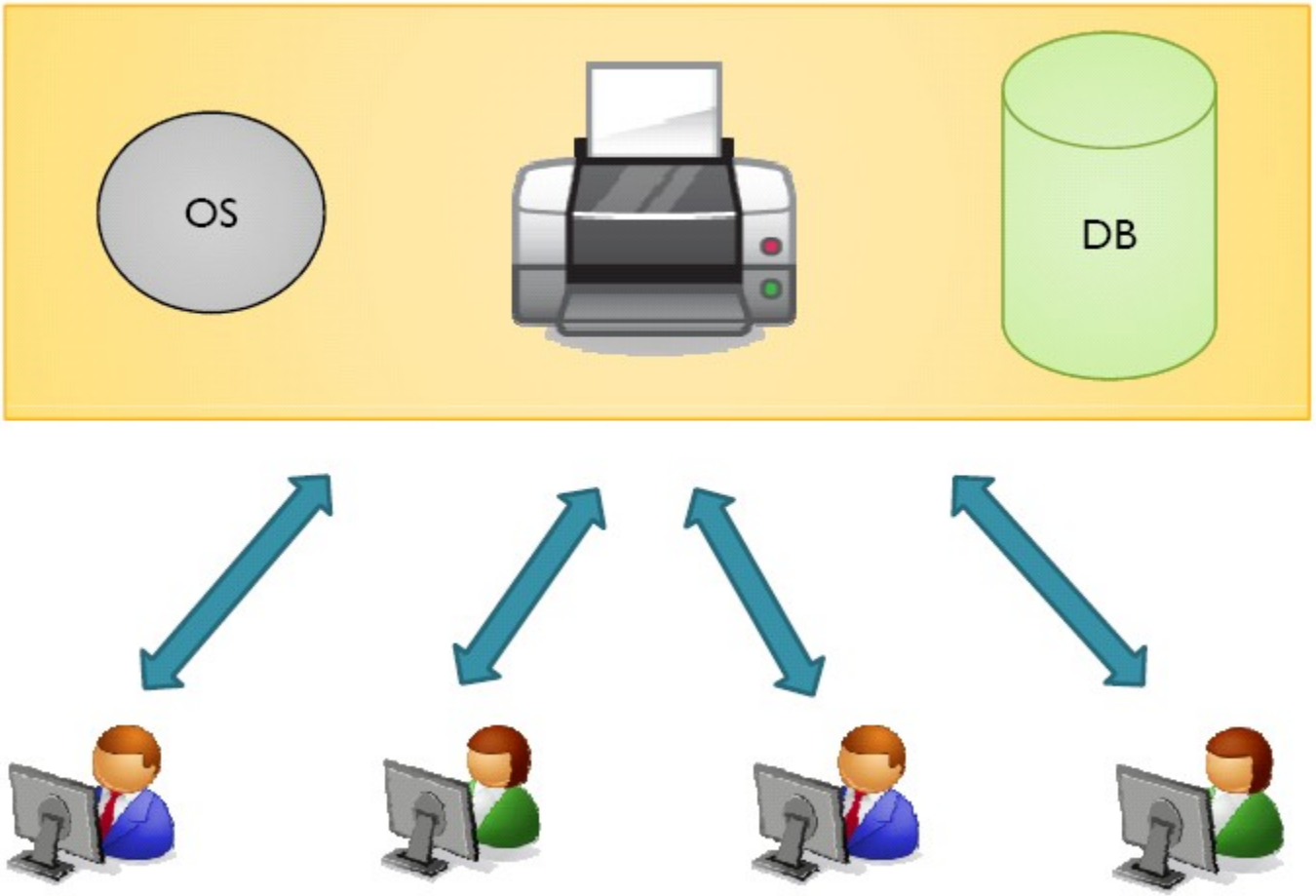
Adatbázisok II.

2-3

Jánosi-Rancz Katalin Tünde
tsuto@ms.sapientia.ro
327A

Tranzakciókezelés

Osztott erőforrások



konfliktus helyzetek (azonos erőforrás igény)

Tranzakciókezelés

- Eddig feltételeztük:
  - egy felhasználó van csak
  - a lekérdezések/módosítások hiba nélkül lefutnak
- A valóságban a tranzakciókezelő dolga:
  - párhuzamosság, többfelhasználós működés(pl. banki rendszerek, helyfoglalás)
  - rendszerhibák utáni helyreállítás (az adatbázist inkonzisztens állapotból konzisztens állapotba kell hozni)
- Megoldások:
  - várakoztatás ->várakozó sorok (nyomtató)
  - adatbáziskezelésben egyedi vonások
- Cél: párhuzamos hozzáférés biztosítása, de úgy, hogy a konzisztencia megmaradjon

Megoldandó problémák

- **Többfelhasználós működés**

1. felhasználó: $u1$; $u2$; : : : ; $u10$
2. felhasználó: $v1$; $v2$; : : : ; $v103$

Ez a két utasítássorozat nem elkülönülve jön, hanem összefésülődnek:

$u1$; $v1$; $v2$; $u2$; $u3$; $v3$; : : : ; $v103$; $u10$

Ebből viszont problémák származhatnak:

1. felhasználó: READ A, A + +, WRITE A
2. felhasználó: READ A, A + +, WRITE A

Ha ezek úgy fésül ődnek össze, hogy:

(READ A)1, (READ A)2, (A + +)1,(A + +)2, (WRITE A)1, (WRITE A)2

akkor a végén csak egyel nő A értéke, holott kettővel kellett volna.

Megoldandó problémák

- **Rendszerhibák**

Példa: átutalunk egyik helyről a másik helyre pénzt:

$$A := A - 50 \qquad B := B + 50$$

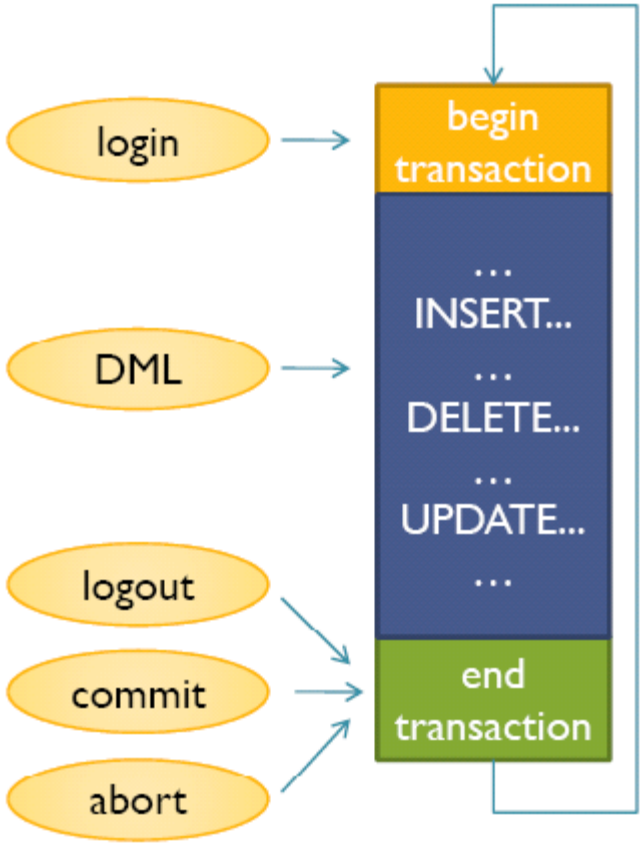
Ha az a közepén meghal: hibás állapot jön létre.

