SQL 2000 kurzorok használata

Sok haladó SQL szerver programozó számára is ismeretlen a kurzorok fogalma. Pedig helyes használatuk alapvetően meghatározza alkalmazásaink teljesítőképességét, különösen sokfelhasználós környezetben. Mit is tudnak ők? Mire valók? A cikk választ ad ezekre a kérdésekre. Sőt továbbmegyek. A cikk második felében kilépünk a tiszta SQL világból a való világba, és átmegyünk ADO, VB programozóba. Megtanuljuk, hogyan kell százmillió rekordból ezred másodpercek alatt leválogatni az első pár száz sort. Semmi `TOP 100`, semmi varázslás, csak kurzorok. Meglepő lesz, tartsanak velem.

Mi az a kurzor?

Természetesen mindenki tudja a választ. Az a kis bigyó, ami a szövegszerkesztőben a sor végén villog, és kijelöli, hogy a következő karakter hová fog kerülni, ha leütjük a billentyűzet valamely gombját. És ez nem is áll messze az SQL Server kurzorától!
Az SQL Server halmazokban gondolkodik. Amikor végrehajtunk egy lekérdezést, akkor annak van egy kimenete, ami egy n darab sorból álló halmaz, amit *eredményhalmaznak* (result set) hívunk. Amikor a `WHERE` feltétellel szűrjük az adatokat, akkor halmazműveleteket végzünk. Pl. `WHERE Fizetés > 100`, akkor arra kérjük a kiszolgálót, hogy a teljes halmazból (ami például egy teljes tábla tartalma) csak azokat az elemeket tartsa meg, amelyekre teljesül a megadott feltétel. Nem mondjuk azt SQL-ben, hogy:

```
for menjünk végig az összes soron a táblában
  ha a Fizetés > 100, akkor rakjuk bele a kimenetbe
next
```

Azért nem kell leírnunk ilyeneket, mert az SQL nyelv halmazokban gondolkodik, így elég a fenti `WHERE` kifejezés is. Ez nagyon jól van így, mert a lényegre tudunk koncentrálni, nem kell ciklusszervezéssel bajlódnunk. Azonban az élet nem ilyen egyszerű. Rengeteg helyen nem halmazokra, hanem egyedi rekordokra van szükségünk, amelyeken egy ciklusból végigszaladunk, és valamilyen műveletet végzünk a segítségükkel vagy rajtuk. Például írunk egy alkalmazást, ami kilistázza az ügyfelek tábla tartalmát. Megjelenítjük az ügyfelek nevét egy listában, és ha a program kezelője kiválaszt egy nevet, a lista alatt megjelenik az ügyfél összes adata. Ha akarja, módosíthatja ezeket, majd a listában rákattint a következő ügyfélre, és azzal foglalkozik. Azért ez szekvenciális feldolgozás nem? Ez teljesen eltér az SQL Server halmazos lelkivilágától. Szaknyelven ezt a felfogáskülönbséget impedancia különbözőségnek hívjuk (több ilyet nem mondok, ígérem :).
Az SQL Server teljességgel használhatatlan lenne, ha nem oldaná fel a két világ közötti ellentétet, az egyedi rekordokkal foglalkozó való világ és a tiszta SQL-es halmazokból építkező világ között. Erre lesznek jók a kurzorok! Ők a kapu, az átjáró a két világ között.
Képzeljünk el egy felhőt, amiben kis, masnis, sorszámozott csomagocskák lebegnek. Ez az SQL eredmény*halmaz*. Most képzeljük el, hogy jön egy óriás, aki egymásra rakja a dobozkákat, szépen sorban. Azt mondjuk neki: „Te, Küklopsz, add ide nekem a *legfelső* doboz tartalmát!" És ő odaadja nekünk a dobozban található dolgokat, információkat, *egy* rekord tartalmát. Majd, hadd mozogjon, azt mondjuk neki, hogy „Kükkancs, most nekem a *legutolsó* kellene!", és már a miénk is a legalsó dobozka tartalma. Sőt, ez a behemót olyan okos, hogy azt is megérti: „Nagyfiú a legutóbbi *előtt öttel* levő doboz tartalma kellene!", és már adja is az alulról hatodik dobozka tartalmát.
A kurzor pont olyan okos, mint Küki. Ha egy SQL eredményhalmazra nyitunk egy kurzort, akkor a kiszolgáló kivasalja az eredményhalmazt, és készít belőle egy hurkát, amin végig lehet gyalogolni. Lehet neki olyan parancsokat adni, mint „kettővel előre", vagy „a legelsőt" satöbbi. És természetesen az aktuális pozícióban található sor (rekord) adatait ki lehet olvasni, sőt lehet módosítani is.
Másképpen fogalmazva a kurzor egy olyan nevesített eredményhalmaz, amiben a kiszolgáló mindig nyilvántartja az aktuális pozíciót, és amiben léptethetjük azt előre-hátra. Olyan ez, mint amikor a telefonkönyvben keresünk valakit. Az ujjunk mindig rajta áll azon a soron, amit éppen olvasunk. A telefonkönyv az eredményhalmaz, az ujjunk pedig a kurzor, ami mindig egy bizonyos rekordra mutat.
Az SQL Serverrel kétféle kurzort használhatunk, és a kettő közötti különbségtétel nagyon fontos! Az egyik típus az, amit Transact SQL-ben, általában tárolt eljárásokban, triggerekben használunk, és a `DECLARE CURSOR` kulcsszóval hozunk létre. Ezek akkor jók, ha az SQL programunkban sorról-sorra kell valamilyen műveletet végezni, olyat, amit a hagyományos `SELECT`, `UPDATE` stb. utasításokkal nem lehet megfogalmazni. A cikk első fele ezekkel fog foglalkozni. A másik fajta az, amit az adatbázismeghajtó programok (manapság zömében OLE-DB-n, ADO-n keresztül) valósítanak meg. Ezek előnye, hogy eltakarják a „piszkos részleteket", viszont csak valamilyen egyéb nyelven és eszközzel (VB, VC, stb.) lehet használni. A cikkünk második felében az ilyen típusú kurzorokkal foglalkozunk.

