

SQL Server 2008 Change Tracking

A Change Tracking (CT) egy új változáskövetési technológia az SQL Server 2008-ban, amely segítségével egyszerűen lehet információkat gyűjteni és kinyerni a felhasználói táblák változásáról. Alapvető építőelem lehet olyan szinkronizációs megoldások kifejlesztéséhez, amelyet valami oknál fogva nem tudunk lefedni a beépített replikációval, például valamilyen közvetlenül nem támogatott adatbázissal kell adatokat cserélni, vagy nagyon más a tábláink és a másik rendszer sémája. A CT szinkron módszer, azaz hasonló, mint amit saját kezűleg triggerekkel tudnánk írni, azonban jóval gyorsabb annál. Kb. annyi költsége van, mintha egy új nonclustered indexet raknánk fel a táblára, ez pedig tényleg nem sok. A bekapcsolása nem változtatja meg a táblák szerkezetét, teljesen láthatatlan kifelé. Oszlopszinten, azaz finom felbontásban jegyzi a változásokat, nem soronként. Ez nyilván fontos a szinkronizációs megoldás számára, ha minimalizálni szeretné a hálózati forgalmat, illetve csökkenteni a módosítási ütközések valószínűségét.

Változáskövetés kézi hajtánnyal v1.0

Hogy jobban megértsük, mit tud a CT, illetve milyen lehetőségek vannak az SQL Server 2008 előtti fejlesztőknek, nézzük meg nagy vonalakban, hogyan lehet követni és lekérdezni a változásokat egyedi megoldással.

Első nekifutásra elégedjünk meg az új sorok és a módosítások kezelésével, a törlések kezelése macerásabb.

A szinkronizálendő táblák elsődleges kulcsa nem a megszokott `int`, `identity`, hanem `guid` (`uniqueidentifier`), generált, egyedi értékkel. Azaz nem generált egész szám, mert ha a szinkron megoldásunkat szeretnénk több résztvevőssé és kétirányúvá tenni, akkor több helyen is fognak generálódni kulcsok, amelyek ütköznének, hisz az `identity` csak az adott gép, adott tábláján lokálisan generál egyedi sorszámokat. Lehetne játszani összetett kulcsokkal, ahol az egyik tag egy `identity`-s oszlop, a másik egy gépazonosító, de szerintem nem érdemes ezzel bonyolítani a rendszert, mert akkor minden, a kulcsra vonatkozó join is többszörös lesz, nehezen kezelhetőek lesznek a lekérdezések. Ráadásul `guid`-ot nem csak az adatbázis tud generálni, hanem az adatelérő réteg is, ami plusz előny lehet sok esetben (akinek fáj, hogy nincs sequence generátor az SQL Serverben, az gondoljon erre, ezzel sok esetben kiváltható).

Jelen példában a `guid` generálását a `newsequentialid` függvényre bízom, ami olyan véletlenszerű értékeket állít elő, amelyek eleje viszonylag folytonos, így gyorsabbak lesznek az indexelt műveletek, mint egy `newid`-val generált teljesen véletlenszerű számnál. Bár a szerver oldalon generálok `guid`-ot, ha a kliens nem ad meg egyet explicit, de ettől még meg lehet adni kézzel is értéket, a default ekkor nem fut le.

A sorok verzióját egy `rowversion` nevű típus jegyzi. `Timestamp` néven is fut, de nem szeretem ezt a nevet, mert sokan ebből időre asszociálnak, miközben ennek nincs köze a fizikai időhöz, maximum egyfajta logikai időnek fogható fel. A `rowversion` adatbázisban egyedileg generált sorszám. Tehát

nem táblánként egyedi, mint az `identity`, hanem adatbázisonként. Az `identity`hez hasonlóan nem lehet neki explicit értéket adni, hanem magától generálja az adatbáziskezelő. Egy sor beszúrásakor kap egy egyedi értéket, majd minden egyes sormódosítás automatikusan lépteti az értékét. A háttérben a `rowversion` egy 8 bájtos bináris típus. A B gépen, amire szinkronizálunk nem `rowversion`ként, hanem `binary(8)`-ként vettem fel a verziót tartalmazó oszlopot, mivel a célhelyen már nem generáljuk az értéket, csak letároljuk azt, ami az elsődleges adatforrásról jött.

Az első oszlop a forrás gépen végrehajtandó parancsokat mutatja, a második a célgépen, amire szinkronizálni akarunk, az idő lefelé telik a táblázatban.

"A" gép	"B" gép
<pre>create table Gyumolcs (id uniqueidentifier not null primary key default (newsequentialid()), nev nvarchar(100) not null, verzio rowversion not null)</pre>	<pre>create table Gyumolcs2 (id uniqueidentifier not null primary key, nev nvarchar(100) not null, verzio binary(8) not null) create table SzinkronMetaadat (tablaNev sysname not null, utolsoVerzio binary(8) not null)</pre>
<pre>insert Gyumolcs (nev) values (N'alma') insert Gyumolcs (nev) values (N'körte') insert Gyumolcs (nev) values (N'barack')</pre>	<pre>insert SzinkronMetaadat (tablaNev, utolsoVerzio) values ('Gyumolcs2', 0x)</pre>
	<pre>create proc Szinkronizalas as begin tran begin try --Egyszer lehozzuk az összes változást, --hogy ne kelljen 2x a hálózaton --keresztülvinni a sorokat. declare @t table (id uniqueidentifier not null primary key, nev nvarchar(100) not null, verzio binary(8) not null) insert @t(id, nev, verzio) select id, nev, verzio from Gyumolcs where verzio > ISNULL((select utolsoVerzio from SzinkronMetaadat where tablaNev = 'Gyumolcs2'), 0x) insert Gyumolcs2 (id, nev, verzio) select id, nev, verzio from @t where id not in (select id from Gyumolcs2) update Gyumolcs2 set nev = t.nev, verzio = t.verzio from Gyumolcs2 gy join @t t</pre>

	<pre>on gy.id = t.id update SzinkronMetaadat set utolsoVerzio = (select MAX(verzio) from Gyumolcs2) where tablaNev = 'Gyumolcs2' commit --Lássuk, mit kaptunk select * from Gyumolcs2 select * from SzinkronMetaadat end try begin catch print 'Baj van' rollback end catch</pre>																
<pre>select * from Gyumolcs</pre> <table><tr><th>id</th><th>nev</th><th>verzio</th></tr><tr><td>CB909BC3-9322-DE11-9D95-002170CE91AB</td><td>alma</td><td>0x000000000000007E5</td></tr><tr><td>CC909BC3-9322-DE11-9D95-002170CE91AB</td><td>körte</td><td>0x000000000000007E6</td></tr><tr><td>CD909BC3-9322-DE11-9D95-002170CE91AB</td><td>barack</td><td>0x000000000000007E7</td></tr></table>	id	nev	verzio	CB909BC3-9322-DE11-9D95-002170CE91AB	alma	0x000000000000007E5	CC909BC3-9322-DE11-9D95-002170CE91AB	körte	0x000000000000007E6	CD909BC3-9322-DE11-9D95-002170CE91AB	barack	0x000000000000007E7					
id	nev	verzio															
CB909BC3-9322-DE11-9D95-002170CE91AB	alma	0x000000000000007E5															
CC909BC3-9322-DE11-9D95-002170CE91AB	körte	0x000000000000007E6															
CD909BC3-9322-DE11-9D95-002170CE91AB	barack	0x000000000000007E7															
	<pre>select * from SzinkronMetaadat</pre> <table><tr><th>tablaNev</th><th>utolsoVerzio</th></tr><tr><td>Gyumolcs2</td><td>0x0000000000000000</td></tr></table>	tablaNev	utolsoVerzio	Gyumolcs2	0x0000000000000000												
tablaNev	utolsoVerzio																
Gyumolcs2	0x0000000000000000																
	<pre>exec Szinkronizalas</pre> <table><tr><th>id</th><th>nev</th><th>verzio</th></tr><tr><td>CB909BC3-9322-DE11-9D95-002170CE91AB</td><td>alma</td><td>0x000000000000007E5</td></tr><tr><td>CC909BC3-9322-DE11-9D95-002170CE91AB</td><td>körte</td><td>0x000000000000007E6</td></tr><tr><td>CD909BC3-9322-DE11-9D95-002170CE91AB</td><td>barack</td><td>0x000000000000007E7</td></tr></table> <table><tr><th>tablaNev</th><th>utolsoVerzio</th></tr><tr><td>Gyumolcs2</td><td>0x000000000000007E7</td></tr></table>	id	nev	verzio	CB909BC3-9322-DE11-9D95-002170CE91AB	alma	0x000000000000007E5	CC909BC3-9322-DE11-9D95-002170CE91AB	körte	0x000000000000007E6	CD909BC3-9322-DE11-9D95-002170CE91AB	barack	0x000000000000007E7	tablaNev	utolsoVerzio	Gyumolcs2	0x000000000000007E7
id	nev	verzio															
CB909BC3-9322-DE11-9D95-002170CE91AB	alma	0x000000000000007E5															
CC909BC3-9322-DE11-9D95-002170CE91AB	körte	0x000000000000007E6															
CD909BC3-9322-DE11-9D95-002170CE91AB	barack	0x000000000000007E7															
tablaNev	utolsoVerzio																
Gyumolcs2	0x000000000000007E7																
<pre>update Gyumolcs set nev = 'sargabarack' where nev = 'barack'</pre>																	
	<pre>exec Szinkronizalas</pre> <table><tr><th>id</th><th>nev</th><th>verzio</th></tr><tr><td>CB909BC3-9322-DE11-9D95-002170CE91AB</td><td>alma</td><td>0x000000000000007E5</td></tr><tr><td>CC909BC3-9322-DE11-9D95-002170CE91AB</td><td>körte</td><td>0x000000000000007E6</td></tr><tr><td>CD909BC3-9322-DE11-9D95-002170CE91AB</td><td>sargabarack</td><td>0x000000000000007E8</td></tr></table> <table><tr><th>tablaNev</th><th>utolsoVerzio</th></tr><tr><td>Gyumolcs2</td><td>0x000000000000007E8</td></tr></table>	id	nev	verzio	CB909BC3-9322-DE11-9D95-002170CE91AB	alma	0x000000000000007E5	CC909BC3-9322-DE11-9D95-002170CE91AB	körte	0x000000000000007E6	CD909BC3-9322-DE11-9D95-002170CE91AB	sargabarack	0x000000000000007E8	tablaNev	utolsoVerzio	Gyumolcs2	0x000000000000007E8
id	nev	verzio															
CB909BC3-9322-DE11-9D95-002170CE91AB	alma	0x000000000000007E5															
CC909BC3-9322-DE11-9D95-002170CE91AB	körte	0x000000000000007E6															
CD909BC3-9322-DE11-9D95-002170CE91AB	sargabarack	0x000000000000007E8															
tablaNev	utolsoVerzio																
Gyumolcs2	0x000000000000007E8																
<pre>insert Gyumolcs (nev) values (N'dinnve')</pre>																	

	exec Szinkronizalas		
	id	nev	verzio
	CB909BC3-9322-DE11-9D95-002170CE91AB	alma	0x00000000000007E5
	CC909BC3-9322-DE11-9D95-002170CE91AB	körte	0x00000000000007E6
	CD909BC3-9322-DE11-9D95-002170CE91AB	sargabarack	0x00000000000007E8
	84EF610B-9522-DE11-9D95-002170CE91AB	dinnye	0x00000000000007E9
	tablaNev		utolsoVerzio
	Gyumolcs2		0x00000000000007E9

A szinkronizáláshoz a B gépen nyilvántartjuk, hogy melyik táblához melyik volt az utolsó verzió. A forrástáblából leszűrjük az újabb verziójú sorokat, amelyek vagy frissek, beszúrtak, vagy módosítottak. A kétféle esetet le tudjuk kezelni a céloldalon, összepárosítva a lehozott változások tábla tartalmát a céltáblával, az elsődleges kulcsok mentén. Egy feltételezéssel élünk csak, az elsődleges kulcsot nem szabad módosítani. Ez adatbázis-elméletileg sem rossz, nem nagy kompromisszum.

