

SQL kulcsszógyűjtemény

#####

Mező orientált -> Rekord orientált

Kell 2 új tábla: attributes (lehetséges oszlopnevek) és a record (a szétszedés utáni tábla)

1/a.: attributes tábla:

```
create table attributes (  
    attrib_id int primary key,  
    attrib_name nvarchar(100),  
    attrib_type nvarchar(100)  
)
```

1/b.: megadni a lehetséges attribútumokat (oszlopnevek a mező-or... táblából)

```
insert attributes (attrib_id, attrib_name, attrib_type) values  
(1, 'Last name', 'text'), (2, 'Title', 'text'), (3, 'City', 'text') --meghatározni melyik attrib_id lesz melyik oszlop
```

2.: records tábla (id, attrib_id, attrib_value (nvarchar) és összetett elsődleges kulcs)

```
create table employee_record (  
    emp_id int not null,  
    attrib_id int not null references attributes (attrib_id),  
    attrib_value nvarchar(500)  
    primary key (emp_id, attrib_id)  
)
```

3.: átalakítás record-orientáltba:

```
insert employee_record (emp_id, attrib_id, attrib_value)  
select employeeid, 1, lastname from employee_field where lastname is not null  
union  
select employeeid, 2, title from employee_field where title is not null  
union  
select employeeid, 3, city from employee_field where city is not null
```

#####

Rekord orientált -> Mező orientált

```
select emp_id as employeeid,  
    min(  
        case --MIN helyén lehetne MAX is, a lényeg, hogy csak csak egy rekordot tartson meg  
            when attrib_id=1 then attrib_value  
            else null  
        end) as lastname, --visszaalakítás a fix attrib_id -k alapján  
    min(  
        case  
            when attrib_id=2 then attrib_value  
            else null  
        end) as title,  
    min(  
        case  
            when attrib_id=3 then attrib_value
```

```

        else null
    end) as city
from employee_record
group by emp_id

```

Visszaalakítás PIVOT-al:

```

select *
from (
    select emp_id, attrib_name, attrib_value
    from employee_record e inner join attributes a on e.attrib_id = a.attrib_id
    ) as forras
    pivot (min(attrib_value) for attrib_name in ([Title], [City], [Last name])) as forras --oszlopnevek

```

#####

Pivot

pivot(AGGREG_FGV_AZ_ÉRTÉKEKHEZ(érték oszlop neve) **for** MELYIK_OSZLOP_ALAPJÁN_SZÉTSZEDNI **in** (ÚJ OSZLOPOK NEVEI)) **as** alias

```

select csapat_nev, nap_nev, pt.alma, pt.szilva
from eredm_pivot
    pivot (sum(leadott_lada) for gyumolcs_nev in (alma, szilva)) as pt
order by csapat_nev, nap_nev

```

	csapat_nev	nap_nev	gyumolcs_nev	leadott_lada
1	Szorgos	hétfő	alma	50
2	Szorgos	kedd	alma	60
3	Szorgos	szerda	alma	70
4	Szorgos	hétfő	szilva	100
5	Szorgos	kedd	szilva	120
6	Szorgos	szerda	szilva	140
7	Lusta	hétfő	alma	5
8	Lusta	kedd	alma	6
9	Lusta	szerda	alma	7
10	Lusta	hétfő	szilva	10
11	Lusta	kedd	szilva	12
12	Lusta	szerda	szilva	14

==>

	csapat_nev	nap_nev	alma	szilva
1	Lusta	hétfő	5	10
2	Lusta	kedd	6	12
3	Lusta	szerda	7	14
4	Szorgos	hétfő	50	100
5	Szorgos	kedd	60	120
6	Szorgos	szerda	70	140

```

select * --pt.csapat_nev, pt.alma, pt.szilva
from (
    select csapat_nev, gyumolcs_nev, sum(leadott_lada) as leadott_lada
    from eredm_pivot
    group by csapat_nev, gyumolcs_nev) as forras
    --az alias (as forras) szintaktikai okból kell, hiába nincs használva, nélküle nem működik
    pivot (sum(leadott_lada) for gyumolcs_nev in ([alma], [szilva])) as pt