Egy egyszerű kurzor létrehozása

Annyit már tudunk a kurzorokról, hogy kell hozzájuk valamilyen lekérdezés, ami egy eredményhalmazt generál, és ehhez lehet valamilyen módon hozzáilleszteni egy kurzort, amivel azután szabadon mozoghatunk a leválogatott rekordok között. Nézzük meg részleteiben, hogyan néz ki ez a folyamat.

1. Deklaráljuk a kurzort a `DECLARE` utasítás segítségével. Ez már ismerős lehet, hiszen a változókat is ezzel lehetett létrehozni. A különbség az, hogy a kurzor nevében nem használhatunk @-ot, ellentétben a változókkal, ahol meg kötelező odarakni. A deklaráció során kétféle viselkedését lehet specifikálni, az egyik azt szabja meg, hogy a kurzor által bejárni kívánt recordset mennyire tükrözze az eredeti adathalmazban bekövetkező változtatásokat, a másik pedig azt, hogy milyen irányokban (előre-hátra, csak előre) lehet bejárni a kapott eredményhalmazt. Itt kell megadni azt a `SELECT` utasítást is, aminek az eredményhalmaza táplálja a kurzort. Mivel a kurzorral bejárt rekordok már egy rendezett halmazt alkotnak, azért sokszor van értelme használni az `ORDER BY`-t is a rekordok rendezésére. Az eddigieknek megfelelő leegyszerűsített példakód így néz ki:

```
DECLARE curTest CURSOR
FOR
SELECT
  EmployeeID, LastName, FirstName
FROM
  Employees
```

2. Megnyitjuk a kurzort. A legegyszerűbb ezt úgy felfogni, mint hogy végrehajtjuk a deklarációban kijelölt lekérdezést.

```
OPEN curTest
```

3. Mozgatjuk, pozícionáljuk a kurzort a megfelelő bejegyzésre, és felhasználjuk az aktuális pozíción található sorok tartalmát. A mozgatásokat a `FETCH` utasítás segítségével tehetjük meg. A `FETCH FIRST` ráállítja a kurzort az első rekordra. Ennek megfelelően a `FETCH LAST` az utolsóra. A következő rekordot a `FETCH NEXT`-tel érhetjük el. Mivel általában rekordról-rekordra lépkedünk, ezt az utasítást használjuk leggyakrabban. Aki szeret visszafele bejárni egy eredményhalmazt, annak jól jön a `FETCH PRIOR` utasítás, ami az előző sorra lép vissza. Valamivel izgalmasabb a `FETCH ABSOULTE n` és a `FETCH RELATIVE n`. A nevükből elég jól kiviláglik a feladatuk. A `FETCH ABSOULTE n` a kisimított eredményhalmaz n. rekordjára pozícionálja rá a kurzort, függetlenül attól, hogy hol ált éppen. A `FETCH RELATIVE n` pedig az aktuális pozíciótól tudja n távolságra elmozgatni a kurzort. Ezt a két utasítást nem minden típusú kurzorra lehet kiadni, de ezekről majd egy kicsit később.

```
FETCH NEXT FROM curTest
```

4. Amellett, hogy barangolunk a kurzorral és kiolvassuk az általa kijelölt sor tartalmát, még módosíthatjuk és törölhetjük is az adott sort. Ezeket a lehetőségeket ritkán használjuk ki, de azért jól jöhetnek még. Az érdekességük ezeknek a speciális `UPDATE` és `DELETE` utasításoknak, hogy a `WHERE` feltételben nem egy „hagyományos” sort kiválasztó feltételt adunk meg, hanem a `CURRENT OF kurzornév` kifejezést, amiben a kurzornév az általunk definiált kurzort reprezentálja. Azaz a módosító és törlő utasítások valahogy így néznek ki:

```
UPDATE tábla SET ... WHERE CURRENT OF kurzornév
DELETE tábla         WHERE CURRENT OF kurzornév
```

5. A Mór megtette a kötelességét, lezárjuk a kurzort. A későbbiekben újra megnyithatjuk anélkül, hogy újra deklarálnánk, de már nem olvashatunk ki rajra keresztül rekordokat, illetve nem pozícionálhatjuk a kurzort.

```
CLOSE curTest
```

6. Felszabadítjuk a kurzort. Miután megnézzük a kurzorok típusait, látni fogjuk, hogy a kurzorunk által bejárni kívánt eredményhalmaz fenntartása igen sok kiszolgálóoldali erőforrást köthet le. Emiatt érdemes azonnal felszabadítani, amint felhasználtuk az eredményeket.

```
DEALLOCATE curTest
```