Tranzakció

- A tranzakció egy felhasználóhoz tartozó elemi utasítások olyan sorozata, melynek fő jellemzői az (ACID):
  - ★ Atomiság, Atomicity: vagy az összes utasításnak végre kell hajtódnia vagy egynek sem szabad
  - ★ Konzisztencia, Consistency: *konzisztens állapotból konzisztens állapotba* vigyen
  - ★ Elkülönítés, Isolation: több tranzakció egyidejű futása után úgy kell kinéznie az adatbázisnak, mintha a tranzakciók nem lettek volna összefésülve (az ütemező dolga lesz ennek biztosítása)
  - ★ Tartósság, Durability: a befejezett tranzakciók hatása nem veszhet el, *lezárt tranzakciók eredményeit az* adatbáziskezelő mindenáron őrizze meg

Tranzakciókezelés

- az adatbáziskezelés központi fogalma
- minden művelet tranzakcióba szervezetten hajtódik végre
- a DBMS nem enged tranzakción kívüli művelet végrehajtást



Tranzakciók leírása

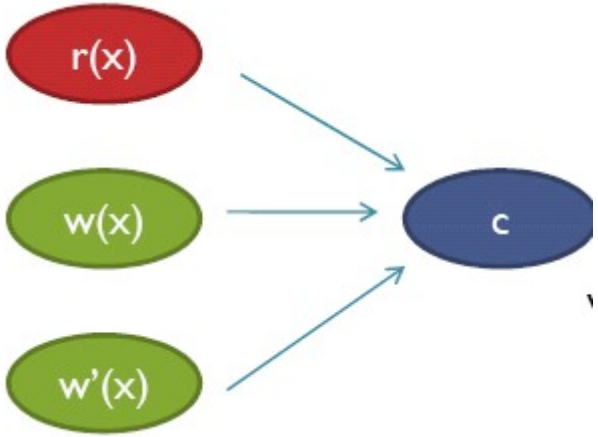
- műveletsort kell megadni
- elemei:
  - ★ műveletek:
    - ❖ olvasás: **r**, írás: **w**
    - ❖ objektum, amelyre hatnak paraméterként: **r(x), w(x)**
  - ★ tranzakció lezárása
    - ❖ véglegesítés **(commit): c**
    - ❖ visszagörgetés **(abort): a**
  - ★ sorrendiség: ->
    - ❖ megelőzési reláció: **p1->p2**

Tranzakciók ábrázolása

- ábrázolás gráffal:
- csomópontok a műveletek, irányított élei a megelőzési relációk

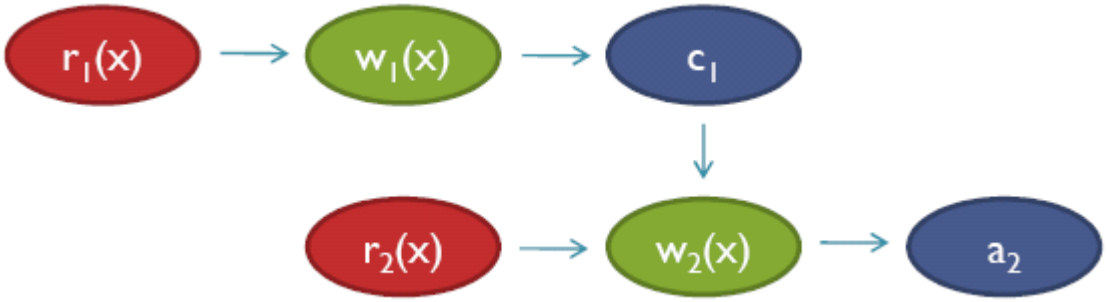
- kötöttségek érvényes tranzakció esetén:
  - ★ tartalmaznia kell egy lezárási műveletet (c,a)
  - ★ pontosan csak az egyiküket tartalmazhatja
  - ★ a lezárási művelet az utolsó, az összes többi megelőzi

- egyértelmű végrehajthatósági feltétel: a műveletsor eredménye azonos kezdőfeltételek mellet mindig egyezzen meg a leírt tranzakció eredményével



History fogalma

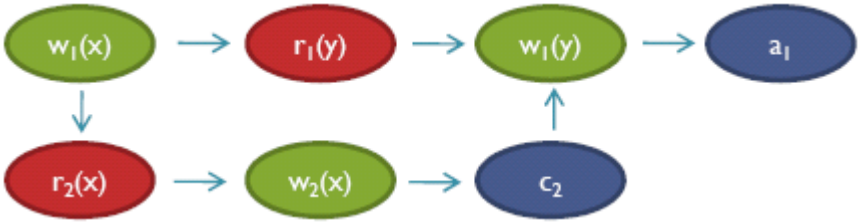
- a rendszerben futó tranzakciók összessége
- megadása hasonló a tranzakció megadásához
- műveletekből áll
- lényeges a végrehajtási sorrend
- ábrázolása: gráffal
- a tranzakció azonosítóját is feltüntetjük, mint művelet indexet



Konkurenciavezérlés

- tranzakció, mely nem véglegesített adatokat olvas „piszkos adat olvasása”(uncommited dependency)
- az elveszett módosítás (lost update) problémája
- helytelen analízis (inconsistent analysis)

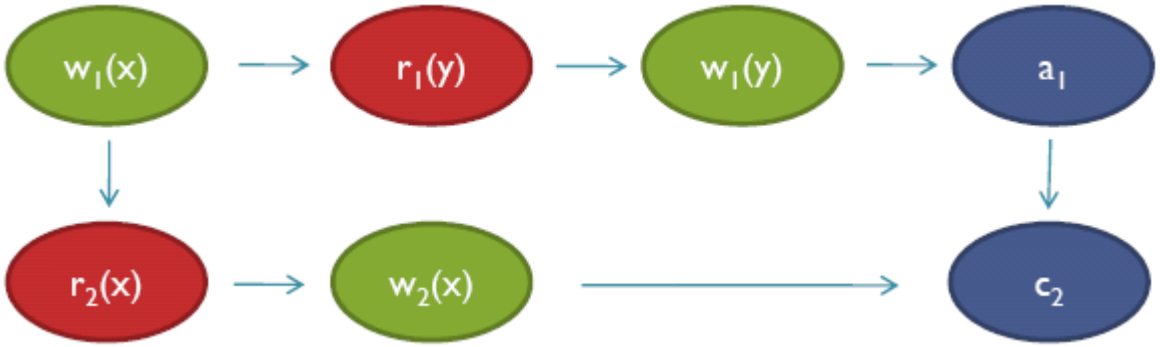
Piszkos adat olvasása



Mi a probléma?

