NTNU
Kunnskap for en bedre verden

Øving: Normalisering og Transaksjoner (obligatorisk)

NTNU
Innleveringsfrist: se Blackboard

Løsningsforslag legges ut i etterkant.

Alle obligatoriske øvinger må være godkjente for å få karakter i emnet.

DEL 1: Normalisering

Et firma som leier ut feriehus lagrer følgende informasjon knytet til utleie:

kundedata: navn, adresse, telefon
data om eiendommen: adresse
data om eieren: navn, adresse, telefon
data om utleieforholdet: fra og med uke, til og med uke, pris pr uke

Gitt at vi har én tabell med 13 attributter (kolonner) som ser altså slik ut (skrevet på relasjonell form uten nøkler):

```
leieforhold (kunde_id, kunde_navn, kunde_adresse, kunde_tlf, eiendom_id,
eiendom_adresse, eier_id, eier_navn, eier_adresse, eier_tlf, fra_uke,
til_uke, pris)
```

I tillegg til dataene nevnt innledningsvis er det lagt inn kunde_id, eier_id og en eiendoms_id, som entydig identifiserer henholdsvis en person og en eiendom.

Foreslå kandidatnøkler for denne tabellen. Anta at en person kun kan leie en eiendom av gangen, og at en eiendom kan leies ut til kun en person av gangen.

En kandidatnøkkel må være en minimal mengde attributter som unikt identifiserer hver rad. Siden eier_id og eiendom_id entydig identifiserer en person og en eiendom kan vi anta at eiendom_id viser til eier_id. I tillegg kan en person leie en eiendom flere ganger så for å identifisere perioden av leieforholdet burde vi også inkludere fra_uke og til_uke. Siden en eiendom ikke kan leies ut til flere samtidig vil perioden hvor en eiendom leies ut være unik som vil være nødvendig for å finne prisen. Med dette vil kandidatnøkkelen være (kunde_id, eiendom_id, fra_uke, til_uke).

Tabellen er ikke problemfri mht til registrering og sletting av data. Forklar hvorfor.

For det første er det redundans i tabellen. Dersom et nytt leieforhold skapes vil det bli duplikater av verdiene. For eksempel, hvis en kunde ønsker å leie flere ganger vil informasjonen om kunden bli duplisert. For det andre er det anomalier ved sletting. Hvis vi sletter et leieforhold kan informasjon om eiendom eller kunden

forsvinne helt. Til slutt er det anomalier ved oppdatering. Altså, ved endring av adresser, navn eller telefonnummer må det endres flere ganger. Det store problemet med tabellen er egentlig at alt er i én tabell når informasjonen burde deles inn i flere tableller.

Sett opp funksjonelle avhengigheter mellom alle attributtene.

Vi kan identifisere funksjonelle avhengigheter med kandidatnøkklene (kunde_id, eiendom_id, fra_uke, til_uke). Kunde_id peke vise til kunde_navn, kunde_adresse og kunde_tlf. Eiendom_id vil vise til eiendom_adresse og eier_id. Ved bruk av eier_id, som er fått fra eiendom_id, får vi eier_navn, eier_adresse og eier_tlf. Ved bruk av alle kandidatnøkklene (kunde_id, eiendom_id, fra_uke, til_uke) får vi prisen.

Du skal nå bruke funksjonelle avhengigheter og BCNF til å foreslå en oppsplitting av tabellen i mindre tabeller slik at problemene vedr. registrering og sletting av data unngås.

Sett opp, direkte fra de funksjonelle avhengighetene, relasjoner som er på BCNF.

Vi kan dele opp leieforhold tabellen til relasjons-tabeller. For kunden burde vi ha en kunde-tabell med kunde_id, kunde_navn, kunde_adresse og kunde_tlf. For eiendom burde vi ha en eiendoms-tabell med eiendom_id, eiendom_adresse og eier_id. For eier burde vi ha en eier-tabell med eier_id, eier_navn, eier_adresse og eier_tlf. Til slutt burde vi ha en tabell for utleieforhold med kunde_id, eiendom_id, fra_uke, til_uke og pris.

Kan vi løse denne oppgaven ved å gjennomføre prosessen 1NF --> 2NF --> 3NF? Begrunn svaret.

Tabellen er i 1NF fordi alle attributtene har enkelte verdier uten gjentakelse. For å oppnå 2NF og 3NF må vi fjerne partielle avhengigheter og transitive avhengigheter. Ved å dele inn tabellen i ulike tabeller kunde, eiendom, eier og leieforhold oppnår vi 2NF og 3NF. Dette er fordi attributter i tabellene kun blir avhengig av attributter som er del av en primærnøkkel. I tillegg vil ikke attributter avhenge av attributter som er avhengig av primærnøkkelen som transitive avhengigheter. Et eksempel på transitive avhengigheter er eksempel tabellen (eier_id, eier_navn, eiendom_id, eiendom_adresse) hvor eier_eiendom_adresse blir avhengig av eiendom_id som er avhengig av eier_id. Her er det smartere å dele opp til to tabeller.

DEL 2: Transaksjoner

I denne øvingen forventes det at man har gjennomført delen ' Oppgaver om transaksjoner' til slutt i oppgavesettet (resten er beskrevet som frivillig). Merk deg at det ikke er mulig å skrive transaksjoner i phpMyAdmin. Du må bruke MySQL console eller MySQL Shell (se foilsett).