A bejárás zegzugos részletei

Láttuk, hogy a `FETCH` utasítás variánsaival keresztül-kasul bejárhatjuk az eredményhalmazt. De honnak tudjuk, hogy a végére értünk? És hogy már az elején járunk? És azt honnan vesszük észre, hogy az éppen kiolvasni kívánt sort valaki más már törölte alólunk? Nos, ezekre a státuszinformációk kiolvasására hozták létre a `@@FETCH_STATUS` nevű függvényt (globális változót, ki hogy szereti). Ha a `@@FETCH_STATUS` értéke 0, akkor a megelőző `FETCH` utasítás sikeresen hajtódott végre, és felhasználhatjuk a kurzor által kijelölt rekordot. Ha -1-et kapunk vissza, akkor túlszaladtunk az eredményhalmazon, azaz vagy a legelső sor elé akartunk menni egy `FETCH PRIOR`-ral, vagy a legutolsón túlra a `FETCH NEXT`-tel. Ha ciklusban járjuk végig a sorokat, akkor erre szoktuk építeni a ciklus végét jelző feltételt. A -2 a legravaszabb. Ez vagy azért jön elő, mert az aktuális sort törölték, vagy pedig azért, mert úgy módosították az adott sort, hogy az már nem esik bele abba az eredményhalmazba, amit a deklarációnál használt `SELECT` jelöl ki. Például a `SELECT`-tel kiválasztjuk a budapesti alkalmazottakat, és az így leválogatott halmazban barangolunk a kurzorunkkal. Eközben Mariska a HR-ről átírja Nagy Elek lakcímét Szegedre. És mi a következő lépésben (`FETCH NEXT`) ki akarjuk olvasni Nagy Elek adatait, és kapunk egy nagy -2-t mert Elek már nem budapesti, így nem is lehet benne a kiinduló `SELECT` által leválogatott halmazban. Lássunk hát egy olyan példát, amiben végigmegyünk az összes alkalmazotton egy kurzorral:

```
DECLARE curTest CURSOR
FOR
SELECT
  EmployeeID, LastName, FirstName
FROM
  Employees
OPEN curTest
FETCH NEXT FROM curTest

WHILE @@FETCH_STATUS = 0
BEGIN
  FETCH NEXT FROM curTest
END

CLOSE curTest
DEALLOCATE curTest
```

Ez így nagyon szép, de mi történik a `FETCH NEXT`-ek során kiválasztott sorokkal? Ha a Query Analyser-ben kipróbáljuk az előbbi példát, akkor az alábbi kimenetet kapjuk:

```
EmployeeID  LastName             FirstName
----------- -------------------- ----------
1           Davolio              Nancy

(1 row(s) affected)

EmployeeID  LastName             FirstName
----------- -------------------- ----------
2           Fuller               Andrew

(1 row(s) affected)
...
```

Azaz minden egyes `FETCH NEXT` egy új eredményhalmazt generál! Hát ez minden, csak nem álom (hacsak nem rémálom) feldolgozni egy ügyfélalkalmazásból, arról nem is beszélve, hogy minden egyes `FETCH NEXT` eredményeképpen előállt eredményhalmaz egyenként át kell, hogy utazzon a hálózaton. Egy sima `SELECT` összes sora egy nagy csomagban (itt nem hálózati csomagról, hanem körülfordulásról van szó) utazik, míg a kurzor minden egyes sora külön csomagban. Ez aztán az erőforrás pazarlás netovábbja. Általában nem is használjuk így a kurzorokat, legalábbis a Transact SQL kurzorokat. Hisz milyen előnyét élveztük annak, hogy a

```
SELECT
  EmployeeID, LastName, FirstName
FROM
  Employees
```

helyett egy jó bonyolult kódot hordtunk össze? Semmit. Ilyen módon nem jól hasznosíthatók a kurzorok. Azonban a bejárás során érintett sorokat elraktározhatjuk változókba is, és ekkor lesz csak igazán nagy az örömünk. Nulla darab eredményhalmaz generálódik, a hálózaton csönd lesz, és mi hatékonyan, gyorsan dolgozunk a kurzorunkkal kiszolgálóoldalon. Hogyan is?
Deklarálunk lokális változókat, ezekbe rakjuk az aktuálisan felolvasott rekord mezőinek a tartalmát:

```
DECLARE @nEmployeeID INT
DECLARE @cLastName VARCHAR(20)
DECLARE @cFirstName VARCHAR(20)
```

A `FETCH ...` utasítás után elhelyezünk egy `INTO` kulcsszót, és felsoroljuk az előbbi változóinkat, amelyekbe szeretnénk belerakni a kurzor által „kiolvasott” rekord tartalmát:

```
FETCH NEXT FROM
  curTest
INTO
  @nEmployeeID, @cLastName, @cFirstName
```

A felsorolt változók sorrendje meg kell, hogy egyezzen a `SELECT` által generált eredményhalmaz elemeinek a sorrendjével, hogy egymásra találjanak az értékek.
Hogy kerek legyen a példánk, írjunk egy olyan kódot, ami összegyúrja egy nagy listává az összes alkalmazott nevét, azaz összefűzi őket egy sztringgé. Ezt elég nehéz, ha egyáltalán lehetséges megoldani „hagyományos” `SELECT` felhasználásával:

```
DECLARE @nEmployeeID INT
DECLARE @cLastName VARCHAR(20)
DECLARE @cFirstName VARCHAR(20)
DECLARE @cAllNames VARCHAR(4000)
SET @cAllNames = ''

DECLARE curTest CURSOR
FOR
SELECT
  EmployeeID, LastName, FirstName
FROM
  Employees
OPEN curTest

FETCH NEXT FROM
  curTest
INTO
  @nEmployeeID, @cLastName, @cFirstName

WHILE @@FETCH_STATUS = 0
BEGIN
  SET @cAllNames = @cAllNames +
      @cLastName + ' ' +  @cFirstName + ', '

  FETCH NEXT FROM
    curTest
  INTO
    @nEmployeeID, @cLastName, @cFirstName
END

CLOSE curTest
DEALLOCATE curTest
--Teszt:
SELECT @cAllNames AS Names
```

Az utolsó teszt `SELECT` eredményeként láthatjuk, hogy a `@cAllNames` tartalma:

```
Davolio Nancy, Fuller Andrew, Leverling Janet, Peacock Margaret, Buchanan Steven, ...
```

Kurzortípusok

Mint korábban említettem, többféle kurzort hozhatunk létre, annak megfelelően, hogy mi a célunk a keletkező eredményhalmazzal. Nézzük végig a lehetőségeket, és azt, hogy mikor melyikkel érdemes élni.

