6.4. Beágyazott ciklusú összekapcsolások

Mielőtt a később sorra kerülő szakaszokban áttérnénk az összetettebb algoritmusokra, foglalkozzunk először az összekapcsolás operátor „beágyazott ciklusú”-nak nevezett algoritmuscsaládjával. Ezek az algoritmusok bizonyos értelemben „másfél menetesek”, mivel minden változatban a két argumentum egyikéhez tartozó sorokat csak egyszer olvassuk be, a másik argumentumot viszont többször olvassuk. Beágyazott ciklusú összekapcsolások használhatók bármekkora méretű relációra, nem szükséges, hogy az egyik reláció elférjen a memóriában.

6.4.1. Sor alapú beágyazott ciklusú összekapcsolás

Kezdjük a tárgyalást a beágyazott ciklusú témakör legegyszerűbb változatával, ahol a ciklusok a kérdéses relációk minden egyes sorára ismétlődnek. Ebben a *sor alapú beágyazott ciklusú öszekapcsolásnak* nevezett algoritmusban a

$R(X, Y) |><| S(Y, Z)$

összekapcsolást a következőképpen számítjuk ki:

```
FOR S minden egyes s sorára DO
   FOR R minden egyes r sorára DO
      IF r és s összekapcsolható egy t sorrá THEN
         t kiírása;
```

Ha nem figyelünk arra, hogy hogyan pufferezzük az R és S relációk blokkjait, akkor a fenti algoritmus akár $T(R)T(S)$ számú lemez I/O-műveletet is igényelhet. Ugyanakkor sok esetben az algoritmus módosításával sokkal alacsonyabb költség is elérhető. Ilyen eset például az, amikor az R-beli összekapcsoló attribútum(ok)ra indexet tudunk használni, hogy megkeressük R sorai között azokat, amelyek megfelelnek S egy adott sorának, és ilyenkor nem kell a teljes R relációt beolvasnunk. Az index alapú összekapcsolásokat a 6.7.3. részben ismertetjük. Egy második javítási lehetőség sokkal figyelmesebben veszi szemügyre azt, hogy R és S sorai miként vannak megosztva a blokkok között, és a lehető legtöbb memóriát használja fel a lemez I/O-műveletek számának csökkentésére, amikor a belső cikluson végighalad. Ezt a blokk alapú beágyazott ciklusú összekapcsolási változatot a 6.4.3. részben tárgyaljuk.

6.4.2. Egy iterátor a sor alapú beágyazott ciklusú összekapcsoláshoz

A beágyazott ciklusú összekapcsolás egyik előnye az, hogy jól beleillik az iterátoros megközelítésbe, és így egyes esetekben segít elkerülni a köztes relációk lemezen való tárolását. Ezt majd a 7.7.3. részben fogjuk látni részletesebben. Az $R |><| S$ iterátora könnyen felépíthető R és S iterátoraiból, amelyeket a 6.2.6. részhez hasonlóan most is `R.Open`, … stb.-vel jelölünk. A beágyazott ciklusú összekapcsolás három iterátor függvényének a kódja a 6.12. ábrán látható. Feltételezzük, hogy sem az R, sem az S reláció nem üres.

6.4.3. Egy algoritmus a blokk alapú beágyazott ciklusú összekapcsoláshoz

Javíthatunk a 6.4.1. részben megismert sor alapú beágyazott ciklusú összekapcsoláson, ha $R |><| S$-t a következő módon számítjuk ki:

1. Mindkét argumentum relációnál megvalósítjuk a blokkonkénti hozzáférést.
2. A lehető legtöbb memóriát használjuk az S reláció, azaz a külső ciklushoz tartozó reláció sorainak tárolására.

Az 1. pont biztosítja, hogy amikor a belső ciklusban végigmegyünk R sorain, akkor R beolvasásához a lehető legkevesebb lemez I/O-műveletet használjuk fel. A 2. pont révén lehetőségünk nyílik arra, hogy R minden egyes beolvasott sorát ne csupán egy S-beli sorral kapcsoljuk össze, hanem annyival, amennyi csak elfér a memóriában.

```
FOR S minden M-1 blokkból álló darabjára DO BEGIN
   olvassuk be ezeket a blokkokat a memóriapufferekbe;
   a sorokat szervezzük egy keresési struktúrába,
   melynek kulcsa az R és S közös attribútumai;
   FOR R minden egyes b blokkjára DO BEGIN
      olvassuk be b-t a memóriába;
      FOR b minden t sorára DO BEGIN
         keressük meg S azon sorait a memóriában,
         amelyek kapcsolódnak t-vel;
         írjuk ki e sorok t-vel való összekapcsolását;
      END ;
   END ;
END ;
```

6. 13. ábra. *A beágyazott ciklusú összekapcsolás algoritmusa*

Ugyanúgy mint a 6.3.3. részben, tegyük fel, hogy $B(S) \leq B(R)$, de emellett tegyük fel még azt is, hogy $B(S) > M$, azaz egyik reláció sem fér be teljesen a memóriába. Ismétlődően beolvassuk S-nek $M - 1$ darab blokkját a memóriapuffereibe. S-nek a memóriában lévő sorai számára létrehozunk egy olyan keresési struktúrát, amelynek kulcsa megegyezik R és S közös attribútumaival. Ezt követően végigvesszük R összes blokkját, azokat egyenként a memória legutolsó blokkjába beolvasva. Ha ez megtörtént, akkor R blokkjának összes sorát összehasonlítjuk S éppen memóriában lévő blokkjainak összes sorával. Az összekapcsolódó sorok esetén az összekapcsolt sort a kimenetbe tesszük. A most ismertetett algoritmus beágyazott ciklusú struktúráját a 6.13. ábra formális bemutatása jól szemlélteti.