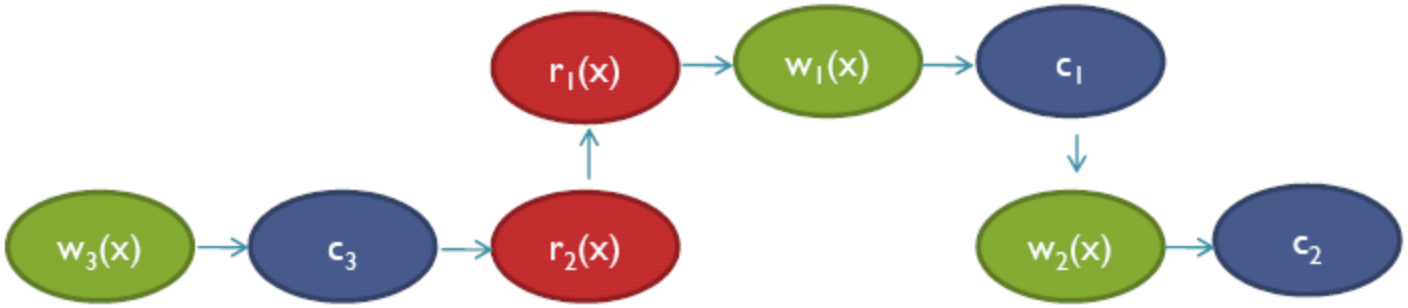
- T1 kiír egy értéket x-be
- T2 olvassa ezt, majd ez alapján új értéket ír x-be
- T2 véglegesíti az eredményt
- a T1 tovább fut, olvassa y-t, majd módosítja
- T1 abortálódik
  - ★ w1(x) utasítás-t hogyan görgetjük vissza?
  - ★ T2 nem véglegesített adatokat használt fel, ezért T2 tranzakciót is vissza kell görgetni
- tartósság elvét kellene felrúgni
- konzisztencia vs. ACID-elvek

Megoldás: RecoverAble(RA) history



- Egy history visszagörgethető, ha minden tranzakció később zárul le minden olyan tranzakciónál, amiből ő olvasott

Az elveszett módosítás (lost update) problémája



T1: normál fizetés tranzakció
T2: rendkívüli jutalmak tranzakció

lost update

T1 beli olvasás megelőzi T2 beli írást, T1 beli írás követi T2 beli olvasást

SeRializable history

- **soros ütemezés**: olyan ütemezés, amikor a különböző tranzakciók utasításai nem keverednek, először lefut az egyik összes utasítása, aztán a másiké, aztán a harmadiké, stb.

- **Cél:** olyan sorrend (összefésülődés) kikényszerítése, ami sorosítható ütemezés
- **Módszer:** az ütemező (az adatbáziskezelő része) felelős azért, hogy csak ilyen sorrendek legyenek. Figyeli a tranzakciók műveleteit és késleltet/ABORT-ál tranzakciókat. (részletezzük)

Példa sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		x+100	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			y+100
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot (y + 100)$

A táblázat baloldali részén azt jelezzük, hogy milyen műveleteket végeznek a tranzakciók, a jobboldalon pedig az látszik, hogy eközben mi történik az ***A*** és ***B adategységekkel. Ezek*** kezdeti értékei ***x és y.***

Read(A,t)= olvassuk be ***A*** értékét a ***t változóba***
Write(A,t)= írjuk ki a ***t változó*** értékét ***A-ba***

Látható, hogy ez nem egy soros ütemezés, mert össze vannak fésülődve a két tranzakció utasításai.

Viszont sorosítható, mert a hatása ***A-n és B-n is azonos a T1T2 soros ütemezés hatásával, (x; y)-ból (2 (x + 100); 2 (y + 100)) lesz.***

Példa nem sorosíthatóra

T_1	T_2	A	B
Read(A,t)		x	y
$t := t + 100$			
Write(A,t)		x+100	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot y$
Read(B,t)			
$t := t + 100$			
Write(B,t)			$2 \cdot y + 100$

- Ez nem egy sorosítható ütemezés, mert se a ***T1T2 soros ütemezés, se a T2T1 soros ütemezés*** hatása nem az, hogy ***(x; y)-ból (2 (x + 100); 2 y + 100) lesz.***
- A ***T1T2 ütemezés (2 (x + 100); 2 (y + 100)) eredményt ad,***
- a ***T2T1 pedig (2 x + 100; 2 y + 100)-t***

Konfliktusok

- Konkfliktust okoz: olyan egymást követő műveletpár az ütemezésben, amelynek ha a sorrendjét felcseréljük, akkor legalább az egyik tranzakció viselkedése megváltozhat.
- Nem cserélhetjük fel a műveletek sorrendjét ha:
- Ugyanannak a tranzakciónak két művelete konfliktus, pl: $r_i(X)$; $w_i(Y)$. Egy tranzación belül a műveletek sorrendje rögzített, az ABKR ezt a sorrendet nem rendezheti át.
- Különböző tranzaciók ugyanarra az adatbáziselemre vonatkozó írása konfliktus. Pl. $w_i(X)$; $w_j(X)$. Ebben a sorrendben X értéke az marad, melyet T_j ír, fordított sorrendben pedig az marad, melyet T_i ír. Ezek az értékek pedig nagy valószínüséggel különbözőek.
- Különböző tranzacióknak ugyanabból az adatbáziselemből való olvasása és írása konfliktus. Tehát $r_i(X)$; $w_j(X)$ konfliktus, mivel ha felcseréljük a sorrendet, akkor a T_i által olvasott X-beli érték az lesz, melyet a T_j ír és az nagy valószínűséggel nem egyezik meg az X korábbi értékével. Hasonlóan a $w_i(X)$; $r_j(X)$ is konfliktus.

