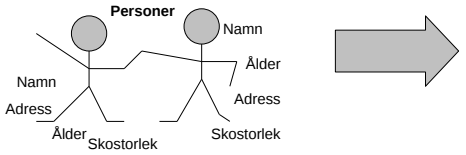
Design och underhåll av databaser

1. Modell av verkligheten — Kap. 10
2. Normalformer — Kap. 10
3. Introduktion till DDL — Kap. 11-12
4. Skapa databaser — Kap. 11-12
5. Skapa tabeller — Kap. 11-12
6. Skapa index — Kap. 11-12
7. Restriktioner — Kap. 11-12
8. Ta bort databaser, tabeller och index — Kap. 11-12
9. Ändra tabeller — Kap. 11-12

1. Modell av verkligheten (kallas också ”schema”)

- Genom databaser så skapas en *modell av verkligheten i dess sammanhang*
- En *tabell* motsvara oftast en typ av saker i den verkliga världen, med en sådan sak per rad
- *Attributen* (kolumnerna) motsvarar ett objekts egenskaper (attribut)
- Varje rad i tabellen (en ”post”) motsvarar en enhet (ett objekt, en instans av objektttypen)

Modellering av verkligheten



Personer

Namn	Ålder	Adress	Skostorlek

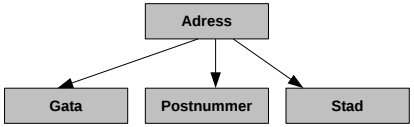
Notera

- Teknik som ofta används för att modellera databaser är *Entity-Relationship-modellering* (ER-modellering)

1.1. Identifiera objekt och attribut – del1

- Ofta finns det redan information om objekt
  - *Ex.:* register över anställda, telefonkatalogen, *etc.*
- Ofta finns det även redan ett förhållande mellan objekten
  - *Ex.:* personer är anställda på ett företag
- Objekt har attribut (egenskaper)!
- Attribut som består av flera delar går ofta att dela upp (i fler attribut)
- *Det är lättare att återskapa sammansatta attribut än att dela upp hopsatta attribut!*

Uppdelning av attribut



1.2. Identifiera objekt och attribut – del2

- *Identifierade objekttyper* blir tabeller och *identifierade attribut* blir attribut
- Innan tabeller skapas bör följande frågor ställas:
  - Behövs alla identifierade attribut?
  - Behövs det flera attribut för att kunna skapa kopplingar mellan tabeller?

Person-objektens alla identifierade attribut:

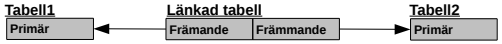
- `Namn`
- `Ålder`
- ~~`Personnummer`~~ `-- Är personnummer ett bra alternativ till primärnyckel?`
- *`PersID`* `-- Kan det vara bättre med en identitetskolumn?`
- `Adress`
- ~~`Skostorlek`~~ `-- Är detta attribut verkligen nödvändigt för systemet?`

1.3. Identifiera nyckelförhållanden

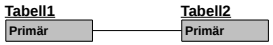
- Nyckelförhållande mellan tabeller beskriver *verkliga kopplingar*
- Ett attribut (eller i nödfall en kombination av attribut) i varje tabell ska vara *primärnyckel!*
  - Finns inte redan ett lämplig attribut så kan en *identitetskolumn* läggas till
- Ett förhållande till en annan tabell skapas genom en *främmande nyckel*
  - Refererar till ett attribut (*primärnyckeln* eller annat *unikt* attribut) i en annan tabell

En-till-många (1-to-M)

Många-till-många (M-to-N)



En-till-en (1-to-1)



1.4. Upprätthålla kopplingar mellan tabeller

- Att upprätthålla kopplingarna mellan tabeller är viktigt!!
  - Upprätthålls genom *restriktioner* (integritetsvillkor) på nyckelattribut
- Skapas en koppling mellan en *främmande nyckel* och en *primärnyckel* i en annan tabell så kan *restriktionerna* användas för att upprätthålla kopplingarna