A 6.13. ábra programja látszólag három egymásba ágyazott ciklust tartalmaz. Ha azonban a kódot a helyes absztrakciós szinten nézzük, akkor valójában csak két ciklust találunk. Az első, a külső ciklus S sorain fut végig, a másik két ciklus pedig R sorain fut. Ez utóbbi folyamatot annak hangsúlyozására tüntettük fel két ciklusból állóként, hogy az a sorrend, amelyben R sorait végigvesszük, nem tetszőleges. Ezeket a sorokat blokkonként kell végigvennünk (a második ciklus szerepe), és egy blokkon belül annak összes sorát végignézzük, mielőtt átlépnénk a következő blokkra (a harmadik ciklus szerepe).

6.14. példa: Tegyük fel, hogy $B(R) = 1000$, $B(S) = 500$ és legyen $M = 101$. 100 darab memóriablokkot fogunk használni S-nek 100 blokkos darabokban történő pufferezésére, így a 6.13. ábra külső ciklusát ötször kell végrehajtani. Minden egyes iteráció alkalmával 100 lemez I/O-művelettel olvassuk be S egy darabját, majd R-et teljes egészében be kell olvasnunk a második ciklusban, amihez 1000 lemez I/O-műveletre van szükség. Így az összes lemez I/O-műveletek száma 5500.

Vegyük észre, hogy ha R és S szerepét felcseréltük volna, akkor az algoritmus valamivel több lemez I/O-műveletet használt volna fel. Ekkor 10 alkalommal hajtódna végre a külső ciklus, alkalmanként 600 lemez I/O-műveletet végezve, azaz az összes I/O-műveletek száma 6000 lenne. Általában igaz, hogy a kisebb relációnak a külső ciklusban való használata némi előnyt jelent. □

A 6.13. ábra algoritmusát néha „beágyazott blokkos összekapcsolás”-nak nevezik. Mi továbbra is megmaradunk az egyszerű *beágyazott ciklusú összekapcsolás* (nested-loop join) névnél, hiszen a beágyazott ciklusú séma gyakorlatban leggyakrabban megvalósított változatáról van szó. Ha szükség van a 6.4.1. rész sor alapú beágyazott ciklusú összekapcsolásától való megkülönböztetésre, akkor a 6.13. ábra sémájára mint „blokk alapú beágyazott ciklusú összekapcsolás”-ra fogunk hivatkozni.

6.4.4. A beágyazott ciklusú összekapcsolás elemzése

A 6.14. péda elemzése megismételhető tetszőleges $B(R)$, $B(S)$ és M esetén. Tegyük fel, hogy S a kisebbik reláció, és a darabok, vagyis a külső ciklus iterációinak száma $B(S)/(M - 1)$. Minden iteráció során S-nek $M - 1$ blokkját és R-nek $B(R)$ számú blokkját olvassuk be. A lemez I/O-műveletek száma tehát

$$\frac{B(S)}{M-1}\left(M-1+B(R)\right)$$

vagy átalakítva

$$B(S)+\frac{B(S)B(R)}{M-1}$$

Feltéve, hogy mind M, mind $B(S)$ és $B(R)$ nagy, és közülük M a legkisebb, a fenti formula $B(S)B(R)/M$-mel közelíthető. Ez más szavakkal azt jelenti, hogy a költség a két reláció méretének szorzata és a rendelkezésre álló memória hányadosával arányos. Sokkal jobban járunk akkor, ha mindkét reláció nagy, bár észrevehető, hogy megfelelően kis példák esetén (mint pl. a 6.14. volt) egy beágyazott ciklusú összekapcsolás költsége nem sokkal haladja meg egy egymenetes összekapcsolását, amely ez esetben 1500 lemez I/O-művelet lenne. Valóban, ha $B(S) \leq M-1$, akkor a beágyazott ciklusú összekapcsolás a 6.3.3. rész egymenetes összekapcsolási algoritmusával azonossá válik.

Noha általában véve a beágyazott ciklusú nem a rendelkezésünkre álló lehető leghatékonyabb összekapcsolási algoritmus, meg kell jegyeznünk azt is, hogy néhány korai ABKR esetében ez volt az egyetlen elérhető típus. Még manapság is szükség van rá bizonyos esetekben, hatékonyabb összekapcsolási algoritmusok szubrutinjaként, például amikor az egyes relációk nagyszámú sorához az összekapcsoló attribútum(ok) ugyanazon értéke tartozik. Olyan esetre, amikor a beágyazott ciklusú összekapcsolás alapvető fontosságú, a 6.5.5. részben láthatunk példát.