cmdlet számára, hogy kire vagy mire gondoltunk.

Például a get-MailboxDatabase cmdletnél az Identity paraméter a következőket jelentheti:

```
   -Identity <DatabaseIdParameter>
      The Identity parameter specifies a mailbox database. You
can use the following values:
      * GUID
      * Distinguished name (DN)
      * Server\storage group\database name
      * Server\database name
      * Storage groupname\database name
      If you do not specify the server name, the cmdlet will
search for databases on the local server. If you have multiple
databases with the same name, the cmdlet will retrieve all data-
bases with the same name in the specified scope.
```

Azaz, a megjegyzés alapján, akár elég csak az adatbázis nevét megadni, ha az egyértelmű, és nem kell az adatbázis distinguished nevét használni, ami elég elrettentő lenne. Az alábbi példában lekérdezem ezt, a válasz három és fél sor!

```
[PS] C:\>(Get-MailboxDatabase „syd-dc1\mailbox database").
DistinguishedNameCN=Mailbox Database,CN=First Storage
Group,CN=InformationStore,CN=SYD-DC1,CN=Servers,CN
=Exchange Administrative Group (FYDIBOHF23SPDLT),CN
=Administrative Groups,CN=Adatum Organization,CN
=Microsoft Exchange,CN=Services,CN=Configuration,DC
=Adatum,DC=com
```

Filter paraméter

A másik hatékonyságnövelő paraméter számos Exchange-cmdletnél a Filter paraméter. Nézzük meg, hogy PowerShell-ismereteink alapján hogyan keresnénk meg az összes olyan felhasználónkat, akinek ki van töltve a beosztás tulajdonsága:

```
[PS] C:\>get-user | Where-Object {$ _.Title -ne „"}

Name                  RecipientType
----                  -------------
Brian Cox             UserMailbox
Kim Akers             UserMailbox
```

Hogyan hajtódik ez végre? A get-user cmdlettel az összes felhasználói objektumot kigyűjtjük, és maga a PowerShell-környezet a kliens oldalon válogatja ki közülük azokat, amelyeknél a Title paraméter nem üres sztring. Ez egy tízezres felhasználószámnál komoly feldolgozási munkát és memóriát igényel, és nagy hálózati forgalmat generál. Ezért kidolgoztak egy kiszolgálóoldali feldolgozási lehetőséget is az ilyen jellegű AD-cmdleteknél a Filter paraméter alkalmazásával.

Itt úgynevezett OPATH filtereket, „prescanned filter"-eket használhatunk, ami a szerveroldalon hajtódik végre:

```
[PS] C:\>Get-User -filter {Title -ne „"}

Name                  RecipientType
----                  -------------
Brian Cox             UserMailbox
Kim Akers             UserMailbox
```

Ennek köszönhetően jóval hatékonyabbá és gyorsabbá tehető a feldolgozás. A Filter paraméterben is használhatjuk a PowerShellben megszokott összehasonlító operátorokat és az ott alkalmazott szintaxist, azonban nem az „igazi" attribútumnevekkel hivatkozunk,

hanem a speciális OPATH-attribútumnevekkel.

Miért kell megtanulni „PowerShellül”?

Végezetül azt emelném ki, hogy miért kell egy rendszergazdának megtanulnia a PowerShellt. Az Exchange Server 2007 példája megmutatja számunkra, hogy a jövőben a Microsoft kiszolgálószoftverek grafikus kezelői felületére nem lesz minden kivezetve, csak a leggyakoribb feladatok elvégzéséhez szükséges elemek. Viszont a .Net Frameworkben minden felügyeleti objektum benne lesz, és konfigurálásukhoz szükséges PowerShell-cmdletek rendelkezésre fognak állni.

Példaként nézzük meg, hogy melyek azok a legfontosabb tevékenységek, amelyeket nem lehet a grafikus felületen, az Exchange Management Console-on végrehajtani.

Postafiók-exportálás

A postafiókokkal kapcsolatos egyik legmélyrehatóbb rendszergazdai tevékenység azok tartalmának kiexportálása. A korábbi Exchange-verzióknál erre egy grafikus eszköz, az ExMerge állt rendelkezésre. Az Exchange Server 2007-ben ez az eszköz nem támogatott, helyette egy PowerShell-cmdlet, az Export-Mailbox áll rendelkezésünkre, amely sokkal több lehetőséget biztosít számunkra.

Például az export során szűrhetünk a tárgysorban, üzenettörzsben vagy a csatolmányban található szóra, címzettben vagy feladóban található karaktersorozatra, kizárhatunk bizonyos mappákat a postafiókon belül, kitörölhetjük az exportált tartalmat az eredeti postaládából. Mindezt egy szkriptbe ágyazva elég összetett feladatokat is elvégezhetünk. Az alábbi kétsoros PowerShell-kifejezéssel például a „Jogi osztály” összes munkatársának olyan üzeneteit exportálom az Ellenőr Elemér postafiókjának „Export” nevű mappájába, amely üzenetek tartalmazzák a „titkos” szót:

```
[PS] C:\>$dn= (get-group „Jogi osztály”).DistinguishedName
[PS] C:\>Get-Mailbox -filter {MemberOfGroup -eq $dn} |
Export-Mailbox -AllContentKeywords „titkos” -TargetMailbox
Administrator -TargetFolder Export
```

Erőforrás-naptárak kezelése

Újdonság az Exchange Server 2007-ben a kifejezetten erőforrás (például: tárgyaló, projektor stb.) céljaira létrehozott levelesládák létrehozásának lehetősége. Ezt megtehetjük a grafikus felületen, de az erőforrás naptárjának tulajdonságait már csak PowerShell-cmdletekkel állíthatjuk be. Nézzünk erre is egy példát! Létrehoztam egy tárgyalót, és beállítottam a Titkárnőt teljes jogkörrel. Ezenkívül azt szeretném, hogy a Főnök és a Titkárnő bármikor szervezhessen találkozót, a Beosztott pedig csak igényelhesse a tárgyalót, de azt jóvá kell hagynia a Titkárnőnek. Ezenkívül a Főnök bármikor, akár munkaidőn túl is lefoglalhassa a tárgyalót, de a Beosztott csak munkaidőben. Továbbá szeretném, ha a tárgyalónál látnák a munkatársak, hogy 10 fő fér el benne, és hogy hűtőszekrény és projektor is tartozik a felszereltséghez:

```
[PS] C:\>Set-MailboxCalendarSettings tárgyaló
-AutomateProcessing autoaccept
[PS] C:\>Set-MailboxCalendarSettings tárgyaló
-ResourceDelegates Titkárnő
[PS] C:\>Set-MailboxCalendarSettings tárgyaló
-RequestOutOfPolicy Főnök -Request
InPolicy Beosztott -AllBookInPolicy $false
[PS] C:\>Set-ResourceConfig -ResourcePropertySchema room/
projektor, room/hűtőszekrény
[PS] C:\>Set-Mailbox Tárgyaló -ResourceCapacity 10 -
ResourceCustom projektor, hűtőszekrény
```