```

Belső select:			
	csapat_nev	gyumolcs_nev	leadott_lada
1	Lusta	alma	18
2	Szorgos	alma	180
3	Lusta	szilva	36
4	Szorgos	szilva	360

==>

Pivot (külső select)			
	csapat_nev	alma	szilva
1	Lusta	18	36
2	Szorgos	180	360

#####

Unipivot

```
select *  
from #temp  
    unpivot (leadott_lada for gyumolcs_nev in (alma, szilva)) as upt  
order by csapat_nev, nap_nev, gyumolcs_nev
```

unpivot (EGYESÍTETT_ÉRTÉK_OSZLOP_NEVE for EGYESÍTETT_OSZLOPOK_NEVEINEK_OSZLOPA in (KIINDULÁSI_OSZLOPOK)) as alias

Mentett pivot tábla							Unipivot után				
	csapat_nev	nap_nev	alma	szilva				csapat_nev	nap_nev	leadott_lada	gyumolcs_nev
1	Lusta	hétfő	5	10			1	Lusta	hétfő	5	alma
2	Szorgos	hétfő	50	100			2	Lusta	hétfő	10	szilva
3	Lusta	kedd	6	12			3	Lusta	kedd	6	alma
4	Szorgos	kedd	60	120			4	Lusta	kedd	12	szilva
5	Lusta	szerda	7	14			5	Lusta	szerda	7	alma
6	Szorgos	szerda	70	140			6	Lusta	szerda	14	szilva
							7	Szorgos	hétfő	50	alma
							8	Szorgos	hétfő	100	szilva
							9	Szorgos	kedd	60	alma
							10	Szorgos	kedd	120	szilva
							11	Szorgos	szerda	70	alma
							12	Szorgos	szerda	140	szilva

#####

Procedurális kiterjesztések

Vezérlő szerkezetek:

DECLARE: változók deklarálása. A változók neve @-tel kezdődik.

SET: explicit értékadás változónak

```
declare @i int, @eredm int  
set @i = 1
```

Deklaráláskor értéket is lehet adni:

```
declare @i int = 1, @eredm int = 0
```

Az első értékadásig a változó értéke NULL. A változó csak a kötegen belül látható.

PRINT 'szöveg': szöveg kiírása a konzolra

BEGIN...END: egynél több utasítást tartalmazó utasításblokk

IF/ELSE/ELSE IF: feltételes elágazás

```
if @row_Count > 1 begin  
    print 'Nem jó'  
end
```

WHILE/BREAK/CONTINUE: elől tesztelő ciklus. Másféle ciklus nincs.

```

while @i < 50 begin
    set @eredm = @eredm + @i
    set @i = @i + 1
end

```

RETURN: feltétel nélküli kilépés kötegből, utasításblokkból vagy tárolt eljárásból

`return @i` --tárolt függvénynél vagy eljárásnál lesz jó

TRY/CATCH/THROW/RAISERROR: kivételek kezelése

```

begin try
    if @row_Count > 1 begin
        print 'Több, mint egy sor'
        RAISERROR ('An error occurred.', 16, 1) --hiba dobása a catch ágba
    end
end try
begin catch
    --a hiba táblába rakása
    select
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage
    insert #log (time_stamp, err_num) values (getdate(), ERROR_NUMBER())

    --VAGY csak simán print

    print ERROR_MESSAGE() --alias ide nem kell
end catch

```

A globális `@@error` változó: (hibák típusára lehet szűrni vele)

```

select 1/0
if @@ERROR=8134 print '0-val osztottunk'

```

WAITFOR/DELAY/TIME: várakozás időtartamig vagy időpontig

`waitfor delay '00:00:10'` --10 mp várakozás

GOTO <címke>: másik sorra ugrik (erre nincs példa sehol a jegyzetben)

Változó beolvasás táblából: select-en belül az oszlopokból, szűréssel

```

select @prod_id = productID, @stock = unitsInStock
from products where productName like '%' + @prod_name + '%'

