Oppgaver

Velg en brukers MySQL-konto som dere kobler opp mot.

Opprett innhold i database (evt. slett gammelt - delete from konto).
> Kjør create table skript fra foiler.

Manuell bruk av skrive- og leselåser

For de som har lyst å prøve å sette lese- og skrivelåser manuelt. Her vil det settes låser på hele tabeller. Merk at man normalt heller (ihvertfall når man programmerer) vil bruke transaksjoner med ønsket isolering (se neste «kapittel»).

Her må man være to stykker.
Kjør manuell skrive- og leselåser. For syntaks på manuell låsing se:
https://dev.mysql.com/doc/refman/5.7/en/lock-tables.html

Hvis en klient blir hengede i denne oppgaven, trykk ctrl-c for å kunne skrive nye kommandoer. Merk også at manuell låsing (som på lenken over) låser hele tabellen!

Start med at en person sørger for å sette leselås på konto-tabellen.

- Prøv å kjøre «SELECT * from konto;» i begge klientene. Hva skjer?
- Er det mulig å sette leselås også i den siste klienten?
- Fjern leselåsen i den ene klienten og kjør i begge klienter
- «UPDATE konto SET saldo=0 WHERE kontonr=1;» ?
    - Forklar resultatet.

Fjern låsene!
Start med at en person/klient sørger for å sett skrivelås på konto-tabellen.

- Prøv å kjøre «SELECT * from konto;» i begge klientene. Hva skjer?
- Lås opp tabellen og sett skrivelås på nytt.
- Er det mulig å sette leselås i den siste klienten?
    - hvis ikke (dvs. at den henger) avbryt med ctrl-c for å kunne skrive nye kommandoer
- Er det mulig å kjøre «UPDATE konto SET saldo=0 WHERE kontonr=1;» ?
    - Sjekk i begge klienter og forklar svaret.

Transaksjoner

Bruk MySQL-dokumentasjonen til å finne ut hvordan man starter og avslutter en transaksjon. For å hjelpe litt på vei så se på denne siden: https://dev.mysql.com/doc/refman/5.7/en/commit.html

Sjekk innhold på kontoer, start en transaksjon og oppdater en konto. Sjekk at commit og rollback gir ønskede resultater (her må du starte transaksjonen to ganger, en gang avslutt med commit og en gang med rollback).

Teori

Hvilke typer låser har databasesystemene?

Hva er grunnen til at at man gjerne ønsker lavere isolasjonsnivå enn SERIALIZABLE?

Hva skjer om to pågående transaksjoner med isolasjonsnivå serializable prøver select sum(saldo) from konto?

Hva er to-fase-låsing?

Hvilke typer samtidighetsproblemer (de har egne navn) kan man få ved ulike isolasjonsnivåer?

Hva er optimistisk låsing/utførelse? Hva kan grunnen til å bruke dette være?

Hvorfor kan det være dumt med lange transaksjoner (som tar lang tid)? Vil det være lurt å ha en transaksjon hvor det kreves input fra bruker?

Oppgaver om transaksjoner

Når kommando for Klient 1 og Klient 2 er satt på samme linje så spiller det ingen rolle i hvilken rekkefølge de utføres.

Oppgave 1

Tid	Klient 1	Klient 2
1	Sett isolasjonsnivå til **read uncommitted**.	Sett isolasjonsnivå til **serializable**.
2	Start transaksjon.	Start transaksjon.
3		select * from konto where kontonr=1;
4	select * from konto where kontonr=1;	
5	update konto set saldo=1 where kontonr=1;	
6		commit;
7	commit;	

Hva skjer og hvorfor?
Hva hadde skjedd om Klient 2 hadde brukt et annet isolasjonsnivå?

Svar:
Klient 1 stopper på linje 5 pga. leselås fra Klient 2.
Hvis klient 2 bruker et annet isolasjonsnivå så blir ikke leselås satt og Klient 1 kan kjøreferdig.

Oppgave 2

a)

Tid	Klient 1	Klient 2
1	Sett isolasjonsnivå til **read uncommitted**.	Sett isolasjonsnivå til **read uncommitted**.
2	Start transaksjon	Start transaksjon
3	update konto set saldo=1 where kontonr=1;	
4		update konto set saldo=2 where kontonr=1;
5	update konto set saldo=1 where kontonr=2;	
6	**commit;**	update konto set saldo=2 where kontonr=2;

7		**commit;**

Hva blir resultatet? Hvorfor må det være slik?

