Transaksjoner og flerbrukerproblematikk

Læreboka, kapittel 8.2, side 248-265

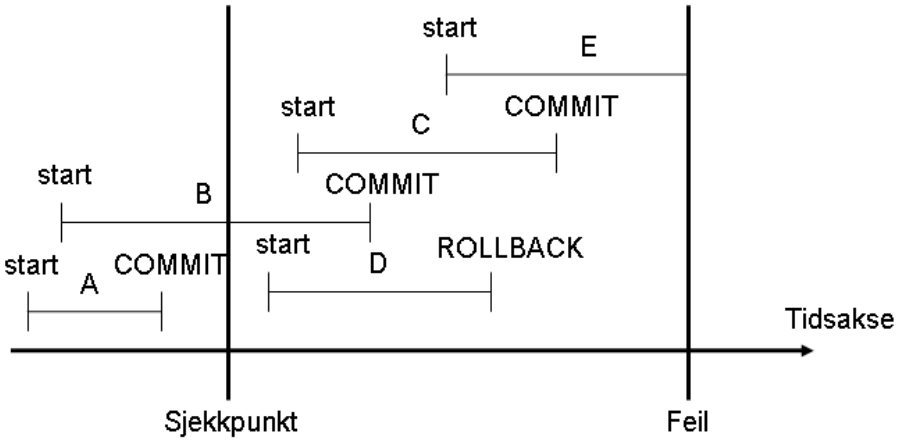
Transaksjoner

- En *transaksjon* er en logisk enhet med arbeid (ett sett med database-operasjoner) som må utføres i sin helhet. Eksempler:
  - overføre penger fra en konto til en annen
  - registrere en boktittel og 1.eksemplar av tittelen (insert + insert)
    - registrere nytt eksemplar med nr 1 større enn hittil brukte (select + insert)
- Dersom alt gikk bra, bekreftes transaksjonen (*commit*), hvis ikke rulles den tilbake (*rollback*)
- Egenskaper ved transaksjoner (ACID)
  - **A**tomic. En transaksjon er ***atomisk***. Enten utføres hele transaksjonen, eller ingenting.
  - **C**onsistency. Transaksjonen må føre databasen fra en stabil tilstand til en annen stabil tilstand. Data***konsistens*** må sikres.
  - **I**solation. Dataene som berøres av en transaksjon må ***isoleres***, slik at ikke andre transaksjoner kan arbeide med de samme dataene. Forskjellige nivåer av isolering.
  - **D**urability. Etter at en transaksjon er avsluttet, skal endringene lagres **varig**/permanent og bli synlige for andre brukere.
- Samtidige transaksjoner må utføres *som om de ble utført i serie*, dvs. etter hverandre.
  - Det betyr at transaksjonene av og til vil måtte vente på hverandre
- Transaksjoner er enheten både ved utførelse og ved gjenoppretting (recovery).

Gjenoppretting av en database (”recovery”)

- Mekanismer for å gjenopprette en database etter feil
  - periodiske sikkerhetskopier
  - logg som inneholder
    - endringer (dataverdi før og etter endring) siden forrige sikkerhetskopi, inkl transaksjoner som ikke er bekreftet
    - sjekkpunkt = tidspunkt som viser når permanente endringer ble utført (databasebufrene skrives til disk)
    - transaksjoner kan pågå på et sjekkpunkt
- Bruk av sikkerhetskopi + loggfil
  - kan være tidkrevende
  - nyeste sikkerhetskopi legges inn
  - hele loggfilen kjøres
    - bekreftede transaksjoner kjøres på nytt
    - tilbakerullede transaksjoner glemmes
    - transaksjoner under utførelse tilbakerulles, og kjøres eventuelt på nytt senere
- Bruk av loggfil pga uforutsette feil
  - går tilbake til siste sjekkpunkt
  - pass på at transaksjonen som forårsaket feilen ikke kjøres på nytt