Kopplingsproblem som måste behandlas

1. När en post med *primärnyckel* tas bort så kommer refererande *främmande nycklar* att sakna referens
2. Om en post med *primärncykel* ändras så återspeglas inte ändringarna i *främmande nycklar*
3. När en post med en *främmande nyckel* läggs till så är det inte säkert att det finns en *primärnyckel* att referera till
4. Om en post med *främmande nyckel* ändras så är det inte säkert att nyckeln ändras till en existerande *primärnyckel*

Notera

- *Triggers* är användbara för att upprätthålla kopplingar mellan tabeller (senare i kursen)!
- En *främmande nyckel* som inte refererar till en *primärnyckel* kallas för en *övergiven nyckel.*

1.5. Identifiera behov av indexering

- Ett index kan ge förbättrad prestanda vid sökning i tabellen (automatiskt)
- Ett index förbättrar *sökhastigheten* för `WHERE`-uttryck och `JOIN`-uttryck (men det är inte alltid det går)
- Index måste *uppdateras* varje gång som posterna ändras (automatiskt)
- I SQL Server finns det alltid ett index på *primärnyckeln* (automatiskt)

När bör index (utöver *primärnyckeln*) användas?

- Då attribut är *främmande nycklar* (så det går snabbt att söka (”baklänges”)
- Då attribut är vanligt förekommande i sökvillkor eller `JOIN`-uttryck
- Då attribut innehåller många unika värden
- Då attribut sällan ändras

2. Normalformer

- *Normaliserade databaser* innehåller i regel fler tabeller
  - Varje tabell har en *indexerad primärnyckel* och är därför effektiva vid sökning
- Mindre tabeller är effektivare när poster *läggs till*, *ändras* eller *tas bort*
- Målet med normalisering är att att minimera *dataredundans*
  - Underlättar vid underhåll och minskar lagringsutrymmet
- När en databas normaliseras så finns det ett antal *normalformer* att följa

Normalformer

- Första normalformen (1NF) – attribut ska innehålla *atomära värden* (dvs. ett värde per cell)
- Andra normalformen (2NF) – alla attribut som inte tillhör *primärnyckeln* måste vara helt beroende av *hela primärnyckeln,* dvs det får inte finnas några attribut som är beroende av en del av *primärnyckeln*
- Tredje normalformen (3NF) – det får inte finnas några beroende mellan de olika attributen utanför primärnyckeln
- Boyce-Codds normalform (BCNF) – ännu lite högre krav, men mest att den är enklare att formulera

Notera

- En normalform förutsätter att alla tidigare normalformer är uppnådda.
- En tabell anses (lite grovt) vara normaliserad då den uppfyller den *tredje normalformen*.

2.1. Exempel

- ***Denna tabell uppfyller inte första normalformen eftersom det finns o-atomära värden:***

Cyklar

Märke	Årsmodell	Beskrivning
Crescent	1997	Blå färg, aningen begagnad, 3 växlar...
Kronans	1988	Svart färg, begagnad, inga växlar...

- ***Denna tabell uppfyller INTE andra normalformen eftersom alla övriga attribut INTE är helt beroende av den sammansatta primärnyckeln LärarID och AkademiID:***

Lärare

Förnamn	Efternamn	LärarID	AkademiID	Akademi
Andreas	Persson	1	3	Handelshögskolan
Johan	Petersson	2	3	Handelshögskolan

- ***Dessa tabeller uppfyller andra normalformen eftersom övriga attribut för varje tabell är helt beroende av primärnycklarna LärarID respektive AkademiID:***

Lärare

Förnamn	Efternamn	LärarID	Akademi
Andreas	Persson	1	3
Johan	Petersson	2	3

Akademier

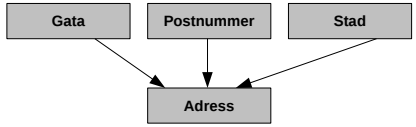
AkademiID	Namn
3	Handelshögskolan
4	Akademin för Naturvetenskap och teknik

2.2. Denormalisera (men bara om det verkligen behövs)

- **Används attribut frekvent i `JOIN`-uttryck så kan det ibland vara en idé att *denormalisera tabellerna***
  - **Slå ihop tabeller eller attribut**
- **Tabeller normaliserade över *tredje normalformen* (3NF) kräver i regel flera `JOIN`-uttryck för att återhämta sammansatta värden**
- **Tabeller som sällan ändras men som används frekvent kan även ge bättre prestanda ifall de *denormaliseras***