Az ütemező eszközei a sorosíthatóság elérésére

- Az ütemezőnek több lehetősége is van arra, hogy kikényszerítse a sorosítható ütemezéseket:

- **zárak** (ezen belül is még: protokoll elemek, pl. 2PL)
- **időbélyegek** (time stamp)
- **érvényesítés**

Zárolási módszerek

- a tranzakció használat idejére lefoglalja az objektumot
- a többi tranzakció korlátozva van az objektum elérésében várakozás
- várakozás feloldás:
  - elérhető már az objektum
  - kiderül, hogy nem érdemes várni
- nyilván kell tartani objektumonként kiegészítő információkat:
  - szabad-e,
  - ki foglalja (felszabadításnál tudni kell)
- helytöbblettel jár a zárolás

Jelölések

- ★ $r_i(X)$ - a T_i tranzakció olvassa az X adatbáziselemet (r - read)
- ★ $w_i(X)$ - a T_i tranzakció írja az X adatbáziselemet (w -write)
- ★ $sl_i(X)$ - a T_i tranzakció osztott zárat kér az X-re
- ★ $xl_i(X)$ – a T_i tranzakció kizárólagos zárat kér az X-re
- ★ $u_i(X)$ - a T_i tranzakció feloldja (unlock) X adatbáziselemen tartott zárát.

Azért, hogy a tranzakciók konzisztenciája megmaradjon, minden T_i tranzakció esetén bevezetjük a következő követelményeket:

- Az $r_i(X)$ olvasási műveletet meg kell előzze egy $sl_i(X)$ vagy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$

- A $w_i(X)$ olvasási műveletet meg kell előzze egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$

- Ha $xl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ vagy $sl_j(X)$ valamely i-től különböző j-re anélkül, hogy közben ne szerepelne $u_i(X)$

❖ Ha $sl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ i-től különböző j-re anélkül, hogy közben ne szerepelne $u_i(X)$

❖ **upgrade lock to exclusive:** Ha egy tranzakciónak már van osztott zára egy adatbáziselemen, melyet módosítani akar, felminősítheti a zárat kizárólagossá

Zárolási szintek

- Mező szintű zárolás
- Rekord szintű zárolás
- Tábla szintű zárolás

Előnyök és hátrányok

	Mező szintű zárolás	Tábla szintű zárolás
Előny	• nagyobb fokú párhuzamosság • jelentősebb konkurencia	• sokkal egyszerűbb nyilvántartani • gyorsabban adminisztrálható
Hátrány	• sokkal több kiegészítő információt kell tárolni és karbantartani • jelentős hely, idő költségnövekedés	• nagyon lecsökkenti a konkurenciát • nagyon sokáig kell várakozni a párhuzamosan futó tranzakcióknak egymásra

Zárolási Alapelv:
csak addig zároljunk egy objektumot, ameddig szükséges
lefoglalás: amikor szükség van rá
felengedés: adatok véglegesítése után, tranzakció végén
hiba: a művelet után azonnal felengednénk
T2 olvas T1 által írt, nem véglegesített objektumot
RA, ACA feltétel sérülne

Kétfázisú zárolás (2PL)

- 1. fázis: zárolások lefoglalása
- 2. fázis: zárolások felengedése a tranzakció végén

- a legtöbb RDBMS-ben 2PL zárolás biztosítja az ACID-elveknek megfelelő history-k megvalósítását
- Tehát egy tranzakció, mely betartja a szigorú két-fázisú lezárási protokollt a következő sorrendben hajtja végre a munkát:
  - ★ minden szükséges zárat a tranzakció elején kér;
  - ★ mikor elérkezik a COMMIT ponthoz beírja az adatbázisba a módosított adatbáziselemeket;
  - ★ majd felszabadítja a zárakat.

Zárolások gyenge pontja

- a zárolások biztos megoldást nyújtanak a helyes history megvalósításra
- gyenge pont:
  - ★ tranzakciók várakozásra kényszerítése
  - ★ többen várakoznak körbevárakozás alakulhat ki végtelen várakozás

Holtpont, olyan állapot, mikor két vagy több tranzakció várási állapotban van, mindenik vár a másik által lezárt objektumra.

A holtpont megoldása

- Módosítási zárak alkalmazása
- Várakozási gráf (Wait-For-Graph)
- Időkorlát mehanizmus (Timeout)

Módosítási zárak

- Az uli(X) módosítási zár (update lock) a Ti tranzakciónak csak X adatbáziselem olvasására ad jogot, az X írására nem, viszont csak a módosítási zárat lehet később felminősíteni, az olvasásit nem.

- Ha egy tranzakciónak szándékában áll módosítani az X adatbáziselemet, akkor módosítási zárat kér rá.

- Módosítási zárat engedélyez a rendszer az X-en akkor is, ha már van osztott zár X-en. Ha már van egy módosítási zár X-en, akkor viszont minden más zárat elutasít a rendszer, legyen az osztott, módosítási vagy kizárólagos.

Osztott(S), kizárólagos (X) és módosítási (U) zárak kompatibilitási mátrixa

	S	X	U
S	Igen	Nem	Igen
X	Nem	Nem	Nem
U	Nem	Nem	Nem

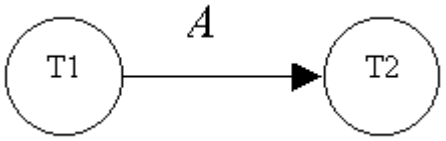
Elveszett módosítás problémája

Tranzakció 1	Idő	Tranzakció 2
$ul_1(P)$; $r_1(P)$	t_1	-
-	|	-
-	t_2	$ul_2(P)$ **vár**
-	|	-
$xl_1(P)$; $w_1(P)$; $u_1(P)$	t_3	-
-	|	-
-	t_4	$ul_2(P)$; $r_2(P)$
	|	$xl_2(P)$; $w_2(P)$; $u_2(P)$

Elveszett módosítás probléma megoldva, 2-es tranzakció késleltetve van, míg 1-es felszabadítja a zárat.

Várakozási gráf

- A gráf csúcsait a tranzakciók alkotják. (Legyen T1 tranzakció az egyik csúcsban és T2 tranzakció egy másikban.)
- T1 és T2 csúcs között élet rajzolunk, ha:
  - ★ T1 lezárva tartja az A adatbáziselemet.
  - ★ T2 kéri az A adatbáziselemet, hogy zárolhassa azt.
- Az él irányítása a T1-től a T2 felé lesz és A az él címkéje.



Várakozási gráf

- Ha a gráf tartalmaz **ciklust**, azt jelenti, hogy van holtpont.
- Ebben az esetben valamelyik tranzakciót, mely a körben szerepel meg kell szakítani és visszapörgetni. A rendszer ki kell derítse, hogy holtpont probléma áll fenn.
- A tranzakciót megszakítani azt jelenti, hogy valamelyik tranzakciót kiválasztja, mely a holtpontban szerepel – ez az áldozat (victim) — és visszapörgetni, így felszabadul a lock, amit ez a tranzakció adott ki, és lehetősége nyílik a többi tranzakciónak, hogy folytatódjon.

