

10

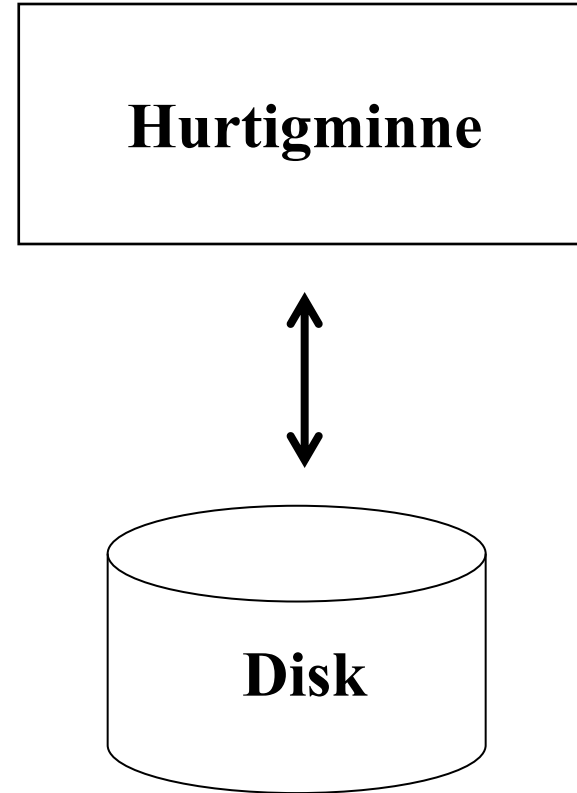
Transaksjoner

Læringsmål

- ☐ Forstå hva transaksjoner er, og hvilke egenskaper de bør ha.
- ☐ Kunne bekrefte og avbryte transaksjoner med SQL.
- ☐ Forstå hvilke utfordringer feilsituasjoner og samtidighet medfører for systemet med hensyn til transaksjoner.
- ☐ Forstå hvordan loggføring brukes for å håndtere feilsituasjoner.
- ☐ Forstå hvordan låser brukes for å håndtere samtidighetsproblemer.

Øyeblikksbilde av en database

- ❑ Data **lagres** på disk, men må leses inn i hurtigminnet for **beregning**.
- ❑ På et **bestemt tidspunkt** vil en del av databasen eksistere **kun i minnet**.
- ❑ Databasesystemer er i produksjon over lang tid.
- ❑ Må håndtere **feilsituasjoner**.



Hva er en «operasjon» ?

- ❑ SQL-setninger kan behandle **mange rader**:

UPDATE Ansatt

SET Lønn = Lønn * 1.1

- Lønnen til samtlige ansatte blir oppdatert.
- 1 SQL-setning kan altså utføre mange «operasjoner».

- ❑ Motsatt kan vi ønske å betrakte **flere SQL-setninger** som én «sammensatt» operasjon.

- Eksempel fra Hobbyhuset: Ordrebestilling krever innsetting av rader i Ordre og Ordrelinje, samt oppdatering av Vare.

Transaksjonsbegrepet

- ❑ En **transaksjon** er en logisk operasjon mot databasen.
 - Kan involvere en eller flere tabeller.
 - Kan bestå av en eller flere SQL-setninger.

- ❑ Transaksjoner kan avsluttes på to måter:
 - **COMMIT**: bekreft, endringene gjøres permanente
 - **ROLLBACK**: angre, transaksjonen «rulles tilbake»

- ❑ Utfordringer:
 - **Feil** - for eksempel diskkrasj og strømbrudd
 - **Samtidige brukere** – som kan ødelegge for hverandre

Transaksjoner i SQL

- ❑ Ansatt 22 mottar bestilling av 5 enheter av produkt 4034 fra kunde 1003 - og oppretter ordre 2020:

```
INSERT INTO Ordre(OrdreNr,KNr,AnsNr)
```

```
VALUES (2020,1003,22);
```

```
INSERT INTO Ordrelinje(OrdreNr,VNr,Antall)
```

```
VALUES (2020,'4034',5);
```

```
UPDATE Vare
```

```
SET Antall = Antall-5
```

```
WHERE VareNr='4034';
```

```
COMMIT;
```

- ❑ Etter den første/de to første SQL-setningene er ikke databasen i en lovlig tilstand.