A változásokat azért rakom be egy tábla típusú változóba, hogy ne kelljen az insert és az update miatt kétszer lekérdezni ugyanazt a halmazt a forrásgépről. Az insert létrehozza a hiányzó sorokat, az update pedig megmódosítja, ami a forrás oldalon is módosult. Ezek után már csak le kell jegyeznünk, mi volt az utolsó sorverzió a táblához, hogy a következő szinkront innen folytassuk. A módosító műveletek tranzakcióban futnak, hiba esetén mindent visszagörgetünk.

Változáskövetés kézi hajtánnyal v2.0

Az előző megoldás jó volt arra, hogy szemléltesse a `rowversion` típus használatát, de valójában **használatatlan**. Miért? A gond az, hogy attól, hogy mi szinkronizálunk még nem áll le az élet a forrás adatbázison, folyamatosan módosíthatják azt. Nekünk úgy kell felépíteni a szinkronizációs rendszerünket, hogy a szinkronizálás közben történt változásokat már ne vigye le, azokat majd a következő szinkron fogja. A szinkron kezdetén konzisztensnek tekintett adathalmazt szeretnénk leszűrni. Azaz a szűrésben nem csak a múlt felől kell megfogni a még le nem hozott változtatásokat, de a jövő felől is, azaz nem szabad kiengednünk azokat, amely a szinkron kezdete óta módosulnak. Ahhoz, hogy ezt hatékonyan meg tudjuk oldani az adatbázisnak kell segíteni, neki meg kell tudni adni az utolsó generált `rowversion` értékét. Erre szolgál a @@DBTS.

Az előbb példánk végén a `select @@DBTS 0x00000000000007E9` értéket adna vissza. A szinkronizálás ötlete a következő:

1. Lekérdezzük a @@DBTS értékét (anchor verzió).

2. Leszűrjük az adatokat, alsó határként megadva a kliens oldalon letárolt legutolsó anchor verzió értékét, felső határként pedig az előbb lekért anchor verziószámot.
3. Megismételjük 2-t minden táblára
4. Letároljuk az anchor verziót a következő szinkron alapvonalaként.

A megoldás kicsit egyszerűbb is, mint az előző, mert nem kell táblánként külön nyilvántartani a verziókat, csak egy fő verziót, az anchort kell letárolni. Bemásolom ide a komplett, módosított megoldást, hogy ha valaki futtatni akarja a gépén, egyben legyen az egész:

```
--A
use A
go
drop table Gyumolcs
go
create table Gyumolcs
(
    id uniqueidentifier not null
    primary key
    default (newsequentialid()),
    nev nvarchar(100) not null,
    verzio rowversion not null
)
go
--B
use B
go
drop table Gyumolcs2
go
create table Gyumolcs2
(
    id uniqueidentifier
    not null primary key,
    nev nvarchar(100) not null,
    verzio binary(8) not null
)
go
drop table SzinkronMetaadat
go
create table SzinkronMetaadat
(
    utolsoVerzio binary(8) not null
)
go
insert SzinkronMetaadat
(utolsoVerzio) values (0x)

--A
use A
go
insert Gyumolcs (nev)
values (N'alma')
insert Gyumolcs (nev)
values (N'körte')
insert Gyumolcs (nev)
values (N'barack')

--B
use B
go
```

```

select * from Gyumolcs2
select * from SzinkronMetaadat

go
drop procedure Szinkronizalas
go

--Szinkron, ez hívjuk majd többször is
create proc Szinkronizalas
as
begin tran
begin try
    --Egyszer lehozzuk az összes változást,
    --hogy ne kelljen 2x a hálózaton
    --keresztülvinni a sorokat.
    declare @t table (
        id uniqueidentifier
        not null primary key,
        nev nvarchar(100) not null,
        verzio binary(8) not null)

    declare @anchor binary(8)
    exec sp_executesql
        @stmt = N'use A; select @a = @@DBTS',
        @params = N'@a binary(8) output',
        @a = @anchor output

    insert @t(id, nev, verzio)
    select id, nev, verzio from A..Gyumolcs
    where verzio >
        (select utolsoVerzio
        from SzinkronMetaadat)
    and verzio <= @anchor

    insert Gyumolcs2 (id, nev, verzio)
    select id, nev, verzio from @t
    where id not in
        (select id from Gyumolcs2)

    update Gyumolcs2
    set nev = t.nev, verzio = t.verzio
    from Gyumolcs2 gy
    join @t t
    on gy.id = t.id

    update SzinkronMetaadat
    set utolsoVerzio = @anchor

    print 'ok'
    commit
    --Lássuk, mit kaptunk
    select * from Gyumolcs2
    select * from SzinkronMetaadat
end try
begin catch
    print 'Baj van'
    rollback
end catch
--Szinkron vége
go

--A
use A

```

```

go

select * from Gyumolcs

--B
use B
go
select * from Gyumolcs2
select * from SzinkronMetaadat
exec Szinkronizalas

--A
use A
go
update Gyumolcs
set nev = 'sargabarack'
where nev = 'barack'

--B
use B
go
exec Szinkronizalas

--A
use A
go
insert Gyumolcs (nev)
values (N'dinnye')

--B
use B
go
exec Szinkronizalas

```

Figyeljük meg, hogy a SzinkronMetaadat tábla már csak egy értéket tárol, az utoljára szinkronizált verziószámot. A demóban a @@DBTS lekérése azért ilyen körülményes, mert a szinkronizáló eljárás a B adatbázisban fut, nekünk meg az A-ban kell végrehajtani a @@DBTS lekérdezését. A valóságban ez egy sima select lenne, hisz a szinkronizáló programunkból úgyis lenne külön kapcsolat a két oldal felé.

A táblaszűrés így néz ki:

```

where verzio > (select utolsoVerzio from SzinkronMetaadat)
and verzio <= @anchor

```

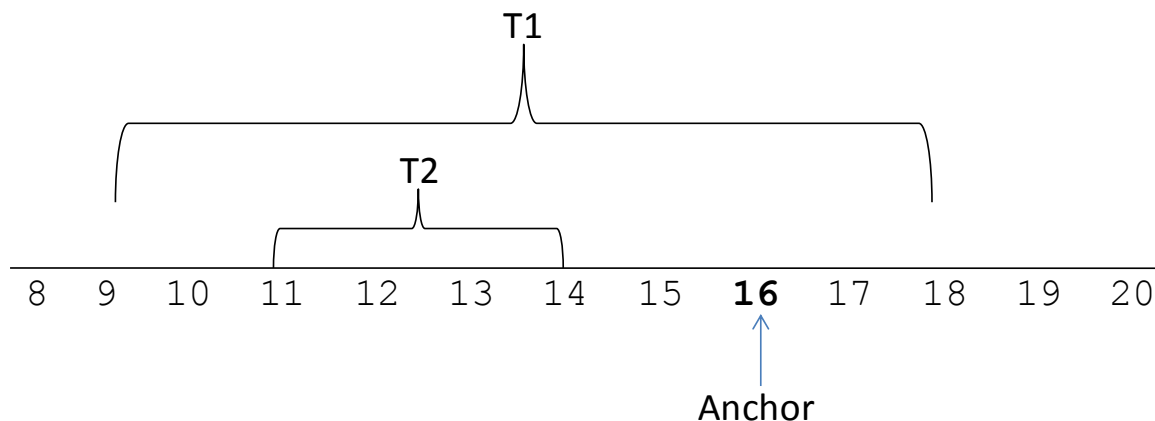
Azaz kérjük azokat az adatokat, amelyek frissebbek, mint a legutóbbi szinkron során lehozott rowversion, ez a rész a SzinkronMetaadatban van letárolva, és emellett régebbiek vagy egyenlők, mint a szinkron kezdetén lekért anchor verzió, azaz nem kérjük a szinkron kezdete után módosult adatokat.

Változáskövetés kézi hajtánnyal v3.0

Az új megoldás sajnos még mindig nem tökéletes. A probléma oka az, hogy a @@DBTS az identityhez hasonlóan osztja a sorszámot tekintet nélkül arra, hogy az őt életre hívó tranzakciót a végén véglegesítik vagy visszavonják. Ennek nyilvánvalóan az az oka, hogy így nem torlódnak a tranzakciók, amíg az egyik

vakaródnak, hogy most akkor mi is legyen a vége, addig a többi, párhuzamosan futó folyamat boldogan használja a később kiosztott új sorszámokat.

Az előbbi szinkronizációs sémákra azonban ez végzetes következményekkel jár, egyes változások lemaradhatnak. Képzeljük el az 1. ábrán látható helyzetet:



1. ábra: nyitott tranzakció okozta problémák szinkronizációs megoldásokban

A T1-es tranzakció elindult, módosított adatokat, így a @@DBTS verziószám felment 8-ról 10-re. Itt elindult egy párhuzamos tranzakció, amely 11-14-ig használt el verziószámokat, majd commitálja, véglegesíti a változásait. A T1 folytatja a tevékenységét, közben jön a kliens szinkronizálni. Lekéri a @@DBTS-t, visszakap 16-ot. Leszűri az adatokat. A T1-es nyitott tranzakció adatait nem látja, hisz read committed izolációs szinten nem látszanak a nyitott tranzakcióban módosított adatok. Ha a read committed snapshot izoláció nincs bekapcsolva, akkor a szinkronizáció kénytelen várni, míg T1 eldönti, mit akar. Tegyük fel T1 18-as verzió után véglegesíti a tranzakciót. Ekkor a szinkronizációs folyamat elviszi T1 9-10-es változásait, de nem viszi el a 17-18-at, csak a következő szinkronkor. Azaz egy tranzakció részben kerül átvitelre, ami vagy konzisztens, vagy nem, ez a kliens logikájától függ, általában nem. A mostani rendszerünk nem alkalmas tranzakciók mentén szinkronizálni, nem véletlenül találták ki erre például a tranzakciós replikációt - vagy a Change Trackinget.

Mi van akkor, ha T1 visszavonja a módosításait? Ebben az esetben a szűrésben nem lesz semmilyen adat a T1 által módosítottakból, így a rendszer helyesen fog működni.

Ha a read committed snapshot be van kapcsolva, akkor viszont még rosszabb lesz a helyzet. A 16-os verzióig tartó szűrés egyszerűen nem hozza le a T1 által függőben tartott változtatásokat, ki hagyja őket, így nem blokkolja a szinkronizációs folyamatot. Ezután T1 véglegesít, és úgymond megjelennek a módosításai az adatbázisban. A következő szinkron elviszi a 16 utáni adatokat, azaz T1 utolsó módosításait, de az eleje, a 9-10 elveszik! Azaz a szinkron szem elől téveszt sorokat, nem fogja levinni a kliensre. Ez már nagy baj. Mi a probléma gyökere? Az, hogy a @@DBTS a nyitott tranzakciókban futó műveletek részére kiosztott verziószámok maximumát adja vissza. Ami nekünk kellene, az a **lezárt** tranzakciók részére kiosztott verziószámok maximuma.