Statikus kurzorok

Ha a kurzor deklarációja során a statikus típust kérjük, akkor az SQL Server végrehajtja a kért lekérdezést, és a teljes eredményhalmazt elhelyezi a tempdb-ben, egy ideiglenes táblában. Azaz kapunk egy *másolatot* a lekérdezés eredményéből, így a megnyitás után akár le is törölhetik az összes sort a kiinduló táblából, mi vidáman lépkedünk a másolaton. Azaz ennél a kurzornál soha nem lesz a `@@FETCH_STATUS` értéke –2, nem lapátolják ki alólunk a sorokat. Más kérdés, hogy nem is fogjuk fel, hogy közben elszaladt mellettünk a világ. Mivel másolaton dolgozunk, az aktuális sort nem is módosíthatjuk, azaz a korábban ismertetett `UPDATE`, `DELETE` nem használható erre a kurzorra.
Deklaráció:

```
DECLARE Kurzornév CURSOR STATIC
FOR ...
```

Mikor jó egy statikus kurzor? Hmm. Igazából nem tudok jó felhasználást. Ha minden sorra szükségünk van, akkor kár az egész táblát átmásoltatni a tempdb-be, egyszerűbb lekérdezni a teljes eredményhalmazt a hagyományos módon, mindenféle kurzor felhasználását mellőzve (ezt hívják Default Result Set-nek). Ne nagyon használjuk a statikus kurzort, csak ha valami különleges indokunk van rá.

Keyset kurzorok

A keyset kurzor már sokkal gazdaságosabban bánik a tempdb-vel, mint a statikus kurzor. Nem a kurzordeklarációban előírt teljes sorokat tárolja le egy átmeneti táblába, csak az egyes sorok egyedi azonosítóit. Ebből következően a lekérdezésben szereplő táblának kell lennie legalább egy olyan oszlopának, ami garantáltan azonosítja a sorokat, garantáltan egyedi. Ez lehet egy `UNIQUE` index, egy `PRIMARY KEY` vagy egy clustered index (lehet, hogy a clustered index értékei nem egyediek, de az SQL Server minden clustered index bejegyzéshez hozzácsatol egy belső azonosítót, amitől azok belülről egyediek lesznek).

Mivel csak a kulcsokat tároljuk a tempdb-ben, a kiválasztott valódi sorokat lehet, hogy valaki más módosította a kurzor megnyitása óta. Ilyenkor a megváltozott mező értékeket kapjuk vissza. Ha valamelyik sort közben törölték, akkor a `@@FETCH_STATUS` –2 jelzi, hogy már nem létezik a sor, amit ki szeretnénk olvasni. Ha olyan értékeket szúrnak be a táblába, aminek a deklarációban szereplő `SELECT` alapján benne kellene lennie a kurzor eredményhalmazában, nos, azokat nem fogjuk látni. A legenerált kulcshalmaz fix, nem változik, csak ha lezárjuk, és újra megnyitjuk a kurzort. Formátum:

```
DECLARE curTest CURSOR KEYSET
FOR ...
```

Mikor érdemes használni ezt a kurzort? Akkor, ha csak a kiválasztott sorokkal szeretnénk foglalkozni, és nem érdekel bennünket az, hogy esetleg közben további sorokat szúrtak be egy táblába, de érdekel bennünket a kiválasztott sorokat érintő változtatási kísérlet.

Dinamikus kurzorok
Láttuk, hogy a statikus kurzornak mindegy mi történik a kiinduló adatokkal, mi vígan olvassuk a legenerált másolat eredményhalmazt. A keyset kurzor már figyelmesebb, a sorokon végzett `UPDATE` és `DELETE` utasításokat már visszatükrözi a kurzort használó alkalmazásnak. De az `INSERT`-eket nem, azaz nem jelennek meg a kurzor megnyitása *után* beszúrt sorok a kurzor eredményhalmazában. Érezzük, hogy már csak egy lépés van hátra egy olyan kurzortípushoz, ami az összes változtatást átvetíti a kurzor eredményhalmazára. Természetesen ez a dinamikus kurzor. Ez nem nyúl a tempdb-hez (legalábbis nem jelentős mértékben). Nem hoz létre átmeneti táblákat. A kurzorral sétáló alkalmazás észreveszi, ha új sort szúrnak be a táblába, mert az új sorok megjelennek a kurzor eredményhalmazában! Nem véletlenül dinamikus a neve. A deklaráció nem fog meglepetést okozni:

```
DECLARE curTest CURSOR DYNAMIC
```