Figur 8.1 Eksempel på logg av transaksjoner



Transaksjoner og AUTOCOMMIT

- AUTOCOMMIT er en tilstand som kan slås av og på
- AUTOCOMMIT er en egenskap som den enkelte klient har i forhold til sin databasetilknytning
- AUTOCOMMIT *på* betyr at
  - oppdateringer av databasen (INSERT/DELETE/UPDATE) skjer umiddelbart etter hver enkelt SQL-setning. Ikke mulig å angre ved å bruke ROLLBACK
  -
- JDBC
  - AUTOCOMMIT *på* er standard

```
try {
  forbindelse.setAutoCommit(false);  // slå av autocommit før transaksjoner
  ... mer enn én UPDATE/INSERT/DELETE-setning) utføres ...
  forbindelse.commit()
} catch (SQLException e) {
   rullTilbake();  // vi angrer det som måtte være gjort (forbindelse.rollback())
   ... skriv eventuelle meldinger ...
} finally {   // utføres alltid (unntatt hvis System.exit() påtreffes før)
   ... lukk databaseressurser ...
  slåPåAutoCommit();  // forbindelse.setAutoCommit(true);
}
```

  - Stol ikke på at det ene (commit) eller det andre (rollback) skjer dersom man ikke avslutter på ryddig måte

Transaksjoners start og slutt

- Vanlig at hver enkelt SQL-setning utgjør en transaksjon.
  - Det vil for de fleste DBMS si at bryteren AUTOCOMMIT er på
- Hvis vi vil at en transaksjon skal bestå av flere setninger, må vi definere start og slutt av transaksjonen
- Transaksjonen start
  - ofte/vanligvis
    - sørg for at AUTOCOMMIT er slått av
    - kjør deretter setningene som utgjør transaksjonen
  - noen DBMS har egen syntaks for dette, f.eks. START TRANSACTION
- MySQL (Shell) ?
  - https://dev.mysql.com/doc/refman/5.7/en/commit.html
- Transaksjonen slutter ved første
  - COMMIT eller ROLLBACK
  - husk å sette tilbake AUTOCOMMIT

Låseteknikker

- Låsing begrenser andre transaksjoners tilgang til dataene.
- Granulariteten, dvs hvor stor del av databasen som låses:
  - Verdi, rad, tabell eller hele databasen – kalles ”objekt” nedenfor
  - Låsing på radnivå er standard i Oracle, JavaDB og MySQL (ikke ved manuell låseing)
- To typer lås
  - Delt lås (shared lock) - *leselås*:
    - Brukes kun ved lesing og godtar at flere transaksjoner leser de samme dataene.
    - Flere transaksjoner kan ha delt lås på de samme objektene.
    - Dersom en transaksjon holder delt lås på et objekt, kan andre transaksjoner lese, men ikke endre dataene.
  - Eksklusiv lås (exclusive lock) – *skrivelås:*
    - Kan settes av kun én transaksjon av gangen.
    - Forutsetter at det ikke er andre låser på objektet, heller ikke delte låser.
    - Andre transaksjoner har ikke tilgang til dataene i det hele tatt.
- To-fase-låsing
  - Garanterer korrekt utførelse av transaksjoner
  - Fase 1, utvidelsesfasen: Låser settes etter hvert som de trengs i transaksjonen.
  - Fase 2, avviklingsfasen (COMMIT/ROLLBACK): Alle låsene åpnes på en gang.
- Vranglås kan oppstå hvis to transaksjoner blir stående å vente på hverandre. Løses ved
  - timeout. En av transaksjonene ofres (tilbakerulles)
  - at alle låser settes ved begynnelsen av transaksjonen. Ueffektivt.
  - avanserte løsninger prøver å oppdage mulige vranglåser på forhånd

Ulike isolasjonsnivåer

- Dataene som berøres av en transaksjon må *isoleres*, slik at ikke andre transaksjoner kan arbeide med de samme dataene.
- Isolasjonsnivå gjelder den enkelte klient.
- Fullstendig isolasjon (isolasjonsnivå SERIALIZABLE)
  - SELECT setter delt lås (untatt SELECT ... FOR UPDATE)
  - UPDATE setter eksklusiv lås
- I praksis er dette et for strengt krav
- Ulike nivåer defineres ut fra i hvor stor grad de bryter kravet til fullstendig isolasjon (SERIALIZABLE)
  - **Dirty read**
    - Transaksjon B kan se data som transaksjon A har endret, men ikke bekreftet (committed). Det vil si at dataene kan bli rullet tilbake. (“dirty read”)
  - **Non-repeatable read**
    - B leser en verdi som A senere endrer. Hvis B leser verdien på nytt i samme transaksjon, men nå ser den endringen A foretok, har vi et eksempel på ”ikke-repeterende lesing”
  - **Phantom read**
    - B leser et sett med rader begrenset av et WHERE-uttrykk
    - A legger inn data som passer inn i intervallet gitt av WHERE-uttrykket
    - B leser på nytt, men ser nå også de dataene A la inn (data som tidligere ikke fantes, fantomdata)

Isolasjonsnivåer, forts.