Változáskövetés kézi hajtánnyal v4.0

Jöhet a `min_active_rowversion()`. Ő egy új függvény, amelyet az SQL Server 2005 SP2-ben vezettek be (a 2008 nyilván alapban tartalmazza). Visszaadja azt a legalacsonyabb verziószámot, amelyet egy *nyitott* tranzakció használ. Azaz, ha ebből az értékből kivonunk egyet, megkapjuk azt az utolsó sorszámot, amelyet még fel lehet használni szinkronizálásra, maga a `min_active_rowversion` már csak feltételes, még lehet, hogy visszagörgetik a hozzá tartozó változást, nem szabad élni vele.

```
set nocount on

insert Gyumolcs (nev)
values (N'alma')

print N'Tranzakció előtt'
select cast(@@DBTS as bigint) [@@DBTS], MIN_ACTIVE_ROWVERSION()-1
[MIN_ACTIVE_ROWVERSION()-1]

begin tran

update Gyumolcs
set nev = 'aaa'

insert Gyumolcs (nev)
values (N'körte')

select nev, cast(verzio as bigint) verzio from Gyumolcs

print N'Tranzakcióban'
select cast(@@DBTS as bigint) [@@DBTS], MIN_ACTIVE_ROWVERSION()-1
[MIN_ACTIVE_ROWVERSION()-1]

rollback

print N'Tranzakció visszavonása után'
select cast(@@DBTS as bigint) [@@DBTS], MIN_ACTIVE_ROWVERSION()-1
[MIN_ACTIVE_ROWVERSION()-1]

select nev, cast(verzio as bigint) verzio from Gyumolcs
```

A futtatás kimenete:

```
Tranzakció előtt
@@DBTS                MIN_ACTIVE_ROWVERSION()-1
-----
4141                  4141

nev                    verzio
-----
aaa                    4142
körte                  4143

Tranzakcióban
@@DBTS                MIN_ACTIVE_ROWVERSION()-1
-----
4143                  4141

Tranzakció visszavonása után
@@DBTS                MIN_ACTIVE_ROWVERSION()-1
-----
4143                  4143
```

nev	verzio
-----	-----
alma	4141

A tranzakció lefutása előtt a 4141-es verziószám volt a legfrissebb, ekkor még nem volt nyitott tranzakció. A tranzakcióban egy update és egy insert történik, a kiosztott verziószámok 4142 és 4143 lesz. A @@DBTS vissza is adja a 4143-at, mint utolsó sorszámot. Azonban a `min_active_rowversion()` -1 4141-et ad vissza, azaz nagyon helyesen a tranzakció elindulása előtti verziószámot. Akkor is ezt adná vissza, ha a tranzakción kívülről futtatnánk, mint az igazi szinkronizálás során, csak az egyszerűbb tesztelés kedvéért raktam most bele.

A tranzakció visszagörgetése után a két verziószám újra egyenlő lesz. A tranzakció elhasznált ugyan 2 sorszámot, amelyek nem látszanak a táblában, hisz a módosítást visszavontuk, de nem véletlenül 8 bájtos a rowversion, ezt elég sokáig lehet pörgetni. Ha a tranzakció véglegesítésre (commit) került volna, akkor is ugyanezek a verziók jönnének vissza a függvényekből, csak akkor a táblában is látszanának.

Az új függvényt tehát úgy kell használni, hogy ahol eddig @@DBTS volt a kódban, ezentúl `min_active_rowversion()` -1-et kell írni.

Összegezve elmondhatjuk, hogy `amin_active_rowversion()` megoldja azt, hogy egy tranzakció módosításai csak egyszerre jöjjenek le, azonban nyitva hagy még egy problémát: ha hosszú ideig nyitva marad egy tranzakció, akkor sok változás, amit más, már rég lezárt tranzakciók okoztak nem kerül leszinkronizálásra a kliensek felé. A probléma forrása az, hogy a verziók a módosító műveletek futtatásakor kerülnek kiadásra, nem a tranzakciók rögzítésekor. Ezen viszont már csak egy dolog tud segíteni: a Change Tracking.

Gondolatok szinkronizálás alkalmazásokról

Mielőtt rátérnénk a Change Trackingre, előtte már pár záró gondolat általában a szinkronizáláshoz. Sok esetben még a szerveren, az adatforrásnál külön szeretnénk választani a módosításokat és a beszúrásokat azért, hogy ne kelljen a subselectes-joinos szűrést végrehajtani kliens oldalon, a kétféle művelet szétválogatásához. Azért nem akarjuk ezt, mert sok sor esetén ez erőforrásigényes lehet, főleg, hogy kliens adatbázisként esetenként SQL Server Compact Edition-t vagy más kis adatbáziskezelőt használunk.

SQL Server, mint adatforrás esetén a szétválasztás az alábbi trükkel lehetséges:

```
create table Gyumolcs
(
    id uniqueidentifier not null primary key
    default (newsequentialid()),
    nev nvarchar(100) not null,
    updverzio rowversion not null,
    insverzio binary(8) default @@DBTS + 1
)
```

Az `updverzio` `sima rowversion` alapú, ezt már ismerjük. Az `insverzio` pedig úgy van összeállítva, hogy insertkor az értékét egy `default` állítja be, amely `@@DBTS + 1` értéket rendeli hozzá, amely pont az `updverzio` oszlophoz generált verziószám. Az `insverzio` értéke már nem változik meg a módosítások során, így szelektíven tudunk szűrni a kétféle műveletre. Egyedül arra kell ügyelni, hogy az `update` műveletek leszűrésénél nem szabad lehozni azokat a sorokat, amelyekben az `insverzio` azonos az `updverzio`val, hisz ezek a beszűrt sorok. Másképpen is meg lehet fogalmazni a feltételt, mondhatjuk azt, hogy az `updverzio` normál szűrése mellett még teljesüljön az a feltétel, hogy az `insverzio` legyen kisebb, mint az előző szinkronizáláskor felhasznált `anchor`, magyarul a sor nem az előző és a mostani szinkron között keletkezett, hanem korábban.

```
updverzio > @sync_last_received_anchor AND updverzio <= @sync_new_received_anchor AND  
insverzio <= @sync_last_received_anchor
```

Aki szinkronizálós megoldást akar kiépíteni, annak javaslom, hogy esze ágában se legyen saját maga nekifogni, ez is egy olyan feladat, amely nagyon jól általánosítható, így jól előkészített sablonmegoldás adható rá. A Microsoft Synchronization Services tálcán kínál erre egy keretrendszert, amely segítségével minimális kóddal tudunk összerakni kétirányú, ütközéskezeléssel felszerelt szinkronizációs megoldást, tetszőleges adatbáziskezelők között. Ez keretrendszer, nem kész megoldás, azaz nem klikk, klikk, next varázslás, mint a replikáció (habár..., de nem akarom előre lelőni a poént). Ennek alá kell dolgozni, pont azokkal a megoldásokkal, amelyekről eddig szó volt, cserébe viszont könnyen testre szabható bármilyen adatforrása, akár nem relációsa is, pl. ha egy WCF végpontról jönnek az adatok.

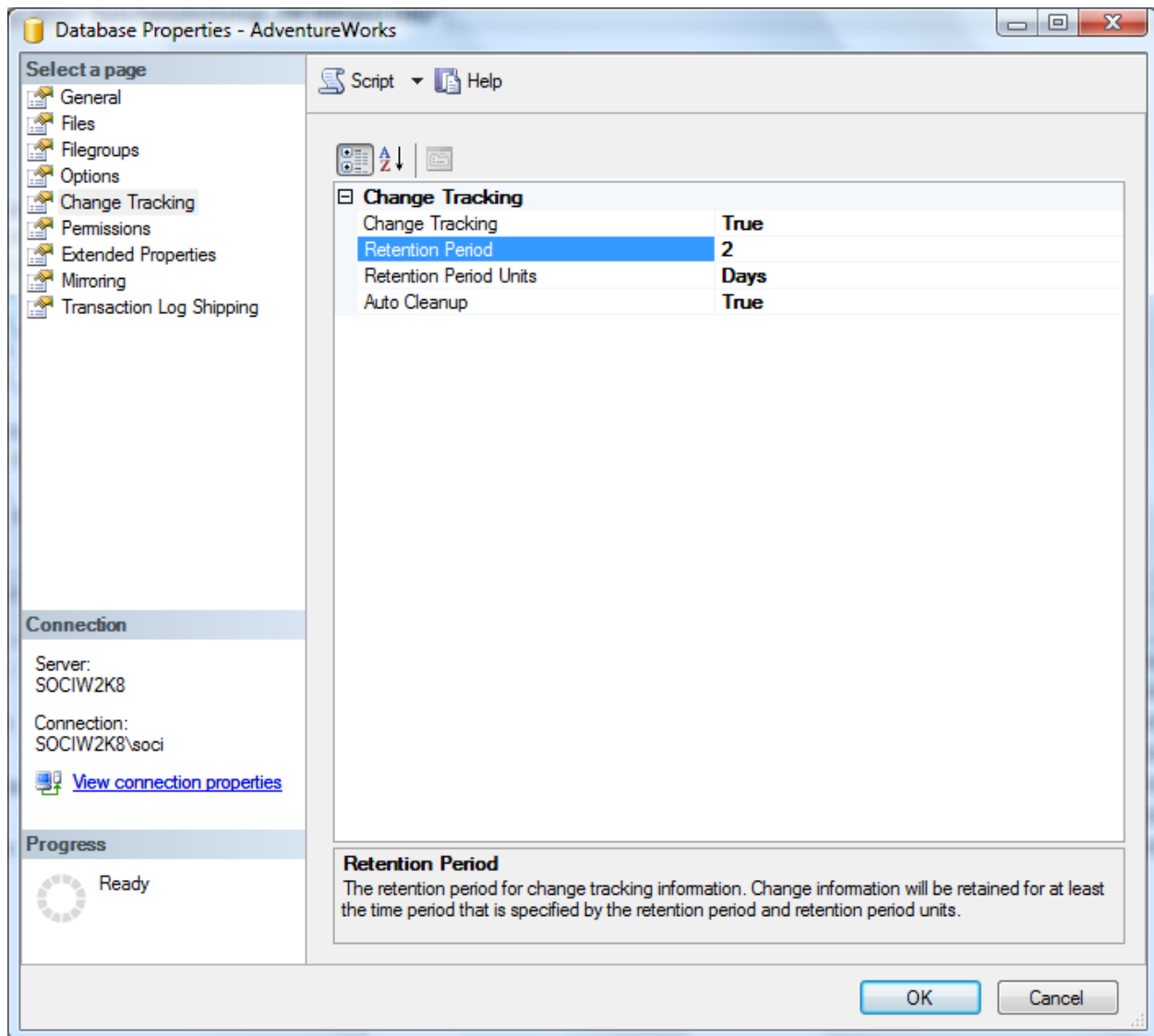
A korábbi megoldásaink nyitva hagytak pár kérdést. Meg kellene oldalunk a törlések kezelését, ehhez általában egy tombstone (sírkő) nevű táblát szoktak létrehozni minden egyes követendő táblához, ebbe kerülnek a törölt sorok. A törlés során a sor mozgatását általában táblánkénti triggerekkel oldják meg. A sírkő táblákat időnként ki kell pucolni, ha már elvitte minden szinkronizáló kliens az törlés lenyomatát. A Change Tracking természetesen ezekre is megoldást nyújt.

