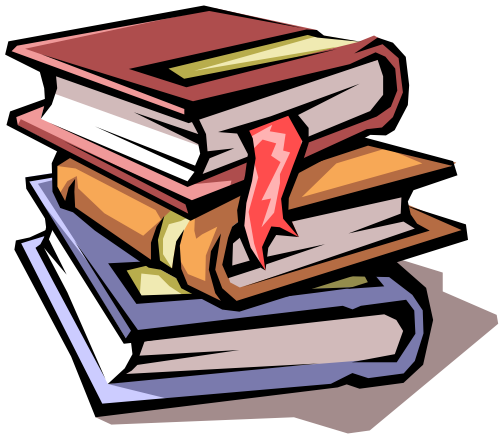
(P:\Data\GEM\Intern_Utbildning\SQL kurser\1.Lärobok SQL GrundForts.doc)

Lärobok i SQL

Grund och fortsättning



Tomas Lennefors / Olle Lindgren / 2018-03-06

(Filnamn: P:\Data\GEM\Intern_Utbildning\SQL kurser\1.Lärobok SQL GrundForts.doc)

Förord

Syftet med denna lärobok i SQL är att få grundläggande
- förståelse i databashantering
- färdighet i att använda Management Studio
- färdighet i att ställa enkla och avancerade databasfrågor
- färdighet i att utföra enklare DDL-hantering (create och drop)
- färdighet i att utföra enklare DML-hantering (insert, update och delete)
- färdighet i att skriva enklare sql-skript
- uppslagsbok för mer avancerad SQL för SCBs behov

Denna lärobok innehåller mer kunskap än som man normalt hinner med på en kurs och är därför lämplig att användas för **både grund- och fortsättningskurser**. På fortsättningskurserna kan man med fördel använda de mer grundläggande avsnitten för repetition.

Den röda tråden i läroboken är att det ska finnas **många exempel** att pröva på och ändra i. Efter exemplen i varje avsnitt kommer översiktliga beskrivningar för att få en djupare förståelse.

Databasservern för SCBs utbildningar heter **utbildning.prod.sql**. Alla exempel går att köra mot **databasen UTB_Kurs1,** som är skrivskyddad för eleverna. Här finns det även 12 elevdatabaser att skapa egna tabeller i. Alla på SCB har åtkomst till dessa databaser att öva mot på egen hand, även när man inte går på kurs.

Dessutom finns allt kursmaterial tillgängligt för alla på SCB i katalogen **P:\data\GEM\Intern_Utbildning\SQL kurser** i skrivskyddade filer.

De viktigaste exemplen finns i början av varje avsnitt och de sista exemplen är för dem som vill fördjupa sig lite mer. De avsnitt och exempel som är lämpliga för fortsättningskurser är märkta "*". Överkurs är markerat med "**".

Lycka till!

Innehållsförteckning

1. Inledning

1.1. Login, server, databas och tabell

Det **login** som vi använder på SCB, när vi loggar in i Windows, används automatiskt när vi går in i databaserna (windows authentication) och ligger till grund för vilka rättigheter (behörigheter) vi har i databaserna.

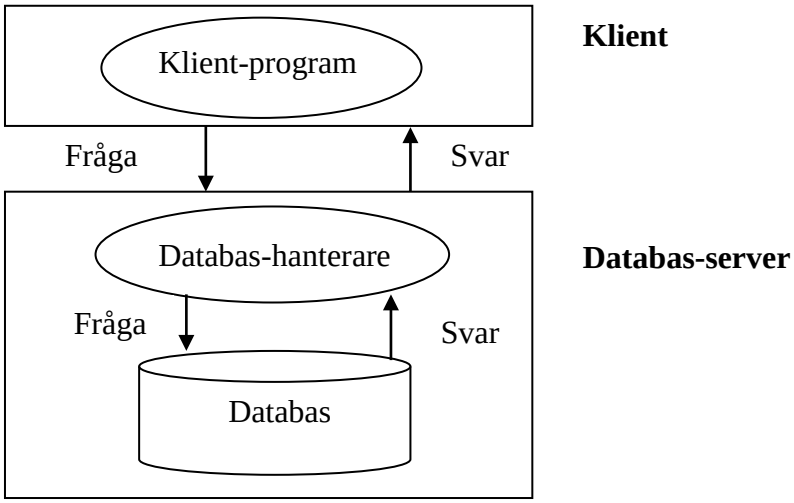
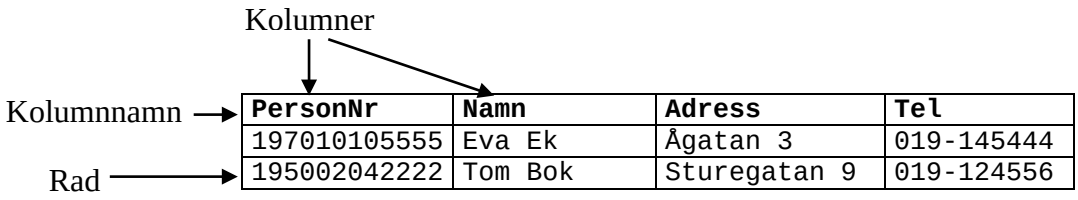


Bild 1: Visar hur en SQL-fråga skickas från klient till databasserver

Klienten är PC:n som står på skrivbordet hos användaren. På klienten finns en **klientprogramvara** som kan kommunicera med databasservern (t ex **MS SQL Server Management Studio)**. **Databasservern** (t ex **utbildning.prod.sql**), innehåller ett antal databaser samt ett program kallat **databashanterare** (t ex **MS SQL Server**). Användaren formulerar en fråga med hjälp av SQL-språket, klientprogrammet skickar frågan till servern som utför vad som begärts och skickar tillbaka svaret till klienten.



PersonNr	Namn	Adress	Tel
197010105555	Eva Ek	Ågatan 3	019-145444
195002042222	Tom Bok	Sturegatan 9	019-124556

Bild 2: Tabell

En **databas** (t ex UTB_Kurs1) innehåller ett antal objekt, t ex tabellerna Kommun och Lan. **Tabeller** är uppdelade på **kolumner** och **rader** där data lagras. Data består oftast av tal eller text.

För att hämta och/eller förändra data i en tabell används språket **SQL** (Structured Query Language). Detta språk ingår i MS SQL Server och SQL Server Management Studio. De flesta databasprogram idag kan hantera SQL, som är en världsstandard, t ex Excel, Access, SAS och SPSS.

1.2. MS SQL Server Management Studio (SSMS)

Här följer en översikt över det viktigaste om SSMS. För mer info så rekommenderas manualen **SQL Server Management Studio**, av Kjell Lindqvist.

Användargränssnittet

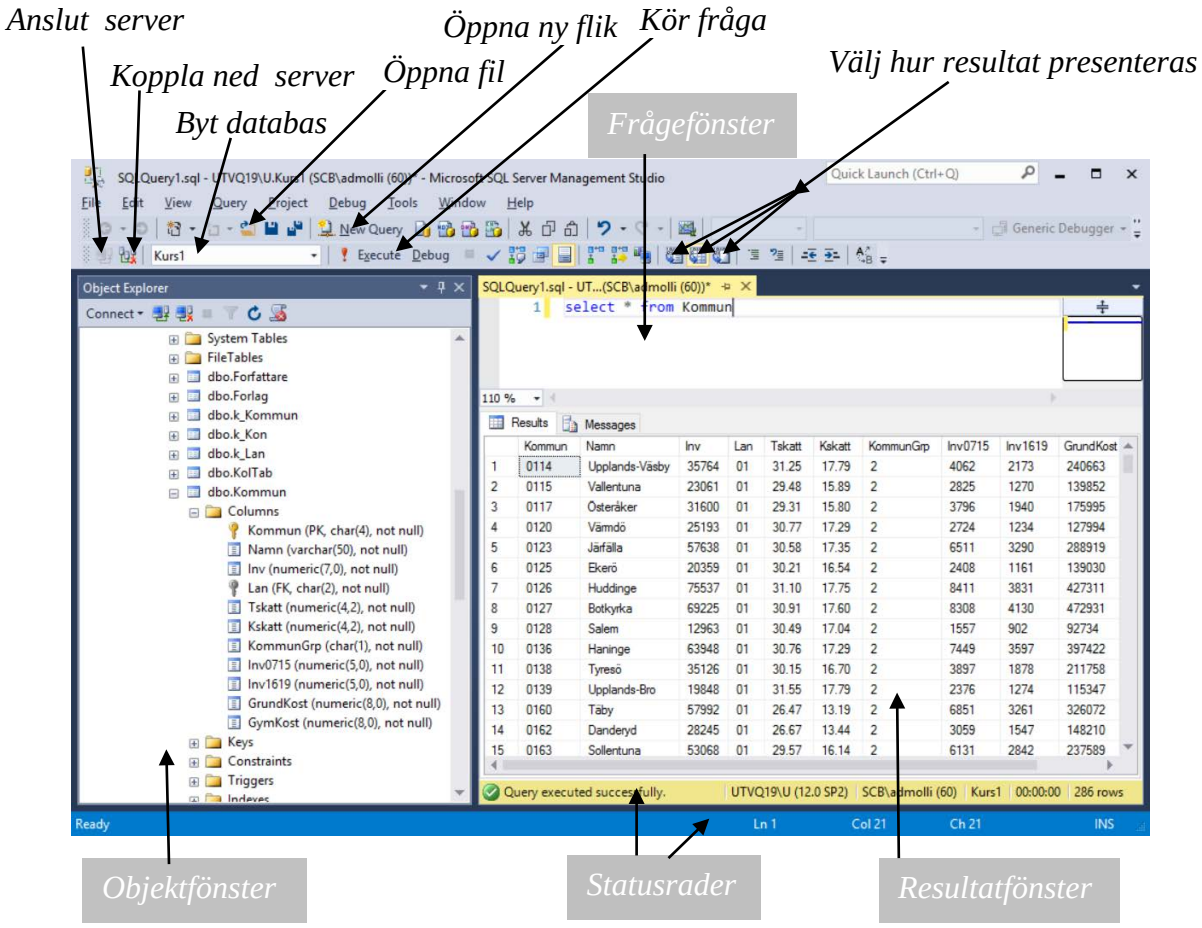


Bild 3: SQL Server Management Studio

För att kunna arbeta i databaserna på databasservern måste man ha en klientprogramvara. Denna klientprogramvara finns installerad på den PC som man sitter vid.

Det program man oftast använder mot **SQL Server** är **SQL Server Management Studio (SSMS)**. Andra klientprogram som man kan koppla upp sig mot databasen med är Access, Excel, Visual Basic och SAS.

De viktigaste fönstren i SSMS är …

- **Frågefönstret** är det centrala i SSMS och innehåller en inbyggd editor för redigering av SQL-kod. Där kan man arbeta med koden ungefär som när man arbetar med text i en ordbehandlare d v s skriver, ändrar, kopierar, tar bort, söker, sök och ersätt, spara dokumentet, öppna dokument etc.
- **Resultatfönstret** innehåller svaret på frågan man ställt i frågefönstret. Resultatet kan presenteras på tre sätt; **Grid**, **Text** och **File**.
- **Objektfönstret** visar definitionen av de olika objekten i databasen t ex servrar, databaser, tabeller, kolumner, index, vyer, lagrade procedurer. Här kan man även ändra definitionerna och/eller det data som ligger i tabellerna.

Man kan dra i kanterna på fönstren för att disponera om utrymmet på skärmen. Andra fönster är t ex **Registered Servers**, **Object Explorer Details** och **Template Explorer**. Dessa och flera andra finns under menyn **View**. Det kan vara lite knepigt i början att veta hur man får fönstren att flyta, docka eller gömma sig automatiskt.

Hur man arbetar med SSMS

Frågefönstret öppnas när man väljer *New Query* på verktygsraden. Tänk på att ange rätt databas!

1. **Skriv in** en eller flera SQL-satser i frågefönstret.

2. **Kör SQL-koden** genom att trycka på tangenten F5 på tangentbordet. Man kan även köra en del av koden genom att först markera den och sedan trycka på F5 eller klicka på Execute på verktygsraden.

3. Resultatet visas nu i resultatfönstret. Vill man nu byta **presentationssätt av resultatet** så klickar man på någon av de tre knapparna på verktygsraden och kör om frågan. I *Bild 3* ser vi att resultatet av frågan blir alla rader och kolumner i tabellen Kommun. Resultatet presenteras som default i en grid (Excel-liknande tabell).

Påloggning av SSMS

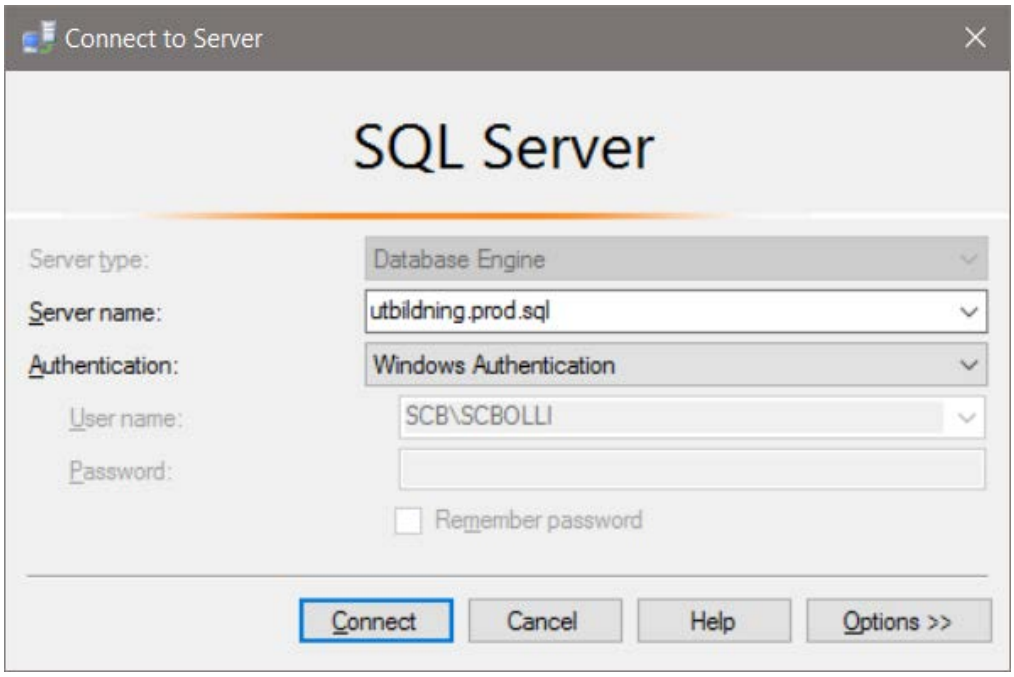


Bild 4: Påloggning av SSMS

När man startar SSMS visas en påloggningsruta. Skriv in uppgifterna som i dialogrutan ovanför! Det enda som man oftast behöver ändra på, är servernamnet. De senaste ligger alltid kvar, med den allra senaste överst. Antingen väljer man något av dessa eller så skriver man in det manuellt. Windows Authentication innebär, att man loggas in med samma användarnamn som i Windows.

Inställningar av SSMS

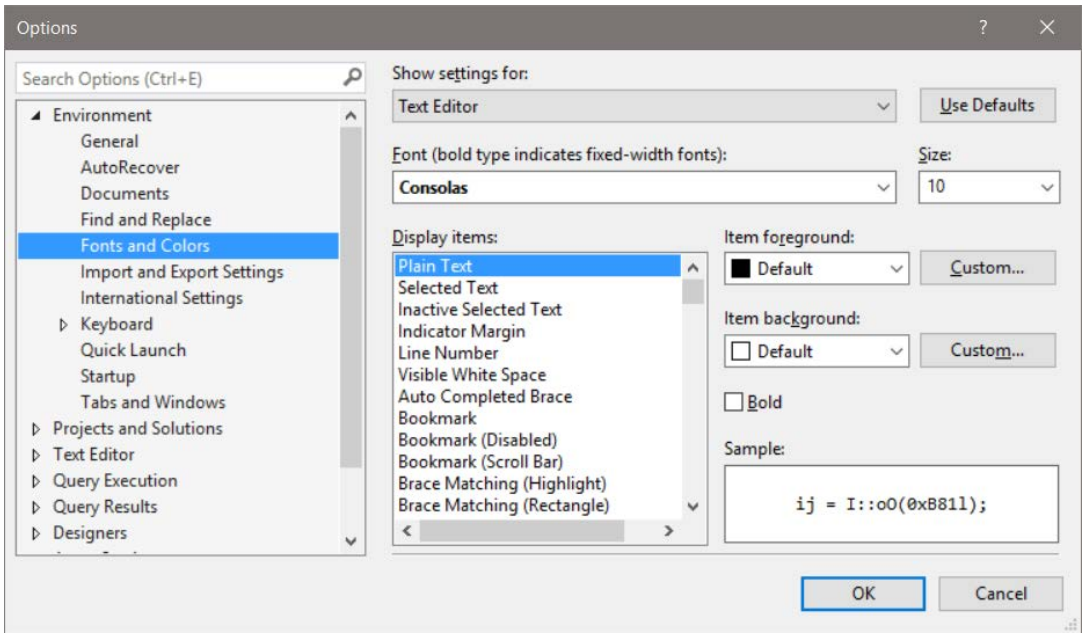


Bild 4e: Inställningar av SSMS

Det finns många olika inställningar i SSMS. Denna dialogruta hittas i menyn ”Tools/Options…”. Man kan t ex ändra fonter, textstorlek, textfärg, lägga till radnummer i frågefönstret, skapa några egna kortkommandon, ställa in längd på tabhopp och typ av resultatfönster man vill ha som default.

De vanligaste kortkommandona i Management Studio

Editorn i frågefönstret

Öppna nytt frågefönster	CTRL + N
Öppna gammal SQL-fil som nytt frågefönster	CTRL + O
Byt databas för frågefönstret	CTRL + U
Spara SQL-koden som fil	CTRL + S
Stäng frågefönstret /fliken	CTRL + F4
Markörförflyttningar	Piltangenterna, HOME, END, PAGE UP, PAGE DOWN, CTRL + piltangent, CTRL + HOME, CTRL + END
Markera text	SHIFT + Markörförflyttning
Markera all text	CTRL + A
Indentera texten	TAB
Lägg till ny rad före aktuell rad	HOME, ENTER eller CTRL + ENTER
Lägg till ny rad efter aktuell rad	END, ENTER
Foga samman rad med föregående rad	HOME, BACKSPACE
Foga samman rad med nästa rad	END, DELETE
Ta bort markerad text eller ett tecken till höger	DELETE
Kopiera markerad text	CTRL + C
Klipp ut markerad text	CTRL + X
Klistra in text från urklipp	CTRL + V
Ångra inmatning/borttag av text (max 20 steg)	CTRL + Z
Återställ ångrat	CTRL + Y
Sök text	CTRL + F
Upprepa sökning	F3
Ersätt text	CTRL + H
Kommentera markerad text	CTRL + K, CTRL + C
Bortkommentera markerad text	CTRL + K, CTRL + U
Visa bokmärkesfönstret och stäng det sedan	CTRL + K, CTRL + W, SHIFT + ESC
Sätt på/Stäng av bokmärke på aktuell rad	CTRL + K, CTRL + K
Hoppa till nästa bokmärke	CTRL + K, CTRL + N
Hoppa till föregående bokmärke	CTRL + K, CTRL + P
Kör den markerade SQL-koden eller all kod	**F5**
Kontrollera om SQL-koden är syntaxmässigt rätt	CTRL + F5
Avbryta fråga som körs	ALT + BREAK
Skriv ut SQL-koden	Klicka i frågefönstret och CTRL + P

Ytterligare några kortkommandon för SQL Server Management Studio

Hjälp	F1
Hoppa till menyraden	ALT
Öppna menyn för höger musknapp	SHIFT + F10
Avsluta programmet	ALT + F4
Skriv ut innehållet i resultatfönstret	Klicka i resultatfönstret och CTRL + P
Öppna eller hoppa till objektfönstret	F8
Öppna/Dölj resultatfönstret	CTRL + R
Hoppa till nästa fönster (Fråge/Results/Message etc)	F6
Hoppa till föregående fönster	SHIFT + F6
Hoppa till nästa flik	CTRL + F6
Hoppa till föregående flik	CTRL + SHIFT + F6
Uppdatera innehållet i objektsfönstret, t ex för att visa nya tabeller	**Markera databasnamnet i objektsfönstret och tryck F5**
Ta bort en tabell i objektsfönstret	Markera tabellnamnet, tryck DELETE

1.3. Tabeller i databasen UTB_Kurs1

Här är de tabeller i databasen som används mest i kursmaterialet.

Innehåll

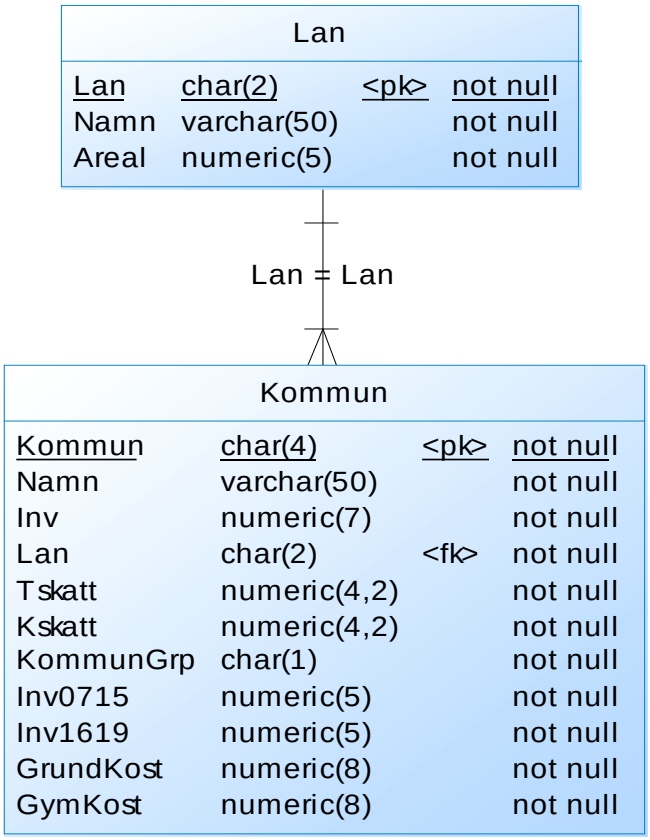
Lan (5 första raderna)
Lan2 är nästan likadan men har några dubblettrader

Lan	Namn	Areal
1	Stockholms län	6490
3	Uppsala län	6989
4	Södermanlands län	6060
5	Östergötlands län	10562
6	Jönköpings län	9944
:	:	:

Kommun (5 första raderna)
Kommun2 är nästan likadan men har NULL-förekomster i kolumnerna TSkatt och KommunGrp på 10 av raderna

Kommun	Namn	Inv	Lan	Tskatt	Kskatt	KommunGrp	Inv0715	Inv1619	GrundKost	GymKost
0114	Upplands-Väsby	35764	01	31.25	17.79	2	4062	2173	240663	86339
0115	Vallentuna	23061	01	29.48	15.89	2	2825	1270	139852	57157
0117	Österåker	31600	01	29.31	15.80	2	3796	1940	175995	66359
0120	Värmdö	25193	01	30.77	17.29	2	2724	1234	127994	47471
0123	Järfälla	57638	01	30.58	17.35	2	6511	3290	288919	54443
:	:	:	:	:	:	:	:	:	:	:

Tabelldefinitioner



1.4. SQL-språket

Exempel

`Exempel 1` visar hur man ställer en SQL-fråga och hur resultatet presenteras på skärmen. Resultatet visar kolumnerna Kommun och Namn hämtat från tabellen Kommun.

Fråga:

```
select Kommun, Namn from Kommun
```

Resultat:

Kommun	Namn
0114	Upplands-Väsby
0115	Vallentuna
0117	Österåker
:	:

`Exempel 2` visar hur man kan skriva select-satsen på flera rader för att det ska vara lättare att läsa och redigera den. Den ger samma resultat som ovan.

```
select Kommun, Namn
from Kommun
```

`Exempel 3` visar hur man kan skriva samma select-sats. Fördelen med detta skrivsätt är att det underlättar framtida redigering. Dessutom blir det lättare att läsa koden. Nackdelen är att koden tar större plats på skärmen.

Här är kolumnnamnen indragna med hjälp av TAB-tangenten. Steglängden för TAB-tangenten kan man ställa in i SQL Server Management Studio. 2 är en lagom steglängd. Dessa indrag kallas för **indentering**. Den ger samma resultat som ovan.

```
select
  Kommun
  , Namn
from Kommun
```

Beskrivning av SQL-språket

SQL innehåller fyra beståndsdelar:

- Query Language (**QL**) Frågor mot databasen
- Data Manipulation Language (**DML**) Tillägg, borttag och uppdatering av rader
- Data Definition Language (**DDL**) Tillägg, borttag och förändring av objekt
- Data Control Language (**DCL**) Hanterar säkerhet och behörighet

Alla **SQL-exempel** ligger i gråa rutor. Databastabellerna som används i exemplen finns i databasen **UTB_Kurs1**. Nyckelorden i SQL-språket har fått fet stil i exempelrutorna av pedagogiska skäl.

Varje bit SQL-kod som kan köras självständigt kallas en **sats**, t ex en select-sats. Vissa nyckelord inleder en ny del av en SQL-sats. Dessa delar av en sats kallas för **bisats**, t ex select-bisatsen i en select-sats.

I SQL-språket är det **fritt inskrivningssätt.** Man väljer det sätt som man tycker är mest lättläst och som gör att det är lätt att ändra i koden. En SQL-sats kan man antingen skriva på en rad eller fördela på flera rader. Mellan alla ord är det valfritt om man vill lägga in blanktecken eller ny rad i SQL-koden. Antal blanktecken mellan variabler och operatorer har ingen betydelse. Oftast krävs att nyckelorden som ingår i SQL-språket kommer i rätt ordning.

Det har ingen betydelse om man skriver nyckelorden eller objektnamnen i SQL med **versaler eller gemener**, d v s med stora eller små bokstäver. Däremot är det viktigt att lösenordet vid inloggningen skrivs med versaler eller gemener.

1.5. Kommentarer i SQL-koden

Exempel

`Exempel 1` visar hur man kan skriva kommentarer efter de två bindestrecken. Detta kommentartecken måste upprepas på alla rader med kommentarer.

```
-- Detta är en
-- en kommentar
select 'Hej'   -- Här kan man också skriva en kommentar
```

`Exempel 2` visar hur man kan skriva kommentarer inneslutet mellan `/*` och `*/`. Fördelen med detta skrivsätt är att man kan fördela kommentarerna på flera rader utan att behöva upprepa kommentartecknen.

```
/* Detta är också en
kommentar */
select 'Hej'   /* Här kan man också skriva en kommentar */
```

Beskrivning av kommentarer i SQL-koden

Om man ska spara SQL-koden för framtida behov är det viktigt att man kommenterar den. Det händer ingenting om man försöker köra kommentaren. Den är bara till för människor.

Kommentarer kan läggas in i SQL-koden på två sätt:

- innesluten mellan /* ..... */
- till höger om två bindestreck: --.

2. Select-satsen

Översikt av select-satsen

```
select    Kolumn-lista
into      Till-tabell
from      Från-tabeller
where     Selektions-villkor
group by  Grupperings-kolumner
having    Grupperings-villkor
order by  Sorterings-kolumner
```

Beskrivning av select-satsen

`select` är det viktigaste kommandot i SQL och används till att ställa frågor mot databastabellerna. Dessa frågor kan bli mycket komplexa och det finns stora möjligheter att härleda fram nya variabler och producera statistik. Antalet inbyggda statistiska funktioner är inte lika stort som i t ex SAS, men tack vare att SQL är en världsstandard för databasfrågor, är det möjligt att samarbeta med många olika applikationer.

Resultatet av frågan visas i normalfallet, som en **tabell på skärmen**. Man kan även spara resultatet av frågan som en tabell i databasen eller spara det som en textfil utanför databasen.

En select-sats kan inte förstöra någon tabell i databasen. Däremot kan en dålig select-sats, som frågar mot en eller flera stora tabeller, belasta både databasservern och nätverket, så att hela systemet går långsamt.

Select-satsen kan innehålla upp till 7 bisatser samtidigt. Varje bisats inleds med ett nyckelord t ex select, from eller where. Select-satsen måste alltid innehålla en select-bisats och alla bisatser måste komma i den ordning som beskrivs högst upp på sidan.

Detta kapitel beskriver de grundläggande dragen för hur man använder de olika bisatserna. Senare kapitel tar upp mer avancerade aspekter.

2.1. Viktigaste exemplen på hur man kan använda select-satsen

Exempel

Exempel 1

Visa rader från en tabell, med hjälp av **from**.

```
select
   Kommun
   , Namn
from Kommun
```

Kommun	Namn
0114	Upplands-Väsby
0115	Vallentuna
0117	Österåker
:	:

Exempel 2

Spara rader i en ny tabell, med hjälp av **into**.

```
select
   Kommun
   , Namn
into #Kommun
from Kommun

(286 row(s) affected)
```

Exempel 3

Visa rader sorterat, med hjälp av **order by**.

```
select
   Namn
   , Kommun
from Kommun
order by Namn
```

Namn	Kommun
Ale	1521
Alingsås	1582
Alvesta	0764
:	:

Exempel 4

Visa selekterade rader enligt selekteringsvillkoret, med hjälp av **where**.

```
select
   KommunNamn = Namn
   , Inv
from Kommun
where Inv > 400000
```

KommunNamn	Inv
Stockholm	692954
Göteborg	437313

Exempel 5

Visa aggregerade rader från en tabell, med hjälp av **group by**.

```
select
  Lan
  , AntalInnevanare = sum(Inv)
from Kommun
group by Lan
```

Lan	AntalInnevanare
01	1686230
03	283006
04	259199
:	:

Exempel 6

Visa aggregerade rader från en tabell, med grupperingsvillkor, med hjälp av **having**. Jämför med Exempel 4!

```
select
  Lan
  , AntalInnevanare = sum(Inv)
from Kommun
group by Lan
having sum(Inv) > 400000
```

Lan	AntalInnevanare
01	1686230
05	411212
12	800122
14	754442
15	447949

Exempel 7

Visa unika värden för en kolumn i en tabell, med hjälp av **distinct**.

```
select distinct Lan
from Kommun
```

Lan
01
03
04
:

Exempel 8

Visa matchande rader från två tabeller, med hjälp av **join.**

```
select *
from
  Man A
  join
  Kvinna B
    on A.ID = B.ID
```

ID	Man	ID	Kvinna
4	Dag	4	Doris
5	Erik	5	Elin

Exempel 9

Visa både matchande och icke matchande rader från två tabeller, med hjälp av **full outer join** på ett matchningsvillkor.

```
select *
from
   Man A
   full outer join
   Kvinna B
      on A.ID = B.ID
```

ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL
4	Dag	4	Doris
5	Erik	5	Elin
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

*Exempel 10

Visa matchande rader från tre tabeller, med hjälp av **två joinar**.

```
select
   A.Namn
   , B.Titel
   , C.Forlag
from
   Forfattare A
   join Titlar B
      on A.ForfID=B.ForfID
   join Forlag C
      on B.ForlID=C.ForlID
order by
   A.Namn
   , B.Titel
   , C.Forlag
```

Namn	Titel	Forlag
Jan Guillou	Fiendens fiende	Prisma
Jan Guillou	Gustav	Prisma
Jan Guillou	Vendetta	Rabén & Sjögren
Liza Marklund	Gömda	Bonniers
Liza Marklund	Sprängaren	Bonniers
Liza Nilsson	Skidresan	Prisma

*Exempel 11

Visa rader från två tabeller, som inte matchar varandra på matchningsvillkoret, med hjälp **full outer join och NULL-villkor**, d v s de rader i som finns i första tabellen men inte i andra tabellen.

```
select *
from
   Man A
   full outer join
   Kvinna B
      on A.ID = B.ID
where B.ID is NULL
```

ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL

Exempel 12

Visa alla rader från bägge tabellerna, under varandra, med hjälp **union**. Likadana rader dubblettrensas automatiskt bort. **Union all** dubblettrensar ej, utan även dubbletter kommer med.

```
select
  ID
  , Namn = Man
from Man
union
select
  ID
  , Kvinna
from Kvinna
```

ID	Namn
1	Anders
2	Bo
3	Carl
4	Dag
4	Doris
5	Elin
5	Erik
6	Fia
7	Gun
8	Helen

*Exempel 13

Visa datum och tid just nu, med hjälp av funktionen **getdate()**.

```
select DatumTidNu = getdate()
```

DatumtidNu
2009-08-21 15:01:02.477

*Exempel 14

Visa en ny härledd kolumn, utifrån olika villkor, med hjälp av **case**.

```
select
  PersonNr
  , Kon =
    case
      when substring(PersonNr, 11, 1)
           in ('1', '3', '5', '7', '9') then 'Man'
      when substring(PersonNr, 11, 1)
           in ('0', '2', '4', '6', '8') then 'Kvinna'
      else '?'
    end
from Person
```

PersonNr	Kon
198012319929	Kvinna
198012318818	Man
1980123188X8	?

