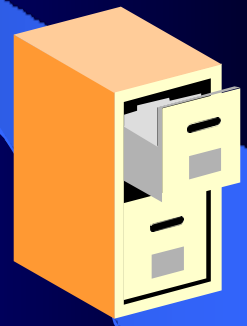
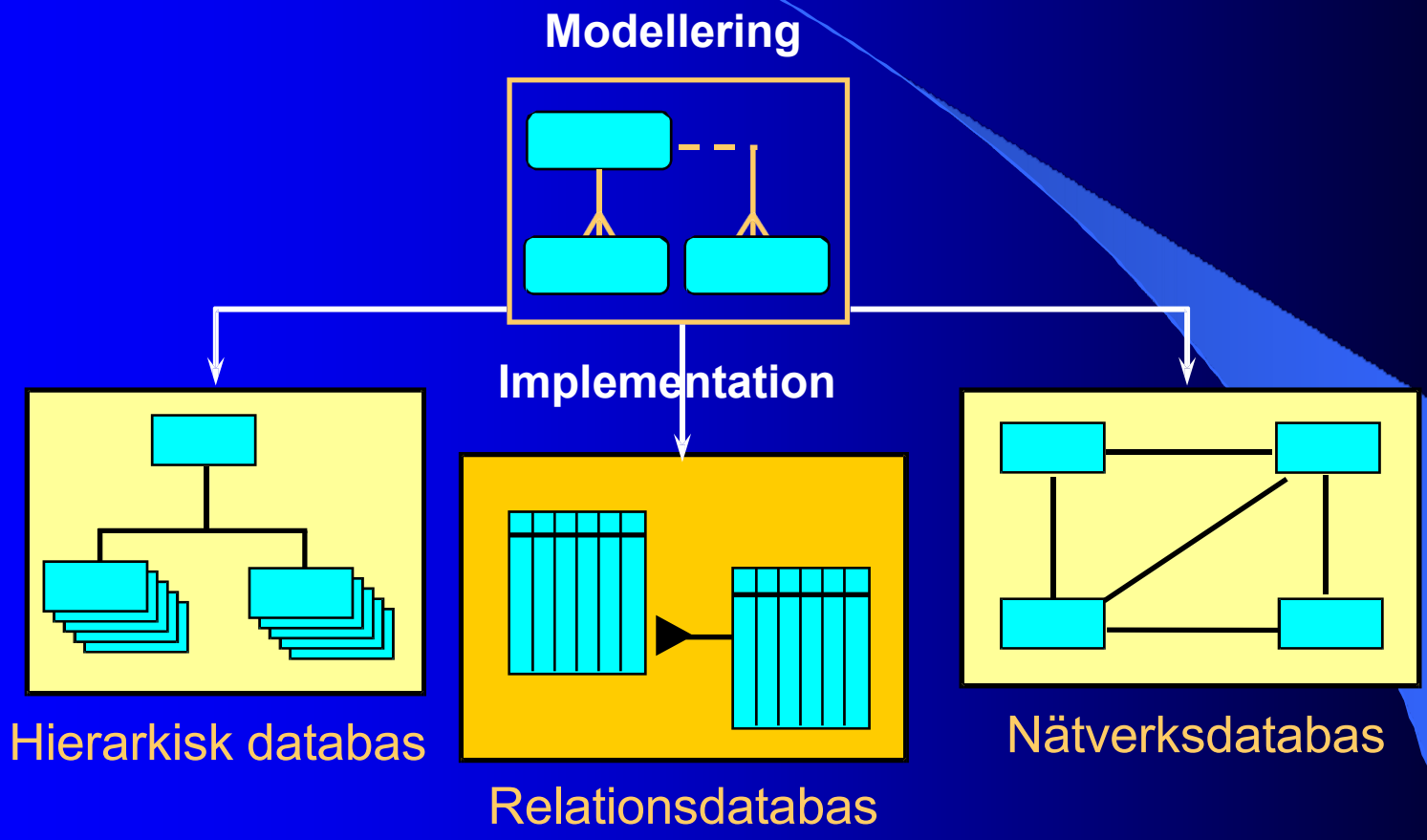
Vad är en databas ?