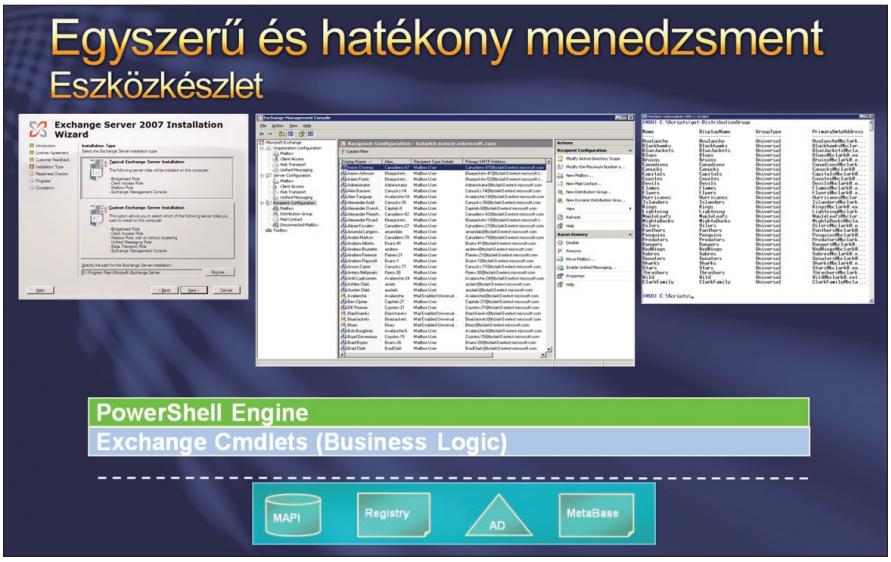
Az első három sort csak a jobb olvashatóság miatt írtam külön. A grafikus felületen csak az utolsó sornak megfelelő beállítást lehetett volna elvégezni, de azt meg kell előznie a negyedik kifejezésnek. Ebben a példában igazából nem volt PowerShell-szinten nagy trükk, de ha bonyolultabb lenne azoknak a személyeknek a köre, akik delegáltjai vagy igénylői az erőforrásoknak, ráadásul nem is egy, hanem több erőforrásra akarjuk ezeket beállítani, akkor nem árt megint csak PowerShellül tudni.

Standby Continuous Replication

Az egyik legösszetettebb és legfelelősségteljesebb rendszergazdai tevékenység a rendszer meghibásodásakor a minél gyorsabb és minél kevesebb adatveszteség nélküli helyreállítás. Az Exchange Server 2007 erre a célra az SP1 javítócsomaggal egy nagyon praktikus, új lehetőséget kínál, a *Standby Continuous Replication*t, amelynek során egy „tartalék” kiszolgálóra töltjük át folyamatosan az adatbázisunk tranzakciós naplóit, és ezen a tartalék kiszolgálón építjük folyamatosan az „árnyék” adatbázist. Ha az elsődleges kiszolgálónk tönkremegy, akkor ezen a tartalék kiszolgálón be tudjuk üzemelni az árnyékpéldányt. Ez nem fürtözési technológia, azaz nem automatikus az átállás, előnye azonban, hogy sokkal egyszerűbb a beállítása, és nem igényel speciális hardvert. Az egész módszert azonban csak PowerShell-cmdletek segítségével tudjuk elindítani, a grafikus felületen nem is látszik ebből semmi.

Nézzük tömören a lépéseket. A *syd-dc1* mester gépen, az *scr-sg* tároló csoport *scr-db* adatbázisából szeretnék egy másolati példányt a *syd-ex2* kiszolgálóra. Az SCR replikáció elindításához, a forrásgépen, a „mester” példánynál kiadjuk a következő parancsot:

```
[PS] C:\>Enable-StorageGroupCopy syd-dc1\scr-sg
-StandbyMachine syd-ex2 -ReplayLagTime 0.0:1:0
```



Az Exchange Server 2007 számtalan eszközzel felügyelhető

A legvégén található idő paraméter a tranzakciós logfájlok visszajátszásának késleltetését állítottam be 1 percre, mert az alapbeállítás 1 nap, ami tesztelés esetében kicsit hosszú.

Ha jön a meghibásodás, és esetleg nem kapcsolódott volna le a meghibásodott adatbázis, akkor állítsuk le:

```
[PS] C:\>Dismount-Database syd-dc1\scr-db

Confirm
Are you sure you want to perform this action?
Dismounting database „syd-dc1\scr-db".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?]
Help
(default is „Y"):y
```

Ezután következik az SCR másolati gépre az adatbázis helyreállítása, azaz a másolati példány „beélesítése":

```
[PS] C:\>Restore-StorageGroupCopy syd-dc1\scr-sg
-StandbyMachine syd-ex2
```

Miután az SCR másolati gépen nincs definiálva ennek az adatbázisnak a helye „hivatalosan", az így épített adatbázis azonnal nem használható, így az SCR másolati példányt egy nem használt, „igazi" mailbox-adatbázis helyett kell bekötni:

```
[PS] C:\>Move-StorageGroupPath „syd-ex2\First
Storage Group" -SystemFolderPath c:\scrdb
-LogFolderPath C:\scrdb -ConfigurationOnly

[PS] C:\>Move-DatabasePath „syd-ex2\Mailbox Database"
-EdbFilePath C:\scrdb\scr-db.edb -ConfigurationOnly
```

Ezután a helyreállítást engedélyezni kell, hiszen az eredeti adatbázis SCR-árnyékra cserélését a rendszer helyreállításként értelmezi:

```
[PS] C:\>Set-MailboxDatabase „syd-ex2\Mailbox
Database" -AllowFileRestore $true
```

Szükségessé válhat itt az SCR-célgépen az adatbázis javítása, mert a tranzakciós naplófájlok nevének más lehet az előtagja a két SCR-gép között.

Jelen esetben a mestergépen E02 az előtag, az SCR-másolaton meg E00:

```
C:\scrdb>eseutil -r E02

Extensible Storage Engine Utilities for Microsoft(R)
Exchange Server
Version 08.01
Copyright (C) Microsoft Corporation. All Rights Reserved.
Initiating RECOVERY mode...
    Logfile base name: E02
            Log files: <current directory>
        System files: <current directory>
Performing soft recovery...
                Restore Status (% complete)

  0  10  20  30  40  50  60  70  80  90  100
  |----|----|----|----|----|----|----|----|----|----|
  ..................................................

Operation completed successfully in 33.929 seconds.
```

Ezután az adatbázist már elindíthatjuk:

```
[PS] C:\>Mount-Database „syd-ex2\Mailbox Database"
```

Most már csak a felhasználók mailbox-attribútumát kell átirányítani az új adatbázisra:

```
[PS] C:\>Get-Mailbox -database syd-dc1\scr-
db |where {$ _ .ObjectClass -NotMatch '
(SystemAttendantMailbox | ExOleDbSystemMailbox)'}|
Move-Mailbox -TargetDatabase „syd-ex2\Mailbox
Database" -ConfigurationOnly
```

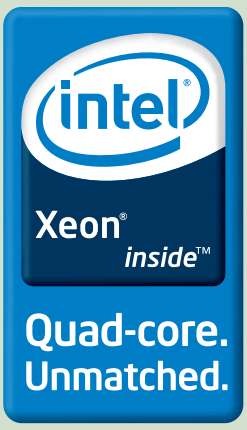
Ezzel készen is vagyunk. Ezt az utolsó kifejezést az teszi bonyolulttá, hogy a rendszer által használt speciális mailboxokat nem szabad átirányítani.