Valójában elég nehéz elképzelni, hogy az SQL Server tervezői hogyan valósították meg ezt a funkcionalitást. Biztos nem volt egyszerű. De sikerült nekik, így van egy szuperjó kurzorunk, ami nagyon kicsi kiszolgálóoldali terhelést okoz. Akkor jön igazán jól, ha egy lekérdezés kimenete nagyon nagy eredményhalmazt adna vissza, de nekünk ebből csak az első, például 50 sorra van szükségünk. Az SQL guruk ilyenkor `TOP 50`-ért kiáltanak, amivel kapásból egy 50 eleműre vágott eredményhalmazt kapunk. De mi van, ha nekünk csak az 550. és a 600. sor közötti rekordok kellenek? A `TOP 600`-zal leválogatom az első hatszáz eredményt (ami azt jelenti, hogy azok át is csurognak a hálózaton az ügyfélprogramra, a modemesek nagy örömére), és mi fütyülve eldobjuk az első 550-et, hogy élvezzük az utolsó ötvenet. Mi nem akarunk ilyen pazarlók lenni, és megbecsüljük a modemezőket is. Ilyenkor jön jól a dinamikus kurzor.
Mivel a dinamikus kurzor eredményhalmaza illékony, másodpercről-másodpercre változik, a `FETCH ABSOULTE` ennél a kurzornál nem támogatott. Hiszen a `FETCH ABSOULTE 100` ebben a pillanatban lehet, hogy teljesen mást sort választ ki, mint 10 másodperc múlva, ha közben 500 felhasználó püföli a tábla tartalmát. Ennek ellenére (surprise :), a `FETCH RELATIVE` működik. A `FETCH FIRST` is, így a kettőből már össze lehet rakni egy `FETCH ABSOLUTE`-ot. Hogy miért van ez így? Nem tudom. De működik, és ezt nagyon jól ki tudjuk használni.
Az eddigiekből az következik, hogy a dinamikus kurzornak kell lenni a leggyorsabbnak, hisz ez nem épít semmilyen átmeneti táblát. Ez egészen addig igaz, amíg csak egy táblából kérdezünk le. Amint illesztéssel (`JOIN`) több táblát összekapcsolunk lehet, hogy gyorsabb az elején felépíteni egy átmeneti táblát az összekapcsolt eredményekből, mint mindig menet közben összekapcsolni újra és újra a táblákat. Magyarra fordítva illesztett táblák esetén nem biztos, hogy a dinamikus kurzor a leggyorsabb.

Forward only kurzor
Az előbbi három kurzor alapértelmezésben támogatja azt, hogy előre-hátra mozogjunk vele az eredményhalmazban (`SCROLL`). Azonban az esetek igen jelentős részében csak előre akarunk végighaladni a kurzorral, mert egyszerűen kiíratjuk az eredményeket egy riportban vagy egy weblapon. Ebben az esetben adhatunk egy kis könnyítést az SQL Servernek azzal, hogy jelezzük, hogy a kurzorunkkal csak előre fogunk lépegetni. Ezáltal a kiszolgálónak kevesebb munkába kerül nyilvántartani a pozíciónkat.
A forward only nem egy negyedik kurzortípus, hanem az előző három kiegészítése, pontosítása. Pl. egy keyset kurzor egyirányúsítása:

```
DECLARE curTest CURSOR FORWARD_ONLY KEYSET
FOR ...
```

Fast forward only kurzorok
Ez a negyedik típusú kurzorunk. Ez egyetlen hálózati körülfordulásban visszaküldi a teljes eredményhalmazt a kliensre, majd automatikusan zárja a kurzort. Így az ügyfélprogramnak gyakorlatilag csak kérni kell az adatokat, és nem kell foglalkozni a kurzor megnyitásával - bezárásával. Az ADO dokumentáció hallgat erről a lehetőségről, de ODBC-n keresztül el lehet érni. Mivel azonban manapság már nem nagyon használunk ODBC-t, ezzel a lehetőséggel nem nagyon foglalkozunk. Mellesleg az ADO „sima” forward only kurzora kísértetiesen hasonlít viselkedésben erre a kurzorra. Nem véletlenül. Ha ADO-n keresztül csak olvasható, forward only kurzort kérünk, akkor az egy fast forward only (régebbi nevén firehouse) kurzor lesz.

Jogosan kérdezheti bárki, hogy miért foglalkozunk ennyit a kurzorokkal, amikor olyan ritkán használjuk. Biztos? Lehet, hogy a Transact SQL kurzorok felhasználása elég speciális, és csak kevesen fognak belegabalyodni. Azonban mi történik akkor, amikor egy ügyfélprogram ADO vagy ODBC segítségével végrehajt egy lekérdezést az SQL Serveren? Hmmm? Hogyan nyissuk meg a kapcsolatot a szerver felé? Ja, hogy lehet valamit állítani a lekérdezés végrehajtásakor? És még valami kurzortípust is? Ejha, nézzünk csak ennek a körmére!

API kurzorok

Amikor ADO segítségével végrehajtunk egy lekérdezést az SQL Serveren, akkor két választási lehetőségünk van.

1. Teljes egészében letöltődik az egész eredményhalmaz a kliensre, és ott navigálunk a rekordok között – ekkor beszélünk kliensoldali kurzorról.
2. Jelezzük, hogy kiszolgálóoldali kurzort szeretnénk használni, és akkor csak azok a rekordok kerülnek át az ügyféloldalra, amelyekre ténylegesen rápozícionálunk, és kiolvasunk.

