“Capítol 14: Transaccions”

Fitxers i bases de dades

November 14, 2014

Capítol 14: Transaccions

- Concepte de transacció
- Estat de la transacció
- Les execucions simultànies
- Serializabilidad
- Recuperabilitat
- Aplicació de aïllament
- Definició de transaccions en SQL
- Proves per serializabilidad

Concepte de transacció

- Una transacció és una unitat d'execució del programa que accedeix i possiblement actualitza diversos elements de dades.
- Per exemple, transacció per transferir 50$ de compte A al compte B:
  1. **llegir** (A)
  2. A:= A − 50
  3. **escriure** (A)
  4. **llegir** (B)
  5. B:= B + 50
  6. **escriure** (B)
- Dos temes principals a tractar:
  - Les falles de diversos tipus, com ara errors de maquinari i fallades del sistema
  - Execució simultània de múltiples transaccions

Exemple de Transferència de Fons

- Transacció per transferir 50$ de compte A al compte B:
  1. **llegir** (A)
  2. A:= A − 50
  3. **escriure** (A)
  4. **llegir** (B)
  5. B:= B + 50
  6. **escriure** (B)
- Requisit d'atomicitat
  - Si l'operació falla després del pas 3 i abans del pas 6, els diners es "perdrà" el que condueix a un estat de base de dades inconsistents

    La manca pot ser degut a programari o maquinari
  - El sistema ha d'assegurar que les actualitzacions d'una transacció parcialment executada no es reflecteixen a la base de dades
- Requisit de durabilitat
  - Una vegada que l'usuari ha estat notificat que la transacció s'ha completat (és a dir, la transferència de els 50$ ha tingut lloc), els canvis a la base de dades de la transacció han de persistir encara que hi ha errors de programari o maquinari .

Exemple de Transferència de Fons (Cont.)

- Transacció per transferir 50$ de compte A al compte B:
  1. **llegir** (A)
  2. A:= A − 50
  3. **escriure** (A)
  4. **llegir** (B)
  5. B:= B + 50
  6. **escriure** (B)
- Requisit de consistència en l'exemple de dalt:
  - La suma de A i B és sense canvis per l'execució de la transacció

    En general els requisits de consistència inclouen:
    - Restriccions d'integritat explícitament especificats com claus principals i claus
    - Restriccions d'integritat implícits

      Per exemple, la suma dels saldos de totes les comptes, menys la suma dels sumes dels préstecs ha de ser igual valor del efectiu en mà

Exemple de Transferència de Fons (Cont.)

- Una transacció ha de veure una base de dades consistent
- Durant l'execució de transaccions, la base de dades pot ser temporalment inconsistent
- Quan la transacció es completa correctament la base de dades ha de ser consistent
  - Una transacció lògica errònia pot donar lloc a inconsistències

Exemple de Transferència de Fons (Cont.)

- Requisit d'aïllament

  Si entre els passos 3 i 6, es permet una altra transacció T2 per accedir a la base de dades parcialment actualitzada, veureu una base de dades inconsistents (la suma de A + B serà menor del que hauria de ser).

T1	T2
1. **llegir**(A)	
2. A:=A − 50	
3. **escriure** (A)	
	llegir(A), llegir(B), escriure(A+B)
4. **llegir**(B)	
5. B:= B + 50	
6. **escriure** (B)	

- L'aïllament es pot assegurar trivialment mitjançant l'execució de transaccions en sèrie (that is, one after the other)
- No obstant això, l'execució de múltiples operacions a la vegada té importants beneficis, com veurem més endavant

ACID Properties

Una transacció és una unitat d'execució del programa que accedeix i possiblement actualitza diversos elements de dades. Per preservar la integritat de les dades del sistema de base de dades ha de garantir que:

- Atomicidad. O totes les operacions de la transacció es reflecteixen adequadament a la base de dades o cap.
- La consistència. L'execució d'una transacció de manera aïllada conserva la consistència de la base de dades.
- Aïllament. Encara que diverses transaccions poden ejecutaser simultàniament, cada transacció ha de ser conscient de les altres transaccions que s'executen simultàniament. Els resultats de les transaccions intermèdies han d'estar ocults a les altres operacions executades simultàniament.
  - És a dir, per cada parell de transaccions T_i i T_j, aparegui T_i o bé T_j, l'execució finalitza abans que T_i comenci, o l'execució d' T_j comença després que T_i acabi.
- Durabilitat. Després d'una transacció es realitza correctament, els canvis que ha fet a la base de dades persisteixen, fins i tot si hi ha errors en el sistema.

Estat de la transacció

- Actiu - l'estat inicial; la transacció roman en aquest estat mentre s'està executant.
- Parcialment comès - després de la declaració final ha estat executat.
- Error - després que el descobriment que l'execució normal ja no pot continuar.
- Aborted - després de la transacció s'ha retrotret i la base de dades restaurada al seu estat previ a l'inici de la transacció. Dues opcions després d'haver estat avortat:
  - Reiniciar la transacció (es pot fer només si hi ha error lògic intern)
  - Matar la transacció
- Comès - després de completar amb èxit.