Change Tracking megoldás v1.0

A CT használatát engedélyezni kell adatbázis szinten, majd be kell kapcsolni a nyomkövetendő táblákra. A táblákon kívülről semmilyen változást nem fogunk tapasztalni. Nem keletkeznek rajta triggerek. Nem kell hozzáadni verzió oszlopot. Nem kell létrehozni és karbantartani sírkő táblákat. Nem kell törölni a régi információkat a sírkövekből és a verziótáblákból. És ami a legfontosabb: a változáskövetés belső verziószáma a véglegesített tranzakciókhoz igazodik, nem az egyedi műveletek lefutásához, így az összes korábban tárgyalt hibákat kiküszöböli. Itt az ideje bekapcsolni (2. ábra).

Scriptből:

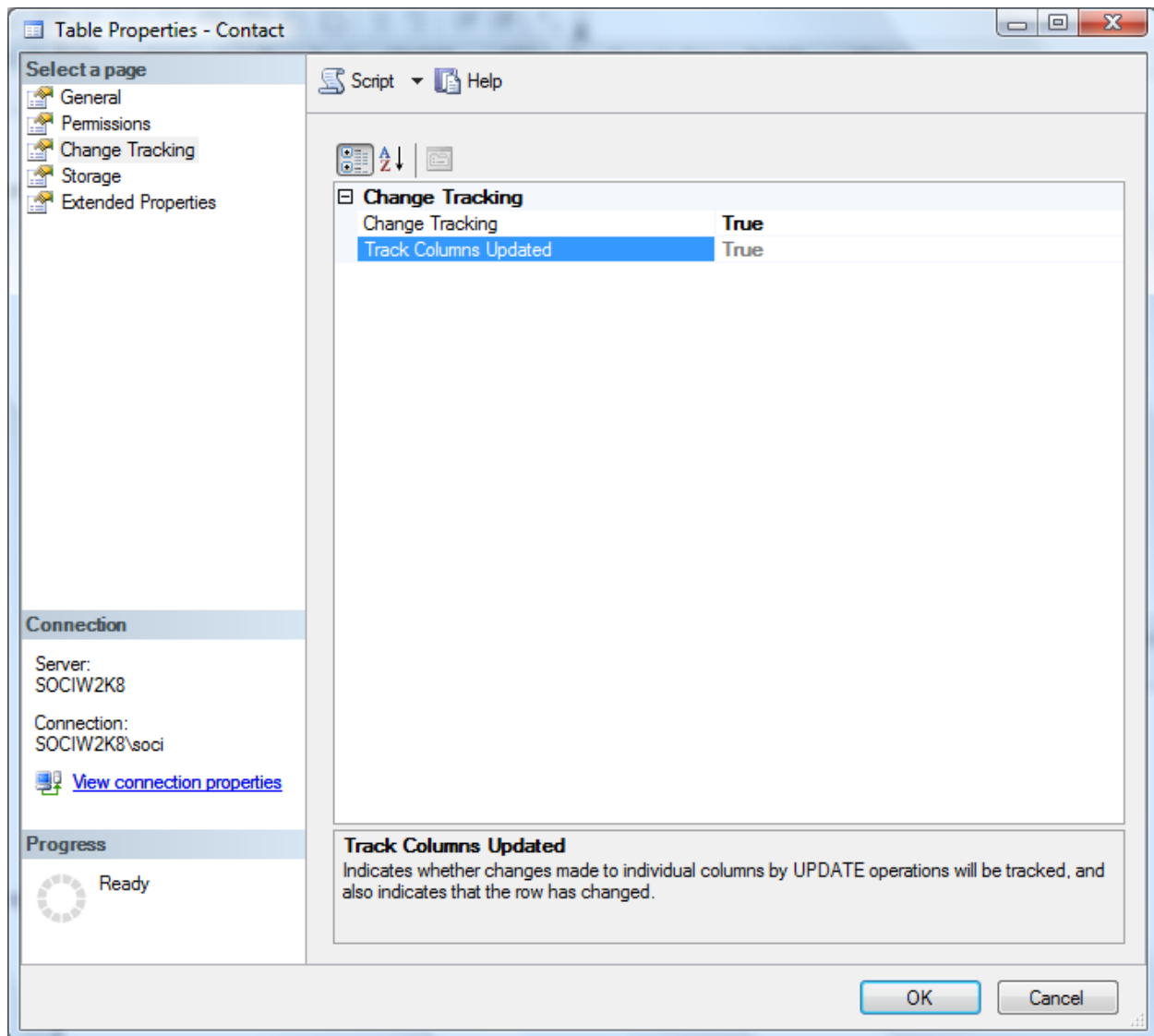
```
ALTER DATABASE AdventureWorks  
SET CHANGE_TRACKING = ON  
(CHANGE_RETENTION = 2 DAYS, AUTO_CLEANUP = ON)
```



2. ábra: Change Tracking bekapcsolása egy adatbázisra

A `CHANGE_RETENTION = 2 DAYS`, `AUTO_CLEANUP = ON` változáskövető háttértáblák takarításáról szól, 2 napnál régebbi adatokat már nem kérünk, azaz feltételezzük, hogy a kliensek ennél sűrűbben szinkronizálnak. Ha beütne a ménkö, és valamiért mégis meg kellene még tartani a metainformációkat, akkor az `AUTO_CLEANUP = OFF`-al átmenetileg ki tudjuk kapcsolni a takarítást. Az alkalmazások persze le tudják kérdezni, lemaradtak-e valamiről, ha igen, akkor lekérlik újra az összes számukra szükséges adatot, újra inicializálják magukat.

A táblákra külön-külön be kell kapcsolni a változáskövetést (3. ábra).



3. ábra: Change Tracking engedélyezése egy táblára

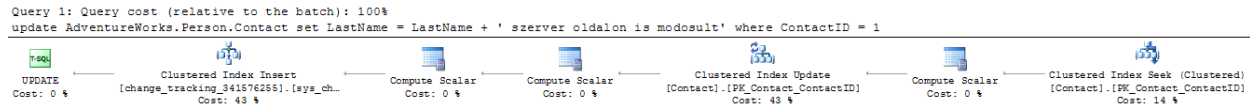
TSQL-ben:

```
ALTER TABLE Person.Contact
ENABLE CHANGE_TRACKING
WITH (TRACK_COLUMNS_UPDATED = ON)
```

Az oszloponkénti változáskövetés előnye, hogy kisebb lesz a szinkronizációs hálózati forgalom, illetve kevesebb konfliktus lesz a változások összefésülésekor, viszont több helyet igényel a változások tárolása a szerveren. Alapban ezért ennek kikapcsolt az értéke.

Egyetlen követelmény a táblával szemben, hogy legyen elsődleges kulcsa, mert a CT-nek össze kell tudni párosítani a módosításokat az alaptáblával. A kulcs lehet egész is, nem kell, hogy guid legyen.

A háttérben a CT is táblákat használ a módosításokról szóló adatok tárolására, ez jól látszik a végrehajtási tervből:



4. ábra: adatmódosítás egy olyan táblán, amin engedélyezve van a CT. A 2. operátorban, a Clustered Index Insertben rögzítik a módosítás lenyomatát.

A következő példában az AdventureWorks adatbázis egy tábláját használom, amitől realisztikusabb a példa, de kissé hosszabbak is a scriptek. A kliensoldalra nem viszünk át minden oszlopot a forrástáblából, hogy egyszerűbbek legyenek a kódok:

```
create type [dbo].[NameStyle] from [bit] NOT NULL
go
create type [dbo].[Name] from [nvarchar](50) NULL
go
create table dbo.[Contact] (
    [ContactID] [int] NOT NULL primary key clustered,
    [NameStyle] [dbo].[NameStyle] NOT NULL,
    [Title] [nvarchar](8) NULL,
    [FirstName] [dbo].[Name] NOT NULL,
    [LastName] [dbo].[Name] NOT NULL,
)
go
drop table SzinkronMetaadat
go
create table SzinkronMetaadat
(
    utolsoVerzio bigint null
)
go
insert SzinkronMetaadat (utolsoVerzio) values(null)
```

Amikor a kliens az első alkalommal akar szinkronizálni, akkor a korábban látott gondolatmenet alapján lekéri az anchor verziót, eltárolja a következő szinkronhoz, majd lehozza az összes sort:

```
--Első szinkron
use AdventureWorks

begin tran
begin try

declare @anchor bigint = CHANGE_TRACKING_CURRENT_VERSION();
select @anchor [@anchor] --csak, hogy lássuk az értékét

update SzinkronMetaadat
set utolsoVerzio = @anchor

insert into [B].[dbo].[Contact] (
    [ContactID]
, [NameStyle]
, [Title]
, [FirstName]
, [LastName])
select
```

```

[ContactID]
, [NameStyle]
, [Title]
, [FirstName]
, [LastName]
from [AdventureWorks].[Person].[Contact]

commit tran

end try
begin catch
    print 'Baj van'
    rollback
end catch
--Első szinkron vége

@anchor
-----
8

```

A `CHANGE_TRACKING_CURRENT_VERSION` függvény hasonló, mint a korábbi példákban használt `min_active_rowversion`, csak nem utasítás, hanem tranzakció szinten működik, az utolsó, véglegesített tranzakció verziószámát adja vissza. Mivel a verziószámokat a tranzakció végén, és nem az elején osztják ki, ezért nem fognak várakozni a később kezdődött, de hamar lefutott tranzakciók a korább, nyitottak miatt.

Mielőtt továbbmennénk, fel kell tűnjön valami. Lekérjük az utolsó verziószámot, majd elkezdjük áttölni a sorokat. De közben már új tételek keletkezhetnek a forrástáblában, amelyek a következő szinkronizáláskor újra le fognak tölteni, konfliktust okozva a céltáblában. Mit lehet tenni? Snapshot izolációs szinten kell futtatni a teljes folyamatot. A snapshot biztosítja azt, hogy minden táblából a tranzakció kezdetén érvényes állapotot fogjuk látni, akkor is, ha sok tábla tartalmát kell áttölni. Igaz ugyan, hogy a `CHANGE_TRACKING_CURRENT_VERSION` tartalma nem táblából jön, de a CT úgy van megvalósítva, hogy a támogató függvényei is az adott izolációs szintnek megfelelően működjenek.

Nagyon fontos megérteni, hogy nem a read committed snapshot adatbázis beállításról beszélek, hanem a snapshot izolációs szintről. Aki nem ismeri a két fogalmat, az most álljon meg, és olvassa el ezt a cikket:

<http://www.microsoft.com/hun/technet/article/?id=59d276d3-a22f-48ae-b977-325cd175f342>

Engedélyezzük a snapshot szint használatát az adatbázisokon. Normál esetben ezt csak a forrásra kellene megtenni, de mivel most az egyszerűség kedvéért egy kapcsolaton keresztül és egy tranzakcióban kezeljük a két oldalt, mindkét adatbázisnak támogatni kell a snapshotot:

```

ALTER DATABASE AdventureWorks
SET ALLOW_SNAPSHOT_ISOLATION ON

ALTER DATABASE B
SET ALLOW_SNAPSHOT_ISOLATION ON

```

Írjuk át a korábbi szinkronizáló kódot, hogy snapshot szinten fusson:

```

set transaction isolation level snapshot
begin tran
... --mint az előbb

```

A további módosítások lekéréséhez a Change Tracking `changetable (changes . . .)` függvényét használjuk. Egyelőre csak önállóan nézzük meg:

```
--Némi módosítás
update AdventureWorks.Person.Contact
set FirstName = FirstName + 'A'
where ContactID = 1

declare @utolsoAnchor bigint = (select utolsoVerzio from B.dbo.SzinkronMetaadat);
select @utolsoAnchor [@utolsoAnchor]

declare @anchor bigint = CHANGE_TRACKING_CURRENT_VERSION();
select @anchor [@anchor]    --csak, hogy lássuk

select
ct.*, C.ContactID, C.FirstName
from AdventureWorks.Person.Contact as C
right outer join
changetable(changes AdventureWorks.Person.Contact, @utolsoAnchor) as CT
on C.ContactID = CT.ContactID
```

A könnyebb olvashatóság kedvéért kettétörtem a változást tartalmazó hosszú sort:

```
@utolsoAnchor
-----
8

(1 row(s) affected)

@anchor
-----
9
```

SYS_CHANGE_VERSION	SYS_CHANGE_CREATION_VERSION	SYS_CHANGE_OPERATION
9	NULL	U

SYS_CHANGE_COLUMNS	SYS_CHANGE_CONTEXT	ContactID	ContactID	FirstName
0x0000000004000000	NULL	1	1	GustavoaA

Látható, hogy a korábbi szinkron által beállított utolsó verzió 8 volt, most viszont már 9-nél jár az adatbázis.

