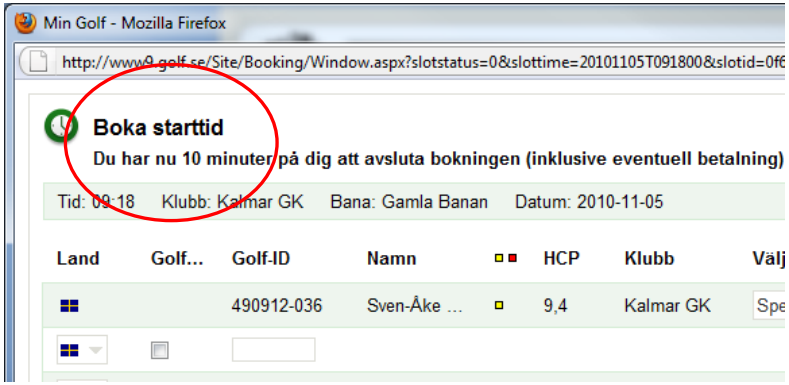
Innehåll Concurrency (samtidighet)

- Fleranvändarsystem
- Concurrency conflict
- Transaktion
- ACID
- Deadlock
- Lås / Locks
- Dynamic Locking
- Isolation

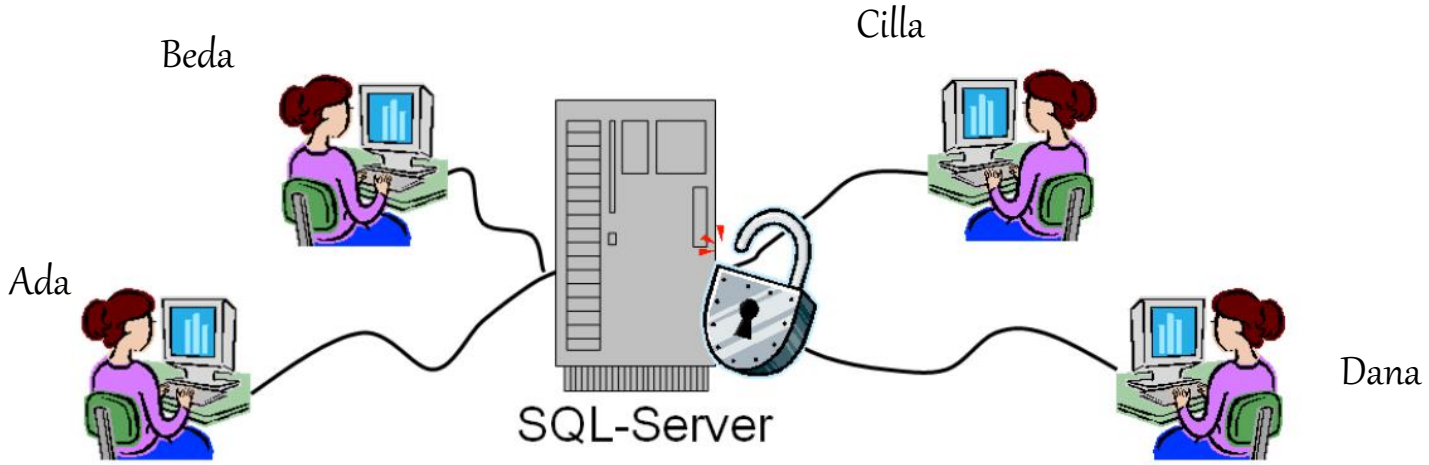
Chapter 8
Beginning SQL Server 2008 for Developers

Fleranvändarsystem



När jag ska boka en tid eller en plats så kan det finnas andra som vill boka samma tid eller plats. Hur fungerar detta?

Concurrency conflict



Följande händer i tidsordning:

1. Ada läser upp en post från en tabell.
2. Beda läser upp samma post något senare.
3. Beda ändrar och sparar posten.
4. Hur ska Ada veta vad som gäller. Ada har felaktiga data.

Det innebär att det finns en konflikt i samtidighet av data. Normalt gäller det att två eller flera användare vill påverka en och samma post samtidigt.

Concurrency conflict

Använd ex datatypen
TIMESTAMP/ROWVERSION
Ger unikt värde för datainnehållet.