Estat de la transacció (Cont.)

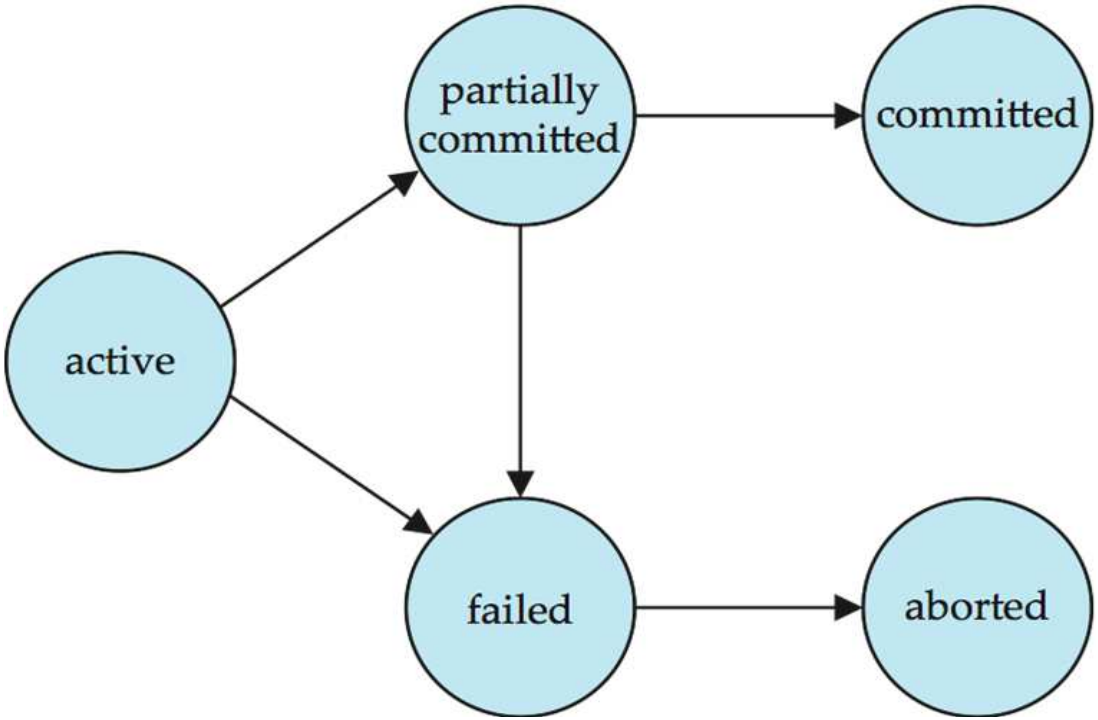

Figure: Diagrama l'estat de la transacció

Les execucions simultànies

- Múltiples transaccions es poden executar al mateix temps en el sistema. Els avantatges són:
  - **Augment de la utilització del processador i de disc**, el que porta a un millor rendiment de la transacció

    Per exemple, una transacció pot ser l'ús de la CPU, mentre que un altre està llegint o escrivint en el disc
  - **Reducció del temps de resposta mitjana per a les transaccions**: les transaccions curtes no han d'esperar darrere de les llargues.
- Esquemes de control de concurrència - mecanismes per aconseguir aïllament
  - És a dir, per controlar la interacció entre les transaccions simultànies per tal d'evitar que la destrucció de la consistència de la base de dades.

    Ho estudiarem en el capítol 16, després d'estudiar la noció de la correcció de les execucions concurrents.

Schedules

- Schedules - una seqüència d'instruccions que especifica l'ordre cronològic en què s'executen les instruccions de transaccions concurrents.
  - Un schedule per a un conjunt de transaccions ha de comprendre totes les instruccions d'aquestes transaccions
  - Han de preservar l'ordre en què apareixen en les instruccions de cada transacció individual
- Una transacció que completa amb èxit la seva execució tindrà una confirmació instruccions com l'última declaració
  - La transacció per defecte, assumeix executar la instrucció acordada com el seu últim pas
- Una transacció que no es completa amb èxit la seva execució tindrà una instrucció d'interrupció com l'última declaració

Schedules 1

- Atès que T_1 transfereix 50$ de A a B, i T_2 transfereix 10% del saldo d'A a B
- Un schedule en sèrie en el qual T_1 és seguit per T_2

T1	T2
llegir(A)	
A:=A − 50	
escriure (A)	
llegir(B)	
B:= B + 50	
escriure (B)	
fer	
	llegir(A)
	temp:= A∗0.1
	A:= A − temp
	escriure(A)
	llegir(B)
	B:= B + temp
	escriure(B)
	fer

Schedule 2

- Un schedule en sèrie en el qual T_2 és seguit per T_1

T1	T2
	llegir(A)
	temp:= A$*0.1$
	A:= A $-$ temp
	escriure(A)
	llegir(B)
	B:= B $+$ temp
	escriure(B)
	fer
llegir(A)	
A:=A $-$ 50	
escriure (A)	
llegir(B)	
B:= B $+$ 50	
escriure (B)	
fer	

Schedule 3

- Atès que T1 i T2 són les transaccions definides prèviament. El següent schedule no és un horari en sèrie, però és equivalent a l'horari 1.

