Microsoft SQL Server 2000 Transact SQL – 3. rész
Összetett lekérdezések

Az előző cikkünkben belemélyedtünk az illesztések lelkivilágába. Megnéztük, hogy egymásba ágyazott lekérdezésekkel milyen egyszerűen meg lehet oldani bonyolult problémákat is. Most további igen hasznos nyelvi elemekkel ismerkedünk meg, amelyek segítségével csoportosíthatjuk adatainkat, műveleteket végezhetünk a csoportokkal, és nagyon látványos riportokat tudunk készíteni. Ehhez kapcsolódóan megnézzük, hogy a szerverbe beépített függvények segítségével mennyire át lehet alakítani az adatok jelentését.

Csoportosítsunk!

Bemelegítésül nézzük meg az alábbi lekérdezést:

```
SELECT
  od.OrderID, p.ProductName,
  od.UnitPrice * od.Quantity AS Amount
FROM
  [Order Details] od
INNER JOIN
  Products p
ON
  od.ProductID = p.ProductID
ORDER BY
  OrderID


OrderID  ProductName     Amount
10248    Queso Cabrales  168.00
10248    Singaporean      98.00
10248    Mozzarella      174.00
10249    Tofu            167.40
```

Egyszerűen kilistáztuk a megrendeléseket, kiszámolva az adott tétel értékét (`od.UnitPrice * od.Quantity`). Szép ez a lista, csak túl részletes. Például a 10248-as megrendeléshez három sort listázott ki, mert a megrendelés három altételből állt. Egy riportban nem érdekesek az ilyen részletek, általában csak arra van szükség, hogy egy megrendelés *összesen* mekkora értékű volt.
Első felindulásunkban a már ismertetett `SUM` függvényt használnánk, ami képes arra, hogy összegezze a tételeinket:

```
SUM(od.UnitPrice * od.Quantity) AS Amount
```

Persze ez nem azt tenné, amit várnánk tőle, hanem az összes megrendelés értékét összeadná, és eredményül egy számot kapnánk, amelyben minden megrendelés együttes értéke lenne. Mi lenne, ha lenne olyan utasításunk, amivel megmondhatnánk, hogy csoportosítsa a sorokat az `OrderID` mező alapján, és a csoportokra végezze el az összegzést? Természetesen van ilyenünk, a `GROUP BY` az. Segítségével a `GROUP BY` mögé írt oszlopok szerint történik az eredményhalmaz csoportosítása, azaz az azonos `OrderID`-jú sorokból egyet készít, és a csoportokra kiszámítja az aggregált eredményeket.
Alakítsuk át a példánkat úgy, hogy megrendelésenként összegzett listát készítsen a `GROUP BY` és a `SUM` segítségével:

```
SELECT
  od.OrderID,
  SUM(od.UnitPrice * od.Quantity)
  AS Amount
FROM
  [Order Details] od
INNER JOIN
  Products p
ON
  od.ProductID = p.ProductID
GROUP BY
  od.OrderID
ORDER BY
  OrderID

OrderID     Amount
10248        440.00
10249       1863.40
10250       1813.00
```

Nagyszerűen működik! Miért vettem ki a `p.ProductName` oszlopot? Azért, mert semmi értelme, hisz pont az volt a célunk, hogy termékektől és megrendelés tételektől *független* listát kapjunk. Ha ezt elfelejtenénk, figyelmeztetni fog a szerver:

```
Column 'p.ProductName' is invalid in the select list because it is not contained in either an aggregate
function or the GROUP BY clause.
```

Azaz a fordító a `p.ProductName`-et csak akkor fogadja el `SELECT` mögött, ha azt vagy felsoroljuk a `GROUP BY`-ban, vagy egy aggregáló függvénybe foglaljuk bele. Az előbbinek az lenne a következménye, hogy a megrendelés tételeken belül termékenként tovább lenne bontva a részösszeg. Sokszor ez is cél lehet.
A második javaslattal kapcsolatban: nehéz lenne olyan beépített aggregáló függvényt keresni, ami a terméknéven valami hasznosat tudna végezni. Úgyhogy ezt felejtsük el.