Exempel

- ***Denormalisering till ett sammansatt attribut Adress:***



3. Introduktion till DDL

- ***Data definition language* (DDL) är samlingsnamnet för de uttryck som används för att *skapa*, *ändra* och *ta bort objekt***
- **Ofta är det databasadministratörer som sköter DDL, medan utvecklare använder DML**
  - ***Utvecklare bör dock känna till grundera i DDL för att kunna skapa test-databaser, etc.!***
  - ***Ofta är man själv DBA, även om man egentligen är utvecklare***

DDL-uttryck

- **`CREATE` {`DATABASE`|`TABLE`|`INDEX`|`FUNCTION`|`PROCEDURE`|`TRIGGER`|`VIEW`}**
- **`ALTER` {`TABLE`|`FUNCTION`|`PROCEDURE`|`TRIGGER`|`VIEW`}**
- **`DROP` {`DATABASE`|`TABLE`|`INDEX`|`FUNCTION`|`PROCEDURE`|`TRIGGER`|`VIEW`}**

Notera

- **Ofta används ett grafiskt verktyg (t.ex. *Management Studio*) tillsammans med DDL.**
- **Index behöver endast skapas för ett attribut.**

3.1. Objektnamn

- **Ett *objektnamn* (namnet på t.ex. en databas) kallas för *identifierare***
- ***Identifierare* måste inledas med: *bokstav* (Unicode), *understreck* (_), *at-tecken* (@) eller *nummertecken* (#)**
- **Resterande tecken i identifieraren kan bestå av alla tecken i *Unicode standarden***
- ***Identifierare* skrivna mellan citattecken (`""`) eller hakparenteser (`[]`) kan även innehålla *mellanrum***
- ***Identifierare* får inte ha samma namn som ett *nyckelord***

Exempel

```
Tabell_namn
Tabell3
#Temp_tabell       -- Temporär tabell
[Tabell namn]      -- Hakparenteserna tillåter mellanrum
```

Notera

- **Inleds @-tecken eller #-tecken indikerar att det är en *parameter* respektive en *temporär tabell.***

4. Skapa databaser

- Genom `CREATE DATABASE`-uttrycket så skapas en ny databas med *defaultinställningar*
  - Skapas i gällande arbetskatalogen
- Det går även att skapa en *databaskopia* genom att använda `FOR ATTACH`-uttrycket tillsammans med `ON PRIMARY`-uttrycket

Syntax

```
CREATE DATABASE Databasnamn
    [ON [PRIMARY] (FILENAME = 'sökväg + filnamn')]
    [FOR ATTACH]
```

Exempel

- *Skapar en kopia av Ekonomi-databasen:*

```
CREATE DATABASE Ekonomi2
    ON PRIMARY (FILENAME = 'C:\Temp\Ekonomi.mdf')
    FOR ATTACH
```

Notera

- *Vissa system kan skriva över en redan existerande databas med* `CREATE DATABASE`*!*

5. Skapa tabeller

- *Tabeller* skapas genom `CREATE TABLE`-kommandot
- I samband med `CREATE TABLE`-kommandot så definieras även: *tabellens attribut*, *egenskaper* för attributen och *tabellrestriktioner* (integritetsvillkor)
- *Varje attribut måste ha ett unikt namn och en angiven datatyp!*

Syntax

```
CREATE TABLE Tabell
    (Attribut1 Datatyp [Egenskaper]
    [, Attribut2 Datatyp [Egenskaper]...
    [, Tabellegenskaper])
```

Egenskaper för attribut

```
NULL|NOT NULL        -- Ifall attributet tillåter NULL-värden
PRIMARY KEY|UNIQUE   -- Indikerar primärnyckel eller annat unikt attribut
IDENTITY             -- Identitetskolumn (ofta primärnyckeln)
DEFAULT Värde        -- Anger ett defaultvärde för attributet
SPARSE               -- Optimerar lagring
```