Tabellstruktur

	Column Name	Data Type	Allow Nulls
🔑	ID	int	☐
	Enamn	varchar(25)	☐
	Fnamn	varchar(25)	☐
	Version	timestamp	☐

Tabellen i SSMS

	ID	Enamn	Fnamn	Version
▶	1	Ek	Otto	<Binary data>
	2	Holm	Anna	<Binary data>

SELECT * FROM Medlem

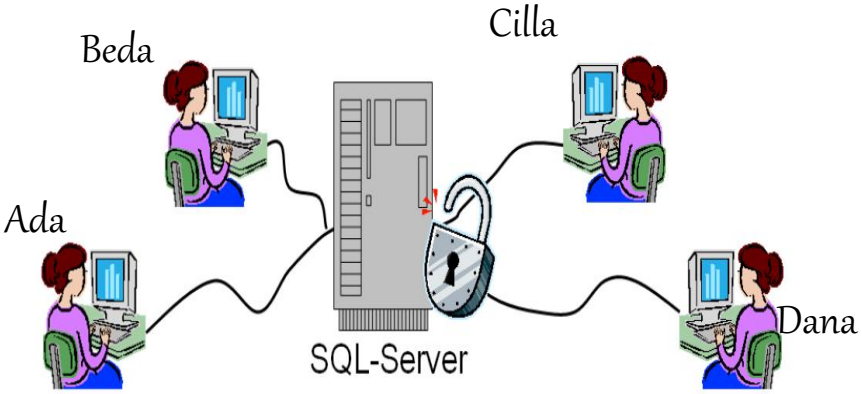
	ID	Enamn	Fnamn	Version
1	1	Ek	Otto	0x00000000000007D1
2	2	Holm	Anna	0x00000000000007D3

Efter ändring:
SELECT * FROM Medlem

	ID	Enamn	Fnamn	Version
1	1	Ek	Otto	0x00000000000007D1
2	2	Blom	Anna	0x00000000000007D4

1. Ada läser upp posten och får ett värde på fältet Version.
2. Beda läser upp posten och får samma värde i fältet Version.
3. Beda ändrar och sparar vilket skapar ett nytt värde i Version.
4. Ada ändrar och vill spara. Vad gör vi nu?

Rita o berätta



Hur tycker du att vi ska hantera detta?

- ✓ När vi ska uppdatera ändring måste vi kolla om posten har ändrats från det att vi hämtade den.
- ✓ Om posten är ändrad kan vi göra på olika sätt.
  a) Hämta upp posten igen och låta användaren mata in ändringar igen?
  b) Lägga upp två formulär med data från det egna och data från data som finns på disk. Ett av dem går att spara och ändra i.
  c) Kasta ändring och visa ett felmeddelande. Användaren bestämmer!
- ✓ Om posten inte är ändrad då kan vi genomföra uppdateringen.

Problem / Begrepp

Lost Update	En användarens uppdatering skriver över en annan uppdatering.
Dirty Data	Användare kan se andra användares uncommited data. Ändring pågår!!
Unrepeatable reads	En andra läsning ger ett annat resultat än tidigare läsning av samma post.
Optimistic Locking	Låsning sker vid verkliga behov. Detta är den vanligast modellen. Ada o Beda kan läsa. Problemet är Lost Updates som vi måste ta hand om.
Pessimistic Locking	Låser alltid. Ingen annan kommer åt den låsta posten? Kan användas vid kortare låstider. I fallet med Beda o Ada så hade Ada låst posten så att Beda inte kom åt den.

Transaktioner

- En ensats transaktion kan bara lyckas eller misslyckas.
- En flersats transaktion kan lyckas helt, delvis lyckas (=delvis misslyckas) och misslyckas helt.

Ordningen på tabellerna!!!

```
BEGIN TRY
    BEGIN TRAN
      UPDATE Artikel
      SET antal=antal-23
      WHERE artikelid=104;
      INSERT INTO Fakturarad (Fakturaid,Artikelid,Antal,Pris,rabatt,momsid)
      VALUES (2,104,23,6.50,0,1);
    COMMIT TRAN
  END TRY

  BEGIN CATCH
    ROLLBACK TRAN
    RAISERROR('Transaktionen gick inte att genomföra!',16,1)
  END CATCH
```

ACID

ACID är en minnesregel för de krav som enligt ANSI ska ställas på en transaktion. Vi bör rätta oss efter detta och definiera transaktioner när vi ska medverka till att det uppfylls.