Transaksjoner i MySQL

❑ **Auto-commit** er standard. Kan skrus av:

```
SET autocommit = 0;
```

❑ Eller start **eksplisitt** med BEGIN:

```
START TRANSACTION
```

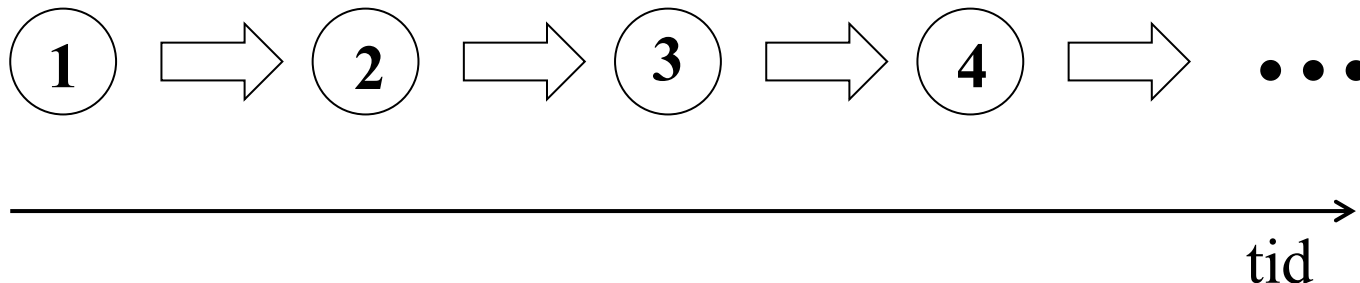
```
INSERT INTO Ordre(OrdreNr,KNr,AnsNr)  
VALUES (2020,1003,22);
```

```
-- Flere kommandoer ...
```

```
COMMIT;
```

Transaksjoner og tilstander

- ❑ En **tilstand** er innholdet i databasen på et bestemt tidspunkt.
- ❑ En **transaksjon** bringer databasen fra én tilstand til en annen.
- ❑ Hvis vi tenker oss at kun én transaksjon blir utført av gangen, kan **livsløpet** til en database visualiseres slik:



ACID-egenskapene

Atomicity: Alle eller ingen av deloperasjonene i en transaksjon må fullføres.

Consistency: En transaksjon fører databasen fra en lovlig tilstand til en annen lovlig tilstand.

Isolation: Effekt av transaksjoner under utførelse skal ikke kunne observeres av andre transaksjoner.

Durability: Effekt av fullførte (committed) transaksjoner lagres i databasen og skal ikke gå tapt på grunn av feil.

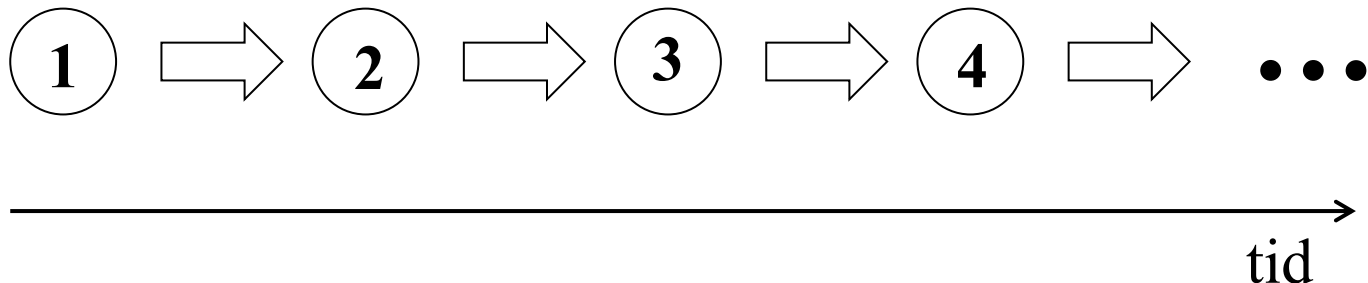
Transaksjonsloggen

TransID	Table	RowID	Attribute	Before	After
101	*** BEGINTRANS				
101	VARE	102375	ANTALL	112	113
101	VARE	102542	ANTALL	56	66
101	*** COMMIT				