Nivå (ANSI SQL)	Dirty read	Nonrepeatable Read	Phantom Read
READ UNCOMMITTED	tillatt	tillatt	tillatt
READ COMMITTED	ikke tillatt	tillatt	tillatt
REPEATABLE READ	ikke tillatt	ikke tillatt	tillatt
SERIALIZABLE	ikke tillatt	ikke tillatt	ikke tillatt

- Standard isolasjonsnivå er gjerne **READ COMMITTED**/REPEATABLE READ
  - select låser ikke data
  - kun insert, delete og update
- SERIALIZABLE
  - også select/spørring låser

Isolasjonsnivåer og DBMS-støtte

- Databasesystemer støtter ikke nødvnedigvis alle isolasjonsnivå
  - Oracle støtter READ COMMITTED (standard) og SERIALIZABLE ++

http://www.oracle.com/technetwork/testcontent/o65asktom-082389.html

  - Java DB støtter alle nivåene
    - READ COMMITTED er standard.
  - MySQL (InnoDB) støtter alle nivå
    - REPEATABLE READ er standard
- **Dersom du trenger å bruke et høyt isolasjonsnivå i en transaksjon, *husk* å sette tilbake til standard etter bruk.**
- JDBC, interface java.sql.Connection
  - int getTransactionIsolation()
  - void setTransactionIsolation(int level)

JDBC isolering m.m

JDBC, interface java.sql.Connection

- int getTransactionIsolation()
- void setTransactionIsolation(int level) -> **se neste foil ulike levels**

Connection con = ...;

Statement stmt = con.createStatement();

boolean isOk = stmt.**execute**(«LOCK TABLES konto READ»); //lås på konto-tabellen

Java DB, navn på isolasjonsnivå

Table 1. Mapping of JDBC transaction isolation levels to Derby isolation levels
http://db.apache.org/derby/docs/10.5/devguide/

Isolation levels for JDBC	Isolation levels for SQL
Connection.TRANSACTION_READ_UNCOMMITTED (ANSI level 0)	UR, DIRTY READ, READ UNCOMMITTED
Connection.TRANSACTION_READ_COMMITTED (ANSI level 1)	CS, CURSOR STABILITY, READ COMMITTED
Connection.TRANSACTION_REPEATABLE_READ (ANSI level 2) setter delt lås på intervall omfattet av where-uttrykk i select-setning, men forhindrer ikke insert. (krever søk på index, ellers låses hele tabellen)	RS
Connection.TRANSACTION_SERIALIZABLE (ANSI level 3) setter delt lås på intervall omfattet av where-uttrykk i select-setning. (krever søk på index, ellers låses hele tabellen)	RR, REPEATABLE READ, SERIALIZABLE

Flerbrukerproblemer som kan oppstå ved transaksjonsutføring

- Problemene kan klassifiseres i tre grupper
  - Tapt oppdatering, dvs. at en oppdatering blir skrevet over av en annen oppdatering
  - Ikke-bekreftede data, dvs. at en transaksjon får se data fra en annen transaksjon før disse er bekreftet
  - Inkonsistent uthenting av data, dvs. at en transaksjon leser data som er delvis oppdatert av en annen transaksjon
- Som oftest løses problemene ved at dataene som oppdateres reserveres (”låses”) for den transaksjonen som først begynte å arbeide med dem.
- Forsøker i det lengste å unngå å sette isolasjonsnivå SERIALIZABLE
- Husk også å vurdere om ”optimistisk låsing/utførelse” kan løse problemet
  - registrere nytt eksemplar med nr 1 større enn hittil brukte (select + insert)
    - Select max()
    - Insert into (maksnr + 1)
    - Hvis sqlexception (primærnøkkelfeil), prøv på nytt et begrenset antall ganger
  - MySQL
    - CREATE Table (ID int NOT NULL **AUTO_INCREMENT,** ....