Observera att även endast en SQL-sats är att betrakta som en transaktion.

Atomicity Transaktionen ska vara odelbar som en Atom.
Allt eller inget ska genomföras – COMMIT / ROLLBACK

Consistency Utan inre motsägelser. Databasen ska behålla sitt stabila tillstånd efter en transaktion. Oavsett om den lyckas eller misslyckas.

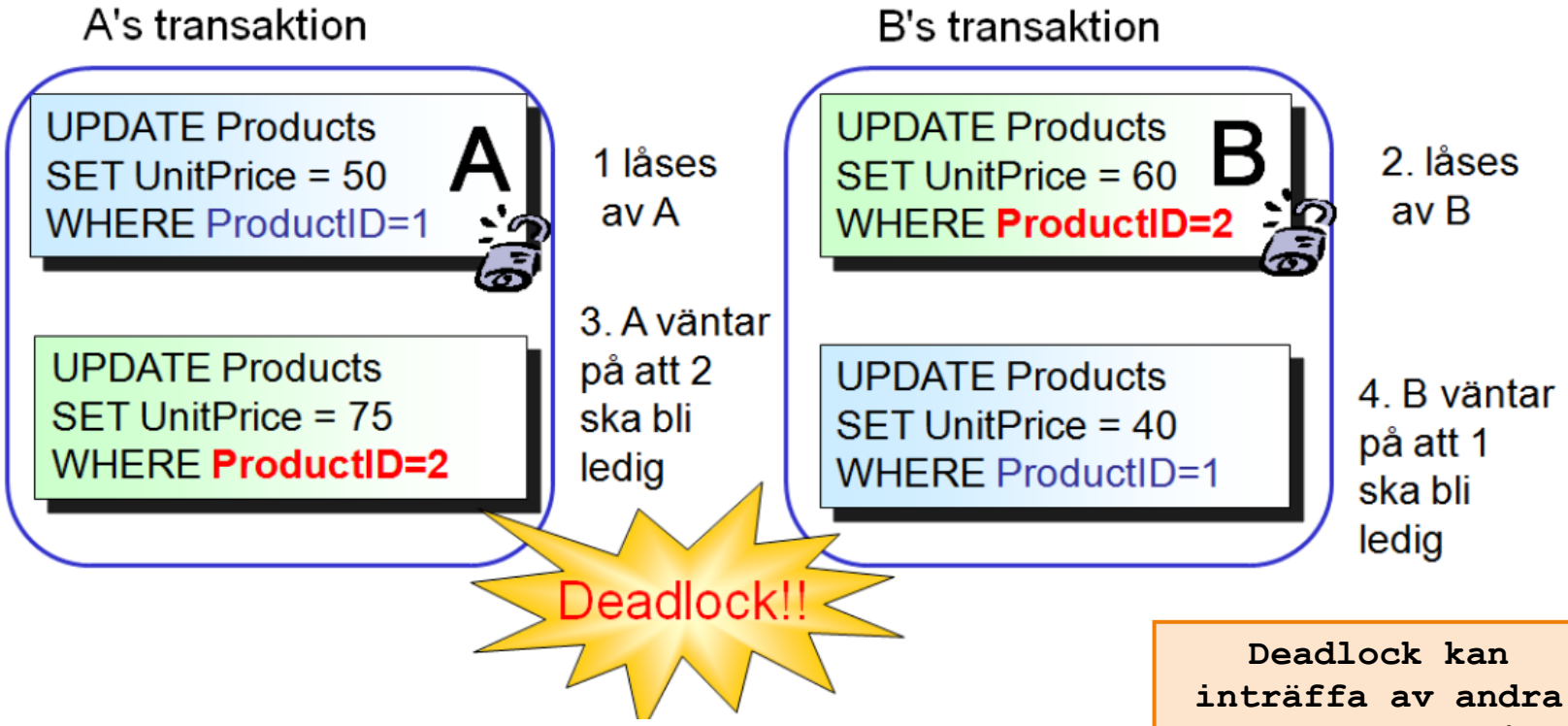
Isolation Resultatet av en transaktion ska vara oberoende av andra transaktioner. Databashanteraren ska kunna hantera samtidiga transaktioner (concurrency) och tillfälligt låsa för andra användare / transaktioner.