- ☐ Alt registreres i loggen før databasen blir oppdatert.
- ☐ Når **COMMIT** er skrevet til loggen, er beslutningen tatt.
- ☐ Loggen kan brukes for å oppdatere databasen.
- ☐ **ROLLBACK**: Eventuelle oppdateringer av databasen blir «rullet tilbake».
- ☐ Loggen brukes også ved gjenoppretting etter feil.

Transaksjoner og tilstander

- ❑ En **tilstand** er innholdet i databasen på et bestemt tidspunkt.
- ❑ En **transaksjon** bringer databasen fra én tilstand til en annen.
- ❑ Hvis vi tenker oss at kun én transaksjon blir utført av gangen, kan **livsløpet** til en database visualiseres slik:



Samtidighetskontroll

❑ Ønsker **flere samtidige brukere / transaksjoner**:

- Utnytte parallellitet (CPU og I/O)
- Redusere gjennomsnittlig ventetid (lange transaksjoner)

_____ 1 av gangen

_____ Flere samtidig

❑ Når er det trygt å tillate samtidig aksess?

- Mange brukere kan **lese** samme data samtidig
- To brukere kan skrive samtidig kun hvis de jobber med **forskjellige** deler av databasen.

Les-Beregn-Skriv

- ❑ Hva skjer når en «celle» på ytre lager skal oppdateres?

UPDATE Ansatt

SET Lønn = Lønn * 1.1

WHERE AnsNr = 14

- ❑ Transaksjonen består av følgende deloperasjoner:

1. **Les** post med AnsNr = 14 fra ytre lager
2. **Beregn** ny lønn
3. **Skriv** resultat til ytre lager

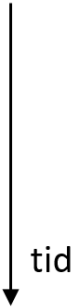
- ❑ Les-Beregn-Skriv er ikke en atomær operasjon.

- ❑ Det betyr at to oppdateringer kan «flettes» i tid !

Tapt oppdatering

- ❑ Samtidighetsproblemer kan oppstå når to transaksjoner jobber med **samme data til samme tid**.
- ❑ Vi studerer hva som skjer når to transaksjoner skal oppdatere en verdi A på disken (en verdi i en bestemt kolonne i en bestemt rad i en tabell).

