Microsoft SQL Server 2000 Transact SQL – 4. rész
A felhasználói függvények titkai

Még be sem fejeződött az SQL Server 7 fejlesztése, és máris több oldalas volt az SQL Server fejlesztő csapat „kívánságlistája", azaz, hogy a megoldásszállító fejlesztők milyen funkciókat szeretnének látni a következő SQL Server verzióban. Ennek eredményeként született – az XML támogatás mellett – az SQL 2000 legnagyobb újítása a felhasználói függvények formájában. A cikkben nagyon tömören megnézzük a téma elméleti hátterét, hogy azután megírjunk néhány függvényt, amelyek segítségével szövegeket manipulálunk, megírjuk a Basic Split függvény SQL párját, megtanulunk körlevelet küldeni felhasználói függvényekkel, és kifejlesztünk egy behatolásjelző programot. Hosszú, de nagyon izgalmas rész lesz ez cikksorozatunkban, de érdemes végigolvasni!

Mindent, amit tudni akartál a felhasználói függvényekről, de nem merted megkérdezni

Ez a rész a felhasználói függvények lelkivilágával, formai és működésbeli tulajdonságaival foglalkozik. Aki tudja, miért fontos a determinizmus kérdése a függvényeknél, az nyugodtan ugorjon az utolsó fejezetre, ahol fokozatosan bonyolódó példákat találhat. Aki nem, az tartson velem a következő részekben is.

Az SQL Serverben nagyon sok beépített függvény található (lásd cikksorozatunk előző része), azonban ezek nyilvánvalóan nagyon általános függvények, mint például a (ruhaiparból ismert) `LEN` függvény, ami egy szöveg hosszát adja vissza. Nagyon jók ezek a beépített függvények, köszönjük őket, de a gyakorlati problémák megoldásához – pont az általános voltuk miatt – nem elegek. Mint építőkockák kitűnőek, de hogyan építünk belőlük várat? Nos, hosszú várakozás után a Microsoft elékészítette a habarcsot, megalkotta a felhasználói függvényeket, így most már semmi akadálya, hogy megalkossuk a saját `PAMUT` vagy a `GYAPJU` nevű függvényeinket, amelyek belső működését mi írhatjuk elő.

Egyszerűen megfogalmazva a felhasználói függvény olyan Transact SQL utasítások sorozata, amelyeket azért csomagolunk egybe, hogy több helyen is felhasználhassuk. Nagyon jól kiegészítik a tárolt eljárásokat, mert minden olyan helyen felhasználhatjuk őket, ahol a beépített függvényeket is, azaz ahol a tárolt eljárásokat legtöbbször nem. A legegyszerűbb példa erre a `SELECT`-ben való felhasználás. Például, ha van egy `osszeadas` nevű függvényünk, akkor azt felhasználhatjuk két oszlopban található számok összeadására, a `SELECT` utasítás részeként:

```
SELECT osszeadas(Ár, ÁFA), Termék FROM ...
```

Ennél kevésbé kézenfekvő helyeken is használhatjuk a függvényeinket: `WHERE` feltételben, `HAVING`-ben, `CHECK CONSTRAINT`-ekben, `DEFAULT CONSTRAINT`-ekben, számított oszlopok képzésében. Mindenhol működnek, ahol a szerver valamilyen *kifejezést* vár (mint a>b, c=4 vagy 2x2=5).

Azok kedvéért, akik nem szeretnek tömény oldalakat kódok nélkül látni, megmutatom az előbbi függvény deklarációját. Részletes magyarázatot a cikk második felében talál a lelkes Olvasó.

```
CREATE FUNCTION osszeadas
(
  @a INT,
  @b INT
)
  RETURNS INT
BEGIN
  RETURN @a + @b
END
```

Az SQL Server a függvényeket sokszor tranzakciók, illetve `SELECT`, `UPDATE`, satöbbi utasítások kellős közepén hívja meg. Emiatt rendetlen az a függvény, ami menet közben módosítja egy tábla tartalmát, miközben egy `SELECT` (ami őt hívta meg) éppen dolgozik rajta - nos ilyen esetben nagy lárma és kalamajka támadhatna. Az SQL Server azonban nem keresi a bajt, ezért megpróbálja megkötni a kezüket, hogy ne csináljunk felfordulást. Azaz a felhasználói függvényekben nem tehetünk meg akármit, csak a következőket:

- Definiálhatunk saját változókat és kurzorokat a `DECLARE` utasítással. Csak lokális kurzorokat készíthetünk így, globálisakat, azaz amelyek a függvény lefutása után is léteznének nem.
- A függvényben deklarált lokális változóknak értéket adhatunk (naná, e nélkül akár ki is dobhatnák a függvényeinket).
- Használhatunk kurzorműveleteket, de csak úgy, hogy a `FETCH` utasítás eredményeit lokális változókba rakjuk el (a kurzorokkal egy teljes cikk fog foglalkozni a következő hónapban).
- Bevethetjük a programfolyam-vezérlő utasításokat: `if`, `then`, `for`, `while`, `goto`, satöbbi. Ezek nélkül nem is lehetne egy komolyabb függvényt megírni.
- Alkalmazhatjuk az összes adatmódosító utasítást (`INSERT`, `UPDATE`, `DELETE`), ha azok csak lokális táblákon végeznek műveleteket. Ebből következően nem lehet módosítani külső táblákat. Természetesen lekérdezésekben szerepelhetnek.
- Meghívhatunk külső tárolt eljárásokat (Extended Stored Procedure) az `EXECUTE` utasítással. „Hagyományos" tárolt eljárásokat nem lehet meghívni belőlük, hisz azokból már könnyedén beavatkozhatnánk a „külvilágba".

Látható, hogy minden pontban arról van szó, hogy megtehetünk szinte bármit, amit csak akarunk, de csak lokálisan, azaz a függvény nem avatkozhat be a külvilágba. Van egy kis szemétdombunk, ott kapirgáljunk. Bár az utolsó pont, azaz, hogy külső tárolt eljárásokat is meghívhatunk, azért egy nagyon tág fogalom. Mert mit csinálhat egy külső tárolt eljárás? Bármit! Amit akar. Azaz például megteheti azt, hogy visszafelé nyit egy kapcsolatot a kiszolgálóra, és azon keresztül megváltoztatja azt a táblát, amiben éppen dolgozik a kódunk a függvény hívása során. De ez általában már túlmutat a normális használaton. Megtehették volna a fejlesztők, hogy teljesen letiltják a külső eljáráshívásokat, de akkor meg elestünk volna olyan nagyszerű lehetőségektől, mint külső parancsok meghívása (`xp_cmdshell`), levélküldés (`xp_sendmail`) vagy event log írás (`xp_logevent`) (és még sok egyéb hasznos funkció).

Az imént felsorolt három külső tárolt eljárás azonban pont olyan, aminek nem szabadna lefutni egy függvényben. Miért? Azért mert egy függvény nem változtathatja meg globálisan a rendszer állapotát. A rendszeren nem csak az SQL Server belső lelkivilágát értjük, hanem az egész világot. Így például az `xp_cmdshell` segítségével akár le is formázhatjuk kollégánk merevlemezét. Fogadjunk, hogy megváltozik a kolléga (lelki)állapota. :) Azaz ezeket a külső tárolt eljárásokat nem szabadna meghívni egy felhasználói függvényből, amire nyomatékosan fel is hívja a figyelmet a dokumentáció (Books Online).

Azonban, a fordító egy szót sem szól, ha olyan függvényt írunk, amiben felhasználjuk a veszélyes tárolt eljárások valamelyikét! Ezt még ki fogjuk használni a cikk végén található programokban. Pont olyan ez, mint a C programozás: ha meggondoltan csináljuk, miénk a világ. Ha nem, akkor csak General Protection Fault-okat generálunk.

A nemdeterminisztikus jövő

Vannak még más problémás elemek is, amelyeket bizonyos esetekben szintén nem szabad használni függvényekben. Ezek a nemdeterminisztikus függvények. Mik is ezek? Ők a függvények azon fajtái, amelyeknek a működése vagy az általa visszaadott érték időben vagy a szerver állapotától függően nem megjósolható módon változik. Azaz ugyanazokkal a paraméterekkel meghívva egyszer a-t mond, másszor b-t. A legegyszerűbb példa erre a `GetDate()` beépített függvény, ami a pillanatnyi időt adja vissza (a GetTime szerencsésebb név lett volna). Ez minden egyes meghívás pillanatában más értéket ad vissza, legalábbis addig, amíg jár a gépünkben a kvarckristály. A fordítóprogram nem engedi meg, hogy ilyen nemdeterminisztikus beépített függvényeket helyezzük el a saját függvényeinkben. Például a következő függvény törzsre: `RETURN RAND(10)` a fordító az „`Invalid use of 'rand' within a function.`” hibaüzenettel válaszol.
Miért ilyen problémás pont a determinizmus kérdése az SQL Serverben? Azért, mert vannak benne olyan új szolgáltatások, amelyek nem tudnának helyesen működni a „bizonytalan” nemdeterminisztikus függvényekkel. Két helyen nem lehet felhasználni a nemdeterminisztikus függvényeket:

- Indexelt számított oszlopokon, azaz, ha olyan oszlopra szeretnénk indexet készíteni, amelynek érékei egy másik (egy vagy több) oszlopból származnak, és a számított érték valamilyen nemdeterminisztikus függvényen alapul.
- Olyan nézetekben, ahol a nézetre clustered indexet szeretnénk használni.