5.1. Exempel

- *Skapa tabellen Utbetalningar (inkl. en primärnyckel):*

```
CREATE TABLE Utbetalningar
    (UtID       SMALLINT     PRIMARY KEY IDENTITY,
     AnställdID SMALLINT     NOT NULL,
     Belopp     INT          NULL,
     Månad      NVARCHAR(50) DEFAULT 'Juli')
```

Utbetalningar

UtID	AnställdID	Belopp	Månad

Notera

- En *primärnyckel* tillåter ej `NULL`-värden.
- Ett `UNIQUE`-attribut tillåter `NULL`-värden.
- *Det kan endast finnas en identitetskolumn i en tabell!*
- Ifall inget anges så tillåter attributet `NULL`-värden (undantag *primärnyckel*).

6. Skapa index

- *Index* i tabeller skapas genom `CREATE INDEX`-kommandot
- Det skapas automatiskt ett *clustered index ("grupperat", "klustrat")* för *primärnyckeln*
- Det skapas även ett *non-clustered index* för alla *unika attribut*
- Ett *fulltabellsindex* är ett index som använder alla poster i en tabell
  - Motsats: Ett *filtrerat index* innehåller ett `WHERE`-uttryck

Syntax

```
CREATE [CLUSTERED|NONCLUSTERED] INDEX Index
    ON Tabell (Attribut1 [ASC|DESC] [, Attribut2 [ASC|DESC]...])
    [WHERE Sökvillkor]
```

Notera

- Det kan finnas ett *clustered* och 249 *non-clustered index* i en tabell.
- Ett *clustered index* är i samma ordning som den som posterna faktiskt ligger lagrade, så det kan förstås bara finnas ett per tabell
- Ifall ett *clustered index* önskas på annat än primärnyckel så måste `ALTER TABLE`-kommandot först användas för att ta bort indexet på primärnyckeln.

6.1. Exempel

- ***Ett fulltabellsindex skapas på attributet Belopp i tabellen Utbetalningar:***

```
CREATE NONCLUSTERED INDEX IX_Belopp
    ON Utbetalningar (Belopp ASC)
```

- ***Ett filtrerat index som utesluter NULL-värden skapas på attributet Belopp i tabellen Utbetalningar:***

```
CREATE NONCLUSTERED INDEX IX_Belopp
    ON Utbetalningar (Belopp ASC)
    WHERE Belopp IS NOT NULL
```

Notera

- **Prefixet `IX_` är rekommenderat för namnen på index.**
- **Både nyckelorden `NONCLUSTERED` och `ASC` är *default* (anges här för att förtydliga).**

7. Restriktioner (”integritetsvillkor”)

- **Restriktioner kan användas både på *attribut-* och *tabellnivå***
- **På *attributnivå* så läggs restriktioner endast på det attributet**
- **På *tabellnivå* så läggs restriktioner på alla attribut i tabellen**
- ***Operationer testas mot restriktionerna innan operationen utförs!***

Restriktioner

- `PRIMARY KEY` -- En eller flera attribut används som primärnyckel
- `UNIQUE` -- Unika värden över en eller flera attribut
- `CHECK` -- Utför en kontroll på en eller flera attribut
- [`FOREIGN KEY`] `REFERENCES` -- En eller flera attribut utgör en främmande nyckel (referens)

Notera

- **Utförs en operation som strider mot restriktionerna så svarar databasen med ett fel (error)**

7.1. Exempel och CHECK-uttrycket

- **Genom `CHECK`-uttrycket så kan posterna begränsas till endast poster som uppfyller en *kontroll***
  - **Kan användas både på *attribut-* och *tabellnivå***

Exempel

- ***Endast poster som har ett Belopp större än noll (> 0) ska tillåtas i tabellen Utbetalningar. Dessutom ska primärnyckel utgöras av attributen UtID och AnställdID:***

```
CREATE TABLE Utbetalningar
(UtID       SMALLINT     IDENTITY,
 AnställdID SMALLINT     NOT NULL,
 Belopp     INT          NULL  CHECK( Belopp > 0),
 Månad      NVARCHAR(50) DEFAULT 'Juli',
 PRIMARY KEY (UtID, AnställdID))
```