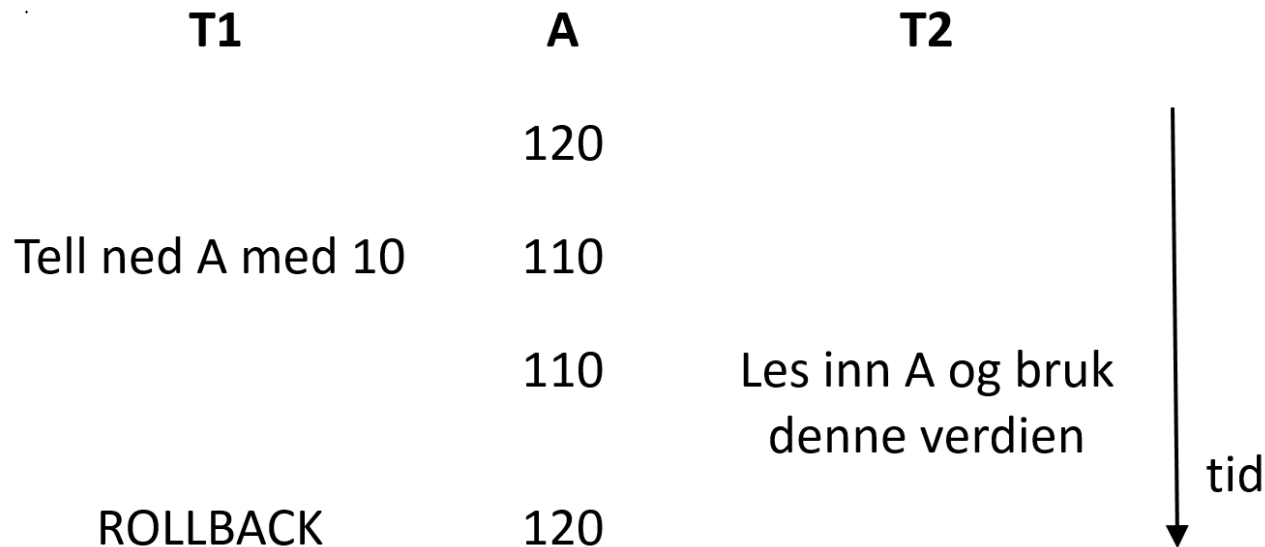
Lokal kopi T1	T1	Verdi A på disk	T2	Lokal kopi T2
120	Les inn	120		
110	Tell ned med 10	120	Les inn	120
110	Skriv til disk	110	Øk med 20	140
		140	Skriv til disk	140



- ❑ Oppdateringen til transaksjon T1 blir skrevet over av transaksjon T2 og går tapt. Hvis $A=120$ ved start, så får vi her $A=140$ ved slutt. Korrekt resultat er $A=130$ ($120+20-10$).

Angret oppdatering


- ❑ Transaksjon T2 bygger på et resultat fra transaksjon T1 som transaksjon T1 seinere «angrer» på at den produserte.



- ❑ Transaksjoner må ikke få lov til å avlese andres «mellomresultater».

Inkonsistent analyse

❑ Transaksjon T2 produserer en rapport basert på noen «gamle» og noen «nye» resultater.

T1	A	B	T2	Lokal sum	
	10	50	Nullstill sum	0	 tid
	10	50	Legg til A i sum	10	
Øk A med 10	20	50		10	
Øk B med 20	20	70		10	
			Legg til B i sum	80	

❑ Selv 1 «leser» og 1 «skriver» kan altså gi samtidighetsproblemer !

Låser

- ❑ Generell løsning på samtidighetsproblemene:
Transaksjoner må vente på hverandre.
- ❑ En **lås** (lock) er en «ventemekanisme».
- ❑ En transaksjon kan låse større eller mindre deler:
 - hele databasen
 - en tabell
 - en rad i en tabell
 - en «celle» i en rad i en tabell
- ❑ Vi skiller mellom **skrivelåser** (exclusive locks) og **leselåser** (shared locks).

Løsning: Tapt oppdatering

Lokal kopi	T1	Verdi A på disk	T2	Lokal kopi
	Skrivelås på A	120		
120	Les inn	120	Skrivelås på A?	
110	Tell ned med 10	120	Vent	
110	Skriv til disk	110	Vent	
	Lås opp A	110	Vent	
		110	Les inn	110
		110	Øk med 20	130
		130	Skriv til disk	130
		130	Lås opp A	


tid

- ❑ Angret oppdatering og inkonsistent analyse lar seg løse med låser på tilsvarende måter.

Serialiserbarhet

- ❑ Et **forløp** er en «fletting» i tid av deloperasjonene til en samling transaksjoner.
 - Opplagt riktig: Kun 1 transaksjon av gangen = **sekvensiell forløp**.
 - **Men**: Det er mer effektivt med samtidige transaksjoner.
- ❑ Et **serialiserbart forløp** tillater samtidighet («fletting»), men har samme effekt som et sekvensielt forløp.

