

## Lektion 15

Fredag den 4. april 2025

### Læsestof til denne lektion

[Kifer] side 455-456 og 458 – 471 (samme som sidst)  
Powerpoints om samtidighedskontrol

### Emner

Kort snak om opgave 19 og 20.

Transaktioner

- Lille repetition fra sidst
- Demo via scripts
- Deadlock
- AutoCommit, eksplicit og implicit
- Låsningsparametre
- Versionering
- Optimistisk låsning

### Opgaver

Opgave 21, 22 og 23

### Læsestof til næste lektion (mandag den 7. april 12:30 – 14:00):

[Ramakrishnan] kap 18 side 579 – 596 øverst

## Opgave 20

I denne opgave skal du modificere programmet fra opgave 19, så de anvender transaktioner.

Transaktioner kan startes fra Java på to måder. Den ene er via de metoder, der findes på Connection – i denne situation er det JDBC-driveren der sendes de faktiske kommandoer til databasen. Den anden måde er selv at sende kommandoerne direkte til SQL-server via execute kommandoen.

Herunder er vist begge muligheder (jeg antager, der et Connection-objekt *con* og et statement-objekt *stmt*).

Start af en transaktion:

Via JDBC:	<code>con.setAutoCommit(<b>false</b>)</code>
Direkte til SQL-server	<code>stmt.execute("begin tran")</code>

Rollback af en transaktion:

Via JDBC:	<code>con.rollback()</code>
Direkte til SQL-server	<code>stmt.execute("rollback tran")</code>

Commit af en transaktion:

Via JDBC:	<code>con.commit()</code>
Direkte til SQL-server	<code>stmt.execute("commit tran")</code>

Bemærk at hvis du siger commit/rollback direkte til SQL-server skal der være startet en transaktion ellers får du en fejl – via JDBC kan du derimod godt sige commit/rollback en gang for meget – den sidste commit/rollback gør så bare ingenting.

Jeg anbefaler, at du bruger JDBC-udgaverne.

Prøv at sætte start og afslutning af dine transaktioner ind i programmet fra opgave 15 og test igen!

Test derefter programmet, hvor dit program er startet to gange. For helt at kunne styre afviklingen er det nødvendigt at lave en tastaturindlæsning mellem den sidste select og den første update. Man kan ikke umiddelbart køre det samme program i Intelli J to gange, så man skal lige

- Gå ind i Run | Edit Configurations
- Tryk på modify options i øverste høje hjørne
- Set flueben ved Allow multiple instances

Er du skuffet over resultatet – Hvis svaret er ja virker det som det skal, i næste opgave reparerer vi problemet!

## Opgave 21

Denne opgave er en fortsættelse af opgave 20.

I denne opgave skal du arbejde med Isolation levels.

Da du testede dit program i opgave 20, opdagede du, at programmet ikke forhindrede samtidighedsproblemer. Dette skyldes, at der som default anvendes et meget lavt isolation level.

Der kan anvendes følgende isolation levels (de enkelte levels beskrives i kopierne side 465 - 466)

- READ UNCOMMITTED
- READ COMMITTED (default i SQL Server)
- REPEATABLE READ
- SERIALIZABLE

Man kan sætte isolation-levels på følgende måde (isolation-level sættes inden transaktionen starter)

Via JDBC:

```
con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE)
```

Direkte til SQL-server:

```
stmt.execute("set transaction isolation level serializable")
```

Prøv at sætte dit isolation-level til henholdsvis REPEATABLE READ og SERIALIZABLE og undersøg om programmet nu giver det rigtige resultat.

Isolation leves sættes inden transaktioner startes (dvs. inden begin tran)

## Opgave 22

Denne opgave er en fortsættelse af opgave 21.

I denne opgave skal du arbejde med parameterstyring af låsning.

Parameteren sættes i parentes efter tabelnavnet. F.eks.

```
Select * from konto (nolock)
```

Der kan sættes en del forskellige parametre (kaldet lock hints)

De nedenstående kan alle anvendes ved select-sætninger, mens NOLOCK, HOLDLOCK og TABLOCK ikke kan anvendes ved insert, update eller delete.

Blandt de anvendelige er:

NOLOCK:	sætter ikke låse og respekterer ikke låse.
UPDLOCK	sætter en update lås og holder til der er committed
XLOCK	sætter en eksklusiv lås og holder til der er committed
ROWLOCK	gennemtvinger, at der låses på recordniveau
PAGLOCK	gennemtvinger, at der låses på pageniveau
TABLOCK	gennemtvinger en shared lås på hele tabellen
TABLOCKX	gennemtvinger en eksklusiv lås på hele tabellen

Prøv at anvende disse i jeres programmer, så deadlock undgås.

Prøv endvidere at køre jeres Java-program samtidigt med at I laver transaktioner fra Management-tool.

## Opgave 23

Til denne opgave skal du igen anvende Konto-tabellen fra opgave 17.

### Opgave 23.1

Du skal nu fra Java lave et program (i en konsol-applikation) som kan ændre saldoen på en konto. Programmet skal ses som et eksempel på en U i CRUD.

Du kan måske med fordel bruge programmet fra opgave 19 som udgangspunkt og så rette/slette i det.

Programmet **skal** igen laves efter en bestemt skabelon (Husk at du skal bruge samme connection-objekt gennem hele programmet og at det ikke må closes før til slut).

- Indlæs et kontonummer fra bruger
- Vis alle oplysninger om den pågældende konto
- Indlæs en ny saldo fra bruger
- Opdater saldo på den pågældende konto til den værdi brugeren har indtastet

**Test at programmet virker – du må ikke gå videre, før programmet er testet.**

Sæt nu transaktioner på, så programmet virker med traditionel låsning.

### Opgave 23.2

Tag en kopi af programmet og lav det om til optimistisk låsning.

Tag en kopi af programmet fra opgave 23.1 og prøv at lave det med versionering.

For at kunne bruge snapshot i en database kontodb skal du udføre følgende kommando

```
alter database kontodb set allow_snapshot_isolation on
```

Mens du udfører kommandoen, må ikke have vinduer, der bruger kontodb.