*Exempel 15

Visa en pivoterad tabell, med hjälp av **sum(case .. end)**.

```
Select
  KommunNamn
  , [2001] = sum(case Ar when '2001' then Inv else 0 end)
  , [2002] = sum(case Ar when '2002' then Inv else 0 end)
  , [2003] = sum(case Ar when '2003' then Inv else 0 end)
  , [2004] = sum(case Ar when '2004' then Inv else 0 end)
  , [2005] = sum(case Ar when '2005' then Inv else 0 end)
from KommunInv2001_2005
group by KommunNamn
order by KommunNamn
```

KommunNamn	2001	2002	2003	2004	2005
Ale	25593	25835	25993	26288	26405
Alingsås	35257	35327	35530	35761	36010
:	:	:	:	:	:

*Exempel 16

Visa en tabell med en avancerad selektering av rader, med hjälp av en **subquery**.

```
select *
from Lan
where Lan in
  (select Lan
   from Kommun
   group by Lan
   having count(*) = 1)
```

Lan	Namn	Areal
09	Gotlands län	3140

2.2. Select-bisatsen

Exempel

Exempel 1 visar hur select-satsen används för att göra att skriva ut ett tal på skärmen. I detta fall är **talet** en **konstant**, d vs säga den förändrar inte sitt värde till skillnad mot kolumnvariabler som man hämtar med from- bisatsen i nästa avsnitt. Resultatet blir en rad och en kolumn. Under "Resultat:" visas hur resultatet kommer att se ut på skärmen.
Fråga:

```
select 7
```

Resultat:

(No column name)
7

Exempel 2 visar hur select-satsen används för att göra att skriva ut en text på skärmen. I detta fall är även texten en konstant. Man brukar kalla sådana texter för **textsträngar**. Observera att texter alltid måste omslutas av enkelfnuttar. Enkelfnutten sitter på samma tangent som *-tangenten.

```
select 'Hej alla barn'
```

(No column name)
Hej alla barn

Exempel 3 visar hur man skriver ut flera kolumner på skärmen. Man skiljer de olika kolumnerna åt med kommatecken. Resultatet blir en rad och två kolumner.

```
select
  7
  , 'Hej alla barn'
```

(No column name)	(No column name)
7	Hej alla barn

Exempel 4 visar hur man anger **kolumnnamn** till sina kolumner. Namnen bör bara innehålla tecknen "a-z", siffror och understrykningstecknet "_". Namnet bör inte vara längre än 30 tecken. Det första tecknet bör alltid vara en bokstav.

```
select
  Antal_Veckodagar = 7
  , Halsning = 'Hej alla barn'
```

Antal_Veckodagar	Halsning
7	Hej alla barn

Exempel 5 visar ett äldre alternativt sätt att ange kolumnnamn till sina kolumner. Den ger samma resultat som ovan.

```
select
  7 as Antal_Veckodagar
  , 'Hej alla barn' as Halsning
```

`Exempel 6` visar hur select-satsen används för att göra en beräkning. Beräkningen är ett exempel på ett **uttryck**. Eftersom uttrycket bygger på konstanta värden så blir även uttrycket en konstant. Om kolumnen består av ett beräkning, så visas resultatet av beräkningen vid körningen av select-satsen.

```
select
   Antal_sekunder_pa_ett_ar = 60*60*24*365
```

Antal_sekunder_pa_ett_ar
31536000

`Exempel 7` visar hur man kan skapa uttryck även med textsträngar. Detta uttryck består av en sammanslagning av tre textsträngar. Detta kallas för **konkatenering**.

```
select Namn = 'Kalle' + ' ' + 'Anka'
```

Namn
Kalle Anka

Regler för namnsättning av kolumner

- Kolumn-namn behövs ej när man bara visar tabellen på skärmen.
- Kolumn-namn krävs när man sparar resultatet av en fråga i databasen. Dessutom måste då varje kolumn ha ett unikt namn inom tabellen.
- Namnet kan vara maximalt 128 tecken. Rekommendationen är max 30 tecken.
- Första tecknet ska vara en bokstav a-z. De övriga tecknen ska vara något av bokstäverna a-z, siffrorna 0-9 eller understrykningstecknet _. Man kan visserligen använda andra tecken, men det är inte att rekommendera eftersom det inte kommer att fungera i alla situationer. Tänk speciellt på att undvika mellanslag och ÅÄÖ i namnet.
- För att öka läsbarheten kan man använda versaler i början av varje ord, t ex AntalKommuner och/eller lägga till understrykningstecken för att ytterligare öka läsbarheten, t ex Antal_Kommuner.

Beskrivning av select-bisatsen

Syntax

```
select
  [Kolumnnamn = ] uttryck
  [, [Kolumnnamn = ] uttryck]
  [, ...]
```

Reglerna för hur man skriver koden kallas för **syntax**. SQL har en syntax baserad på engelsk-orienterade nyckelord, t ex `select`, `from` och `where`.

I **syntax-beskrivningarna** har de reserverade orden i SQL-språket fått fet stil av pedagogiska skäl. Övriga delar är kursiverade. Hakparenteserna [] anger att det är valfritt att skriva den delen av select-satsen.

Efter nyckelordet **`select`** i select-bisatsen anger man de kolumner man vill visa. Att välja ut vilka kolumner man vill visa kallas för **projektion**.

Den enklaste select-satsen innehåller bara select-bisatsen. En sådan select-sats kan bara skapa nya kolumner och visa dem på skärmen, eftersom det inte finns någon tabell att hämta kolumnerna ifrån. Resultatet av en sådan select-sats blir en tabell på skärmen med bara en rad.

2.3. From-bisatsen

Exempel

`Exempel 1` visar hur man hämtar alla kolumner från tabellen Kommun, genom att i select-bisatsen skriva `*` , som representerar alla kolumner. I from-bisatsen anger Kommun den tabell som man hämtar kolumnerna ifrån. Tabellen Kommun ligger i **databasen UTB_Kurs1.**

OBS: Tänk på att man först måste välja vilken databas man vill ställa frågan mot innan man ställer frågan.

Det gör man enklast genom att i SQL Server Management Studio välja databasen UTB_Kurs1 på verktygsraden. Denna inställning finns kvar tills man ändrar den eller avslutar SQL Server Management Studio. Tänk på att tabellen Kommun ligger i databasen UTB_Kurs1!

```
select *
from Kommun
```

Kommun	Namn	Inv	Lan	Tskatt	Kskatt	:
0114	Upplands-Väsby	35764	01	31,25	17,79	:
0115	Vallentuna	23061	01	29,48	15,89	:
0117	Österåker	31600	01	29,31	15,80	:
:	:	:	:	:	:	:

`Exempel 2` visar hur man väljer ut de kolumner man vill visa. Man räknar upp de olika kolumnerna med kommatecken emellan. Fördelen med att skriva varje kolumn på ny rad och med kommatecknet först är att det är lättare att ta bort, kopiera eller flytta kolumner i SQL-koden.

```
select
   Kommun
   , Namn
   , Inv
from Kommun
```

Kommun	Namn	Inv
0114	Upplands-Väsby	35764
0115	Vallentuna	23061
0117	Österåker	31600
:	:	:

`Exempel 3` visar hur man döper om kolumnerna i select-bisatsen genom att skriva det nya namnet först och det gamla efter likhetstecknet.

```
select
   KommunKod = Kommun
   , KommunNamn = Namn
   , AntalInnevanare = Inv
from Kommun
```

KommunKod	KommunNamn	AntalInnevanare
0114	Upplands-Väsby	35764
0115	Vallentuna	23061
0117	Österåker	31600
:	:	:

Exempel 4 visar hur man skapar två nya beräknade kolumner. KommunNamn är en sammanslagning av kommunkoden och kommunnamnet. Detta kallas för **konkatenering** och kan bara göras med text-kolumner. Man använder ett plustecken för detta och ska inte blandas ihop med plustecken vid addition. AntalInnevanare0719 är en summering av kolumnerna Inv0715 och Inv1619. Detta kan bara göras för tal-kolumner.

```
select
  KommunKodNamn = Kommun + ' ' + Namn
  , AntalInnevanare0719 = Inv0715 + Inv1619
from Kommun
```

KommunKodNamn	AntalInnevanare0719
0114 Upplands-Väsby	6235
0115 Vallentuna	4095
0117 Österåker	5736
:	:

Exempel 5 tar fram de 10 första kommunerna i tabellen kommun med hjälp av nyckelordet top utan någon sortering.

```
select top 10
  Kommun
  , Namn
from Kommun
```

Kommun	Namn
0114	Upplands-Väsby
0115	Vallentuna
0117	Österåker
0120	Värmdö
0123	Järfälla
0125	Ekerö
0126	Huddinge
0127	Botkyrka
0128	Salem
0136	Haninge

Beskrivning av from-bisatsen

Syntax

```
select [top [Antal rader] | [Antal procent percent]]
  [nytt kolumnnamn = ] kolumnnamn
  [, [nytt kolumnnamn = ] kolumnnamn]
  [, ...]
from tabell
```

Efter nyckelordet **from** anger man från vilken tabell som man hämtar sina kolumner.

I select-bisatsen räknar man upp de kolumner som man vill hämta från tabellen. Mellan kolumnerna ska det vara ett kommatecken.

Med funktionen **top** kan man visa de n första raderna eller n% första raderna.

2.4. Into-bisatsen

Exempel

Exempel 1 visar hur man skapar en #-tabell utifrån konstanta värden, d v s värdena hämtas inte från någon tabell. Den nya #-tabellen kommer att bestå av bara en rad. Observera att resultatet av frågan inte syns på skärmen, eftersom den sparas som tabell i databasen.

```
select Summa = 5 + 7
into #Summa
```

Exempel 2 visar hur
select into omges av *drop table* och *select * from.*
Observera att man inte alltid kan köra all 3 stegen på en gång.

```
drop table #Summa
go

select Summa = 5 + 7
into #Summa
go

select * from #Summa
```

Exempel 3 visar hur man kopierar en permanent tabell till en temporär tabell.

```
drop table #Kommun1
go

select *
into #Kommun1
from Kommun
go

select * from #Kommun1
```

Exempel 4 visar hur man kopierar några utvalda kolumner från en temporär tabell till en annan temporär tabell. I detta fall döper man även om kolumnerna.

```
drop table #Kommun2
go

select
   KommunKod = Kommun
   , KommunNamn = Namn
into #Kommun2
from #Kommun1
go

select * from #Kommun2
```

KommunKod	KommunNamn
0114	Upplands-Väsby
0115	Vallentuna
0117	Österåker
:	:

1. #-Tabeller

- Typ av tabell: Lokal temporär tabell
- Räckvidd: Aktuell användare inom aktuell flik
    - Den kan alltså inte ses under några andra flikar för aktuell användare
    - Den kan alltså inte av ses av någon annan användare
    - Samma användare kan alltså skapa tabeller under olika flikar med samma namn, men som är olika tabeller
    - En annan användare kan alltså skapa tabeller med samma namn, som den förste användaren
- Livslängd:
    - Skapas i en flik i Management Studio
    - Försvinner automatiskt:
        - Manuellt arbete: När fliken stängs
        - Sql-skript: När fliken stängs
        - Lagrade procedurer: När exekveringen är klar
        - Alltid: När tabellen tas bort med kommandot *drop table*
- Lagring: Sparas automatiskt i databasen tempdb. Den vanlige användaren behöver inte tänka på var #-tabellen sparas. Vill man veta vilka #-tabeller som finns på servern skriver man
  ”`select * from tempdb.sys.tables`”
- Namn: Max 116 tecken
- Användningsområde:
    - Vid exekvering av en lagrad procedur eller ett SQL-skript sparas ofta resultatet för varje steg i #-tabeller. Man avslutar med att spara slutresultatet i en permanent tabell
    - Vid manuell testning i Management Studio slipper man tänka på att städa efter sig eftersom #-tabellerna tas bort automatiskt
    - Fördelen är att tabellerna kan tas bort automatiskt
    - Dessutom kan man köra flera exekveringar samtidigt utan att #-tabellerna krockar, dock inte i samma flik
    - Om man saknar behörighet att skapa permanent tabell i aktuell databas

2. ##-Tabeller

- Typ av tabell: Global temporär tabell
- Räckvidd: För alla användare i valfri flik i Management Studio i valfri databas på samma server
    - Den kan alltså ses av alla användare på servern
    - Man kan alltså inte skapa en till ##-tabell med samma namn på samma server oavsett vem som skapar den
    - Tänk alltså på att ge tabellen ett unikt namn så att man inte krockar med andra användares namnsättning!
- Livslängd:
    - Samma som för #-tabeller
    - Undantag: ##-tabeller försvinner inte automatiskt när exekveringen av en lagrad procedur är klar. ##-tabeller försvinner först när fliken stängs där den lagrade proceduren exekverades
- Lagring: Samma som för #-tabeller
- Namn: Samma som för #-tabeller
- Användningsområde:
    - Samma som för #-tabeller
    - Ytterligare fördelar:
        - Man kan observera ##-tabeller under en exekvering i en flik från en annan flik
        - Kan ses från vilken databas som helst på servern av vilken användare som helst
        - I dynamisk SQL för att kunna fortsätta att använda ##-tabell som skapats av dynamisk SQL, efter att dynamisk SQL är exekverad

3. Permanenta tabeller

- Typ av tabell: Permanent tabell
- Räckvidd:
    - Alla användare som har behörighet till tabellen
    - Inom databasen. Ibland måste man kvalificera tabellnamnet med namn på server, databas och/eller schema. Då utökas räckvidden. **Se kapitel om kvalificering av tabellnamn!**
- Livslängd:
    - Skapas i en flik i Management Studio
    - Försvinner när tabellen tas bort med kommandot *drop table*
- Lagring: Sparas automatiskt i vald databas eller i en annan databas med hjälp av kvalificering av tabellnamnet. Vill man veta vilka tabeller som finns i aktuell databas skriver man
  ”`select * from sys.tables`”
- Namn: Max 128 tecken
- Användningsområde:
    - Om man vi spara slutresultatet i en permanent tabell
    - Om man delar tabell med andra användare
    - Vid manuell testning i Management Studio
        - Man måste tänka på att städa efter sig, eftersom permanenta tabeller inte tas bort automatiskt

Regler för alla tabeller

- Namnsättning
    - Det första tecknet i tabellnamnet bör vara en bokstav a-z
    - Övriga tecken i tabellnamnet bör vara något av tecknen bokstäverna a-z, siffrorna 0-9 samt understrykningstecknet ”_”
    - Man kan visserligen använda andra tecken, men det är inte att rekommendera eftersom det inte kommer att fungera i alla situationer t ex om något annat program arbetar mot databasen t ex SAS. Tänk speciellt på att undvika mellanslag och ÅÄÖ i namnet
    - Om man vill använda otillåtna tecken ska namnet se ut som t ex: [2012] eller [Antal år] eller [Andel i %]
    - För att öka läsbarheten kan man använda versaler i början av varje ord, t ex KopiaAvLan eller #KopiaAvLan

Beskrivning av into-bisatsen

Syntax 1

```
select [top Antal rader] | [Antal procent percent]
   [nytt kolumnnamn = ] kolumnnamn
  [, [nytt kolumnnamn = ] kolumnnamn]
  [, ...]
into ny tabell
[from gammal tabell]
```

Efter nyckelordet **`into`** anger man namnet på den nya tabellen.

Syntax 2

```
drop table Tabellnamn
```

Man kommer inte att se resultatet på skärmen efter *`select into`* , utan man får bara ett meddelande om hur många rader som skapats i den nya tabellen.

Alla kolumner måste ha ett namn och dessa namn måste vara olika. Detta behövs dock inte när man gör *`select`* utan *`into`* och resultatet bara visas på skärmen.

När man behöver ställa svårare frågor, brukar man använda *`select into #-tabell`* i flera steg och avsluta med att spara slutresultatet i en permanent tabell. Detta brukar kallas för ett SQL-skript och kan sparas som en textfil, som vilket annat dokument som helst. Man brukar avsluta filnamnet med ”.SQL”, för tydlighetens skull.

! Tips: Vid uppbyggnad av ett SQL-skript eller en lagrad procedur.

1. Rekommenderas att *`select into`* omges av *`drop table`* och *`select * from`*
2. Efter att man skapat en tabell eller tagit bort en tabell, bör man skriva *go* på en egen rad efter, för att systemtabellerna ska hinna uppdateras. Ofta brukar det fungera utan go i alla fall
3. När man är klar kan man kommentera bort dessa rader.

Se *Exempel 2* ovan!

Något att tänka på:

- Man kan inte droppa en tabell som inte finns. Vänta med att köra *`drop table`* tills det finns en tabell!
- Man kan inte skapa en tabell som redan finns. Ta då bort den gamla tabellen eller byt flik eller skapa en tabell med nytt namn
- Temporära tabeller försvinner automatiskt när man är klar. Ibland vill man dock ta bort dem tidigare, t ex om man vill skapa om tabellerna.

I kapitlet om **kvalificering av tabellnamn** längre fram i läroboken, går vi igenom hur man sparar i permanenta tabeller.

2.5. Order by-bisatsen

Exempel

Exempel 1 visar hur man sorterar efter kommunnamn i stigande ordning.

```
-- Tänk på att tabellen Kommun ligger i databasen UTB_Kurs1
select
   Namn
   , Kommun
from Kommun
order by Namn
```

Exempel 2 visar hur man sorterar efter länskod i stigande ordning och inom varje län därefter TSkatt i sjunkande ordning.

```
select
   Lan
   , TSkatt
   , Kommun
from Kommun
order by
   Lan
   , TSkatt desc
```

Exempel 3 tar fram de 10 första kommunerna i tabellen kommun, sorterat efter kommunnamn i stigande ordning

```
select top 10
  Kommun
  , Namn
from Kommun
order by Namn
```

Exempel 4 tar fram de 10 sista kommunerna i tabellen kommun, sorterat efter kommunnamn i sjunkande ordning

```
select top 10
  Kommun
  , Namn from Kommun
order by Namn desc
```

Beskrivning av order by-bisatsen

Syntax

```
select [top [Antal rader] | [Antal procent percent]]
   [nytt kolumnnamn = ] kolumnnamn
  [, [nytt kolumnnamn = ] kolumnnamn]
  [, ...]
[into ny tabell]
from gammal tabell
order by
  kolumnnamn [asc | desc]
  [, kolumnnamn [asc | desc]]
  [, ...]
```

Efter nyckelordet **`order by`** anger man den eller de kolumner som resultatet ska sorteras efter. Som standard sorteras raderna i stigande ordning. Vill man sortera i fallande ordning, ska man skriva nyckelordet ***`desc`*** (descending) efter kolumnamnet.

Observera att man kan sortera på kolumner som finns i tabellen man hämtar ifrån, men inte visar i resultatet.

2.6. Where-bisatsen

Exempel

Exempel 1a tar fram raden för Örebro kommun. Resultatet blir en rad.

```
-- Tänk på att tabellen Kommun ligger i databasen UTB_Kurs1
select *
from Kommun
where Namn = 'Örebro'
```

Exempel 1b tar fram alla kommuner som tillhör län '18'. Resultatet blir flera rader.

```
-- Tänk på att tabellen Kommun ligger i databasen UTB_Kurs1
select *
from Kommun
where Lan = '18'
```

Exempel 2 tar fram alla kommuner som inte tillhör län '18'

```
select *
from Kommun
where Lan <> '18'
```

Exempel 3 tar fram alla kommuner som har en TSkatt mindre än 27 kr

```
select *
from Kommun
where TSkatt < 27
```

Exempel 4 tar fram alla kommuner vars namn börjar på en bokstav före 'B' i alfabetet

```
select *
from Kommun
where Namn < 'B'
```

Exempel 5 tar fram alla län vars länskoder ligger i intervallet från och med '03' till och med '05'. Observera att både startvärdet och slutvärdet ingår i villkoret. Detta kallas för ett slutet intervall.

```
select *
from Lan
where Lan between '03' and '05'
```

Exempel 6 tar fram alla kommuner vars TSkatt ligger i intervallet från och med 26 till och med 27

```
select *
from Kommun
where TSkatt between 26 and 27
```

Exempel 7 tar fram alla kommuner som tillhör länen i listan ('03', '10' och '18')

```
select *
from Kommun
where Lan in ('03', '10', '18')
```

Exempel 8 tar fram alla kommuner vars namn börjar på S. Like ska bara användas för textvariabler

```
select *
from Kommun
where Namn like 'S%'
```

Exempel 9 tar fram alla kommuner vars namn slutar på S
```
select *
from Kommun
where Namn like '%S'
```

Exempel 10 tar fram alla kommuner vars namn innehåller S
```
select *
from Kommun
where Namn like '%S%'
```

Exempel 11 tar fram alla kommuner vars namn har S som andra tecken
```
select *
from Kommun
where Namn like '_S%'
```

Exempel 12 tar fram alla kommuner vars namn består av exakt tre tecken
```
select *
from Kommun
where Namn like '___'
```

Exempel 13 tar fram alla kommuner vars namn börjar på A eller Ö
```
select *
from Kommun
where Namn like '[AÖ]%'
```

Exempel 14 tar fram alla kommuner vars namn börjar på något tecken från och med Å till och med Ö
```
select *
from Kommun
where Namn like '[Å-Ö]%'
```

Exempel 15 tar fram alla kommuner vars namn endast innehåller tecknen från och med A till och med Z, och som består av exakt tre tecken
```
select *
from Kommun
where Namn like '[A-Z][A-Z][A-Z]'
```

Exempel 16 tar fram alla kommuner vars namn börjar på en bokstav från och med A till och med H och som slutar på STAD
```
select *
from Kommun
where Namn like '[A-H]%STAD'
```

Exempel 17 tar fram alla personnummer som innehåller ogiltiga tecken.
```
select *
from Person2
where PersonNr not like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
```

Exempel 18 tar fram alla kommuner som saknar uppgift om TSkatt i tabellen Kommun2
```
select *
from Kommun2
where TSkatt is NULL
```

Exempel 19 tar fram alla kommuner vars länskoder inte ligger i intervallet från och med '02' till och med '24'

```
select *
from Kommun
where Lan not between '02' and '24'
```

Exempel 20 tar fram alla kommuner som inte tillhör länen i listan ('09', '12' och '18')

```
select *
from Kommun
where Lan not in ('09', '12', '18')
```

Exempel 21 tar fram alla kommuner vars namn inte börjar på S

```
select *
from Kommun
where Namn not like 'S%'
```

Exempel 22 tar fram alla kommuner som inte saknar uppgift om TSkatt i tabellen Kommun2

```
select *
from Kommun2
where TSkatt is not NULL
```

*Exempel 23 tar fram alla län vars länskoder ligger i intervallet från och med '03' till och med '05'. Observera att både startvärdet och slutvärdet ingår i villkoret. Detta kallas för ett slutet intervall. Samma resultat som exempel 5.

```
select *
from Lan
where Lan >= '03' and Lan <= '05'
```

*Exempel 24 tar fram alla län vars länskoder ligger i intervallet mellan '03' och '05'. Observera att varken startvärdet eller slutvärdet ingår i villkoret. Detta kallas för ett öppet intervall. Jämför med exempel 5 och 23!

```
select *
from Lan
where Lan > '03' and Lan < '05'
```

*Exempel 25 tar fram alla kommuner som tillhör län '01' och som har en TSkatt mindre än 30 kr

```
select *
from Kommun
where
  Lan = '01'
  and Tskatt < 30
```

*Exempel 26 tar fram alla kommuner som tillhör län '01' eller som har en TSkatt mindre än 30 kr

```
select *
from Kommun
where
  Lan = '01'
  or Tskatt < 30
```

*Exempel 27 tar fram alla kommuner som tillhör län '01' till '02' med TSkatt under 30 kr, eller som tillhör kommungrupp 1. De två sista villkoren tas fram först innan de jämförs med första villkoret.

```
select *
from Kommun
where
   KommunGrp = '1'
   or
   Lan between '01' and '02'
   and
   TSkatt < 30
```

*Exempel 28 tar fram alla kommuner i tabellen Kommun som tillhör kommungrupp 1 eller Län '01' till '02', och har en TSkatt under 30 kr. De två första villkoren tas fram först innan de jämförs sista villkoret.

```
select *
from Kommun
where
   (
   KommunGrp = '1'
   or
   Lan between '01' and '02'
   )
   and
   TSkatt < 30
```

*Exempel 29 tar fram alla kommuner i tabellen Kommun2 som inte tillhör kommungrupp 1 eller som saknar värde. Oftast vill man få med även de poster som saknar värden när man söker med ett villkor som innebär *inte lika med.* Observera att tabellen Kommun2 tillåter NULL i kolumnen KommunGrp.

```
select *
from Kommun2
where
   KommunGrp <> '1'
   or KommunGrp is NULL
```

Beskrivning av where-bisatsen

Syntax

```
select [top [Antal rader] | [Antal procent percent]]
    [nytt kolumnnamn = ] kolumnnamn
  [, [nytt kolumnnamn = ] kolumnnamn]
  [, ...]
[into ny tabell]
from gammal tabell
where selektionsvillkor
[order by
  kolumnnamn [asc | desc]
  [, kolumnnamn [asc | desc]]
  [, ...]
]
```

Efter nyckelordet **`where`** anger man ett eller flera villkor. Dessa **villkor** kallas tillsammans för **selektionsvillkoret**. Ett selektionsvillkor talar om vilka rader i en tabell man vill välja ut att visa. Rader i en tabell brukar även kallas poster. Resultatet kallar man för en **selektion** eller ett **urval**.

När man gör en selektion (i where-bisatsen) talar man om vilka rader man vill välja ut, till skillnad mot projektion (i select-bisatsen), som talar om vilka kolumner man vill välja ut. En selectsats består oftast både av en projektion och en selektion, d v s man väljer ut både rader och kolumner ur en tabell.

Ett villkor innebär att två värden ska jämföras med varandra. Dessa värden kan bestå av konstanter, kolumn-namn eller uttryck. Komihåg att en textsträng måste omges av enkelfnuttar. Tal ska inte ha några fnuttar.

I en selectsats hämtas bara de rader som får SANT i hela selektionsvillkoret. Man brukar ibland även kalla dessa rader för sökträffarna.

Om man inte anger något selektionsvillkor visas alla rader som svar.

Where-bisatsen är mycket kraftfull i SQL och man kan göra selektioner på många olika sätt.

Enkla villkor (exempel 1-4)

Nedan visas 4 villkor som är antingen SANT eller FALSKT.

1) Två tal jämförs med varandra: 1 = 1
2) En kolumn jämförs med en textsträng: Lan = '01'
3) Två kolumner jämförs med varandra: Inv0715 > Inv1619
4) Ett uttryck jämförs med ett tal: GrundKost + GymKost > 300000

Intervall (exempel 5-6)

`where` `kolumnnamn` ***`between`*** `startvärde` ***`and`*** `slutvärde`

`between` tillhör gruppen jämförelseoperatorer som jämför värden i en kolumn med alla värden från ett startvärde till ett slutvärde. Både startvärdet och slutvärdet ska alltså ingå i villkoret.

Lista (exempel 7)

where *kolumnnamn* ***in*** *(värde1[, värde2][, ...])*

in tillhör gruppen jämförelseoperatorer som jämför en kolumn med en lista som innehåller alla värden som ska matchas.

Strängmatchning (exempel 8-17)

where *kolumnnamn* ***like*** *textmönster*

like tillhör gruppen jämförelseoperatorer som jämför ett textvärde med ett textmönster som byggs upp mellan två enkelfnuttar. Like ska bara användas för textvärden även om det ibland även fungerar för andra datayper.

Textmönstret byggs upp av valfria tecken samt symboler som har en viss funktion.

Symbolerna för wild cards d v s procent ”**%**” och understrykning ”_ ” beskriver att det i den eller de positionerna i textsträngen, ska finnas ett eller flera valfria tecken.

% Procenttecknet beskriver en sträng med noll eller flera valfria tecken
ex: ’A%’ kan matchas mot ’A’, ’apa’, ’alla barnen leker’ etc

_ Understrykningstecknet beskriver en sträng med ett valfritt tecken
ex: ’A_B%’ kan matchas mot ’apbur’, ’Arboga’ etc

[ ] Hakparenteserna ska innehålla möjliga tecken i en viss position i textsträngen
ex: ’[ABC]%[G-Z]’ kan matchas mot ’Bo’, ’apbur, ’bollen’, ’cykel’ etc

^ Icke-tecknet ger värdena innanför hakparenteserna motsatt betydelse. Tryck på SHIFT samtidigt som ^ och sedan på mellanslagstangenten.
ex: ’[^A-Z]’ kan matchas mot ’å’, ’ä’ och ’ö’

Värde saknas d v s *is NULL (exempel 18 + 22 + 29)*

where *kolumnnamn* ***is NULL***

is NULL tillhör gruppen jämförelseoperatorer som kollar om en kolumn saknar ett värde. Att värdet saknas kan t ex bero på att det är okänt eller att det inte ska finas något värde.

Man kan söka fram alla rader som saknar värde i en kolumn. Nyckelordet är **is NULL**.

Observera att NULL är inte detsamma som noll (0) eller blank (' ') utan ska tolkas som ”uppgift saknas”. Alla datatyper, t ex tal och texter, kan ha värdet NULL. Man ska inte omge NULL med enkelfnuttar. Man kan både lagra NULL i en kolumn och använda NULL i villkor.

Var vaksam om du vet att NULL är ett tillåtet värde för en kolumn. Det kan du se i objektsfönstret för respektive tabell.

SCB-rekommendationer för användning av NULL

- Sträva efter att minimera användningen av NULL vid skapandet av nya databastabeller eftersom risken för fel ökar. Ex då NULL ställer till problem:
  - `where kolumn <> 5` hittar inga NULL-förekomstn.
  - NULL kan inte finnas i en primärnyckel
  - I vissa applikationer kan det vara svårt att se skillnad på NULL och BLANK.
  - NULL kan inte indexeras och därför blir prestanda dåliga vid bearbetningar av stora datamängder.
  - Slösar med lagringsutrymmet
- Undvik NULL i alfanumeriska kod-kolumner. Byt ut NULL mot kod som betyder uppgift saknas t ex ett bindestreck '-'. Detta kan göras med t ex funktionen isnull().
- Däremot, använd NULL för numeriska värden, för att beskriva att värde saknas. Skapa aldrig en kod i en numerisk variabel för att beskriva detta.

