

## Transaktioner

### 1. Transaktioner

### 2. Samtidighet ("concurrency") och lås

Kap. 17

### 3. Deadlock

## 5.1. Skapa transaktioner

- En transaktion påbörjas genom `BEGIN TRANSACTION` (eller om man är lat `BEGIN TRAN`)
- Som standard så kör SQL Server med *autocommit*
  - Varje SQL-fråga behandlas som en *enskild transaktion*
  - Enskilda transaktioner* som resulterar i *errors* blir *auto-rollback*
- `ROLLBACK` kan användas utan att en transaktion *explicit* har påbörjats med `BEGIN TRAN`
- `COMMIT` fungerar inte utan påbörjad transaktionen

### Syntax

- `BEGIN {TRAN|TRANSACTION}` -- Början på transaktionen
- `SAVE {TRAN|TRANSACTION} Punkt` -- Sparar en markering i transaktionen
- `COMMIT [TRAN|TRANSACTION]` -- Utför transaktionen
- `ROLLBACK [[TRAN|TRANSACTION] [,Punkt]]` -- Ångrar transaktionen

### Notera

- `TRAN` (eller `TRANSACTION`) nyckelordet är valfritt men rekommenderas för en tydligare syntax.

## 5. Transaktioner

- En *transaktion* är en grupp av frågor samlade till en *logisk enhet*
- Normalt är varje SQL-fråga en *transaktion*
  - Flera SQL-frågor kan dock grupperas till en *gemensam transaktion*
- Förändringar genom *funktionsfrågor* blir permanenta först efter *commit* på *transaktionen*
- Innan *commit* så kan alla ändringar "ångras" med *rollback* (= undo på allt sen starten)
  - Efter *commit* finns dock ingen *rollback*!

### Användningsområden:

- När flera *funktionsfrågor* påverkar samma data
- Vid uppdatering av *främmandenyckler*
- Vid flytt av poster mellan tabeller
- När *funktionsfrågor* är baserad på resultat av `SELECT`-frågor
- Vid SQL-frågor som kan orsaka *errors* som påverkar *dataintegriteten*

## 5.2. Exempel

- Lägger till tre poster i en *gemensam transaktion*:  

```

BEGIN TRAN          -- Börjar transaktionen

INSERT INTO Utbetalningar VALUE ( 3, 12500, 'Juli')
INSERT INTO Utbetalningar VALUE ( 3, 13200, 'Augusti')
INSERT INTO Utbetalningar VALUE ( 3, 8700, 'September')

COMMIT TRAN        -- Utför transaktionen (ändringarna blir permanenta)
            
```
- Av säkerhetsskäl så vill vi inte att en `DELETE`-fråga ska kunna ta bort mer än 2 poster:  

```

BEGIN TRAN
DELETE FROM Utbetalningar
WHERE AnställdID = 5

IF @@ROWCOUNT > 2
    ROLLBACK TRAN -- Ångrar transaktion
ELSE
    COMMIT TRAN   -- Permanenta ändringar
            
```

Utbetalningar

UtID	AnställdID*	Belopp	Månad
1	5	18700	Juli
2	5	1600	Juli
4	10	21200	Augusti
5	7	NULL	Juli
7	2	22500	Juli
8	7	19900	September
10	2	22500	Augusti
11	5	5600	September

### 5.3. Nästlade transaktioner (skräp!) och sparade markeringar (bättre)

- Nästlade transaktioner är transaktioner påbörjade inuti andra transaktioner
  - Systemfunktionen @@TRANCOUNT håller reda på nivån på transaktionerna
- Commit på undertransaktioner (nivå > 1) gör ingenting mer än att minska @@TRANCOUNT
- Rollback gäller för hela transaktionen (ignorerar nivån)
  - Genom en sparad markering (save point) så möjliggörs en delvis rollback

**Exempel**

- Nästlade transaktioner och dess avsaknad av permanent påverkan av tabeller:

```

BEGIN TRAN
DELETE Lärare
BEGIN TRAN -- Nästlad transaktion
DELETE Akademier
COMMIT TRAN -- Ändringar permanent?!
ROLLBACK TRAN -- Angrar transaktionen?
    
```

Lärare

Förnamn	Efternamn	Akademi
Andreas	Persson	3
Johan	Petersson	3
Kalle	Räisänen	3

Akademier

AkademID	Namn
3	Handelshögskolan
4	Akademien för Naturvetenskap och teknik

### 6. Samtidighet ("concurrency") och lås

- Samtidighet kan bli problem när olika transaktioner samtidigt ändrar/läser samma datakälla
  - Lösningen är att använda lås på datakällan
- SQL Server sköter automatiskt låsprocessen (om man säger till)
  - Dock kan det ibland behövas bättre lås!