Svar: Klient 2 utfører sist og de oppdateringene blir lagret. Ettersom det kun er oppdateringer vil isolasjonsnivå ikke bety noe.

b)

Her vil Klient 2 utføre sine update-setninger i motsatt rekkefølge av over!

Tid	Klient 1	Klient 2
1	Sett isolasjonsnivå til **read uncommitted**.	Sett isolasjonsnivå til **read uncommitted**.
2	Start transaksjon	Start transaksjon
3	update konto set saldo=1 where kontonr=1;	
4		update konto set saldo=2 where kontonr=2;
5	update konto set saldo=1 where kontonr=2;	
6		update konto set saldo=2 where kontonr=1;
7		

Hva blir resultatet? Forklar forskjellen fra oppgave a).
Vil det ha noe å si om man endrer isolasjonsnivå på klientene?

Svar:
Deadlock. Klientene må vente på at den andre transaksjonen skal låse opp for å komme videre.
Isolasjonsnivå er uten betydning. Påvirker bare leselåser.

Oppgave 3

Tid	Klient 1	Klient 2
1	Sett isolasjonsnivå til **read uncommitted**.	Sett isolasjonsnivå til **serializable**.
2	Start transaksjon.	Start transaksjon.
3	select sum(saldo) from konto;	
4		update konto set saldo=saldo + 10 where kontonr=1;
5	select sum(saldo) from konto;	
6		commit;
7	select sum(saldo) from konto;	
8	commit;	

Hva skjer?

Svar: Denne oppgaven viser at selv ved serializable så vil ikke det påvirke lesing i klienten som har read uncommitted.

Hva vil skje om Klient 1 bruker read committed, repeatable read eller serializable?

Svar: Ved serializable vil ikke update i Klient 2 gå før leselås er sluppet av Klient 1 (transaksjon avsluttet).
READ COMMITTED og REPEATABLE READ vil vise det samme på linje 5, men forskjellig på linje 7.

Oppgave 4

Lag en kjøring med to klienter som tester phantom reads. Her kan det være lurt å tenke igjennom isolasjonsnivå. Om resultatet ikke er som forventet så kan det være lurt å sjekke dokumentasjonen.

Svar: InnoDB sikrer mot phantoms i repeatable read! Dog ikke ved read committed;

Oppgave 5 - Frivillig

I denne oppgaven skal det lages en løsning for budrunder. Løsningen skal fungere slik at som kan være interessert i å legge inn bud først får se hva gjeldende bud er på. Det er så mulig å legge inn et høyere bud om man ønsker det. Budet gis da som differansen fra gjeldende bud (altså hvor mye høyere det blir).

Databasen som lagrer dette skal lages som følger. Det skal være en tabell Bud som inneholder det siste budet, altså beløpet samt navnet på personen som har lagt inn budet. Man skal altså ikke ha historikk over alle bud (la oss si at dette er mot en eller annen lov), mao. vil det bare være en rad i denne tabellen (vi har kun en budrunde gående for å gjøre det enkelt).

Det som er viktig å tenke på når man skal lage denne løsningen er at man kan ha flere samtidige klienter. De bør alle kunne lese og prøve å oppdatere nåværende bud uten å måtte vente på at andre personer skal gjøre seg ferdig med sitt bud, altså vil det å bruke transaksjoner med isolasjonsnivå SERIALIZABLE være uaktuelt. Det er også viktig at man ikke får gjennomføre et bud basert på feil forutsetninger. Mao. om noen klienter oppdaterer budet, etter at du har lest ut gjeldende bud så kan ikke budet ditt gjennomføres uten at du får vite at budet er endret/høyere enn det du har lest ut.

Tips til løsning: Optimistisk låsing/utførelse er nok den beste løsningen her. Tenk igjennom hvordan man får til dette. Prøv med to/flere klienter, hvor alle leser ut nåværende bud og prøver å kjøre en oppdatering - uten å vite hva andre byr.

Det mest naturlige er å lage Java-klienter i dette tilfellet, men det vil også være mulig å simulere det samme direkte med MySQL-klienter. Man trenger da en select for å hente ut nåværende bud, samt en update for å legge inn et nytt. Merk at denne update-setningen bare må gjennomføres hvis forutsetningene er riktige (mao. at ingen har oppdatert budet etter at du leste det ut).