ACID-Transaktioner

Atomicity, Consistency, Isolation, Durability

www.everyloop.com

ACID-Transaktioner

- En transaktion är en följd av operationer som hör ihop som en enhet.
- Ett exempel kan vara att flytta pengar från ett bankkonto till ett annat. Då drar man först bort beloppet från det ena kontot, och adderar det sen till det andra kontot.
- Man brukar i databassammanhang tala om ACID.
- Bokstäverna i ACID anger fyra egenskaper som krävs för att transaktioner ska kunna genomföras korrekt och säkert.
- ACID är implementerat i databashanteraren för att garantera transaktionernas giltighet.

Atomicity

En transaktion ska var odelbar. Antingen utförs hela transaktionen, eller inget alls. Om transaktionen avbryts mitt i, innan den hunnit gå klart så måste databashanteraren rulla tillbaks de ändringar som redan gjorts och säkerställa att data är i samma skick som innan.

Transaktionen till höger för över 100 kronor från konto x till konto y. Om den avbryts efter steg 1, men innan steg 2, så är informationen i databasen ogiltig. Därför måste transaktionen utföras i sin helhet (eller rullas tillbaks).

Steg 1	**Steg 2**
Före: x = 500	**Före:** y = 200
Läs in (x) x := x – 100 Spara (x)	Läs in (y) y := y + 100 Spara (y)
Efter: x = 400	**Efter:** y = 300

Consistency

- Databasen får inte innehålla några inre motsägelser.
- I exemplet ovan måste t.ex det totala beloppet efter transaktionen (400+300) vara samma som innan transaktionen (500+200).
- Om transaktionen avbröts mellan steg 1 och steg 2 så skulle totalbeloppet ändras och informationen i databasen vara fel.
- Man brukar tala om dataintegritet, och sätta upp integritetsvilkor som all data i databasen måste följa.
- Vi kommer kolla mer på dataintegritet längre fram i kursen.

Isolation

Transaktioner måste vara isolerade från varandra, även om de utförs parallellt. Den ena transaktionens ändringar får inte visas för den andra transaktionen förrän **hela** första transaktionen är klar.

Antag att x = 5 och y = 100, och att vi har två transaktioner som körs parallellt. Om den andra transaktionen startar när den första precis sparat x, så kommer den läsa in det nya x-värdet, men det gamla y-värdet, eftersom transaktion 1 inte hunnit uppdatera det. Då skulle vi (felaktigt) få z = 500 + 100 = 600, istället för det förväntade z = 500 + 50 = 550.

Transaktion 1	Transaktion 2
Läs in (x) x := x * 100 Spara (x) Läs in (y) y := y – 50 Spara (y)	Läs in (x) Läs in (y) z := x + y Spara (z)

Durability

När en transaktion är (korrekt) genomförd måste alla uppdateringar och modifikationer i databasen lagras på ett beständigt sätt. Dvs, skrivas till disk (i motsats till att endast hållas i minnet), så att alla ändringar kvarstår även om ett systemfel inträffar.

När en transaktion är klar ska den finnas kvar i systemet på ett säkert sätt och inte kunna gå förlorad.

Commit och Rollback

- Kommandot ”Commit” används för att tala om för databashanteraren att transaktionen i sin helhet är klar och ska genomföras.
- Det är först när databashanteraren stöter på ”Commit”-kommandot som den permanent lagrar de ändringar som utförts genom pågående transaktion.
- Kommandot ”Rollback” talar om att den pågående transaktionen ska avbrytas och rullas tillbaks i sin helhet. Databashanteraren återställer då all data så som det var innan transaktionen påbörjades.

Exempel på en transaktion

Så här kan en överföring mellan två konton se ut i SQL-kod:

```
BEGIN TRANSACTION;
UPDATE Accounts SET amount = amount – 100 where id = 345234563;
UPDATE Accounts SET amount = amount + 100 where id = 348754372;
COMMIT;
```

Om man inte omger de två update-satserna med ”Begin transaction” och ”Commit” så behandlas de som två separata transaktioner.

Återställning vid krasch

Om strömmen går eller datorn kraschar mitt i en transaktion så kommer databashanteraren behöver göra en återställning (recovery) vid nästa uppstart.

- Avbrutna, ej färdiga, transaktioner behöver tas bort. Dvs, om de hunnit göra några ändringar i databasen så måste de rullas tillbaks.
- Även färdiga transaktioner måste kontrollerar. Om dessa gjort ändringar som inte hunnit skrivas på disk så måste det göras nu.

Transaktionslogg

För att möjliggöra för databasen att rulla tillbaks transaktioner (vid t.ex krasch eller då den stöter på ”rollback”-kommandot) så krävs det att den loggar alla förändringar till en fil **innan** de utförs.

För varje ändring som görs skriver databashanteraren först i loggfilen:

- Vilken transaktion det rör sig om.
- Vilket dataobjekt som berörs.
- Vad det gamla värdet på det objektet var, och
- Vad det nya värdet ska bli.

Backup

Man kan även använda transaktionsloggen för att återskapa alla tidigare transaktioner om det skulle behövas.

Säg att man tar en backup av databasen varje natt, men att hela databasen krashar (t.ex en hårddisk går sönder) på eftermiddagen. Om man då återställer backupen så skulle alla transaktioner som skett under dagen (sedan senaste backup) varit förlorade om man inte haft tillgång till transaktionsloggen.

Med transaktionsloggen (som man förhoppningsvis skrivit på en annan disk) kan man gå igenom alla transaktioner sedan gårdagen steg för steg och på så vis få tillbaks all data fram till precis innan krashen.

Lås och blockeringar

För att säkerställa att alla transaktioner är isolerade (I:et i ACID) från varandra så använder databashanteraren en låsmekanism. I exemplet vi hade så kommer databasen låsa den första transaktionen till det dataobjekt den ska uppdatera, och på så sätt blockera alla andra transaktioner från att använda det objektet innan den första transaktioner är klar och blockeringen hävs.

Detta är fullt normalt, men kan skapa prestandaproblem om vissa transaktioner låser ute andra under längre tid.

Det kan även hända att sessioner låser en resurs och aldrig släpper den t.ex en applikation som tappar anslutning och därför aldrig sänder en ”commit transaction”. Man kan då behöva ”döda” den processen.

Deadlocks

När två transaktioner blockerar varandra uppstår ett dödläge där ingen transaktion kan avslutas, och därmed inte låsa upp sina resurser.

- Transaktion 1 har låst tabell A
- Transaktion 2 har låst tabell B
- Transaktion 1 begär ett lås på tabell B, men blockeras av transaktion 2
- Transaktion 2 begär ett lås på tabell A, men blockeras av transaktion 1

Databashanteraren kommer upptäcka detta och välja att avsluta (och rulla tillbaks) den ena transaktionen med felmeddelandet ”Transaction was deadlocked and has been chosen as victim. Rerun the transaction.”

Deadlocks beror på dåligt designad applikationskod i kombination med ett databasschema som resulterar i ett åtkomstmönster som leder till ett cykliskt beroende av resurser.