## 9.1. Soros és sorba rendezhető ütemezések

A konkurenciavezérlés tanulmányozását azzal kezdjük, hogy megvizsgáljuk, a konkurensen végrehajtott tranzakciók milyen feltételekkel tudják megőrizni az adatbázis-állapot konzisztenciáját. Az alapfeltevésünk, amelyet „helyességi elv”-nek (correctness principle) neveztünk, a 8.1.3. részben az volt, hogy ha minden egyes tranzakciót elkülönítve hajtunk végre (anélkül, hogy más tranzakció konkurensen futna), akkor az adatbázist konzisztens állapotból konzisztens állapotba alakítjuk. A gyakorlatban azonban a tranzakciók általában más tranzakciókkal egyidejűleg konkurensen futnak, emiatt a helyességi elvet közvetlenül nem alkalmazhatjuk. Így olyan „ütemezéseket” kell tekintenünk, amelyek biztosítják, hogy ugyanazt az eredményt állítják elő, mintha a tranzakciókat egyesével hajtottuk volna végre. Az egész fejezet fő témáját adják azok a módszerek, amelyek biztosítják, hogy a tranzakciók csak olyan módon legyenek konkurensen végrehajtva, mintha sorban egyesével futottak volna le.

### 9.1.1. Ütemezések

Az *ütemezés* (schedule) egy vagy több tranzakció által végrehajtott lényeges műveletek időrendben vett sorozata. Amikor a konkurenciavezérlést tanulmányozzuk, a lényeges olvasási és írási műveletek a központi memória puffereiben történnek, nem pedig lemezen. Vagyis egy *A* adatbáziselemet, amelyet valamelyik *T* tranzakció hozott be a pufferbe, ebben a pufferben nemcsak a *T* tudja olvasni vagy írni, hanem más tranzakciók is hozzáférhetnek az *A*-hoz. Idézzük fel a 8.1.4. részből, hogy a READ (OLVASÁS) és a WRITE (ÍRÁS) műveletek először meghívnak egy INPUT utasítást, hogy az adatbáziselemet a lemezről betöltsék, ha még nincs a pufferben, egyébként pedig a READ és WRITE műveletek közvetlenül a pufferben hozzáférnek az elemhez. Ezért csupán a READ és WRITE műveletek és a sorrendjük számít, amikor a konkurenciával foglalkozunk, és az INPUT, illetve OUTPUT műveleteket figyelmen kívül fogjuk hagyni.

**9.1. példa:** Tekintsünk két tranzakciót és az adatbázison való hatásukat, amikor egy meghatározott sorrendben hajtjuk végre a műveleteiket. A *T*1 és *T*2 tranzakciók fő műveletei a 9.2. ábrán találhatók. A *t* és *s* változók a *T*1-nek, illetve *T*2-nek megfelelő helyi változók, és *nem* adatbáziselemek.

|  |  |
| --- | --- |
| T1 | T2 |
| READ(A,t) | READ(A,s) |
| t := t+100 | s := s\*2 |
| WRITE(A,t) | WRITE(A,s) |
| READ(B,t) | READ(B,s) |
| t := t+100 | s := s\*2 |
| WRITE(B,t) | WRITE(B,s) |

**9.2. ábra.** Két tranzakció

Tételezzük fel, hogy az adatbázis-állapoton az egyetlen konzisztencia megszorítás az *A* = *B*. Mivel a *T*1 az *A*-hoz és a *B*-hez is hozzáad 100-at, és a *T*2 az *A*-t és a *B*-t is megszorozza 2-vel, tudjuk, hogy az egyes tranzakciók, egymástól elkülönítve futva megőrzik a konzisztenciát. 

|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | A | B |
|  |  | 125 | 125 |
| READ(A,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(A,t) |  | 125 |  |
| READ(B,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(B,t) |  |  | 125 |
|  | READ(A,s) |  |  |
|  | s := s\*2 |  |  |
|  | WRITE(A,s) | 250 |  |
|  | READ(B,s) |  |  |
|  | s := s\*2 |  |  |
|  | WRITE(B,s) |  | 250 |

**9.3. ábra.** Soros ütemezés, amelyben T1 megelőzi T2-t

### 9.1.2. Soros ütemezések

Azt mondjuk, hogy egy ütemezés *soros* (serial schedule), ha úgy épül fel a tranzakciós műveletekből, hogy először az egyik tranzakció összes műveletét tartalmazza, majd azután egy másik tranzakció összes műveletét stb., miközben nem cseréli fel a műveleteket. Pontosabban kifejezve, egy *S* ütemezés soros, ha bármely két *T* és *T* tranzakcióra, ha *T*-nek van olyan művelete, amely megelőzi a *T* valamelyik műveletét, akkor a *T* összes művelete megelőzi a *T* valamennyi műveletét.