```

aggregáló fgv-eket is lehet használni, ha több rekord van:

```

select

```

```

        @salary=max(Salary),
        @row_Count=count(*),
        @emp_id=max(EmployeeID),
        @first_name=max(FirstName),
        @last_name=max(LastName)
from Employees
where (LastName + ' ' + FirstName) like ('%' + @employeeName + '%')

```

Változó mentése adatbázisba:

```

update Employees set Salary = @newSalary
where EmployeeID = @emp_id

```

#####

Tranzakció kezelés

set xact_abort off: ez az alapértelmezés, a tranzakció nem gördül vissza rollback utasítás nélkül
set xact_abort on: bármi hiba van, a tranzakció automatikusan visszagördül

begin tran: tranzakció kezdése

commit tran: tranzakció végrehajtása

rollback tran: tranzakció visszagörgetése

end tran NINCS!!!, a fenti utasításokat bárhova beírhatjuk, nem kell sehova end tran!
Mert a tran nem blokkindító!

Mivel a tran nem blokkindító, a **@@TRANCOUNT**-al lehet követni, hogy épp milyen mélységű tranzakcióban járunk.

A commit tran és rollback tran mindig csak a legutóbbit görgeti vissza.

Behúzásokkal ábrázolva, hogy a **@@trancount** követhető legyen, de a behúzások nem blokkokat jelölnek:

```

begin tran
    print @@trancount --1
    begin tran
        print @@trancount --2
        commit tran
        print @@trancount --1
    rollback tran
    print @@trancount --0

```

#####

Izoláció kezelés

Zárak:

- Megosztott (shared, S): a SELECT-hez használt, olvasási zár
- Kizárólagos (exclusive, X): az adatmódosító SQL utasításokhoz használt zár. A tranzakció által megszerzett X típusú zárok mindig megőrződnek a tranzakció befejezéséig. Erre a tranzakció logikai egységének biztosítása céljából van szükség (az ún. lost update elkerülése céljából).
- Update (U): ideiglenes zár, melyet az adatmódosításra készülő tranzakció a szűrőfeltétel kiértékelésének idejére szerez meg. Ha a feltétel egy rekordra teljesül, akkor X típusú zárra próbálja alakítani.

Izoláció szintek:

- Read uncommitted (más néven „dirty read”): a SELECT utasítások nem helyeznek el S zárat, ezért egy

másik tranzakcióhoz tartozó X zár nem blokkolja az olvasást.

- Read committed: a tranzakció az olvasás idejére S zárat szerez (ha tud), majd a sikeres olvasás után azonnal megszünteti. Ezért az olvasás után nem blokkolja más tranzakciók módosító utasításait (melyhez X zár elnyerése kell). Bármilyen technológiájú kliens csatlakozik is az adatbázishoz, ez a default izolációs szint.
- Repeatable read: mint az előző, azonban az S zár a tranzakció végéig megmarad, és blokkolja más tranzakciók módosító utasításait.
- Serializable: az S zárok nemcsak az olvasott rekordra, hanem az egész tartományra a tranzakció végéig megmaradnak, ezért más tranzakciók nem tudnak a tartományba rekordokat beszúrni/törölni sem. Tehát a „semmitől” megjelenő vagy eltűnő rekord (az ún. phantom read jelenség) nem fordulhat elő.

Röviden:

Olvasom a rekordot, de bárki módosíthatja közben:

`set transaction isolation level read uncommitted`

Olvasom a rekordot, addig ne módosítsa senki, de ha már nem olvasom, bárki módosíthatja:

`set transaction isolation level read committed`

Olvasom a rekordot és a tranzakció végéig más ne módosítsa:

`set transaction isolation level repeatable read`

Olvasom és írom a rekordot, senki nem nyúlhat hozzá a tranzakció végéig:

`set transaction isolation level serializable`

#####

Kurzorok

`declare KURZOR_NEVE cursor [KURZOR_TULAJDONSÁGAI] KURZOR_TÍPUSA for CÉL_TÉBLA`

`declare cursor_emp cursor fast_forward for --kurzor neve és típusa`

`select employeeid, lastname, address from employees order by lastname`

--a céltábla kiválasztása select-el (where, order by, group by stb..), amiben majd mozog a kurzor