Negationer d v s NOT (exempel 17+19-22)

where not *villkor*

not tillhör gruppen logiska operatorer och ger ett villkor motsatt betydelse, t ex `not Tal<5` är samma sak som `Tal>=5.`

Flera villkor d v s AND och OR (exempel 23-29)

and och or tillhör gruppen logiska operatorer som kombinerar flera olika villkor med varandra. Med parenteser kan man påverka i vilken prioritetsordning villkoren ska utföras.

where *villkor1* ***and*** *villkor2*

”villkor1 and villkor2” är sant om båda villkoren är sanna samtidigt.

where *villkor1* ***or*** *villkor2*

”villkor1 or villkor2” är sant om minst ett av villkoren är sant.

Operatorer

För att kunna bygga upp ett bra selektionsvillkor måste man förstå hur man använder de olika operatorerna.

Aritmetiska operatorer

I beräkningarna kan man använda sig av aritmetiska operatorer och diverse funktioner som finns i SQL-språket.

`+`	addition	t ex `Summa = 10 + 2`
`-`	subtraktion	t ex `Differens = 10 -2`
`*`	multiplikation	t ex `Produkt = 10 * 2`
`/`	division	t ex `Kvot = 10 / 2`

Jämförelseoperatorer

För att veta hur värdena ska jämföras med varandra. Observera att de olika värdena som ska jämföras med varandra måste ha samma datatyp, t ex int eller char(2). Läs mer om detta i avsnittet Datatyper**!** Versaler och gemener har exakt samma innebörd i en where-bisats. `'Bo Ek'` är lika med `'BO EK'`. Skillnaden är bara estetisk.

`=`	lika med	t ex `Namn = 'Örebro'`
`<>`	ej lika med	t ex `Namn <> 'Örebro'`
`<`	mindre än	t ex `Namn < 'B'`
`>`	större än	t ex `Namn > 'Ö'`
`<=`	mindre än eller lika med	t ex `Namn <= 'Örebro'`
`>=`	större än eller lika med	t ex `Namn >= 'Örebro'`

Ytterligare operatorer som används vid jämförelser

`between`	intervall	t ex `TSkatt between 26 and 27`
`in`	lista	t ex `Lan in ('03', '06', '09')`
`like`	strängmatchning	t ex `Namn like 'S%'`
`is NULL`	kollar om värdet saknas	t ex `TSkatt is NULL`

Logiska operatorer

Kopplas till ett eller två villkor

`and` och d v s bägge villkoren ska vara uppfyllda t ex `Lan='01' and TSkatt<30`

`or` eller d v s minst ett av villkoren ska vara uppfyllt t ex `Lan='01' or TSkatt<30`

`not` negation d v s framför villkoret en motsatt betydelse t ex `not Lan='01'` som är samma sak som `Lan<>'01'`

Prioriteringsordning i villkor

Om många operatorer används så måste man förstå i vilken ordning de utförs. De operatorer med högst prioritet är överst och utförs först i ett selektionsvillkor.

1. `()`
2. `*, /`
3. `+, -`
4. Jämförelseoperatorer och de övriga som används vid jämförelser
5. `not`
6. `and`
7. `or`

***Prestanda för selektionsvillkor*

När man ställer frågor mot stora tabeller är det viktigt att ställa frågorna på rätt sätt för att det inte ska ta för lång tid. Här följer några tips.

1. De kolumner som ingår i villkoren bör vara indexerade.
2. Undvik selektionsvillkor som innehåller <> (inte lika med)!
3. Undvik selektionsvillkor som innehåller `not` (icke)!

2.7. Group by-bisatsen

Exempel

*A. Aggregeringar utan group by-bisatsen (**Totaler**)*
(Ger en rad som resultat eftersom **`group by`** saknas)

`Exempel 1` tar fram antal rader i Kommun-tabellen d v s antal kommuner i Sverige. **`count`**(*) är en aggregeringsfunktion som beräknar antal rader i en tabell.

```
select AntalKommuner = count (*)
from Kommun

AntalKommuner
-------------
286
```

`Exempel 2` tar fram antal unika värden på Lan, d v s antal län i Sverige. Eftersom flera kommuner tillhör samma län, så blir antalet lägre än i föregående exempel.

```
select AntalLanISverige = count(distinct Lan)
from Kommun

AntalLanISverige
----------------
24
```

`Exempel 3` tar fram summa areal på alla län, d v s Sveriges areal.

```
select SverigesAreal = sum(Areal)
from Lan

SverigesAreal
---------------------------------------
410934
```

`Exempel 4` tar fram högsta skattesatsen.

```
select HogstaSkattesatsen = max(TSkatt)
from Kommun

HogstaSkattesatsen
------------------
33.47
```

`Exempel 5` tar fram lägsta skattesatsen.

```
select MinstaSkattesatsen = min(TSkatt)
from Kommun

MinstaSkattesatsen
------------------
26.47
```

`Exempel 6` tar fram medel-arealen på Sveriges län. Bägge beräkningarna ger samma resultat.

```
select
  MedelArealen = avg(Areal)
  , MedelArealen2 = sum(Areal) / count(Lan)
from Lan

MedelArealen                  MedelArealen2
----------------------------- -----------------------------
17122.250000                  17122.250000
```

Exempel 7 tar fram antal värden på TSkatt. Eftersom tabellen Kommun inte har några NULL-förekomster på TSkatt, så är detta samma sak som antal rader i tabellen.

```
select AntalSkatteSatser = count(TSkatt)
from Kommun

AntalSkatteSatser
-----------------
286
```

Exempel 8 tar fram antal värden på TSkatt. Tabellen Kommun2 har NULL-förekomster på TSkatt för vissa rader. En NULL-förekomst tas aldrig med i en beräkning, vilket medför att alla rader i tabellen inte räknas med när **count**(TSkatt) utförs.

```
select AntalSkatteSatser = count(TSkatt)
from Kommun2

AntalSkatteSatser
-----------------
276
```

Exempel 9 tar fram antal värden på TSkatt som är NULL. Eftersom count(*) räknar antal rader utan hänsyn till NULL- förekomster, så kan man beräkna antal NULL- förekomster om man kombinerar det med villkoret i where-bisatsen.

```
select AntalKommunerSomSaknarTSkatt = count(*)
from Kommun2
where TSkatt is NULL

AntalKommunerSomSaknarTSkatt
----------------------------
10
```

Exempel 10 tar fram lägsta skattesatsen i län ’01’ .

```
select MedelSkatt = min(TSkatt)
from Kommun
where Lan = '01'

MedelSkatt
----------
26.47
```

Exempel 11 beräknar flera olika värden samtidigt.

```
select
  AntalKommuner = count(*)
  , AntalUnikaLan = count(distinct Lan)
  , MedelSkatt = avg(TSkatt)
from Kommun

AntalKommuner AntalUnikaLan MedelSkatt
------------- ------------- ------------
286           24            31.646923
```

*B. Aggregeringar med group by-bisatsen (**Deltotaler**)*

(Ger en eller flera rader som resultat eftersom **`group by`** finns)

`Exempel 1` tar fram antal rader per län d v s antal kommuner per län. Observera att varje kolumn man lägger till i select-bisatsen måste även finnas i group by-bisatsen. Antal rader som visas i resultatet är lika många som antal unika förekomster av Lan.

```
select
  Lan
  , AntalKommuner = count(*)
from Kommun
group by Lan

Lan  AntalKommuner
---- -------------
01   25
03   6
04   9
:     :
```

`Exempel 2` tar fram antal kommuner per län, sorterat efter antal kommuner i fallande ordning.

```
select
  Lan
  , AntalKommuner = count(*)
from Kommun
group by Lan
order by AntalKommuner desc

Lan  AntalKommuner
---- -------------
01   25
12   20
15   18
:     :
```

`Exempel 3` tar fram medelskattesats per län, sorterat efter medelskatten.

```
select
  MedelSkatt = avg(TSkatt)
  , Lan
from Kommun
group by Lan
order by MedelSkatt

MedelSkatt         Lan
------------------ ----
29.835000          12
29.889600          01
30.284615          11
:                  :
```

Exempel 4 tar fram antal innevånare per län och kommungrupp sorterat efter län och kommungrupp.

```
select
   Lan
   , KommunGrp
   , Inv = sum(Inv)
from Kommun
group by
   Lan
   , KommunGrp
order by
   Lan
   , KommunGrp
```

```
Lan  KommunGrp Inv
---- --------- ----------
01   1         692954
01   2         839849
01   3          81489
:    :         :
```

Exempel 5 tar fram antal innevånare totalt Sverige, per län och per län och kommungrupp, sorterat efter län och kommungrupp. **Rollup-operatorn tar fram **hierarkiaggregatsrader** genom att beräkna kombinationen av kolumner på varje nivå i group by-instruktionen.

```
select
   Lan
   , KommunGrp
   , Inv = sum(Inv)
from Kommun
group by
   Lan
   , KommunGrp
   with rollup
order by
   Lan
   , KommunGrp
```

```
Lan  KommunGrp Inv
---- --------- -------
NULL NULL      8746651
01   NULL      1686230
01   1          692954
:    :         :
```

Exempel 6 tar fram antal innevånare totalt Sverige, per län, per kommungrupp och per län och kommungrupp, sorterat efter län och kommungrupp. **Cube-operatorn tar fram **superaggregatsrader** genom att använda varje möjligt kombination av kolumnerna i group by-instruktionen.

```
select
   Lan
   , KommunGrp
   , Inv = sum(Inv)
from Kommun
group by
   Lan
   , KommunGrp
   with cube
order by
   Lan
   , KommunGrp

Lan  KommunGrp Inv
---- --------- -------
NULL NULL      8746651
NULL 1         1367705
:    :         :
01   NULL      1686230
01   1          692954
:    :         :
```

*C. D**istinct** – Värdemängden för en variabel eller för en kombination av flera variabler*

`Exempel 1` tar fram alla olika (unika) länskoder i tabellen Kommun, d v s alla identiska extrarader rensas bort i resultatet. Är ett förenklat skrivsätt av aggregering utan aggregeringsberäklning.

```
select distinct UnikaLanKoder = Lan
from Kommun

-- ger samma resultat som

select UnikaLanKoder = Lan
from Kommun
group by Lan

AntalUnikaLanKoder
------------------
01
03
04
:
```

`Exempel 2` tar fram alla olika kombinationer av länskoder och kommungrupper i tabellen Kommun.

```
select distinct
  Lan
  , KommunGrp
from Kommun

-- ger samma resultat som
select
  Lan
  , KommunGrp
from Kommun
group by
  Lan
  , KommunGrp

Lan  KommunGrp
---- ----
01   1
01   2
01   3
:    :
```

`Exempel 3` **`distinct *`** tar fram alla olika rader i en tabell. Lan2 innehåller dublettrader för vissa län. Detta är ett enkelt sätt att rensa bort dublettrader i resultatet. Här sparas resultatet från tabellen Lan2 i en #-tabell.

```
select distinct *
into #Lan2_Utan_Dubletter
from Lan2
```

Beskrivning av group by-bisatsen

Syntax

Beräkning av en total

```
select
  aggregeringsberäkning
  [, ...]
from tabell
```

Gör en eller flera aggregeringsberäkningar på totalen. En aggregeringsberäkning består av en valfri kolumn som parameter i en aggregeringsfunktion. En sådan aggregering ger bara en rad i resultatet d v s resultatet är en **Total**. Resultatet av en aggregering kallas för aggregat.

Detta är den enklaste formen av aggregering eftersom **ej finns någon group by-bisats** utan består bara av en eller flera aggregeringsfunktioner i select-bisatsen .

Beräkning av en deltotal

```
select
  kolumnnamn
  [, ...]
  aggregeringsberäkning
  [, ...]
from tabell
group by
  kolumnnamn
  [, ...]
```

Gör en eller flera aggregeringsberäkningar per värde i kolumnen eller per kombination av kolumner. Den eller de kolumner som man grupperar på ska även finnas i select-bisatsen. Dessutom ska man tänka på att man inte får lägga till en kolumn i select-bisatsen, som inte finns i group by-bisatsen. Däremot får man lägga till hur många kolumner som helst innehållande aggregeringsberäkningar.

Den normala formen av aggregering har **en group by-bisats** innehållande en eller flera kolumnnamn. En sådan aggregering ger oftast flera rader i resultatet d v s resultatet är ett antal rader med **Deltotaler** grupperade på en eller flera kolumner.

Ta fram unika värden

```
select
  distinct
  kolumnnamn
  [, ...]
from tabell
```

Detta är egentligen inte ett aggregat men det påminner om group by eftersom samma sak kan åstadkommas med group by. Det blir bara lite mindre att skriva om man använder distinct.

Aggregeringsfunktioner

SQL innehåller aggregeringsfunktioner för t ex antal rader, summering, högsta och lägsta värde, medelvärde, unika rader samt ett bestämt antal översta rader. Det finns även andra funktioner men de tas inte upp här.

Aggregeringsfunktionen beräknar i normalfallet något för en hel kolumn t ex summan av alla värden i kolumnen. Istället för ett kolumnnamn kan det ibland stå ett uttryck som innehåller kolumnnamn. I andra fall aggregeras hela raderna t ex count(*). Glöm inte att sortera resultatet!

I de fall som NULL-förekomster finns i en kolumn hoppas dessa värden över. Det innebär att t ex **`count`**`(*)` och **`count`**`(TSkatt)`ger olika resultat om TSkatt har NULL-förekomster på vissa rader.

`count``(*)`	Antal rader.
`count`*`(kolumn)`*	Antal värden för en kolumn eller ett uttryck .
`count`*`(distinct kolumn)`*	Antal olika (unika) värden för en kolumn eller ett uttryck .
`sum`*`(kolumn)`*	Summa för ett en kolumn eller ett uttryck innehållande tal.
`max`*`(kolumn)`*	Högsta värde för en kolumn eller ett uttryck.
`min``(`*`kolumn`*`)`	Lägsta värde för en kolumn eller ett uttryck.
`avg`*`(kolumn)`*	Medelvärde för en kolumn eller ett uttryck innehållande tal.
`stdev`*`(kolumn)`*	Standardavikelsen för en kolumn eller ett uttryck innehållande tal.
`var`*`(kolumn)`*	Variansen för en kolumn eller ett uttryck innehållande tal.

Övriga funktioner som påverkar antal rader

`distinct` ***`kolumnlista`***	Visar alla olika (unika) kombinationer av kolumnerna
`distinct *`	Visar alla olika (unika) kombinationer av hela rader

2.8. Having-bisatsen

Exempel

`Exempel 1` tar fram alla län som innehåller fler än 15 kommuner. Having utförs efter att aggregeringen är gjord.

```
select
  Lan
  , AntalKommuner = count(*)
from Kommun
group by Lan
having count(*) > 15
```

```
Lan  AntalKommuner
---- -------------
01   25
12   20
15   18
16   17
17   16
```

`Exempel 2` gör samma sak som `Exempel 1` fast i två steg. Tycker man att Having är svår att förstå är det OK att göra det i två steg istället.

```
-- Steg 1
select
  Lan
  , AntalKommuner = count(*)
into #A
from Kommun
group by Lan

-- Steg 2
select
  Lan
  , AntalKommuner
from #A
where AntalKommuner > 15
```

`Exempel 3` tar fram alla län i tabellen Lan2 som har fler än en rad. Tabellen Lan2 innehåller dublettrader.

```
select
  Lan
  , AntalLan = count(*)
from Lan2
group by Lan
having count(*) > 1
```

```
Lan  AntalLan
---- -----------
01   2
03   2
04   3
05   3
06   2
```

Beskrivning av having-bisatsen

Syntax

```
select [top [Antal rader] | [Antal procent] percent]
  [nytt kolumnnamn = ] kolumnnamn
  [, [nytt kolumnnamn = ] kolumnnamn]
  [, ...]
  [, [nytt kolumnnamn = ] aggregeringsfunktion]
  [, ...]
[into ny tabell]
from gammal tabell
[where selektionsvillkor]
group by
  kolumnnamn
  [, kolumnnamn]
  [, ...]
having grupperingsvillkor
[order by
  kolumnnamn [asc | desc]
  [, kolumnnamn [asc | desc]]
  [, ...] ]
```

Efter nyckelordet **`having`** anger man ett eller flera grupperingsvillkor. Having används oftast ihop med group by. Man kan även ta med en where-bisats.

Databashanteraren arbetar då enligt följande logiska steg:
1. Först väljs de rader ut som ska bearbetas med hjälp av selektionsvillkoret.
2. Därefter utförs aggregeringar och beräkningar av aggregeringsfunktioner.
3. Slutligen väljs de rader ut som uppfyller de villkor som står i having-bisatsen.

3. Select-satser med flera tabeller

3.1. Join med ett-till-ett-relation

Bild

Man

ID	Man
1	Anders
2	Bo
3	Carl
4	Dag
5	Erik

Kvinna

ID	Kvinna
4	Doris
5	Elin
6	Fia
7	Gun
8	Helen

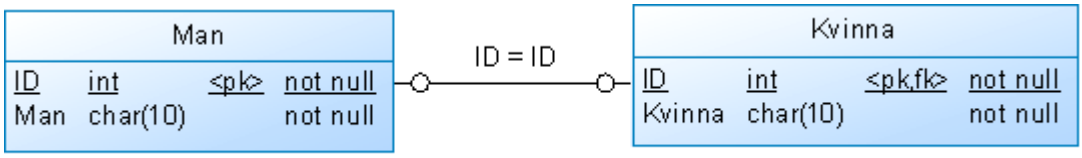
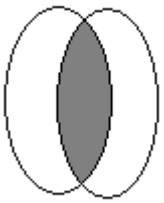


Bild 5: 1:1-relationen mellan man och kvinna

Exempel

`Exempel 1` **Inner join**. Tar fram de rader från tabellerna Man och Kvinna, där de matchar varandra på ID-begreppet. Kolumnerna från tabellerna läggs bredvid varandra. Betydelsen av ID är t ex äktenskap mellan man och kvinna.

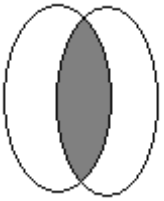
```
select *
from
  Man
  join Kvinna
    on Man.ID = Kvinna.ID
order by Man.ID
```



ID	Man	ID	Kvinna
4	Dag	4	Doris
5	Erik	5	Elin

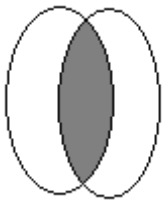
`Exempel 2` gör samma som ovan, fast tabellnamnen ersätts med de aliasnamn (förkortningar) som definieras i from-bisatsen.

```
select *
from
  Man A
  join Kvinna B
    on A.ID = B.ID
order by A.ID
```



Exempel 3 visar bara kolumnerna Man och Kvinna

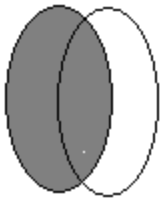
```
select
  A.Man
  , B.Kvinna
from
  Man A
  join Kvinna B
    on A.ID = B.ID
order by A.Man
```



Man	Kvinna
Dag	Doris
Erik	Elin

Exempel 4 **Left outer join.** Tar fram alla rader från tabellen Man, men bara de rader från tabellen Kvinna, som matchar på ID-begreppet. De rader som inte matchar kommer att få NULL i de kolumner som tillhör tabellen Kvinna. Tabellen Man står till vänster om join-operatorn och är därför den yttre tabellen. Tabellen Kvinna är den inre tabellen.

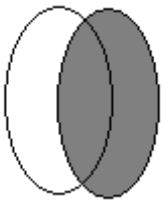
```
select *
from
  Man A
  left outer join Kvinna B
    on A.ID = B.ID
order by A.ID
```



ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL
4	Dag	4	Doris
5	Erik	5	Elin

Exempel 5 **Right outer join**. Tvärtom jämfört med exempel 4. Nu är tabellen Kvinna den yttre tabellen som alla rader hämtas ifrån.

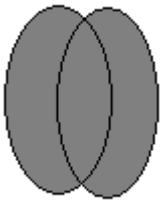
```
select *
from
  Man A
  right outer join Kvinna B
    on A.ID = B.ID
order by B.ID
```



ID	Man	ID	Kvinna
4	Dag	4	Doris
5	Erik	5	Elin
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

Exempel 6 **Full outer join**. tar fram alla rader från bägge tabellerna, både de som matchar varandra och de som inte matchar varandra. Eftersom man gör outer join mot bägge tabellerna så kallas det för full outer join.

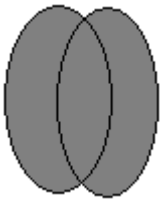
```
select *
from
  Man A
  full outer join Kvinna B
    on A.ID = B.ID
order by
  B.ID
  , A.ID
```



ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL
4	Dag	4	Doris
5	Erik	5	Elin
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

Exempel 7 **Alla kombinationer av rader mellan tabellerna**. Detta kallas för **cross join** eller **cartesisk produkt**. Resultatet blir en tabell med 5*5 d v s 25 rader. Observera att man inte har något join-villkor i en cartesisk produkt.

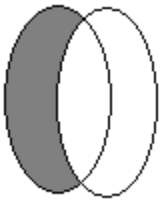
```
select *
from
  Man A
  cross join Kvinna B
order by  A.ID, B.ID
```



ID	Man	ID	Kvinna
1	Anders	4	Doris
1	Anders	5	Elin
1	Anders	6	Fia
1	Anders	7	Gun
1	Anders	8	Helen
:	:	:	:

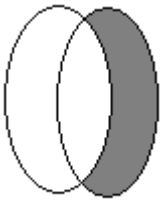
Exempel 8 **Differensen mellan två tabeller (Man – Kvinna)** d v s alla rader i första tabellen som inte matchar någon rad i andra tabellen. Detta är detsamma som de rader som efter joinen är utförd, kommer att ha NULL i andra tabellens ID-kolumn. Innebörden av denna fråga är att visa de män som är singlar. Vill man fråga **Kvinna-Man** så byter man ut "B.ID is NULL" mot "A.ID is NULL".

```
select *
from
  Man A
  full outer join Kvinna B
    on A.ID = B.ID
where B.ID is NULL
order by A.ID
```



ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL

```
select *
from
  Man A
  full outer join Kvinna B
    on A.ID = B.ID
where A.ID is NULL
order by B.ID
```



ID	Man	ID	Kvinna
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

Exempel 9 **Motsatsen till inner join** d v s visa alla rader i någon av tabellerna, som inte matchar den andra tabellen. Detta är detsamma som de rader som innehåller NULL i någon av ID-kolumnerna.

```
select *
from
  Man A
  full outer join Kvinna B
    on A.ID = B.ID
where
  A.ID is NULL
  or B.ID is NULL
order by
  B.ID
  , A.ID
```

ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

Man_Kvinna

ID	Person	Kon	GiftMed
1	Anders	M	0
2	Bo	M	0
3	Carl	M	0
4	Dag	M	5
5	Doris	K	4
6	Elin	K	7
7	Erik	M	6
8	Fia	K	0
9	Gun	K	0
10	Helen	K	0

*Exempel 10. **Self join**. Innebär att man joinar tabellen med sig själv. Om man vill visa alla personer som har samma kön som Doris, så anger man Doris namn i selektionsvillkoret och matchar köns-kolumnerna mot varandra.

```
select B.Person
from
  Man_Kvinna A
  join
  Man_Kvinna B
    on  A.Kon = B.Kon
where A.Person = 'Doris'
```

Person
Doris
Elin
Fia
Gun
Helen

*Exempel 11. **Self join**. Om man vill hitta alla äktenskap i tabellen Man_Kvinna, så matchar man kolumnen GiftMed med kolumnen ID. Genom att i where-bisatsen ange att första kolumnen ska vara en Man, undviker man att alla par nämns två gånger. Tar man bort where-bisatsen så får man 4 rader.

```
select *
from
  Man_Kvinna A
  join
  Man_Kvinna B
    on A.GiftMed = B.ID
where A.Kon = 'M'
order by  A.ID, B.ID
```

ID	Person	Kon	GiftMed	ID	Person	Kon	GiftMed
4	Dag	M	5	5	Doris	K	4
7	Erik	M	6	6	Elin	K	7

*Exempel 12. Vill man hitta alla tänkbara kombinationer av män och kvinnor, så **kombinerar man self-join med cross join**, 5*5=25 rader. Genom att i where-bisatsen ange att första kolumnen ska vara en Man och andra kolumnen ska vara en kvinna, undviker man att alla par nämns två gånger och att man kombinerar samma kön med varandra. Tar man bort where-bisatsen så får man alla kombinationer av personer d v s 10*10=100 rader.

```
select *
from
   Man_Kvinna A
   cross join Man_Kvinna B
where
   A.Kon = 'M'
   and B.Kon = 'K'
order by A.ID, B.ID
```

ID	Person	Kon	GiftMed	ID	Person	Kon	GiftMed
1	Anders	M	0	5	Doris	K	4
1	Anders	M	0	6	Elin	K	7
1	Anders	M	0	8	Fia	K	0
1	Anders	M	0	9	Gun	K	0
1	Anders	M	0	10	Helen	K	0
2	Bo	M	0	5	Doris	K	4
2	Bo	M	0	6	Elin	K	7
:	:	:	:	:	:	:	:

3.2. *Join med ett-till-många-relation

Bild

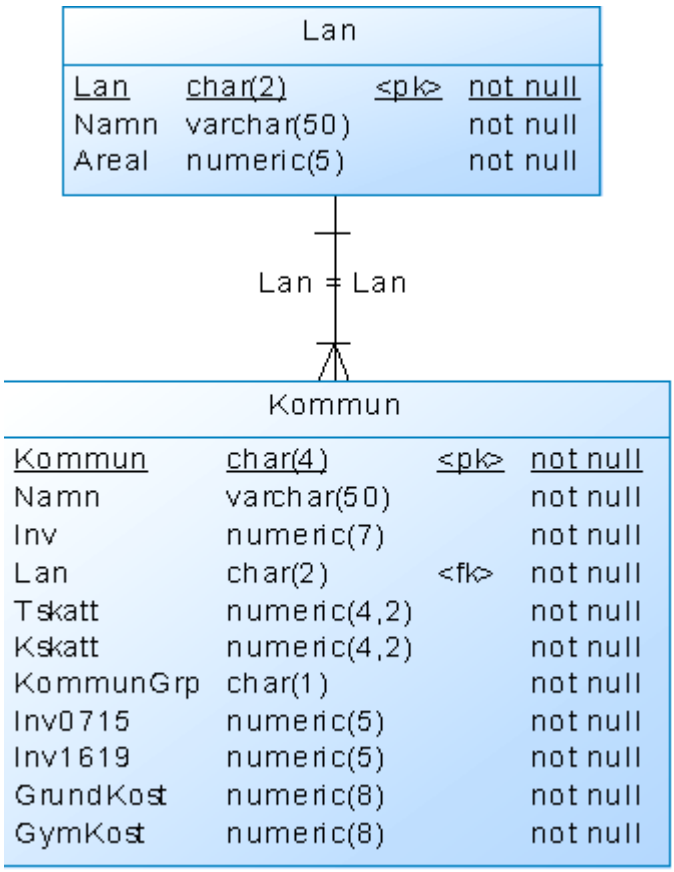


Bild 6: 1:m-relationen mellan Lan och Kommun

***Exempel**

`*Exempel 1` hämtar Namn och Areal från Lan-tabellen samt Namn och Inv från Kommun-tabellen. Tabellerna joinas på Lan-kolumnen.

```
select
    LanNamn = A.Namn
  , KommunNamn = B.Namn
  , LanAreal = A.Areal
  , KommunInv = B.Inv
from
  Lan A
  join
  Kommun B
    on A.Lan = B.Lan
order by
    LanNamn
  , KommunNamn
```

LanNamn	KommunNamn	LanAreal	KommunInv
Blekinge län	Karlshamn	2941	31462
Blekinge län	Karlskrona	2941	59753
Blekinge län	Olofström	2941	14941
:	:	:	:

3.3. *Join med ett-till-många-relation mot fler än två tabeller

Bild

Forfattare

ForfID	Namn
01	Liza Marklund
02	Jan Guillou
03	Bo Ek
04	Liza Karlsson
05	Liza Nilsson

Titlar

TitelID	Titel	ForfId	ForlId
111	Sprängaren	01	22
112	Gömda	01	22
221	Vendetta	02	23
222	Fiendens fiende	02	24
335	Gustav	02	24
733	Skidresan	05	24

Forlag

ForlID	Forlag
22	Bonniers
23	Rabén & Sjögren
24	Prisma
25	Liber

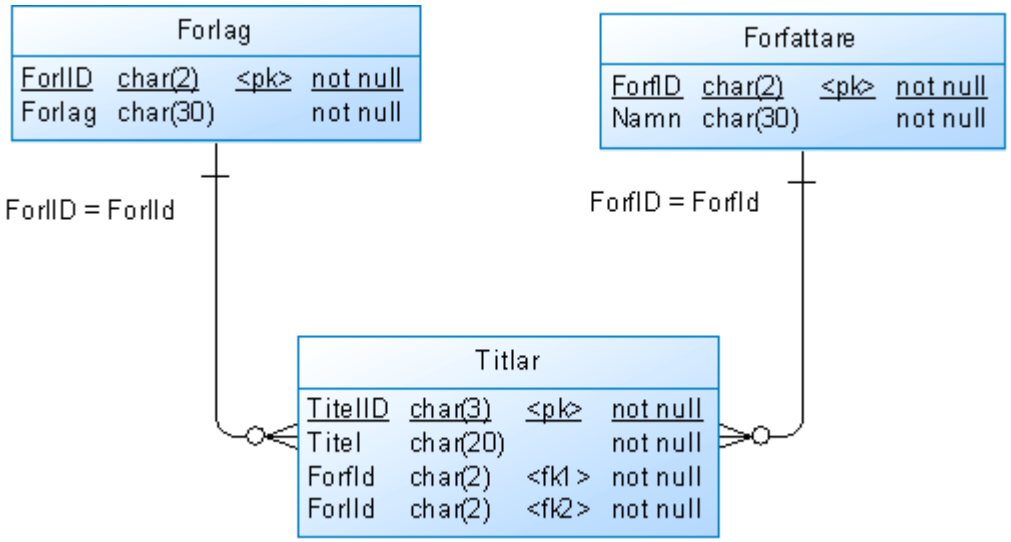


Bild 7: 1:m-relationen mellan tabellerna

***Exempel**

*Exempel 1 tar fram de rader från alla tre tabellerna som matchar varandra.

```
select
  A.Namn
  , B.Titel
  , C.Forlag
from
  Forfattare A
  join Titlar B
    on A.ForfID=B.ForfID
  join Forlag C
    on B.ForlID=C.ForlID
order by
  A.Namn
  , B.Titel
  , C.Forlag
```

Namn	Titel	Forlag
Jan Guillou	Fiendens fiende	Prisma
Jan Guillou	Gustav	Prisma
Jan Guillou	Vendetta	Rabén & Sjögren
Liza Marklund	Gömda	Bonniers
Liza Marklund	Sprängaren	Bonniers
Liza Nilsson	Skidresan	Prisma

*Exempel 2 gör en **full join mot tre tabeller**. Då kommer alla rader med från alla tabellerna.

```
select
  A.Namn
  , B.Titel
  , C.Forlag
from
  Forfattare A
  full outer join Titlar B
    on A.ForfID=B.ForfID
  full outer join Forlag C
    on B.ForlID=C.ForlID
order by
  A.Namn
  , B.Titel
  , C.Forlag
```

Namn	Titel	Forlag
NULL	NULL	Liber
Bo Ek	NULL	NULL
Jan Guillou	Fiendens fiende	Prisma
Jan Guillou	Gustav	Prisma
Jan Guillou	Vendetta	Rabén & Sjögren
Liza Karlsson	NULL	NULL
Liza Marklund	Gömda	Bonniers
Liza Marklund	Sprängaren	Bonniers
Liza Nilsson	Skidresan	Prisma

*Exempel 3 gör en **left join mot tre tabeller**. Då kommer inte raden med Forlag='Liber' med.

```
select
  A.Namn
  , B.Titel
  , C.Forlag
from
  Forfattare A
  left outer join Titlar B
    on A.ForfID=B.ForfID
  left outer join Forlag C
    on B.ForlID=C.ForlID
order by
  A.Namn
  , B.Titel
  , C.Forlag
```

Namn	Titel	Forlag
Bo Ek	NULL	NULL
Jan Guillou	Fiendens fiende	Prisma
Jan Guillou	Gustav	Prisma
Jan Guillou	Vendetta	Rabén & Sjögren
Liza Karlsson	NULL	NULL
Liza Marklund	Gömda	Bonniers
Liza Marklund	Sprängaren	Bonniers
Liza Nilsson	Skidresan	Prisma

*Exempel 4 ger samma resultat som exempel 3. Detta sätt kan ofta vara lämpligare, eftersom det blir lättare att förstå logiken genom att arbeta i mindre steg. Dessutom kan prestanda ibland bli bättre för stora tabeller.

```
drop table #A

select
  A.Namn
  , B.Titel
  , B.ForlId
into #A
from
  Forfattare A
  left outer join Titlar B
    on A.ForfID=B.ForfID

select * from #A

select
  A.Namn
  , A.Titel
  , B.Forlag
from
  #A A
  left outer join Forlag B
    on A.ForlID=B.ForlID
order by
  A.Namn
  , A.Titel
  , B.Forlag
```

3.4. Beskrivning av join

Allmänt

```
select kolumnlista
from   tabell-A  tabell-alias
       join-operator tabell-B  tabell-alias
         on join-villkor
```

t ex

```
select
  LanNamn = A.Lan
  , KommunNamn = B.Namn
from
  Lan A
  join Kommun B
    on A.Lan = B.Lan
```

LanNamn	KommunNamn
Blekinge län	Karlshamn
Blekinge län	Karlskrona
Blekinge län	Olofström
:	:

Operatorn `join` används för att hämta kolumner från två tabeller. I resultattabellen läggs de matchande kolumnerna bredvid varandra på samma rad.