A két megszorítás alapján már eléggé érthető, hogy miért kell foglalkozni a determinizmus kérdésével. Mindkét esetben indexet építünk táblában található adatokra. Próbált már valaki megülni egy vásári bikát? Nem egyszerű. Hasonló módon az SQL Server sem tud indextáblát építeni olyan adatokra, amelyek minden pillanatban változnak. A clustered index az adatok fizikai sorrendjét határozza meg. Ezen a héten így legyek sorban az adatok, a következő héten meg másképp, csak azért, mert meggondolta magát a transzformáló függvény? Na nem, ez nonszensz lenne. Ezért nem is tehetünk ilyet.

Ragaszkodás a barátokhoz

Egyetlen apró fogalom maradt már csak hátra, hogy ténylegesen megírhassuk első függvényünket. Ez a séma-kötés fogalma. A felhasználói függvények igen erősen kötődnek azokhoz a táblákhoz, és egyéb objektumokhoz, amelyekre hivatkoznak. Ha azok módosulnak anélkül, hogy erről a függvény tudna, akkor a kapcsolatuk vége barátságtalan lesz, és a függvény nem fog jól működni. Azért, hogy a jó viszonyban ne következhessen be szakadás, a függvény létrehozásakor (`CREATE FUNCTION`) megadhatjuk, hogy a függvény legyen hozzákötve az általa használt objektumokhoz. Ezt az SQL Server megjegyzi, és nem engedi módosítani vagy törölni az ily módon leláncolt objektumokat. A kötés jelzését a `RETURNS` és a függvény törzsét kezdő `BEGIN` közé kell írni:

```
...
RETURNS ...
WITH SCHEMABINDING
BEGIN
...
```

Függvénytípusok

Háromféle felhasználói függvénytípust hozhatunk létre az SQL 2000-ben:

- Skaláris függvények, melyeknek visszatérési értéke skaláris, azaz egy érték (scalar functions)
- Egy utasításból álló, tábla visszatérési értékű függvények (inline table valued functions)
- Több utasításból álló, tábla visszatérési értékű függvények (multi statement table valued functions)

Az utóbbi két fajta nagyon hasonlít egymásra, mint ez a részletes tárgyalásból hamarosan kiderül.

Skaláris függvények

A skaláris függvények nagyon egyszerűek: kapnak néhány paramétert, azokon végeznek valamilyen művelet, majd az eredményt egy skaláris értékként visszaadják. Azaz visszaadnak egy számot, egy szöveget, egy dátumot satöbbi. Leginkább a procedurális nyelvek függvényeihez hasonlítanak.
Rutinos tárolt eljárás programozók! A felhasználói függvényeknek nincsenek kimeneti paramétereik! Azaz nem lehet valamelyik paramétert megjelölni, hogy az visszafelé fog majd valamilyen információt szolgáltatni a hívónak. Ezt a lehetőséget azért kellett bevezetni a tárolt eljárásoknál, mert azok csak egy egész számot tudnak visszaadni visszatérési értékként, így nem tudtunk volna például egy dátumot visszaadni a hívónak. Erre szolgáltak a kimeneti paraméterek. Hogy teljesen érthető legyen, álljon itt egy tárolt eljárás, amelynek a harmadik paramétere kimeneti paraméter:

```
CREATE PROCEDURE osszead
  @a INT,
  @b INT,
  @c INT OUTPUT
AS
SET @c = @a + @b

--Eddig a tárolt eljárás deklarációja.
--Látható, hogy egy tárolt eljárásban nem
--kötelező a visszatérési értéket megadni
```

```
--Azaz lehetne egy záró RETURN ..., de
--nem szükséges, mert most nem használjuk
--fel a visszatérési értéket.

DECLARE @osszeg INT
--hívjuk meg
EXECUTE osszead 1,4, @osszeg OUTPUT
SELECT @osszeg
5  --Működik!
```

Nos, kimeneti paraméter nincs a felhasználói függvényekben. Viszont segítségükkel sokkal egyszerűbben meg lehet fogalmazni az előbbi problémát:

```
CREATE FUNCTION osszeadas(
  @a INT,
  @b INT)
  RETURNS INT
BEGIN
  RETURN @a + @b
END

SELECT dbo.osszeadas(1,4)
```