Mi van, ha csak azokat a megrendeléseket akarjuk kilistázni, amelyek össz-megrendelés értéke nagyobb, mint 1000$? A WHERE használata sajnos nem vezet eredményre, mert a WHERE csak az egyes Order Details sorokban található értékekre tud szűrni, és nem pedig azok összegére. Másképpen fogalmazva valami ilyesmit szeretnénk látni a WHERE-ben:

```
WHERE
  SUM(od.UnitPrice * od.Quantity) > 1000
```

vagy

```
WHERE Amount > 1000
```

Csakhogy a WHERE-ben nem lehet használni aggregáló függvényeket, így a SUM-ot sem. Hogyan juthatunk túl ezen a dilemmán? Úgy, hogy van egy olyan speciális záradék (clause), amelyet arra találtak ki, hogy a GROUP BY által definiált csoporton lehessen vele feltételeket érvényesíteni. Ez a záradék a HAVING. A HAVING nagyon hasonló a WHERE-hez, a különbség abban rejlik, hogy a záradékok után álló kifejezés mikor kerül kiértékelésre. A WHERE után álló kifejezést a csoportosítás *előtt* értékeli ki a végrehajtó egység, azaz a WHERE segítségével előre kiválogatjuk azokat a sorokat, amelyeket csoportosítani szeretnénk. Ezután jön maga a GROUP BY-ban előírt csoportosító művelet. Létrejönnek a csoportok, valamint kiszámítódnak a csoportokra kijelölt aggregáló kifejezések. Ekkor jön a képbe a HAVING. A parancsvégrehajtó kidobálja azokat a *csoportokat*, amelyekre nem teljesül a HAVING után álló feltétel. A szenzációs az a dologban, hogy a HAVING után használhatunk aggregáló függvényeket, ellentétben a WHERE-el!
Mivel a WHERE segítségével drasztikusan le lehet csökkenteni a csoportosítandó sorok számát, ezért érdemes minden olyan feltételt, ami nem a csoportokra vonatkozik a WHERE-be rakni a HAVING helyett. Ha ellenkezően cselekszünk, a fordító nem fog figyelmeztetni minket. Ő szolgai módon végrehajtja a lekérdezést az általunk előírt módon. A felhasználók viszont szólnak majd az alkalmazásunk lassúsága miatt...
Szerencsére azonban a Query Optimizer ennél okosabb. Általában, hangsúlyozom, *általában* észreveszi, hogy nem optimálisan írtuk meg a lekérdezést, és a nem megfelelő helyre írt kifejezéseket a végrehajtás idejére átrakja a megfelelő helyre. Még sokszor tapasztaluk majd, ahogy SQL szerver fejlesztők intelligenciája igyekszik pótolni a buta alkalmazásfejlesztőét.
Hogy érthetőbb legyen a HAVING és a WHERE közötti különbség, egy táblázatban összefoglaltam, hogy milyen helyzetben melyik záradékot használhatjuk.

	WHERE	HAVING
Mikor szűr?	A csoportosítás előtt	A csoportosítás után
Mit szűr?	Sorokat	Csoportokat
Tartalmazhat-e aggregáló függvényeket?	Nem	Igen

Nézzünk egy példát, amelynek segítségével közelebb kerülhetünk a GROUP BY és a HAVING szelleméhez. Lássunk egy elég bonyolult lekérdezést, amiben minden benne van, amit eddig tanultunk:

```
SELECT
  p.ProductID, p.ProductName,
  'Amount' = SUM(od.UnitPrice *
   od.Quantity)
FROM
  [Order Details] od
INNER JOIN
  Orders o
ON
  o.OrderID = od.OrderID
INNER JOIN
  Products p
ON
  p.ProductID = od.ProductID
WHERE
  o.OrderDate >= '1998.05.05' AND
  o.OrderDate <= '1998.05.07'
GROUP BY
  p.ProductID, ProductName
HAVING
  SUM(od.UnitPrice * od.Quantity) > 800
ORDER BY
  Amount DESC
```

Az SQL kód magyarra fordítása: készítsünk egy olyan listát, amely az 1998. május ötödike és hetedike közötti megrendelések összértékét listázza ki, termékenkénti bontásban. Csak azokra a termékekre vagyunk kíváncsiak, amelyek

megrendeléseinek összege nagyobb, mint 800 dollár. A lista legyen rendezve az eladási érték alapján, csökkenő sorrendben. Huh, magyarul nehezebb megfogalmazni, mint SQL-ül! Nézzük meg a kimenetét:

```
ProductID ProductName    Amount
64        Wimmers gute   4389
 2        Chang          1178
16        Pavlova         802
```