T1	T2
llegir(A)	
A:=A − 50	
escriure (A)	
	llegir(A)
	temp:= A∗0.1
	A:= A − temp
	escriure(A)
llegir(B)	
B:= B + 50	
escriure (B)	
fer	
	llegir(B)
	B:= B + temp
	escriure(B)
	fer

En els schedules 1, 2 i 3, la suma de A + B es conserva.

Schedule 4

- El següent schedule concurrent no conserva el valor de (A + B).

T1	T2
llegir(A)	
A:=A − 50	
	llegir(A)
	temp:= A∗0.1
	A:= A − temp
	escriure(A)
	llegir(B)
escriure (A)	
llegir(B)	
B:= B + 50	
escriure (B)	
fer	
	B:= B + temp
	escriure(B)
	fer

Serializabilidad

- **Supòsit bàsic** - Cada transacció preserva la coherència de base de dades.
- Per tant l'execució en sèrie d'un conjunt de transaccions conserva la consistència de base de dades.
- Un programa (possiblement concurrent) és serialitzada si és equivalent a un schedule en sèrie. Diferents formes d'equivalències horàries donen lloc a les nocions de:
  1. Conflicte de seqüencialitat
  2. Visió de seqüencialitat

Visió simplificada de les transaccions

- Ignorem operacions diferents de llegir i escriure instruccions
- Suposem que les transaccions poden realitzar càlculs arbitraris en les dades en buffers locals entre les lectures i escriptures
- Els nostres horaris simplificats consisteixen de només llegir i escriure instruccions

Instruccions en conflicte

- Les instruccions l_i i l_j de les transaccions T_i i T_j respectivament, entren en conflicte si i només si hi ha algun element Q visitat per les dues l_i i l_j, i almenys una d'aquestes instruccions escriure Q.
  1. l_i = llegir(Q), l_j = llegir(Q). l_i i l_j no entren en conflicte.
  2. l_i = llegir(Q), l_j = escriure(Q). Entren en conflicte.
  3. l_i = escriure(Q), l_j = llegir(Q). Entren en conflicte.
  4. l_i = escriure(Q), l_j = escriure(Q). Entren en conflicte.
- Intuïtivament, un conflicte entre l_i i l_j força un ordre temporal (lògic) entre ells.
  - Si l_i i l_j són consecutius en el schedule i elles no entren en conflicte, els seus resultats seguirien sent els mateixos, fins i tot si s'haguessin intercanviat en el schedule.

Conflicte de serializabilidad

- Si un schedule S es pot transformar en un schedule S' per una sèrie d'intercanvis d'instruccions que no causin conflictes, diem que S i S' són equivalents en conflicte.
- Diem que un schedule S està en conflicte serialitzada si és equivalent en conflicte a un schedule seqüencial.

Conflicte de serializabilidad (Cont.)

- Schedule 3 es pot transformar en schedule 6, una planificació seqüencial on T2 segueix T1, per una sèrie d'intercanvis d'instruccions que no causin conflictes. Per tant el schedule 3 és conflicte serialitzada.

T1	**T2**	**T1**	**T2**
llegir(A)		**llegir**(A)	
escriure (A)		**escriure** (A)	
	llegir(A)	**llegir** (B)	
	escriure(A)	**escriure** (B)	
llegir(B)			**llegir** (A)
escriure(B)			**escriure** (A)
	llegir(B)		**llegir** (B)
	escriure(B)		**escriure** (B)
Schedule3		**Schedule 6**	

Table: Conflicte de serializabilidad

Conflicte de serializabilidad (Cont.)

- Exemple d'un schedule que no està en conflicte serialitzada:

T3	T4
llegir(Q)	
	escriure (Q)
escriure (Q)	

Table: No conflicte de serializabilidad

- No podem intercanviar les instruccions en el programa anterior per obtenir ja sigui la planificació seqüencial <T3, T4>, o la planificació seqüencial <T4, T3>.