Azért ez sokkal természetesebb, mint a tárolt eljárásos változat. De azért szedjük csak szét ízekre a függvény deklarációt! A `CREATE FUNCTION` jelzi, hogy ez egy felhasználói függvény lesz. Ezután jön a függvény neve. Általában a függvényeknek vannak paramétereik, ezeket zárójelben soroljuk fel a függvény neve után. A `@` nem opcionális, nem esztétikai okokból raktam bele, vagy azért, mert ettől olyan tudományos lesz, hanem azért, mert Transact SQL-ben minden változót kötelező `@`-al kezdeni. A paraméter neve után meg kell adni az ő típusát. Itt majdnem az összes, a kiszolgáló által támogatott adattípust fel lehet használni, egy-két elvarázsolt `image`, `text` vagy `cursor` típust kivéve. A `RETURNS` után kell definiálni a visszatérési érték adattípusát. A kötöttségek ugyanazok, mint a paramétereknél, azaz csak „normális” változókat használhatunk. A függvény törzsét, ahol az általunk megálmodott funkcionalitást írjuk le, a `BEGIN` és `END` kulcsszavak közé kell elhelyezni. Ennyi.
Mondja azt valaki, hogy bonyolultak a felhasználói függvények! Ha a fenti mintapélda kéznél van, minden problémát csuklóból megoldunk. Persze enyhe túlzással, és ha egy kimeneti érték elég a feladat leírásához. :)
Még egy fontos tudnivaló. A skaláris visszatérési értékű függvényekre minimum 2 tagú névvel kell hivatkozni. Azaz legalább a függvény tulajdonosát meg kell adnunk ahhoz, hogy az SQL Server felismerje a függvényünket. Ennek megfelelően, a:

```
SELECT osszeadas(1,4)
```

hibát fog jelezni. Helyesen:

```
SELECT dbo.osszeadas(1,4) vagy

SELECT Northwind.dbo.osszeadas(1,4)
```

De mi van, ha több értéket kell visszaadnunk? Mi van, ha ráadásul azt sem tudjuk, hogy igazából hány kimeneti értékünk lesz, mert azt a tábláinkban található információk pillanatnyi állapota szabja meg? Ebben az esetben kapaszkodunk a tábla kimenetű felhasználói függvényekbe. (A továbbiakban nem írom ki mindenhol a felhasználói jelzőt, de ott van.)
Ezt értsük úgy, hogy, ha a skaláris függvények egy skaláris mennyiséget adnak vissza, akkor a tábla kimenetűek meg egy táblát? Igen. De, hát nincs is ilyen adattípus az SQL Server 7-ben! Abban tényleg nincs, de az SQL 2000-ben van. És nagyon szeretjük is őket. Képzeljük el: van egy olyan változótípusunk, ami akár egy tízmillió sorból és huszonhat oszlopból álló teljes táblát el tud tárolni. Csoda, hogy szeretjük? Ez a tábla (table) adattípus.
Miért olyan szenzációs ez? Eddig is létre lehetett hozni átmeneti táblákat, és azokba is lehetett ideiglenes eredményeket beleírni. Persze, de a tábla adattípus felhasználásával egyrészt átláthatóbban, a természetes gondolkodáshoz közelebb álló kódot hozhatunk létre, másrészt olyan dolgokat is megvalósíthatunk, amelyeket korábban csak nagyon trükkösen vagy sehogyan sem tudtunk megtenni.
Hol használhatjuk fel a tábla kimenetű függvényeket? Minden olyan helyen, ahol eddig egy táblát adhattunk meg. Azaz leginkább a `FROM` záradék után.

```
SELECT cica, egér

FROM AzElsoTablaFuggvenyem('sajt')
```

Paraméterezett nézetek felhasználói függvényekkel, avagy az egy utasításból álló, tábla visszatérési értékű függvények

Mit tudtunk tenni SQL7-ben, ha azt kérték tőlünk, hogy kellene egy nézet, ami a megrendeléseket listázza ki, de úgy, hogy megadhassuk paraméterként, hogy melyik megrendelőhöz tartozó tételeket kívánjuk látni. Azaz valami ilyesmit akartunk írni:

```
CREATE VIEW OrdersByCustomer(
  @CustomerID varchar(5))
AS
SELECT * FROM Orders
```

```
WHERE
  CustomerID = @CustomerID
--Nem működik, nem fordul le!
```

Nos, ilyen nincs SQL7-ben, sőt SQL2000-ben sem! Ilyenkor jön a felmentő sereg, az egy utasításból álló, tábla visszatérési értékű függvény. Az előbbi majdnem működő nézetet könnyen átalakíthatjuk egy tábla visszatérési értékű függvénnyé, ami már az elvárt funkciót valósítja meg:

```
CREATE FUNCTION OrdersByCustomer(
@CustomerID varchar(5))
RETURNS TABLE
AS
  RETURN (
  SELECT * FROM Orders
  WHERE
    CustomerID = @CustomerID)

--Teszt:
SELECT CustomerID, ShippedDate
FROM OrdersByCustomer('THEBI')

THEBI      1996-09-27
THEBI      1997-11-05
THEBI      1998-01-09
THEBI      1998-04-03
```

Mit kellett tennünk, hogy a majdnem-működő, de azért mégiscsak-ramaty nézetünkből egy jólfésült függvény legyen? A `CREATE VIEW` helyett `CREATE FUNCTION`-t írunk. Jelezzük, hogy a függvény visszatérési értéke nem holmi skalár, hanem tábla: `RETURNS TABLE`. Látható, hogy nem specifikáltuk az eredménytábla szerkezetét, csak egyszerűen megadtuk, hogy tábla lesz. Emiatt van, hogy az ilyen típusú függvényekben csak 1, azaz egy darab `SELECT` utasítás lehet, hiszen annak az eredményhalmaza határozza meg a visszatérési értékként generálódó tábla típusát. Pontosabban lehet benne egymásba ágyazva több `SELECT` utasítás is, de a teljes lekérdezés csak egy eredményhalmazt adhat vissza. Azaz pont ugyanaz a helyzet, mint a nézeteknél volt.