A `changetable (changes . . .)` kimenetében a `SYS_CHANGE_VERSION` az utolsó módosításhoz tartozó verziószámot adja vissza. Ez olyan, mint a korábbi példában volt az `updverzio`. A `SYS_CHANGE_CREATION_VERSION` a sor létrehozásának verziószámát tartalmazza, hasonlóan az `insverzio`hoz. Mivel esetünkben már létezett a sor a CT bekapcsolásakor, null az értéke. A `SYS_CHANGE_OPERATION` a soron történt művelet kódja, I, U vagy D, az insert, update, delete-nek megfelelően.

A `SYS_CHANGE_COLUMNS` azt mutatja meg, mely oszlopokat érintett egy művelet. Nyilván csak akkor lesz benne nem null érték, ha az oszlopszintű változásokövetés be van kapcsolva, illetve insertnél és delete-nél ennek nincs értelme, hisz azok mindig egy teljes sort érintenek. A dekódolásához egy másik függvény kell:

```

use AdventureWorks
go
select CHANGE_TRACKING_IS_COLUMN_IN_MASK
(columnproperty(object_id('Person.Contact'), 'FirstName', 'ColumnId')
,0x0000000004000000);

```

1 vagy 0 (TSQL true/false) a kimenete, attól függően, hogy az adott oszlop módosult-e. Vigyázni kell a fenti kifejezés használatával, mert az `object_id` nem ad vissza hibát, ha rossz nevet adunk meg, vagy rossz adatbázisban vagyunk, egyszerűen null jön vissza belőle, ettől meg a teljes kifejezés értéke 0 lesz, félrevezethet minket.

Végül a `SYS_CHANGE_CONTEXT` egy érdekes oszlop. Az adatmódosító művelethez megadhatunk egy tetszőleges értéket, amelyet a CT letárol, és visszaad ebben az oszlopban:

```

declare @modosito varbinary(128) = cast('AkarmilyenApp' as varbinary(128))
;WITH CHANGE_TRACKING_CONTEXT (@modosito)
update AdventureWorks.Person.Contact
set FirstName = FirstName
where ContactID = 2

```

Ezzel lehet azonosítani a módosítókat, például ha valamilyen prioritásos rendszer szeretnénk kitalálni a konfliktuskezeléshez, vagy csak egyszerűen látni akarjuk, ki a módosító.

A változások listáját tehát megkapjuk a `changetable (changes . . .)`-lel, helyezzük ezeket be a céltáblába. Ezt tipikusan nem SQL kóddal, hanem pl. C#-ban szokták megoldani, mivel sok esetben nem is SQL Server a céladatbázis. Most viszont szemléletes lehet a TSQL megoldás, amely a merge utasításra épül:

```

--Inkrementális szinkron
use AdventureWorks;

set transaction isolation level snapshot
begin tran
begin try

declare @utolsoAnchor bigint = (select utolsoVerzio from B.dbo.SzinkronMetaadat);
select @utolsoAnchor [@utolsoAnchor]

declare @anchor bigint = CHANGE_TRACKING_CURRENT_VERSION();
select @anchor [@anchor]    --csak, hogy lássuk

update B.dbo.SzinkronMetaadat
set utolsoVerzio = @anchor

merge B.dbo.Contact as T
using
(select
ct.SYS_CHANGE_OPERATION, CT.ContactID,
C.NameStyle, C.Title, C.FirstName, C.LastName
from AdventureWorks.Person.Contact as C
right outer join
changetable(changes AdventureWorks.Person.Contact, @utolsoAnchor) as CT
on C.ContactID = CT.ContactID) S
on T.ContactID = S.ContactID
when matched and S.SYS_CHANGE_OPERATION = 'U' then

```

```

        update set
        T.NameStyle = S.NameStyle,
        T.Title = S.Title,
        T.FirstName = S.FirstName,
        T.LastName = S.LastName
    when matched and S.SYS_CHANGE_OPERATION = 'D' then
        delete
    when not matched and S.SYS_CHANGE_OPERATION = 'I' then
        insert (ContactID, NameStyle, Title, FirstName, LastName)
        values (S.ContactID, S.NameStyle, S.Title, S.FirstName, S.LastName)
    output $action, deleted.*, inserted.*;

commit tran

end try
begin catch
    print 'Baj van'
    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; rollback
end catch
--Inkrementális szinkron vége

```

Egy apró részletre oda kell figyelni, hogy a példa jól működjön. A forrást képező selectben CT.ContactID van és nem C.ContactID, mivel a törölt sorok esetén az alaptábla oldaláról nulljön vissza, csak a Change Tracking saját táblája tartalmazza még a kulcsértéket.

A CT beállítások között oszlopszintű változáskezelést kértünk, de mégis minden oszlopot módosítottunk, ez nem helyes. A korábban látott CHANGE_TRACKING_IS_COLUMN_IN_MASK függvénnyel meg tudjuk határozni melyik oszlop módosult, ezzel és case-es kifejezésekkel át tudnánk írni az update-et, hogy tényleg csak a módosult oszlopokat frissítse, ha nincs módosítás az oszlopban, akkor a céltábla azonos oszlopát megadva alternatív kifejezésként:

```

update set
--Szelektív oszlopszintű update, a többi oszlopra is hasonló lenne
T.NameStyle =
case when CHANGE_TRACKING_IS_COLUMN_IN_MASK
(columnproperty(object_id('S.Person.Contact'), 'NameStyle', 'ColumnId'))
,SYS_CHANGE_COLUMNS) = 1 then S.NameStyle else T.NameStyle end,

```

Ez sem tökéletes azonban, mivel a forrásban nem változott oszlopokat a célhelyen saját magára módosítjuk. Ez ártatlannak ható művelet, de ha trigger van a céltáblán, annak logikáját megzavarhatja egy ilyen dummy update. A korrekt megoldás dinamikus SQL használata, amelyben úgy rakjuk össze az update parancsot, hogy csak a megváltozott oszlopok legyenek benne. Erre viszont - szerintem - már nem való a TSQL, ezt valamilyen hagyományos, procedurális nyelven és környezetben kell megírni, pl. C#-ban.

Az output os rész csak azért van a példában, hogy lássuk, milyen adatmozgások történtek. A zöld adatok a céltábla módosítás előtt adatai, a sárgák a módosítás utániak.

\$action	ContactID	NameStyle	Title	FirstName	LastName	ContactID	NameStyle	Title	FirstName	LastName
UPDATE	1	0	Mr.	Gustavo	Aachong	1	0	Mr.	Gustavo	Aachong
DELETE	1570	0	NULL	Katie	Jordan	NULL	NULL	NULL	NULL	NULL
INSERT	NULL	NULL	NULL	NULL	NULL	19978	0	NULL	Gizella	Csintalan

Change Tracking megoldás v2.0

További finomításra szorul az első megoldásunk. Észlelnünk kell, ha esetleg már túl régen szinkronizáltunk, és a szerveroldali a takarító eljárás már kitörölt a retentionként megadott időnél régebbi változásokat. Ebben segít a `CHANGE_TRACKING_MIN_VALID_VERSION` függvény.

Ez a legrégebbi, a verziótárban levő változat verziószámát adja vissza. Ha a kliens által letárolt anchor verzió (`SzinkronMetaadat` tábla `utolsoVerzio` oszlopa) ennél régebbi, akkor már törlődtek olyan változások, amelyeket még nem vittünk el - újra kell kezdeni a szinkronizálást az elejétől.

Az eddigiek alapján a következőképpen összegezhethetjük a szinkronizálás lépéseit:

1. Beállítjuk a snapshot izolációs szintet, majd elindítunk egy tranzakciót.
2. Leellenőrizzük, hogy nem-e túl régen szinkronizáltunk a `CHANGE_TRACKING_MIN_VALID_VERSION` és a kliensen letárolt anchor verzió összehasonlításával. Ha túl régen, újra inicializáljuk a táblákat.
3. Lekérdezzük az aktuális verziót a `CHANGE_TRACKING_CURRENT_VERSION` függvénnyel, letároljuk kliens oldalon, erre szükség lesz majd a következő szinkronhoz.
4. Letöltjük a változásokat a `CHANGETABLE (CHANGES ...)` hívásokkal minden táblához.
5. Ha minden rendben volt, véglegesítjük a tranzakciót, ha nem visszavonjuk.

Change Tracking megoldás v3.0

Ha a kliensen is módosíthatják az adatokat, amelyeket kétirányú szinkronizálással szeretnék vissza is tölteni a szerverre, akkor előfordulhatnak ütközések, amikor ugyanazt a sort, vagy oszlopszintű változáskövetésnél ugyanazt az oszlopot módosítják minkét oldalon. A kétirányú szinkron általában úgy zajlik, hogy a kliens feltölti a változásait a szerverre, majd lehúzza a szerveroldali változásokat magára (eddig csak az utóbbit implementáltuk).

Feltöltés előtt érdemes kizárni azokat a sorokat, amelyek ütközést okoznának. Ehhez felhasználhatjuk azt a már látott információt, hogy a háttérben minden egyes, a CT bekapcsolása óta módosított sornak van egy verziószáma, amely minden módosításkor növekszik. Ezt a számot láthattuk a `changetable(changes ...) SYS_CHANGE_VERSION` oszlopában. A hatékony szűréshez rendelkezésünkre áll egy másik függvény is, a `changetable(version ...)`. Ez csak a legszükségesebb információkat adja vissza, így tisztán verziószűrésre hatékonyabb, mint a `changetable(changes ...)`.

A gyakorlatban a felfelé szinkronizálás általában úgy szokott lezajlani, hogy a kliens egyesével felküldi a módosított sorokat és az általa ismert anchor verziószámot. Ott nem hajtjuk végre mechanikusan az update/delete műveleteket, hanem előtte megnézzük, hogy nem-e módosították közben a serveren is a sort. Ezt onnan tudjuk, ha lekérjük az feltöltendő sorhoz tartozó serveroldali sor verziószámát, és azt összevetjük a kliens által felküldött anchorral. Ha a serveroldali verzió nagyobb, mint az anchor, akkor az azt jelenti, hogy azóta módosították fenn a sort, mióta leszinkronizáltuk, azaz fenn és lenn is módosították, konfliktus van.