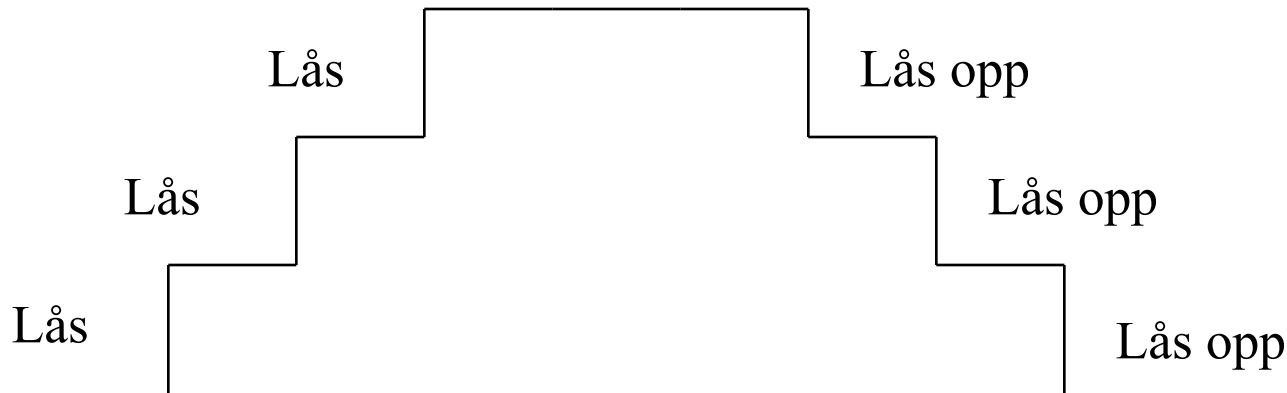
T1	A	B	T2
Skrivelås A	10	10	Skrivelås B
Tell ned A med 10	0	20	Øk B med 10 hvis større enn 0
Lås opp A	0	20	Lås opp B
Skrivelås B	0	20	Skrivelås A
Tell ned B med 10	0	10	Øk A med 10 hvis større enn 0
Lås opp B	0	10	Lås opp A



NB! Ikke alle forløp er serialiserbare selv om de bruker låser.

To-fase-låsing

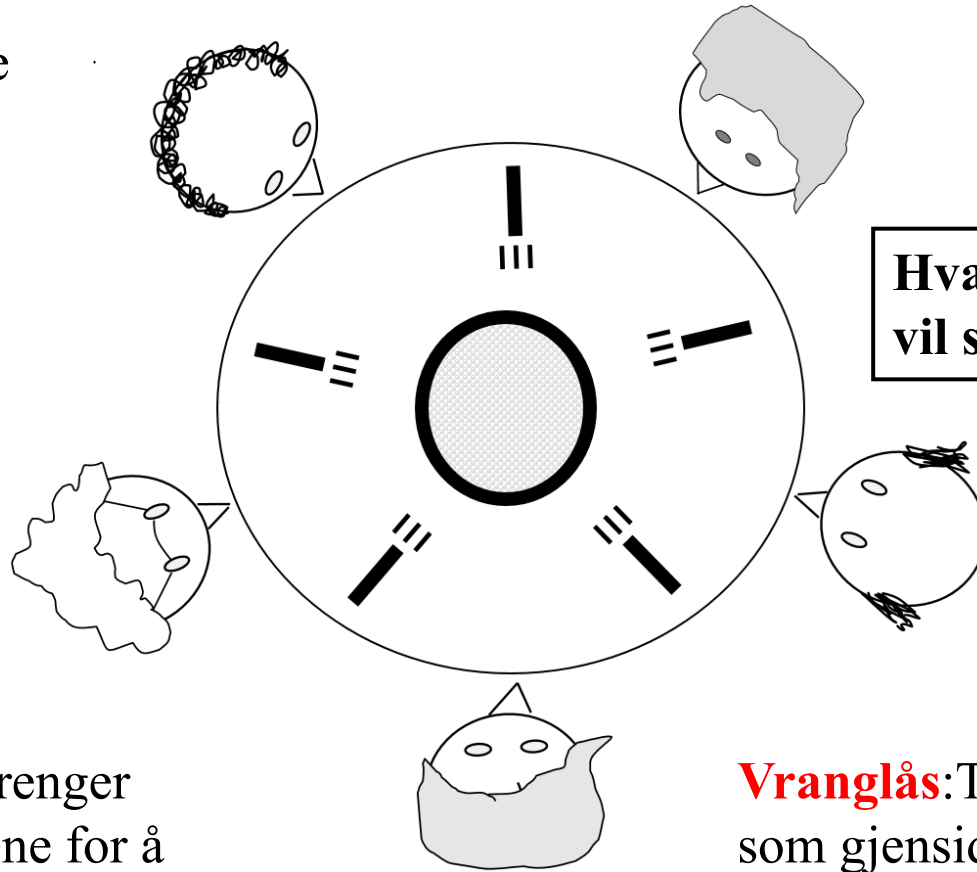
- En transaksjon følger reglene for **to-fase låsing** hvis alle låseoperasjoner gjøres før første frigivelse (opplåsing).
 - Det vil si, ingen låsing etter første opplåsing!