Soós Tibor
(soost@iqjb.hu)
MCT, IQSOFT – John Bryce Oktatóközpont

A Fujitsu Siemens Computers áttörő IT infrastruktúra-megoldásai éppoly rugalmasan alkalmazkodnak a váratlan helyzetekhez, mint Ön.

A Fujitsu Siemens Computers FlexFrame™ Infrastructure rendszere tökéletesen illeszkedik az adatközponti környezetekbe. Az egyes szoftverfejlesztők által támogatott rugalmas IT-platform dinamikusan rendeli hozzá a szervererőforrásokat az egyes alkalmazásokhoz. A nagy teljesítményű, **négymagos Intel® Xeon® processzorral ellátott PRIMERGY RX300** szerverekre épülő megoldás minden speciális igényt kielégít az alkalmazások támogatása terén. Hiba esetén az érintett alkalmazások dinamikus áthelyezésével tartja fenn a zavartalan működést. Miért is jó ez Önnek? Mert a kivételes rugalmasságnak köszönhetően végre minden energiájával az üzleti működésre koncentrálhat. És ez csak egy a Fujitsu Siemens Computers Dinamikus Adatközponthoz készült innovatív megoldásai közül!

www.fujitsu-siemens.hu



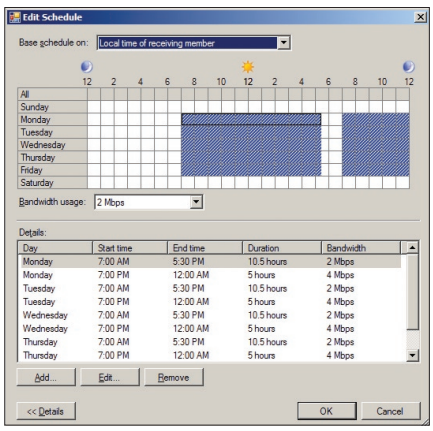
A Distributed File System

Az adatok elhelyezésének és szervezésének számtalan módja van. Tárolhatjuk őket adatbázisban, floppylemezen, pendrive-on, publikálhatjuk webes tartalomként, vagy elmenthetjük szalagra. Most mégis maradjunk az irodai környezetben talán leghagyományosabbnak mondható módszernél: helyezzük el adatainkat a fájlszerveren, és tegyük ott elérhetővé a felhasználók számára.

Voltaképpen persze nem túl izgalmas a történet. A hálózati felhasználó kitallózza magának a megfelelő kiszolgálót, rálel a szerencsés esetben beszédes nevű megosztásra, ott a mögöttes könyvtárstruktúrában biztos kézzel kattintgatva eljut a tizenhatodik alkönyvtárba – és a kívánt fájl már meg is van. Nincs itt semmi hiba, a rendszer gazdája időről időre elmagyarázza az újonnan belépő alkalmazottaknak, hogy mit hol találnak. Igen ám! De a cég növekszik, egyre több a felhasználó, egyre több a napi használatú adat, és ha lehunyjuk a szemünket egy-két évre, egy olyan infrastruktúrát látunk ébredéskor, amelyben több tucat szerver több telephelyen tárolja felhasználók százainak vagy akár ezreinek adatait.

Ebbe a környezetbe képzeljük bele Béla bácsit és Juliska nénit mint a rendszerrel épp most ismerkedő felhasználókat. Rendszergazda jön, nagy levegőt vesz, és elkezdi hosszasan sorolni, hogy az adott munkakörhöz tartozó állományok mely *szerverEK* melyik *megosztásAIBAN*, mely *alkönyvtárAKban* találhatók. Míg Béla bácsi pislogva memorizál, és csak három perc múlva adja fel, addig Juliska néni már az első perc után könnyes szemmel papírzsebkendő után kutat... Nem beszélve arról, hogy egy olyan telephelyen fognak dolgozni, amelyik az adatközponttal csak egy jóféle 512 kbit/másodperces kapcsolattal bír. Míg egy-egy fájl megnyitása közben Béla bácsi átfuthatja a teljes sportrovatot, addig Juliska néni a társkeresőt olvashatja el. Főleg, ha megtalálják a szükséges fájlt. Feltéve, hogy akarnak ilyen környezetben dolgozni egyáltalán.

Itt kap szerepet az a szolgáltatás, amelyet névről ismerhetünk már a Windows NT-s korszak óta, bár ott addendumként illeszthettük a rendszerhez, a Windows 2000-ben már a telepítőcsomag része: a DFS, azaz a Distributed File System. Ezt a DFS-t újították meg, egészítették ki és tették nagyon sok meglévő problémára megoldásként felhasználhatóvá a Windows Server 2008-ban. Mit tudott és mit tud most a DFS?



1. ábra. A DFS replikációjának időzítése

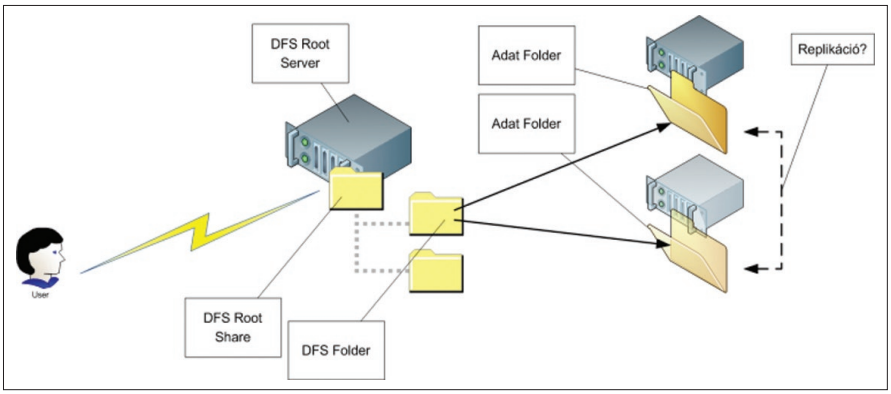
DFS az előző Windows-verziókban

Standalone DFS. Noha ez az írás nem a múltba igyekszik tekinteni, nézzük meg, hogy milyen környezetet alakíthattunk ki DFS-sel régebben. A probléma adott: sok szerveren, sok megosztásban, bonyolult könyvtárstruktúrában sokak adatai laknak, valószínűleg a legjobban csoportosítva. A legjobban, de nem biztos, hogy mindenki számára megfelelően. Ebben az esetben a DFS – mondhatni tradicionálisan – segítségünkre lehet. Azoknak a felhasználóknak, akiknek nem megfelelő a jelenlegi adatszervezés, mutassunk egy olyan adatstruktúrát, ami nem valós ugyan, de mutatókat, pointereket tartalmaz a valós adathelyekre. Ideális megoldás, hiszen az adatot nem kell elmozdítani a je-