Visió de serializabilidad

- Siguin S i S' dos schedules amb el mateix conjunt de transaccions. S i S' són visió equivalent si es compleixen les tres condicions següents, per a cada element de dades Q:
  1. Si en el schedule S, la transacció T_i llegeix el valor inicial de Q, a continuació, en el schedule de S' també transacció T_i ha de llegir el valor inicial de Q.
  2. Si en el schedule S la transacció T_i executa **llegir**(Q), i aquest valor va ser produït per transacció T_j (si n'hi ha), llavors en el schedule S' la transacció T_i també ha de llegir el valor de Q que va ser produït per la mateixa operació **d'escriptura** (Q) de la transacció T_j.
  3. La transacció (si existeix) que porta a terme l'operació final **d'escriptura** (Q) en el schedule de S també ha de realitzar l'operació final **d'escriptura**(Q) en el schedule S'.

  Com es pot veure, la visió d'equivalència també es basa exclusivament en lectures i escriptures soles.

Visió de serializabilidad (Cont.)

- Un schedule S és visió serialitzada si és visió equivalent a una planificació seqüencial.
- Cada schedule conflicte serialitzada és també visió serialitzada.
- A continuació es mostra un schedule que és visió serialitzada però no conflicte serialitzada.

T_{27}	T_{28}	T_{29}
llegir(Q)		
	escriure (Q)	
escriure (Q)		
		escriure (Q)

Table: Visió de serializabilidad

- Quin schedule de sèrie està per sobre equivalent a?
- Cada schedule visió serialitzada que no està en conflicte serialitzada té escriptures cecs.

Altres nocions de serializabilidad

- El schedule de dalt genera el mateix resultat que la planificació seqüencial <T1, T5>, encara que no és conflicte equivalent o visió equivalent a ella.

T1	T5
llegir(A)	
A:=A − 50	
escriure (A)	
	llegir(B)
	B:= B − 10
	escriure(B)
llegir(B)	
B:= B + 50	
escriure (B)	
	llegir(A)
	A:= A + 10
	escriure(A)

- Determinar tal equivalència requereix l'anàlisi d'operacions diferents de lectura i escriptura.

Proves per serializabilidad

- Penseu alguns schedules d'un conjunt de transaccions T_1, T_2, ..., T_n.
- Grafo de precedència - un graf dirigit, on els vèrtexs són les transaccions (noms).
- Dibuixem un arc de T_i a T_j si les dues transaccions entren en conflicte, i T_i accedir a l'element de dades en què el conflicte va sorgir abans.
- Podem etiquetar l'arc per l'element al qual va tenir accés.

Exemple 1

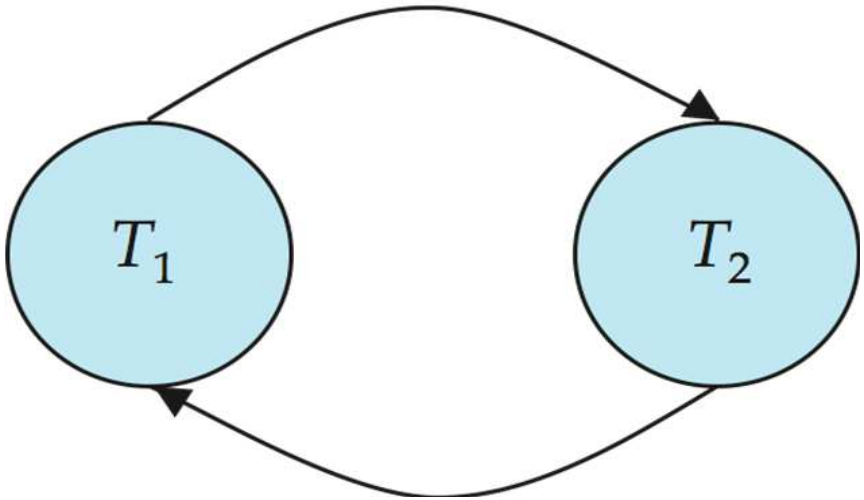

Figure: Grafo de precedència

Proves per conflictes de serializabilidad

- Un schedule és serialitzada si i només si el seu graf de precedència és acíclic.
- Hi algorismes de detecció de cicle, que prenen un ordre de n^2, on n és el nombre de vèrtexs del graf.
  - (Millors algorismes prenen un ordre $n + e$ on e és el nombre d'arestes.)
- Si el graf de precedència és acíclic, l'ordre de serialització es pot aconseguir per una classificació topològica del graf.
  - Això és un ordre lineal consistent amb l'ordre parcial del graf.
  - Per exemple, una ordre de la seqüencialitat del schedule A seria:
    $T_5 \longrightarrow T_1 \longrightarrow T_3 \longrightarrow T_2 \longrightarrow T_4$
    Hi ha altres?

Proves per conflictes de serializabilidad (Cont.)