Néhány szó a szintaktikával kapcsolatban. Az

```
'Amount' = SUM(od.UnitPrice *
   od.Quantity)
```

kifejezés a

```
SUM(od.UnitPrice *
   od.Quantity) AS Amount
```

kifejezéssel egyenértékű. Lehet így is írni, meg úgy is írni. Használjuk az, amelyik olvashatóbb számunkra. A leglustábbak a második formát használják, úgy, hogy még az `AS`-t is elhagyják (én is ilyen vagyok).
Az `ORDER BY`-ban az `Amount` álnevet használhattuk a bonyolult `SUM(od.UnitPrice * od.Quantity)` helyett. Ezt a szintaktikai könnyítést sajnos csak az `ORDER BY`-ban használhatjuk ki, a `HAVING`-ben már nem. Kár.
Az

```
  o.OrderDate >= '1998.05.05' AND
  o.OrderDate <= '1998.05.07'
```

szűrőfeltételt elegánsabban is megfogalmazhatjuk a `BETWEEN` operátor segítségével:

```
  OrderDate BETWEEN
  '1998.05.05' AND '1998.05.07'
```

A két kifejezés logikai értéke azonos.
Gyakori kérés a marketing vagy a pénzügy részéről, hogy olyan összesített statisztikát kérnek, amelyben az eladási adatok napi bontásban láthatók. Mivel a generálódó listát emberek fogják kiértékelni, ezért őket elsősorban az irányvonalak érdeklik, nem pedig az összes részeredmény az utolsó bitig. Így például feltételként szabják, hogy a napi listában csak azok a termékek szerepeljenek, amelyekből több mint egyet rendeltek meg egy adott napon (a nap slágere):

```
SELECT
  OrderDate,
  p.ProductName,
  'Amount' =
  SUM(od.UnitPrice * od.Quantity),
  COUNT(*) AS OrderedProducts
FROM
... --ugyanaz, mint az előző lekérdezésben
WHERE
  OrderDate BETWEEN
  '1998.05.01' AND '1998.05.07'
GROUP BY
  OrderDate, p.ProductID, ProductName
HAVING
  COUNT(*) > 1
ORDER BY
  OrderDate, Amount DESC


OrderDate  ProductName Amount OProd
1998-05-05 Chang       532    2
1998-05-06 Chang       646    2
1998-05-06 Grandma's   525    2
1998-05-06 Tofu        488    2
```

Ebből olyan okosságokra lehet következtetni, hogy a Chang nagyon finom lehet, mert ötödikén és hatodikán is megrendeltek belőle kettőt is! Ennél tovább azonban nem megyünk, ez nem a mi szakmánk.
Szakmai szempontból az utolsó oszlop érdekes a számunkra: `COUNT(*) AS OrderedProducts`. A `COUNT(*)` a többi aggregáló függvényhez hasonlóan másként viselkedik, ha `GROUP BY` van a közelben: nem az egyedi sorokat számolja meg, hanem a csoportokon belüli sorok számát. Másképpen: nem a teljes eredményhalmazra ad egy eredményt, hanem minden egyes csoportra külön-külön. Pont ez az, ami nekünk kellett, és a `HAVING` volt olyan szíves aggregáló függvényt beengedni a feltételek közé. Hurrá!

Beépített skaláris függvények

Mi is az a skaláris függvény? A függvény állatfaj azon alfaja, amely *egy* értékből *egy* értékre képez le. Azaz nem olyan, mint a `SUM` volt, ami sok sor tartalmát összegezve adott vissza egyetlen számot, mert ő az aggregáló típusú függvények képviselője, azaz, amelyek *több* értékből állítanak elő *egy*et. Inkább gondoljunk az abszolút értéket képző `ABS` függvényre (a blokkolásgátló függvény :). `ABS(-4) = 4`. Azaz a mínusz négyet leképezte plusz négyre. Nagy csodák vannak ebben a Serverben!

Az SQL Server nagyon sok beépített, skaláris függvénnyel rendelkezik. Vannak matematikai célúak: abszolút érték (`ABS`), trigonometrikus függvények (`SIN`, `COS`, `TAN`, `COT`, valamint ezek arcus megfelelői), logaritmus függvények (`LOG`, `LOG10`), exponenciális függvény (`EXP`), kerekítések (`ROUND`, `FLOOR`, `CEILING`), véletlen szám generáló függvény (`RAND`) stb. Majdnem olyan gazdag a kínálat, mint más „polgári" nyelvekben, mint pl. a Visual Basic-ben.