- Exempel på databaser:
  - *Kortregister på kontor*
  - *Sjukvårdsjournal*
  - *Bokregister på bibliotek*
  - *Medlemsregister i en förening*
  - *Kundregister på företag*
  - *Eniro (Telefonkataloger)*
- Databas = Organiserad samling och *lagring* av information.



Olika databasmodeller



Hierarkisk databas: Samling av filer och mappar i trädstruktur.

Nätverksdatabas: Samling av filer och mappar i någon annan topologi.

Relationsdatabaser

1970 Dr. E.F.Codd: *”A Relational Model of Data for Large Shared Data Banks.”*

- En modell för organisation av stora mängder av data som är relaterade till varandra.
- Modellens byggsten (modul): *Tabell*
- Databas = samling av *tabeller*.
  Tabell = *relation* mellan mängder av data (kolumner).
- *Modularisering:* Information om *en* sak ska lagras endast i *en* tabell.
  Leder till primär- och främmande nycklar samt relationer.
- *Relationsdatabas* = samling av *relationer*.

Tabell: rader & kolumner

Rad (post)

- Innehåller all data till *ett* exemplar av tabelltyp, t.ex. all information om *en* anställd i tabellen *Anställda*.
- Kan identifieras med ett unikt värde resp. en unik värdekombination (primärnyckeln, se bild 9).
- Ordningen i tabellen är inte definierad, obestämd.

Kolumn (fält)

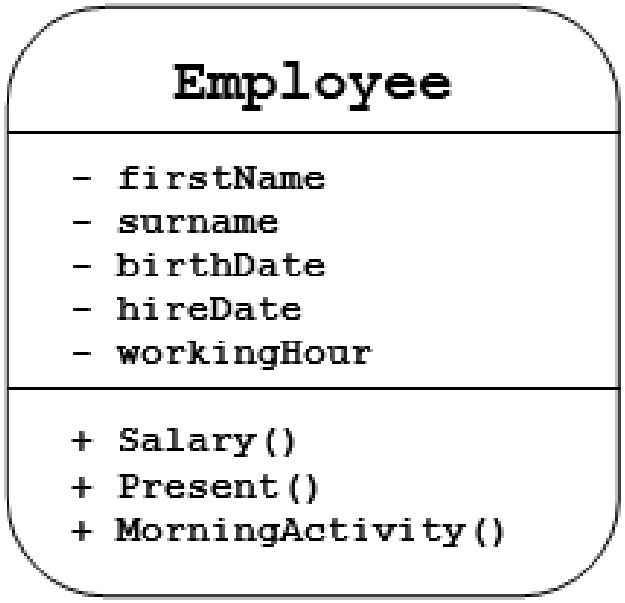
- Innehåller en *typ* av information om varje rad i tabellen.
- Måste ha ett namn = kolumnhuvudet = kolumnrubriken
- Måste ha en datatyp. Kan ha NULL i vissa poster.
- Har en position i tabellen: Ordningen är definierad.

Liknelse mellan klass och tabell

- *Tabell*
  - En tom tabell med fördefinierade kolumnrubriker kan jämföras med en *klass* där kolumnerna (bortsett från primär- och främmande nycklar) är dess datamedlemmar (egenskaper, attribut).
- *Rad*
  - Varje rad som läggs i tabellen kan jämföras med ett *objekt* av denna klass (tabellen). Varje data i en rad är ett värde till en objektmedlem.
  - Finns en primärnyckel används den för att identifiera raden på entydigt sätt (objektets namn).

Klassen och tabellen Employee

Låt oss titta på klassdiagrammet till höger. Om vi bortser från metoderna (markerade med +) och koncentrerar oss på datamedlemmarna (markerade med –) kan vi jämföra klassen `Employee` med en tom tabell vars kolumner är klassens datamedlemmar, se nedan. Tabellens struktur är identisk med klassens struktur när det gäller datamedlemmarna, vilket ger oss en ledtråd om hur vi ska bygga våra tabeller. Klassens metoder kommer att bli funktionaliteter som sedan måste läggas till med kod.



Förnamn	Efternamn	Födelsedatum	Anställn.datum	Arbetstid

Vad är en relation ?

I ett hyreshus bor Ola, Eva och Jimmy i lägenhet 1,
Alexander och Helen i lägenhet 2,
David och Diana i lägenhet 3.