Durability Hållbarhet. När en transaktion avslutats (COMMIT) ska transaktionen bestå permanent även om servern går ner. Backup + log.

Deadlock

Deadlock uppstår exempelvis då samma resurser läses och låses av en eller flera transaktioner men i olika ordning.

En transaktion kan komma att vänta i evighet på att andra transaktioner som har resurser låsta.



Deadlock kan inträffa av andra orsaker också

Deadlock

Default
5 sekunder

- Deadlock upptäcks automatiskt av SQL Server.
  SQL Server väljer då ut en av transaktionerna, gör ROLLBACK på den och visar ett felmeddelande. @@ERROR_NUMBER = 1205.

- Deadlock kan undvikas genom att alltid skapa låsen i samma ordning.
  Att låsa i tabellordning och primärnyckelordning är vanligt.

- Deadlock kan minimeras genom att
  - Skapa korta transaktioner
  - Reducera låstiden med isolation level eller lock hints
  - Undvika användarinterface-kod inuti en transaktion

- En transaktion kan ta på sig att frivilligt göra en ROLLBACK i händelse av deadlock. Kör i början av transaktionen:

```
SET DEADLOCK_PRIORITY LOW
Alternativt kan NORMAL, HIGH eller -10 till 10.
```

Kod för att hantera Deadlock

```
DECLARE @retry AS TINYINT = 1;
DECLARE @retrymax AS TINYINT = 2;
DECLARE @retrycount AS TINYINT = 0 ;

WHILE @retry = 1 AND @retrycount <= @retrymax
    BEGIN
        SET @retry = 0 ;
        BEGIN TRY
            UPDATE  HumanResources.Employee
            SET     LoginID = '54321'
            WHERE   BusinessEntityID = 100 ;
        END TRY
        BEGIN CATCH
            IF (ERROR_NUMBER() = 1205)     -- 1205 errornr deadlock
                BEGIN
                    SET @retrycount = @retrycount + 1 ;
                    SET @retry = 1 ;
                END
        END CATCH
    END
```