`set @i=1 --sorszámláló`

`open cursor_emp --kurzor megnyitása`

--aktuális sor változókba töltése:

`fetch next from cursor_emp into @oszlop1_valtozoja, @oszlop2_valtozoja...`

--az adott kötegre érvényes (2 go utasítás között) @@fetch_status megmondja, hogy a kurzor a végére ért-e már a forrás táblának. Ha =0, akkor még nem, mehet tovább a ciklus:

`while @@fetch_status = 0 begin`

`set @i = @i + 1 --sorszámláló`

--aktuális sor változókba töltése ismét:

`fetch next from cursor_emp into @oszlop1_valtozoja, @oszlop2_valtozoja...`

`end`

`close cursor_emp`

`deallocate cursor_emp --nem csak close, hanem deallocate is kell!!`

Kurzor típusok:

fast_forward: gyors, csak előre menni képes kurzor. Csak a **fetch next** működik

scroll: szabadon mozgatható kurzor, ekkor a FETCH FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE

Kurzor tulajdonságok (opcionálisan):

global: nem csak egy kötegen belül érhető el, bárholnan vezérelhető (a neve alapján), akár tárolt eljárásban is

local: ez az alapértelmezés, csak kötegen belül elérhető

static: a céltábla le lesz választva, a változásokat nem követi nyomon
dynamic: ha a forrás tábla idő közben változik, akkor a kurzor által lekért sor is

Példák:

```
declare c cursor global dynamic scroll for select LastName from Employees  
declare c cursor scroll for select LastName from Employees
```

```
fetch prior from c into @oszlop1_valtozoja, @oszlop2_valtozoja... --előző sor visszatöltése  
fetch last from c into @oszlop1_valtozoja, @oszlop2_valtozoja... --utolsó sor betöltése  
fetch relative 3 from c into @oszlop1_valtozoja, @oszlop2_valtozoja... --3-al ezutáni sor betöltése  
fetch absolute 5 from c into @oszlop1_valtozoja, @oszlop2_valtozoja... --5. sor betöltése
```

Ablakozás

OVER: új ablak nyitása (ami egy csoport (partíció), tehát használhatóak rajta aggregáló függvények is).
Az ablakok függvényeinek eredményei új oszlopokban jelennek meg.

```
select oszlop1, oszlop2, ..., függvény(oszlopN) over  
  ([partition by particionálási_kifejezés] [order by rendezési_kifejezés] [rows ablak_specifikáció])
```

Habár mindhárom opcionális, de valamelyik(ek) azért kell(enek)!

partition by: sorok alcsoportjának (partíciójának) létrehozása valamilyen feltétel alapján

order by: sorok rendezése a csoporton belül

rows: sorok kiválasztása a jelenlegi sorhoz képest

Egyedi ablakfüggvények, amik használhatóak itt:

FIRST_VALUE, LAST_VALUE(kifejezés) **OVER**...: a rendezett lista/partíció első/utolsó rekordjából számított kifejezés

ROW_NUMBER() **OVER**...: a rendezett lista/partíció minden rekordjához egy futó sorszámmat ad

LAG(kifejezés, [offset], [default]) **OVER**...: a rendezett lista/partíció aktuális rekordja előtti offset számú rekord alapján számítja a kifejezést. Jól használható idősorok elemzéséhez.

PERCENT_RANK() **OVER**...: a rendezett lista/partíció minden rekordjához megadja a rendezési feltétel alapján számított rangot, tehát a relatív pozíciót 0 és 1 közötti szám formájában