Notera

- **Resultatet av `CHECK`-uttrycket är av typen *bool.***

7.2. Främmande nycklar

- ***Främmande nycklar* är restriktioner som skapar via kopplingar mellan tabeller**
- ***Främmande nycklar* skapas med `FOREIGN KEY`-uttrycket,**
  - **Skapar en referens till ett (unikt) attribut i en annan tabell**
- **Restriktioner för *främmande nycklar* upprätthålls genom `ON DELETE` och `ON UPDATE`-uttryck tillsammans med nyckelorden `CASCADE` eller `NO ACTION`**
  - **`CASCADE` ändrar även på refererande främmande nycklar**
  - **`NO ACTION` tillåter inga ändringar på attribut som främmande nycklar refererar till**

Exempel

- ***Skapa en tabell Lärare med en främmande nyckel till attributet AkademiID i tabellen Akademier:***

```
CREATE TABLE Lärare
(Förnamn       NVARCHAR(50) NOT NULL,
 Efternamn     NVARCHAR(50) NOT NULL,
 Akademi       INT          FOREGIN KEY REFERENCE Akademier (AkademiID))
```

Notera

- **`NO ACTION` är default för både `ON DELETE` och `ON UPDATE`-uttrycket.**
- **Nyckelorden `FOREGIN KEY` kan utlämnas (men ger en tydligare syntax).**

8. Ta bort databaser, tabeller och index

- Genom **DROP**-kommandot kan *databaser*, *tabeller* och *index tas bort*
- När tabeller tas bort så kommer även all *data*, *index*, *triggers* och *restriktioner* tas bort
  - *Vyer* och *lagrade procedurer* måste dock tas bort explicit! (men: CASCADE)
- Det går inte att ta bort en tabell ifall det finns refererande *främmande nycklar* till tabellen
- Det går inte att ta bort det automatiskt skapade indexet på en *primärnyckel* (**ALTER TABLE**-uttrycket måste användas)

Syntax

```
DROP INDEX Index1 ON Tabell1 [, Index2 ON Tabell2, ...]  -- Tar bort index
DROP TABLE Tabell1 [, Tabell2, ...]                      -- Tar bort tabeller
DROP DATABASE Databas1 [, Databas2, ...]                 -- Tar bort databaser
```

Syntax

- *Ta bort hela tabellen Utbetalningar:*

```
DROP TABLE Ekonomi.Utbetalningar   -- Använder fullständigt namn
```

Notera

- Det går inte att ”ångra” en borttagning så ta alltid *backup innan borttagning!!*

9. Ändra tabeller

- Genom **ALTER TABLE**-kommandot kan *tabeller ändras*
- Med **ADD**, **DROP** och **ALTER COLUMN** så kan attribut *läggas till*, *tas bort* och *ändras*
- *Restriktioner* kan läggas till eller tas bort med **ADD [CONSTRAINT]** respektive **DROP [CONSTRAINT]**
- Genom **WITH CHECK** respektive **WITH NOCHECK** så avgörs ifall existerande värden i tabellen ska kontrolleras

Syntax

```
ALTER TABLE Tabell [WITH CHECK|WITH NOCHECK]
    {ADD Attribut Datatyp [Egenskaper] |
     DROP COLUMN Attribut |
     ALTER COLUMN Attribut Datatyp [NULL|NOT NULL] |
     ADD [CONSTRAINT] NyRestriktion
     DROP [CONSTRAINT] Restriktion }
```

Notera

- För att ändra en restriktion så måste ”namnet” på restriktionen vara känt.
- Nyckelordet **CONSTRAINT** är valfritt men rekommenderas för en tydligare syntax.

9.1. Exempel

- *Lägg till attributet Dag för kunna ange datum som månad och dag i tabellen Utbetalningar:*

```
ALTER TABLE Utbetalningar WITH CHECK
    ADD Dag TINYINT NOT NULL
```

- *Lägg till en restriktion så att det nyskapade attributet Dag endast accepterar värden i intervallet 1 till 31:*

```
ALTER TABLE Utbetalningar WITH CHECK
    ADD CONSTRAINT CHECK (Dag >= 1 AND Dag <= 31)
```