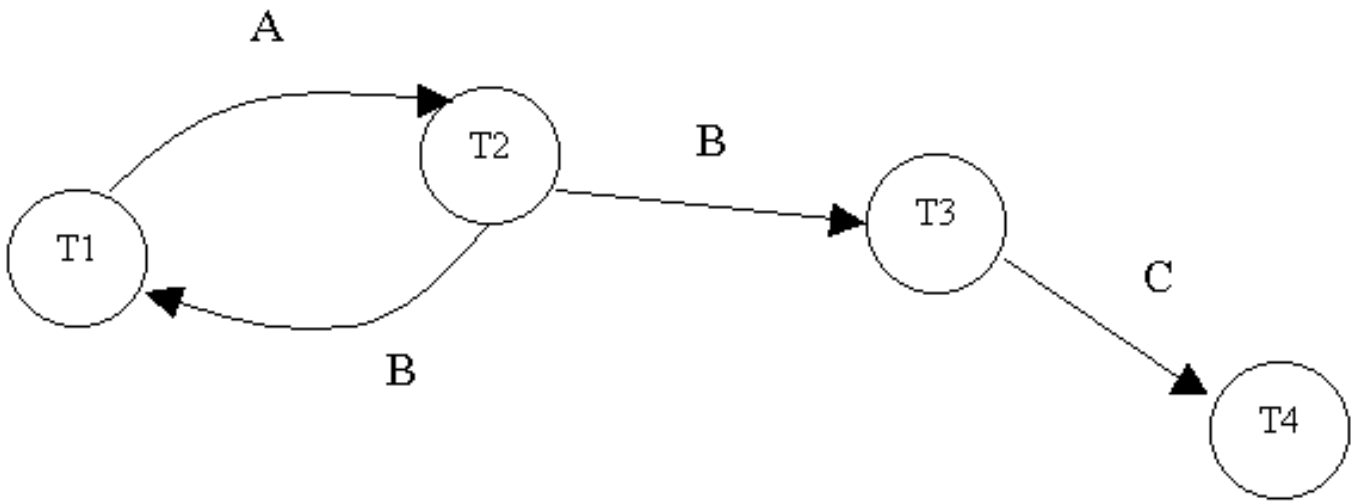
Időkorlát mehanizmus (Timeout - TM)

- a rendszer figyeli, mennyi ideig várakozott a tranzakció a zárolás feloldására
- ha a várakozási idő túllép egy megadott időhatárt, akkor a TM deadlock jelenségként értékeli, melyet fel kell oldani
- egyik megoldás:
  - valamelyik tranzakció abortálása
  - abortálandó tranzakció kiválasztására számos stratégia létezhet

Példa

Idő	T_1	T_2	T_3	T_4
t_1	$xl_1(A)$; $w_1(A)$;			
t_2		$xl_2(B)$; $w_2(B)$;		
t_3			$xl_3(C)$; $w_3(C)$;	
t_4				$xl_4(C)$; **vár**
t_5	$xl_1(B)$; **vár**			
t_6		$xl_2(A)$; **vár**		
t_7			$xl_3(B)$; **vár**	

Példa várakozási gráfra



Idő	T_1	T_2	T_3	T_4
t_1	$xl_1(A)$; $w_1(A)$;			
t_2		$xl_2(B)$; $w_2(B)$;		
t_3			$xl_3(C)$; $w_3(C)$;	
t_4				$xl_4(C)$; **vár**
t_5	$xl_1(B)$; **vár**			
t_6		$xl_2(A)$; **vár**		
t_7			$xl_3(B)$; **vár**	
t_8	ABORT T_1 $u_1(A)$			
t_9		$xl_2(A)$; $w_2(A)$; $u_2(B)$; $u_2(A)$		
t_{10}			$xl_3(B)$; $w_3(B)$; $u_3(C)$; $u_3(B)$	
t_{11}				$xl_4(C)$; $w_4(C)$; $u_4(C)$;

Más sorosítható módszer: időbélyegek (Timestamp ordering - TO)

- Minden tranzakcióhoz rendel egy időbélyeget, mely jelzi hogy mikor jött létre a tranzakció a többihez viszonyítva
  - időbélyeg egy sorszám
  - korábbi tranzakciók időbélyege kisebb, későbbi tranzakcióké nagyobb
- A különböző tranzakciókhoz tartozó, egymással konfliktusban álló műveletek esetén a műveletek végrehajtási sorrendje feleljen meg a tranzakciók időbélyeg sorrendjének.
- Vagyis azon műveleteknek kell előbb végrehajtódniuk, amelyek időbélyege kisebb.

- Amennyiben egy tranzakció olyan objektumot akar módosítani vagy olvasni, melyet egy fiatalabb tranzakció egyszer már módosított
  - a műveletet a TM megakadályozza
  - abortálja az idősebb tranzakciót
  - abortálás után újraindítja a tranzakciót nagyobb időbélyeggel
  - újra próbálkozhat

Tranzakciók az SQL-ben

- A gyakorlatban általában nem lehet megkövetelni, hogy a műveletek egymás után legyenek végrehajtva, mivel túl sok van belőlük, ki kell használni a párhuzamosság által nyújtott lehetőségeket.

- Az ABKR-ek biztosítják a sorbarendezhetőséget, a felhasználó úgy látja, mintha a műveletek végrehajtása sorban történt volna, valójában nem sorban történik.

Csak olvasó tranzakciók

- Ha a tranzakció csak olvas és nem módosítja az adatbázis tartamát, közölhetjük ezt az ABKR-el és akkor optimálisabb ütemezést valósít meg. Az utasítás amivel ezt közölhetjük az ABKR-el:

```
SET TRANSACTION READ ONLY;
```

SQL utasítások

- SET TRANSACTION ISOLATION LEVEL szint;
- NOLOCK
  - nem foglalja le a tranzakció által érintett objektumokat
- READ UNCOMMITED
  - piszkos, véglegesítés előtti adatokat is olvashatnak
  - átlapolt írás nem megengedett
- READ COMMITED
  - csak véglegesített, tiszta adatok olvashatók
- REPEATABLE READ
  - teljesül az ismételhető olvasás
  - két olvasás között bővülhet a tábla
- SERIALIZABLE
  - a teljesen soros végrehajtást kérvényezi
  - nincs elméleti izolációs szint
  - minden nemű módosítás tiltott