A feltöltéshez a kliens előveszi az anchor verzióját, azaz az általa ismert legfrissebb adat verzióját, valamint előkészíti az nála módosított adatokat a feltöltéshez:

```
declare @utolsoAnchor bigint = (select utolsoVerzio from B.dbo.SzinkronMetaadat);
select @utolsoAnchor [@utolsoAnchor]

declare @LastName Name = N'Uj nev'
declare @ContactID int = 1
```

Ezek után paraméterekben a serverre felküldött változásokat a server csak akkor rögzíti, ha nem volt konfliktus:

```
update AdventureWorks.Person.Contact
set LastName = @LastName
from AdventureWorks.Person.Contact C
outer apply changetable(version Person.Contact, (ContactID), (C.ContactID)) AS CT
where
C.ContactID = @ContactID and
(CT.SYS_CHANGE_VERSION <= @utolsoAnchor or CT.SYS_CHANGE_VERSION is null)

if @@ROWCOUNT = 0 PRINT N'Ütközés volt, nem módosítottuk a sort a serveren'
```

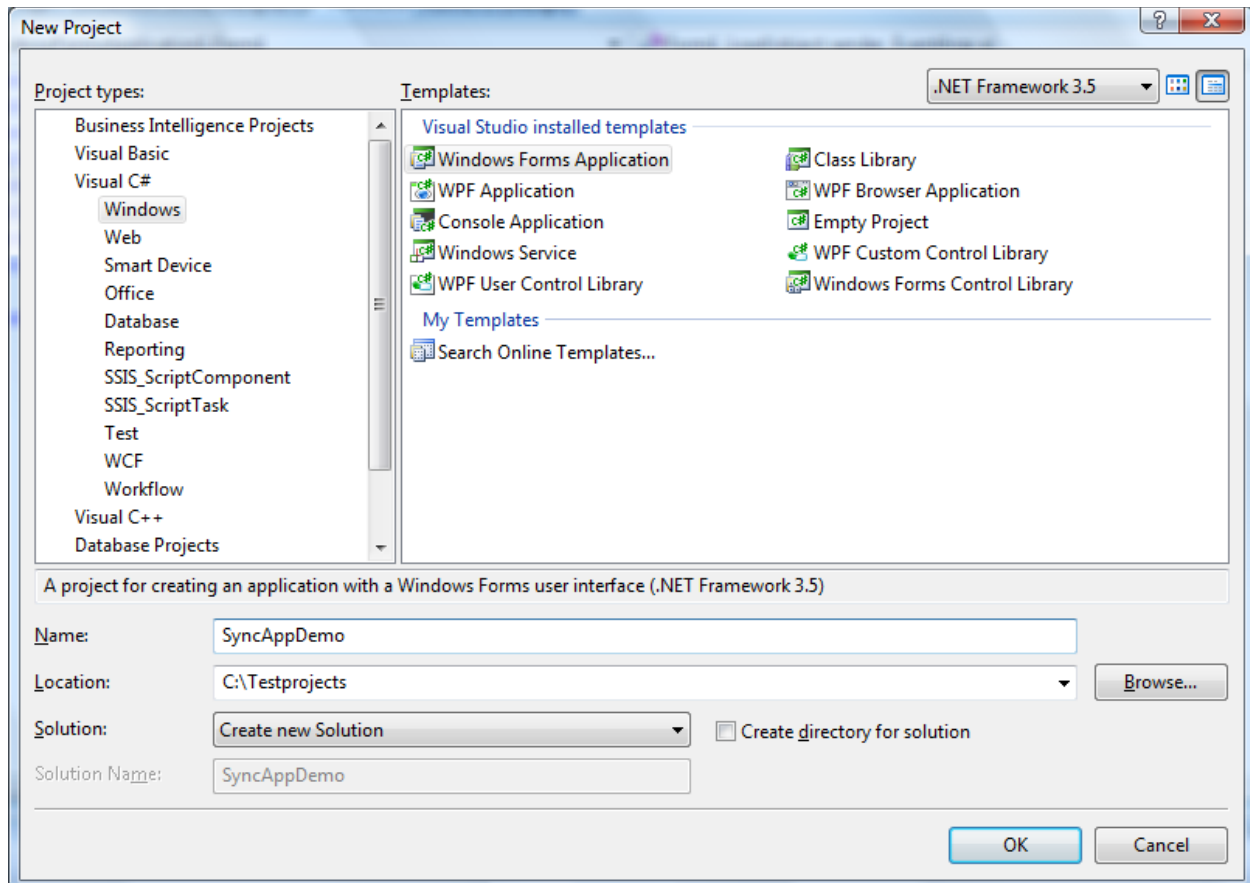
Mi történik itt? Végre akarunk hajtani egy update-et a serveren, paraméterül megkapja a kliensen módosított oszlopok új értékét (most csak egy van, a LastName). Hozzá join-oljuk a changetable(version ...)-t, amelyben megmondjuk, hogy a kérdéses Contact sorhoz (C.ContactID) kérjük a serveroldali legfrissebb verziószámot. Az outer apply olyan, mint az outer join, csak függvényeknél ezt kell használni, nem join-t, de logikailag a kettő ugyanaz. A szűrésben egyrészt rögzítjük a feldolgozandó sort az elsődleges kulcs mentén (C.ContactID = @ContactID), valamint megnézzük, hogy a serveroldali verzió (CT.SYS_CHANGE_VERSION) kisebb-e vagy egyenlő, mint a kliens által ismert (@utolsoAnchor), vagy esetleg nincs is még verziója a serveroldali sornak, mert soha nem módosították (CT.SYS_CHANGE_VERSION is null). Ha ezek a feltételek teljesültek, akkor módosítunk egy sort a serveren, azaz megtörténik egy sor felfelé szinkronizálása. A @@ROWCOUNT segít eldönteni, volt-e módosítás. Ha az értéke nulla, azaz nem volt módosítás, akkor konfliktus volt, amit le kell kezelnünk (el kell döntenünk, ki győz, aztán annak a módosítását véglegesíteni).

Szinkronizálás alkalmazás összeállítása

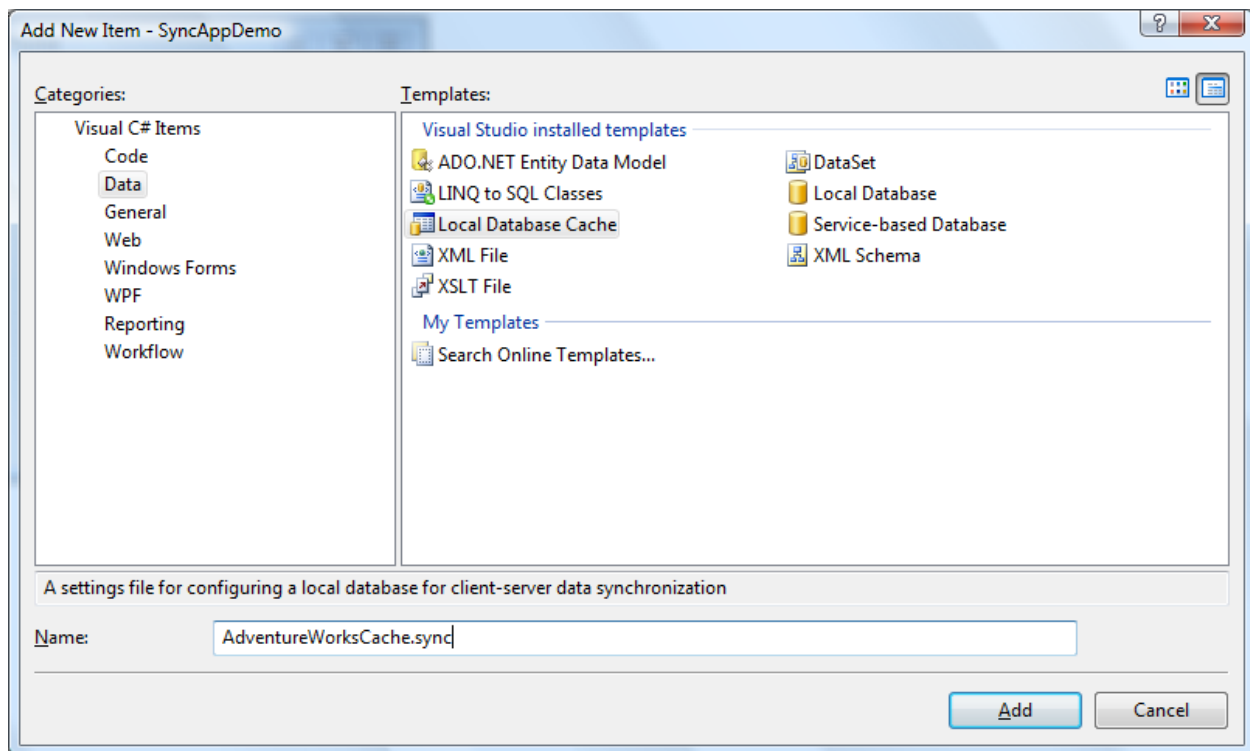
Tegyük fel a koronát az eddigi tudásunkra, indítsuk be a varázslókat, és élvezzük a modern technikát. A következőkhöz Visual Studio 2008 SP1 szükséges, illetve mondanom sem kell SQL Server 2008.

A következőkben minimális munkával összerakunk egy olyan alkalmazást, amely SQL Server 2008-on tárolt adatokat szinkronizálja SQL Server Compact adatbázisba. Az alkalmazás Compact adatbázist használ mint helyi tárat addig, amíg nem tud szinkronizálni a nagy SQL Serverhez. Occasionally connected alkalmazásoknak hívják az ilyeneket. Lássunk neki!

File, New Project:



A Solution Explorerben Add New Item:



Felül beállítjuk a fő adatforrásként szolgáló SQL Server-t. A Client Connectionnél a New... gombra kattintva létrehozuk a Compact adatbázist:

Configure Data Synchronization

Select or create a Server connection to the remote database and then add the tables to cache locally. The Client connection represents the local database where the cached tables are stored after synchronization. Open the code example link to view how to invoke synchronization.

Cached Tables:

Application



Database Connections

Server connection:

sociw2k8.AdventureWorks2008.dbo

New...

Client connection:

AdventureWorks2008.sdf (new)

New...

☒ Use SQL Server change tracking

☒ Advanced

☒ Synchronize tables in a single transaction

Create synchronization components:

Client and Server

Server project location:

SyncAppDemo

Client project location:

SyncAppDemo

+ Add

- Remove

[Show Code Example...](#)

OK

Cancel

Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server Compact 3.5 (.NET Framework) Change...

Data Source

☒ My computer
☐ ActiveSync connected device

Connection Properties

Database:
AdventureWorks Create... Browse...

Password:
.....
☒ Save my password

Advanced...

Test Connection OK Cancel

Create New SQL Server Compact Database

Enter the new SQL Server Compact database filename.
AdventureWorks Browse...