Det finns 5 olika typer av joinoperatorer. Man behöver inte skriva orden *inner* och *outer* men det blir tydligare.

- inner join
- left outer join
- right outer join
- full outer join
- cross join

När man matchar två tabeller måste man använda sig av ett **joinvillkor** som beskriver hur kolumnerna ska matchas (utom för cross join). De rader som uppfyller joinvillkoret hämtas från tabellerna. Joinvillkoret skrivs efter nyckelordet **`on`**. I ett join-villkor sätter man oftast två kolumner lika med varandra t ex A.ID=B.ID.

Det rekommenderas att man i from-bisatsen definierar **aliasnamn** (namn-förkortning) för tabellnamnen. Det rekommenderas även att dessa aliasnamn används i select-bisatsen och join-villkoret för att tala om vilken tabell som kolumnen tillhör.

När man skapar en ny tabell med **`select ... into`** gäller även följande: Kolumnnamnen *måste* dessutom vara unika. Om två kolumner som kommer från olika tabeller har samma kolumnnamn, så måste man ge minst en av kolumnerna ett nytt kolumnnamn. Det räcker inte att kvalificera kolumnnamnen.

Det är lätt att göra fel när man ska joina två tabeller. Det säkraste sättet att matcha tabeller är att matcha tabeller som är kopplade med **primärnyckeln i ena tabellen mot främmande nyckeln i andra tabellen**. Det viktiga är att de kolumner som man jämför har samma innebörd. Kolumnnamnen behöver inte vara lika. De matchande kolumnerna ska även ha samma datatyp, annars får man konvertera.

Primärnyckel är den eller de kolumner som identifierar en rad i tabellen.
Främmande nyckel är en kolumn/kolumner som i en tabell som går att matcha mot en kolumn/kolumner i en annan tabell. I enklaste fallet är en kolumn/kolumner både primärnyckel och främmande nyckel, t ex om man ska joina två persontabeller där PersonNr är primärnyckel i bägge tabellerna.

Man kan matcha fler än två tabeller i samma SQL-sats, men då behöver man använda sig av flera joinoperatorer. Det rekommenderas, att man hellre joinar 2 tabeller i taget, i flera steg med, mellanlagring av resultatet i #-tabeller, än att joina alla tabellerna på en gång.

Inner join

Den enklaste och vanligaste typen av join heter **`inner join`**. När man gör en inner join så innehåller resultatet de gemensamma raderna från bägge tabellerna, d v s de rader som matchar varandra parvis, d v s de rader som joinvillkoret ger träff på. Se exempel 1-3!

Outer join

När man gör en **`outer join`** så innehåller resultatet alla rader från den yttre tabellen plus de gemensamma raderna från inre tabellen. De rader som bara finns i yttre tabellen kommer att ha NULL i de kolumner som tillhör inre tabellen.

Yttre tabell Den tabell, som alla rader hämtas från.
Inre tabell Den tabell, som de gemensamma raderna hämtas från.

`Left outer join.` Den tabell som står till vänster om join-operatorn i join-villkoret är den yttre tabellen och den tabell som står till höger är den inre tabellen. Från den vänstra tabellen hämtas då alla rader och från den högra tabellen hämtas bara de gemensamma raderna. Se exempel 4!

`Right outer join` fungerar enligt samma princip fast tvärtom. Se exempel 5!

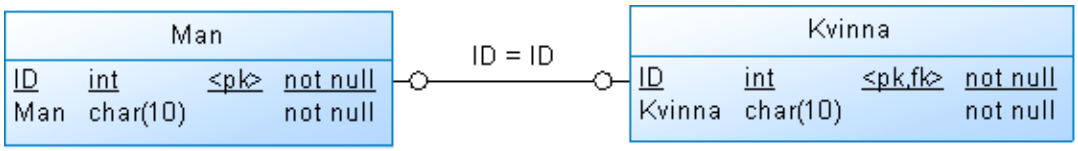
`Full outer join` hämtar alla rader från bägge tabellerna och lägger dem parvis brevid varndra precis som tidigare. Se exempel 6!

`Cross join` hämtar alla kombinationer av rader mellan tabellerna. Join-villkor får ej användas då. Se exempel 7!

Self join

Self join har ingen egen operator utan man nyttjar någon av de fem join-operatorerna och joinar tabellen med sig själv. Se exempel 10 och 11!

Beskrivning av relationen mellan tabellerna Man och Kvinna



Det råder **1 till 1–förhållande** (1:1) mellan tabellerna.

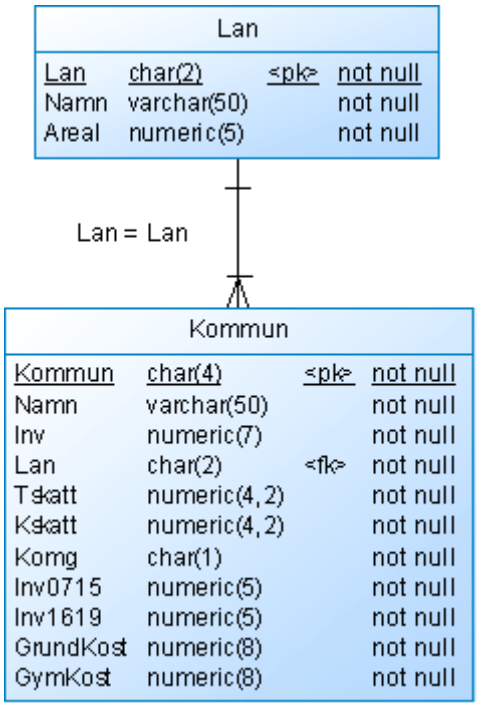
Man.ID är primärnyckel och dessutom främmande nyckel till Kvinna.ID.
Kvinna.ID är primärnyckel och dessutom främmande nyckel till Man.ID.

En man får bara vara gift med noll eller en kvinna.
En kvinna får bara vara gift med noll eller en man.

När man ska joina två tabeller är det viktigt att först förstå innebörden av tabellerna. I detta fall innebär ID-begreppen i bägge tabellerna samma sak. Betydelsen är ID för äktenskap.

Ringen betyder: Det är tillåtet för en rad i en tabell att inte få träff i den andra tabellen.

***Beskrivning av relationen mellan tabellerna Lan och Kommun**



Det råder **1 till många–förhållande** (1:M) mellan tabellerna.

Lan. Lan är primärnyckel.
Kommun.Kommun är primärnyckel.
Kommun.Lan är främmande nyckel till primärnyckeln Lan.Lan.

Tabellen Lan är förälder till tabellen Kommun.
Tabellen Kommun är barn till tabellen Lan.

Alla län innehåller en eller flera kommuner.
Alla kommuner tillhör alltid ett län.

***Beskrivning av relationen mellan tabellerna Forfattare, Titlar och Forlag**

Innehåll i tabellen Forfattare

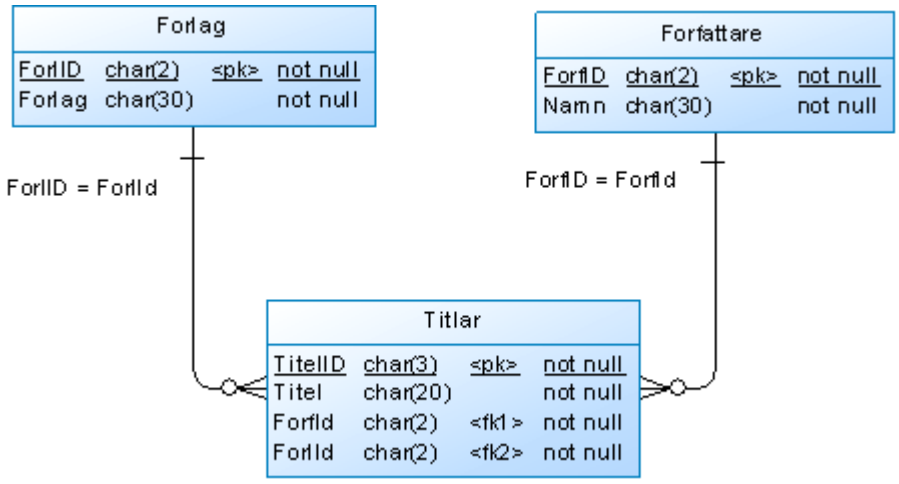
```
ForfID Namn
------ ----------------------------
01     Liza Marklund
02     Jan Guillou
03     Bo Ek
04     Liza Karlsson
05     Liza Nilsson
```

Innehåll i tabellen Titlar

```
TitelID Titel                ForfId ForlId
------- -------------------- ------ ------
111     Sprängaren           01     22
112     Gömda                01     22
221     Vendetta             02     23
222     Fiendens fiende      02     24
335     Gustav               02     24
733     Skidresan            05     24
```

Innehåll i tabellen Forlag

```
ForlID     Forlag
------     ----------------------------
22         Bonniers
23         Rabén & Sjögren
24         Prisma
25         Liber
```



Det råder **1 till många–förhållande** (1:M) mellan tabellerna.

Forfattare.ForfID är primärnyckel.
Forlag.ForlID är primärnyckel.
Titlar.TitelID är primärnyckel.
Titlar.ForfID är främmande nyckel mot primärnyckeln Forfattare.ForfID.
Titlar.ForlID är främmande nyckel mot primärnyckeln Forlag.ForlID.

Tabellen Forfattare är förälder till tabellen Titlar.
Tabellen Forlag är förälder till tabellen Titlar.
Tabellen Titlar är barn till tabellerna Forlag och Forfattare.

Alla författare har skrivit 0, 1 eller flera titlar.
Alla förlag har gett ut 0, 1 eller flera titlar.
Alla titlar har skrivits av exakt 1 författare.
Alla titlar ges ut av exakt 1 förlag.

Syntax

inner join *mellan två tabeller*

```
select kolumnlista
[into NyTabell]
from
  Tabell-A
  [inner] join Tabell-B
    on join-villkor
[where selektionsvillkor]
[group by kolumnlista]
[having grupperingsvillkor]
[order by kolumnlista]
```

outer join *mellan två tabeller*

```
select  kolumnlista
[into NyTabell]
from
  Tabell-A
  left | right | full [outer] join Tabell-B
    on  join-villkor
[where selektionsvillkor]
[group by kolumnlista]
[having grupperingsvillkor]
[order by kolumnlista]
```

cross join *mellan två tabeller*

```
select  kolumnlista
[into NyTabell]
from
  Tabell-A
  cross join Tabell-B
[where selektionsvillkor]
[group by kolumnlista]
[having grupperingsvillkor]
[order by kolumnlista]
```

self join *mellan två tabeller*

```
select  kolumnlista
[into NyTabell]
from
  Tabell-A aliasnamn-1
  join-operator Tabell-A aliasnamn-2
    [on join-villkor]
[where selektionsvillkor]
[group by kolumnlista]
[having grupperingsvillkor]
[order by kolumnlista]
```

3.5. **Outer-join med selektionsvillkor i where- eller from-bisatsen

****Exempel**

Exempel 1: **Fall-1. Selektionsvillkoret i **from**-bisatsen mot **yttre** tabellen.
Selektionen utförs före joinen.

```
select
  A.Man
  , B.Kvinna
from
  Man A
  left outer join Kvinna B
    on A.ID = B.ID
        and A.Man like '%A%'

-- Skriv på detta sätt istället! Mer lättförståeligt.

-- Steg 1
select *
into #A
from Man
where Man like '%A%'

select * from #A
```

ID	Kvinna
1	Anders
3	Carl
4	Dag

```
-- Steg 2
select B.*
into #B
from
  #A A
  join Kvinna B
    on A.ID = B.ID

select * from #B
```

ID	Kvinna
4	Doris

```
-- Steg 3
select
  A.Man
  , B.Kvinna
from
  Man A
  left outer join #B B
    on A.ID = B.ID
```

Man	Kvinna
Anders	NULL
Bo	NULL
Carl	NULL
Dag	Doris
Erik	NULL

Exempel 2: **Fall-2. Selektionsvillkoret i **from**-bisatsen mot **inre** tabellen. Selektionen utförs före joinen.

```
select
  A.Man
  , B.Kvinna
from
  Man A
  left outer join Kvinna B
    on A.ID = B.ID
        and B.Kvinna like '%E%'

-- Skriv på detta sätt istället! Mer lättförståeligt.

-- Steg 1
select
  ID
  , Kvinna
into #A
from Kvinna
where Kvinna like '%E%'

select * from #A
```

Man	Kvinna
5	Elin
8	Helen

```
-- Steg 2
select
  A.Man
  , B.Kvinna
from
  Man A
  left outer join #A B
    on A.ID = B.ID
```

Man	Kvinna
Anders	NULL
Bo	NULL
Carl	NULL
Dag	NULL
Erik	Elin

Exempel 3: **Fall-3. Selektionsvillkoret i **where**-bisatsen mot **yttre** tabellen.
Selektionen utförs efter joinen.

```
select
  A.Man
  , B.Kvinna
from
  Man A
  left outer join Kvinna B
    on A.ID = B.ID
where A.Man like '%A%'

-- Skriv på detta sätt istället! Mer lättförståeligt.

-- Steg 1
select
  A.Man
  , B.Kvinna
into #A
from
  Man A
  left outer join Kvinna B
    on A.ID = B.ID

select * from #A
```

Man	Kvinna
Anders	NULL
Bo	NULL
Carl	NULL
Dag	Doris
Erik	Elin

```
-- Steg 2
select *
from #A
where Man like '%A%'
```

Man	Kvinna
Anders	NULL
Carl	NULL
Dag	Doris

Exempel 4: **Fall-4. Selektionsvillkoret i **where**-bisatsen mot **inre** tabellen. Selektionen utförs efter joinen.
Left outer join fyller ingen funktion, eftersom det blir samma resultat med inner-join.

```
Select
  A.Man
  , B.Kvinna
from
  Man A
  left outer join Kvinna B
    on A.ID = B.ID
where B.Kvinna like '%E%'

-- Skriv på detta sätt istället! Mer lättförståeligt.

select
  A.Man
  , B.Kvinna
from
  Man A
  join Kvinna B
    on A.ID = B.ID
where B.Kvinna like '%E%'
```

Man	Kvinna
Erik	Elin

****Beskrivning**

När man gör en join, så går det även att skriva selektionsvillkor i from-bisatsen, som normalt sett bara innehåller ett joinvillkor.

Vid inner-join så spelar det ingen roll om selektionsvillkoret skrivs i from-bisatsen eller where-bisatsen. Det rekommenderas då att man skriver det i where-bisatsen.

```
select kolumnlista
from   tabell-A  tabell-alias
       join-operator tabell-B  tabell-alias
         on  join-villkor
             and selektionsvillkor
```

Är det däremot en outer-join, så får man olika resultat, beroende på var man skriver selektionsvillkoret. Tyvärr är det ganska svårt att förstå hur resultatet kommer att bli. Därför rekommenderas det att man delar upp bearbetningen i flera steg, med mellanlagring i #-tabeller.

Fall-1. Selektionsvillkoret i **from**-bisatsen mot **yttre** tabellen. Selektionen utförs före joinen.

Fall-2. Selektionsvillkoret i **from**-bisatsen mot **inre** tabellen. Selektionen utförs före joinen.

Fall-3. Selektionsvillkoret i **where**-bisatsen mot **yttre** tabellen. Selektionen utförs efter joinen.

Fall-4. Selektionsvillkoret i **where**-bisatsen mot **inre** tabellen. Selektionen utförs efter joinen. Samma resultat som inner-join.

Rekommendation: Dela alltid upp sql-koden i flera steg enligt exemplen tidigare, för att minska risken för feltolkning.

3.6. Union

Exempel

`Exempel 1` Lägger ihop två tabeller under varandra med hjälp av mängdoperatorn **`union`**. Lägger dessutom till en kolumn med konstanta värden från respektive tabell.

```
drop table #Person

select  Kon = 'M', Namn = Man
into #Person
from Man
union
select 'K', Kvinna
from Kvinna
order by Kon, Namn

select * from #Person
```

```
Kon  Namn
---  ----
K    Doris
K    Elin
K    Fia
K    Gun
K    Helen
M    Anders
M    Bo
M    Carl
M    Dag
M    Erik
```

Beskrivning av union

Syntax

```
select kolumnlista med eventuellt nya kolumnnamn
[into NyTabell]
from TabellA
union [all]
select kolumnlista
from TabellB
[order by kolumnlista]
```

Mängdoperatorn **`union`** används för att lägga ihop rader från två eller flera select-satser. Raderna i kommer att läggas under varandra.

Som standard rensas dubblettrader bort automatiskt. Vill man ha med dubbletterna anger man **`all`** efter **`union`**. **`Union all`** är snabbare än **`union`**.

Observera att…

- antalet kolumner måste vara lika i alla kolumnlistor
- motsvarande kolumner bör ha samma datatyp
- om resultatet ska få nya kolumnnamn, ska de anges i första select-satsen.
- om resultatet ska sparas i en ny tabell , ska de anges i första select- satsen.
- om resultatet ska sorteras, ska order by anges i sista select- satsen.

3.7. *Intersect och except

***Exempel**

*Exempel 1
Skapa test-tabeller i UTB_ElevO01. Välj din databas

```
use UTB_ElevO01
go

-- Tar bort tabellerna om de finns
if object_id('tempdb..#A', 'U') is not null
  drop table #A

if object_id('tempdb..#B', 'U') is not null
  drop table #B
go
-- Skapar testdata
create table #A (#A int, #A int)
insert #A select 1,1
insert #A select 2,2
insert #A select 3,3
insert #A select 4,4

create table #B(K1 int, K2 int)
insert #B select 2,0
insert #B select 3,3
insert #B select 4,4
insert #B select 5,5

-- Tabell A
select * from #A
```

K1	K2
1	1
2	2
3	3
4	4

```
-- Tabell B
select * from #B
```

K1	K2
2	0
3	3
4	4
5	5

*Exempel 2
Mängdoperatorn **intersect**. Visar de unika rader som är exakt lika i bägge tabellerna.

```
select * from #A
intersect
select * from #B
```

K1	K2
3	3
4	4

*Exempel 3
Mängdoperatorn **intersect**. Visar de unika värden i kolumn K1 som är exakt lika i bägge tabellerna.

```
select K1 from #A
intersect
select K1 from #B
```

K1
2
3
4

*Exempel 4
Mängdoperatorn **except**. Visar de unika rader som finns i #A men inte i #B.

```
select * from #A
except
select * from #B
```

K1	K2
1	1
2	2

*Exempel 5
Mängdoperatorn **except**. Visar de unika rader som finns i #B men inte i #A.

```
select * from #B
except
select * from #A
```

K1	K2
2	0
5	5

*Exempel 6
Mängdoperatorn **except**. Visar de unika värden i kolumn K1 som finns i #A men inte i #B.

```
select K1 from #A
except
select K1 from #B
```

K1
1

*Exempel 7

Använd databas UTB_Kurs1. Visa identiska rader från två tabeller, med hjälp **intersect**,
d v s de hela unika rader som finns i bägge tabellerna.

```
select *
from Kommun2
intersect
select *
from Kommun
```

Kommun	Namn	Inv	Lan	Tskatt	Kskatt	KommunGrp	Inv0715	Inv1619	GrundKost	GymKost
0114	Upplands-Väsby	35764	01	31.25	17.79	2	4062	2173	240663	86339
0115	Vallentuna	23061	01	29.48	15.89	2	2825	1270	139852	57157
0117	Österåker	31600	01	29.31	15.80	2	3796	1940	175995	66359
0120	Värmdö	25193	01	30.77	17.29	2	2724	1234	127994	47471
0123	Järfälla	57638	01	30.58	17.35	2	6511	3290	288919	54443
:	:	:	:	:	:	:	:	:	:	:

*Exempel 8

Visa unika ej identiska rader från två tabeller, med hjälp **except**, d v s de rader som finns i första tabellen, men inte i andra tabellen.

```
select *
from Kommun2
except
select *
from Kommun
```

Kommun	Namn	Inv	Lan	Tskatt	Kskatt	KommunGrp	Inv0715	Inv1619	GrundKost	GymKost
0604	Aneby	7269	06	NULL	18.25	NULL	986	398	49471	12684
0764	Alvesta	19813	07	NULL	18.75	NULL	2465	1121	111771	51627
1521	Ale	25204	15	NULL	20.00	NULL	2968	1523	142975	56810
:	:	:	:	:	:	:	:	:	:	:

**Exempel 9

Visa de rader ifrån bägge tabellerna, som bara finns i minst en av tabellerna. Raderna är kompletterade med en kolumn som anger vilken tabell som är källan. Dessutom är de sorterade efter primärnyckeln i varje tabell, så att man lättare kan se vad som skiljer tabellerna. Här ser vi även hur man kan jämföra en kolumn mellan två tabeller.

```
select *
into #K1
from Kommun
except
select *
from Kommun2

select *
into #K2
from Kommun2
except
select *
from Kommun

-- Jämför hela raderna
select Kalla='Kommun', *
from #K1
union
select Kalla='Kommun2', *
from #K2
order by Kommun, Kalla
```

Kalla	Kommun	Namn	Inv	Lan	Tskatt	Kskatt	KommunGrp	Inv0715	Inv1619	GrundKost	GymKost
Kommun	0604	Aneby	7269	06	31.35	18.25	6	986	398	49471	12684
Kommun2	0604	Aneby	7269	06	NULL	18.25	NULL	986	398	49471	12684
Kommun	0764	Alvesta	19813	07	31.52	18.75	8	2465	1121	111771	51627
Kommun2	0764	Alvesta	19813	07	NULL	18.75	NULL	2465	1121	111771	51627
	:	:	:	:	:	:	:	:	:	:	:

```
-- Jämför en kolumn
select A.Kommun, TSkatt1 = A.TSkatt, TSkatt2 = B.TSkatt
from #K1 A join #K2 B on A.Kommun = B.Kommun
```

Kommun	TSkatt1	TSkatt2
0604	31.35	NULL
0764	31.52	NULL
1521	32.22	NULL
1582	32.76	NULL
:	:	:

****Beskrivning av intersect och except**

Syntax

```
select kolumnlista
[into NyTabell]
from TabellA
[where selektionsvillkor]
[group by kolumnlista]
[having grupperingsvillkor]
intersect | except
select kolumnlista
from TabellB
[where selektionsvillkor]
[group by kolumnlista]
[having grupperingsvillkor]
[order by kolumnlista]
```

Intersect är och **except** är mängdoperatorer precis som **union**. Det innebär att de används för att låta tabeller jämföras med varandra utan att behöva något vilkor mellan kolumnerna som vid join. De fungerar som union med följande skillnader

- **Intersect** hämtar de rader som är lika mellan delfrågorna
- **Except** hämtar de rader som skiljer sig mellan delfrågorna. Här är ordningsföljden viktig, d v s hämta alla rader från första delfrågan som inte finns i andra delfrågan.

Bägge delfrågorna måste innehålla lika många kolumner eftersom de jämförs i den ordning de kommer i.

Observera att man inte behöver ta med alla kolumner i tabellerna. Tar man med färre kolumner i jämförelsen är chansen större att man får fler rader i resultatet för intersect och färre rader i resultatet för except.

Fördel med **intersect** och **except**

- Man behöver inte skriva joinvillkor. Detta är speciellt smidigt om man vill jämföra många kolumner mellan två delfrågor. Det blir mindre att skriva.

Fördel med **join**

- Man kan visa andra kolumner än de som matchas mellan delfrågorna.
- Man kan skapa nya härledda kolumner i frågan som inte påverkar matcningen mellan tabellerna.
- Bätte prestanda med join under förutsättning att kolumnerna som man joinar med, är indexerade.
- Man kan visa även dubblettraderna. Till skillnad från *union* kan inte *intersect* och *except* visa även dubblettraderna. *Union* kan ju skrivas som *union all*.

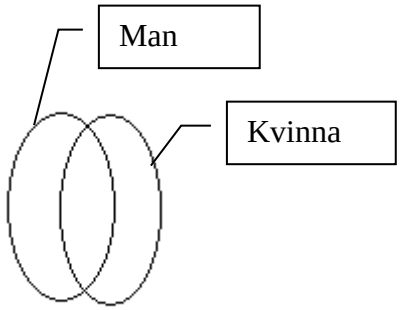
Mängdoperatorer

Jämförelser mellan select-satser

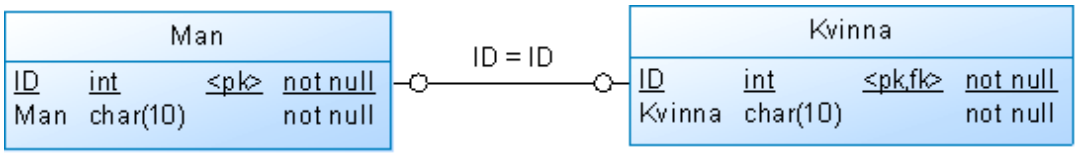
union	Hämtar alla unika rader från tabellerna (A *union* B). *Union all* ger även dubblettraderna.
intersect	Hämtar gemensamma rader från tabellerna (A snitt B)
except	Hämtar rader som är skillnaden från tabellerna (A minus B)

3.8. *Sammanfattning

Venndiagram



Datamodell



Tabellen Man matchar på 0 eller 1 rad i tabellen Kvinna.
Tabellen Kvinna matchar på 0 eller 1 rad i tabellen Man.

Tabeller

Man

ID	Man
1	Anders
2	Bo
3	Carl
4	Dag
5	Erik

Kvinna

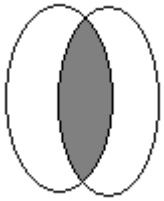
ID	Kvinna
4	Doris
5	Elin
6	Fia
7	Gun
8	Helen

Man kan kombinera tabeller i SQL på två sätt:

- **Union**
- **Join**

Exempel

1) Matchande rader från Man och Kvinna (snittet mellan Man och Kvinna).



```
select A.ID, A.Man, B.ID, B.Kvinna
from  Man A join Kvinna B
      on A.ID = B.ID
```

ID	Man	ID	Kvinna
4	Dag	4	Doris
5	Erik	5	Elin

Specialfallet av en **inner join** är en **self join**. Då matchas en tabell mot sig själv.

```
select A.ID, A.Man, B.ID, B.Man
from  Man A join Man B
      on A.ID = B.ID
```

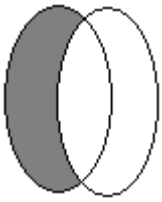
ID	Man	ID	Man
1	Anders	1	Anders
2	Bo	2	Bo
3	Carl	3	Carl
4	Dag	4	Dag
5	Erik	5	Erik

Jämförelse med intersect

```
select ID from Man
intersect
select ID from Kvinna
```

ID
4
5

2) Rader från Man som inte matchar Kvinna (differensen mellan Man och Kvinna).



```
select A.ID, A.Man, B.ID, B.Kvinna
from  Man A left outer join Kvinna B
      on A.ID = B.ID
where B.ID is NULL
```

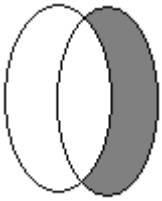
ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL

Jämförelse med intersect

```
select ID from Man
intersect
select ID from Kvinna
```

ID
1
2
3

3) Rader från Kvinna som inte matchar Man (differensen mellan Kvinna och Man).



```
select A.ID, A.Man, B.ID, B.Kvinna
from  Man A right outer join Kvinna B
      on A.ID = B.ID
where A.ID is NULL
```

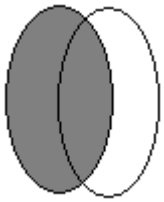
ID	Man	ID	Kvinna
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

Jämförelse med intersect

```
select ID from Kvinna
except
select ID from Man
```

ID
6
7
8

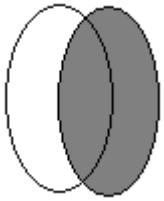
4) Alla rader från Man plus de rader från Kvinna som matchar.



```
select A.ID, A.Man, B.ID, B.Kvinna
from  Man A left outer join Kvinna B
      on A.ID = B.ID
```

ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL
4	Dag	4	Doris
5	Erik	5	Elin

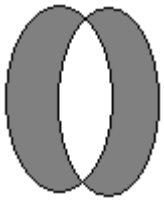
5) Alla rader från Kvinna plus de rader från Man som matchar.



```
select A.ID, A.Man, B.ID, B.Kvinna
from  Man A right outer join Kvinna B
      on A.ID = B.ID
```

ID	Man	ID	Kvinna
4	Dag	4	Doris
5	Erik	5	Elin
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

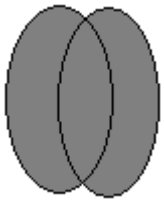
6) Ej matchande rader från Man och Kvinna.



```
select A.ID, A.Man, B.ID, B.Kvinna
from  Man A full outer join Kvinna B
      on A.ID = B.ID
where
  A.ID is NULL
  or B.ID is NULL
```

ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

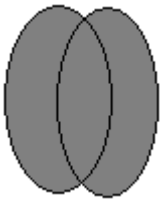
7) Alla rader från Man och Kvinna med kolumnerna bredvid varandra.



```
select A.ID, A.Man, B.ID, B.Kvinna
from  Man A full outer join Kvinna B
      on A.ID = B.ID
```

ID	Man	ID	Kvinna
1	Anders	NULL	NULL
2	Bo	NULL	NULL
3	Carl	NULL	NULL
4	Dag	4	Doris
5	Erik	5	Elin
NULL	NULL	6	Fia
NULL	NULL	7	Gun
NULL	NULL	8	Helen

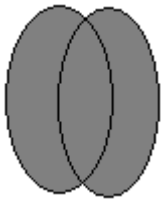
8) Alla kombinationer av rader från Man och Kvinna med kolumnerna bredvid varandra (kartesisk produkt mellan Man och Kvinna).



```
select A.ID, A.Man, B.ID, B.Kvinna
from Man A cross join Kvinna B
```

ID	Man	ID	Kvinna
1	Anders	4	Doris
2	Bo	4	Doris
3	Carl	4	Doris
4	Dag	4	Doris
5	Erik	4	Doris
1	Anders	5	Elin
2	Bo	5	Elin
3	Carl	5	Elin
4	Dag	5	Elin
5	Erik	5	Elin
1	Anders	6	Fia
2	Bo	6	Fia
3	Carl	6	Fia
4	Dag	6	Fia
5	Erik	6	Fia
1	Anders	7	Gun
2	Bo	7	Gun
3	Carl	7	Gun
4	Dag	7	Gun
5	Erik	7	Gun
1	Anders	8	Helen
2	Bo	8	Helen
3	Carl	8	Helen
4	Dag	8	Helen
5	Erik	8	Helen

9) Alla rader från Man och Kvinna med kolumnerna under varandra.



```
select ID, Namn = Man
from Man
union
select ID, Kvinna
from Kvinna
```

ID	Namn
1	Anders
2	Bo
3	Carl
4	Dag
4	Doris
5	Elin
5	Erik
6	Fia
7	Gun
8	Helen

Notera!
- **Union** ger dubletteliminering av rader
- **Union all** ger alla rader

4. Mer om select-satsen

4.1. Kvalificering

Exempel

`Exempel 1` visar hur man skapar två tabeller med samma namn, men olika scheman. För att vara säker på vilken tabell man frågar emot, måste man ange schemanamn framför tabellnamnet. Schema DBO finns i alla SCB:s databaser. För att man ska kunna använda ett annat schema än DBO, måste det skapas i databasen innan, av en användare med rätt behörighet.