- **Følgende gjelder:** Hvis alle transaksjoner følger to-fase låsing vil ethvert forløp bli serialiserbart.

Vranglås: The Dining Philosophers

Filosofene
tenker og
spiser.



Hva skjer hvis alle
vil spise samtidig?

En filosof trenger
begge gaflene for å
spise.

Vranglås: Transaksjoner
som gjensidig venter på at
de andre skal frigi låser.

Håndtere vranglås

❑ Oppdage og «løse opp» vranglås

- Bygge opp en **ventegraf**: Hvis transaksjon T1 må vente på transaksjon T2, så legg til en kant (pil) fra T1 til T2.
- Av og til: Gå gjennom ventegrafen og sjekk om det har oppstått en **cykel**, for eksempel at T1 venter på T2 som venter på T3 som igjen venter på T1.
- Velg en av transaksjonene i cyklen. **Avbryt** transaksjonen («rull den tilbake»). Gjennomfør de andre, og start avbrutt transaksjon etterpå.

❑ Forhindre vranglås

- Alle transaksjoner gis et unikt **tidsstempel**.
- Hvis en transaksjon vil måtte vente på en **eldre** transaksjon blir den **avbrutt**. Det betyr at yngre transaksjoner aldri vil vente på eldre og dermed kan det ikke oppstå sykler.

Pessimistisk og optimistisk låsing

- ❑ **Pessimistisk** låsing = standard låsing.
- ❑ **Optimistisk** låsing:
 - Hensiktsmessig i systemer med få konflikter, for eksempel hvis de fleste kun leser, og i «interaktive» databaseapplikasjoner.
 - Transaksjoner blir utført uten restriksjoner fram til COMMIT, men skriver til lokal kopi.
 - Validering før lokal kopi blir skrevet til databasen.
- ❑ Kan velge mellom optimistisk/pessimistisk låsing i Access.

Oppsummering

- ❑ En transaksjon er en logisk operasjon.
 - Blir definert ved COMMIT og ROLLBACK.
 - Skal tilfredsstille ACID-egenskapene.
- ❑ DBHS må håndtere feil og samtidighet.
- ❑ Feil:
 - Hvilke transaksjoner ble avbrutt når?
 - Strømbrudd: Transaksjonslogg.
 - Diskkrasj: Sikkerhetskopi + transaksjonslogg.
- ❑ Samtidighet:
 - Transaksjoner må ikke få lov til å «forstyrre» hverandre.
 - DBHS bruker leselåser og skrivelåser.
 - Låser alene sikrer ikke et korrekt resultat.
 - Tofase-låsing sikrer serialiserbare forløp.
 - Kan fremdeles få vranglås.
 - Vranglås kan oppdages eller forhindres.