script / procedur / trigger

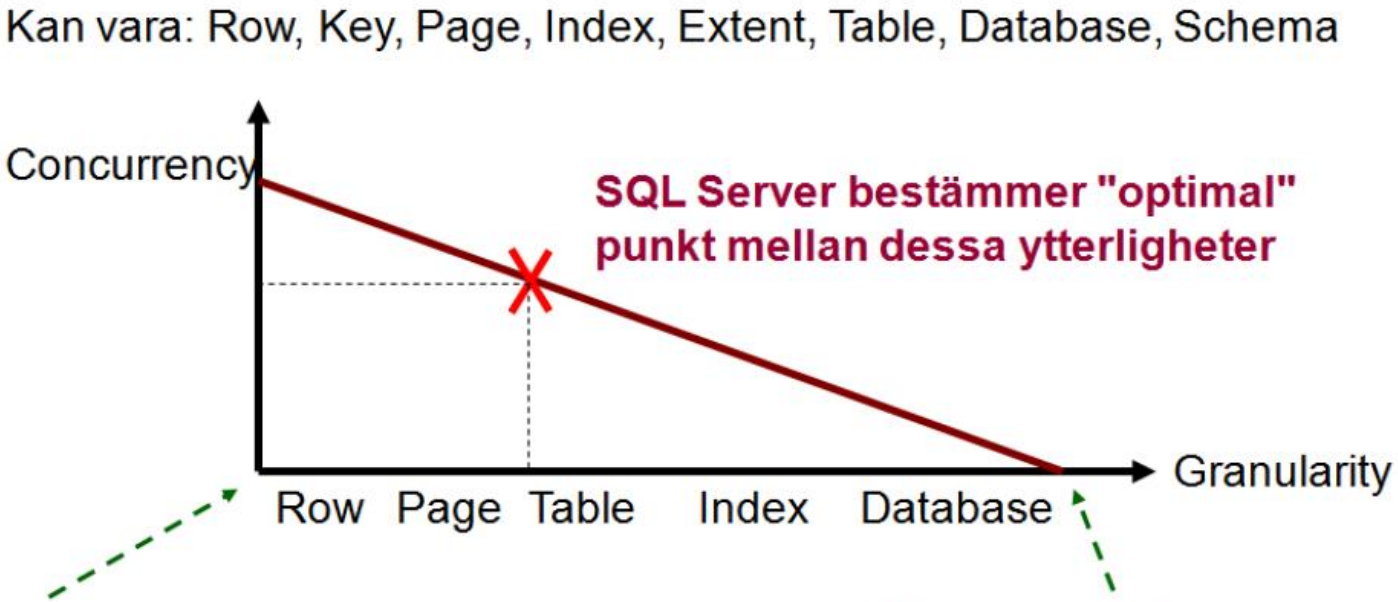
```
CREATE PROCEDURE [dbo].[usp_Artikel_Change]
@artikelID int = 0,
@namn varchar(30) = NULL,
@antal smallint = NULL,
@pris decimal(6,2) = NULL,
@artkatid int = NULL,
@plats char(10) = NULL
-- IN parametrar ska sättas till DEFAULT så långt som möjligt
-- IN parametrar ska vara av samma längder o datatyp som i tabell
AS
BEGIN
    -- Finns ID som är angivet
    -- Är de värden vi ska lägga in/ändra OK
    -- Använd TRANSACTION/COMMIT/ROLLBACK om fler än en SQL sats dvs - det ska vara en transaction
    -- Tänk på datatyper och längder - speciellt om globala variabler används!
    -- Använd TRY/CATCH
    -- Lämna vettiga felmeddelanden
    BEGIN TRY
        IF EXISTS(SELECT Artikelid From Artikel WHERE Artikelid=@artikelID)
            UPDATE dbo.Artikel
            SET
                dbo.Artikel.namn = ISNULL(@namn, dbo.Artikel.namn),
                dbo.Artikel.Antal = ISNULL(@antal, dbo.Artikel.Antal),
                dbo.Artikel.Pris = ISNULL(@pris, dbo.Artikel.Pris),
                dbo.Artikel.artkatid = ISNULL(@artkatid, dbo.Artikel.artkatid),
                dbo.Artikel.Plats = ISNULL(@plats, dbo.Artikel.Plats)
            WHERE dbo.Artikel.ArtikelID = @artikelID;
        ELSE
            RAISERROR ('Artikeln saknas - det går inte att uppdatera!',16,1)
    END TRY
    BEGIN CATCH
        RAISERROR ('Ett fel har uppstått! Uppdatering går inte att genomföra.',16,2)
    END CATCH
END
```

Lock Size (Granularity)

Lock Size	Beskrivning
Row Lock	Låser en enstaka rad
Page Lock	Låser en sida (=8KB) eller flera rader som då måste finnas på en sida.
Extent Lock	Låser 8 sidor (64KB)
Table Lock	Låser hela tabellen
Database Lock	Låser hela databasen.
Key Lock	Låser noder utifrån index

Granularity på låsningen

Granularity (upplösning) Mängden data som låses samtidigt.



Liten granularitet

- Mer overhead
  - ”jobbigt” att låsa smådelar
- Hög concurrency!
  - Många samtidiga användare

Stor granularitet

- Mindre overhead
  - ”Lätt” att låsa hel databas
- Låg concurrency
  - Fåtal samtidiga användare

Lock modes

Hanteras Dymaniskt

Share (read)	Vanligast. Många användare kan samtidigt läsa men inte skriva resurser som har Shared Locks
Update	En transaktion läser data med syftet att uppdatera senare. Kommer bli en X. Andra användare kan läsa poster som är Updated Locked
EXklusive (write)	Endast en användare kan ha Exklusive Lock. Ingen annan kan göra något med en resurs som är Exklusive Locked
Intent	Indikerar att en transaktion ämnar skaffa sig ett lås med finare granularitet senare. Hindrar andra att sätta row exclusive lock. (IS = Intent Share)
Bulk Update	Sätts vid Bulk Copy operationer till en tabell Ingen annan kan komma åt tabellen
Schema	Låser på data dictionary. Tex då en kolumn läggs till.) Sch-S = Stability låst för ändring DDL Sch-M = Låst pga av modifiering. Kan ej läsas.