Több utasításból álló, tábla visszatérési értékű függvények

Bonyolultabb esetben a visszatérési érték nem állítható elő egyetlen `SELECT` utasítás segítségével, ilyenkor kell használnunk ezt a függvénytípust. Mivel ilyenkor már nem egyértelmű, hogy melyik lekérdezés kimenetét szeretnénk visszaadni, explicit deklarálnunk kell a visszatérési értékként szolgáló tábla szerkezetét egy tábla típusú változóként. A változót `INSERT` utasítások segítségével feltöltjük (akárhány lépésben), és a `RETURN` utasítás ezt fogja visszaadni a hívónak.
Erre a függvénytípusra összetettebb példákat a következő fejezetben találhatunk.

Praktikus felhasználói függvények

Annak örömére, hogy megkaptuk a felhasználói függvényeket, használjuk ki az alkalmat, és írjuk meg néhány olyan probléma megoldását, ami a minden napi fejlesztések során sokszor előjött-előjön.

Szövegelőfordulás számláló

Gyakori feladat, hogy egy szövegben meg kell keresni azt, hogy egy másik szöveg hányszor fordul elő benne. Milyen algoritmust használjunk? Az egyik legegyszerűbb, bár nem feltétlen a leghatékonyabb módszer az, hogy a keresendő szöveg minden egyes előfordulását cseréljük ki egy üres sztringre a „nagy” szövegben (amiben keresünk), és az eredeti szöveg hosszából vonjuk ki az így kapott szöveg hosszát. Ezt az eredményt már csak le kell osztani a keresendő szöveg hosszával, hisz minden csere után ennyivel csökkent az „nagy” szöveg hossza. Hogy néz ez ki függvényként? (A bemutatott példa egy nagyon *nem* normalizált adatbázis, annyira nincs formában, hogy még 0. normál formában sincs. Csak demócélokra szolgál, nem adatbázistervezési minta!)

```
CREATE FUNCTION StringOccur
(
  @cString  AS varchar(8000),
  @cLookFor AS varchar(100)
)
RETURNS int
AS

BEGIN
  RETURN
  (LEN(@cString)
  -LEN(REPLACE(@cString, @cLookFor, '')))
  / LEN(@cLookFor)
END

--Teszt tábla

CREATE TABLE T1
(
  cMenu varchar(100) NOT NULL
```

```
)

--Tesztadatok
INSERT INTO T1 VALUES('Töltött káposzta, Almáspite, Diósbejgli')

INSERT INTO T1 VALUES('Pulykarizottó, Mákosbejgli, Diósbejgli')

INSERT INTO T1 VALUES('Székelykáposzta, Rántottbéka, Mákosbejgli')

INSERT INTO T1 VALUES('Stefániasült, Káposztáspite, Túrósbejgli')


SELECT
  cMenü AS Menü,
  dbo.StringOccur(cMenu, 'káp') AS
  Káposztásfogás,
  dbo.StringOccur(cMenu, 'bejgli') AS
  Bejglitartalom,
  dbo.StringOccur(cMenu, 'Mákos') AS
  Mákosfogás
FROM T1

--A kimenet (nyomdai okokból táblázatban):
```

Menü			Káposztás fogás	Bejglitartalom	Mákos fogás
Töltött káposzta	Almás pite	Diós bejgli	1	1	0
Pulykarizottó	Mákos bejgli	Diós bejgli	0	2	1
Székelykáposzta	Rántott béka	Mákos bejgli	1	1	1
Stefániasült	Káposztás pite	Túrós bejgli	1	1	0

A függvény elég trükkös, megér néhány szót. „Izomból” nekifutva hogyan oldanánk meg a példát? Egy ciklusban keresnénk a keresendő szöveg előfordulásait a „nagy” szövegben, mindig a következő pozíción (karakteren) folytatva a „nagy” szövegben, mint ahol az előző lépésben abbahagytuk. Ehhez a megoldáshoz ciklust kellene szerveznünk, ami jelentősen megbonyolítaná a megoldást. Ehhez képest a fenti függvény sokkal egyszerűbb, hisz a bonyolultabb funkcionalitást átadtuk a `REPLACE` függvénynek.
Más kérdés, hogy az imént vázolt algoritmus és a fenti algoritmus más kimenetet ad például a következő szövegekre:

```
--A fenti függvény (a LEN-es)
SELECT dbo.StringOccur('babababababa', 'baba')
2
```

Ezzel ellentétben, ha lenne egy függvényünk, ami az imént említett módon működne, akkor a visszaadott érték 4 lenne, hisz:

```
babababababa

babababababa

babababababa

babababababa
```

A kérdés az, hogy átlapolhatják-e egymást a keresendő szöveg előfordulások? Ha nem, akkor jó a fenti függvény, ha igen, akkor meg kell írni a másik verziót. Ezt a konkrét feladat határozza meg.