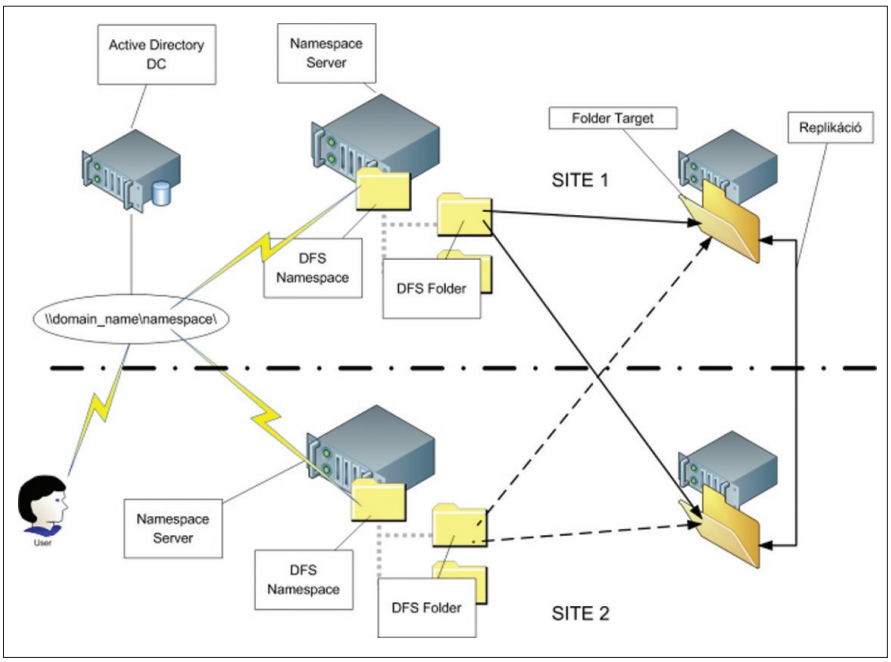
lenlegi helyéről (sokaknak pont úgy jó, ahogy van), de láttathatjuk úgy, mint ha átcsoportosítottuk volna azokat.

Hogyan is néz ki ez a gyakorlatban? Legyen A és B a két tényleges fájlszerver. Mindkettőnek számtalan megosztása van, köztük A-nak Dokumentumok1 és B-nek Dokumentumok2. Egy adott felhasználó kénytelen tudatosan csatlakozni mindkét szerverhez, sőt annak konkrét share-jeihez és az általa használt Dokumentumokat erről a két helyről megnyitni. Ha szeretnénk számára leegyszerűsíteni az elérést, vegyük elő a Z szerver, jelen esetben az A szerver Dokumentumok1 és a B szerver Dokumentumok2 megosztásaira mutatnak.

Tetszőleges mennyiségű pointert helyezhetünk el a DFS root alatt, amelyek a valós célhelyekre mutatnak. Ennyi lenne a DFS? Nos, tulajdonképpen ennyi a ***Standalone DFS***-változat, annyi kiegészítéssel, hogy a pointerek kialakításakor megadhatunk alternatívákat is a cél tekintetében, azaz ha a C szerver Dokumentumok1 megosztása ugyanazzal az adattartalommal bír, mint az A szerver Dokumentumok1-e, akkor a hibatűrés és a terhelésmegosztás jegyében a DFS root alatti „könyvtár" mindkét helyre egyaránt mutat. Ezt a DFS a kliensgép számára listaként, mintegy folyamatosan változó sorrendiséggel közvetíti, tehát nagyszámú kliens esetén azok kb. egyenlő arányban oszlanak meg a tényleges célhelyek között.



2. ábra. A Standalone DFS architektúrája



3. ábra. Az Active Directory integrált DFS architektúrája

vert, és jelöljük ki DFS rootnak! A DFS root egy olyan „megosztott könyvtár", amelynek alkönyvtárai nem (feltétlenül) valósak, csak

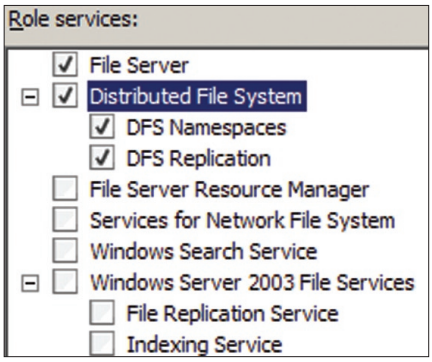
Itt jegyzendő meg, hogy a DFS root által felkínált „virtuális" könyvtárstruktúrára kattintva nem a DFS-kiszolgáló keresi fel a valós célt, hogy leplezze a csalást! A kliensgép DFS-ügyfélkomponensének üzen, amelyik felkeresi a tényleges adathelyet. Telepítettünk mi ilyet? Nem, de ez bizony szerényen megbújó része a Windowsnak ősidők óta.

Igen ám, de mi a helyzet a módosított Dokumentumokkal az azonos tartalmú célhelyeken? Ha az A szerver Dokumentumok1-ében módosítunk, az eljut valahogy a C szerver Dokumentumok1-ébe? Sajnos nem. Standalone DFS esetén (hangsúlyozom, hogy nem a Windows szerver 2008-ban található DFS-ről van szó!), a célhelyek csak olvasható Dokumentumokkal legyenek tele, vagy változások, módosítások esetén gondoskodjunk mi a konzisztenciáról. Jó, de hogyan? Akárhogyan – mondja a régi DFS – ez már nem rám tartozik. Hát... köszi. Igazán kedves...

Domain-Based, azaz Active Directory-integrált DFS. Amikor megjelent a Windows 2000 és az Active Directory, a DFS sokat változott – előnyére. Bár a Standalone-változat megmaradt, a Domain-Based sokat ígért, és azt be is váltotta. Root-replikát alakíthattunk ki, azaz ha a DFS rootunk valamiért kiesett, fordulhattunk egy másikhoz, ahol ugyanaz a „virtuális" struktúra fogadott minket a valós célhelyekre mutató pointerekkel. Nagy előrelépés! Főleg ha azt is számításba vettük, hogy nem is konkrétan a DFS-szerverre kellett hivatkozni, ha a rootot akartuk elérni. Az AD felkínált nekünk egy olyan belépési pontot, amely egy igen furcsa UNC path: \\domainnév\dfsrootnév.