Dynamic Locking

SQL Server använder sig av Dynamic Locking.

- ✓ Väljer automatiskt locking granularity för varje tabell i varje fråga.

  Query Optimizer väljer den bästa granularity baserad på typen av fråga.

- ✓ Dynamisk låsning har att välja på
  - Ett stort antal små lås
  - Ett litet antal stora lås

SQL Server hanterar både granularity och mode automatiskt.

SQL Server väljer också en lock mode för varje fråga.

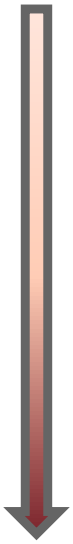
- ✓ Viktigast för en utvecklare är
  - Granularity för share och exclusive locks
  - Varaktigheten hos share och exclusive locks

Vi vill inte ha stora lås hållna under lång tid!!!

Isolation Levels

Möjligheten att isolera en transaktion från andra transaktioner är reglerat i ANSI-standarden med ett antal nivåer.

Låg



Hög

Isolation Level	Beskrivning
Read uncommited	Använder bara så mycket lås att det kan säkerställas att felaktiga data inte kan läsas.
Read commited (**Default**)	Share locks hålls under tiden en sida läses. Exclusive locks hålls under hela transaktionen.
Repeatable read	Samma som read commited, men share locks hålls under hela transaktionen.
Serializable	Samma som repeatable read men håller också ett exklusive locks för att hindra inserts.

Vill du förändra nivå för en transaktion kan du exempelvis skriva följande sats:

```
SET TRANSAKTION ISOLATION LEVEL Read uncommited;
```

Locking Hints

Isolationsnivåerna påverkar alla tabeller för den aktuella anslutningen.

Locking hints ger möjlighet att ändra för en enstaka tabell där Locking hints ges direkt efter tabellnamnet. Har du flera locking hints läggs de efter varandra med komma som separator.

Läs data från Artikel-tabellen utan att vänta på eller sätta några lås:

```
SELECT *
FROM Artikel WITH (NOLOCK)
WHERE ArtikelID=1;
```

Uppdatera Artikel med pagelocks:

```
UPDATE Artikel WITH (PAGLOCK)
SET Pris = Pris*1.1
WHERE ArtikelID=1
```

SQL Server Dynamic locking sätter vanligen rätt låstyp. Ändra bara om det är nödvändigt.

Locking Hints

Locking hint	Beskrivning
NOLOCK	Sätt inte share locks eller vänta på exclusive locks
HOLDLOCK	Håll share locks under hela transaktionen
PAGLOCK	Använd pagelocks
READUNCOMMITED	Samma som motsvarande isolationen level
READCOMMITED	- - - - - - - - - - - ” - - - - - - - - - - - -
READREPEATABLEREAD	- - - - - - - - - - - ” - - - - - - - - - - - -
SERIALIZEABLE	- - - - - - - - - - - ” - - - - - - - - - - - -
READPAST	Hoppar över låsta rader istället för att vänta
ROWLOCK	Använd row locks
TABLOCK	Använd table locks
TABLOCKX	Använd exclusive table locks
UPDLOCK	Använder update locks istället för share locks och håller under hela transaktionen
Xlock	Håller exclusive locks under hela transaktionen.

Lock Timeouts

Som standard är väntetiden oändlig för en transaktion på att ett lås ska släppa.

Det kan ändras genom följande:

```
SET LOCK_TIMEOUT [tid I millisekunder]
```

-1 anger att ingen tid finns definierad. Dvs det är default. =5 sekunder

När en timeout för ett lås löser ut så erhålls @@ERROR_NUMBER = 1222

Vid tester kan du använda WAITFOR och PRINT:

```
WAITFOR TIME '14:30'           - Vänta till klockan är 14:30
WAITFOR DELAY '00:03:00'       - Vänta I tre minuter
PRINT 'Nu har vi väntat nog!' - PRINT Skriver ut enbart strängar
```