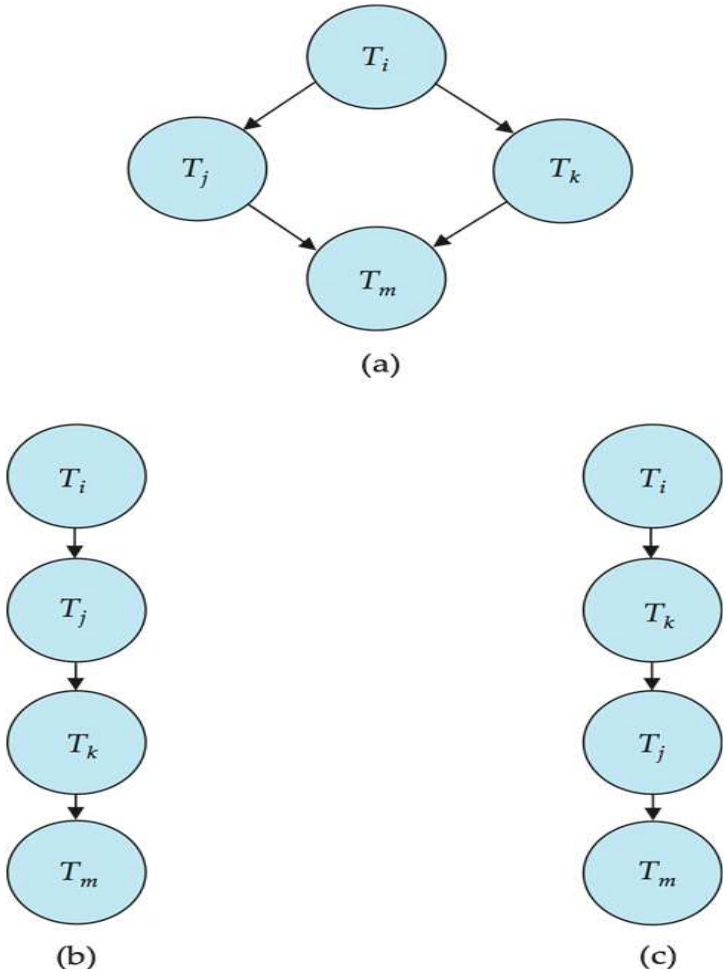

Figure: Classificació topològica

Prova per veure la serializabilidad

- La prova per al conflicte de serializabilidad del graf precedent no es pot utilitzar directament per provar sobre view serializability.
  - L'extensió per provar per view serializability ha tingut un cost exponencial pel que fa a la mida del graf precedent.
- El problema de comprovar si un schedule és view serializability cau en la classe de problemes NP-complets.
  - Per tant és extremadament improbable l'existència d'un algoritme eficient.
- No obstant això algoritmes pràctics que només comproven algunes condicions suficients per a la view serializability es poden seguir utilitzant.

Schedules recuperables

Necessitat d'abordar els efectes dels errors de transacció sobre l'execució de transaccions simultànies.

- Schedule recuperable - si una transacció T_j llegeix un element de dades prèviament escrit per una transacció T_i, llavors l'operació de commit de T_i apareix abans de l'operació de commit de T_j.
- El següent schedule (schedule 11) no és recuperable si T9 commit immediatament després de la lectura.

T8	T9
llegir(A)	
escriure (A)	
	llegir(A)
	commit
llegir(B)	

- Si T8 ha avortar, T9 hauria llegit (i, possiblement, es mostra a l'usuari) un estat de base de dades inconsistents. Per tant, la base de dades ha de garantir que els schedules són recuperables.

Cascading rollbacks

- Cascading rollback un sol error transacció dóna lloc a una sèrie de reversions de transacció. Considereu el schedule en què cap de les transaccions encara s'ha commit (pel que el schedule és recuperable).

T10	T11	T12
llegir(A)		
llegir (B)		
escriure (A)		
	llegir(A)	
	escriure(A)	
		llegir(A)
avortar		

  Si falla T_{10}, T_{11} i T_{12} també fer-la retrocedir.

- Pot conduir a la ruïna d'una quantitat significativa de treball.

Cascadeless schedules

- Cascadeless schedules — cascading rollbacks no poden ocórrer; per a cada parell de transaccions T_i i T_j tal que T_j llegeix un element de dades prèviament escrit per T_i, apareix l'operació de commit de T_i abans de l'operació de lectura de T_j.
- Cada cascadeless schedule també és recuperable.
- És desitjable restringir els schedules als quals són cascadeless.

Control de concurrència

- Una base de dades ha de proporcionar un mecanisme que asseguri que tots els schedules possibles són:
  - conflicte o visió serialitzada, i
  - són recuperables i preferiblement cascadeless
- Una política en la qual només una transacció pot executar alhora genera schedules de sèrie, però ofereix un pobre grau de concurrència.
  - Són els schedules de sèrie recuperable / cascadeless?
- Provar un schedule per a la seqüencialitat després que ha executat és una mica massa tard!.
- Objectiu - desenvolupar els protocols de control de concurrència que assegurin la seqüencialitat.