A kliensoldali kurzorok akkor hatékonyak, ha kis (<1000 sor) eredményhalmazokkal dolgozunk, és az összes sorral szeretnénk dolgozni. Ilyenkor a teljes eredményhalmaz - kiszolgálóoldali kurzor felhasználása nélkül - egy kötegben átmegy a kliensoldali adatbázismeghajtó programra, ami implementálja az eredményhalmaz navigálásához szükséges függvényeket. A kiszolgáló gyorsan megszabadul a munkától, a zárolások gyorsan feloldódnak. Az ügyfélprogram akármeddig dolgozhat a rekordokon, a szervert nem terheli a ténykedése. És ami még fontos, a navigáció nem generál járulékos hálózati forgalmat.
A kiszolgálóoldali kurzorok akkor használhatók ki jól, ha a lekérdezés eredményhalmaza nagyon nagy, és nem akarjuk feldolgozni az egészet, csak egy részét. A legtöbb böngésző, listázó típusú adat megjelenítés ilyen. Így a kliensprogramot nem árasztja el a kiszolgáló megabájt méretű eredményhalmazzal, megint csak a lassú vonalak végén ülők örömére, és a bérelt vonali szolgáltatók bánatára.
A kiszolgálóoldali kurzorok esetén ugyanaz a választékunk, mint amint a Transact SQL kurzoroknál már megnéztünk.
Hogyan lehet elképzelni az ADO által megvalósított kurzort? Valahogy úgy, hogy a lekérdezés végrehajtásakor a távolban, a kiszolgálóoldalon létrejön egy kurzor. Nem a `DECLARE CURSOR` és az `OPEN` utasításokkal, hanem speciális, csak erre a célra használat „ál" tárolt eljárásokkal, mint az sp_cursor és társai. Ezek a tárolt eljárások a valóságban nem is léteznek (bár úgy látszanak), hanem az SQL Server kernel tudja, hogy egy adatbázismeghajtó program kiszolgálóoldali kurzort akar létrehozni, és ennek megfelelően belül létrehozza a kurzort. További speciális tárolt eljárások hívásával az ügyféloldali adatbázismeghajtó program `FETCH` és egyéb kurzorműveletet tud végre hajtatni a létrehozott kurzoron a távolban, a kiszolgálón. Ezeket könnyedén megfigyelhetjük az SQL Profiler felhasználásával.
Ezeket az adatbázismeghajtó program által létrehozott kurzorokat API kurzoroknak nevezzük. Nem úgy, mint a Transact SQL kurzorokkal, ezekkel nap mint nap találkozunk, vagy tudatosan, vagy nem, hisz az ügyfélprogram, adatbázisműveletek használják őket.

Írjuk meg az altavistát!

Zárásul tegyük fel a koronát a tudásunkra. Tervezzünk egy olyan alkalmazást, ami képes végrehajtani egy paraméterezett lekérdezést, célszerűen olyat, aminek nagyon nagy az eredményhalmaza (például az altavistán rákeresünk a sex szóra :). Azután ebből a halmazból jelenítsük meg a 200 és a 220. sor közötti rekordokat.
Egy valódi, például webes alkalmazásban megjeleníténénk egy sorszámozott menüt, amivel a felhasználó közvetlenül kérheti a megfelelő lapok megjelenítését. Valami ilyesmire gondolok:
20 40 60 80 100 120 140 160 180 200 >>400>>
A megoldáshoz az ADO kurzoraihoz folyamodunk. (Elnézést azoktól, akik szeretik a tiszta SQL kódokat, de most egy kicsit át kell mennünk ügyféloldalra, és VB-re.)
Mivel a kedvenc Northwind adatbázisomban nincs elég nagy tábla, ezért létre fogunk hozni nagyon nagy teszt táblákat a megfelelő méretű eredményhalmaz reményében. Az `Order Details` tábla 2155 sorból áll. Ez elég lesz az első referenciamérésünkhöz, azonban gyengus lenne demonstrációs célra, hisz azt ígértem a cikk elején, hogy milliós nagyságrendű táblából fogunk kiválogatni sorokat ezred másodpercek alatt. Lássunk egy teszttáblát, amit a méréseinkhez fogunk felhasználni:

```
CREATE TABLE BigTable1000000(
  nID INT NOT NULL PRIMARY KEY IDENTITY(1,1),
  nCol1 INT NOT NULL,
  nCol2 INT NOT NULL,
  cText1 VARCHAR(500),
  cText2 VARCHAR(500))

--Tartalom generáló kód a weben: [1]
```

Nézzük meg a megjelenítést végző ügyféloldali teszt kódunkat, csak a legfontosabb részleteket tanulmányozva.
Deklaráljuk a lapozást definiáló paramétereket!

```
Dim nRecordNumber
Dim nPosition
nPosition = 200    'Ez a kiíratandó első rekord
nRecordNumber = 20 'Ennyi rekodot írunk ki
```

Kapcsolódjunk rá az SQL Serverre!

```
Dim conTest
'Létrehozzunk az ADO Connection objektumot
Set conTest = CreateObject("ADODB.Connection")
'Megadjuk a megfelelő connection string-et
conTest.ConnectionString = "Provider=..."

'kiszolgálóoldali kurzort használunk!
conTest.CursorLocation = adUseServer

'Rákapcsolódunk a kiszolgálóra
conTest.Open
```

Létrehozzuk a nagy eredményhalmazt generáló ADO parancsot.

```
Dim cmdList, rsOrders
'Létrehozunk egy ADO Command objektumot
Set cmdList = CreateObject("ADODB.Command")
'Hozzákapcsoljuk a már felépített Connection-höz
Set cmdList.ActiveConnection = conTest
'Ez az SQL parancs generálja a négymilliós
'eredmény halmazt
cmdList.CommandText = "SELECT ..."
```

Az eredményrekordokat tároló recordset létrehozása:

```
'Az itt létrehozott ADO Recordset objektum
'fogja tárolni a kiszolgálóról érkező
'eredményhalmazt
Set rsOrders = CreateObject("ADODB.Recordset")
'Megjelöljük a rekordok forrását
'(a lekérdezésünket)
Set rsOrders.Source = cmdList
```