Utprøving av flerbrukerproblematikken

- Testdata

```
DROP TABLE konto;
CREATE TABLE konto(
   kontonr INTEGER PRIMARY KEY,
   saldo INTEGER NOT NULL);
INSERT INTO konto(kontonr, saldo) VALUES(1, 35);
INSERT INTO konto(kontonr, saldo) VALUES(2, 120);
INSERT INTO konto(kontonr, saldo) VALUES(3, 100);
INSERT INTO konto(kontonr, saldo) VALUES(4, 35);
INSERT INTO konto(kontonr, saldo) VALUES(5, 100);
INSERT INTO konto(kontonr, saldo) VALUES(6, 30);
```

Linux; Kommandobasert grensesnitt

- Merk! Må bruke konsoll/kommandobasert grensesnitt!
- Linux kan bruke innebygget mysql-klient
  - eller MySQL Shell som beskrevet under
- Installeres slik (Ubuntu):

  sudo apt install mysql-client

- Eksempel på bruk

  mysql -h mysql.stud.idi.ntnu.no -u <brukernavn> -p
  - Passord må så oppgis (grunnet -p)

Alle OS; Kommandobasert grensesnitt

MySQL Shell

- Gå til https://dev.mysql.com/downloads/shell/
- Velg operativsystem
- Godta «Recommended Download»
  - (kun denne funker for Windows)
- Godta det som trengs for å installere MySQL Shell
- Merk at det kan ta litt tid å laste alt som trengs, vær litt tålmodig

- Start opp MySQL Shell fra startmenyen og skriv
  - **\connect <brukernavn>@mysql.stud.idi.ntnu.no**
- Når oppkobling er gjort så må vi endre fra JavaScript-modus til sql, skriv
  - **\sql**
- Velg så din database ved å skrive
  - **\use <brukernavn>**
- Pil opp henter forrige kommando

Kommandoer, syntaks og dokumentasjon

Transaksjoner: https://dev.mysql.com/doc/refman/5.7/en/commit.html

- START TRANSACTION
- Hva er alternativene? Bruk dokumentasjonen.

Isolasjonsnivå: https://dev.mysql.com/doc/refman/5.7/en/set-transaction.html

- SET **SESSION** TRANSACTION ISOLATION LEVEL SERIALIZABLE;
- session brukes for å slippe å angi isolasjon for hver transaksjon!

Låsing direkte: https://dev.mysql.com/doc/refman/5.7/en/lock-tables.html

- se dokumentasjonen og finn ut hvordan man låser og låser opp
- hint: dokumentajonen inneholder gjerne eksempler

Flerbrukerproblem A: Tapt oppdatering
Ikke leseslås

Tid	Transaksjon A	Verdi som A ser	Transaksjon B	Verdi som B ser
1	**select saldo from konto where kontonr = 1**	**35**		
2			**select saldo from konto where kontonr = 1**	**35**
3	***tar ut 30 kr*** **update konto set saldo = 5 where kontonr = 1**	**5**		
4			***tar ut 20 kr*** **update konto set saldo = 15 where kontonr = 1**	**15**
5	**commit**			
6			**commit**	

Merk at B tar beslutning om å endre saldo på feil grunnlag.
Hvilke resultater kan vi få ved seriell utførelse av transaksjonene?
Løser standard isolasjonsnivå (READ COMMITTED) problemene?
Alternativer?

Flerbrukerproblem B: Ikke-bekreftede data
READ UNCOMMITED

Tid	Transaksjon A	Verdi som A ser	Transaksjon B	Verdi som B ser
1	select saldo from konto where kontonr = 1	35		
2	*setter inn 65 kr* update konto set saldo = 100 where kontonr = 1	100		
3			select saldo from konto where kontonr = 1	100 (”dirty read”)
4	rollback			
5			*tar ut 50 kr* update konto set saldo = 50 where kontonr = 1	50
6			commit	

Merk at B tar beslutning om å endre kvantum på feil grunnlag.
Resultater ved seriell utføring:
Løser «standard» isolasjonsnivå (READ COMMITTED) problemene?

Flerbrukerproblem C: Inkonsistent uthenting av data
READ UNCOMMITTED

Tid	Transaksjon A	Verdier som A ser	B (overføring av penger fra en konto til en annen)
1	select sum(saldo) from konto;	420	
2			update konto set saldo = saldo - 20 where kontonr = 3
3	select sum(saldo) from konto;	400	
4			update konto set saldo = saldo + 20 where kontonr = 2
5			commit

Resultater ved seriell utføring?

Løser isolasjonsnivå (READ COMMITTED) problemene?