☐ Overwrite existing database file

Destination Database Properties

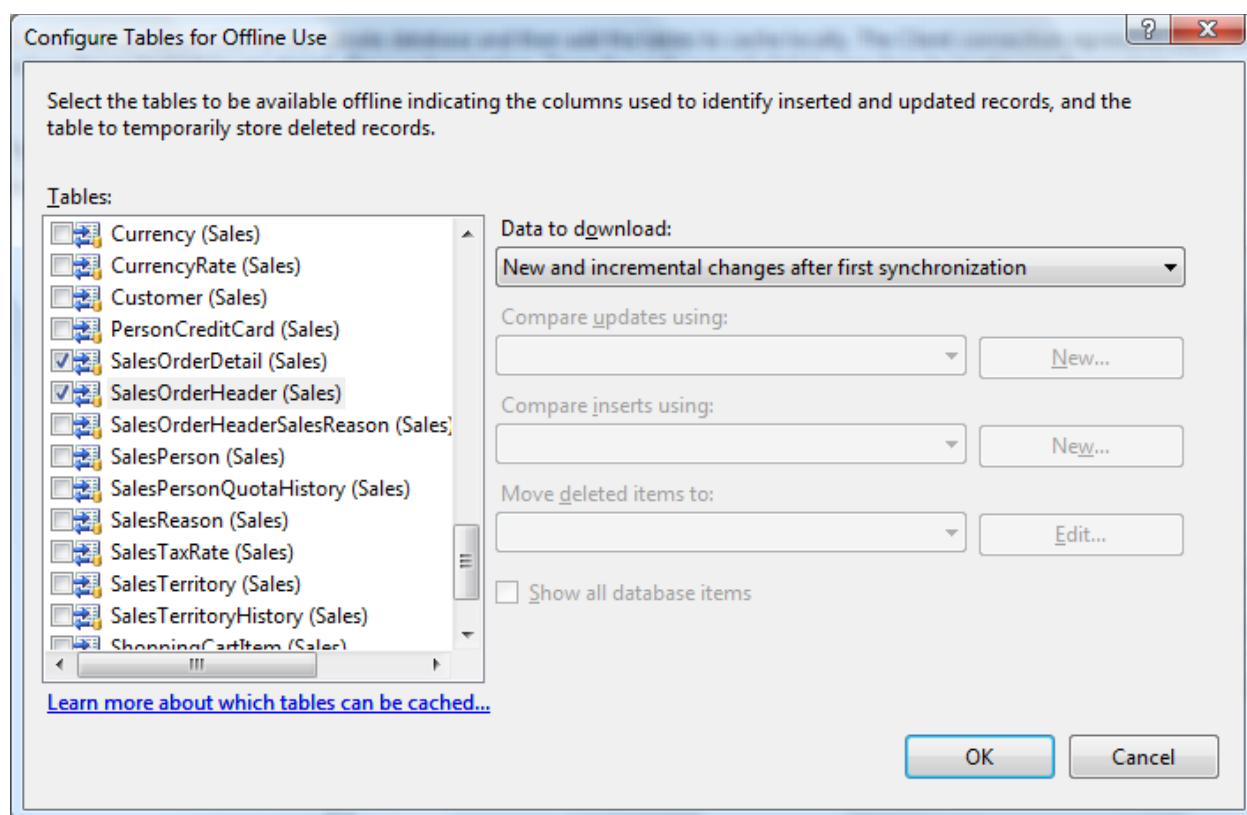
Collation: English - United States ▼
☐ Case sensitive

Encryption Properties

New password:
Confirm password:
Encryption mode: Platform Default ▼

OK Cancel

Bejelöljük, mely táblákat szeretnénk szinkronizálni:



Látható, hogy a változáskövetést a Change Tracking fogja biztosítani szerveroldalon:

Configure Data Synchronization

Select or create a Server connection to the remote database and then add the tables to cache locally. The Client connection represents the local database where the cached tables are stored after synchronization. Open the code example link to view how to invoke synchronization.

Cached Tables:

- Application
 - SalesOrderDetail (Sales)
 - SalesOrderHeader (Sales)

Database Connections

Server connection: sociw2k8.AdventureWorks2008.dbo [New...](#)

Client connection: AdventureWorks.sdf [New...](#)

☒ Use SQL Server change tracking

☒ Advanced

☒ Synchronize tables in a single transaction

Create synchronization components: Client and Server

Server project location: SyncAppDemo

Client project location: SyncAppDemo

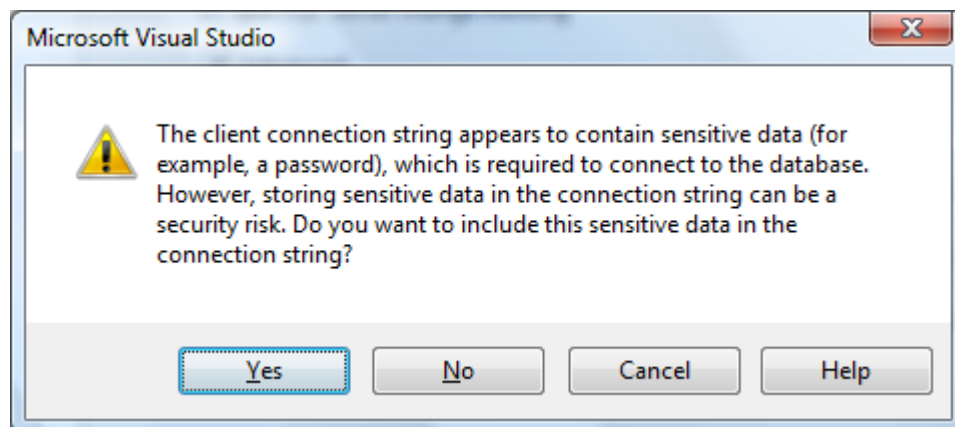
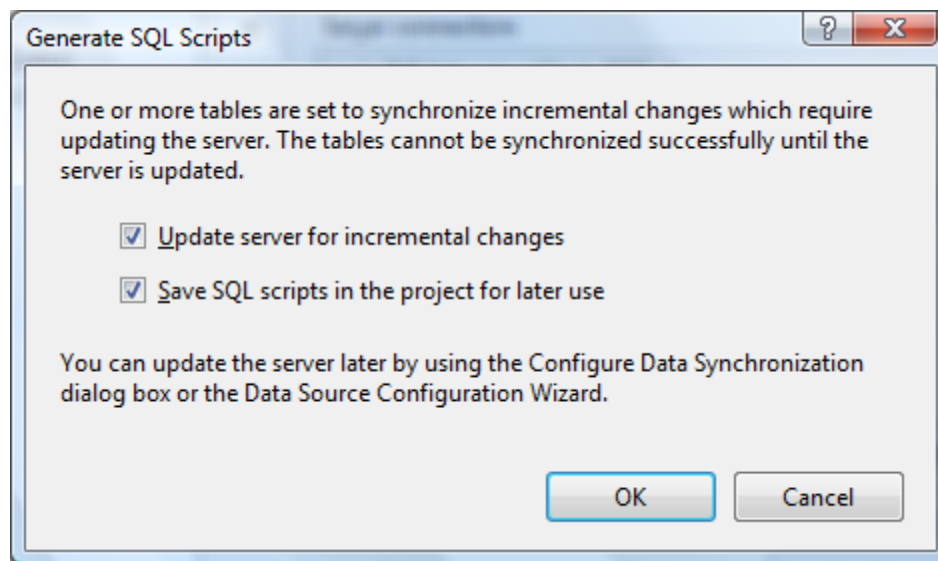
[Show Code Example...](#)

[Add](#) [Remove](#) [OK](#) [Cancel](#)

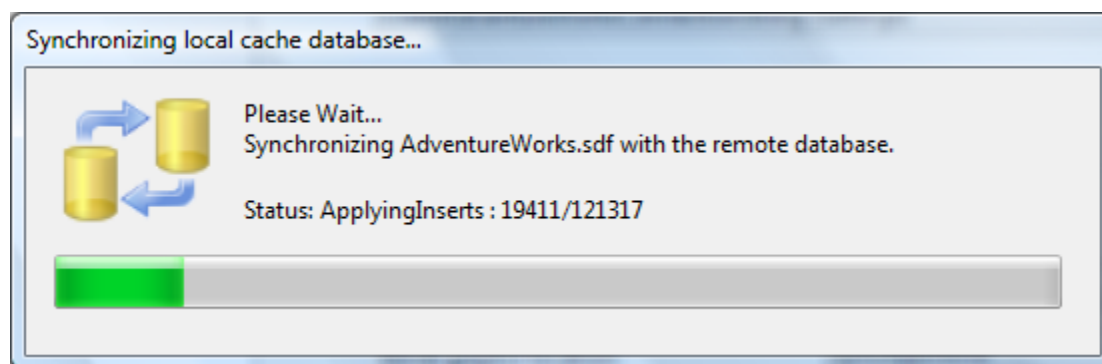
Microsoft Visual Studio

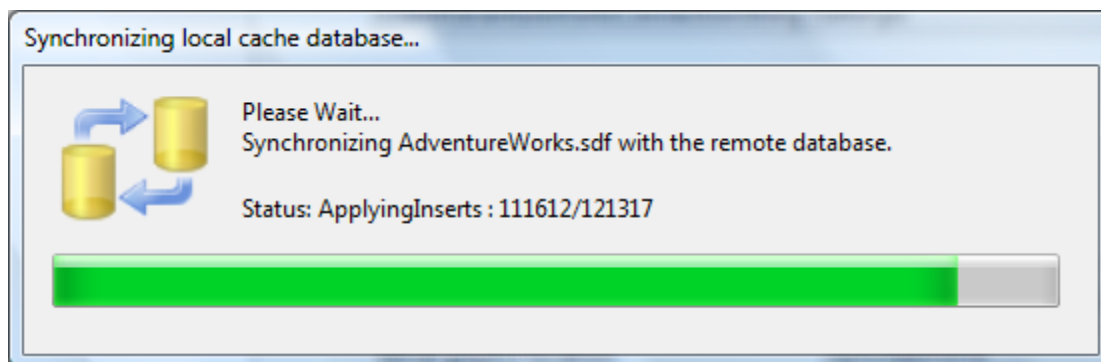
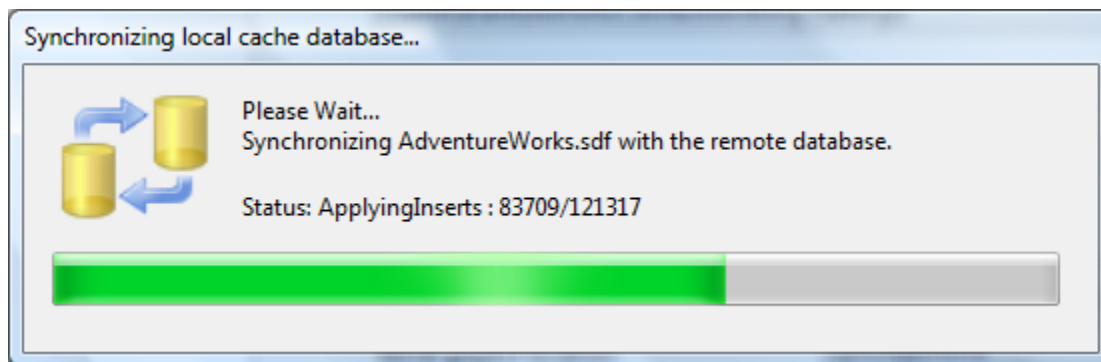
 Your old local data cache might not be workable with new change tracking setting. Click Yes if you want to recreate the local data cache. Click No to continue using the old local data cache. Click Cancel to go back to designer.