Control de concurrència (Cont.)

- Els schedules han d'estar en conflicte o ser visió serialitzada, i recuperable, en nom de la coherència de base de dades, i preferiblement cascadeless.
- Una política en la qual només una transacció pot executar alhora genera schedules de sèrie, però ofereix un pobre grau de concurrència.
- Esquemes de control de concurrència equilibri entre la quantitat de concurrència que permeten i la quantitat de sobrecàrrega en què incorren.
- Alguns esquemes només permeten que es generin schedules conflicte-serializables, mentre que altres permeten schedules-visió serialitzada que no són conflicte-serializables.

Control de concurrència vs Proves d'serializabilidad

- Els protocols de control de concurrència permeten schedules concurrents, però assegureu-vos que els schedules són conflicte / visió serialitzada, i són recuperables i cascadeless.
- Els protocols de control de concurrència en general no examinen el graf de precedència, ja que està sent creant.
  - En lloc d'un protocol imposa una disciplina que evita schedules nonseralizable.
  - Estudiem aquests protocols en el capítol 16.
- Els diferents protocols de control de concurrència ofereixen diferents solucions de compromís entre la quantitat de concurrència que permeten i la quantitat de sobrecàrrega que es produeixen.
- Les proves per a la seqüencialitat ens ajuden a entendre per què un protocol de control de concurrència és correcta.

Nivells febles de consistència

- Algunes aplicacions estan disposats a viure amb nivells febles de consistència, permetent schedules que no són serializables:/
  - Per exemple, una sola transacció de lectura que vol aconseguir un equilibri total aproximat de tots els comptes.
  - Per exemple, les estadístiques de base de dades calculades per a l'optimització de consultes poden ser aproximades (per què?).
  - Aquestes operacions no necessiten ser serializables respecte a altres transaccions.
- Exactitud de tradeoff per acompliment.

Nivells de consistència en SQL-92

- Serialitzada - per defecte.
- Lectura repetible - només els registres committed per ser llegits, lectures repetides del mateix registre han de tornar el mateix valor. No obstant això, una transacció pot no ser serialitzada - pot trobar alguns registres inserits per una transacció, però no trobar altres.
- Lectura committed - només registres committed es poden llegir, però successives lectures de registres poden retornar diferents (però compromesos) valors.
- Lectura uncommited - fins i tot els registres no confirmats poden llegir.
- Menors graus de consistència útils per a la recopilació d'informació aproximada.
- Avís: alguns sistemes de bases de dades no garanteixen schedules serializables per defecte.
    - Per exemple, Oracle i PostgreSQL per defecte suporten un nivell de coherència anomenat aïllament de instantània (no és part de l'estàndard SQL).

Definició de transaccions en SQL

- El llenguatge de manipulació de dades ha d'incloure una construcció per especificar el conjunt d'accions que componen una transacció.
- En SQL, una transacció comença implícitament.
- Una transacció en SQL acaba per:
  - **Commit work** - commits treball compromet la transacció actual i comença una nova.
  - **Rollback work** - causa l'avortament de transacció actual.
- En gairebé tots els sistemes de bases de dades, per defecte, cada sentència SQL també es commits implícitament, si s'executa correctament.
  - Implícit commit es pot apagar per una directiva de base de dades.

    Per exemple, en JDBC, connection.setAutoCommit (false);