Példák:

```
select  
  year(orderdate) as ev,  
  row_number() over  
    --készíteni egy partíciót (elemek halmazát) a dátumok évszámai alapján  
    (partition by year(orderdate) order by orderdate) as sorszam, --éven belüli futó sorszám --az "as"  
    elhagyható, de így jobban érthető  
  cast(orderdate as date) as datum,  
  value as osszeg, --a value nem egy spec változó, hanem az oszlop neve  
  sum(value) over  
    --éven belüli kumulált összeg. "rows between unbounded preceding and current row" -> ez a  
    kifejezés sokat elmond  
    (partition by year(orderdate) order by orderdate rows between unbounded preceding and current  
    row) as evs_kumulalt_osszeg,  
  avg(value) over  
    --a 3 megelőző és 3 következő (összesen 7 sor) átlaga.  
    --Pl. 4. sor mozgó átlaga: 822,62 = (1176 + 2376,8 + 2990 + 121,6 + 608 + 424 + 1614,8) / 7  
    --itt nincs partition by, mert a partíciót a sorok helyzete jelöli ki  
    (order by orderdate rows between 3 preceding and 3 following) as mozgo_atlag_osszeg,
```

```

sum(value) over
  (partition by year(orderdate) ) as eves_total, --az év teljes összege
avg(value) over
  (partition by year(orderdate)) as eves_atlag_total, --adott év teljes átlaga
avg(value) over
  (partition by year(orderdate) order by orderdate rows between unbounded preceding and current
  row) as eves_mozgo_atlag --adott év mozgó átlaga (átlag változása minden új beérkezett
  termékkel)
from #tmp

```

```
select * from #tmp
```

	orderdate	value
1	1996-07-25 00:00:00.000	1176,00
2	1996-08-09 00:00:00.000	1200,80
3	1996-08-14 00:00:00.000	613,20
4	1996-09-02 00:00:00.000	121,60
5	1996-09-09 00:00:00.000	608,00
6	1996-09-17 00:00:00.000	424,00
7	1996-09-23 00:00:00.000	1614,80
8	1996-09-24 00:00:00.000	182,40
9	1996-10-11 00:00:00.000	1810,00
10	1996-10-28 00:00:00.000	3354,0001
11	1996-11-04 00:00:00.000	2924,80
12	1996-11-29 00:00:00.000	1689,7801
13	1996-12-11 00:00:00.000	863,2799
14	1996-12-19 00:00:00.000	1228,80
15	1996-12-24 00:00:00.000	1440,00
16	1996-12-30 00:00:00.000	2505,6001
17	1997-01-03 00:00:00.000	1591,2499
18	1997-01-07 00:00:00.000	1194,00
19	1997-01-14 00:00:00.000	224,83
20	1997-01-22 00:00:00.000	49,80

ablakkal:

	ev	sorszam	datum	osszeg	eves_kumulalt_osszeg	mozgo_atlag_osszeg	eves_total	eves_atlag_total	eves_mozgo_atlag
1	1996	1	1996-07-25	1176,00	1176,00	777,90	21757,0602	1359,8162	1176,00
2	1996	2	1996-08-09	1200,80	2376,80	743,92	21757,0602	1359,8162	1188,40
3	1996	3	1996-08-14	613,20	2990,00	690,60	21757,0602	1359,8162	996,6666
4	1996	4	1996-09-02	121,60	3111,60	822,6285	21757,0602	1359,8162	777,90
5	1996	5	1996-09-09	608,00	3719,60	680,6857	21757,0602	1359,8162	743,92
6	1996	6	1996-09-17	424,00	4143,60	767,7142	21757,0602	1359,8162	690,60
7	1996	7	1996-09-23	1614,80	5758,40	1159,2571	21757,0602	1359,8162	822,6285
8	1996	8	1996-09-24	182,40	5940,80	1559,7143	21757,0602	1359,8162	742,60
9	1996	9	1996-10-11	1810,00	7750,80	1714,2543	21757,0602	1359,8162	861,20
10	1996	10	1996-10-28	3354,0001	11104,8001	1777,0085	21757,0602	1359,8162	1110,48
11	1996	11	1996-11-04	2924,80	14029,6001	1721,8657	21757,0602	1359,8162	1275,4181
12	1996	12	1996-11-29	1689,7801	15719,3802	1901,5228	21757,0602	1359,8162	1309,9483
13	1996	13	1996-12-11	863,2799	16582,6601	2000,8943	21757,0602	1359,8162	1275,5892
14	1996	14	1996-12-19	1228,80	17811,4601	1749,0728	21757,0602	1359,8162	1272,2471
15	1996	15	1996-12-24	1440,00	19251,4601	1501,8157	21757,0602	1359,8162	1283,4306
16	1996	16	1996-12-30	2505,6001	21757,0602	1292,5371	21757,0602	1359,8162	1359,8162
17	1997	1	1997-01-03	1591,2499	1591,2499	1176,3257	70444,1401	1718,1497	1591,2499
18	1997	2	1997-01-07	1194,00	2785,2499	1227,0685	70444,1401	1718,1497	1392,6249
19	1997	3	1997-01-14	224,83	3010,0799	1043,64	70444,1401	1718,1497	1003,3599
20	1997	4	1997-01-22	49,80	3059,8799	975,4114	70444,1401	1718,1497	764,9699