Az erre tett hivatkozáskor az AD szerepe nem ért véget, hiszen a kliens IP-címéből és a belépési pont mögötti root-replikák IP-címeiből a DNS-sel karöltve könnyen irányíthatta a felhasználót egy olyan DFS roothoz, ami a klienssel azonos Site-on van. Azonos tartalmú DFS rootok különböző telephelyeken? Bizony, de ha az adott telephelyen lévő root nem érhető el, akkor még mindig ott a távoli – igaz, sokkal lassabban. Komoly változást hozott a Domain-Based DFS a célhelyek alternatíváinak kezelésében is: a File Replication Service, azaz FRS (NTFRS) segítségével meg

lehetett tartani a különböző fizikai szerverek azonos adattartalmú területeinek konzisztenciáját. Hogy ez mennyire volt szoros,



4. ábra. A DFS a Fájlkiszolgáló-szerepkör egy szolgáltatása

az már az FRS-en múlt, mindenesetre tette a dolgát. Mit tett a DFS, ha ezek az azonos tartalmú célhelyek különböző telephelyen voltak? Akárcsak a root kiválasztásakor, itt is az AD-ban definiált Site-struktúra döntött: a klienst – lehetőség szerint – a vele azonos telephelyen lévő cél felé irányította.

Hát ez már egész jó, nem? Kell ennél több? Dehogy! Elkezdtük használni, és... és elég sokat bajlódtunk vele. No nem a DFS-sel magával, az rendben működött, hanem az a fránya FRS elég sok borsot tört az orrunk alá! Akár a root-replikáknál, de főleg a célhelyek közti replikációnál misztikus hibák fordulhattak elő, amelyekre sok esetben a „net stop ntfrs – net start ntfrs" volt a megoldás. Gond volt a teljesítménnyel és a kommunikáció megbízhatóságával is, az FRS nem különösebben modern módszerekkel dolgozik. És az időzíthetőség? Ha nem szeretném, hogy hétköznap napközben az FRS foglalja a telephelyeket összekötő vonal amúgy is szűkös áteresztőképességét? Vagy esetleg pont azt szeretném, hogy napközben biztosítsa a legszorosabb konzisztenciát? Igényként merült fel gyakran az is, hogy „de jó lenne DFS nélkül jól összereplikálni két szerveren adatterületeket, no és ha már úgyis itt van az FRS, nem lehetne esetleg vele?"

DFS a Windows Server 2008-ban

A fent említett problémákra és igényekre mind van megoldás, itt, a Windows Server 2008-ban. Vegyük át a legfontosabb újításokat szépen sorjában! A DFS a Windows Server 2008 egy szerepköre (role), ami ennek megfelelően telepítendő, bár első ránézésre nem is látszik, hiszen a File kiszolgáló-szerepkör része. Amit már a telepítés fázisában észrevehetünk, az az, hogy a klasszikus DFS-funkcionalitás és a Replikáció különkülön is kiválaszthatók. Számomra ez a klasszikus DFS-konfiguráció nélküli fájlreplikáció lehetőségét jelenti, bár ez a Windows Server 2003 R2-ben volt újdonság. A DFS Managementben azután szemmel láthatóan külön kategóriát képviselnek a Névterek és a Replikáció.

Névterek, namespaces. Lássuk, milyen lehetőségeink vannak, ha DFS-t szeretnénk kialakítani! Az első lépés talán egy kis fogalom-újraértelmezés, hiszen az új DFS-ben sok dolgot nem úgy hívnak, mint régen (lásd a táblázatot).

Windows 2000; Windows 2003	Windows Server 2008
DFS Root	= Namespace, azaz névtér
DFS Root Szerver	= Namespace-szerver
DFS Root Szerver replika	= Namespace-szerver
DFS Link	= Folder Target
DFS Link Replika	= Folder Target

1. táblázat. Minek mi a neve?

Az első lépés a névtér létrehozása, amelyben annak neve és legalább egy namespace-szerver megadása kötelező. Ezután következik a DFS típusa, tehát előttünk a döntéshelyzet.

Lehet hagyományos Standalone, amely nem sokban különbözik a „ősi" DFS-változattól, a varázsló a failover clusterrel való integrálhatóságra felhívja ugyan a figyelmet, de ennek lehetősége – igaz, nem látszott a DFS konzoljából, csak a Cluster adminisztrációs felületéből – régebben is adott volt. Persze a Clusterre ebben az esetben szükség lehet, elsősorban akkor, ha a Root szerverünket – elnézést, a Standalone Namespace szerverünket szeretnénk hibatűrővé tenni.

A másik lehetőség a Domain-Based névtér kialakítása, ami lehet „hagyományos" vagy Windows Server 2008 módú. Ez utóbbi egy jelentős újdonságot rejt, amelynek neve **Access Based Enumeration.**

Mit is jelent ez? Segítségével elérhetjük, hogy felhasználóink a megosztáson belül csak azokat a könyvtárakat lássák, amelyeknek az eléréséhez valamilyen szintű jogosultságuk van. Végre! Egy rendszergazda nap mint nap kénytelen magyarázkodni azoknak a felhasználóknak, akiknek nincs megfelelő jogok egy egyébként csábító nevű és feltehetőleg „roppant érdekes" tartalmú könyvtárhoz. Képzeljünk csak el egy, a publikus adatterületen, kizárólag a HR-csoport számára fenntartott „Tervezett Béremelések 2008-ban" nevű mappát! Hány sikertelen megnyitási kísérletet naplózhatnánk itt naponta? Ha ezt a mappát mindenképpen a közös elérésű területen kell tartani, hát nyissunk parancssort a namespace-szerveren, és gépeljük be a **„dfsutil property abde enable *domainnév**namespacenév*"** utasítást, és a probléma egy csapásra megszűnik! Ott van ugyan a könyvtár, de a DFS felől közelítő felhasználók nem látják. Amiről nem tudunk, az nem fáj... – tartja a bölcs mondás. Windows Server 2008 módú Domain-Based DFS-t akkor készíthetünk, ha a Namespace-szervereink Windows Server 2008-at futtatnak, és az Active Directory domainünk funkcionalitási szintjét is átkapcsoltuk Windows Server 2008-ra. Ez természetesen kizár minden régebbi operációs rendszert futtató tartományvezérlőt a domainből, a lépést éles környezetben jól gondoljuk meg!

Ott tartottunk, hogy megvan a Namespace és legalább egy Namespace-szerver is, jöhet a folderek kialakítása a névtér alatt! Az új folder nevének megadásával a felhasználó által látható könyvtárnevet, míg a Folder Target definiálásával a célterületet határozzuk meg. Már most megadhatunk több Folder Targetet, amelyek a Domain-Based DFS esetén replikációs kapcsolatba kerülnek. Standalone DFS alatt a Folder Targetek hagyományosan nem replikálhatók? Dehogynem, most már bármikor beállíthatunk ebben az esetben is replikációt, de ne vágjunk elébe a dolgoknak. Ha Domain-Based Namespace alatt készítjük a foldert és a hozzá tartozó Folder Targeteket, a DFS itt is figyelembe veszi az AD és a DNS segítségével a kliens fizikai helyét, sőt ezt lehetőségünk van úgy finomítani, hogy a kliens számára esetleg nem is engedélyezzük a távoli site-on való célterület elérését. Új lehetőség továbbá a failback beállítása: ha a kliensünk a közeli Folder Target elérhetetlensége miatt egy távoli telephelyen lévő cél elérésére kényszerült, biztosíthatjuk számára az automatikus visszatérést, amennyiben a helyi, azaz preferált Folder Target ismét munkába állt. Ne felejtsük el, hogy az ügyfélgép DFS-kliens

komponensét távirányítjuk, és ez cache-ben őrzi a tényleges célhelyeket. Lehetőségünk van a hivatkozások tárolási idejének módosítására is, amit akár a Namespace-en, akár az alá szervezett folder szintjén is megtehetünk.