Az elkülönítés szintjei SQL-ben

ISOLATION LEVEL (Elkülönítési szint)	Dirty read (Piszkos adat olvasása)	Unrepeteable read (Nem ismételhető olvasás)	Fantom
READ UNCOMMITED (nem olvasásbiztos)	Igen	Igen	Igen
READ COMMITED (olvasásbiztos)	Nem	Igen	Igen
REPETEABLE READ (megismételhető olvasás)	Nem	Nem	Igen
SERIALIZABLE (sorba rendezhető)	Nem	Nem	Nem

Nem ismételhető olvasás

- Nem ismételhető olvasás (**Unrepeteable read**) akkor fordul elő, ha a tranzakciók párhuzamosan futnak, *A* tranzakció beolvas egy adatbáziselemet, időközben *B* tranzakció módosítja ugyanazt az adatbáziselemet, majd *A* tranzakció ismét beolvassa ugyanazt az adatbáziselemet, tehát *A* tranzakció “ugyanazt” az adatbáziselemet olvasta kétszer és két különböző értéket látott.

elmélet - gyakorlat

- Az elmélet szerint a sorba rendezhető elkülönítési szint a biztos, hogy semmilyen konkurencia probléma nem merül fel, vagyis a tranzakció úgy fut le, mintha minden más tranzakció teljes egészében vagy előtte vagy utána ment volna végbe.

- A gyakorlatban viszont megengednek nagyobb interferenciát a tranzakciók között.

Nem ismételhető olvasás

A Tranzakció	Idő	B Tranzakció
$sl_1(P)$; $r_1(P)$; $u_1(P)$	t_1	-
-	|	-
-	t_2	$sl_2(P)$; $r_2(P)$;
-	|	$xl_2(P)$; $w_2(P)$;$u_2(P)$
-	|	-
$sl_1(P)$; $r_1(P)$; $u_1(P)$	t_3	-
-	|	-

Fantomok

- A rendszer csak létező adatbáziselemeket tud zárolni, nem könnyű olyan elemeket zárolni, melye nem léteznek, de később beszúrhatók.

- T1 olvassa azon sorok halmazát, melyek adott feltételnek eleget tesznek, T2 új sort illeszt a táblába, mely kielégíti a feltételt, így a T1 eredménye helytelen.

Fantom

T1:

- SELECT AVG(Atlag)
  FROM Diakok
  WHERE
  CsopKod = ‘531’
- megismétli a kérést
- fantom-ot lát

- T2 beszúr egy új diákot a Diakok táblába

Fantom

- ★ Ez valójában nem konkurencia probléma, ugyanis a (T1, T2) soros sorrend ekvivalens azzal ami történt.

- Tehát van egy fantom sor a Diakok táblában, melyet zárolni kellett volna, de mivel még nem létezett nem lehetett zárolni.

- A megoldás, hogy sorok beszúrását és törlését az egész relációra vonatkozó írásnak kell tekinteni és X zárat kell kérni az egész relációra, ezt nem kaphatja meg, csak ha a minden más zár fel van szabadítva, a példa esetén a T1 befejezése után.

- Egy más megoldás arra, hogy a rendszer megelőzze a fantom megjelenését: le kell zárjnia a hozzáférési utat (Acces Path), mely a feltételnek eleget tevő adathoz vezet

Mit tehet a felhasználó?

- Az izolálási szintet a tranzakciók esetén a felhasználó beállíthatja a:
  - SET TRANZACTION ISOLATION LEVEL TO <elkülönítési szint>
- a felhasználó nem ad semmi zárolást, rábízza a rendszerre.

Elkülönítési szintek közötti relációk

- a > erősebb feltételt jelent, akkor az elkülönítés szintjei között fennállnak a következő relációk:

SERIALIZABLE > REPEATEBLE READ >
> READ COMMITED > READ UNCOMMITED

- Ha minden tranzakciónak SERIALIZABLE az elkülőnítés szintje, akkor több tranzakció párhuzamos végrehajtása esetén a rendszer garantálja, hogy az ütemezés sorbarendezhető.

- Ha egy ennél kisebb elszigeteltségi szinten fut egy tranzakció a sorbarendezhetőség meg van sértve.

SERIALIZABLE

- a tranzakció betartja a *szigorú kétfázisú lezárási protokoll*t, lezárást alkalmaz, írás és olvasás előtt, tranzakció végéig tartja, úgy az objektum halmazon (indexen) is (ne jelenjen meg fantom)

REPEATABLE READ

- serializable –től abban különbözik, hogy indexet nem zárol – csak egyedi objektumokat, nem objektum halmazokat is.
- Olvasás előtt SLOCK, írás előtt XLOCK, tranzakció végéig tartja.

READ COMMITED

- – implicit – nem enged meg az adatbázisból visszatéríteni olyan adatot, mely nincs véglegesítve (uncommited) (dirty read nem fordulhat elő).

- Share lock –t kér olvasás előtt a tranzakcióban szereplő objektumokra, utána rögtön felengedi, XLOCK írás előtt, tartja tranzakció végéig.

READ UNCOMMITED

- share lock –t nem kér olvasás előtt, sem XLOCK –t írás esetén
  - elolvashatja egy futó tranzakció által végzett változtatást, mely még nem volt véglegesítve
  - ha valaki más közben kitörli az adatot, melyet olvasott, hibát sem jelez, vagy törli az egész táblát
  - nem ajánlott egyetlen applikációnak sem
    - esetleg, olyan statisztikai kimutatások esetén, ahol egy-két változtatás nem lényeges.

Oracle tranzakció-kezelése

- alapvetően zároláson alapszik, de megtalálhatóak benne bizonyos TO elemek is

- csak írási zárolás létezik
  - legnagyobb konkurencia biztosítása végett
  - olvasási művelet sohasem gátol más műveletet
  - olvasást bármikor végre lehet hajtani
    - soha nem várakozik más múveletre
    - más műveletek sem várakoznak miatta

- az adatbázisban konzisztens adatok vannak
  - csak a lezárt tranzakciók eredményeit tárolja
  - olvasási műveletek mindig konzisztens képet adnak
  - a módosítási utasítások egy közös munkaterületen kerülnek bejegyzésre, véglegesítéskor kerülnek bele a konzisztens adatbázisba

Oracle tranzakció-kezelése

- a közös munkaterületen minden objektum csak egy példányban foglal helyet
  - egyidejűleg csak egy tranzakció módosíthat egy objektumot

- ST elvek betartása:
  - addig nem enged egy objektumot módosítani, amíg az őt utoljára módosító tranzakció le nem záródik
  - a TM a közös munkaterületen zárolja az objektumokat

Oracle zárolás

- 2PL zárolás
- nincs olvasási zárolás
  - konkurencia megnövekedése
  - kisebb mértékű biztonság
    - lost update jelenség fenáll
- hosszú tranzakciónál nagy valószínűséggel megváltozik a konzisztens adatbázis képe
  - más tranzakciók sikeresen lezáródtak
  - hatásai láthatóvá váltak a többiek számára
- bizonyos esetekben előnyös lenne a tranzakción belüli állandó tartalom olvasása
  - tranzakció szintű olvasási-konzisztencia
  - külön igényelni kell
- létezik művelet szintű olvasási-konzisztencia (alapért.)