Szövegdarabolás

Visual Basic programozók gyakran keresik a Basic Split függvény Transact SQL párját. Mindhiába, mert nincs. A Split egy nagyon hasznos függvény, arra való, hogy egy szöveget valamilyen határoló karakter mentén feldaraboljon, és a darabokat visszaadja egy tömbben. Segítségével egy mondatot feldarabolhatunk szavakra, egy vesszővel elválasztott listát listaelemekre, satöbbi.
Mivel nincs ilyen függvényünk, implementáljunk egyet! Az első akadály, amibe rögtön beleütközünk az, hogy a TSQL-ben nincs tömb típus. Emiatt a függvény kimenete tábla típusú kell, hogy legyen, mert skalárban nem tudunk visszaadni több elemet. Azaz, írjunk egy olyan függvényt, ami a megadott szöveg és az elválasztó karakter ismeretében szétdarabolja a szöveget, és egy táblában visszaadja a szöveg komponenseket. Legyen a visszaadott mező neve cStringPart!

```
CREATE FUNCTION Split
(
  @cOriginalString AS varchar(8000),
  @cDelimiter char(1))
  RETURNS @SplitString table
(
  nID int IDENTITY(1,1) NOT NULL,
  cStringPart varchar(8000) NULL)
AS
BEGIN
```

```
  DECLARE @nNumberOfDelimiters AS int
  --Számoljuk  meg, hány határoló
  --karakterünk van.
  --Használjuk fel az előzőleg megírt
  --szöveg-előfordulás számláló
  --függvényünket.
  SET @nNumberOfDelimiters =
  dbo.StringOccur(
  @cOriginalString, @cDelimiter)

  DECLARE @i AS int
  SET @i = 0
  --Végigmegyünk az összes szövegdarabon
  WHILE @i < @nNumberOfDelimiters
  BEGIN

  --A forrás szöveg baloldalából
  --kivágjuk az ott található szöveget
  --a határoló karakterig,
  --és beszúrjuk az eredménytáblába.
    INSERT INTO
      @SplitString
    SELECT
      LEFT(@cOriginalString,
      CHARINDEX(@cDelimiter,
      @cOriginalString)-1)

    --Levágjuk a már feldolgozott
    --szöveget, így az elején mindig
    --megtaláljuk a következő darabot.
    SET @cOriginalString =
    SUBSTRING(@cOriginalString,
    CHARINDEX(@cDelimiter,
    @cOriginalString)+1, 8000)

    --Továbblépünk a következő darabra
    SET @i = @i + 1

  END

  --Az utolsó határoló karakter után
  --még maradt egy darab, azt is
  --szúrjuk be az eredményhalmazba.
  INSERT INTO
    @SplitString
  VALUES
    (@cOriginalString)

  --Összeállt az eredménytábla, ideje
  --visszaadni azt a hívónak.
  --Itt már nem kell jelezni, hogy mit
  --adunk vissza, mert az már a RETURNS-
  --nél (az elején) megtettük.
  RETURN

END

--Teszt

SELECT
  T1.*
FROM
  Split('Dec 24,Dec 25,Dec 26,Dec 31',',')
  AS T1

--Eredmény:

nID         cStringPart
1           Dec 24
2           Dec 25
3           Dec 26
4           Dec 31
```

Látható, hogy a függvények egymásba ágyazhatók, éppúgy, mint a tárolt eljárások. Ezzel élve nagyon jól átlátható, moduláris programokat írhatunk az SQL Serverre.

Spam-re fel!

Az SQL Server segítségével könnyedén írhatunk körleveleket, ha van egy címzett (áldozat) adatbázisunk. A klasszikus megoldásban kurzort használnánk, és az `xp_sendmail` külső tárolt eljárást hívnánk meg egy ciklusban. Azonban a kurzorok használata elég körülményes dolog. Keressünk egy jóval egyszerűbb megoldást, természetesen a felhasználói