**Fyra typer av problem med samtidighet**

- Förlorade uppdateringar** Då två transaktion utför ändringar samtidigt på samma tabell kan detta resultera i att den senare ändringen skriver över den första ändringen
- Dirty reads** En transaktion hämtar data som inte är commit av en annan transaktion
- Nonrepeatable reads** Samma SELECT-fråga ger olika resultat eftersom en annan transaktion har ändrat datakällan mellan SELECT-frågorna
- Fantomvärden** Medan en transaktion ändrar eller tar bort poster så lägger en annan transaktion till poster som då strider mot den första transaktionens ändringar

**Notera**

- Ett lås låser hela hierarkier i databasen.
  - Ex.: låses en tabell så låses även alla poster i tabellen: Tabell1 -> Post

### 6.1. Isoleringsnivå på transaktioner

- Genom en isoleringsnivå SET TRANSACTION ISOLATION LEVEL så kan avskildheten mellan transaktioner kontrolleras
  - Högre isoleringsnivå förhindrar concurrency problem men kräver mera systemresurser
  - Lägre isoleringsnivå accepterar vissa concurrency problem och ger bättre prestanda

**Syntax**

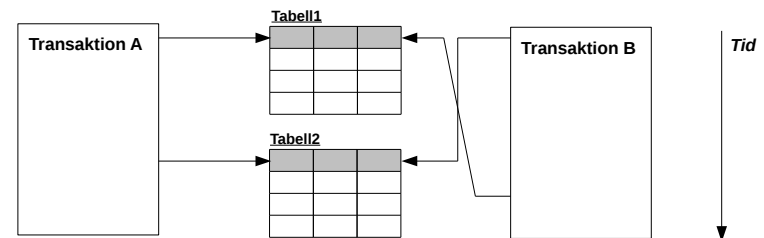
- SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SNAPSHOT | SERIALIZABLE}

Isoleringsnivå	Dirty reads	Lost updates	Nonrepeatable reads	Phantom reads
READ UNCOMMITTED	Accepterar	Accepterar	Accepterar	Accepterar
READ COMMITTED	Förhindrad	Accepterar	Accepterar	Accepterar
REPEATABLE READ	Förhindrad	Förhindrad	Förhindrad	Accepterar
SNAPSHOT	Förhindrad	Förhindrad	Förhindrad	Förhindrad
SERIALIZABLE	Förhindrad	Förhindrad	Förhindrad	Förhindrad

### 7. Deadlock

- Deadlock uppstår när två transaktioner inte kan COMMIT för att de har satt ett varsitt lås på en datakälla som den andra behöver
- SQL Server behandlar automatiskt deadlocks
  - Väljer en av transaktionerna medan den andra blir deadlock victim
- Det är dock bra att vet hur deadlocks kan förhindras!

**Exempel**



## 7.1. Tips för att förhindra deadlock

- Är det möjligt så använd en låg *isoleringsnivån*
  - *Default* är isoleringsnivån READ COMMITTED
- Hålla transaktionerna korta
  - Försök hålla SELECT -frågor utanför transaktionerna
  - Låt *aldrig* en transaktion vara öppen i vänta på en inmatning från användare!
- Vänta med större transaktioner till *strategiska tidpunkter*
  - *Ex.:* säkerligen få användare av databasen kl 23.59 på nyårsafton
- Utföra ändringar i samma ordning i transaktionerna:
  - *Dåligt ex.:*

<u>Transaktion A</u>	<u>Transaktion B</u>
UPDATE Tabell1	UPDATE Tabell2
...	...
UPDATE Tabell2	UPDATE Tabell1
  - *Bättre ex.:*

<u>Transaktion A</u>	<u>Transaktion B</u>
UPDATE Tabell1	UPDATE Tabell1
...	...
UPDATE Tabell2	UPDATE Tabell2