Oracle zárolás

- SET TRANSACTION READ ONLY
  - új tranzakció kezdődik
  - tranzakció szintű olvasási-konzisztencia
  - csak olvasási műveleteket enged

- nagyobb erőforrás ráfordítás
  - SCN mechanizmus

SCN mechanizmus

- olvasási konzisztenciát igénylő végrehajtási egységhez feljegyzésre kerül
  - mikor indult el
  - a konzisztens képbe kerülő elemek, mikor kerületek a konzisztens DB-be
- ezen időpontjelző az SCN (system change number)
  - időbélyeg jellegű szerepe van
- egy hosszabb tranzakció alatt több lezárt tranzakció is módosíthatta az érintett blokkot
  - a blokknak több különböző SCN azonosítású példányt is nyílván kell tartani
- lekérdezés során:
  - az indulási SCN értéknek megfelelő SCN adatblokkokat fogja felhasználni

SAVEPOINT

- sikertelenség esetén a tranzakciókat vissza kell görgetni az utolsó konzisztens állapotig
  - hosszú tranzakciók -> nagy veszteség
- tranzakción belül definiálhatunk több savepoint-ot is:
  - SAVEPOINT azonosító
- a tranzakció a savepoint-ig is visszagörgethető
  - ROLLBACK TO SAVEPOINT azonosító

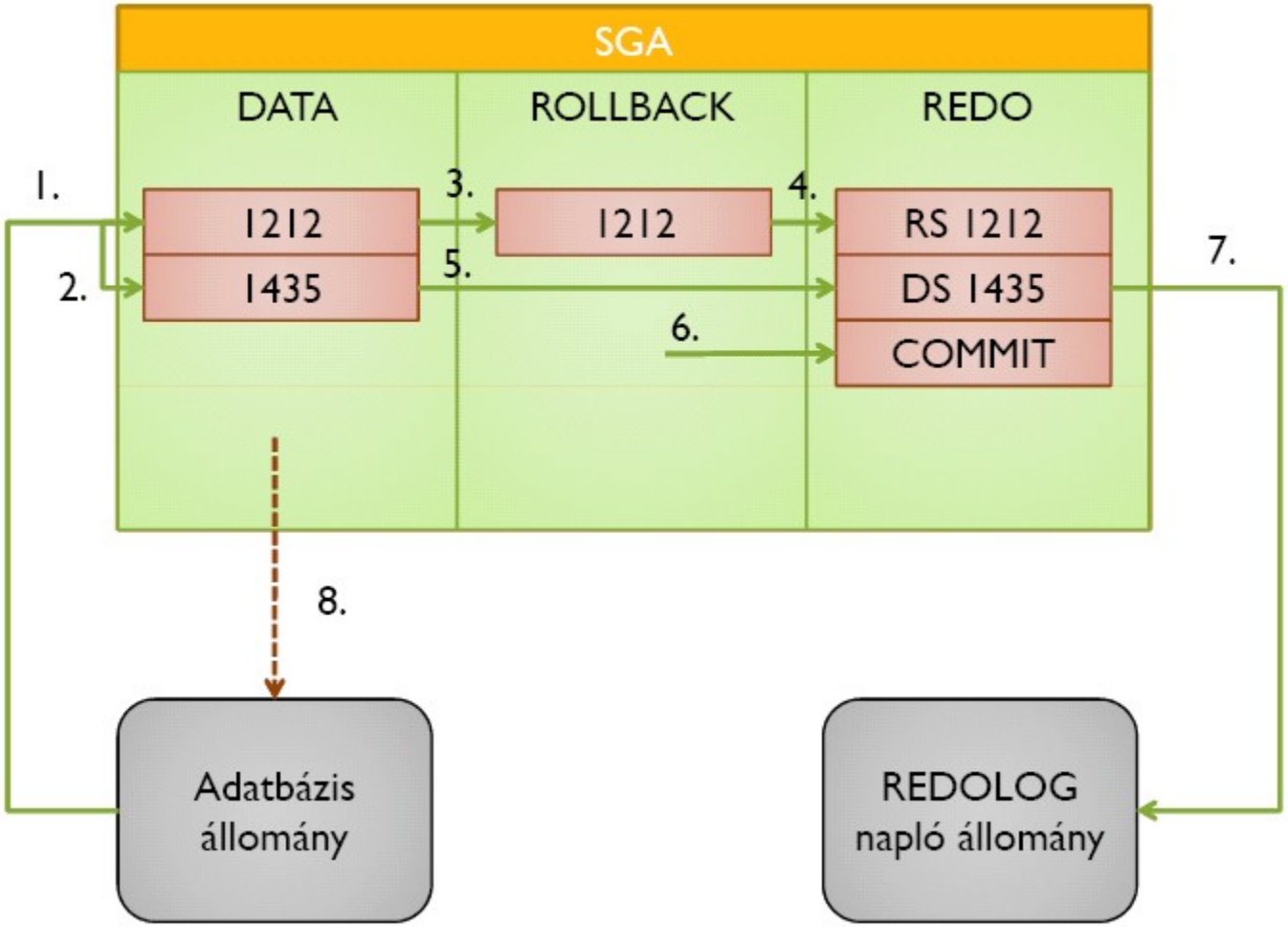
Megjegyzések zárolásokhoz

- DDL utasításoknál is figyelembe kell venni
  - szerkezetmódosítás v. törlés csak teljesen szabad táblánál lehetséges
  - a DDL utasítás csak a művelet idejére zárol
- Példa:
  - több különböző tranzakció ugyanazt a táblát bővíti
  - PRIMARY KEY szerepel a táblában
  - azonos kulcs felvitele esetén
    - az első megteheti a felvitelt
    - a második várakozik az első sikerességére
    - azután kap hibajelzést, miután az első befejeződött

Recovery mechanizmus -RM

- az adatok konzisztens állapotba való visszaállítása
- tekintettel kell lenni
  - a korábban elvégzett és lezárt műveletekre
  - az éppen futó tranzakciókra
- RM szükségessége:
  - tranzakció abortálásakor visszagörgetés
  - váratlan hibaesemény esetén újra felépítés
    - rendszer összeomlás (közös munkaterület elvész)
    - commit-tal lezárt adatmódosítások is elveszhetnek
- fizikai meghibásodás esetén helyreállítás