#####

Tárolt eljárások

Tárolt eljárás eldobása, ha módosítani akarjuk:

```
drop procedure if exists ELJÁRAS_NEVE
```

Tárolt eljárás létrehozása (nem fut le, csak elmentődik, hogy van ilyen):

```
create procedure ELJÁRAS_NEVE
```

```
@employeeName nvarchar(40), --paraméterek megadása, 1. paraméter
```

```
@salaryModificationPercent float as --paraméterek megadása, 1. paraméter. Az "as" után kezdődik az eljárás!
```

```
begin try --az eljárás törzse
```

```
--bármilyen kód
```

```
end try
```

```
begin catch
```

```
--bármilyen kód
```

```
end catch
```

Tárolt eljárás meghívása:

```
execute ELJÁRAS_NEVE 'Davolio Nancy', 110 --2 átadott paraméter a végén
```

Rövidítések:

proc: procedure

exec execute

Visszatérési érték: output kulcsszóval

```
create procedure ELJÁRAS_NEVE
```

```
@employeeName nvarchar(40), --paraméterek megadása, 1. paraméter
```



```
@salary float output as --output kulcsszó a változó típusa után!
begin
    --visszatérés a dolgozó salary értékével
    select @salary=salary from Employees where LastName = @employeeName
end
```

Meghívásnál:

```
execute ELJÁRAS_NEVE 'Davolio Nancy', @salary = @finalValue output --output: @finalValue változó
értéke változni fog, ahogy a tárolt eljárásan belül értéket kap
```

Saját függvények

Beépített T-SQL függvények:

A T-SQL beépített skalár értékű függvényei:

- Sztring-kezelő függvények, például LEFT, LEN, CHARINDEX, CONCAT, SUBSTRING, LTRIM, REPLACE, UPPER stb.
- Dátum-kezelő függvények, például GETDATE, DATEPART, DAY, MONTH, YEAR, DATEDIFF, DATEADD, ISDATE stb.
- Matematikai függvények, például ABS, ROUND, EXP, RAND, LOG stb.
- Konverziós függvények, például CAST, CONVERT, TRY_CAST, TRY_CONVERT, PARSE
- Logikai függvények: IIF, CHOOSE
- Analitikai és rang-függvények, melyek tipikusan több mint egy rekordot dolgoznak fel, például FIRST_VALUE, LEAD, ROW_NUMBER, RANK
- Egyéb rendszerfüggvények, például @@IDENTITY, @@TRANCOUNT, ERROR_NUMBER, NEWID, HOST_NAME stb.

A T-SQL beépített aggregáló függvényei:

Egy értékhalmoz alapján egyetlen értéket adnak vissza, a NULL értékek figyelembe nem vételével (kivéve a COUNT függvényt). Az aggregáló függvényeket OVER nélkül csak a SELECT listában és a HAVING után lehet meghívni: GROUP BY használata esetén az alcsoportokra működnek.