[Yes](#) [No](#) [Cancel](#) [Help](#)

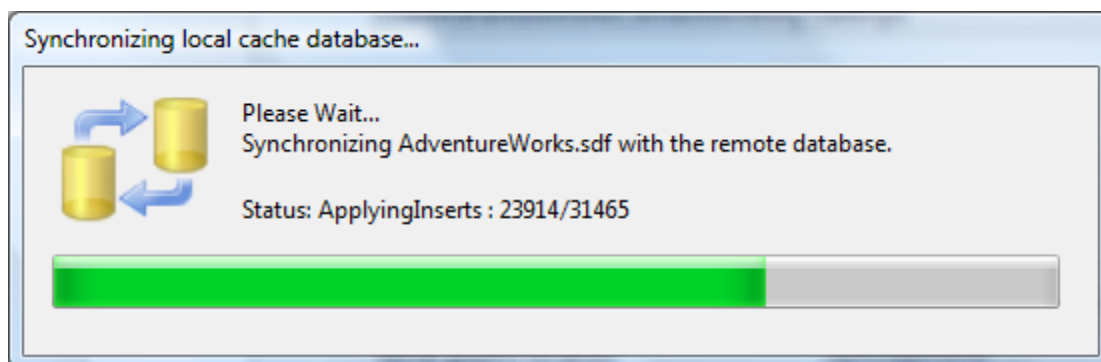


Indul az első szinkronizálás:

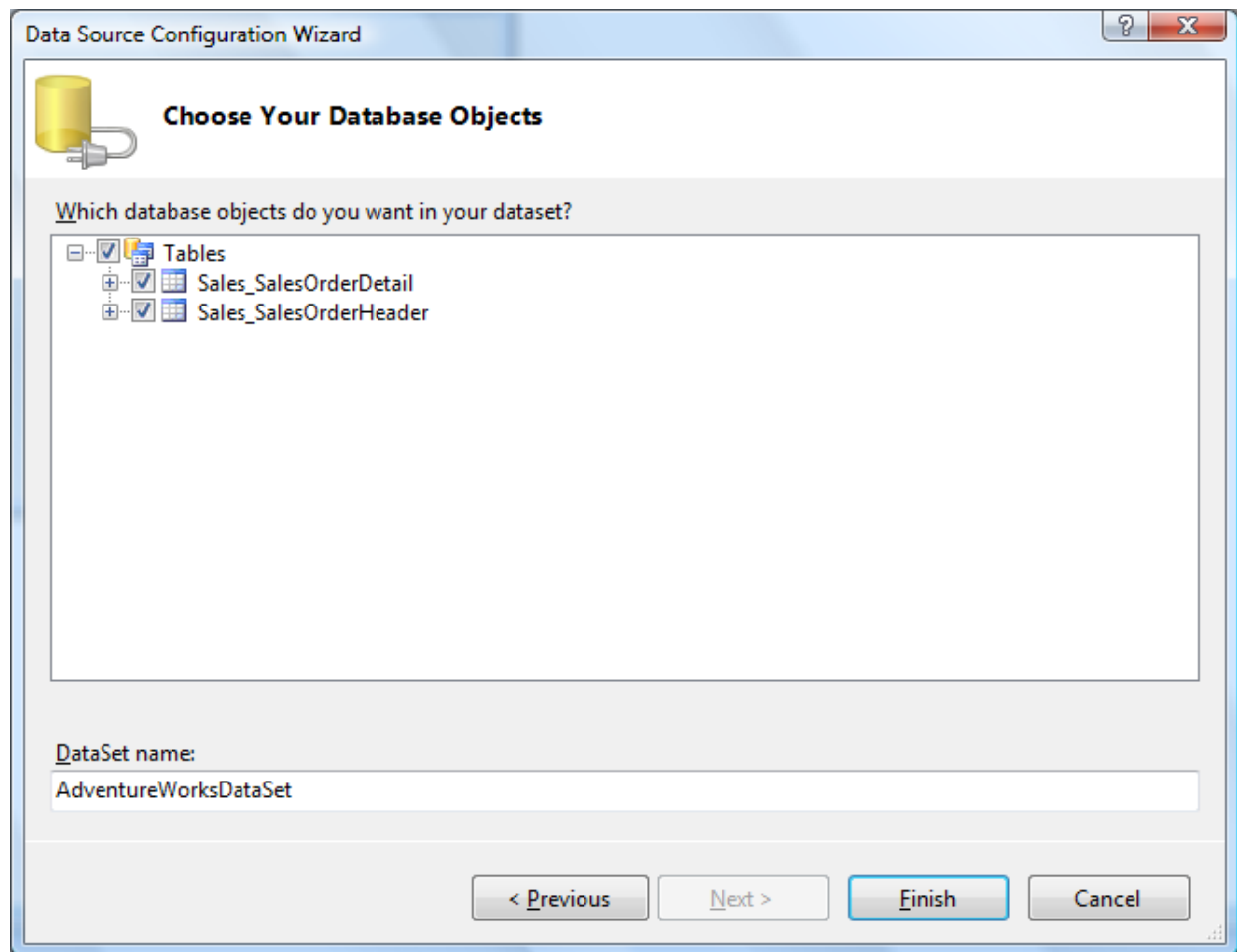




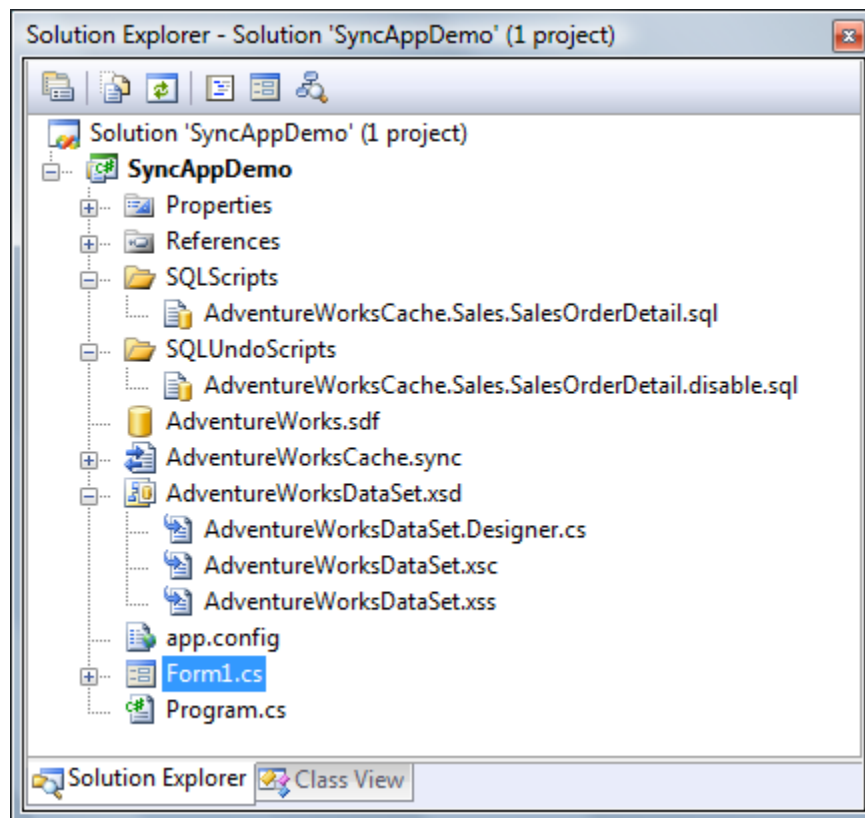
A másik tábla is:



Ha régi DataSetes stílusban akarunk programozni, akkor megjelölhetjük, mely táblákat szeretnénk látni a generált típusos DataSetben. Ha a kliens oldali Compact adatbázist szeretnénk az alkalmazásból használni, akkor az alapján generáltassuk le a DataSetet:



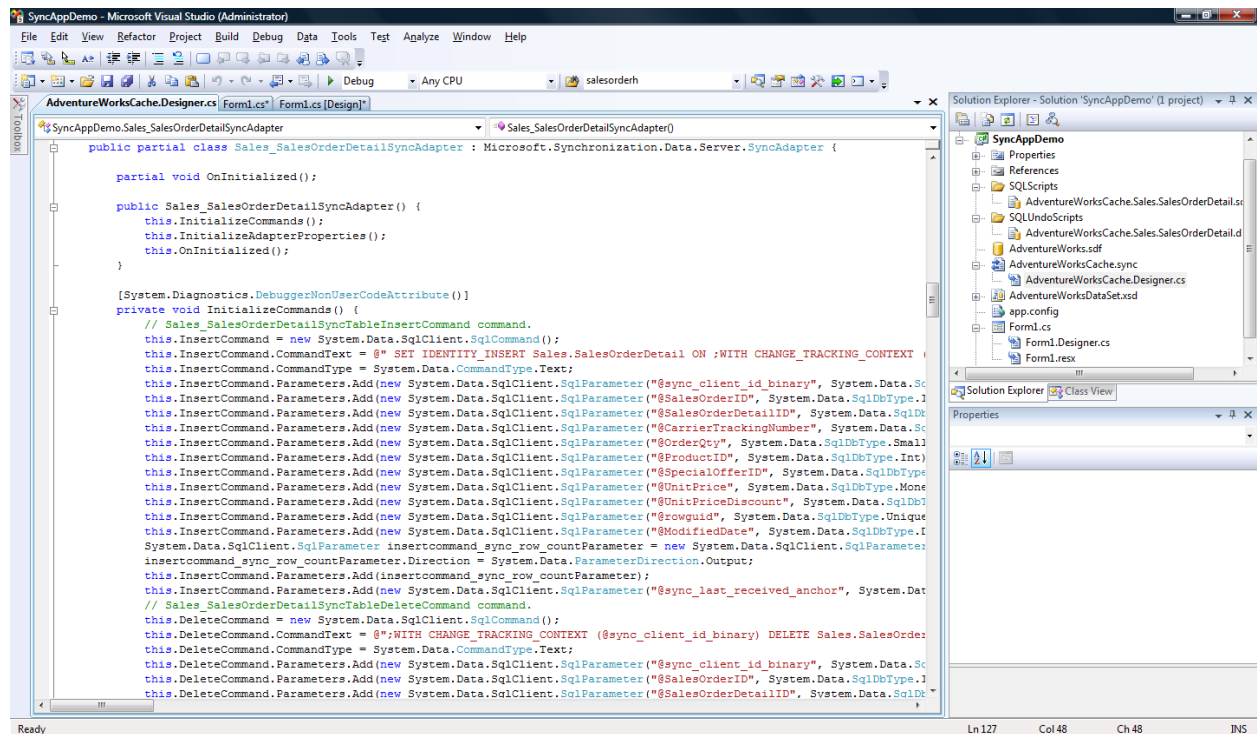
Az elkészült Solution:



AdventureWorksCache.Sales.SalesOrderDetail.sql:

```
IF NOT EXISTS (SELECT * FROM sys.change_tracking_tables
WHERE object_id = OBJECT_ID(N'[Sales].[SalesOrderDetail]'))
    ALTER TABLE [Sales].[SalesOrderDetail]
        ENABLE CHANGE_TRACKING
GO
```

A generált szinkronizáló osztályok. A kód a Synchronization Services osztályaira épül, azokat specializálja:



Most jön a munka neheze, a két oldal közötti szinkronizálás. Ehhez sajnos már gépelni kell, 1 sort:

```
private void button1_Click(object sender, EventArgs e)
{
    new AdventureWorksCacheSyncAgent().Synchronize();
}
```

Ezzel készen is van egy egyirányú szinkronizálásra felkészített alkalmazás. Egyszerűen lehet kétirányosítani is, de ehhez már kicsivel több kódot kell írni, hisz le kell kezelni a konfliktusokat is. Ha nem kívánunk a régi DataSetes filozófiával programozni, akkor elővehetjük az Entity Frameworköt, és ráhúzhatjuk azt a Compact adatbázisunkra, így már objektumorientáltan, entitásokkal érhetjük el az adatokat.

Összefoglalás

A Change Tracking egy új, DML műveletek okozta változások követésére létrehozott kis költségű, szinkron szolgáltatás. Számos dologra használható, jelen cikkben az adatszinkronizálásra történő felhasználást részleteztem ki, amely a Synchronization Services-zel és a Visual Studio 2008 varázslóival együtt rendkívül gyors alkalmazásfejlesztést tesz lehetővé.