**9.2. példa:** A 9.2. ábrán szereplő tranzakcióknak két soros ütemezése van, az egyikben *T*1 megelőzi *T*2-t, a másikban *T*2 előzi meg *T*1-et. A 9.3. ábra azt az eseménysorozatot mutatja, amikor *T*1 megelőzi *T*2-t, és a kezdeti állapot *A* = *B* = 25. Azt a megállapodást követjük, hogy időrendi sorrendben függőlegesen lefelé írunk. Továbbá a megjelenített *A* és *B* értékek a központi memória pufferbeli értékeire utalnak, nem szükségképpen a lemezen tárolt értékeire.

A 9.4. ábrán látjuk a másik soros ütemezést, amelyben *T*2 megelőzi *T*1-et. A kezdeti állapot legyen megint *A* = *B* = 25. Megjegyezzük, hogy *A* és *B* végső értéke különböző a két ütemezésben, mégpedig mindkettő értéke 250, ha a *T*1 fut először, és 150, ha a *T*2 fut előbb. De nem is a végeredmény a központi kérdés addig, amíg a konzisztenciát megőrizzük. Általában nem várjuk el, hogy az adatbázis végső állapota független legyen a tranzakciók sorrendjétől. 

|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | A | B |
|  |  | 125 | 125 |
|  | READ(A,s) |  |  |
|  | s := s\*2 |  |  |
|  | WRITE(A,s) | 150 |  |
|  | READ(B,s) |  |  |
|  | s := s\*2 |  |  |
|  | WRITE(B,s) |  | 150 |
| READ(A,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(A,t) |  | 150 |  |
| READ(B,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(B,t) |  |  | 150 |

**9.4. ábra.** Soros ütemezés, amelyben T2 megelőzi T1-t

A soros ütemezést úgy ábrázolhatjuk, mint ahogyan a 9.3. ábrán vagy a 9.4. ábrán látható, a műveleteket az előfordulásuk sorrendjében soroljuk fel. Másrészt, mivel a soros ütemezésben a műveletek sorrendje csak magától a tranzakciók sorrendjétől függ, ezért a soros ütemezést néha a tranzakciók felsorolásával fogjuk megadni. Így a 9.3. ábra ütemezését (*T*1, *T*2) reprezentálja, a 9.4. ábráét pedig (*T*2, *T*1).

### 9.1.3. Sorba rendezhető ütemezések

A tranzakciókra vonatkozó helyességi elv szerint minden soros ütemezés megőrzi az adatbázis-állapot konzisztenciáját. Vajon van-e más ütemezés is, amely szintén biztosítja a konzisztencia megmaradását? Igen, ilyen létezik, ahogyan ezt a következő példa mutatja. Általában azt mondjuk, hogy egy ütemezés *sorba rendezhető* (serializable schedule), ha ugyanolyan hatással van az adatbázis állapotára, mint valamelyik soros ütemezés, függetlenül attól, hogy mi volt az adatbázis kezdeti állapota.

**9.3. példa:** A 9.5. ábrán látjuk a 9.1. példában szereplő két tranzakciónak egy sorba rendezhető, ám nem soros ütemezését. Ebben az ütemezésben *T*2 azután van hatással az *A*-ra, miután a *T*1 volt, de mielőtt a *T*1 hatással lenne a *B*-re. Mégis azt látjuk, hogy ebben az ütemezésben a két tranzakció hatása megegyezik a 9.3. ábrán látható (*T*1, *T*2) soros ütemezés hatásával. Ahhoz, hogy meggyőződjünk az állítás igazságáról, nemcsak azt az esetet kell megnéznünk, amely a 9.5. ábrán látható, amikor az adatbázis-állapot *A* = *B* = 25-ről indul, hanem bármely konzisztens adatbázis kiindulási állapotból kiindulva. Mivel minden konzisztens adatbázis-állapotban az *A* = *B* = *c* valamely *c* konstanssal, nem nehéz levezetnünk, hogy a 9.5. ábra ütemezésében az *A*-nak is és a *B*-nek is 2(*c* + 100) lesz az értéke, és így bármelyik konzisztens állapotból indulunk ki, a konzisztenciát megőrizzük.