- A legfontosabb aggregáló függvények a SUM, MIN, MAX, COUNT, AVG, VAR, COUNT, STDEV.
- Ablakozásnál használható függvények még: (OVER előtt) FIRST_VALUE, LAST_VALUE, ROW_NUMBER

De ezeken felül definiálhatunk sajátokat is:

Függvény eldobása, ha módosítani vagy törölni akarjuk:

```
drop function if exists FÜGGVÉNY_NEVE
```

Függvény létrehozása (nem fut le, csak elmentődik, hogy van ilyen):

Vannak megkötések, amiket be kell tartani:

- Visszatérési érték: egyetlen visszaadott érték, amely lehet skalár érték vagy tábla is lehet
- Tartalom: tartalmazhat belül deklarált változókat, lokális kurzorokat, vezérlési szerkezeteket, más függvények hívását, de nem tartalmazhat olyan kódot, mely módosítja az adatbázis tartalmát, vagy bármilyen mellékhatása van:
 - UPDATE, INSERT, DELETE utasításokat
 - Nem tartalmazhatja olyan függvények hívását, melyek nem determinisztikusak, például RAND, GETDATE.
 - A felhasználó által készített tárolt eljárások sem hívhatók.
 - Hibakezelés (TRY/CATCH illetve RAISERROR) használata sem megengedett függvényekben

1. eset: szimpla visszatérési érték

Szintaxis:

```

create function FÜGGVÉNY_NEVE (@datum datetime) --paraméter megadás itt, zárójelekben,
nem az "as" előtt!
returns nvarchar(50) as --visszatérési érték returns, s-el a végén!!, majd "as"
begin --mindig begin-el kezdődik
    declare @eredm nvarchar(50) = 'Valamilyen eredmény'
    return @eredm --egy visszatérési érték kell!      --return, s nélkül!!
end --és a return után mindig egy end-el végződik

```

Függvény meghívása: (mint a többi függvényt, de kell elé a "dbo." megnevezés)

```
select dbo.FÜGGVÉNY_NEVE(BirthDate) from Employees
```

2. eset: tábla visszatérési érték: "tábla függvények"

Szintaxis:

```

create function TÁBLA_FÜGGVÉNY_NEVE (@kat_id int)
returns table as --nem kell meghatározni a visszatérési típust, mert az SQL server ki tudja
következtetni
--nem szükséges sem a begin, sem az end
return (
    select *
    from ... inner join ... on ... = ...
    where ...
    group by ...
) --a visszatérési érték egy jól meghatározott tábla, nem kell visszatérési értéket megadni

```

Függvény meghívása: (a tábla használható a from után. Itt is kell a "dbi." tartomány kijelölés)

```

select * from dbo.TÁBLA_FÜGGVÉNY_NEVE(BirthDate)
where ...
order by ...
group by ...

```

Triggerek

Insert, update és delete esetén futnak le. Táblánként kell őket definiálni. Alapesetben a 3 előbbi művelet után, de konfigurálható úgy is, hogy előttük fusson le.

Egy táblán több, akár azonos típusú trigger is lehet, ebben az esetben a végrehajtásuk sorrendjét különféle szabályok alapján állapítják meg.

Hibakeresés céljára a triggerok tartalmazhatnak PRINT utasításokat, azonban visszatérési értékük nincs, ahogy nincsenek bemeneti paraméterek sem. Ha egy triggerben hiba történik, akkor a trigger kiváltó SQL utasítás is visszagördül.

Speciális, használható táblák a triggeren belül:

- „inserted” nevű tábla: az új rekordok, INSERT és UPDATE triggerok esetén. Ne feledjük, hogy egyetlen UPDATE, DELETE, INSERT utasítás több rekordot is módosíthat!
- „deleted” nevű tábla: a régi rekordok, DELETE és UPDATE triggerok esetén

Trigger eldobása, ha törölni vagy módosítani akarjuk:

```
drop trigger if exists TRIGGER_NEVE
```

Szintaxis:

```

create trigger TRIGGER_NEVE on TÁBLA_NEVE after insert as --after, before --insert, update, delete
begin -- begin-el kezdődik
    declare @i int
    select @i=count(*) from inserted --speciális tábla, ami a triggerben elérhető
    print 'Beszúrt rekordok száma: ' + cast(@i as varchar(50))

```

```
update VALAMI_AKÁR_MÁSIK_TÁBLA_NEVE set VALAMI_OSZLOP
  where VALAMI_OSZLOP in (select Id from inserted) --csinálható egy Id lista a módosított
    rekordokból
end -- end-el végződik
```

Tranzakció kezelés nem kell a triggeren belülre, mert az egész kiváltó tranzakció visszagörgetődik, ha a triggerben hiba van.