Tänk på att byta ut databas-namnet `UTB_ElevO01` i alla exempel mot din egen elevdatabas.

```
use UTB_ElevO01
go

drop table dbo.Summa
go

select Summa = 5 + 7
into UTB_ElevO01.dbo.Summa

drop table scbtole.Summa
go

select Summa = 10 + 15
into scbtole.Summa

select * from dbo.Summa
select * from scbtole.Summa
select * from Summa   -- Väljer default schema
```

`Exempel 2` visar hur man skapar en tabell utan att ange schema. Tabellen får då automatiskt det schema som är *default schema* för användaren i den databasen. I de flesta databaser är **dbo** default schema för alla användare. I detta exempel är DBO default schema för användaren och tabellen som skapas och anropas kommer då automatiskt att vara **dbo.Summa.**

```
use UTB_ElevO01
go

drop table Summa
go

select Summa = 2 + 4
into Summa

select * from Summa
select * from dbo.Summa
```

`Exempel 3` visar hur man kopierar en tabell från en databas till en annan. I exemplet är tabellnamnet kvalificerat som *DATABAS.SCHEMA.TABELL*. Fördelen är att man kan kopiera tabellen mellan dessa databaser, samt anropa tabellen oavsett vilken databas man står i.

OBS: Undvik att skriva ”..” istället för DBO, eftersom det betyder defaultschema. Om man byter defaultschema för en person i en databas kommer den andra select-satsen nedan att peka på fel tabell.

```
drop table UTB_ElevO01.dbo.Lan
go

select *
into UTB_ElevO01.dbo.Lan
from UTB_Kurs1.dbo.Lan

select * from UTB_ElevO01.dbo.Lan order by Lan
select * from UTB_ElevO01..Lan order by Lan
```

Beskrivning av kvalificering

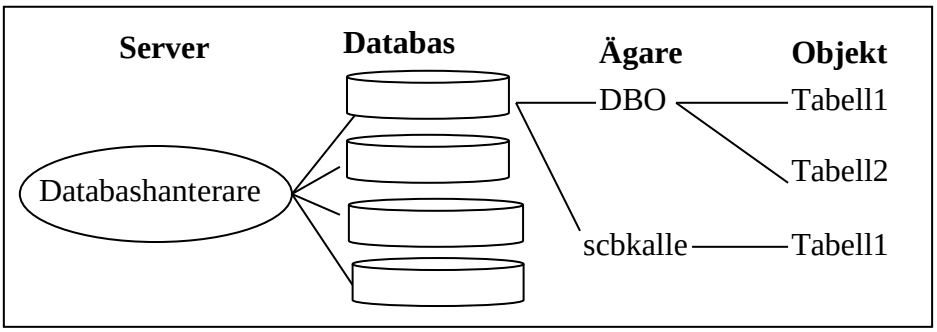


Bild 8: En databasserver och dess innehåll.

Hittills har vi endast sparat data i #-tabeller. Dessa tabeller sparas automatiskt i databasen **TEMPDB**. Tyvärr kan man inte se tabellnamnen där, utan bara anropa sina egna #-tabeller, som man sparat där. Om man vill spara tabellerna på längre sikt, så ska man inte använda #-tabeller, utan *permanenta tabeller*. Då måste man dessutom tänka på vilket *schema* som tabellen ska ha samt i vilken *databas* den ska ligga i.

I varje databas måste alla tabeller ha ett schema. Om man inte anger schema när man skapar en tabell, så blir schemat oftast DBO, med automatik. Det beror på att de flesta SQL-servrarna på SCB är inställda så (default schema är DBO). När man skapar nya databastabeller eller läser från tabeller, bör man alltid för säkerhets skull, ange schemanamnet framför tabellnamnet. Detta kallas för att kvalificera tabellerna och blir tabellens unika namn i databasen. Om man inte anger schema när man hämtar en tabell, så finns risken att man hämtar fel tabell. Flera schema kan ha samma namn på sina tabeller.

Om man enbart arbetar inom en databas behöver man inte kvalificera databasnamnet. Tänk dock på att den select-satsen bara fungerar när du står i rätt databas. Sparar man SQL-kod för framtida behov bör man dock alltid se till att tabellerna är kvalificerade med både schema och databasnamn, eftersom det är så lätt att glömma att stå i rätt databas nästa gång.

Man måste på något sätt tala om vilken databas som tabellen ligger i. Det finns 3 olika sätt.

1. Välj rätt databas i databasrutan på verktygsraden i SQL Server Management Studio. Detta sätt använder vi oftast på kursen för att slippa skriva så mycket.
2. Ge kommandot **`use`** <Databasnamn>. På raden efter **`use`** skriver man kommandot **`go`** om man vill kunna köra use-kommandot och select-satserna samtidigt. Om man kör man dem var för sig, så behövs inte go. När man har kört kommandot så är databasen vald ända tills man byter databas. Observera att innehållet i databasrutan ändras.
3. Kvalificera tabellnamnet med <databasnamn punkt DBO punkt> före tabellnamnet t ex **`UTB_Kurs1.DBO.Kommun`**. Detta sätt är att rekommendera för alla SQL-kod som man sparar i en skriptfil.

Om man vill kopiera data mellan olika servrar kan man göra detta genom att dessutom kvalificera med servernamn t ex **`Utbildning`**`.UTB_Kurs1.DBO.Kommun`.
För att detta ska fungera, krävs det att servrarna är länkade.
Detta görs av DBA-funktionen, beställning av länkad server sker via Servicedesk.
Utbildningsservern är inte länkad till någon annan server av säkerhetsskäl.

Behörighet

För att kunna arbeta mot en databas och dess tabeller måste man ha rätt behörighet.

Om en användare inte ska skapa och förändra data i en databas utan bara kunna ställa frågor, så måste han ha rollen **db_datareader** (läsrättigheter).

Om användare dessutom ska kunna skapa och förändra data i en databas, så måste han dessutom ha rollen **db_datawriter** (skrivrättigheter).

Om användaren dessutom ska kunna skapa och ta bort tabeller, så måste han dessutom ha rollen **SCBPowerUser**.

Rollen **db_owner** (databasägare) krävs för att kunna skapa vissa databasobjekt, t ex scheman och roller. Användare i rollen **db_owner** måste vara godkänd av enhetschef (informationsägare).

Beställning av behörighet i SQL-databaser sker via BBQ-applikationen.

http://inblick.scb.intra/Stod/it-miljo/bestallning-av-behorighet-i-sql-server/

4.2. Decimaltal och avrundning

Exempel

Exempel 1. **Beräkning av division mellan två heltal** blir alltid ett heltal. Heltal har datatypen int.

```
select
  Kvot1 = 1 / 7
  , Kvot2 = 10 / 7
  , Kvot3 = 100 / 7
```

Kvot1	Kvot2	Kvot3
0	1	14

Exempel 2. Om man vill ha **decimaler i resultatet**, så måste något av de **ingående talen i beräkningen vara definierat som ett decimaltal**. Genom att lägga till en decimal till ett av talen, så blir det talet definierat som ett decimaltal. Resultatet av en division består alltid av minst 6 decimaler. Antalet decimaler i resultatet beror på största antal siffror i som finns i ett tal i beräkningen.

```
select
  Kvot1 = 1.0 / 7
  , Kvot2 = 10.0 / 7
  , Kvot3 = 100.0 / 7
```

Kvot1	Kvot2	Kvot3
0.142857	1.428571	14.285714

Exempel 3 visar hur **noggrannheten i beräkningen** ökar beroende på vilken datatyp som är definierad i ingångsdata.

```
select
  Kvot1 = cast(1 as numeric(1, 0)) / 7
  , Kvot2 = cast(1 as numeric(38, 37)) / 7
```

Kvot1	Kvot2
0.142857	0.14285714285714285714285714285714285714

Exempel 4. Efter beräkningen är klar konverterar man till så många decimaler som man vill presentera. Här syns tydligt att noggrannheten i resultatet bli bättre när man ökar noggrannheten från början.

```
select
    Ber1 = cast(1 as numeric(1, 0)) / 7 * 1000000
  , Ber2 = cast(1 as numeric(38, 37)) / 7 * 1000000
into #A

select
    Ber1 = cast(Ber1 as numeric(8, 2))
  , Ber2 = cast(Ber2 as numeric(8, 2))
from #A
```

Ber1	Ber2
142857.00	142857.14

`Exempel 5.` Avrundning av resultatet sker bäst med funktionen **`round(Tal, Precision).`** Parametern efter talet talar om hur många decimaler som talet ska avrundas till. Är parametern negativ så sker en avrundning även till vänster om decimalpunkten. Observera att datatypen inte förändras d v s antalet decimaler är fortfarande lika många som från början. Märk även att avrundning med avslutande femma alltid sker uppåt.

```
select
  Tal1 = round(123.35, 1)
  , Tal2 = round(123.45, 1)
  , Tal3 = round(123.45, 0)
  , Tal4 = round(123.45, -1)
  , Tal5 = round(123.45, -2)
  , Tal6 = round(123.45, -3)
```

Tal1	Tal2	Tal3	Tal4	Tal5	Tal6
123.40	123.50	123.00	120.00	100.00	0.00

`Exempel 6.` **Avrundning med `Floor(Tal)`** sker till **närmast lägre heltal**. `Floor()` kan bara avrundas till heltal.

```
select
  Tal1   = floor(1.4)
  , Tal2 = floor(-1.4)
```

Tal1	Tal2
1	-2

`Exempel 7.` **Avrundning med `ceiling(Tal)`** sker till **närmast högre heltal**. `ceiling()` kan bara avrundas till heltal.

```
select
  Tal1   = ceiling (1.4)
  , Tal2 = ceiling (-1.4)
```

Tal1	Tal2
2	-1

Beskrivning av decimaltal och avrundning

Syntax

funktionen cast()
```
cast(uttryck as datatyp)
```

När man hanterar tal så ska datatypen vara något av tal-datatyperna t ex tinyint, smallint, int, bigint eller numeric().

funktionen round()
```
round(uttryck, avrundningsposition [, 0|1])
```

Avrundningspositionen anger normalt vid vilken decimal avrundningen ska ske. Om värdet är negativt så sker även avrundning till vänster om decimaltecknet d v s på tiotal, hundratal o s v.

Sista parametern som inte är obligatorisk kan vara 0 eller 1. 0 är default och betyder avrundning. 1 betyder trunkering d v s värdet avrundas inte utan ersätts med nollor från och med den position som anges.

I SQL används decimalpunkt istället för decimalkomma. Vill man visa resultatet med decimalkomma så kan man t ex kopiera det till MS Excel och formatera det där.

4.3. * Identity

***Exempel**

Exempel 1. Demo av hur man använder funktionen **identity(int)**

```
drop table #Test
-- Skapar en tabell med automatiskt radnummer
-- Kan bara användas i "select into"
select RadNr = identity(int), Tecken = 'X'
into #Test

-- I fortsättningen skapas radnummer automatiskt
insert #Test select Tecken='Y'
-- Titta på hela tabellen
select * from #Test order by RadNr
```

RadNr	Tecken
1	X
2	y

Exempel 2. Demo av hur man använder **identity** i create table.

```
drop table _Test
-- Skapar en tabell med automatiskt radnummer
create table _Test
(
  RadNr int identity
, Tecken char(1)
)

-- Alla radnummer skapas automatiskt
insert _Test select 'X'
insert _Test select 'Y'
-- Titta på hela tabellen
select * from _Test
```

RadNr	Tecken
1	X
2	Y

Exempel 3. Demo av hur man hämtar senaste posten som insertats i en session. Om en andra session senare gör insert, så påverkas inte värdet av @@identity för den första sessionen.

```
-- (Fortsättning av förra exemplet)
-- Visa den senaste posten
select * from _Test where RadNr = @@identity
```

RadNr	Tecken
2	Y

```
-- Starta en ny session genom att öppna en ny flik i
SQL Server Management Studio
-- Lägg till en ny post i den nya sessionen
insert _Test select 'Z'
-- Kolla att andra sessionens senaste post visas
select * from _Test where RadNr = @@identity
```

RadNr	Tecken
3	Z

```
-- Hoppa tillbaka till den första sessionen
-- Kolla att den första sessionens senaste post
fortfarande visas
select * from _Test where RadNr = @@identity
```

RadNr	Tecken
2	Y

```
-- Titta på hela tabellen
select * from _Test
```

RadNr	Tecken
1	X
2	Y
3	Z

4.4. *Funktioner och systemtabeller

***Exempel**

**Strängfunktioner*

Skapar test-tabeller.

```
select Mening = 'Ni talar bra latin'
into #Latin

create table #Trim (T1 char(5), T2 varchar(5))
insert #Trim select ' 123'  , ' 123'
insert #Trim select ' 123 ' , ' 123 '

select NyRad = 'GOD' + char(13) + char(10) + 'JUL'
into #GODJUL

create table #LetaLitenBokstav (Bokstav char(1))
insert #LetaLitenBokstav select 'A'
insert #LetaLitenBokstav select 'a'
insert #LetaLitenBokstav select 'Z'
insert #LetaLitenBokstav select 'z'
```

*Exempel 1. left(), substring() och right()

Plockar ut ett antal tecken från vänster, höger eller mitt i textsträngen.

```
select
    Vanster = left('Ni talar bra latin', 2)
  , Mitten = substring('Ni talar bra latin', 4, 5)
  , Hoger = right('Ni talar bra latin', 5)

select
    Vanster = left(Mening, 2)
  , Mitten = substring(Mening, 4, 5)
  , Hoger = right(Mening, 5)
from #Latin
```

```
Vanster Mitten Hoger
------- ------ -----
Ni      talar  latin
```

*Exempel 2. replace() och stuff()

Ytterligare exempel på hur man kan byta ut delar i en textsträng mot något annat.

```
select
    Mening1 = replace('Ni talar bra latin', 'talar', 'pratar')
  , Mening2 = stuff('Ni talar bra latin', 4, 5, 'pratar')

select
    Mening1 = replace(Mening, 'talar', 'pratar')
  , Mening2 = stuff(Mening, 4, 5, 'pratar')
from #Latin
```

```
Mening1             Mening2
------------------  ------------------
Ni pratar bra latin  Ni pratar bra latin
```

ASCII-tabell

Kod	Tecken	Kod	Tecken	Kod	Tecken	Kod	Tecken	Kod	Tecken
32		81	Q	130	‚	179	³	228	ä
33	!	82	R	131	ƒ	180	´	229	å
34	"	83	S	132	„	181	µ	230	æ
35	#	84	T	133	…	182	¶	231	ç
36	$	85	U	134	†	183	·	232	è
37	%	86	V	135	‡	184	¸	233	é
38	&	87	W	136	ˆ	185	¹	234	ê
39	'	88	X	137	‰	186	º	235	ë
40	(	89	Y	138	Š	187	»	236	ì
41	)	90	Z	139	‹	188	¼	237	í
42	*	91	[	140	Œ	189	½	238	î
43	+	92	\	141	□	190	¾	239	ï
44	,	93	]	142	Ž	191	¿	240	ð
45	-	94	^	143	□	192	À	241	ñ
46	.	95	_	144	□	193	Á	242	ò
47	/	96	`	145	‘	194	Â	243	ó
48	0	97	a	146	’	195	Ã	244	ô
49	1	98	b	147	“	196	Ä	245	õ
50	2	99	c	148	”	197	Å	246	ö
51	3	100	d	149	•	198	Æ	247	÷
52	4	101	e	150	–	199	Ç	248	ø
53	5	102	f	151	—	200	È	249	ù
54	6	103	g	152	˜	201	É	250	ú
55	7	104	h	153	™	202	Ê	251	û
56	8	105	i	154	š	203	Ë	252	ü
57	9	106	j	155	›	204	Ì	253	ý
58	:	107	k	156	œ	205	Í	254	þ
59	;	108	l	157	□	206	Î		
60	<	109	m	158	ž	207	Ï		
61	=	110	n	159	Ÿ	208	Ð		
62	>	111	o	160		209	Ñ		
63	?	112	p	161	¡	210	Ò		
64	@	113	q	162	¢	211	Ó		
65	A	114	r	163	£	212	Ô		
66	B	115	s	164	¤	213	Õ		
67	C	116	t	165	¥	214	Ö		
68	D	117	u	166	¦	215	×		
69	E	118	v	167	§	216	Ø		
70	F	119	w	168	¨	217	Ù		
71	G	120	x	169	©	218	Ú		
72	H	121	y	170	ª	219	Û		
73	I	122	z	171	«	220	Ü		
74	J	123	{	172	¬	221	Ý		
75	K	124	|	173	-	222	Þ		
76	L	125	}	174	®	223	ß		
77	M	126	~	175	¯	224	à		
78	N	127	□	176	°	225	á		
79	O	128	€	177	±	226	â		
80	P	129	□	178	²	227	ã		

Exempel 3. ascii() och char()
Visar hur man tar fram ascii-kod och ascii-tecken.

```
select
     AsciiKod = ascii('A')
   , AsciiTecken1 = char(65)
   , AsciiTecken2 = char(97)
   , AsciiTecken3 = char(57)
   , AsciiTecken4 = char(233)
   , AsciiTecken5 = char(33)
```

```
AsciiKod    AsciiTecken1 AsciiTecken2 AsciiTecken3 AsciiTecken4 AsciiTecken4
----------- ------------ ------------ ------------ ------------ ------------
65          A            a            9            é            !
```

Exempel 4. char(13) + char(10)
Visar hur man använder ascii-koden för ny rad `char(13) + char(10)`.

```
select NyRad = 'GOD' + char(13) + char(10) + 'JUL'
```

```
NyRad
--------
GOD
JUL
```

Exempel 5. Söka fram små bokstäver (gemener)
Visar hur man söker fram värden med små bokstäver. Observera att man inte hittar dem genom att skriva ”`where Bokstav between 'a' and 'ö'`”, eftersom stora och små bokstäver inte anses vara olika värden, med normala inställningar på SQL-servern. I detta exempel tas bara de vanliga bokstäverna i svenska alfabetet med.

```
select *
from #LetaLitenBokstav
where
  ascii(Bokstav) between 97 and 122
  or
  ascii(Bokstav) in (228, 229, 246)
```

```
Bokstav
--------
a
z
```

**Exempel 6. Lista med ascii-tecken

Demo som räknar upp alla tecken från ascii-kod 32 till 255. Vissa tecken är inte skrivbara tecken utan är istället olika former av kotrollkoder, t ex 0-31 och 127.

```
set nocount on

drop table #AsciiTecken
go

create table #AsciiTecken (Kod int, Tecken char(1))
go

declare
  @Kod int
  , @Tecken char(1)

set  @Kod = 32
set  @Tecken = ''

while @Kod < 255
begin
  insert #AsciiTecken select @Kod, char(@Kod)
  set @Kod = @Kod + 1
end

select * from #AsciiTecken

set nocount off
```

```
Kod         Tecken
----------- ------
32
33          !
34          "
35          #
36          $
:           :
```

*Exempel 7. upper() och lower()

Förvandlar textsträngen till versaler och gemener.

```
select
   Versaler = upper(Mening)
   , Gemener = lower(Mening)
from #Latin
```

```
Versaler           Gemener
------------------ ------------------
NI TALAR BRA LATIN ni talar bra latin
```

**Exempel 8. Reverse()

Visar hur man skriver ut en textsträng baklänges.

```
select Baklanges = reverse(Mening)
from #Latin
```

```
Baklanges
------------------
nital arb ralat iN
```

**Exempel 9. replicate() och space()

Visar hur man upprepar en textsträng samt blanktecken ett bestämt antal gånger.

```
select
    UpprepaText  = replicate('Hi', 3)
  , UpprepaBlank = 'GOD' + space(5) + 'JUL'
```

```
UpprepaText UpprepaBlank
----------- ------------
HiHiHi      GOD     JUL
```

**Exempel 10. ltrim() och rtrim()

Visar hur man trimmar bort blanktecken till vänster och höger. Effekten syns bäst när man konkatenerar variablerna.

När man hämtar värden från en char-deklarerad variabel så hämtas lika många tecken som den är deklarerad för.

När man hämtar värden från en varchar-deklarerad variabel så hämtas bara de tecken som lagrats i den.

Visa resultatet som text för att lättast se blanktecknen.

```
Select
    T3 = T1 + T1
  , T4 = T2 + T2
from #Trim

select
    T5 = ltrim(T1) + ltrim(T1)
  , T6 = rtrim(T1) + rtrim(T1)
  , T7 = ltrim(rtrim(T1)) + ltrim(rtrim(T1))
from #Trim

select
    T8  = ltrim(T2) + ltrim(T2)
  , T9  = rtrim(T2) + rtrim(T2)
  , T10 = ltrim(rtrim(T2)) + ltrim(rtrim(T2))
from #Trim
```

```
T3         T4
---------- ----------
 123  123   123 123
 123  123   123  123

T5         T6         T7
---------- ---------- ----------
123 123     123 123   123123
123 123     123 123   123123

T8         T9         T10
---------- ---------- ----------
123123      123 123   123123
123 123     123 123   123123
```

**Exempel 11. Rensa blank-tecken

Visar hur man rensar bort alla blank-tecken i en textsträng.

```
select NyText = replace(Mening, space(1), space(0))
from #Latin

select NyText = replace(T1 + T1, space(1), space(0))
from #Trim
```

```
NyText
---------------
Nitalarbralatin

NyText
------
123123
123123
```

**Exempel 12. Rensa nyrad-tecken

Visar hur man byter ut alla nyrad-tecken i en textsträng mot blank-tecken.

```
select *
from #GODJUL

select NyRad = replace(NyRad, char(13) + char(10), space(1))
from #GODJUL
```

```
NyRad
--------
GOD
JUL

NyRad
-------
GOD JUL
```

**Exempel 13. len()

Längden av en textsträng. Blanktecken i slutet räknas ej med.

```
select Langd = len(Mening)
from #Latin

select
    Langd1 = len(T1)
  , Langd2 = len(T2)
from #Trim
```

```
Langd
-----
18

Langd1 Langd2
------ -----------
4      4
4      4
```

*Exempel 14. charindex()

Första positionen av en delsträng i en textsträng utifrån en delsträng.

```
select
    TknPos1 = charindex('talar', Mening)
  , TknPos2 = charindex('pratar', Mening)
  , TknPos3 = charindex('a', Mening)
    -- 2:a Positionen för 'A' hårdkodat
  , TknPos4 = charindex('a', Mening, 6)
    -- 2:a Positionen för 'A' dynamiskt med hjälp av
    -- inkaplsade funktioner
  , TknPos5 = charindex('a', Mening, charindex('a', Mening) + 1)
from #Latin
```

```
TknPos1     TknPos2     TknPos3     TknPos4     TknPos5
----------- ----------- ----------- ----------- -----------
4           0           5           7           7
```

**Exempel 15. patindex()

Första positionen av en delsträng i en textsträng med hjälp av samma wildcard-tecken som med like-operatorn.

```
select
    TknPos1 = patindex('ni%', Mening)
  , TknPos2 = patindex('talar%', Mening)
  , TknPos3 = patindex('%[cgzb]%', Mening)
  , TknPos4 = patindex('%[0-9]%', Mening)
  , TknPos5 = patindex('%[^a-s ]%', Mening)
from #Latin
```

```
TknPos1     TknPos2     TknPos3     TknPos4     TknPos5
----------- ----------- ----------- ----------- -----------
1           0           10          0           4
```

*Exempel 16. Dela upp namn i två delar

Slå isär ett namn till ett förnamn och ett efternamn. Vi antar här, för enkelhets skull, att alla namn har ett och endast ett mellanslag mellan för- och efter-namn. Annars får man förarbeta tabellen med ytterligare stränghantering.

```
select
    Namn
  , BlankPos  = charindex(space(1), Namn)
  , ForNamn   = left(Namn, charindex(space(1), Namn) - 1)
  , EfterNamn = substring(Namn, charindex(space(1), Namn) + 1,99)
from Person
```

```
Namn         BlankPos ForNamn EfterNamn
------------ -------- ------- ---------
Anna Ahl     5        Anna    Ahl
Bo Ek        3        Bo      Ek
Test Person  5        Test    Person
```

*Exempel 17. Dela upp namn i två delar i två steg

Arbeta i två steg genom att spara i en temporär tabell och sedan uppdatera.

```
select
    Namn
  , BlankPos  = charindex(space(1), Namn)
into #Namn
from Person

select
    Namn
  , BlankPos
  , ForNamn   = left(Namn, BlankPos - 1)
  , EfterNamn = substring(Namn, BlankPos + 1, 99)
from #Namn
```

```
Namn          BlankPos    ForNamn    EfterN
------------- ----------- ---------- ------
Anna Ahl      5           Anna       Ahl
Bo Ek         3           Bo         Ek
Test Person   5           Test       Person
```

**Datumfunktioner*

*Exempel 1 visar hur datum presenteras som datumdatatyper.

```
----- Visa aktuell datumtid med 2 olika noggrannheter
select
    Getdate = getdate()
  , Sysdatetime = sysdatetime()
```

Getdate	Sysdatetime
2012-05-17 11:04:03.593	2012-05-17 11:04:03.5931234

```
----- Konvertera aktuell datumtid till 3 olika datatyper
select
     Datetime = cast(sysdatetime() as datetime)
   , Datetime2 = cast(sysdatetime() as datetime2)
   , Smalldatetime = cast(sysdatetime() as smalldatetime)
```

Datetime	Datetime2	Smalldatetime
2012-05-17 11:04:03.593	2012-05-17 11:04:03.5931234	2012-05-17 11:04:00

```
----- Konvertera aktuell datumtid till 4 olika datatyper
select
    Date = cast(sysdatetime() as date)
  , Time0 = cast(sysdatetime() as time(0))
  , Time3 = cast(sysdatetime() as time(3))
  , Time7 = cast(sysdatetime() as time(7))
```

Date	Time0	Time3	Time7
2012-05-17	11:04:03	11:04:03.593	11:04:03.5931234

```
----- Konverterar från text till datetime
select TextToDatetime = cast('2012-05-17 11:04:03.593' as datetime)
```

TextToDatetime
2012-05-17 11:04:03.593

*Exempel 2 visar hur datum presenteras som textdatatyp

```
select
     IdagNu1 = convert(char(23), getdate(), 121)
   , IdagNu2 = convert(char(19), getdate(), 121)
   , IdagNu3 = convert(char(24), getdate(), 113)
```

IdagNu1	IdagNu2	IdagNu2
2012-05-17 11:04:03.593	2012-05-17 11:04:03	17 May 2012 11:04:03.593

```
select
     Idag1 = convert(char(10), getdate(), 120)
   , Idag2 = convert(char(8), getdate(), 112)
   , Idag3 = convert(char(11), getdate(), 106)
```

Idag1	Idag2	Idag3
2012-05-17	20120517	17 May 2012

```
select
     Nu1 = convert(char(12), getdate(), 14)
   , Nu2 = convert(char(8), getdate(), 14)
```

Nu1	Nu2
11:04:03.593	11:04:03

**Exempel 3 Pausa med waifor delay eller välj klockslag att starta med waitfor time

```
-- Starta efter angiven tid
select cast(getdate() as time(0))
go
waitfor delay '00:00:02'  -- Pause i 2 sekunder
select cast(getdate() as time(0))
```

16:21:30
16:21:32

```
-- Starta vid ett anvgivet klockslag
select cast(getdate() as time(0))
go
waitfor time '16:25:05'
select cast(getdate() as time(0))
```

16:24:30
16:25:05

**Exempel 4 Veckohantering med SCBDatToVNr(Datum)och SCBVNrToDat(År, VeckoNr)

```
-- Beräkna fram veckonummer utifrån datum
select
    master.dbo.SCBDatToVNr('2012-01-01')
  , master.dbo.SCBDatToVNr('2012-01-02')
  , master.dbo.SCBDatToVNr('2012-02-29')
  , master.dbo.SCBDatToVNr('2012-12-31')
```

201152	201201	201209	201301

```
-- Beräkna fram datum för veckans första dag
-- utifrån år och veckonummer
select
    master.dbo.SCBVNrToDat('2011',52)
  , master.dbo.SCBVNrToDat('2012',1)
  , master.dbo.SCBVNrToDat('2013',1)
  , master.dbo.SCBVNrToDat('2009',53)
  , master.dbo.SCBVNrToDat('2010',53)
```

2011-12-26	2012-01-02	2012-12-31	2009-12-28	NULL

**Exempel 5 Tidsdiff sparad i time-datatypen

```
-- Tidskillnad mellan två tidpunkter
-- Om tidsskillnaden är över ett dygn börjar tiden om från början
select
    cast(  cast('2000-01-01 18:05:10.006' as datetime)
         - cast('2000-01-01 12:00:00.003' as datetime) as time(0)
)
  , cast(  cast('2000-01-02 18:05:10.006' as datetime)
         - cast('2000-01-01 12:00:00.003' as datetime) as time(0)
)
```

**Exempel 6 Tidsdiff med datediff(Datepart, Tidpunkt1, Tidpunkt2)

```
-- Tidskillnad mellan två tidpunkter mätt i en tidsdel (datepart)
-- Avrundningen sker utan att titta på mindre tidsdelar
-- Möjliga datepart-parametrar: ms, ss, mi, hh, dd, dw, dy, ww, mm, qq, yy
-- I detta fall blir resultatet 1 år fast det nästan är två år
select
  DiffSek = datediff(ss, '2000-01-01 12:00:00.000', '2001-12-31 23:59:59.999')
, DiffAr = datediff(yy, '2000-01-01 12:00:00.000', '2001-12-31 23:59:59.999')
-- I detta fall blir resultatet 1 år fast det egentligen bara är 1 ms
select
    DiffSek = datediff(ss, '2000-12-31 23:59:59.999', '2001-01-01 00:00:00.000')
  , DiffAr = datediff(yy, '2000-12-31 23:59:59.999', '2001-01-01 00:00:00.000')
```