Låt *Person* vara mängden av alla personer som bor i hyreshuset:

Person = { Ola, Eva, Jimmy, Alexander, Helen, David, Diana }

Låt *Lägenhet* vara mängden av alla lägenheter i hyreshuset:

Lägenhet = { 1, 2, 3 }

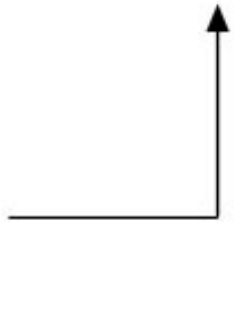
Tabell

”en person tilldelas sin lägenhet”

Person	Lägenhet
Ola	1
Eva	1
Jimmy	1
Alexander	2
Helen	2
David	3
Diana	3

Sambandet ”en *person* tilldelas sin *lägenhet*” är en ***relation***

mellan dessa två mängder och kan beskrivas bl.a. i en ***tabell***



Cartesisk produkt

Den cartesiska produkten av två mängder A och B:

A X B **(A ”kryss” B)**

är mängden av samtliga ordnade par (x, y) där x tillhör A och y tillhör B.

Exempel: ***Person*** **=** { Ola, Eva, Jimmy, Alexander, Helen, David, Diana }

Lägenhet **=** { 1, 2, 3 }

Den cartesiska produkten av dessa två mängder består av mängden:

Person* X *Lägenhet **=** {

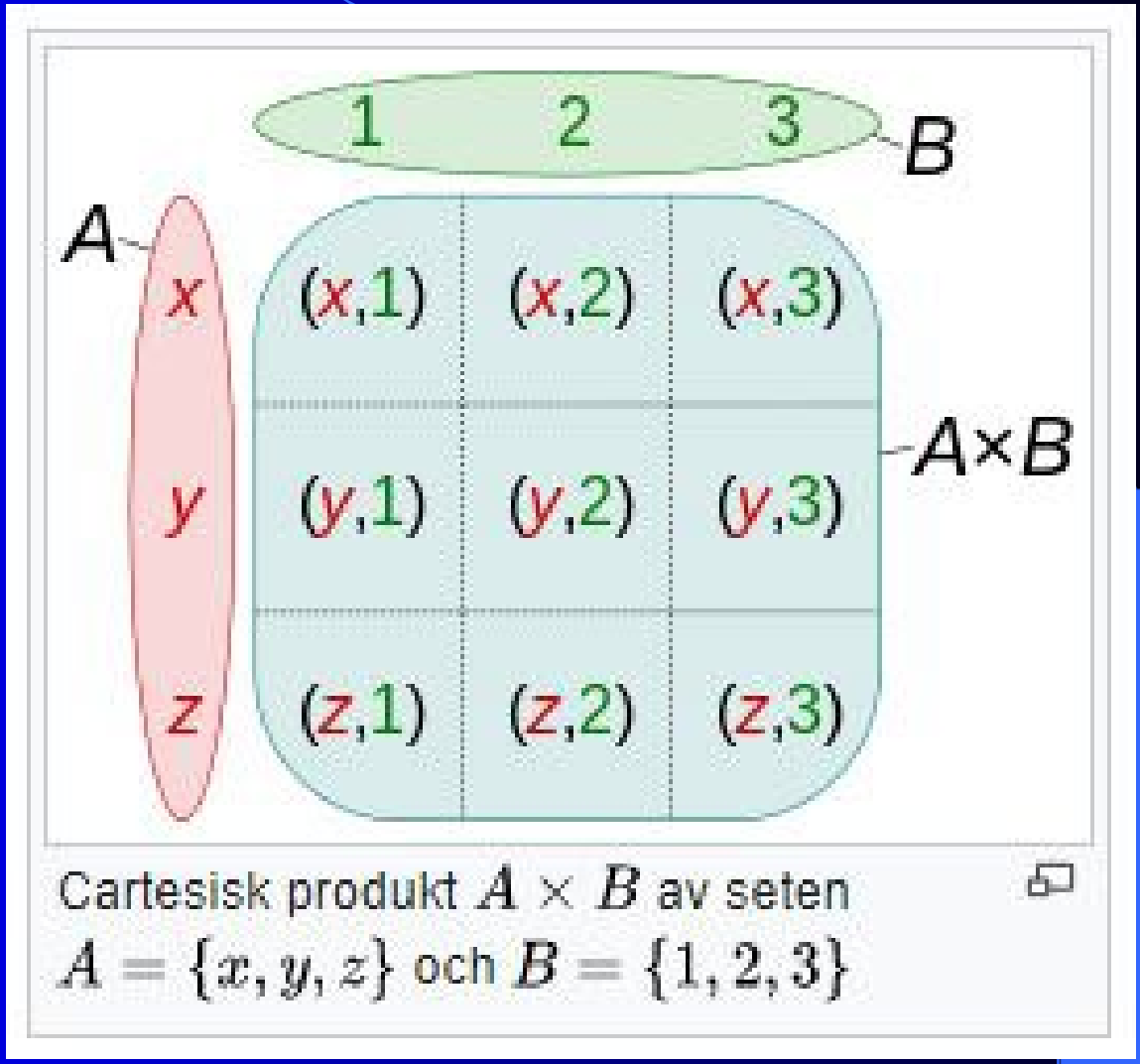
(Ola, 1),	(Ola, 2),	(Ola, 3),
(Eva, 1),	(Eva, 2),	(Eva, 3),
(Jimmy, 1),	(Jimmy, 2),	(Jimmy, 3),
(Alexander, 1),	(Alexander, 2),	(Alexander, 3),
(Helen, 1),	(Helen, 2),	(Helen, 3),
(David, 1),	(David, 2),	(David, 3),
(Diana, 1),	(Diana, 2),	(Diana, 3) }