Gjennomføring:

Denne øvingen er i utgangspunktet tenkt utført i grupper av to personer, om man ikke har noen å jobbe sammen med kan man bruke to konsoller (mot databasen) på samme maskin.

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

Opprett innhold i database (evt. slett gammelt - delete from konto).

> Kjør konto.sql eller create table skript fra foiler.

Frivillig - 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 (eller bruke to konsoller på en maskin). Kjør manuell skrive- og leselåser. For syntaks på manuell låsing se: https:/dev.mysql.com/doc/refman/8.0/en/lock-tables.html

Hvis en klient blir hengende 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 følgende i begge:
- «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/8.0/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).

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?

Klient 2 kjører i serializable, som betyr at den ser en konsistent og isolert versjon av databasen. Klient 1 kjører read uncommitted som betyr at den kan lese ikke-committed data fra andre transaksjoner. Hvis du kjører kommandoene fra klientene vil Klient 1 bli fryst på tid 5. Dette er fordi Klient 2, som er serializable, hindrer den i å fortsette. Hvis Klient 2 hadde vært read committed ville Klient 1 kunne fullføre sine operasjoner. Dersom Klient 2 hadde hatt en ny operasjon etter Klient 1 gjorde endringene ville den ha måtte vente på at Klient 1 committer før den kan lese av endringene. Dersom Klient 2 hadde vært uncommitted så ville Klient 2 kunne lese av endringene selv før Klient 1 committer dem men som «dirty» (ikke-committed) data. Hvis Klient 2 hadde brukt repeatable read ville Klient 2 kunne ha lest av endringene i Klient 1 etter Klient 1 og Klient 2 committer endringene sine. Her fungerer egentlig mange isolasjonsnivåer.

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?

Klient 1 og Klient 2 setter isolasjonsnivå til read uncommitted som betyr at de kan lese ikke-committed data fra andre transaksjoner. Hvis du kjører kommandoene vil det være mulig for Klient 1 og Klient 2 å gjøre endringer og lese endringene fra hverandre. Problemet oppstår på tid 4 hvor Klient 2 prøver å gjøre endringer til saldo på konto 1 som Klient 1 allerede fra tid 3 gjorde endringer på. Klient 2 vil fryse uten å fullføre endringen fordi raden blir låst. Dette kan nok være fordi endringene fra Klient 1 ikke enda er blitt committed enda. Selv om Klient 2 kan lese dem er de Klient 1 litt isolert.

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).

Resultatet blir ganske likt som i forrige deloppgave. Forskjellen er at Klient 2 i tid 4 ikke prøver å gjøre endringer der Klient 1 allerede har gjort endringer. Altså på tid 3 blir konto 1 endret og på tid 4 blir konto 2 endret. Problemet oppstår på tid 5 hvor Klient 1 prøver å gjøre endringer til konto 2 som Klient 2 gjorde endringer på tidligere. Som i forrige deloppgave antar jeg at raden har blitt låst. Her må endringene bli committed før de kan endres på nytt.

Vil det ha noe å si om man endrer isolasjonsnivå på klientene?

Det som skjer er at MySQL bruker låsing av rader under en transaksjon. Når du oppdaterer rader i en transaksjon blir de låst til transaksjonen fram til de er committed eller rullet tilbake med rollback. Når to transaksjoner prøver å oppdatere de samme radene kan det føre til en deadlock eller at en transaksjon må vente på at den andre transaksjonen committer før den fortsetter. På grunn av dette vil det ikke ha så mye å si hvilke isolasjonsnivå som klientene bruker siden du uansett får det samme problemet.

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?

Klient 1 setter isolasjonsnivå til read uncommitted som betyr at den kan lese ikke-committed data fra andre transaksjoner. Klient 2 kjører i serializable, som betyr at den ser en konsistent og isolert versjon av databasen. Resultatet på Klient 1 sin side vil være 420, 430 og 430. Forskjellen fra 420 og 430 er at Klient 2

i tid 4 økte saldo på en konto med 10. Den andre forskjellen mellom 430 og den andre 430 er ikke synlig på Klient 1 med select men den første selecten i tid 5 er «dirty» (ikke-committed) mens den andre er committed.

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

Hvis Klient 1 bruker read committed vil det bli vist 420, 420 og 430. Forskjellen fra read uncommitted er at den ikke viser dirty data. Klient 1 vil altså ikke vise den oppdaterte verdien før Klient 2 har committed endringene. Hvis Klient 1 er repeatable read vil ikke verdien i kontoen endres gjennom hele transaksjonen. Etter du har brukt commit på Klient 1 og den er ute fra transaksjonen så kan du lese de endrede verdiene gjort av Klient 2. Hvis Klient 1 hadde vært serializable ville ikke Klient 2 kunne ha utført endringene i tid 4 fordi det kommer i konflikt med Klient 1.

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.

En phantom read oppstår når en transaksjon leser et sett med rader flere ganger men mengden rader endres av en annen transaksjon før den første transaksjonen er ferdig. Dette kan skje når en ny rad blir inserted eller deleted i en annen transaksjon. I eksempelet mitt vil Klient 1 telle antall rader to ganger. Den første gangen viser den 7 men den andre ganger viser resultatet 8. Dette viser at nye rader har dukket opp i datasettet mellom to select-operasjoner innenfor samme transaksjon. Altså, en phantom read.