Van nagyon sok szövegkezelő függvény: különböző darabolások (LEFT, RIGHT, SUBSTRING), szövegdarabok keresése (PATINDEX, CHARINDEX), kis-nagybetű konvertálók (UPPER, LOWER), szám formátumról szövegre átalakító (STR), kezdő és záró szóközt levágó (LTRIM, RTRIM). Vannak egzotikusabbak is: REVERSE, ami megfordít egy szöveget: „cirmos" → „somric". Van olyan, ami nagyon hasznos lenne, ha magyarul is működne, csak hát a magyar nyelv elég ellenálló a formalizálással szemben: a DIFFERENCE és a SOUNDEX. A DIFFERENCE egy 0-4-es skálán képes megmondani két szövegről, hogy kimondva, hangzásban (!) mennyire hasonlítanak. Nem a karakterláncok írt, hanem kimondott formája. Ez a szolgáltatás nagyon jól jönne például egy telefonkönyv alkalmazásnál, ahol nem lehet tudni, hogy pontosan hogyan írtak egy nevet, de például valahogy úgy hangzott, hogy „socó". A „Smith" és a „Smythe"-re a például DIFFERENCE azt mondja, hogy a távolságuk 4, azaz nagyon hasonlítanak. Az "other" és a „brother" szavak 2 távolságra vannak egymásra, azaz még hasonlítanak, de azért nem annyira. Nagyon jó lenne ez magyarul is! Így talán a Gizike és a gőzeke is kaphatna egy 1-est.

Tovább a függvények útján. Nagyon hasznosak, bár ritkábban használatosak az úgynevezett metaadat-függvények. Ezek a rendszertáblákból kérdeznek le adatokat (manuálisan tilos, mert bármelyik szervizcsomag megváltoztathatja), melynek segítségével belső információkat lehet megtudni az adatbázisunk tulajdonságairól. Ha például az alkalmazásunk kíváncsi, hogy a Employee nevű tábla FirstName nevű oszlopa hány byte-ot foglalhat el az adatbázisban:

```
COL_LENGTH ('Employee', 'FirstName')
```

Ez egy `nvarchar(50)`-es oszlopra 100-at adna eredményül (az `nvarchar` Unicode formátumú, azaz minden karakter 2 bájtot foglal el).

További függvénykategóriákat is találunk még a Serverben, de ezeket terjedelmi okok miatt most nem közöljük. A Books Online-ban részletesen dokumentálva vannak mindannyian.

Végül, de nem utolsó sorban beszéljünk az egyik leghasznosabb függvénycsaládról: a dátumkezelő függvényekről. Ezek megérnek a többinél kicsit több figyelmet.

Dátumzsonglőrködés

Eddig elég egyszerű volt bevetni a `GROUP BY`-t, mivel midig volt egy olyan oszlop, amire természetesen lehetett csoportosítani. Azonban ilyen nem mindig létezik. Például az előző példában az `OrderDate` napra kerekített érték volt, így könnyű volt napra csoportosítani, mivel csak be kellett írni a `GROUP BY`-ba. De mit teszünk, ha heti bontásban várják a kimenetet? Ha nem ismerjük a `DATEPART` függvényt, akkor bajban leszünk. De ha igen, akkor:

```
SELECT
  DATEPART(wk, OrderDate) AS WeekNum,
  'Amount' = SUM(od.UnitPrice *
  od.Quantity),
  COUNT(*) AS OrderedProducts
FROM
...
WHERE
  OrderDate BETWEEN
  '1998.01.01' AND '1998.02.01'
GROUP BY
  DATEPART(wk, OrderDate)
HAVING
  COUNT(*) > 1
ORDER BY
  WeekNum, Amount DESC


WeekNum     Amount        OrderedProducts
1           4691.0000     12
2           30894.6600    30
3           18822.6700    38
4           24330.5400    38
5           22115.8500    34
```

A `DATEPART` függvény az egyik leggyakrabban használt beépített függvény. Visszatér egy egész számmal, ami a `date` paraméterben megadott dátum egy bizonyos darabjának felel meg. A használata nagyon egyszerű:

```
DATEPART(datepart, date)
```

ahol a `datepart` a következő kifejezések valamelyike lehet:

Jelentés	Teljes név	Rövidítés
Év	year	yy, yyyy
Negyedév	quarter	qq, q
Hónap	month	mm, m
Az év n. napja	dayofyear	dy, y
Nap	day	dd, d
Hét	week	wk, ww
A hét n. napja	weekday	dw
Óra	hour	hh
Perc	minute	mi, n
Másodperc	second	ss, s
Ezredmásodperc	millisecond	ms

A függvényben használhatjuk mind a teljes nevet, mind a rövidítést.
A példánk bizony sánta. Mivel minden évben van 1. hét, 2. hét, satöbbi, ezért a lekérdezésünk össze fogja vonni az összes év ugyanazon hetébe eső eladásokat. Ez természetesen nem helyes, és ezt a problémát úgy fogjuk orvosolni az utolsó, szinte tökéletes lekérdezésünkben, hogy a hét sorszáma mellé felsoroljuk az évet is mind a `SELECT` utáni listában, mind a `GROUP BY`-ban, így egyértelműen azonosítva lesz a hét.
Az előző példánkban arra voltunk kíváncsiak, hogy az adott dátum az év hányadik hetébe esik. Azonban az SQL Servert Amerikában írták, ahol a hét első napja a vasárnap, nem pedig a hétfő. Ők tudják, mi a jó nekik, azonban a `DATEPART` is ennek megfelelően működik, aminek mi nem örülünk. Mivel azonban a Microsoft nem csak Amerikában akarja eladni az SQL Servert, ezért beépítette annak lehetőségét, hogy megváltoztassuk a hét első napját:

```
SET DATEFIRST 1
```

Ennek hatására a hét első napja ismét a hétfő lesz, így az összes dátumkezelő függvény helyesen fog működni. Az SQL Serverben sok, a fentihez hasonló beállítás létezik, amelyekkel a szerver alapértelmezett viselkedését változtathatjuk meg. Ezekkel egy későbbi cikkben még részletesen foglalkozunk.
A dátumkezelő függvények további igen hasznos képviselője a `DATEADD` függvény. Ez, mint a neve is sugallja, arra való, hogy egy dátumhoz hozzáad valamilyen időintervallumot. Mi határoz meg egy időintervallumot? A hossza és a mértékegysége. Ennek megfelelően a függvény formátuma a következő:

```
DATEADD (datepart , number, date)
```

A `datepart` az előző táblázatban közölt értéket veheti fel, a `weekday` kivételével, mert annak nincs semmi értelme ebben az összefüggésben. Szerintem a `dayofyear`-nek sincs, de az úgy működik, mintha `day`-t írtunk volna. A `number` egy egész szám, ami az intervallumot írja le. A `date` pedig a kiinduló dátum. Nézzük meg működés közben:

```
SELECT DATEADD(day, 1, '2000/01/05 18:12')
2000-01-06 18:12:00.000

SELECT DATEADD(mi, 5, '2000/01/05 18:12')
2000-01-05 18:17:00.000

SELECT DATEADD(hh, -3, '2000/01/05 18:12')
2000-01-05 15:12:00.000

DECLARE @d DATETIME
SET @d = '2000/01/05 18:12'
SELECT @d + 3
2000-01-08 18:12:00.000
```

Az első három példa morális tanulsága: nincs `DATESUB`, a `DATEADD`-ot kell negatív számmal meghívni.
Az utolsó példa ravasz. Egy dátum típusú mezőhöz hozzáadunk 3-at, egy egész számot, aminek az a jelentése, hogy a dátumot megnöveli 3 nappal. Érdekes, de ha valaki nem tudja explicit a + operátor e polimorf tulajdonságát, az meglepődhet a kódunkon.
A harmadik hasznos dátumkezelő függvény a `DATEDIFF`. Formátuma:

```
DATEDIFF (datepart, startdate, enddate)
```

Azaz a `startdate` és `enddate` dátumok különbségét adja vissza a `datepart`-ben definiált egységben:

```
--Hány másodperc is egy nap?
SELECT DATEDIFF(second, '2000.01.01', '2000.01.02')
86400
```