|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | A | B |
|  |  | 125 | 125 |
| READ(A,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(A,t) |  | 125 |  |
|  | READ(A,s) |  |  |
|  | s := s\*2 |  |  |
|  | WRITE(A,s) | 250 |  |
| READ(B,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(B,t) |  |  | 125 |
|  | READ(B,s) |  |  |
|  | s := s\*2 |  |  |
|  | WRITE(B,s) |  | 250 |

**9.5. ábra.** Sorba rendezhető, de nem soros ütemezés

|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | A | B |
|  |  | 125 | 125 |
| READ(A,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(A,t) |  | 125 |  |
|  | READ(A,s) |  |  |
|  | s := s\*2 |  |  |
|  | WRITE(A,s) | 250 |  |
|  | READ(B,s) |  |  |
|  | s := s\*2 |  |  |
|  | WRITE(B,s) |  | 150 |
| READ(B,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(B,t) |  |  | 150 |

**9.6. ábra.** Nem sorba rendezhető ütemezés

Másrészt tekintsük a 9.6. ábrán található ütemezést. Világos, hogy ez nem soros, de ami lényegesebb, nem is sorba rendezhető. Meggyőződhetünk arról, hogy nem sorba rendezhető, ugyanis legyen a kiindulási konzisztens állapotban *A* = *B* = 25, és az adatbázis inkonzisztens állapotba kerül, amikor *A* = 250 és *B* = 150 lesz. Megjegyezzük, hogy ebben a műveleti sorrendben, a *T*1 dolgozik előbb az *A*-val, viszont *T*2 dolgozik előbb a *B*-vel, ennek hatásaként másképpen kell kiszámolnunk *A*-t és *B*-t, vagyis *A* := 2(*A* + 100), szemben *B* := 2*B* + 100-zal. A 9.6. ábrán található ütemezés olyan viselkedést mutat, amelyet a konkurenciavezérlési működésekkel el kell kerülnünk. 

### 9.1.4. A tranzakció szemantikájának hatása

A sorbarendezhetőségi vizsgálatainkban eddig a tranzakciók által végrehajtott műveleteket néztük meg részletesen annak érdekében, hogy meghatározzuk sorbarendezhető-e az ütemezés.Azonban a tranzakciók részletei is számítanak, ahogyan ezt a következő példából láthatjuk.

**9.4. példa:** Tekintsük a 9.7. ábrán látható ütemezést, amely csak a *T*2 által végrehajtott számításokban különbözik a 9.6. ábrától, mégpedig abban, hogy a *T*2 nem 2-vel szorozza meg *A*-t és *B*-t, hanem 1-gyel.[[1]](#footnote-1) Ekkor *A* és *B* értéke az ütemezés végén megegyezik, és könnyen ellenőrizhetjük, hogy a konzisztens kezdeti állapottól függetlenül a végállapot is konzisztens lesz. Valójában az egyetlen végállapot az, amelyet vagy a (*T*1, *T*2) vagy a (*T*2, *T*1) soros ütemezés eredményez. 

|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | A | B |
|  |  | 125 | 125 |
| READ(A,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(A,t) |  | 125 |  |
|  | READ(A,s) |  |  |
|  | s := s\*1 |  |  |
|  | WRITE(A,s) | 125 |  |
|  | READ(B,s) |  |  |
|  | s := s\*1 |  |  |
|  | WRITE(B,s) |  | 125 |
| READ(B,t) |  |  |  |
| t := t+100 |  |  |  |
| WRITE(B,t) |  |  | 125 |

**9.7. ábra.** Egy olyan ütemezés, amely csak a tranzakciók  
részletezett viselkedése miatt sorba rendezhető

Sajnos, az ütemező számára nem reális a tranzakciós számítások részleteinek figyelembevétele. Mivel a tranzakciók gyakran tartalmaznak általános célú programozási nyelven írt kódokat éppúgy, mint SQL vagy más magas szintű nyelv utasításait, néha nagyon nehéz megválaszolni azokat a kérdéseket, mint pl. „ez a tranzakció az *A*-t egy 1-től különböző konstanssal szorozta-e meg?”. Az ütemezőnek azonban látnia kell a tranzakciók olvasási és írási kéréseit, így tudhatja, hogy az egyes tranzakciók mely adatbáziselemeket olvasták be, és mely elemek *változhattak* meg. Az ütemező feladatának az egyszerűsítésére megszokott az a feltételezés, hogy:

• Bármely *A* adatbáziselemnek egy *T* tranzakció olyan értéket ír be, amely az adatbázis-állapottól függ oly

módon, hogy ne forduljon elő aritmetikai egybeesés.