Oracle Recovery



Oracle Recovery

- UPDATE auto SET ar=1435 WHERE rsz='RST345';
  COMMIT;
- Lépések:
- 1. a rendszer megnézi, hogy a módosítandó objektum benne van-e a memóriában, ha nincs, akkor be kell hozni a megfelelő adatlapot
- 2. ha benne van, akkor átírható a korábbi értéke az új értékre
- 3. a módosítás elvégzése előtt az objektum korábbi értéke a ROLLBACK területre mentődik
- 4. sor kerül a módosításra a memóriában
- 5. az elvégzett tevékenységeket a rendszer naplózza a REDOLOG területen
- 6. naplózás után COMMIT, véglegesítődik a tranzakció hatása
  - 1. a ROLLBACK szegmensbeli bejegyzések törlődnek
  - 2. a REDOLOG területre bekerül a COMMIT végrehajtása is
- 7. a naplóállományba kimentődik a REDOLOG tartalma
- 8. az adatérték ezután íródik ki aszinkron módon

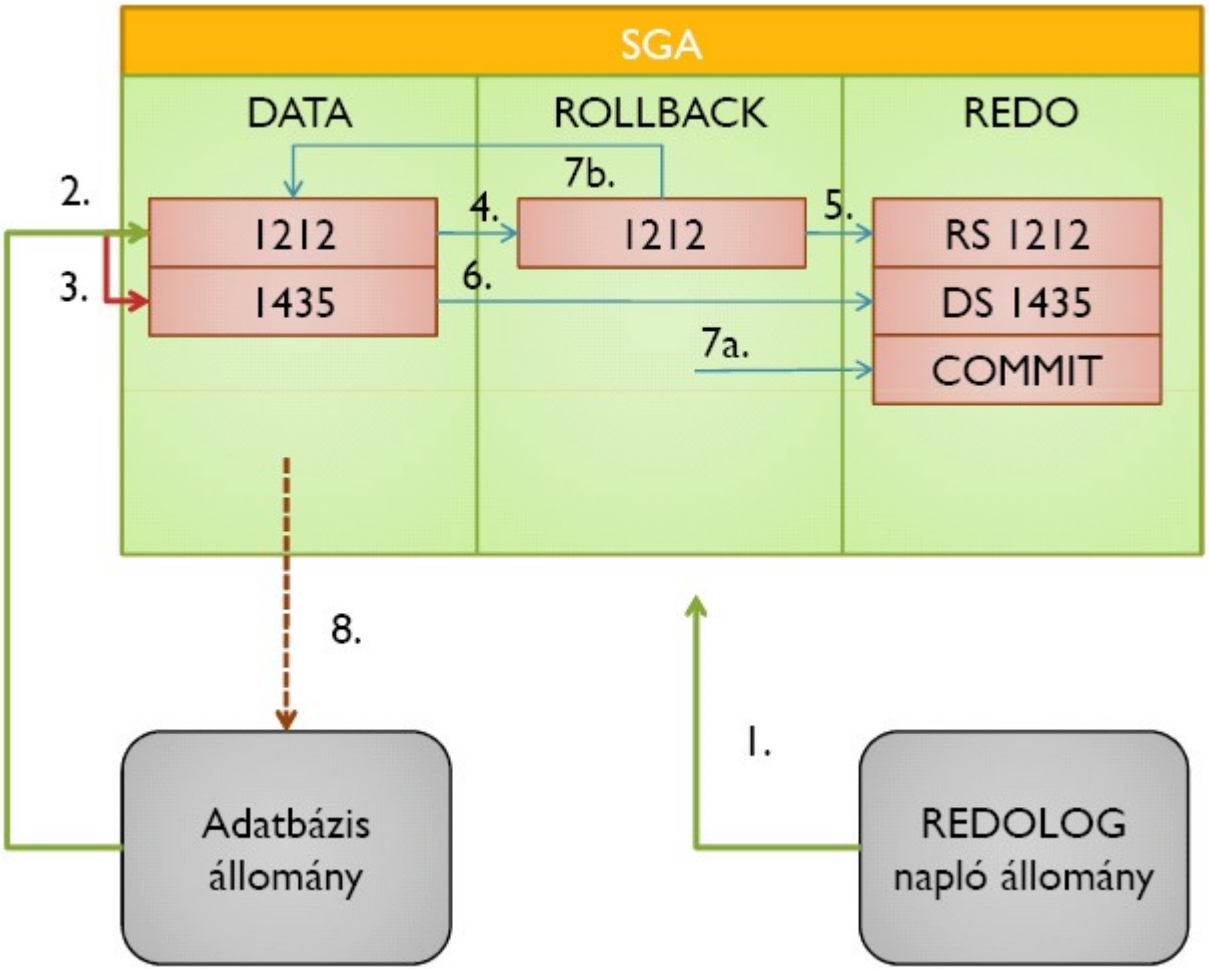
Helyreállítás esetén

- tranzakció abortálása esetén:
- a ROLLBACK szegmens aktuális tranzakcióhoz tartozó bejegyzéseinek végigolvasása
- az ott letárolt értékeket kell beírni az adatterület megfelelő objektumába
- ezután a ROLLBACK és a REDOLOG megfelelő bejegyzései törlődnek

Helyreállítás rendszerösszeomlás esetén

- bonyolultabb a helyzet
  - a REDOLOG-ban nemcsak lezárt tranzakcióra vonatkozó bejegyzések vannak
  - később eldönthető melyek záródtak le
    - tartalmaznak COMMIT bejegyzést is
- visszamentésnél gondolni kell a le nem zárult tranzakciók bejegyzéseire is

Helyreállítás rendszerösszeomlás esetén



Helyreállítás rendszerösszeomlás esetén

- Lépések:
  - 1. az Oracle sorba veszi azon műveleteket a REDOLOG-ból, melyek hatása még nem került át az adatbázis állományba
    - 1. a naplóállományban minden pontosan le van tárolva, így visszajátszhatóak
  - 2. ha a napló COMMIT-ot is tartalmaz, akkor a tranzakció újra véglegesítődik
  - 3. ha nincs COMMIT, akkor egy ROLLBACK utasítással visszagörgetődik az újrajátszott műveletsor

Helyreállítás sérülés esetén

- a REDOLOG segítségével újrajátszhatóak a tranzakciók
- nem lehet tetszőlegesen hosszú időre visszamenni a naplóállományban (kapacitása véges)
- az adatbázis tartalmat is rendszeresen menteni kell
- ha nincs meg a kívánt rendszerállapot, a helyreállítás nem oldható meg