Dátum agytorna

Szép lett az előző példánk listája, de nekem például nem sokat mond az, hogy most a 38. hétben járunk. A menzán és a hivatalokban gyakran láthatjuk ezt a fajta időmeghatározást, de nekem sokkal szimpatikusabb lenne a lista, ha a hetet jelző szám mellett ott láthatnám azt is, hogy mely dátumhatárok zárják az adott hetet. Próbáljuk meg kitalálni, hogyan lehetne ezt összerakni Transact SQL-ben. Nem lesz triviális, kéretik egy dupla KV-t inni a következők előtt!
Adott a dátumunk, tároljuk ezt a `@d` változóban. Azt, hogy ez a dátum az év hányadik hetébe esik, a `DATEPART(week, @d)` függvénnyel könnyedén megtudhatjuk. Hogyan lesz ebből meg a keresett hét kezdő dátuma? Úgy, hogy valahogyan meg kellene találni az adott év első hétfőjét, és ahhoz hozzá kellene adni annyiszor 7 napot, ahányadik héten járunk az első hétfőhöz képest. Az év eleji tört hét is hétnek számít! Nézzük meg mindezt Transact SQL-ben!

```
--A hét első napja a hétfő legyen
SET DATEFIRST 1
--Ehhez a dátumhoz keressük a hetet és a
--hetet záró határokat
DECLARE @d DATETIME
--Ebben lesz az év első napjának dátuma
--(nem első hétfő, hanem január elseje!)
DECLARE @dFirstDayOfYear DATETIME
--Teszt dátum. Ez egy keddi nap a 2. héten
SET @d = '2000/01/04'

A dátumhoz tartozó hét tesztelése
SELECT DATEPART(week, @d)
2

--Az év első napjának megkeresése
SET @dFirstMondayOfYear = CONVERT(CHAR(4), @d, 112)
SELECT @dFirstDayOfYear
2000-01-01 00:00:00.000
```

Itt álljunk meg egy pillanatra. Mi az a `CONVERT` függvény, és mit jelent a `112`-es paraméter? A `CONVERT` a különböző adattípusok közötti konverzióra való. Különösen akkor hasznos, ha dátum formátumot kell szöveggé konvertálni. Az első paramétere mondja meg, hogy milyen típussá szeretnénk konvertálni. Itt `char(4)`-et adtunk meg, ami 4 karaktert képes tárolni. A második paraméter a konvertálandó kifejezés, a harmadik pedig a konvertált eredmény formátumát szabályozza. Dátum bemenet és szöveg kimenet esetén a `112` azt jelenti, hogy a dátumot `yyyymmdd` formátumra konvertálja át. De hisz az eredmény 8 karakter, mi meg `char(4)`-et adtunk meg! Ez benne a trükk. 2000. 01. 04.-ből 20000104 lenne, de mivel a `char(4)` csak az első 4 karaktert képes eltárolni, a maradék négy egyszerűen elveszik. Azaz mi lesz a konverzió eredménye? ”2000”. De akkor miért kaptunk a `SELECT @dFirstDayOfYear` eredményeként `2000-01-01`-et? Azért, mert a `@dFirstDayOfYear` dátum típusú, és a szerver a „2000” sztringet 2000. január 1-é konvertálta. Implicit módon, azaz anélkül, hogy erre külön megkértük volna. Ez azért egy kicsit piszkos munka volt. Inkább segítsünk neki:

```
SET @dFirstDayOfYear = CONVERT(CHAR(4), @d, 112) + '.01.01'
2000-01-01 00:00:00.000
```

Na, ez így már szép. De menjünk tovább. Hogyan kapjuk meg ebből az első hétfőt? Ha tudjuk, hogy január elseje a hét hányadik napja, akkor ebből már könnyű kiszámolni, hogy az első hétfő hányadikára esik: menjünk vissza az év első napjától annyi napot, ahányadik napra esik az a hétben, és adjunk hozzá 8-at. Például 2000. január elseje szombat volt, ami a hét 6. napja.

```
SELECT DATEPART(weekday, @dFirstDayOfYear)
6
```

Ha visszamegyünk 6 napot, az 1999. december 26-a, ami a 2000. év első hétfőjét megelőző hétfő előtti nap (vasárnap).

```
SELECT DATEADD(day, -DATEPART(weekday, @dFirstDayOfYear), @dFirstDayOfYear)
1999-12-26 00:00:00.000
```

Ehhez már csak hozzá kell adni 8 napot, és meglesz a 2000. év első hétfője.

```
SELECT DATEADD(day, -DATEPART(weekday, @dFirstDayOfYear)+8, @dFirstDayOfYear)
2000-01-03 00:00:00.000
```

Eljutottunk a tárgy év első hétfőjéig. Most már nincs más dolgunk, mint ehhez hozzáadni annyiszor 7 napot, ahányadik héthez keressük a hetet kezdő hétfőt.

```
SELECT DATEADD(day, (-DATEPART(weekday, @dFirstDayOfYear)+8) + (DATEPART(week, @d)-2)*7,
@dFirstDayOfYear)
2000-01-03 00:00:00.000
```