Hol is tartja a DFS a konfigurációt? Természetesen az Active Directory adatbázisban. A névtér-szerver ebből időről időre másolatot készít, amit helyben, XML formátumban tárol. Alapértelmezés szerint a DFS metadata publikációja is a PDC Emulátor szerepkörű DC-re hárul, ezen azonban módosíthatunk, elérhetjük, hogy a DFS namespace-szerverünk ne az esetlegesen igen távoli PDC-t, hanem a legközelebbi DC-t faggassa a konfiguráció esetleges változásairól. Előbbit a konzisztenciára, utóbbit a skálázhatóságra optimalizált állapotnak hívjuk.

Végül még egy apróság: a keresés lehetősége. Akármennyire is magától értetődő a funkció, eddig nem volt. Most, íme, rendelkezésre áll. Abba azért belegondolhatunk, hogy a fájlrendszerbe irányuló kereséseink általában egy adott gép adataiban kutatnak. És a DFS beépített keresése? Az bizony végrehajtja több gépen is, ahogy a folderek és a folder targetek megkívánják. Viszont az is igaz, hogy „csak" a keresett könyvtár megtalálására való.

DFS-replikáció, DFS-R

Külön bekezdést kapott a DFS-replikáció és talán nem érdemtelenül. Eddig a Windows 2000-ben vagy a Windows Server 2003-ban egy legyintéssel elintézhettük volna: mi más intézhetné a replikációt, mint az FRS?! És most? Mi is? Hát a DFS! A Windows Server 2003 R2-ben és a Windows Server 2008-ban a DFS új szolgáltatással bővült, ez a DFS-replikáció. Mi lesz akkor az FRS-sel? Lassacskán felejtsük el, a DFS környezetében mindenképp. Egy fontos dolga az FRS-nek azért megmaradt, ez pedig az Active Directory Domain kontrollerek „SYSVOL" megosztásainak replikációja. Windows Server 2008-ban akár ezt a feladatot is átruházhatjuk a DFS-replikációra, bár nem kötelező.

Mitől jobb a DFS-replikáció, mint az FRS?

Ide egy olyan hosszú felsorolás kívánkozik, ami túlmutat az írás terjedelembeli lehetőségein, de a teljesség igénye nélkül:

- A DFS-R a DFS Management MMC konzolból konfigurálható.
- Replikációs csoportokba szervezhetők a közreműködők és az adatterületek.
- A replikációs topológiák több séma és akár az egyéni elgondolások alapján is kialakíthatók.
- Időzíthetőség és sávszélességre való optimalizálhatóság jellemzi.
- „Staging folder" a változott állományok átmeneti tárolására, a „last writer wins" szabály, amiből adódó konfliktusokra egy „ConflictAndDeleted" folder nyújtja a megoldást.
- A változások replikációja akár bitszintű lehet, a replikációs forgalom automatikusan tömörített.
- Nagymértékben leegyszerűsített recovery-mechanizmus, és van Standalone DFS-támogatás.

A valós lista ennél sokkal hosszabb, de így is belátható, hogy nem véletlenül került bele a DFS-R már a Windows Server 2003 R2-be is. Amiknek azonban igazán örülhetünk, azok a következő kérdésre adható válaszok:

Windows Server 2003 R2	Windows Server 2008
Multiple RPC calls	RPC Async Pipes
Synchronous inputs/outputs	Asynchronous I/Os
Buffered I/Os	Unbuffered I/Os
Normal Priority I/Os	Low Priority I/Os (terheléscsökkentés)
4 concurrent file downloads	16 concurrent file downloads

2. táblázat. A sebességnövekedésért felelős változtatások

Mitől jobb a Windows Server 2008 DFS-replikáció, mint a DFS-R a Windows Server 2003 R2-ben?

1. Tartalomfrissesség-vizsgálat (Content Freshness). Szükség lehet erre az új tulajdonságra akkor, ha a replikációban részt vevő szerverek egyike nagyon hosszú ideig van „távol", majd mikor ismét replikál, akkor már esetleg nem derül ki, hogy a rajta lévő adatok régen elavultak. Ha a Content Freshness nem lenne, a régi, elavult adatok a többi tagra visszaíródnának, illetve felülírhatnák az újabb változatokat.

2. A váratlan leállások kezelése. Az NTFS fájlrendszerben bekövetkező változások az esetek többségében először memóriában, átmeneti tárban helyezkednek el, és csak egy kis idő után íródnak le a fizikai lemezre. A DFS-R replikáció adatbázisának szempontjából viszont a változás már a diszkre írás előtt megtörténik. A köztes időben bekövetkező bármilyen katasztrófa – legyen az váratlan szerverleállás, vagy szabálytalan volume-dismount – a DFS-R adatbázis inkonzisztenciájához vezethet. Míg a Windows Server 2003 R2 esetén ez a komplett DFS-R adatbázis újraépítésével és így rengeteg idővel járt, az új változat ezt az adatbázis újraépítése nélkül, azaz sokkal gyorsabban intézi el.

3. Sokkal gyorsabb replikáció. A Windows Server 2008 DFS-R elsősorban ebben jeleskedik. Mérhetően gyorsabban szinkronizál mind kisebb, mind nagyobb fájlok esetén, köszönhető ez a tömörítés mellett a hatékonyabb sávszélesség-kihasználásnak.

4. Report-képzés tesztállomány replikációjával (Propagation Report). A Windows Server 2003 R2-ben is meglévő reportok mellé diagnosztikai célból bekerült report, ami nem a valós adatállományok, hanem egy tesztállomány replikációjával kapcsolatos értékeket mutatja meg.

5. Az azonnali replikáció forszírozásának lehetősége (Replicate Now). Egy replikációs csoportban kijelölt célterületek közti replikációt indítja el, függetlenül annak időzítésétől. (SYSVOL replikációnál nem működik.)

6. A Read Only Domain Controller (RODC) támogatása. Miután a DFS-replikációba a tartományvezérlők SYSVOL-megosztása is belekeverhető, ebből az RODC sem maradhat ki. Itt viszont meg kell felelni az Active Directoryban definiált szabálynak, miszerint az RODC-ről nem származhat semmiféle változás vissza a többi, írható-olvasható adatbázisú DC-re. Ez alól a SYSVOL sem kivétel. Amennyiben a SYSVOL replikációját az FRS helyett a DFS-R végzi, ezt annak is figyelembe kell vennie! Meg is teszi, viszont ez nem érinti az RODC esetleges, a SYSVOL-tól különböző DFS-replikációs feladatait. Ezekre az adatterületekre kimenő és bejövő replikációs kapcsolatokat egyaránt beállíthatunk. Egy pillanat! Mi is hangzott el már többször? A SYSVOL replikációjának lehetősége? Igen, ez az a téma, ami talán egy kicsit összetettebb annál, semmint hogy egy mondatban elintézzük. Nézzünk a körmére ennek az új lehetőségnek!