Cartesisk produkt = Kombination av alla med alla = Alla möjliga par.

Varför ”Cartesisk” produkt?

René Descartes

2D Cartesiskt koordinatsystem med B som x- och A som y-axeln.



Relation mer exakt

Ex.: Relationen ”en person tilldelas sin lägenhet” – låt oss kalla den R – är definierad så här:

” I ett hyreshus bor Ola, Eva och Jimmy i lägenhet 1,
Alexander och Helen i lägenhet 2,
David och Diana i lägenhet 3 ”

Denna relation kan beskrivas i en tabell →

Tabell R:
”en person tilldelas sin lägenhet”

Person	Lägenhet
Ola	1
Eva	1
Jimmy	1
Alexander	2
Helen	2
David	3
Diana	3

Relationen R är en delmängd av den cartesiska produkten *Person* x *Lägenhet* :

R = { (Ola, 1), (Eva, 1), (Jimmy, 1),
(Alexander, 2), (Helen, 2),
(David, 3), (Diana, 3) }

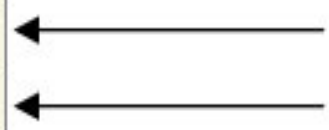
Relation = delmängd av den cartesiska produkten.
Tabell = relation mellan tabellens kolumner. Raderna *beskriver* relationen.

Varför är 2 tabeller bättre än 1?

Exempel:

Information om personers jobb och avdelningars plats där de jobbar lagras i 1 tabell:

ID	Namn	Jobb	Avdelning	Plats
1	Ola	IT-proffs	IT	Kista
2	Eva	Programmerare	IT	Kista
3	Jimmy	Revisor	Ekonomi	Stockholm
4	Alexander	Säljare	Marknad	Göteborg
5	Helen	VD	Marknad	Göteborg
6	David	Chaufför	Logistik	Älvsjö
7	Diana	Grafiker	PR	Liljeholmen



Uppdatering: **IT-avdelningen flyttas till Stockholm.**

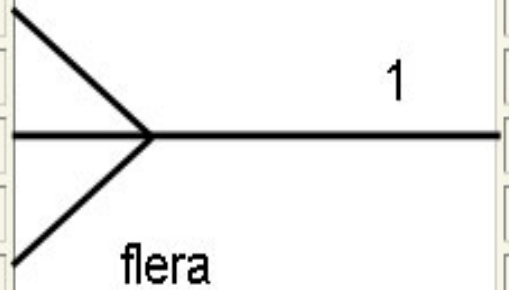
1 tabell: **Ändringen måste göras på flera ställen.** **Ingen Modularisering!**