Más szóval kifejezve, ha a *T* tudna az *A*-ra olyan hatással lenni, hogy az adatbázis-állapot inkonzisztenssé váljék, akkor a *T* ezt meg is teszi. Ezt a feltevést a 9.2. részben pontosítjuk, amikor a sorbarendezhetőség biztosítására adunk meg elégséges feltételeket.

### 9.1.5. A tranzakciók és ütemezések jelölése

Ha elfogadjuk, hogy egy tranzakció által végrehajtott pontos számítások tetszőlegesek lehetnek, akkor nem szükséges a helyi számítási lépések részleteit néznünk, mint amilyen a t := t+100. Csak a tranzakciók által végrehajtott olvasások és írások számítanak. Így a tranzakciókat és az ütemezéseket rövidebben jelölhetjük. Ekkor *rT*(*X*) és *wT*(*X*) tranzakció műveletek, és azt jelentik, hogy a *T* tranzakció olvassa (*r*, az angol *read* = olvasás rövidítése), illetve írja (*w*, az angol *write* = írás rövidítése) az *X* adatbáziselemet. Továbbá, mivel a tranzakcióinkat *T*1, *T*2, …-vel fogjuk általában jelölni, így megállapodunk abban, hogy *ri*(*X*) és *wi*(*X*) ugyanazt jelöli, mint *rTi*(*X*), illetve *wTi*(*X*).

**9.5. példa:** A 9.2. ábrán látható tranzakciók az alábbi módon írhatók fel:

*T*1: *r*1(*A*); *w*1(*A*); *r*1(*B*); *w*1(*B*);

*T*2: *r*2(*A*); *w*2(*A*); *r*2(*B*); *w*2(*B*);

Megjegyezzük, hogy nem említettük sehol a *t* és az *s* helyi változókat, és nem jelöltük azt sem, hogy mi történt a beolvasás után az *A*-val és *B*-vel. Intuíció alapján ezt úgy értelmezzük, hogy az adatbáziselemek megváltozásában a „legrosszabbat fogjuk feltételezni”.

Egy másik példaként nézzük meg a *T*1 és *T*2-nek a 9.5. ábrán látható sorba rendezhető ütemezését. Ezt az ütemezést átírva:

*r*1(*A*); *w*1(*A*); *r*2(*A*); *w*2(*A*); *r*1(*B*); *w*1(*B*); *r*2(*B*); *w*2(*B*);



Pontosítva a jelölést:

1. Egy *tranzakció műveletét* (action of transaction) *ri*(*X*) vagy *wi*(*X*) formában fejezünk ki, amely azt jelenti, hogy a *Ti* tranzakció olvassa *(read),* illetve írja *(write)* az *X* adatbáziselemet.

2. Egy *Ti* *tranzakció* az *i* indexű műveletekből álló sorozat.

3. A T tranzakciók halmazának egy *S* *ütemezése* olyan műveletek sorozata, amelyben minden T halmazbeli *Ti* tranzakcióra teljesül, hogy *Ti* műveletei ugyanabban a sorrendben fordulnak elő az *S*-ben, mint ahogy magában a *Ti* definíciójában szerepeltek. Azt mondjuk, hogy az *S* az őt alkotó tranzakciók műveleteinek *átlapolása* (interleaving).

Például a 9.5. példában található ütemezésben az összes 1-es indexű művelet ugyanabban a sorrendben szerepel, mint ahogy a *T*1 definíciójában volt, és az összes 2-es indexű művelet ugyanabban a sorrendben fordul elő, mint ahogy a *T*2 definíciójában szerepelt.

1. Valaki jogosan kérdezheti, hogy miért is viselkedik így egy tranzakció? Mégis a példa kedvéért ezt most hagyjuk figyelmen kívül. Valójában több elfogadható tranzakciót is helyettesíthetnénk a *T*2 helyére, amely az *A*-t és *B*-t változatlanul hagyná. Például amikor a *T*2 csak egyszerűen beolvassa az *A*-t és *B*-t, és kiíratja az értéküket. Vagy *T*2 a felhasználótól kérhet be adatokat, hogy kiszámoljon egy *F* tényezőt, amivel megszorozza az *A*-t és a *B*-t, és előfordulhat olyan felhasználói input, amelyre az *F* = 1. [↑](#footnote-ref-1)