**Exempel 7 Tidskillnad mellan två tidpunkter angivet i antal dagar och tidsformat. Lite trassligt eftersom datediff() behöver justeras beroende på skillnaden i klockslag

```
declare @DatumTid1 datetime
declare @DatumTid2 datetime
declare @Tid1 time(3)
declare @Tid2 time(3)
declare @DatumDiffDagar int
declare @TidsDiff time(3)
select @DatumTid1 = '2000-12-31 12:00:00.004'
select @DatumTid2 = '2001-01-01 12:00:00.007'
select @Tid1 = cast(@DatumTid1 as time(3))
select @Tid2 = cast(@DatumTid2 as time(3))
select @DatumDiffDagar =
  case
    when datediff(ms, @Tid1, @Tid2) >= 0
      then datediff(dd, @DatumTid1, @DatumTid2)
    else datediff(dd, @DatumTid1, @DatumTid2) - 1
  end
select @TidsDiff = @DatumTid2 - @DatumTid1
select DatumDiffDagar = @DatumDiffDagar, TidsDiff = @TidsDiff
```

*Beskrivning av datumhanteringen

- SCB:s SQL-servrar är inställda att presentera datum på följande sätt:
  `2012-05-17 11:04:03.593.`
- Observera att smalldatetime bara räknar med en noggrannhet på minuten.
- getdate() räknar normalt med en noggrannhet på millisekunder
- Nya datatyper från och med SQL2008: datetime2, date, time(n) där n är ett heltal 0-7.
- datetime2 och time(7) klarar av en noggrannhet på 10-7 sekunder

**Systemfunktioner*

*Exempel 1.
Konvertering till olika tal-datatyper med funktionen cast()

```
----- Till pyttelitet heltal (1 byte)
----- Minsta är 0
----- Största är 255
select Tal = cast(255 as tinyint)

----- Till litet heltal (2 byte)
----- Minsta är -32768
----- Största är 32767
select Tal = cast(32767 as smallint)

----- Till normalt heltal (4 byte)
----- Minsta är -2147483648
----- Största är 2147483647
select Tal = cast(2147483647 as int)

----- Till stort heltal (8 byte)
----- Minsta är -9223372036854775808
----- Största är 9223372036854775807
select Tal = cast(9223372036854775807 as bigint)

----- Till decimaltal med 38 siffror och 0 decimaler (17 byte)
----- Minsta är -99999999999999999999999999999999999999
----- Största är 99999999999999999999999999999999999999
select Tal = cast(99999999999999999999999999999999999999 as numeric(38,0))

----- Till decimaltal med 38 siffror och 38 decimaler (17 byte)
----- Minsta är -0.99999999999999999999999999999999999999
----- Största är 0.99999999999999999999999999999999999999
select Tal = cast(0.99999999999999999999999999999999999999 as numeric(38,38))
```

*Exempel 2.
Konvertering med trunkering (avhuggning av decimalerna) med cast()

```
----- Från decimaltal med 1 decimal till heltal
select Tal = cast(2.9 as int)
```

*Exempel 3.
Konverteringar med automatisk avrundning med cast()

```
----- Från decimaltal med 1 decimal
----- till decimaltal med 0 decimaler
select Tal = cast(2.1 as numeric(1,0))
select Tal = cast(2.5 as numeric(1,0))
select Tal = cast(2.9 as numeric(1,0))
```

*Exempel 4.
Konverteringar till ökad noggrannhet med cast()

```
----- Från heltal till decimaltal med
----- 1 heltalssiffra och 37 decimaler
select Tal = cast(1 as numeric(38,37))
```

*Exempel 5.
Konvertering från text till tal med cast()

```
----- Från heltal i textformat
select Tal = cast('2' as int)

----- Från decimaltal i textformat
select Tal = cast('2.1' as numeric(2,1))
select Tal = cast('2.1' as numeric(1,0))
select Tal = cast('2.9' as numeric(1,0))
```

*Exempel 6.
Felaktig konvertering till tal med cast()

```
----- Arithmetic overflow error converting int
----- to data type numeric
select Tal = cast(10 as numeric(1, 0))

----- Arithmetic overflow error converting numeric
----- to data type numeric
select Tal = cast(9.99 as numeric(1, 0))

----- Error converting data type varchar to numeric
select Tal = cast('9,99' as numeric(3, 2))

----- Conversion failed when converting the varchar value '8.99'
----- to data type int
select Tal = cast('8.99' as int)

----- Fungerar om man först konverterar texten till decimaltal
----- och sedan till heltal
select Tal = cast(cast('8.99' as numeric(1, 0)) as int)
```

*Exempel 7.
Konvertering till text med cast()

```
Select
  KonvToText1= cast(12 as char(2))
  , KonvToText2 = cast(12.123 as char(6))
```

```
KonvToText1 KonvToText2
----------- -----------
12          12.123
```

*Exempel 8.
Konvertering till datum med cast()

```
select KonvToDateTime = cast(getdate() as datetime)

select KonvToSmalldatetime =
        cast(getdate() as smalldatetime)
```

```
KonvToDateTime
-----------------------------------------------------
2007-03-12 16:24:33.797

KonvToSmalldatetime
-----------------------------------------------------
2007-03-12 16:25:00
```

**Exempel 9.

Kontroll av datatyp. Isnumeric() är tyvärr inte vattentät för decimalkomma.

```
select
  OmNull1 = isnull(NULL, '-')
  , OmNull1 = isnull('9', '-')
```

```
OmNull1 OmNull1
------- -----------
-       9
```

```
select
    OmTal1 = isnumeric('1.5')
  , OmTal2 = isnumeric('-1.5')
```

```
OmTal1 OmTal2
------ -----------
1      1
```

```
select OmTal = isnumeric(1.5E5)
```

```
OmTal
-----
1
```

```
select OmTal = isnumeric('1,5') -- Bugg
```

```
OmTal
-----
1
```

```
select OmTal = isnumeric('1A')
```

```
OmTal
-----
0
```

```
select
    OmDatum1 = isdate('2007-03-12 19:37:53.193')
  , OmDatum2 = isdate('2007-03-12')
  , OmDatum3 = isdate('2007')
  , OmDatum4 = isdate('12 mar 2007')
  , OmDatum5 = isdate('19:37')
  , OmDatum6 = isdate('mar')
```

```
OmDatum1    OmDatum2    OmDatum3    OmDatum4    OmDatum5    OmDatum6
----------- ----------- ----------- ----------- ----------- -----------
1           1           1           1           1           0
```

```
**Exempel 10.
Ta fram olika typer av användarnamn. Användbart vid
loggning och test av behörigheter.
----- Rekommenderade sätt att fånga användarnamn vid loggning
select
    -- Domainnamn\LoginNamn
    DomainLoginNamn = Original_Login()
    -- LoginNamn
    -- Blir samma som windows-loginnamnet
  , LoginNamn = stuff(Original_Login(), 1, charindex('\', Original_Login()),
    -- DatabasAnvändarNamn. Är man db_owner blir det alltid DBO.
  , DbAnvNamn = user
    -- ArbetsstationsNamn
  , ArbStnNamn = host_name()
```

AnvNamn1	AnvNamn2	ArbStnNamn
SCB\UTB_ELEVO01	UTB_ELEVO01	SCB1234

```
----- Simulerar en körning som en annan användare
----- Går ej att simulera användare med högre behörighet
----- Påverkar resultatet av nedanstående kommandon
----- Återvänder till föregående användare med revert
exec as user = 'UTB_ElevO01'
select
    DomainSysAnvNamn = Original_Login()
  , DbAnvNamn = user
Revert
```

DomainSysAnvNamn	DbAnvNamn
SCB\UTB_ELEVO01	UTB_ELEVO01

```
exec as user = 'guest'
select
    DomainSysAnvNamn = Original_Login()
  , DbAnvNamn = user
revert
```

DomainSysAnvNamn	DbAnvNamn
public	guest

```
select
    DomainSysAnvNamn = Original_Login()
  , DbAnvNamn = user
```

DomainSysAnvNamn	DbAnvNamn
SCB\scbtole	scbtole

```
**Exempel 11.
Ta fram objektid och kolumnnamn för tabell i aktuell
databas.
object_id() visar internt ID för ett object
col_name(X) visar namnet på kolumn nr X.

select
    IDTabell = object_id(dbo.Kommun', 'U')
  , KolumnNamn = col_name(object_id('dbo.Kommun'), 3)
```

IDTabell	KolumnNamn
430624577	Inv

```
**Exempel 12.
Ta fram antal kolumner för en tabell med hjälp av
object_id(). Är användbart då man måste söka på ID.
select AntalKolumner = count(*)
from sys.columns
where object_id = object_id('UTB_Kurs1.dbo.Kommun',
  'U')
```

AntalKolumner
11

```
**Exempel 13. Diverse systemvariabler.
@@version innehåller version på databashanteraren.
@@language innehåller språk för t ex felhanteringen.
@@servername innehåller aktuell servers namn.
@@ datefirst innehåller nummer för första veckodagen. 1
betyder måndag.
@@error visar felkoden i senaste SQL-instruktionen.
@@rowcount visar antal senast utförda rader.
select
  VersionDBHanterare = @@version
  , Sprak = @@language
  , ServerNamn = @@servername
```

VersionDBHanterare	Sprak	ServerNamn
Microsoft SQL Server 2005 -...	Svenska	Q093\G

```
select @@datefirst
```

Veckodag1
1

```
select * into #B from Kommun
select
  FelKod = @@error
  , AntalRader = @@rowcount
```

FelKod	AntalRader
0	286

**Rankningsfunktioner*

*Exempel 1. Row_number()

Skapar en tabell med tal från 1-24. Det blir lika många rader som rader i tabellen Lan. Vill man ha fler rader väljer man en större tabell.

```
select RadNr = row_number() over(order by Namn)
from Lan
```

Radnr
1
2
3
:

*Exempel 2. Row_number()

Skapar en tabell med tal från 1-10. Antal rader begränsas av `top`.

```
select top 10 RadNr = row_number() over(order by Namn)
from Lan
```

Radnr
1
2
3
:

*Exempel 3. Row_number()

Skapar en tabell med tal från 0-10. Man kan ändra startvärdet genom att addera eller subtrahera efter funktionen.

```
select top 10 RadNr = row_number() over(order by Namn) - 1
from Lan
```

Radnr
0
1
2
:

*Exempel 4. Row_number()

Skapar en tabell med tal från 0-100 000. Här har vi valt att cross-joina systemtabellen sys.all_columns med sig själv. Eftersom denna system-tabell finns i alla databaser kommer sql-koden alltid att fungera. Resultatet av cross-joinen blir över en miljon rader, om man inte begränsar med top.

```
drop table #RadNr
select   top 100000 RadNr = row_number()
         over(order by A.name)
into #RadNr
from
  sys.all_columns A
  cross join
  sys.all_columns B
select * from #RadNr
```

Radnr
1
2
3
:

*Exempel 5. Row_number()

Sorterar raderna efter KommunNamn och skapar den nya kolumnen RadNr.

```
select
  RadNr = row_number()
          over(order by Namn)
  , KommunNamn = Namn
from Kommun
```

Radnr	KommunNamn
1	Ale
2	Alingsås
3	Alvesta
:	:

*Exempel 6. Row_number() med partition by

Sorterar raderna efter Lan och KommunNamn och skapar den nya kolumnen RadNr som börjar om från början för varje nytt län.

```
select
  Lan
  , RadNr = row_number()
            over(partition by Lan
                 order by Lan, Namn)
  , KommunNamn = Namn
from Kommun
```

Lan	Radnr	KommunNamn
01	1	Botkyrka
01	2	Danderyd
01	3	Ekerö
:	:	:
03	1	Enköping
03	2	Håbo
:	:	:

*Exempel 7. Row_number()

Sorterar raderna efter poäng baklänges och Namn samt skapar den nya kolumnen placering.

```
select
  Placering = row_number()
              over (order by Poang desc, Namn)
  , Poang
  , Namn
  , Klubb
from Tavling
```

Placering	Poang	Namn	Klubb
1	10	Adam	AIK
2	10	Ivar	DIF
3	7	Carl	ÖSK
4	7	Filip	AIK
:	:	:	:

**Exempel 8. Rank()

Sorterar raderna efter Poang baklänges samt skapar den nya kolumnen placering med samma Placering för alla med samma Poang.

```
select
  Placering = rank()
              over (order by Poang desc)
  , Poang
  , Namn
  , Klubb
from Tavling
```

Placering	Poang	Namn	Klubb
1	10	Adam	AIK
1	10	Ivar	DIF
3	7	Filip	AIK
3	7	Gustav	ÖSK
:	:	:	:

**Exempel 9. Dense_Rank()

Gör samma sak som rank() med den skillnaden att placering inte får överhoppade nummer.

```
select
  Placering = dense_rank()
              over (order by Poang desc)
  , Poang
  , Namn
  , Klubb
from Tavling
```

Placering	Poang	Namn	Klubb
1	10	Adam	AIK
1	10	Ivar	DIF
2	7	Filip	AIK
2	7	Gustav	ÖSK
:	:	:	:

```
**Exempel 10. Dense_Rank() kombinerat med Rank()
select
  KlubbNr = dense_rank()
            over (order by Klubb)
  , Klubb
  , Placering = rank()
                over(partition by Klubb
                     order by Poang desc, Namn)
  , Poang
  , Namn
from Tavling
```

KlubbNr	Klubb	Placering	Poang	Namn
1	AIK	1	10	Adam
1	AIK	2	7	Filip
1	AIK	3	5	David
:	:	:	:	:

***Systemtabeller*

```
**Exempel 1.
```

Hämtar metadata om tabellerna i en databas, från tabellerna **sys.tables**, **sys.columns** och **sys.schemas**.

```
select
    TabellNamn = A.Name
  , SchemaNamn = C.Name
  , KolumnNr = B.Column_id
  , KolumnNamn = B.Name
from
  UTB_Kurs1.sys.tables A
  join
  UTB_Kurs1.sys.columns B
    on A.object_id= B.object_id
  join
  UTB_Kurs1.sys.schemas C
    on A.schema_id = C.schema_id
order by
  TabellNamn
  , SchemaNamn
  , KolumnNr
```

TabellNamn	SchemaNamn	KolumnNr	KolumnNamn
:	:	:	:
Kommun	dbo	1	Kommun
Kommun	dbo	2	Namn
Kommun	dbo	3	Inv
:	:	:	:

4.5. *Case..end

***Exempel**

*Exempel 1. Här skapas den nya kolumnen KonText utifrån kolumnen Kon. Om **when**-villkoret är sant, utförs instruktionen efter **then** och därefter hoppar programmet vidare till **end**. Om **when** -villkoret inte är sant, stegar programmet vidare till nästa **when**-villkor. Om inget av **when**-villkoren är sanna, utförs instruktionen efter **else**.

```
select
  Person
  , KonText =
    case
      when Kon = 'K' then 'Kvinna'
      when Kon = 'M' then 'Man'
      else '?'
    end
from Man_Kvinna
```

```
Person     KonText
---------- -------
Anders     Man
Bo         Man
Carl       Man
Dag        Man
Doris      Kvinna
Elin       Kvinna
Erik       Man
Fia        Kvinna
Gun        Kvinna
Helen      Kvinna
```

*Exempel 2. Här får man samma resultat genom att placera **Kon** direkt efter **case**.

```
select
  Person
  , KonText =
    case Kon
      when 'K' then 'Kvinna'
      when 'M' then 'Man'
      else '?'
    end
from Man_Kvinna
```

*Exempel 3. Här beräknas kön från ett personnummer m h a stränghantering.

```
select
  PersonNr
  , KonText =
    case
      when substring(PersonNr, 11, 1)
          in ('1', '3', '5', '7', '9') then 'Man'
      when substring(PersonNr, 11, 1)
          in ('0', '2', '4', '6', '8') then 'Kvinna'
      else '?'
    end
from Person
```

PersonNr	KonText
198012319929	Kvinna
198012318818	Man
1980123188X8	?

*Exempel 4 är ett exempel på hur man skapar kolumn med gruppindelning med hjälp av **case** ... **end**. Här skapas den nya kolumnen SkatteGrp, som delar in skattesatserna i grupper av skattesatser.

```
select
  Namn
  , TSkatt
  , SkatteGrp =
    case
      when Tskatt < 29  then '00-029'
      when Tskatt < 32  then '29-032'
      when Tskatt >= 32 then '32-100'
      else '?'
    end
  , SkatteGrpText =
    case
      when Tskatt < 29  then '<29'
      when Tskatt < 32  then '29-32'
      when Tskatt >= 32 then '32-'
      else '?'
    end
from Kommun
```

Namn	TSkatt	SkatteGrp	SkatteGrpText
Ale	32.22	32-100	32-
Alingsås	32.76	32-100	32-
Alvesta	31.52	29-032	29-32
:	:	:	:

***Beskrivning av case**

Syntax

```
case
   when villkor then uttryck
   [when villkor then uttryck]
   [else uttryck]
end
```

eller alternativt om samma uttryck används på alla ställen

```
case uttryck
   when värde then uttryck
   [when värde then uttryck]
   [else uttryck]
end
```

Ett mycket kraftfullt kommando som kan användas i olika bisatser till select-satsen.
Används ofta till att göra gruppindelningar av data, t ex åldersgrupper.

4.6. *Subqueries

***Exempel**

`*Exempel 1` tar fram de 10 sista kommunerna i tabellen kommun sorterat efter kommunnamn i stigande ordning. Här används en #-tabell för att mellanlagra de 10 sista kommunerna och sedan används #-tabellen för att sortera om raderna.

```
select top 10 *
into #Kommun
from Kommun
order by Namn desc

select *
from #Kommun
order by Namn
```

`*Exempel 2` gör samma sak som `Exempel 1` fast på ett annat sätt. Här ligger första frågan i where-bisatsen, som hänvisar till frågan som om den vore en lista med värden. Här använder vi **jämförelseoperatorn in**. Detta är ett exempel på en **subquery i where-bisatsen**.

```
select *
from Kommun
where Namn in
  (select top 10 Namn
   from Kommun
   order by Namn desc)
order by Namn
```

`*Exempel 3` gör samma sak som `Exempel 1` fast på ett annat sätt. Här ligger första frågan i from-bisatsen, som hänvisar till frågan som om den vore en tabell. Vi måste då ge den temporära tabellen ett namn, t ex Temp. Denna tabell slutar att existera så fort hela selectsatsen är körd. Detta sätt, som ger bäst prestanda, är ett exempel på en **subquery i from-bisatsen**.

```
select *
from
  (select top 10 *
   from Kommun
   order by Namn desc) Temp
order by Namn
```

`*Exempel 4` gör aggregatberäkningar från 2 olika tabeller. Detta är ett exempel på två **subqueries i select-bisatsen**. Resultatet ger två oberoende totaler på en rad. Här använder vi **jämförelseoperatorn =** och då får inte underfrågorna ge mer än en rad som svar, vilket är självklart i detta fall.

```
select
  AntalRaderIKommunTabellen =
    (select count(*) from Kommun)
  , AntalRaderILanTabellen =
    (select count(*) from Lan)
```

*Exempel 5 tar fram alla uppgifter från tabellen län, för de län som bara innehåller en kommun. Eftersom subqueryn grupperar på Lan får endast Lan finnas med i dess select-bisats. De övriga kolumnerna får man fram från den överliggande selectsatsen. Den alternativa metoden beskriver hur man gör i flera steg m h a join.

```
select *
from Lan
where Lan in
  (select Lan
   from Kommun
   group by Lan
   having count(*) = 1)

-- Alternativ metod
select Lan
into #A
from Kommun
group by Lan
having count(*) = 1

select B.*
from
  #A A
  join Lan B
    on A.Lan = B.Lan
```

*Exempel 6 tar fram de rader i tabellen Kommun där TSkatt är lika med lägsta värdet för TSkatt.

```
select *
from Kommun
where
  TSkatt =
  (select min(TSkatt)
   from Kommun)
```

*Exempel 7 beräknar antal unika rader i tabellen Lan2. Här använder vi subqueries, bestående av en aggregering, i from-bisatsen. Observera att man inte kan skriva **`select count(distinct *)`**.

```
select count(*)
from (select distinct * from Lan2) Temp
```

4.7. **CTE = Common Table Expressions

****Exempel**

`**Exempel 1 – En CTE innehållande en kortlivad temporärtabell`

Tar fram *Antal innevånare per län och åldersgrupp under 2005*. I första steget skapas en kortlivad temporärtabell med namnet Inv. I sista steget anropas tabellen på samma sätt som om det hade varit en vanlig tabell. När CTE:n är exekverad dör den kortlivade temporärtabellen och är inte åtkomlig längre. Hela CTE:n måste köras i ett svep för att fungera.

```
with
Inv as
(
select
  Lan
  , AlderGrp2 =
    case
      when AlderGrp in ('0-4', '5-14')
        then 'Barn'
      else 'Vuxen'
    end
   , Inv
from dbo.KommunInv2001_2005
where Ar = '2005'
)
select
  Lan
  , AlderGrp2
  , Inv = Sum(Inv)
from Inv
group by
  Lan
  , AlderGrp2
order by
  Lan
  , AlderGrp2
```

Lan	AlderGrp2	Inv
01	Barn	344656
01	Vuxen	1545289
03	Barn	54282
:	:	:

**Exempel 2 – En CTE innehållande 2 kortlivade temporärtabeller

Denna CTE ger samma resultat sak som Exempel 1, fast i ett extra steg. Skillnaden är att om man har flera kortlivade temporärtabeller, så åtskiljs dessa med kommatecken och namnges med egna namn (Inv1 och Inv2). Lägg märke till att när Inv2 skapas så anropar den Inv1. Detta är alltså möjligt men inte nödvändigt.

```
with
Inv1 as
(
select
  Lan
  , AlderGrp2 =
    case
      when AlderGrp in ('0-4', '5-14')
        then 'Barn'
      else 'Vuxen'
    end
   , Inv
from dbo.KommunInv2001_2005
where Ar = '2005'
)
,
Inv2 as
(
select
  Lan
  , AlderGrp2
  , Inv = Sum(Inv)
from Inv1
group by
  Lan
  , AlderGrp2
)
select *
from Inv2
order by
  Lan
  , AlderGrp2
```

Lan	AlderGrp2	Inv
01	Barn	344656
01	Vuxen	1545289
03	Barn	54282
:	:	:

**Exempel 3 – En rekursiv CTE

Först skapas en test-tabell innehållande uppgifter om anställda och Id för vilken annan anställd person som är chef. Vi tittar även på testdata.

Därefter skapar vi en rekursiv fråga som presenterar hela hierarkin mellan anställda och deras chefer, nivå för nivå. Rekursiv innebär att CTE:n anropar sig själv, vilket sker i andra halvan av CTE:n där Anst joinas mot tabellen #Anst. Slutligen anropas CTE:n av den efterföljande select-satsen.

```
-----------------
-- Skapa testdata
-----------------
select
  IdAnst = 1
  , NamnAnst = 'Anna'
  , IdChef = NULL
into #Anst
union
select 2, 'Bo', 1
union
select 3, 'Bea', 1
union
select 4, 'Carin', 2
union
select 5, 'Carl', 2
union
select 6, 'Dan', 3
union
select 7, 'Elin', 3

-- Titta på testdata
select * from #Anst
```

IdAnst	NamnAnst	IdChef
1	Anna	NULL
2	Bo	1
3	Bea	1
4	Carin	2
5	Carl	2
6	Dan	3
7	Elin	3

```
---------------------
-- Fråga mot testdata
---------------------
----- Skapa en rekursiv CTE
;with
Anst as
(
-- Definition av ankare
select
  Niva = 0
  , IdChef
  , NamnChef = cast(NULL as varchar(5))
  -- Samma datatyp för samma kolumn i en union
  , IdAnst
  , NamnAnst
from #Anst
where IdChef is NULL  -- Högsta chefen
union all
-- Definition av rekursion
select
  B.Niva + 1
  , A.IdChef
  , B.NamnAnst
  , A.IdAnst
  , A.NamnAnst
from
  #Anst A
  join
  Anst B  -- Här joinas CTE:n Anst mot tabellen #Anst
    on A.IdChef = B.IdAnst
)

-- Anrop av CTE
select *
from Anst
where IdChef is not NULL  -- IdChef is NULL är
order by                  -- ointressant. Testa!
  Niva
  , IdChef
  , IdAnst
```

Niva	IdChef	NamnChef	IdAnst	NamnAnst
1	1	Anna	2	Bo
1	1	Anna	3	Bea
2	2	Bo	4	Carin
2	2	Bo	5	Carl
2	3	Bea	6	Dan
2	3	Bea	7	Elin

****Beskrivning av CTE (=Common Table Expressions)**

Syntax

```
with
selectsatsens aliasnamn [(kolumnlista)] as (SQL-sats)
[, ...]

efterföljande sql-sats
```

t ex

```
with
A as
(select K1, K2 from Tab1)
, B as
(select K1, K2 from Tab2)
, C as
(select K1, K2 from A join B on A.K1=B.K1)

select * from C
```

- En CTE är ett paket med en eller flera selectsatser som skapar kortlivade temporärtabeller, åtskilda av kommatecken. En CTE måste alltid avslutas med en vanlig sql-sats.
- Dessa selectsatser kan anropa varandra inom CTE:n och från den närmast efterföljande sql-satssen. Därefter existerar ej dessa temporära CTE:er.
- Man refererar till en tabell inom CTE:n som om det vore en vanlig tabell.
- Man kan ge nya kolumnnamn i CTE:ns select-sats eller i kolumnlistan direkt efter resultatsetets namn.
- SQL-satsen som föregår CTE:n måste avslutas med ett semikolon. Eventuellt så kan man skriva semikolon före CTE:n.
- En CTE får inte innehålla select into, order by eller compute utom i sista steget.
- Fördelen med en CTE jämfört med en vanlig #-tabell är att koden blir mer lättläst samt bättre prestanda eftersom den lagras direkt i arbetsminnet i datorn.
- I en CTE kan ingå dessa typer av SQL-satser: select, insert, update, delete och select-frågan i create view.
- CTE:er är mycket användbara när man skapar vyer eftersom man kan anropa tabeller i flera steg inom with-satsen.

5. *DDL (Data Definition Language)

5.1. *Datatyper

Ett DDL-kommandon påverkar ett objekts egenskaper t ex en tabells namn eller datatyp. Varje kolumn i en tabell måste ha en datatyp. Datatypen har satts med tanke på vad som ska lagras i kolumnen, t.ex. datum, text eller siffror.

De viktigaste datatyperna

Datatyp	Beskrivning	Värdemängd
Char(n)	Text med fast längd	max 8000 tecken n=antal tecken
Varchar(n)	Text med variabel längd	max 8000 tecken n=antal tecken
Tinyint	Pyttesmå heltal	0 till 255
Smallint	Små heltal	- 32 768 till 32 767
Int	Heltal	-2 147 483 648 till 2 147 483 647
Bigint	Stora heltal	-9 223 372 036 854 775 808 till 9 223 372 036 854 775 807
Numeric(p,s)	Decimaltal	p=totalt antal siffror s=antal decimaler (max 38 decimaler) -10^38 till 10^38
Smalldatetime	Datum och tid	År 1900-2079 noggrannhet: minut
Datetime	Datum och tid med mer detaljerade tidsuppgifter	År 1753-9999 noggrannhet: millisekund

5.2. *Create table

***Exempel**

*Exempel 1.
Skapa en **permanent** tabell.

```
--  Om tabellen finns så tar man bort den
if object_id('UTB_ElevO01.dbo.Person', 'U') is not
null
  drop table UTB_ElevO01.dbo.Person
go

-- Skapa en ny tabell
create table UTB_ElevO01.dbo.Person
(
PersonNr  char(12) not null
, Namn    char(25) null
, Adress  char(50) null
)
```

*Exempel 2. Skapa en **temporär #-tabell.**

```
--  Om tabellen finns så tar man bort den
if object_id('tempdb..#Person', 'U') is not null
  drop table #Person
go

-- Skapar en ny #-tabell
create table #Person
(
PersonNr  char(12) not null
, Namn    char(25) null
, Adress  char(50) null
)
```

*Exempel 3. Skapa en **temporär ##-tabel**l.

```
--  Om tabellen finns så tar man bort den
if object_id('tempdb..##Person', 'U') is not null
  drop table ##Person
go

-- Skapar en ny ##-tabell
create table ##Person
(
PersonNr  char(12) not null
, Namn    char(25) null
, Adress  char(50) null
)
```

***Beskrivning av create table**

Både permanenta och temporära tabeller droppas och skapas på samma sätt.

Syntax

Skapa tabell

```
create table Tabellnamn
(
kolumnnamn datatyp [[not null]|[null]]
[,...]
)
```

Om `null` eller `not null` inte anges efter datatypen kommer kolumnen, som default, inte att tillåta NULL-förekomster. Genom att lägga till `null` efter datatypen *tillåts* NULL-förekomster i kolumnen. Ange alltid **`null`** eller **`not null`** för alla kolumner.

Kommandot **`create table`** och alla övriga DDL-kommandon ska följas av **`go`** om man tänker köra ytterligare instruktioner samtidigt efteråt.

Som exemplen ovan visar, måste man droppa en redan existerande tabell, om man vill skapa om den.

Om man vill skapa en permanent tabell bör man alltid kvalificera tabellnamnet med databasnamn och ägaren DBO, t ex *UTB_ElevO01.dbo.Person* .

##-tabeller är temporära tabeller som är lite mindre temporära än en #-tabell eftersom den kan delas av flera olika processer och den droppas inte automatiskt förrän alla inblandade processer är avslutade.

5.3. *Drop table och drop view

***Exempel**

Exempel 1
Tar bort tabellen *UTB_ElevO01.dbo.Person*.

```
drop table UTB_ElevO01.dbo.Person
```

Exempel 2
Tar bort tabellen #Person.

```
drop drop #Person
```

*Exempel 3
Om tabellen UTB_ElevO01.dbo.Person finns, så ta bort den. 'U' betyder *User Table* d v s användartabell.

```
if object_id('UTB_ElevO01.dbo.Person', 'U') is not
null
  drop table UTB_ElevO01.dbo.Person
```

*Exempel 4
Om tabellen #Person finns, så ta bort den.

```
if object_id('tempdb..#Person', 'U') is not null
  drop table #Person
```

*Exempel 5
Om tabellen ##Person finns, så ta bort den.

```
if object_id('tempdb..##Person', 'U') is not null
  drop table ##Person
```

*Exempel 6
Om vyn dbo.vy_Person finns, så ta bort den. Man måste befinna sig i den databasen som de skapas i.

```
if object_id('dbo.vy_Person', 'V') is not null
  drop view dbo.vy_Person
```

***Beskrivning av drop table och drop view**

Syntax

Droppa tabell

drop table tabellnamn

Tabeller tas bort med hjälp av kommandot **`drop table`**.

Temporära #-tabeller tas bort automatiskt när fönstret i SQL Server Management Studio stängs. Man behöver alltså inte ta bort #-tabeller manuellt. Det finns tillfällen då det behövs, t ex när man vill köra en sql-sats flera gånger i samma fönster där en #-tabell skapas.

Om man vill ta bort en permanent tabell bör man alltid kvalificera tabellnamnet med databasnamn och ägaren DBO, t ex UTB_ElevO01.dbo.Person .

Om tabellen inte finns, så blir det felmeddelande när man försöker droppa. Därför använder man funktionen **`object_id()`** när man vill undvika felmeddelanden.

Droppa vy

drop view vynamn

Tabeller tas bort med hjälp av kommandot **`drop view`**.

Man måste befinna sig i den databasen som de skapas i.