Amire kimegy a játék:

```
'Minden jó anyja: a dinamikus kurzor
rsOrders.CursorType = adOpenDynamic

'Az eredményhalmaz (és a kurzor) megnyitása
rsOrders.Open
rsOrders.CacheSize = nRecordNumber
```

Álljunk meg egy pillanatra! Mi az a `Recordset.CacheSize`? Ezzel állítjuk be a kurzorunk szélességét. A Transact SQL-es példánk kurzora 1 széles volt, azaz minden egyes, a következő rekordra navigáló lépés újabb és újabb adatbázisművelettel járt. Egy, az SQL Serveren futó programnál ez nem is probléma, de egy ügyfél-kiszolgáló programban az lenne a jó, ha egyszerre több rekord is lejönne a hálózaton, így ritkábban kellene hozzáférni a kiszolgálóhoz további rekordokért. Erre való a `CacheSize` jellemző. Ha beállítjuk 10-re, akkor a kurzorunk 10 kövérségű lesz, így az első elemre navigáláskor nem csak az első rekord töltődik le, hanem további 9 is. Így a következő 9 elemre történő navigáció nem igényel újabb szerverhez fordulást.
Mozogjunk előre a 200. rekordra!

```
rsOrders.Move(nPosition)
```

Sok példaprogram igen helytelenül azt sulykolja belénk, hogy használjuk a

```
rsOrders.AbsolutePosition = nPosition 'nem!
```

utasítást a megfelelő rekordra való mozgásra. Azonban mi tudjuk, hogy a dinamikus kurzornál nincs `FETCH ABSOULTE`, így ez az utasítás is elszáll a

```
ADODB.Recordset (0x800A0CB3)
Current Recordset does not support bookmarks. This may be a limitation of the provider or of the selected
cursortype.
```

hibával. A hiba második része a mi problémánk. Statikus vagy keyset kurzorral menne az abszolút pozícionálás, csak akkor meg a teljesítmény lesz nagyon siralmas, ahogy azt majd hamarosan látjuk.

Ránavigáltunk a 200. rekordra, most már csak végig kell menni a következő 20-on:

```
Dim nDisplayCount
nDisplayCount = 0
Do While (NOT rsOrders.EOF) And (nDisplayCount < nRecordNumber)
  Print rsOrders("OrderID")
  nDisplayCount = nDisplayCount + 1
  If nDisplayCount < nRecordNumber Then rsOrders.MoveNext
Loop
```

Addig gyalogolunk előre a rekordokban, amíg vagy a végére nem érünk, vagy ki nem írattuk a kívánt számú sort.
És természetesen takarítunk magunk után, mert nem szeretjük a memóriát suttyomban fogyasztgató alkalmazásokat:

```
rsOrders.Close
Set rsOrders = Nothing
Set cmdList = Nothing
conTest.Close
Set conTest = Nothing
```

Nem csak a szánk jár, avagy a végső terheléstesztt

És most jöjjenek az izgalmas teljesítmény mutatók, ahol kiderül, hogy megbukik-e az elmélet, vagy megerősítést nyer (mivel ez a cikk megjelent, vélhetőleg beigazolódott :).
A teszt környezet egy 450 MHz-es PII, 128 Mbyte RAM-mal, Windows 2000 Advanced Serverrel, az ügyfél és a szerver egy gépen volt. Az ügyfél alkalmazás egy ASP-ben megvalósított VBScript kód, amely a [1] címen élőben ki is próbálható, valamint a teljes forráskód is letölthető. Az alkalmazás a fentebb ismertetett példa kibővített változata, de a lényege azonos azzal.
Az SQL Server végrehajtási időket SQL Server Profiler-rel mértem, összeadva a parancs végrehajtásához szükséges összes művelet által használt időket. Az ügyfél végrehajtási időt az ASP kód kezdete és vége közt mértem, ebben benne van a kapcsolat kiépítése, a parancs végrehajtása, a rekordokon való végig gyaloglás, és a kapcsolat lezárása is.
Az első lekérdezést az `Order Details` tábla ellenében hajtottam végre.

```
SELECT OrderID, Quantity FROM [Order Details]
```

Ez a tábla még elég kicsi ahhoz (2155 sor), hogy az összes kurzort kipróbálhassuk vele - véges türelemmel is.
A tesztben a leválogatott adatokban elmozogtam a 200. rekordig, majd onnan a következő húszat írattam ki.
Lássuk az így mért eredményeket:

Kurzor	SQL Server végrehajtási idő [sec]	Ügyfél végrehajtási idő [sec]	Logikai diszk művelet [lap]
Dinamikus	<0,001	0,03	106 olvasás 2 írás
Forward only	0,01	0,02	10 olvasás 0 írás
Keyset	0,110	0,13	4521 olvasás 11 írás
Statikus	0,05	0,07	4374 olvasás 7 írás