2 tabeller: **Ändringen måste göras på ett ställe.** **Modularisering!**

Modularisering leder till relation

Tabellen Anställda

ID	Namn	Job	Avdelning
1	Ola	IT-proffs	10
2	Eva	Programmerare	10
3	Jimmy	Revisor	20
4	Alexander	Säljare	30
5	Helen	VD	30
6	David	Chaufför	40
7	Diana	Grafiker	50



Tabellen Avdelningar

Nr	Namn	Plats
10	IT	Kista
20	Ekonomi	Stockholm
30	Marknad	Göteborg
40	Logistik	Älvsjö
50	PR	Liljeholmen

- **Varje anställd arbetar på endast en avdelning.**
- **Varje avdelning är arbetsplats för en eller flera anställda.**

- **Regeln: Information om EN sak (anställda) ska lagras i endast EN tabell (modularisering).**
- **Information om avdelningar (en annan sak) ska separeras och lagras i en annan tabell.**

Primär- och främmande nycklar

- En eller flera kolumner i en tabell kan definieras till tabellens *primärnyckel (PK)*.
- En tabell kan ha *endast en* primärnyckel, ev. av flera kolumner: *sammansatt PK*.
- Varje rad identifieras unikt via primärnyckeln. Därför får inga dubletter förekomma.
- En annan tabells (eller den egna tabellens) primärnyckel i en tabell kallas *främmande nyckel (FK)*: Flera möjligt!

Tabellen `EMPLOYEES`

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
174	Ellen	Abel	80
142	Curtis	Davies	50
102	Lex	De Haan	90
104	Bruce	Ernst	60
202	Pat	Fay	20
206	William	Gietz	110

...

Primärnyckeln **Främmande nyckel**

Tabellen `DEPARTMENTS`

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

Primärnyckeln

Databashanterare

Programvara för att

- skapa,
- utforma (designa),
- lagra och
- administrera databaser,

Kallas *Database management system (DBMS eller RDBMS)* som i regel installeras på en server.

- Exempel på *databashanterare* är *Access, (Excel), SQL-Server, Oracle 21c, MySQL, DB2, FoxPro, Paradox, …*

- Alla DBMS har stöd för *SQL*, men även för procedurala utökningar av SQL:

  Oracle *PL/SQL*,

  Micorsoft *Transact SQL, …*

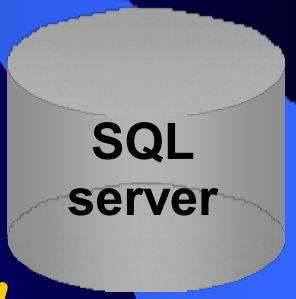
Klient - Server modellen

SQL ställer en fråga, t.ex.:

```
SELECT department_name
FROM   departments;
```

Frågan skickas till servern

dvs ber om en tjänst *(klient)*, beskriver vad den vill ha, men inte hur svaret kommer till.



Servern svarar

DEPARTMENT_NAME
Administration
Marketing
Shipping
IT
Sales
Executive
Accounting
Contracting

Operationens slutresultat är alltid en tabell: *”Resultattabell”*

SQL – databasers språk

Structured Query Language

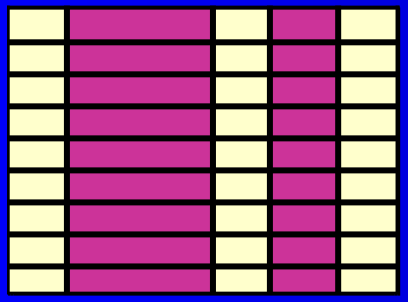
(Strukturerat frågespråk)

- Standardspråk för kommunikation med relationsdatabaser.
- Oberoende av databashanterare.
- Utvecklades på 70-talet av IBM.
  Idag: allmän standard, senaste version: SQL-99
- Med SQL kan man ställa *”frågor”* till databaser för att
  - ta fram, uppdatera,
  - sortera och
  - strukturera information i databaser,
  - skapa tabeller, definiera constraints
  - ge rättigheter till databasobjekt, …