függvények felhasználásával! A megcélzott függvény célja egyszerű: a bemenő paraméterekben meghatározott címzettnek elküldeni egy E-mail-t.

```
--Létrehozzuk a függvényt
CREATE FUNCTION SendMail
(
  @cRecepients AS varchar(200),
  @cSubject AS nvarchar(100),
  @cBody AS nvarchar(3000)
)
  RETURNS INT

BEGIN
  DECLARE @nResultCode INT
  EXEC @nResultCode = master..xp_sendmail
  @recipients = @cRecepients,
  @subject = @cSubject,
  @message = @cBody
  RETURN @nResultCode
END

--Egy teszttábla a „célszemélyekhez”
CREATE TABLE SpamTarget
(
  nID int NOT NULL IDENTITY(1,1),
  cTargetEmail nvarchar(400) NOT NULL,
  cFirstName nvarchar(100) NOT NULL,
  cLastName nvarchar(100) NOT NULL
)

--Két áldozat felvitele
INSERT SpamTarget VALUES ('Zsolt.Soczo@w2ktest1.vodafone.hu', 'Zsolt', 'Soczó')
INSERT SpamTarget VALUES ('ECudar@vadmalac.hu', 'Elek', 'Cudar')

--A levelek elküldése.
SELECT
  dbo.SendMail(cTargetEmail,
  cFirstName +
  '! Nyerj 99999999999 Forintot!',
  'Legyél te is milliomos!')
FROM
  SpamTarget

--Az áldozat által kapott levél:
From: sqlacc
Sent: Saturday, December 23, 2000 6:08 PM
To: Zsolt Soczo
Subject: Zsolt! Nyerj 99999999999 Forintot

Legyél te is milliomos!
```

Anti-hacking toolkit v0.0

Utolsó és egyben legbonyolultabb függvényünkben egy állomány épség (eredetiség) ellenőrző programot írunk. A dupla KV ismét javasolt előtte, mert elég bonyolult lesz.

A feladat, hogy dolgozzunk ki egy olyan módszert, amely segítségével a védendő állományok bizonyos jellemzőit letároljuk, majd egy ellenőrző rutint lefuttatva leellenőrizzük, hogy a jellemző azonos-e a letárolt, háborítatlan értékkel. Ha nem, akkor a megfigyelt állományt egy rosszindulatú hacker vagy egy még rosszabb indulatú telepítőprogram módosította. A példa kedvéért az állomány méretet használjuk fel az ellenőrzéshez. Ennél sokkal profibb megoldás lenne, ha az állományokhoz kiszámítanánk valamilyen ellenőrző értéket (pl. MD5 hash), és ezt tároljuk le az adatbázisban. Így sokkal nagyobb valószínűséggel lehetne jelezni, hogy megváltozott egy állomány.

Hogyan látnánk neki a feladat megoldásának? Mivel a fájlok méretét közvetlenül nem lehet lekérdezni az SQL Serverből, kénytelenek vagyunk kinyúlni a szerverből. Ehhez valamilyen külső tárolt eljárásra lesz szükségünk. Az `xp_cmdshell`, amivel külső parancsokat lehet végrehajtani, szinte kínálja magát, hogy bevessük erre a feladatra. Meghívunk egy VBScript programot, ami visszaadja a paraméterként megadott állomány hosszát. A külső parancs futtatásából származó sorokat, azaz a fájl hosszát az `xp_cmdshell` táblaként adja vissza, aminek az első sora tartalmazza a kívánt eredményt. Hogyan nyerjük ki ebből a táblából az első sort? Próbáljuk belerakni egy átmeneti (temporary) táblába, és abból leválogatni az eredményt. Ez azonban sajnos nem megy, mert függvényben nem használhatunk temporary táblát.

Próbáljuk meg belemásolni az `xp_cmdshell` kimenetét egy table típusú változóba. Ez sem megy, mert az `INSERT Tábla (EXECUTE xp_cmdshell ...)` típusú parancs, (ami egy tárolt eljárás kimenetét beszúrja egy táblába) nem megy tábla típusú változóval, csak valódi táblával. „Valódi” táblát viszont nem módosíthat egy függvény. Van ebből kiút?