Fina del Capítol 14

Figures

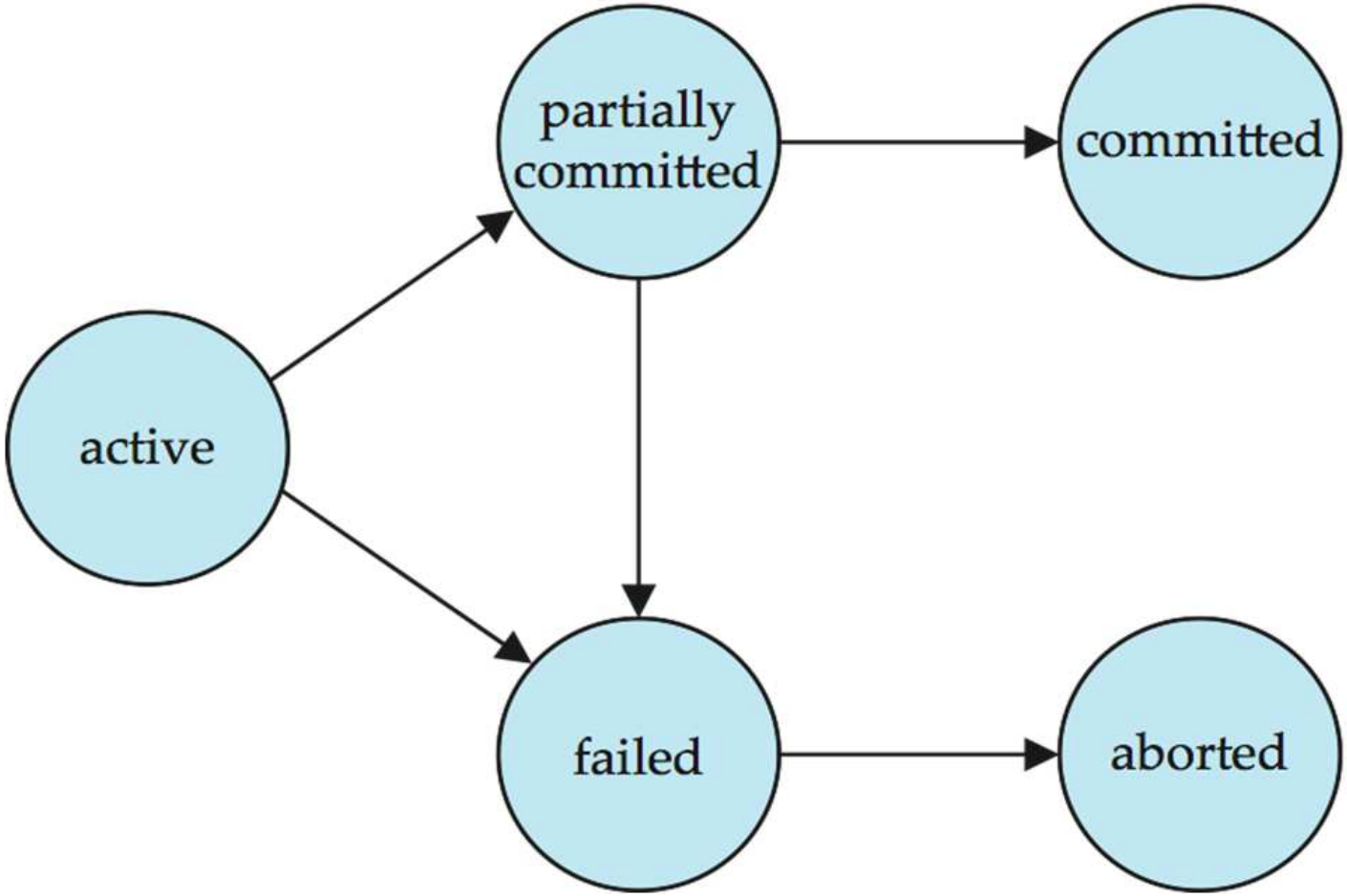

Figure: 14.01

Figures

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Figure: 14.02

Figures

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

Figure: 14.03

Figures

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

Figure: 14.04

Figures

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

Figure: 14.05

Figures

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Figure: 14.06

Figures

T_1	T_2
read (A)	
write (A)	
	read (A)
read (B)	
	write (A)
write (B)	
	read (B)
	write (B)