Azért kellett kettőt kivonni a hét számából, mert 1-től kezdődik a hetek számozása és nem 0-ától, valamint, mert az aktuális napot *megelőző* hétfőre vagyunk kíváncsiak, nem pedig a következőre.
A záró napot innentől kezdve gyerekjáték meghatározni, csak nem kettőt, hanem egyet kell kivonni a hetek számából.

```
SELECT DATEADD(day, (-DATEPART(weekday, @dFirstDayOfYear)+8) + (DATEPART(week, @d)-1)*7,
@dFirstDayOfYear)
2000-01-10 00:00:00.000
```

Alakítsuk át a korábbi GROUP BY-os példánkat úgy, hogy a hetek száma mellé legyen kiírva azok kezdete és vége is. Ehhez az előbb kiagyalt kifejezéseket össze kell vonni, és be kell írni a megfelelő helyre a kiinduló lekérdezésben, valamint rakjuk bele az évet is, ahogy korábban ígértük:

```
SET DATEFIRST 1

SELECT

  DATEPART(year, OrderDate) AS YearNum,
  DATEPART(wk, OrderDate) AS WeekNum,
  DATEADD(day, (-DATEPART(weekday,
  CONVERT(CHAR(4), OrderDate, 112) +
  '.01.01')+8) + (DATEPART(week,
  OrderDate)-2)*7, CONVERT(CHAR(4),
  OrderDate, 112) + '.01.01') AS StartDay,

  DATEADD(day, (-DATEPART(weekday,
  CONVERT(CHAR(4), OrderDate, 112) +
  '.01.01')+8) + (DATEPART(week,
  OrderDate)-1)*7, CONVERT(CHAR(4),
  OrderDate, 112) + '.01.01') AS EndDay,

  'Amount' = SUM(od.UnitPrice *
  od.Quantity),
  COUNT(*) AS OrderedProducts
FROM
...
WHERE
  OrderDate BETWEEN
  '1997.12.22' AND '1998.01.18'
GROUP BY
  DATEPART(year, OrderDate),
  DATEPART(wk, OrderDate),

  DATEADD(day, (-DATEPART(weekday,
  CONVERT(CHAR(4), OrderDate, 112) +
  '.01.01')+8) + (DATEPART(week,
  OrderDate)-2)*7, CONVERT(CHAR(4),
  OrderDate, 112) + '.01.01'),

  DATEADD(day, (-DATEPART(weekday,
  CONVERT(CHAR(4), OrderDate, 112) +
  '.01.01')+8) + (DATEPART(week,
  OrderDate)-1)*7, CONVERT(CHAR(4),
  OrderDate, 112) + '.01.01')

ORDER BY
  WeekNum, Amount DESC


YN   WN StartDay   EndDay     Amount OP
1997 52 1997-12-22 1997-12-29 17678  32
1997 53 1997-12-29 1998-01-05 14871  18!
1998 1  1997-12-29 1998-01-05  4691  12!
1998 2  1998-01-05 1998-01-12 30894  30
1998 3  1998-01-12 1998-01-19 18822  38
```

Konklúzió

Látható, hogy az évváltásnál nem jól működik a dátumkezelő algoritmusunk. Ezt még tökéletesíteni fogjuk a következő számban.

És miért lett ennek az egyszerű feladatnak a megoldása ilyen bonyolult, annak ellenére, hogy nem is tökéletes? Azért, mert majdnem ugyanazt a hosszú kódrészletet négyszer egymás után le kellett írnunk, szinte változatlanul. De hát nem azt tanultuk az iskolában, hogy az ismétlődő kódrészeket függvényekbe kell rakni? De! És nem arról regéltek nekünk, hogy a függvények paraméterezésével még a nem teljesen azonos kódrészeket is össze lehet vonni? De, de, de! Akkor miért nem élünk ezzel a lehetőséggel? Microsoft SQL Server 7-ig azért nem, mert nem volt meg a módunk erre, mert nem voltak User Defined Function-ök, felhasználói függvények. De SQL 2000-ben végre vannak, és pont az ilyen problémákra adnak igen elegáns megoldást. De erről majd a következő részben.

Soczó Zsolt MCSE, MCSD, MCDBA
Protomix Rt.