SELECT-*satsen*

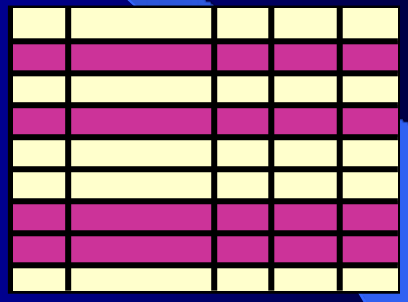
Läser från databasen och visar data i kolumner och rader:

1. Projektion



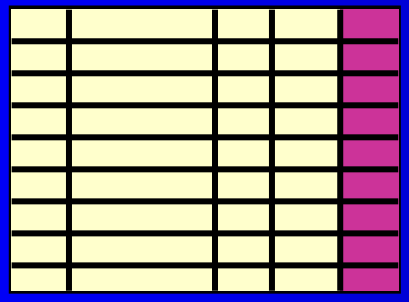
Selekterar kolumner från 1 tabell

2. Selektion

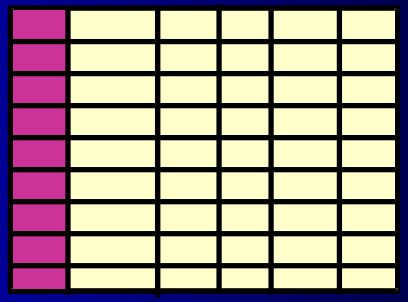


Selekterar rader från 1 tabell

Tabell 1



3. Join
(Kap. 3)



Tabell 2

Tar ut data från flera tabeller

Att ta ut alla kolumner

```
SELECT *
FROM   departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

...

SELECT talar om vilka kolumner som ska tas ut.
FROM talar om från vilken tabell kolumnerna ska tas ut.

Ger samma resultat:

```
SELECT department_id, department_name, manager_id, location_id
FROM   departments;
```

Att ta ut vissa kolumner

```
SELECT department_id, location_id
FROM   departments;
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

...

Kommaseparerad lista över kolumner som anger ordningen endast i den aktuella *utskriften*, inte i databasen.

SELECT-satsen är read-only, kan inte ändra databasen.

Hittills har vi endast använt projektion: Att selektera kolumner.

Att selektera rader: selektion

Tabellen `EMPLOYEES`

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
104	Ernst	IT_PROG	60
107	Lorentz	IT_PROG	60
124	Mourgos	ST_MAN	50

...

Vilka anställda jobbar på avd. 90?

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

För att få ut det måste ett villkor läggas till **`SELECT`**-satsen.

Detta görs med den nya satsdelen (eng. *clause*) **`WHERE`**.

Att lägga till villkor med `WHERE`

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Enkelt *villkor* på likhet mellan tal.
= är här en jämförelseoperator.

OBS! **Villkoret kan involvera en kolumn som inte ens förekommer i `SELECT`-satsen:**

```
SELECT employee_id, last_name, job_id
FROM   employees
WHERE  department_id = 90 ;
```

Radsortering med ORDER BY

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91

...

Satsdelen **ORDER BY** kommer sist i **SELECT**-satsen.
Utan **ORDER BY** är radordningen odefinierad.

ORDER BY sorterar by default i stigande ordning dvs ASC (ASCending).
För fallande ordning kan DESC (DESCending) användas:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date DESC;
```

CREATE TABLE-*satsen*

```
CREATE TABLE Kurser
(
   KursID    INT          IDENTITY (1,1) NOT NULL,
   Namn      VARCHAR(50)                 NOT NULL,
   Längd     INT                             NULL,
   InstID    INT                         NOT NULL
);
```

- Måste specificeras:
  - Tabellnamn: **`Kurser`**
  - Kolumnnamn: **`KursID, Namn, Längd, InstID`**
  - Kolumndatatyper: **`INT, VARCHAR(50)`**

  (OBS! Inte optional)

Regler & konventioner

Några regler för SQL-satser:

- SQL-satser är inte case sensitive.
- Det är inte obligatoriskt att avsluta SQL-satser med semikolon.
- SQL-satser kan skrivas på en eller flera rader.
- Reserverade ord kan ej förkortas.

Konventioner:

- Skriv reserverade ord med versaler.
- Börja reserverade ord på separat rad.
- Avsluta *SQL*-satserna med semikolon.