A SYSVOL és a DFS-replikáció kapcsolata

Mint ahogy már ebben a cikkben is többször szó volt róla, a tartományvezérlők SYSVOL

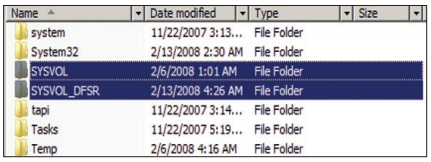
könyvtárainak egyezőségéről alapértelmezés szerint az FRS szolgáltatás gondoskodik, már a Windows 2000 óta. Nincs ez másként a Windows Server 2008-ban sem, viszont az FRS fölött sok szempontból eljárt az idő. Miért nem a DFS-R az alapértelmezés? Talán azért, mert a Windows Server 2008-as tartományvezérlőkből kialakított domain egyéb rendelkezés híján nem zárja ki az esetleges régebbi (2000; 2003) Domain Controllereket sem. Ezek viszont eléggé meglepődnének, ha a SYSVOL ettől kezdve DFS-R módszerrel érkezne... Ha a SYSVOL replikációját a DFS-R-re szeretnénk bízni, első lépésként emeljük a tartomány funkcionalitási szintjét Windows Server 2008-ra! Igen ám, de ettől még mindig marad az FRS, ez csupán a lehetőséget teremti meg az átállásra. Akkor telepítsünk a tartományvezérlőkre DFS-replikációt! Megvan? Így már majdnem kész vagyunk, de még mindig a régi módon replikálunk.

Hogyan tovább? Az FRS-ről DFS-R-re való áttérés egy migrációs folyamat, amelyet kellő körültekintéssel – és ami a legfontosabb – fokozatosan végezzünk! Megjelent a Windows Server 2008-ban egy parancssori segédeszköz, a **dfsrmig.exe.** A neve beszédes, két fő funkciója a migráció állapotának lekérdezése és változtatása. Vigyázzunk vele, mert használata minden tekintetben globális eredménnyel jár: bármelyik DC-n kiadjuk a megfelelő jogosultsággal, az egész tartományt érintő változtatást jelent. Tehát, amennyiben a fenti feltételeknek megfeleltünk, elkezdhetjük a migrációt. Gépeljük be a parancssorba a **„dfsrmig /getglobalstate”** utasítást, és láthatjuk, hogy a migráció státusza „start”. Ez még igazából semmit nem jelent, a SYSVOL-replikációt az FRS végzi. Milyen státuszok lehetségesek (lásd a táblázatot)?

0	START
1	PREPARED
2	REDIRECTED
3	ELIMINATED

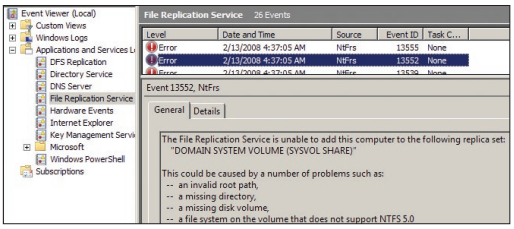
3. táblázat. Migrációs állapotok

A cél természetesen az „Eliminated” állapot elérése, de csak ésszel, óvatosan. Következhet a **„dfsrmig /setglobalstate 1”** utasítás, amelynek hatására a migráció állapota mostantól „prepared”. Most a DFS-R készít magának egy másolatot a SYSVOL mappáról, majd egy másik DC DFS-R szervizével megpróbál egy úgynevezett iniciális replikációt végrehajtani. Sikerült az összes tartományvezérlőn? Kérdezzük le a **„dfsrmig /getmigrationstate”** paranccsal! Tegyük fel, hogy a visszajelzés pozitív. Álljunk át a fent említett módszerrel a 2-es, azaz a „redirected” szintre! Mi is történik éppen? A tartományvezérlőkön futó DFS-R szervizek magukhoz ragadták a SYSVOL replikációjának feladatát, de az FRS még ugyanúgy teszi a dolgát, ahogy régen. Párhuzamosan működik mindkét rendszer! Gyors állapotlekérdezés, megnyugodhatunk, ha tényleg így van. Most következik a legfontosabb döntés: a harmadik szint, azaz az „eliminated” forszírozása. 1-es vagy 2-es szintről bármikor vissza lehet lépni, bűnbánóan az FRS elé állni és megkérni, hogy folytassa mégis csak ő a SYSVOL-replikációt – „eliminated” szintről nincs visszatérés! Ha az állapotjelzés **(„dfsrmig /getmigrationstate”)** kedvező, belevághatunk. Forszírozzuk a 3-as szintet, és lekérdezzük az állapotot: minden OK!



5. ábra. A régi és az új egymás mellett: migráció közben

Mi is történt? A DFS-R letörölte az eredeti SYSVOL-t és annak másolatával, a SYSVOL_DFSR könyvtárral operál a továbbiakban. Eseménynapló, a DFS log biztató bejegyzésekkel tele, hátradőlhetünk. Viszont, ha már az eseménynaplóban járunk, pillantsunk csak rá az FRS naplójára! A helyzet ijesztő, legalábbis az FRS nagyon meg van ijedve: nem leli a replikálandó SYSVOL mappát! Bár mi tudjuk, hogy ez nem igazán baj, de hogy lehetne ezt az FRS-nek is elmondani? Két lehetőség van, egyik sem túl elegáns: vagy leállítjuk és „manual” indítási módba tesszük az ntfrs-szervizt, vagy (gondolva arra, hogy talán valamikor valamiért még szükségünk lesz rá –nem tudnám ugyan megmondani, miért is lenne) a szerviz ideiglenes leállítása mellett kitöröljük a Jet adatbázisát a %systemroot%\ntfrs\jet alól. Ebben az esetben az újraindúló NTFRS létrehoz magának egy teljesen új adatbázist – persze üres, a SYSVOL-replikációs kényszerét nem tartalmazó változatot.



6. ábra. A megijedt FRS-szolgáltatás kiabál

A replikáció beállításai

Ha a replikációs csoportokat megvizsgáljuk a DFS Management konzolban, láthatjuk, hogy azok kialakítása roppant egyszerű – már ha nekünk kell egyáltalán kialakítani azokat. A Domain-Based DFS replikációs feladatai és a SYSVOL-replikáció, amennyiben migráltunk rá, automatikusan bekerülnek a konzolba. Igény szerint ezeknek a paramétereit, azaz topológiáját és időzítését, sőt a szereplőket és a replikálandó adatterületeket is megváltoztathatjuk. Kikényszeríthetjük az azonnali replikációt is, az időzítéstől függetlenül. Nem igaz ez a SYSVOL replikációjára, amit bár látunk a replikációs csoportok között, különösebben konfigurálni nem tudunk, de ez érthető.