5.4. *Vyer

*Exempel

*Exempel 1 skapar en vy för kommuner i Örebro län. Först raderas vyn ifall den redan finns, sedan skapas vyn och sist visas innehållet från vyn på skärmen. Eftersom både create och drop är DDL-kommandon, ska man lägga till ett go efteråt. Observera att man bara kan ta med order by i en vy när top finns i select-satsen. Tyvärr fungerar sorteringen inte i en vy om man skriver top 100 percent, utan man måste ange ett tillräckligt stort tal för att alla poster ska tas med sorterat. Tänk på att stå i den databas som du vill skapa vyn i!

```
use UTB_ElevO01
go

if object_id('dbo.vy_Kommuner_i_Orebro_Lan', 'V')
  is not null
  drop view dbo.vy_Kommuner_i_Orebro_Lan
go

create view dbo.vy_Kommuner_i_Orebro_Lan
as
select top 100000
  Namn
  , Kommun
  , Inv
from UTB_Kurs1.dbo.Kommun
where Lan = '18'
order by Namn
go

select * from dbo.vy_Kommuner_i_Orebro_Lan
```

Namn	Kommun	Inv
Askersund	1882	12320
Degerfors	1862	11698
Hallsberg	1861	16610
:	:	:

*Exempel 2 skapar en vy för full join mellan tabellerna Forfattare, Titlar och Forlag.

```
use UTB_ElevO01
go

if object_id('dbo.vy_Forfattare_Titel_Forlag', 'V')
  is not null
  drop view dbo.vy_Forfattare_Titel_Forlag
go

create view dbo.vy_Forfattare_Titel_Forlag
as
select top 100000
  ForfattarNamn = A.Namn
  , BokTitel = B.Titel
  , ForlagNamn = C.Forlag
from UTB_Kurs1.dbo.Forfattare A
  full join UTB_Kurs1.dbo.Titlar B
    on A.ForfId=B.ForfId
  full join UTB_Kurs1.dbo.Forlag C
    on B.ForlId=C.ForlId
order by
  ForfattarNamn
  , BokTitel
  , ForlagNamn
go

select * from dbo.vy_Forfattare_Titel_Forlag
```

ForfattarNamn	BokTitel	ForlagNamn
NULL	NULL	Liber
Bo Ek	NULL	NULL
Jan Guillou	Fiendens fiende	Prisma
Jan Guillou	Gustav	Prisma
Jan Guillou	Vendetta	Rabén & Sjögren
Liza Karlsson	NULL	NULL
Liza Marklund	Gömda	Bonniers
Liza Marklund	Sprängaren	Bonniers
Liza Nilsson	Skidresan	Prisma

**Exempel 3 skapar en vy med hjälp av CTE. Se kapitel 4.7!

```
if object_id('dbo.v_AlderGrp', 'V')
  is not null
  drop view dbo.v_AlderGrp
go

create view dbo.v_AlderGrp as
with
Inv as
(
select
    Lan
  , AlderGrp2 =
    case
      when AlderGrp in ('0-4', '5-14')
        then 'Barn'
      else 'Vuxen'
    end
   , Inv
from UTB_Kurs1.dbo.KommunInv2001_2005
where Ar = '2005'
)
select top 100000
    Lan
  , AlderGrp2
  , Inv = Sum(Inv)
from Inv
group by
    Lan
  , AlderGrp2
order by
    Lan
  , AlderGrp2
go

select * from v_AlderGrp
```

Lan	AldreGrp2	Inv
01	Barn	344656
01	Vuxen	1545289
03	Barn	54282
:	:	:

***Beskrivning av vyer**

Syntax

```
create view Vynamn
as
select-sats
```

Man kan spara en selectsats som en textfil, som man hämtar upp vid ett senare tillfälle. Ofta är det smidigare att spara selectsatsen som en vy, eftersom den då kommer att ligga i databasen.

Ibland kan man behöva spara selectsatser i flera steg till som en vy. Då måste de ligga samlade som delfrågor i samma fråga. Det går om man använder sig av subqueries (kap 4.5) eller CTE (kap 4.6). Tänk då på att det kan gå mycket långsamt för stora tabeller. Pröva dig fram!

Vyer är mycket användbart om man ofta ställer samma komplexa frågor. De blir också en bra vägledning till dem, som inte vet vilka frågor som är lämpliga att ställa mot tabellerna i databasen.

En vy är ett sätt att lagra en selectsats under ett namn. Detta gör man med `create view`.

Vyn kan t ex heta `vy_Lan_Kommun,` om den innehåller en fråga som är en join mellan tabellerna Lan och Kommun. När man i fortsättningen vill ställa samma fråga, behöver man bara ställa frågan mot vyn, t ex `select * from vy_Lan_Kommun`. Då blir det mycket mindre att skriva.

En annan fördel är att det tar mindre plats att lagra en vy i databasen än att spara resultatet av frågan som en tabell. Det blir alltså ingen dubbellagring av data.

Normalt kan man inte spara `order by` i en vy, men det fungerar ifall man lägger till `top <tillräckligt antal rader>` i frågan när man definierar vyn. Observera att `top <antal> percent` inte ger sortering om man skriver det i en `create view.`

Det går inte att namnge en vy med #-tecken som man gör med tabeller.

Ibland kan man få felmeddelandet ”… permission denied … ” vilket innebär att man saknar behörighet till vyn eller någon av de tabeller som ingår i vyn. Prata då med den som är databasansvarig och säg att du behöver läsrättighet till vyn och/eller de underliggande tabellerna.

Man använder oftast vyer för läsning även om det för vissa typer av vyer går att göra insert och update mot vyn.

5.5 *Skapa och hantera index

Index underlättar och snabbar upp sökningar i tabeller. Det finns två typer av index: klustrat och icke klustrat.

Om det finns ett klustrat index lagras tabellens rader sorterat på de kolumner som ingår i indexet.

Det bör alltid finnas ett klustrat index i alla indexerade tabeller, eftersom icke klustrade index kan utnyttja det klustrade indexet.

Ett unikt index innebär att inga dubbletter tillåts på de kolumner som ingår i indexet.

- En tabell kan ha flera index, dock bara ett klustrat
- Flera kolumner kan ingå i ett index
- Indexera kolumner som ofta används som join-villkor
- Indexera kolumner som ofta används vid sökning
- Kolumner av datatyperna *varchar(max), nvarchar(max), varbinary(max), text* och *image* kan EJ indexeras
- Undvik att indexera stora kolumner med datatypen *varchar*.
- Snabbar upp **`select`** men en **`insert`** tar längre tid.

* Skapa index

Syntax (förenklad)

```
create [unique] [clustered | nonclustered]
       index indexnamn
on tabellnamn | vynamn (Kolumnnamn [ asc | desc ]
      [,Kolumnnamn]...)
```

Ett klustrat index anges med **clustered** och ett icke klustrat med **nonclustered**. Om inget av dessa alternativ anges är det index som skapas med **create index** icke klustrat.

Om indexet inte ska tillåta dubbletter anges detta med **unique**.

Sorteringsordning anges med **asc** (stigande) eller **desc** (fallande) efter kolumnnamnet. Om inget av dessa alternativ anges blir sorteringsordningen stigande.

Exempel

```
-- Unikt klustrat, en kolumn

create unique clustered index I_KundNr
on Kund(KundNr)

-- Ej klustrat, en kolumn

create index I_KundNamn
on Kund(KundNamn)

-- Icke klustrat, flera kolumner
create index I_KundNamn_KundAdress
on Kund(KundNamn, KundAdress)
```

Namnstandard för index: **I_*Kolumnnamn*[_*Kolumnnamn*]**

*** Ta bort index**

Syntax (förenklad):
```
drop index indexnamn ON Tabellnamn
```

Exempel
```
drop index I_KundNamn on Kund
```

För att kunna ta bort ett index måste man stå i den databas där indexet finns.

Den gamla syntaxen för **`drop index`** är tillåten för bakåtkompatibilitet men rekommenderas inte vid nyutveckling.

Syntax (förenklad)
```
drop index Tabellnamn.indexnamn
```

Exempel
```
drop index Kund.I_KundNamn
```

Kommandona **`create`**, **`alter`** och **`drop`** måste följas av **`go`** i ett sammansatt kommando, batch. En batch är flera SQL-frågor och/eller kommandon som samtidigt skickas till databasservern**.**

Exempel
```
drop index I_KundNamn on Kund
go
```

Observera: go måste stå ensamt först på ny rad.

6. *DML (Data Manipulation Language)

6.1. *Insert

***Exempel**

`*Exempel 1.` Lägger till rader i en tabell. Förutsätter att tabellen #Person finns och att datatyperna stämmer överens.

```
-- Tömmer alla rader i tabellen
delete #Person

-- Lägger till en rad
insert #Person (PersonNr, Namn, Adress)
select  '198012319929', 'Anna Ahl', 'A-Gatan 1'

-- Lägger till en rad
insert #Person (PersonNr, Namn, Adress)
select  '198012318818', 'Bo Ek', 'B-Gatan 1'

-- Tittar på tabellens innehåll
select * from #Person
```

PersonNr	Namn	Adress
198012319929	Anna Ahl	A-Gatan 1
198012318818	Bo Ek	B-Gatan 1

`*Exempel 2.` Lägger till rader i en tabell genom att hämta från en annan tabell. Observera att kolumnen `PostAdress` blir NULL. I alternativ-1 blir kolumnen NULL eftersom den är utelämnad i kolumnlistan. I alternativ-2 blir kolumnen NULL eftersom NULL anges i select-satsen.

```
-- Om tabellen #Person2 finns så tar man bort den
if object_id('tempdb..#Person2', 'U') is not null
  drop table #Person2
go

-- Skapar en ny #-tabell
create table #Person2
(
PersonNr      char(12)    not null
, Namn        char(25)    null
, Adress      varchar(50) null
, PostAdress  varchar(50) null
)
go

-- Alt-1: Kopierar rader m h a en kolumnlista
insert #Person2 (PersonNr, Namn, Adress)
select  *
from #Person

-- Tittar på tabellens innehåll
select * from #Person2
```

```
-- Alt-2: Kopierar rader utan kolumnlista
-- m h a NULL i select-satsen
delete #Person2

insert #Person2
select  *, NULL
from #Person

-- Tittar på tabellens innehåll
select * from #Person2
```

PersonNr	Namn	Adress	PostAdress
198012319929	Anna Ahl	A-Gatan 1	NULL
198012318818	Bo Ek	B-Gatan 1	NULL

*Exempel 3. Lägger till rader i en tabell från en annan tabell beroende på innehållet i tabellerna.

```
-- Kopierar några kolumner från tabellen k_Lan
select
  Lan
  , LanNamn
  , Status = cast('Nya län' as varchar(10))
into #Lan
from UTB_Kurs1.dbo.k_Lan

-- Hämtar gamla län från tabellen Lan som
-- inte finns i #Lan
insert #Lan
select
  B.Lan
  , B.Namn
  , StatusLan = 'Gamla län'
from
  #Lan A
  right outer join UTB_Kurs1.dbo.Lan B
    on A.Lan = B.Lan
where A.Lan is NULL

-- Tittar på tabellen #Lan
select * from #Lan
```

***Beskrivning av insert**

Syntax
insert *tabellnamn* [(*kolumnlista*)]
select-sats

Med kommandot `insert` lägger man till nya rader i en tabell. Man hämtar dem antingen från en annan tabell eller räknar upp värdena för posten.

Det är valfritt att använda en kolumnlista. Om man *inte* använder en kolumnlista, så måste alla kolumner fyllas med värden i den ordning som kolumnerna kommer i tabellen. Om man använder en kolumnlista, så måste ordningsföljden och antalet värden i kolumnlistan stämma överens med kolumnerna i select-satsen.

Datatypen på de data som läggs till tabellen måste stämma överens med tabellens definition.

Alla kolumner som är definierade som `not null` i tabellen måste alltid ges värden. De kolumner som är definierade som `null` behöver inte ges värden. Har man en kolumnlista, så löser man detta genom att utelämna de kolumner i kolumnlistan som ska bli NULL. Har man ingen kolumnlistan, så löser man detta genom att skriva NULL för de aktuella kolumnerna.

Eftersom NULL inte är en textsträng, så ska man inte ha citationstecken.

6.2. *Update

***Exempel**

*Exempel 1. Uppdaterar 2 kolumner. Förutsätter att tabellen finns och att den har ett innehåll.

```
update #Person
set
   Namn = 'Annika Ahl'
  , Adress = 'A-Vägen 1'
where
   PersonNr = '198012319929'

select * from #Person
```

*Exempel 2. Uppdaterar kolumnen Namn för alla rader

```
update #Person
set Namn = 'Nisse'

select * from #Person
```

*Exempel 3. Ersätter alla värden i kolumnen Adress med NULL.

```
update #Person
set Adress = NULL

select * from #Person
```

*Exempel 4. Uppdaterar en kolumn i en tabell utifrån en annan tabell.

```
-- Kopierar några kolumner från tabellen Kommun
select
   Kommun
   , Namn
   , Inv
   , Areal = cast(NULL as bigint)
into #Kommun
from UTB_Kurs1.dbo.Kommun

-- Uppdaterar den tomma kolumnen Areal utifrån tabellen
KommunAreal2000
update A
set Areal = B.Areal
from
   #Kommun A
   join UTB_Kurs1.dbo.KommunAreal2000 B
      on A.Kommun = B.Kommun

-- Kollar vilka rader som uppdaterats i tabellen #Kommun
select * from #Kommun
```

***Beskrivning av update**

Syntax

```
update tabellalias
set kolumnnamn = uttryck [,kolumnnamn = uttryck]
[from join-uttryck]
[where selektionsvillkor]
```

Kommandot **update** förändrar befintliga värden för en kolumn i en tabell på de rader som får träff av selektionsvillkoret och eventuellt join-villkor.

Tänk på att utan ett selektionsvillkor så kommer alla rader i tabellen att förändras till samma värde.

6.3. *Delete

***Exempel**

*Exempel 1. Tar bort de rader där PersonNr = '198012319929'

```
delete #Person
where PersonNr = '198012319929'

select * from #Person
```

*Exempel 2. Tar bort alla rader i tabellen

```
delete #Person

select * from #Person
```

*Exempel 3. Tar bort rader i en tabell beroende på innehållet i bägge tabellerna.

```
-- Kopierar tabellen UTB_Kurs1.dbo.Lan
select *
into #Lan
from UTB_Kurs1.dbo.Lan

-- Tar bort de rader i #Lan som finns i
-- UTB_Kurs1.dbo.KommunAreal2000
delete A
from
  #Lan A
  join UTB_Kurs1.dbo.KommunAreal2000 B
    on A.Lan = B.Lan

-- Visar de rader som blev kvar
select * from #Lan
```

***Beskrivning av delete**

Syntax

```
delete tabellalias
[from join-uttryck]
[where selektionsvillkor]
```

Kommandot **`delete`** tar bort de rader i en tabell som får träff av selektionsvillkoret.

Tänk på att utan ett selektionsvillkor så kommer alla rader i tabellen att tas bort.

7. **Avancerad SQL

7.1.1. ** Lokala variabler

****Exempel**

**Exempel 1.

Med hjälp av lokala variabler kan man räkna och bearbeta data på olika sätt. Deras namn börjar alltid på tecknet @. De måste deklareras innan användning med rätt datatyp. En @variabel raderas automatiskt när en körning är klar.

```
declare @Tal1   int
declare @Tal2   int
declare @Summa  int

select @Tal1 = 5
select @Tal2 = 3

select @Summa = @Tal1 + @Tal2

select Summa = @Summa
```

Summa
8

**Exempel 2.

Ofta används lokala variabler till stränghantering.

```
declare @Tal1   int
declare @Tal2   int
declare @Summa  int
declare @Text   varchar(20)

select @Tal1 = 5
select @Tal2 = 3

select @Summa = @Tal1 + @Tal2

select
  @Text =
  cast(@Tal1 as char(1))
  + ' plus '
  + cast(@Tal2 as char(1))
  + ' = '
  + cast(@Summa as char(1))

select Summering = @Text
```

Summering
5 plus 3 = 8

7.1.2. ** Iterationer

****Exempel**

Exempel 1 skapar ett antal årtal automatiskt. En iteration är en process när något upprepas (loopas) ett antal gånger. Kommandot **while används för detta. **Begin** och **end** används för att avgränsa koden för while-loopen.

```
declare @Artal int

set @Artal = 1995

while @Artal <= 2005
begin
  select @Artal
  select @Artal = @Artal + 1
end
```

**Exempel 2 skapar en tabell med årtal. Här konverteras årtalet från tal till text eftersom årtalet ska lagras som text i tabellen.

```
if object_id('tempdb..#Ar', 'U') is not null
  drop table #Ar
go
create table #Ar (Ar char(4))
go
declare @Ar int
select @Ar = 1995

while @Ar <= 2005
begin
  insert #Ar select cast(@Ar as char(4))
  select @Ar = @Ar + 1
end

select * from #Ar
```

Ar
1995
1996
1997
:

7.1.3. ** Dynamisk SQL

Exempel

```
**Exempel 1
```

Här kopieras en tabell med dynamiskt namn till en tabell med statiskt namn, d v s namnet kan variera och beskrivs med en variabel. Detta måste göras med dynamiskt SQL. Utifrån tabellen med statiskt namn kan man sedan arbeta vidare utan att behöva köra dynamisk SQL. I detta fall är tabellen med statiskt namn en fast tabell. Naturligtvis måste in-tabellen ha kolumnerna Lan och Inv.

All SQL-kod måste köras på en gång, eftersom vi använder lokala @-variabler.

```
-- Ta bort tabellen om tabellen finns
if object_id('UTB_ElevO01.dbo.Tab', 'U') is not
null
  drop table UTB_ElevO01.dbo.Tab
go

-- Deklarera lokala variabler
declare
  @SQL    varchar(MAX)
  , @InTab varchar(30)

-- Välj ett tabellnamn
select @InTab = 'UTB_Kurs1.dbo.Kommun'

-- Bygg upp SQL-satsen och lagra den i en
-- lokal variabel
select @SQL =
    'select Lan, Inv '
  + 'into UTB_ElevO01.dbo.Tab '
  + 'from ' + @InTab

-- Kör den dynamiska SQL-satsen
execute(@SQL)

-- Arbeta vidare utifrån en tabell med
-- statiskt namn
select
    Lan
  , Inv = sum(Inv)
from UTB_ElevO01.dbo.Tab
group by Lan
order by Lan
```

**Exempel 2

Här utföres samma sak som ovan. I detta fall är tabellen med statiskt namn en ##-tabell istället. Fördelen med en ##-tabell är att den städas bort automatiskt när man stänger ned fliken i Management Studio. Nackdelen är att ##-tabellen är tillgänglig för alla på servern.

```
-- Ta bort tabellen om tabellen finns
if object_id('Tempdb..##Tab', 'U') is not null
  drop table ##Tab
go

-- Deklarera lokala variabler
declare
  @SQL    varchar(MAX)
  , @InTab varchar(30)

-- Välj ett tabellnamn
select @InTab = 'UTB_Kurs1.dbo.Kommun'

-- Bygg upp SQL-satsen och lagra den i en
-- lokal variabel
select @SQL =
    ' select Lan, Inv '
  + ' into ##Tab '
  + ' from ' + @InTab

-- Kör den dynamiska SQL-satsen
execute(@SQL)

-- Arbeta vidare utifrån en tabell med
-- statiskt namn
select
    Lan
  , Inv = sum(Inv)
from ##Tab
group by Lan
order by Lan
```

**Exempel 3

Här utföres samma sak som ovan. I detta fall är tabellen med statiskt namn en # -tabell istället. Fördelen med en #-tabell är att den städas bort automatiskt när man stänger ned fönstret eller avslutar en lagrad procedur. Nackdelen är att det blir problem när man skapar #-tabellen med hjälp av dynamisk SQL, eftersom #-tabellen då inte kan nås när den dynamiskt SQL-körningen är klar. Det fungerar däremot om man skapar tabellen tom innan den dynamiska SQL-körningen och gör insert mot den tabellen med hjälp av dynamisk SQL.

```
-- Ta bort tabellen om tabellen finns
if object_id('Tempdb..#Tab', 'U') is not null
  drop table #Tab
go

-- Skapa tom tabell före dynamisk SQL
create table #Tab (Lan char(2), Inv int)
go

-- Deklarera lokala variabler
declare
  @SQL    varchar(MAX)
  , @InTab varchar(30)

-- Välj ett tabellnamn
select @InTab = 'UTB_Kurs1.dbo.Kommun'

-- Bygg upp SQL-satsen och lagra den i en
-- lokal variabel
select @SQL =
    'insert #Tab '
  + 'select Lan, Inv '
  + 'from ' + @InTab

-- Kör den dynamiska SQL-satsen
execute(@SQL)

-- Arbeta vidare utifrån en tabell med
-- statiskt namn
select
    Lan
  , Inv = sum(Inv)
from #Tab
group by Lan
order by Lan
```

**Exempel 4

Här byggs en SQL-sats upp, med hjälp av en **mängdorienterad loop**. Lägg märke till, att man måste man skriva 2 fnuttar innanför de omgivande fnuttarna för att kunna skapa 1 fnutt i select-satsen. Detta ”fnuttande” kan upplevas som besvärligt, men underlättas om man tittar på sin SQL-sats i @SQL-variabeln. I MS SQL-server får man automatiskt hjälp att titta på den dynamiska SQL-koden i debugern.

Välj att resultatfönstret ska vara ett textfönster, för att kunna titta på SQL-koden som skapas dynamiskt.

SQL-satsen skapar en ny tabell där alla kommunnamn från tabellen Kommun för ett visst län, konkateneras till en cell i den nya tabellen.

```
-- Stäng av NULL-varningen
set ansi_warnings off

-- Stäng av visningen av antal rader i resultatet
set nocount on

-- Ta bort tabellen om tabellen finns
if object_id('tempdb..#Kommuner', 'U') is not null
drop table #Kommuner
go

-- Skapa tabellen för den konkatenerade cellen
create table #Kommuner
(
Lan         char(2)
, Kommuner  varchar(8000)
)
go

-- Deklarera variabeln som ska innehålla SQL-koden
declare @SQL varchar(MAX)
declare @Lan char(2)

select @Lan = '03'

-- Bygg början av insert-satsen
select @SQL = 'insert #Kommuner '
   + 'select Lan = ''' + @Lan + ''', Kommuner = '''

-- Mängdorienterad loop för konkatenering av raderna
select @SQL = @SQL + Namn + ', '
from UTB_Kurs1.dbo.Kommun
where Lan = @Lan
order by Namn

-- Ta bort det sista kommatecknet
select @SQL = left(@SQL, len(@SQL) - 1)

-- Lägg till en fnutt i slutet på textsträngen
select @SQL = @SQL + ''''

-- Titta på SQL-koden
select SQLKod = @SQL

-- Kör den dynamiska SQL-satsen
execute(@SQL)

-- Titta på innehållet i tabellen
select * from #Kommuner
```

**Exempel 5

Här byggs en SQL-sats upp, med hjälp av en **mängdorienterad loop**.

SQL-satsen skapar en **pivoterad tabell** på skärmen **utan förspalt**. Överspalten innehåller länskoderna och cellerna innehåller innevånarantalet år 2005.

```
-- Stäng av NULL-varningen
set ansi_warnings off

-- Stäng av visningen av antal rader i resultatet
set nocount on

-- Skapa en ny-rad-konstant
declare @NR char(2)
set @NR = char(13) + char(10)

-- Deklarera variabeln som ska innehålla SQL-koden
declare @SQL varchar(MAX)

-- Bygg början av select-satsen
select
  @SQL =
    'select' + @NR
    + '    ''Totalt''' + @NR

-- Bygg mittersta delen av select-satsen med en
-- mängdorienterad loop
-- som skapar SQL-kod för ett dynamiskt antal pivot-kolumner
-- Kolumnnamnen måste omgärdas av hakparenteser
-- eftersom de börjar med en siffra
-- I from-bisatsen skapas värdemängden för överspalten
select
  @SQL = @SQL +
  + '  , [' + Ar + '] = '
  + 'sum(case when Ar = ''' + Ar
  + ''' then Inv else NULL end)'+ @NR
from (select distinct Ar from UTB_Kurs1.dbo.KommunInv2001_2005) A
order by Ar

-- Bygg slutet av select-satsen
select
  @SQL = @SQL
  + 'from UTB_Kurs1.dbo.KommunInv2001_2005'

-- Titta på SQL-koden
select SQLKod = @SQL

-- Kör den dynamiska SQL-satsen
exec(@SQL)
```

**Exempel 6

Här byggs en SQL-sats upp, med hjälp av en **mängdorienterad loop**.

SQL-satsen skapar en **pivoterad tabell** på skärmen **med förspalt** som innehåller kommunnamnen. Överspalten innehåller länskoderna och cellerna innehåller innevånarantalet år 2005.

```
-- Stäng av NULL-varningen
set ansi_warnings off

-- Stäng av visningen av antal rader i resultatet
set nocount on

-- Skapa en ny-rad-konstant
declare @NR char(2)
set @NR = char(13) + char(10)

-- Deklarera variabeln som ska innehålla SQL-koden
declare @SQL varchar(MAX)

-- Bygg början av select-satsen
select
  @SQL =
    'select' + @NR
  + '    KommunNamn' + @NR

-- Bygg mittersta delen av select-satsen
select
  @SQL = @SQL
  + '  , [' + Ar + '] = ' +
  + 'sum(case when Ar = ''' + Ar + ''' then Inv else NULL end)'+ @NR
from (select distinct Ar from UTB_Kurs1.dbo.KommunInv2001_2005) A
order by Ar

-- Bygg slutet av select-satsen
select
  @SQL = @SQL
  + 'from UTB_Kurs1.dbo.KommunInv2001_2005' + @NR
  + 'group by KommunNamn' + @NR
  + 'order by KommunNamn' + @NR

-- Titta på SQL-koden
select SQLKod = @SQL

-- Kör den dynamiska SQL-satsen
exec(@SQL)
```

**Exempel 7

Här byggs en SQL-sats upp, med hjälp av en **mängdorienterad loop**.

SQL-satsen skapar en **kolumnorienterad tabell** (KommunInv2001_2005_KolTab) utifrån en **radorienterad tabell** (KommunInv2001_2005_RadTab_Meta) och en **metatabell** (KommunInv2001_2005_RadTab_Meta) som beskriver kolumnerna.

```
-- Stäng av NULL-varningen
set ansi_warnings off

-- Stäng av visningen av antal rader i resultatet
set nocount on

-- Ta bort tabellen om tabellen finns
if object_id('UTB_ElevO01.dbo.KommunInv2001_2005_KolTab', 'U') is
not null
  drop table UTB_ElevO01.dbo.KommunInv2001_2005_KolTab
go

-- Skapa en ny-rad-konstant
declare @NR char(2)
set @NR = char(13) + char(10)

-- Deklarera variabeln som ska innehålla SQL-koden
declare @SQL varchar(MAX)

-- Bygg början av select-satsen
-- RadNr i KommunInv2001_2005_RadTab
-- knyter ihop alla kolumner till en rad
-- vid vridning till kolumnorienterad tabell
select
  @SQL =
    'select' + @NR
  + '    RadNr' + @NR

-- Bygg mittersta delen av select-satsen
-- Cast() behövs för att KolNr måste
-- konverteras till text för att kunna konkateneras
select
  @SQL = @SQL
  + '  , ' + Variabel + ' = ' +
  + 'min(case when KolNr = '
  + cast(KolNr as varchar(10))
  + ' then cast(Varde as ' + DataTyp +  ') else NULL end)'+ @NR
from UTB_Kurs1.dbo.KommunInv2001_2005_RadTab_Meta
order by KolNr

-- Bygg slutet av select-satsen
select
  @SQL = @SQL
  + 'into UTB_ElevO01.dbo.KommunInv2001_2005_KolTab' + @NR
  + 'from UTB_Kurs1.dbo.KommunInv2001_2005_RadTab' + @NR
  + 'group by RadNr' + @NR
  + 'order by RadNr' + @NR

-- Titta på SQL-koden
select SQLKod = @SQL

-- Kör den dynamiska SQL-satsen
exec(@SQL)
```

7.2. **Dubbletthantering

****Exempel**

Dubbletter i en tabell

Exempel 1
Visa unika rader med hjälp av distinct, d v s dölj dubbletterna i tabellen.

```
select distinct *
from Lan2
```

*Exempel 2
Visa antal unika rader med hjälp av en subquery.

```
select count(*)
from
  (select distinct *
  from Lan2) as T
```

Exempel 3
Visa vilka rader och antal förekomster, som det finns dubbletter på med hjälp av `having count(*) > 1`. Man måste räkna upp alla kolumner.

```
select
  Lan
  , Namn
  , Areal
  , AntalRader = count(*)
from Lan2
group by
  Lan
  , Namn
  , Areal
having count(*) > 1
```

Exempel 4
Visa vilka rader, som det inte finns dubbletter på med hjälp av `having count(*) = 1`. Man måste räkna upp alla kolumner.

```
select
  Lan
  , Namn
  , Areal
from Lan2
group by
  Lan
  , Namn
  , Areal
having count(*) = 1
```

**Exempel 5

Ta bort dubblettrader i en tabell i tre steg.

```
----- Skapa test-tabellen
select *
into UTB_ElevO01.dbo.Lan2
from UTB_Kurs1.dbo.Lan2

----- Steg-1: Spara unika rader i en temporär tabell
select distinct *
into #A
from UTB_ElevO01.dbo.Lan2

----- Steg-2: Töm tabellen
delete UTB_ElevO01.dbo.Lan2

----- Steg-3: Fyll tabellen med de unika raderna
insert UTB_ElevO01.dbo.Lan2
select *from #A
```

Exempel 6

Visa unika värden för en kolumn d v s värdemängden för kolumnen.

```
select distinct Lan
from Lan2
```

Exempel 7

Visa vilka värden på Lan som förekommer flera gånger.

```
select Lan
from Lan2
group by Lan
having count(*) > 1
```

*Exempel 8

Visa hela raderna för de län som förekommer flera gånger.

```
select *
from Lan2
where
  Lan in
  (
  select Lan
  from Lan2
  group by Lan
  having count(*) > 1
  )
order by
  Lan
  , Namn
```

Lan	Namn	Areal
01	Stockholms län	6490
01	Stockholms län	6490
03	Uppsala län	6989
:	:	:

**Exempel 9

Visa senaste kommunnamnet för en kommunkod om man har dubbletter i kolumnen kommun.

```
select
  KommunKod = Kommun
  , SenasteKommunNamn = substring(max(Ar +
KommunNamn), 5, 99)
from Kommun1993_2008
group by Kommun
having count(distinct KommunNamn) > 1
```

KommunKod	SenasteKommunNamn
2023	MALUNG-SÄLEN
0114	UPPLANDS VÄSBY

Exempel 10

Visa dubblettinfo.

```
-- Visa info om dubbletter på kommunkoderna
select
  KommunKod = Kommun
  ----- Antal dubblettrader
  , Antal1 = count(*)
  ----- Antal unika kommunnamn
  , Antal2 = count(distinct KommunNamn)
  , MinKommunNamn = min(KommunNamn)
  , MaxKommunNamn = max(KommunNamn)
from Kommun1993_2008
group by Kommun
having count(distinct KommunNamn) > 1
```

KommunKod	Antal1	Antal2	MinKommunNamn	MaxKommunNamn
0114	16	2	UPPLANDS VÄSBY	UPPLANDS-VÄSBY
2023	16	2	MALUNG	MALUNG-SÄLEN

```
-- Visa info om dubbletter på kommunnamnen
select
  KommunNamn
  ----- Antal dubblettrader
  , Antal1 = count(*)
  ----- Antal unika kommunkoder
  , Antal2 = count(distinct Kommun)
  , MinKommunKod = min(Kommun)
  , MaxKommunKod = max(Kommun)
from Kommun1993_2008
group by KommunNamn
having count(distinct Kommun) > 1
```

KommunNamn	Antal1	Antal2	MinKommunKod	MaxKommunKod
ALE	16	2	1440	1521
ALINGSÅS	16	2	1489	1582
:	:	:	:	:

*Exempel 11

Visa hela raderna för de kommunkoder som har mer än en kombination med ett kommunnamn.

```
select *
from Kommun1993_2008
where
  Kommun in
  (
  -----  Alla kommunkoder som har mer än en
kombination med       -----  KommunNamn
  select Kommun
  from Kommun1993_2008
  group by Kommun
  having count(distinct KommunNamn) > 1
  )
order by
  Kommun
  , Ar
```