Utolsó kapaszkodóként elfeledkezünk az `xp_cmdshell`-ről, és megpróbáljuk felhasználni a külső COM komponensek megívására szolgáló függvényeket. És ez bejön! A FileSystemObject COM komponens közvetlen meghívásával célba érünk. A kódhoz tartozó magyarázatot beleszőttem a kódba, mert kiragadva kevésbé érthető lenne.

```
--A fájlméret lekérdező függvény
--deklarációja.

CREATE FUNCTION GetFileSize
(
  @cFilePath AS nvarchar(4000)
)
 RETURNS INT
BEGIN

  DECLARE @nFileSize int
  DECLARE @hr int
  DECLARE @objFileSystem int
  DECLARE @objFile int

--Hozzunk létre a FileSystemObject-
--ből egy példányt.
  EXEC @hr = sp_OACreate
  'Scripting.FileSystemObject',
  @objFileSystem OUT

--Ha hiba történt egyszerűen visszatérünk
--egy hibakóddal. Ez azért nagyon csúnya,
--mert a kód későbbi részeiben
--bekövetkezett hiba esetén
--felszabadítatlan objektumok maradnak a
--memóriában!
--Így produkciós környezetben le
--kell kezelni a hibákat megfelelő módon.
--Ehhez az sp_OAGetErrorInfo külső tárolt
--eljárást lehet segítségül hívni.
  IF @hr <> 0 RETURN -1

--Meghívjuk a FileSystemObject GetFile
--nevű metódusát, ami visszatér egy
--File típusú objektummal. Ezt az
--@objFile változóban tároljuk el.
--A hívás paramétere az a fájlnév, aminek
--keressük a méretét (@cFilePath).
  EXEC @hr = sp_OAMethod @objFileSystem,
  'GetFile', @objFile OUT, @cFilePath
  IF @hr <> 0 RETURN -1

--A File objektum Size nevű
--tulajdonságának lekérdezésével
--megkapjuk a keresett állomány méretét.
--A kapott szám az @nFileSize-ba kerül.
  EXEC @hr = sp_OAGetProperty @objFile,
  'Size', @nFileSize OUT
  IF @hr <> 0 RETURN -1

--Felszabadítjuk a létrehozott
-- objektumokat.
  EXEC @hr = sp_OADestroy @objFile
  IF @hr <> 0 RETURN -1

  EXEC @hr = sp_OADestroy @objFileSystem
  IF @hr <> 0 RETURN -1

--Visszatérünk a kapott értékkel.
  RETURN @nFileSize

END

--Ebben a táblában tároljuk a fájlokat,
--és a hozzájuk tartozó méreteket.
CREATE TABLE FileAuthority
(
  nID int NOT NULL IDENTITY(1,1),
  cFileName nvarchar(3000) NOT NULL,
  nFileSize int NOT NULL
)

--Néhány tesztállomány. A -2-vel jelezzük
--hogy még soha nem olvastuk ki az adott
--fájl hosszát.
INSERT FileAuthority VALUES
('c:\winnt\notepad.exe', -2)
INSERT FileAuthority VALUES
('c:\boot.ini', -2)
INSERT FileAuthority VALUES
```

```
('c:\ntldr', -2)
INSERT FileAuthority VALUES
('c:\io.sys', -2)
INSERT FileAuthority VALUES
('c:\ntdetect.com', -2)


--Ezzel a tárolt eljárással
--töltjük fel a táblázat méret mezőit.
CREATE PROCEDURE CalculateFileSize
AS
  UPDATE
    FileAuthority
  SET
    nFileSize = dbo.GetFileSize(cFileName)

--Futtassuk le. Ettől kezdve van egy
--táblázatunk arról, hogy melyik fájlnak
--milyen hosszúnak kell lenni.
EXEC CalculateFileSize

--Nézzük meg, mit tartalmaz a táblánk!
--SELECT * FROM FileAuthority

FileName                       FileSize
c:\winnt\notepad.exe           50960
c:\boot.ini                    195
c:\ntldr                       214416
c:\io.sys                      0
c:\ntdetect.com                34468

--Ezzel a tárolt eljárással össze
--lehet hasonlítani a letárolt és
--a futtatás pillanatában aktuális
--állományhosszakat.
--Csak azokat listázza ki, amelyeknél
--eltérés van a két érték között.
CREATE PROCEDURE CheckFileSize
AS
  SELECT
    nID,
    cFileName,
    nFileSize AS OriginalSize,
    dbo.GetFileSize(cFileName) AS
    CurrentSize
  FROM
    FileAuthority
  WHERE
    nFileSize <>
    dbo.GetFileSize(cFileName)

--Tesztképpen megváltoztattam a boot.ini
--fájl hosszát.
--Ellenőrizzük le!
EXEC CheckFileSize

A kimenet:
nID cFileName   OriginalSize CurrentSize

2   c:\boot.ini 195          197
```

Hoppá, a `BOOT.INI`-t valaki megváltoztatta! Működik az ellenőrző eljárásunk.

Zárszó

A cikkben felhozott példák két igen fontos dologra világítanak rá. A felhasználói függvények felhasználásával nagyon sok, a gyakorlatban felmerülő feladatot oldhatunk meg, amelyeket eddig csak átmeneti táblák és kurzorok felhasználásával tudtunk megtenni, általában nagyon bonyolultan, és nehezen olvasható módon. A másik tanulság, hogy a külső tárolt eljárások segítségével sok olyan feladatot is megoldhatunk az SQL Server segítségével, amelyeket általában más programnyelven megírt programokkal (Visual Basic, stb.) végeztettünk el.
A következő részben a kurzorokról lebbentem fel a fátylat, megnézzük, hogy körültekintő felhasználásukkal a felhasználói függvényekhez hasonlóan igen bonyolult feladatokat is elég egyszerűen megoldhatunk.

Soczó Zsolt MCSE, MCSD, MCDBA
Protomix Rt.