Mit mondanak az eredmények? A dinamikus kurzort a szerver nagyon gyorsan végrehajtja, gyakorlatilag a végrehajtási időre az SQL Server Profiler 0-t ad vissza. Ennek ellenére kliens oldalról szemlélve a forward only kurzor teljesített legjobban, és a logikai olvasásokban is az vezet. Nem véletlenül van az, hogy kis táblák tartalmának egyszerű kilistázására javasolják a forward only kurzort. Nagyon kicsi szerver oldali terhelést okoz, és a kliens nagyon gyorsan megkapja az eredményhalmazt. A profiler-ben látszik, hogy valójában ilyenkor az OLE-DB szolgáltató nem is nyit meg kurzort a szerveren, csak az alapértelmezett eredményhalmazt tölti le. Valamit azért elmond a szervernek a lekérendő eredményhalmaz hosszáról, ami azonban nem látszik a profiler-ben. Viszont az látszik, hogy minél több rekorddal léptetjük előre a (nem is létező) kurzort, annál több lapot olvas be.
A táblázatot szemlélve egy további furcsa eredmény üti meg a szemünket. A keyset kurzor lassabb, mint a statikus kurzor! Ez ellentétben áll azzal, amit a működése alapján várnánk tőle. Egy tippem van, miért van ez. A táblánk kicsi (~2000 sor). Ennyit a szerver pillanatok alatt átmásol a tempdb-be, ahonnan a kliensnek már nagyon gyorsan, közvetlenül kiszolgálja a kívánt sorokat. A keyset kurzornál a kulcsokat még gyorsabban be tudja másolni a tempdb-be, mint az előbbi esetben, azonban kiolvasáskor minden egyes kulcshoz elő kell bányászni az adatokat az eredeti táblából. Ilyen kisméretű táblánál nagyobb a bányászás (bookmark lookup) költsége, mint átmásolni az összes eredményt egy másik táblába. Ez nem látszik triviális módon a lekérdezésből.
Töltsük fel a már emlegetett teszt táblánkat 10000 sorral, és válogassuk le az egészet (a teszt adatokat generáló script – terjedelmi okokból – a [1] címen érhető el):

Kurzor	SQL Server végrehajtási idő [sec]	Ügyfél végrehajtási idő [sec]	Logikai diszk művelet [lap]
Dinamikus	<0,001	0.02	105 olvasás 0 írás
Forward only	0,15	0,15	5 olvasás 0 írás
Keyset	0,22	0,22	20422 olvasás 0 írás
Statikus	0,25	0.26	20552 olvasás 0 írás

A dinamikus kurzor fittyet hány a tábla méret változtatásra, ő ugyanolyan gyors maradt. A forward only kurzor jelentősen lassult, azonban emberi léptékkel nézve még mindig villámgyors. A másik két kurzor már kezd belefulladni az adatokba. A lapolvasások száma a tábla sorainak számával arányban nőtt. Ennek van egy nagyon fontos tanulsága. Ha elkészítünk egy alkalmazást, amiben nem ügyelünk a kurzor típusára, és az statikus vagy keyset lesz, akkor eleinte jó gyorsnak fog tűnni a program, mert az összes sort képes benntartani a szerver a fizikai memóriában, és ott a mai processzorok nagyon gyorsan tudnak keresni. Ha azonban az alkalmazás éles üzeme során az adatok elkezdenek záporozni a táblákba, akkor lehet, hogy egy hónap múlva az alkalmazás kifekszik, jönnek az időtúllépésről szóló üzenetek, és a haragos ügyfelek.
A helyzet akkor lesz csak igazán drámai, ha akkorára dagadnak a táblák, amekkorát már soha nem tud egyben benntartani a szerver a memóriában, így elindul a merevlemez reszelés. Ekkor mutatja csak ki a foga fehérjét a rossz tervezés. Nézzük csak meg egy 1,000,000 soros táblával az előbbi számok változását:

Kurzor	SQL Server végrehajtási idő [sec]	Ügyfél végrehajtási idő [sec]	Logikai diszk művelet [lap]
Dinamikus	<0,001	0.03	108 olvasás 0 írás
Forward only	0.17	0.18	17 olvasás 0 írás
Keyset	83.11	83.26	2,845,123 olvasás 3276 írás
Statikus	281.466	288.68	2,988,655 olvasás 8432 írás

Gyakorlatilag a két utolsó kurzor kiesett a játékból, az ADO alapértelmezett 30 másodperces parancs idejéből már rég kifutottak volna a parancsaink. És most csak egy felhasználó terhelte a szervert, nem 1000! Egy élő rendszer ebben az esetben egyszerűen leállna. Sajnos ilyen tervezési hibák miatt elég gyakori, hogy alaptalanul szidnak egy rendszert, merthogy az már egymillió sort sem tud rendesen kezelni. Bezzeg a pityipalkó cég terméke...
Akkor tessék csak megnézni azt az első sort! Végrehatási idő a kliens oldalon mérve is 30 milliszekundum! Pedig a tábla már egymillió sort tartalmaz, ami messze meghaladja az én gépen RAM-jának tárolókapacitását. De nem is ez a lényeg. Hiába nőtt a tábla mérete az első esethez képest az 500 szorosára, a végrehajtási idő gyakorlatilag nem változott. Ezért szeretem én úgy a dinamikus kurzort. Szeressék Önök is, és éljenek vele!

Zárszó

A [1] címen nem csak példakódok találhatók, hanem minden, a cikk második felében szereplő kódot interaktívan kipróbálhat a kedves Olvasó, maga választva ki a mérés összes paraméterét, a kurzor típusokat stb.
A kurzorok elméletének egyik sarkalatos pontja, a zárolás teljesen kimaradt a mostani cikkből, a szokásos terjedelmi okok miatt. Azonban a jövő hónapban egy teljes cikket szánok a zárolások lelki világának megértésére, mert - a kurzorokhoz hasonlóan - a nem megfelelő alkalmazásuk szintén ledegradálhatja az alkalmazásunk teljesítményét - és vele együtt minket is.
Jó kurzorizálást kívánok a web-en és a földön egyaránt!

A cikkben szereplő URL-ek:
[1] http://technet.netacademia.net/feladatok/sql/cursor

Soczó Zsolt MCSE, MCSD, MCDBA
Netacademia Kft.