Van egy probléma, amin azért el kell gondolkodni: az egyirányú replikáció. Ha két adatterületet replikációs kapcsolatba hozunk, azok automatikusan oda-vissza továbbítják a változásokat. Az esetek többségében ez pont megfelelő, de vegyünk elő gondolatban egy „Main office-Branch office” scenariót! A Branch office, tegyük fel, csak olvasható változatot kaphat egy, a Main office telephelyén elhelyezett könyvtárstruktúrából. Mit csinálunk? Fizikailag nagyon könnyen letiltható, de ez nem várt problémákat okozhat. Tegyük fel, hogy a Branch office-ban a helyi adminisztrátor letörli X könyvtárat. Másnap a Main office-ban módosítanak egy fájlt X-ben. Mi lesz? A Branch office szerverének DFS-R adatbázisa szétesik, ilyen esetekre nincs még felkészítve. Javaslat: a Branch office szerverén gondoskodjunk megosztási lehetőségről és NTFS-permissziókkal az írás és módosítás tiltásáról. A replikációt pedig hagyjuk inkább úgy, ahogy létrejön – kétirányúnak.

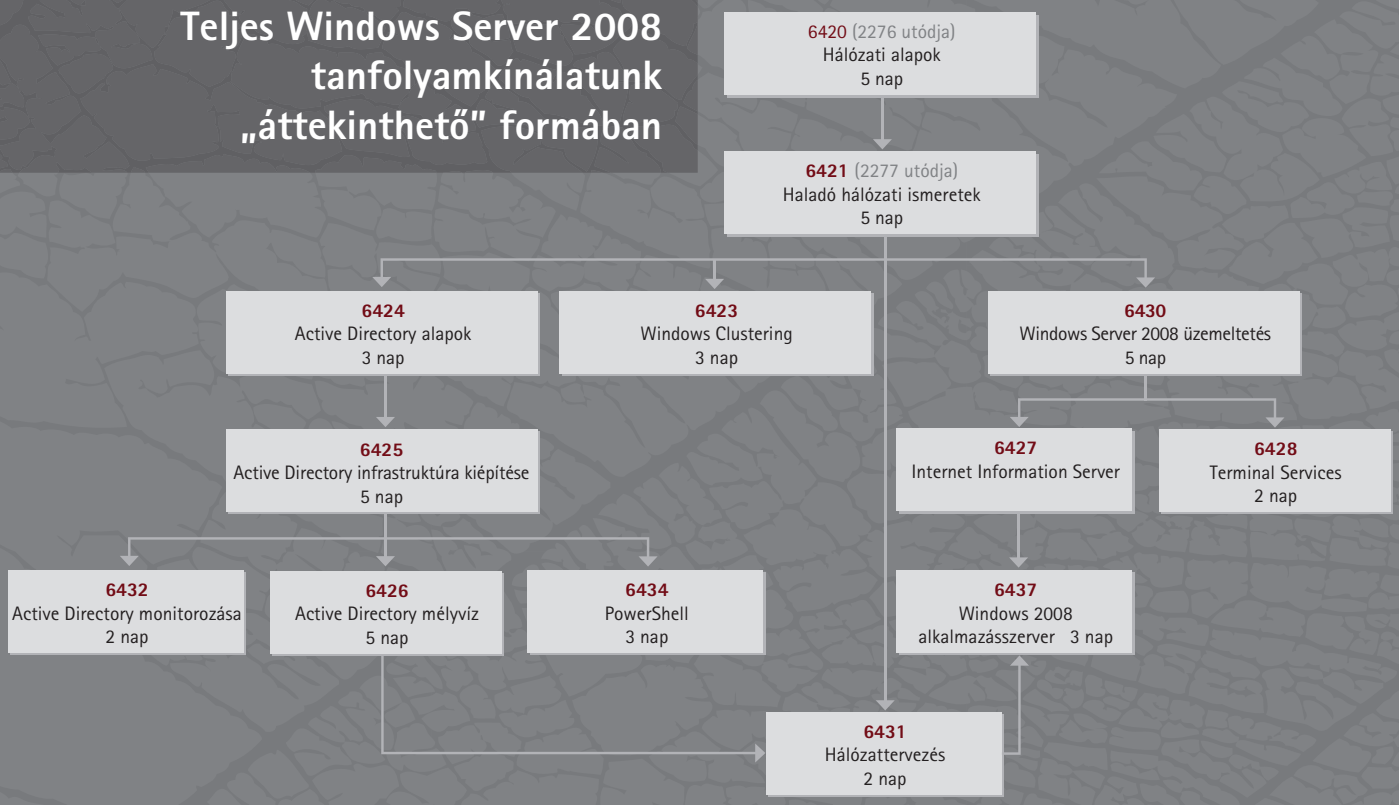
Székács András
(andras@edupro.hu) MCSE, MCTS, MCT, Számalk

Áttértünk! Windows Server 2008 tanfolyamok a NetAcademiánál

Windows Server 2008

A NetAcademia Oktatóközpont a hivatalos Windows Server 2008 tanfolyamok teljes választékát kínálja: harcedzett rendszergazdák és a szakmába frissen belépők egyaránt megtalálják, amire szükségük van.

Teljes Windows Server 2008 tanfolyamkínálatunk „áttekinthető" formában



Tanfolyamok Windows Server 2003 MCSA/MCSE-k részére

6415 - A Windows 2008 hálózati újdonságai és szolgáltatásai (3 nap)
6416 - A Windows 2008 Active Directory újdonságai (3 nap)
6417 - A Windows 2008 alkalmazáskiszolgáló újdonságai (3 nap)

Microsoft® GOLD CERTIFIED Partner

További információért kérjük, látogasson el honlapunkra (www.netacademia.net/windows2008), vagy keresse Szántó Zoltánt (szanto.zoltan@netacademia.net, tel.: 1/472-12-14).

netACADEMIA
A LEGJOBBAKAT TANÍTJUK.

Neked lehetőség. Nekünk kihívás.™

Microsoft®

Windows az asztalodon.
Windows a zsebedben.

HTC Touch

Élvezd a Windows előnyeit az irodán kívül is!

www.WindowsMobile.com

Az elérhető programok, funkciók és szolgáltatások a készülék típusától, Windows Mobile operációs rendszer verziótól és szervertípustól függően változhatnak. Bizonyos készülékekhez a Microsoft Office Mobile programot külön meg kell vásárolni. Kapcsolódáshoz és a szinkronizáláshoz szükség lehet külön megvásárolható termékekre és/vagy vezeték nélküli eszközökre (pl. Wi-Fi kártya, hálózati szoftver, szerver és/vagy átirányító szoftver). Internet, Wi-Fi és telefonos hozzáféréshez szolgáltatási terv szükséges. A használható funkciók és a teljesítmény szolgáltatónként eltérhet, hálózati korlátozások befolyásolhatják. További részleteket megtudhatnak a készülék gyártójától, a szolgáltatótól és/vagy a rendszergazdától.