Kommun	Lan	Ar	KommunNamn
0114	01	1993	UPPLANDS-VÄSBY
0114	01	1994	UPPLANDS-VÄSBY
:	:	:	:
0114	01	2003	UPPLANDS VÄSBY
:	:	:	:
2023	20	1993	MALUNG
:	:	:	:
2023	20	2008	MALUNG-SÄLEN

*Exempel 12

Visa alla kombinationer av kommunkod och kommunnamn i tabellen.

```
select distinct
  KommunKod = Kommun
  , KommunNamn
from Kommun1993_2008
where
  Kommun in
  (
  ----- Alla kommunkoder som har mer än en kombination
  ----- med KommunNamn
  select Kommun
  from Kommun1993_2008
  group by Kommun
  having count(distinct KommunNamn) > 1
  )
```

KommunKod	KommunNamn
0114	UPPLANDS VÄSBY
0114	UPPLANDS-VÄSBY
2023	MALUNG
2023	MALUNG-SÄLEN

Dubbletter mellan tabeller

**Exempel 13
Visa identiska rader från två tabeller, med hjälp **intersect**, d v s de hela rader som är dubbletter mellan tabellerna.

```
select *
from Kommun2
intersect
select *
from Kommun
```

Kommun	Namn	Inv	Lan	Tskatt	Kskatt	KommunGrp	Inv0715	Inv1619	GrundKost	GymKost
0114	Upplands-Väsby	35764	01	31.25	17.79	2	4062	2173	240663	86339
0115	Vallentuna	23061	01	29.48	15.89	2	2825	1270	139852	57157
0117	Österåker	31600	01	29.31	15.80	2	3796	1940	175995	66359
0120	Värmdö	25193	01	30.77	17.29	2	2724	1234	127994	47471
0123	Järfälla	57638	01	30.58	17.35	2	6511	3290	288919	54443
:	:	:	:	:	:	:	:	:	:	:

**Exempel 14
Lägg bara till de rader från tabellen Kommun, som inte finns i tabellen #Kommun2, med hjälp **except**, d v s lägg inte till dubblettraderna mellan tabellerna. Tabellen #Kommun2 har 10 rader med NULL-förekomster. Motsvarande 10 rader i tabellen Kommun har värden i alla kolumner. Övriga rader är dubblettrader mellan tabellerna. Det är de 10 olika raderna som laddas från tabellen Kommun till tabellen #Kommun2.

```
----- Skapa test-tabell
select *
into #Kommun2
from Kommun2

----- Lägg till ej identiska rader
insert #Kommun2
select *
from Kommun
except
select *
from #Kommun2

----- Titta på alla rader
select *
from #Kommun2
order by Kommun
```

Kommun	Namn	Inv	Lan	Tskatt	Kskatt	KommunGrp	Inv0715	Inv1619	GrundKost	GymKost
:	:	:	:	:	:	:	:	:	:	:
0604	Aneby	7269	06	NULL	18.25	NULL	986	398	49471	12684
0604	Aneby	7269	06	31.35	18.25	6	986	398	49471	12684
:	:	:	:	:	:	:	:	:	:	:

**Exempel 15

Visa de rader som har identiska nycklar mellan tabellerna, d v s de rader som har dubblett på nyckelkolumnerna.

```
select distinct A.Namn, B.KommunNamn
from
  Kommun A
  join
  KommunInv2001_2005 B
    on A.Namn = B.KommunNamn
```

Namn	KommunNamn
Ale	Ale
Alingsås	Alingsås
Alvesta	Alvesta
:	:

**Exempel 16

Lägg bara till de rader från tabellen KommunInv2001_2005, som inte matchar tabellen Kommun på nyckelkolumnerna KommunNamn och Namn, d v s lägg bara till raderna som inte har dubblett mellan tabellerna på nyckelkolumnerna. 5 rader med nya KommunNamn och dess tillhörande Inv läggs till.

```
----- Skapa test-tabell
select
  KommunNamn = Namn
  , Inv_Gammal = Inv
  , Inv_Ny = 0
into #Kommun
from Kommun

----- Skapa en temporär-tabell för alla kommunnamn
----- och innevånarantal år 2005
select
  KommunNamn
  , Inv = sum(Inv)
into #A
from KommunInv2001_2005
where Ar = '2005'
group by KommunNamn

----- Lägg till rader med ej identiska nyckelkolumner
insert #Kommun
select
  A.KommunNamn
  , 0
  , A.Inv
from
  #A A
  full outer join
  #Kommun B
    on A.KommunNamn = B.KommunNamn
where B.KommunNamn is NULL

----- Titta på alla rader
select * from #Kommun order by KommunNamn
```

KommunNamn	Inv_Gammal	Inv_Ny
Ale	25204	0
:	:	:
Bollebygd	0	8086
:	:	:

**Exempel 17

Fortsätt att arbeta med tabellerna #Kommun och #A. Uppdatera kolumnerna i tabell #Kommun där raderna i tabell #Kommun matchar mot raderna i tabell #A på nyckelkolumnerna, d v s uppdatera om det finns dubblett på nyckelkolumnerna mellan tabellerna.

```
----- Uppdatera dubblettkolumnerna
update B
set Inv_Ny = A.Inv
from
  #A A
  join
  #Kommun B
    on A.KommunNamn = B.KommunNamn

----- Titta på alla rader
select *
from #Kommun
order by KommunNamn
```

KommunNamn	Inv_Gammal	Inv_Ny
Ale	25204	26405
Alingsås	34212	36010
Alvesta	19813	18684
:	:	:
Bollebygd	0	8086
:	:	:
Upplands Väsby	0	37624
Upplands-Bro	19848	21327
Upplands-Väsby	35764	0
:	:	:

7.3. *Pivotering

***Exempel**

*Exempel 1

- Beräkning av antal personer per kön samt antal personer totalt
- Utan förspalt d v s en resultatrad
- Statiska kolumner

Resultat

Antal_Kvinnor	Antal_Man	Totalt
5	5	10

*Exempel 1a

Lösning med case-satsen. Denna metod är flexiblast eftersom den medger att man kan skriva olika villkor för olika kolumner, som man skapar.

```
select
  Antal_Kvinnor  = sum(case when Kon = 'K' then 1 else 0 end)
  , Antal_Man    = sum(case when Kon = 'M' then 1 else 0 end)
  , Totalt       = count(*)
from Man_Kvinna
```

**Exempel 1b

Lösning med pivot-funktionen tillsammans med CTE.
CTE = Common Table Expression. Innebär att en temporär tabell skapas med with precis före select-satsen och den kommer bara att existera inom den select-satsen. Denna metod ger ofta bra prestanda.

```
with Aggregat
as
  (
select
  Kon
  , Antal = count(*)
from Man_Kvinna
group by
  Kon
)
select
  Antal_Kvinnor= K
  , Antal_Man= M
  , Totalt = K + M
from Aggregat
  pivot
  (
  sum(Antal)
    for Kon in
    (
    K
    , M
    )
  ) Piv
```

**Exempel 1c

Lösning med stegvis uppdatering av pivotkolumnerna.
Denna metod ger ofta sämre prestanda. Den är lätt att förstå men ger onödigt mycket SQL-kod att skriva.

```
-- Droppar tabellen om den finns
if object_id('tempdb..#Aggregat1', 'U') is not null
  drop table #Aggregat1

if object_id('tempdb..#Aggregat2', 'U') is not null
  drop table #Aggregat2

if object_id('tempdb..#Pivot', 'U') is not null
  drop table #Pivot
go

-- Skapar och fyller första aggregat-tabellen
-- som innehåller antal personer per kön
select
  Kon
  , Antal = count(*)
into #Aggregat1
from Man_Kvinna
group by
  Kon

-- Skapar och fyller andra aggregat-tabellen
-- som innehåller antal personer totalt
select Antal = count(*)
into #Aggregat2
from Man_Kvinna

-- Skapar en tom pivot-tabell som sedan ska fyllas
create table #Pivot
(
Antal_Kvinnor int
, Antal_Man    int
, Totalt     int
)

-- Lägger till en rad med värde i första kolumnen
insert #Pivot(Antal_Kvinnor)
select Antal_Kvinnor = Antal
from #Aggregat1
where Kon = 'K'

-- Uppdaterar andra kolumnen
update #Pivot
set Antal_Man = Antal
from #Aggregat1
where Kon = 'M'

-- Uppdaterar tredje kolumnen
update #Pivot
set Totalt = Antal
from #Aggregat2

-- Tittar på resultatet
select * from #Pivot
```

*Exempel 2

- Beräkning av innevånare per år och komunnamn
- Dynamisk förspaltskolumn
- Statiska pivotkolumner eller dynamiska pivotkolumner
- Sortering av rader efter komunnamn

Resultat

KommunNamn	2001	2002	2003	2004	2005
Ale	25593	25835	25993	26288	26405
Alingsås	35257	35327	35530	35761	36010
:	:	:	:	:	:

*Exempel 2a

Lösning med case-satsen. Med denna metod skapas pivotkolumnerna statiskt med hårdkodad i SQL-kod, d v s bara de kolumner som finns beskrivna i SQL-koden kommer att skapas.

```
select
  KommunNamn
  , [2001] = sum(case Ar when '2001' then Inv else 0 end)
  , [2002] = sum(case Ar when '2002' then Inv else 0 end)
  , [2003] = sum(case Ar when '2003' then Inv else 0 end)
  , [2004] = sum(case Ar when '2004' then Inv else 0 end)
  , [2005] = sum(case Ar when '2005' then Inv else 0 end)
from KommunInv2001_2005
group by KommunNamn
order by KommunNamn
```

**Exempel 2b

Lösning med pivot-funktionen tillsammans med CTE.
Denna metod skapar pivot-kolumnerna statiskt.

```
with Aggregat
as
(
select
  KommunNamn
  , Ar
  , Inv = sum(Inv)
from KommunInv2001_2005
group by
  KommunNamn
  , Ar
)
select *
from Aggregat
  pivot
  (
    sum(Inv)
    for Ar in
      (
      [2001]
      , [2002]
      , [2003]
      , [2004]
      , [2005]
      )
  ) Piv
order by KommunNamn
```

**Exempel 2c

Lösning med case-satsen och dynamisk SQL. Med denna metod skapas pivotkolumnerna dynamiskt i SQL-koden, d v s pivotkolumnerna behöver inte beskrivas manuellt i SQL-koden utan skapas automatiskt beroende på värdemängden i intabellen. Nackdelen är att det krävs mer programmering med lite besvärlig stränghantering för att bygga upp en dynamisk SQL-kod. Fördelen är att när den dynamiska SQL-koden är skriven, så går den att återanvända mot olika värdemängder och olika intabeller t ex om ett nytt år läggs till.

```
--=========================================================
-- Steg-0: Deklarationer och inparametrar
--=========================================================
-- Visning av varningsmeddelande för t ex NULL-förekomster vid
aggregering
set ansi_warnings off
-- Visning för antalet rader efter en SQL-sats
set nocount on

-- Droppar tabellen om den finns
if object_id('tempdb..##Agg', 'U') is not null
  drop table ##Agg

if object_id('tempdb..#KolumnNamn', 'U') is not null
  drop table #KolumnNamn
go

----------------
-- Deklarationer
----------------
declare
-- Den tabell som man hämtar data från t ex KommunInv2001_2005
  @InTab                  varchar(100)
-- Den variabel som blir förspalt t ex KommunNamn
  , @FSpalt               varchar(8000)
-- Den variabel som innehåller värdemängden för pivotkolumnernas
-- namn t ex Ar
  , @KolumnVariabel       varchar(8000)
-- Det variabel som aggregatberäkningen utförs på t ex Inv
  , @VardeVariabel        varchar(8000)
-- Den aggregeringsfunktion som beräkningen utförs med t ex sum,
-- count, avg
  , @AggFunk              varchar(8000)
-- Här lagras SQL-koden som sedan exekveras dynamiskt
  , @SQL1                 varchar(8000)

-------------------------------------------------
-- Inparametrar vars värden kan ändras efter behov
-------------------------------------------------
select
 @InTab             = 'UTB_Kurs1.dbo.KommunInv2001_2005'
 , @FSpalt          = 'KommunNamn'
 , @KolumnVariabel  = 'Ar'
 , @VardeVariabel   = 'Inv'
 , @AggFunk         = 'sum'
```

```
--============================================================
-- Steg-1: Aggregat-tabell skapas
--============================================================
select
  @SQL1 =
     ' select '
    + ' FSpalt = ' + @FSpalt
     + ', KolumnNamn = ' + @KolumnVariabel
     + ', Varde = ' + @AggFunk + '(' + @VardeVariabel + ') '
     + ' into ##Agg '
     + ' from ' +  @InTab
     + ' group by '
     + @FSpalt
     + ', '+ @KolumnVariabel
     + ' order by '
     + @FSpalt
     + ', ' + @KolumnVariabel

execute (@SQL1)

--============================================================
-- Steg-2: Pivot-tabell skapas utifrån aggregat-tabellen
--============================================================
select distinct KolumnNamn
into #KolumnNamn
from ##Agg

select
  @SQL1 =
   ' select '
   + @FSpalt + ' = FSpalt '

select
   @SQL1 =  @SQL1
   + ', [' + KolumnNamn + '] = '
   + ' min(case when KolumnNamn = ''' + KolumnNamn + ''''
   + ' then Varde' + ' else NULL end) '
from #KolumnNamn
order by KolumnNamn

select
   @SQL1 =  @SQL1
   + ' from ##Agg '
   + ' group by FSpalt '
   + ' order by FSpalt '
--select SQL1 = @SQL1
execute (@SQL1)
```

*Exempel 3a

- Beräkning av antal innevånare per år
  med 3 olika delummor: Totalt + Lan + KommunNamn
- Dynamisk förspaltskolumn
- Statiska pivotkolumner
- Sortering av rader efter skapad sorteringskolumn

Resultat

Totalt	Lan	KommunNamn	Ar2001	Ar2002	Ar2003	Ar2004	Ar2005	Sort
Totalt	NULL	NULL	8909128	8940788	8975670	9011392	9047752	1_Totalt
NULL	01	NULL	1838882	1850467	1860872	1872900	1889945	2_Lan-01
NULL	01	Botkyrka	74151	75216	75432	75830	76592	2_Lan-01_Kommun-Botkyrka
NULL	01	Danderyd	29632	29755	29884	30100	30226	2_Lan-01_Kommun-Danderyd
:	:	:	:	:	:	:	:	:

Lösning med case-satsen. Här behövs det 3 olika select-satser som grupperas på olika sätt och som sedan slås ihop med union-operatorn. Sort-kolumen skapas med en formel och kodas på olika sätt beroende på hur man vill att raderna ska sorteras.

```
Select
  Totalt = 'Totalt'
  , Lan = NULL
  , KommunNamn = NULL
  , Ar2001   = sum(case when Ar = '2001' then Inv else 0 end)
  , Ar2002   = sum(case when Ar = '2002' then Inv else 0 end)
  , Ar2003   = sum(case when Ar = '2003' then Inv else 0 end)
  , Ar2004   = sum(case when Ar = '2004' then Inv else 0 end)
  , Ar2005   = sum(case when Ar = '2005' then Inv else 0 end)
  , Sort = '1_Totalt'
from KommunInv2001_2005
union
select
  Totalt = NULL
  , Lan
  , KommunNamn = NULL
  , Ar2001   = sum(case when Ar = '2001' then Inv else 0 end)
  , Ar2002   = sum(case when Ar = '2002' then Inv else 0 end)
  , Ar2003   = sum(case when Ar = '2003' then Inv else 0 end)
  , Ar2004   = sum(case when Ar = '2004' then Inv else 0 end)
  , Ar2005   = sum(case when Ar = '2005' then Inv else 0 end)
  , Sort = '2_Lan-' + Lan
from KommunInv2001_2005
group by
  Lan
union
select
  Totalt = NULL
  , Lan
  , KommunNamn
  , Ar2001   = sum(case when Ar = '2001' then Inv else 0 end)
  , Ar2002   = sum(case when Ar = '2002' then Inv else 0 end)
  , Ar2003   = sum(case when Ar = '2003' then Inv else 0 end)
  , Ar2004   = sum(case when Ar = '2004' then Inv else 0 end)
  , Ar2005   = sum(case when Ar = '2005' then Inv else 0 end)
  , Sort = '2_Lan-' + Lan + '_Kommun-' + KommunNamn
from KommunInv2001_2005
group by
  Lan
  , KommunNamn
order by
  Sort
```

*Exempel 3b

Alternativ lösning i två steg.

```
select
  Totalt = 'Totalt'
  , Lan = NULL
  , KommunNamn = NULL
  , Ar
  , Inv = sum(Inv)
  , Sort = '1_Totalt'
into #Agg
from KommunInv2001_2005
group by
  Ar
union
select
  Totalt = NULL
  , Lan
  , KommunNamn = NULL
  , Ar
  , Inv = sum(Inv)
  , Sort = '2_Lan-' + Lan
from KommunInv2001_2005
group by
  Ar
  , Lan
union
select
  Totalt = NULL
  , Lan
  , KommunNamn
  , Ar
  , Inv = sum(Inv)
  , Sort = '2_Lan-' + Lan + '_Kommun-' + KommunNamn
from KommunInv2001_2005
group by
  Ar
  , Lan
  , KommunNamn
order by
  Ar
  , Lan
  , KommunNamn

select
  Totalt
  , Lan
  , KommunNamn
  , Ar2001   = sum(case when Ar = '2001' then Inv else 0 end)
  , Ar2002   = sum(case when Ar = '2002' then Inv else 0 end)
  , Ar2003   = sum(case when Ar = '2003' then Inv else 0 end)
  , Ar2004   = sum(case when Ar = '2004' then Inv else 0 end)
  , Ar2005   = sum(case when Ar = '2005' then Inv else 0 end)
  , Sort
from #Agg
group by
  Totalt
  , Lan
  , KommunNamn
  , Sort
order by
  Sort
```

**Exempel 4

- Pivotering från radorienterad tabell till kolumnorienterad tabell
- Dynamisk förspaltskolumn
- Statiska pivotkolumner
- Sortering av rader efter radnr

RadTab

RadNr	KolumnNr	Variabel	Varde
1	1	FNamn	Adam
1	2	ENamn	Al
1	3	Telefon	112233
2	1	FNamn	Bertil
2	2	ENamn	Bark
2	3	Telefon	223344
3	1	FNamn	Carl
3	2	ENamn	Carlsson
3	3	Telefon	334455

KolTab

RadNr	FNamn	ENamn	Telefon
1	Adam	Al	112233
2	Bertil	Bark	223344
3	Carl	Carlsson	334455

Lösning med case-satsen.

```
select
  RadNr
  , FNamn   = min(case  when Variabel = 'FNamn'
                        then Varde else NULL end)
  , ENamn   = min(case  when Variabel = 'ENamn'
                        then Varde else NULL end)
  , Telefon = min(case  when Variabel = 'Telefon'
                        then Varde else NULL end)
from RadTab
group by RadNr
order by RadNr
```

*Exempel 5

Pivoterar antal kommuner per skattegrupp och med en förspalt för län. Jämför med Exempel 4 i kapitel 4.4. om Case-satsen.

```
select
  Lan
  , TSkatt_29   = sum(case when TSkatt < 29 then 1
else 0 end)
  , TSkatt2932  = sum(case when TSkatt >=29 and TSkatt
< 32
                    then 1 else 0 end)
  , TSkatt32_   = sum(case when TSkatt >= 32 then 1
else 0 end)
  , Totalt      = count(*)
from Kommun
group by
  Lan
order by
  Lan
```

Lan	TSkatt_29	TSkatt2932	TSkatt32_	Totalt
01	5	20	0	25
03	0	3	3	6
04	0	9	0	9
:	:	:	:	:

****Beskrivning av pivot()**

Syntax

```
select förspaltskolumner, rubrikkolumn, värdekolumn |*
 [into tabell]
 from tabell
 pivot  (
          aggregeringsfunktion(värdekolumn)
          for rubrikkolumn in
          kolumnlista
        ) tabellalias
```

- Pivotering innebär (se bild nedan!) att Tabell A vrids upp till Tabell B. Man kan också uttrycka det som en transformering av tabellen från radorienterat format till kolumnorienterat, d v s många rader blir många kolumner.
- I syntaxen motsvarar rubrikkolumn kolumnen Rubrik i Tabell A och kolumnlista kolumnerna A, B och C i tabell B.
- Värdemängden i kolumnen Rubrik i Tabell A blir rubriker i egna pivot-kolumner i Tabell B. Värdena i kolumnen Värde i Tabell A blir värden i de nya kolumnerna i Tabell B. Förspalten ser likadan ut i nya tabellen.
- Tabellalias måste alltid anges.
- Vilka kolumner som ska vara med i pivoteringen anges i kolumnlistan eller med '*'.

Arbetsgång vid **pivot()**

1. Vid behov d v s om det finns fler kolumner i utgångstagbellen än vad som ska tas med i resultatet; Skapa en temporärtabell med CTE med
   - Den eller de kolumner som ska vara förspalt (förspaltskolumn)
   - De kolumner som ska pivoteras.
2. Pivotera med **pivot()**

Tabell A

Förspalt	Rubrik	Värde
XXX	A	1000
XXX	B	500
XXX	C	1500
YYY	A	2000
YYY	B	750
YYY	C	1200

Tabell B

Förspalt	A	B	C
XXX	1000	500	1500
YYY	2000	750	1200

7.4. **Återpivotering

****Exempel**

**Exempel 1

Återpivotering av med hjälp av funktionen **unpivot().**

```
select KommunNamn, Ar, Inv
from KommunInv2001_2005_Piv
  unpivot
  (Inv for Ar in
     (
     [2001]
     ,[2002]
     ,[2003]
     ,[2004]
     ,[2005]
     )
  ) P
```

KommunNamn	Ar	Inv
Ale	2001	25593
Ale	2002	25835
Ale	2003	25993
:	:	:

**Exempel 2

Alternativt sätt med union. Nackdelen är mer kod. Fördelen är större flexibilitet.

```
select
   KommunNamn
   , Ar = '2001'
   , Inv = [2001]
from KommunInv2001_2005_Piv
union
select
   KommunNamn
   , Ar = '2002'
   , Inv = [2002]
from KommunInv2001_2005_Piv
union
select
   KommunNamn
   , Ar = '2003'
   , Inv = [2003]
from KommunInv2001_2005_Piv
union
select
   KommunNamn
   , Ar = '2004'
   , Inv = [2004]
from KommunInv2001_2005_Piv
union
select
   KommunNamn
   , Ar = '2005'
   , Inv = [2005]
from KommunInv2001_2005_Piv
order by
   KommunNamn
   , Ar
```

**Exempel 3

Återpivotering av den kolumnorienterade tabell KolTab med hjälp av union. Se Exempel 4 i kapitel 7.4 om pivotering. Fungerar ej med unpivot().

```
select
 RadNr
 , KolumnNr = '1'
 , Variabel = 'FNamn'
 , Varde = FNamn
from KolTab
union
select
 RadNr
 , KolumnNr = '2'
 , Variabel = 'ENamn'
 , Varde = ENamn
from KolTab
union
select
 RadNr
 , KolumnNr = '3'
 , Variabel = 'Telefon'
 , Varde = Telefon
from KolTab
order by
 RadNr
 , KolumnNr
```

RadNr	KolumnNr	Variabel	Varde
1	1	FNamn	Adam
1	2	ENamn	Al
1	3	Telefon	112233
2	1	FNamn	Bertil
2	2	ENamn	Bark
2	3	Telefon	223344
3	1	FNamn	Carl
3	2	ENamn	Carlsson
3	3	Telefon	334455

****Beskrivning av unpivot()**

Syntax

```
select förspaltskolumner, rubrikkolumn, värdekolumn |*
 [into tabell]
 from tabell
 unpivot (
         värdekolumn
         for rubrikkolumn in
         kolumnlista
         ) tabellalias
```

- Operatorn **`unpivot()`** är motsatsen till **`pivot().`**
- Återpivotering innebär (se bild nedan!) att Tabell B vrids tillbaka till Tabell A. Man kan också uttrycka det som en transformering av tabellen från kolumnorienterat format till radorienterat, d v s många kolumner blir många rader.
- I syntaxen motsvarar rubrikkolumn kolumnen Rubrik i Tabell A och kolumnlista kolumnerna A, B och C i tabell B.
- Rubrikerna i pivot-kolumnerna i Tabell B blir till värden i kolumnen Rubrik i Tabell A. Värdena i pivot-kolumnenerna i Tabell B blir värden i kolumnen Värde i Tabell A. Förspalten ser likadan ut i nya tabellen.
- Tabellalias måste alltid anges.
- Vilka kolumner som ska vara med i pivoteringen anges i *kolumnlista* eller med '*'.

*Arbetsgång vid **`unpivot()`***

1. Vid behov d v s om det finns fler kolumner i utgångstagbellen än vad som ska tas med i resultatet; Skapa en temporärtabell med CTE med
   - Den eller de kolumner som ska vara förspalt (förspaltskolumn)
   - De kolumner som ska återpivoteras.
2. Återpivotera med **`unpivot()`**

Tabell B

Förspalt	A	B	C
XXX	1000	500	1500
YYY	2000	750	1200

Tabell A

Förspalt	Rubrik	Värde
XXX	A	1000
XXX	B	500
XXX	C	1500
YYY	A	2000
YYY	B	750
YYY	C	1200

7.5. **SP-kommandon

****Exempel**

```
**Exempel 1
```

Visa info om aktiva processer på databas-servern, t ex vilka som är inloggade samt vilken databas som de befinner sig på.

```
sp_who

sp_who scbtole

sp_SCBVem

sp_SCBVem scbtole
```

```
**Exempel 2
```

Titta på definitionen för en fast tabell samt #-tabell.

```
use UTB_Kurs1
sp_help 'dbo.Kommun'
go

use tempdb
select Tal=1.0/7 into #A
select * from #A
go
sp_help '#A'
```

```
**Exempel 3
```

Skicka e-mail från SQL-skript eller lagrad procedur. Detta kommando är inte aktiverat på alla servrar, utan måste först aktiveras av en person som har SA-behörighet. Avsändare blir den server som man kör kommandot på. Man kan inte besvara detta e-mail som default, eftersom en profil först måste läggas upp av SA.

```
exec msdb.dbo.sp_send_dbmail
    -- Mottagare
    @recipients = 'tomas.lennefors@scb.se'
    -- Ämnesraden i mailprogrammet
  , @subject = 'Test'
    -- Brödtexten
  , @body = 'Detta är ett test'
```

7.6. **SQLCMD

****Exempel**

```
**Exempel 1
```

Special-läge i SQL Server Management Studio där man bland annat kan skriva till en SQL-fil och sedan läsa tillbaka den för exekvering. Se till att knappen SQLCMD mode är aktiverad i menyn Query.

```
-- ":!!" kör dos-kommandon
-- Visa aktuell katalog
:!!cd

-- Visa filer i aktuell katalog
:!!dir

-- Visa enbart filnamnen sorterade i aktuell katalog
:!!dir /B /N

-- Spara filnamnen i en textfil som alltid skriver
-- över eventuell existerande fil
:!!dir /B /N > H:\Test\Test.sql

-- Visa innehållet i en textfil
:!!type H:\Test\Test.txt
```

```
**Exempel 2
```

Skapa SQL-skript som fil samt läs tillbaka filen i SQL Server Management Studio och kör koden automatiskt.

```
-- Stäng av visning av antal rader i resultatet för
att få en korrekt SQL-sats
set nocount on
-- Skapa en SQL-fil innehållande en SQL-sats
:out H:\Test\Test.sql
select 'select * from sys.sysobjects'

-- Titta på SQL-filen
:!!type H:\Test\Test.sql

-- Kör SQL-koden och visa resultatet
:r H:\Test\Test.sql
```

**Exempel 3.
Som förra exemplet fast resultatet visas i extfönstret istället.

```
set nocount on
-- Skapa en SQL-fil
:out H:\Test\Test.sql
select 'select * from sys.sysobjects'

-- Kör SQL-koden och visa resultatet i text-fönstret
:out stdout
:r H:\Test\Test.sql
```

**Exempel 4.
Skapa ett backup-skript för alla tabeller i aktuell databas. Avsluta med att ta bort alla tabeller som innehåller '_BAK_' i tabellnamnet.

```
set nocount on
-- Skapa en SQL-fil
:out H:\Test\Backup.sql
select 'select *  into ' + name + '_BAK_' +
  convert(char(8), getdate(), 112) + ' from '+ name
from sys.sysobjects
where type = 'U'

-- Titta på SQL-filen
:!!type H:\Test\Backup.sql

-- Kör SQL-koden
:r H:\Test\Backup.sql

-- Kolla vilka tabeller som finns i databasen
select Name
from sys.sysobjects
where type = 'U'
order by name

-- Skapa SQL-kod som tar bort BAK-tabeller
:out H:\Test\Drop.sql
select 'drop table ' + name
from sys.sysobjects
where type = 'U' and name like '%[_]BAK[_]%'

-- Titta på SQL-filen
:!!type H:\Test\Drop.sql

-- Kör SQL-koden
:r H:\Test\Drop.sql

-- Kolla vilka tabeller som finns i databasen
select Name
from sys.sysobjects
where type = 'U'
order by name
```

8. *SQL-Skript

***Exempel**

*Exempel 1

Visar exempel på ett SQL-skript, d vs SQL-satser i flera steg.

```
----- Droppar gamla tabeller om de finns
if object_id('tempdb..#T1', 'U') is not null
  drop table #T1
if object_id('tempdb..#T2', 'U') is not null
  drop table #T2
if object_id('UTB_ElevO01.dbo.LanKommun', 'U')
is not null
  drop table UTB_ElevO01.dbo.LanKommun
go

----- Väljer ut 4 kolumner och döper om dem
select
  LanKod = Lan
  , KommunKod = Kommun
  , KommunNamn = Namn
  , KommunGrupp = KommunGrp
into #T1
from UTB_Kurs1.DBO.Kommun

----- Gör urval på kolumnen KommunGrupp
select
  LanKod
  , KommunKod
  , KommunNamn
  , KommunGrupp
into #T2
from #T1
where KommunGrupp = '2'

----- Joinar mot tabellen Lan
----- och sparar resultatet i en fast tabell
select
  LanKod
  , LanNamn = B.Namn
  , KommunKod
  , KommunNamn
  , KommunGrupp
into UTB_ElevO01.dbo.LanKommun
from #T2 A
  join UTB_Kurs1.DBO.Lan B
    on A.LanKod = B.Lan

----- Visar resultatet
select *
from UTB_ElevO01.dbo.LanKommun
```

*Exempel 2
Här följer ett exempel på hur man kan dokumentera sin kod med ett kommentarshuvud i början av SQL-skriptet.

```
/*
================================================
SQL-skript: Test.sql
================================================
Skapad av: Tomas Lennefors (TL)
Datum: 2001-01-01

Ändringar
---------
2001-02-17 / TL
- Lade till fler kolumner
2001-03-31 / TL
- Korrigerade ett fel i koden för att ...

Kommentarer
-----------
- Man måste ha DBO-behörighet för att kunna köra
  skriptet
- Skriptet ska köras i augusti varje år
- Skriptet ska köras på server ...
================================================
*/

-- Steg-1
```

***Beskrivning av SQL-skript**

Ett SQL-skript består av en mängd SQL-satser som antingen kan utföras ett i taget eller alla på en gång.

Ofta gör man bearbetningar i flera steg och sparar delresultaten som #-tabeller. Sista steget kan vara att spara resultatet i en vanlig permanent tabell. Kommentarer ska alltid finnas i skriptet som dokumentation. Överst ska det alltid finnas ett kommentarhuvud.

Man sparar det i en textfil med suffixet *.sql*. Vid behov öppnar man skriptfilen i SQL Server Management Studio och kör den där.

Resultatet kan sedan flyttas över till MS Excel eller annat lämpligt program för presentation.