Figure: 14.07

Figures

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Figure: 14.08

Figures

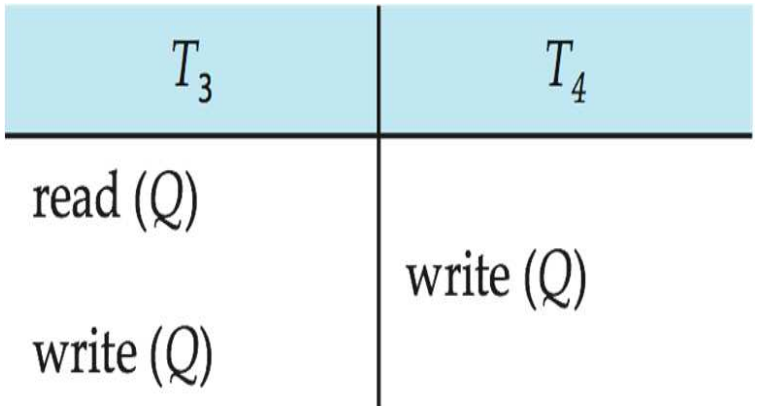

Figure: 14.09

Figures

Figure: 14.10

Figures

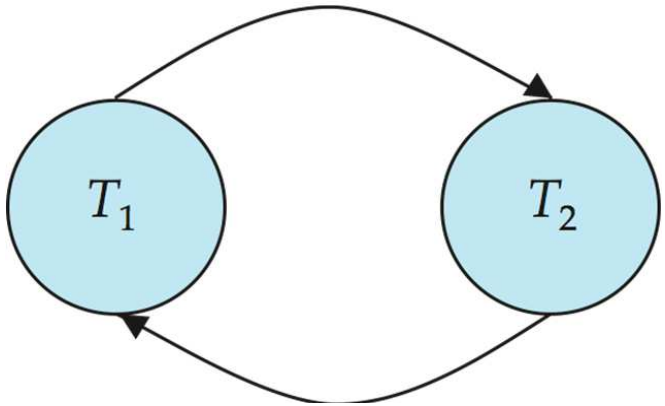

Figure: 14.11

Figures

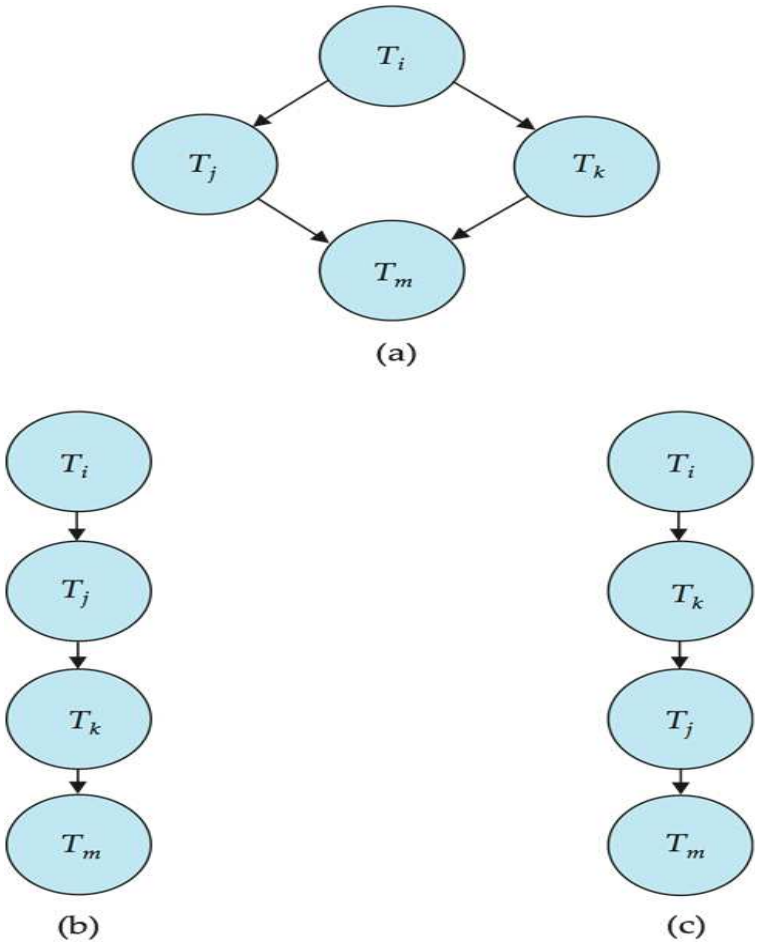

Figure: 14.12

Figures

T_1	T_5
read (A)	
$A := A - 50$	
write (A)	
	read (B)
	$B := B - 10$
	write (B)
read (B)	
$B := B + 50$	
write (B)	
	read (A)
	$A := A + 10$
	write (A)

Figure: 14.13

Figures

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

Figure: 14.14

Figures

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

Figure: 14.15

Figures

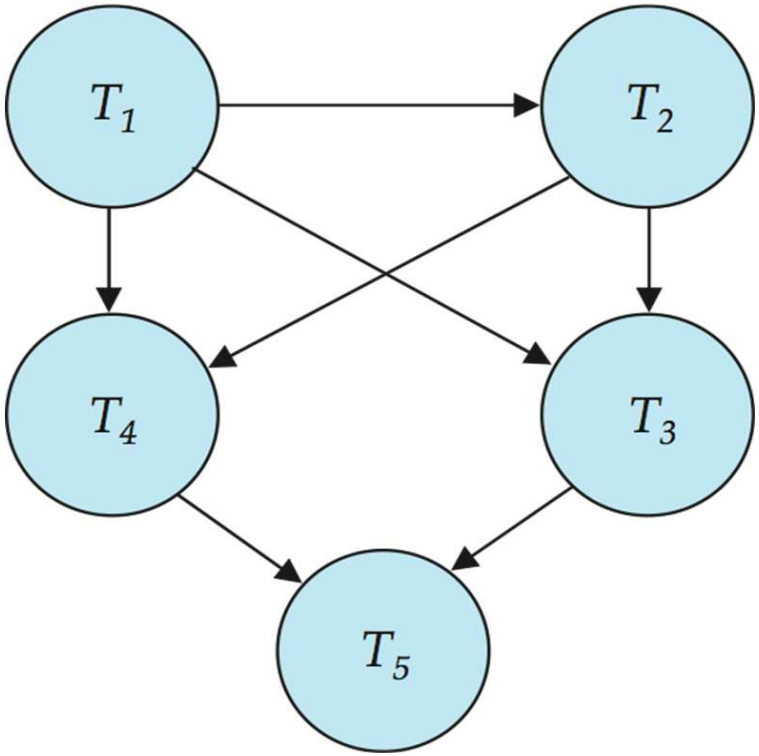

Figure: 14.16