Databaser

- **Varför databas?**
    - Data måste lagras någonstans för att inte försvinna när en applikation eller hemsida stängs.
    - Man kan lagra i bara en fil så varför då en databas?
    - Data måste ofta lagras strukturerat, så att vi vet vad datan betyder.
    - 10 är data men 10 vad?
        - Skolgatan 10 är en adress.
        - 10 kr kan vara ett pris på en vara.

Lite begrepp

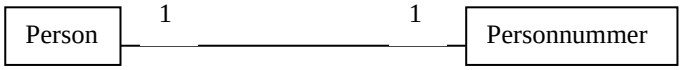
- **Databashanterare** – En databashanterare är ett program som hjälper till att hantera en databas. Exempel på relations-databashanterare är MySQL, SQL-Server och Oracle.
- **Relationsdatabas** – Det finns olika sorters databaser men den sort vi ska lära är relationsdatabas. En relationsdatabas består av tabeller och relationer mellan dessa. Vissa säger dock att själva tabellerna är relationerna.
    - Andra databaser är objekt-databaser. När man sparar objekt från ett programspråk i en relationsdatabas måste man för omvandla objekten. Detta slipper man med objektdatabaser.

Relationer

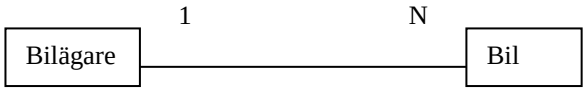
En relations databas kan ha tre sorters relationer (egentligen bara två).

- En till En
- En till många
- Många till många

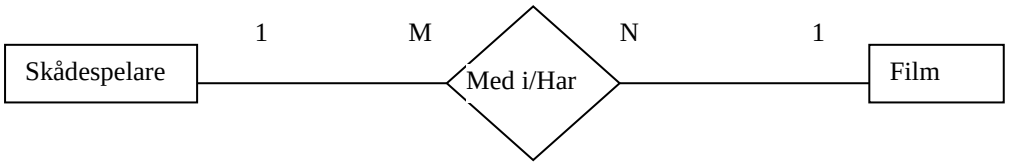
Exempel på **en till en** är personnummer. Ett personnummer tillhör **en** person och varje person har bara **ett** personnummer.



Exempel på **En till många** är en bilägare och dennes bilar. **En** bilägare kan ha **många** bilar men varje bil tillhör bara en bilägare.



Exempel på många till många. Detta är ett specialfall. Egentligen kan en relationsdatabas bara ha en – en och en – många. För att skapa många till många måste man ha en s.k. kopplingstabell.



En skådespelare är med i många filmer och en film har många skådespelare.

S_ID	Namn
1	Kalle
2	Lisa
3	Anna
4	Nils

S_ID	F_ID
1	2
3	4
1	3
4	3

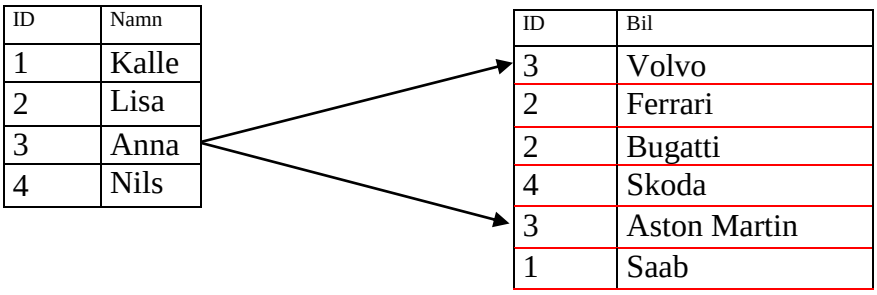
F_ID	Film
1	Borta med vinden
2	Strul
3	The Truman show
4	Macken

Nycklar och unika värden

- **Primärnyckel (primary key) –** Unikt värde som identifierar en post, t.ex. ett personnr.
- **Främmande nyckel (foreign key) –** Används för att koppla tabeller. (*Se papper med om relationsdatabaser*)
- **Sammansatt nyckel** (**Compound key**) – Nyckel som man får av två fält. Ett telefonnummer är inte unikt men tillsammans med riktnummer blir det unikt.

Primärnyckel (Primary key)

Främmande nyckel (Secondary key)



ID	Namn
1	Kalle
2	Lisa
3	Anna
4	Nils

ID	Bil
3	Volvo
2	Ferrari
2	Bugatti
4	Skoda
3	Aston Martin
1	Saab

Några exempel:

Tabell: Personer

Hämta allt i tabell

SELECT *
FROM **Personer;**

ID	Fnamn	ENamn
10	Kalle	Nilsson
20	Olle	Karlsson
30	Lisa	Persson

Hämta bestämt värde i tabell

```
SELECT *
FROM Personer
WHERE ID=20;
```

ID	Fnamn	ENamn
10	Kalle	Nilsson
20	Olle	Karlsson
30	Lisa	Persson

Lägga in värden i tabell:

```
INSERT INTO Personer
VALUES (10, 'Kalle','Nilsson') ;
```

ID	Fnamn	ENamn
10	Kalle	Nilsson
20	Olle	Karlsson
30	Lisa	Persson

Uppdatera tabellen:

```
UPDATE Personer
SET Efternamn = 'Månsson'
WHERE student_id = 20;
```

ID	Fnamn	Enamn
10	Kalle	Nilsson
20	Olle	Månsson
30	Lisa	Persson

Ta bort post i tabellen;

```
DELETE FROM Personer
WHERE ID = 30;
```

ID	Fnamn	ENamn
10	Kalle	Nilsson
20	Olle	Månsson
30	Lisa	Persson

Datatyper

- **Vilken datatyp man använder bestämmer:**
  - Vilken sorts data man kan använda t.ex. heltal (int) eller text (varchar t.ex.)
  - Hur stor plats datatypen tar i minnet. Använder en typ som tar onödigt stor plats så slösar man på minne.
  - Använder man fel typ uppstår fel i databasen.
  - Det gäller att välja rätt sorts typ för nödvändig funktion och storlek.

Lite mer om tabeller se 2.2 (prov)

- **SQL skapades för att hantera relationsdatabaser.**
- **SQL syntax är vad som bestämmer reglerna för hur ett SQL-kommando ska se ut.**
- **När reglerna inte följs får du ett felmeddelande.**

Skapa tabell:

```
CREATE TABLE Studenter
(first_name char(20) not null,
last_name char(20) null,
student_id int not null,
grade int not null);
```

Här skapar vi en tabell som heter **Studenter** och lägger in en **sträng på max 20 tecken** för **förnamn** och **efternamn**, **student_id** är ett **heltal** och **betyg** ett **heltal**.

Lägg märke till där det står **not null**. Det innebär att vi inte får utelämna det värdet. När man matar in i MySQL med insert into skriver man **null** om man inte vill ha ett värde på den platsen. Man kan inte bara utelämna det för då blir det fel antal kolumner.

Man måste inte ha med null i MySQL.

JOIN

Inner Join - Returnerar rader där det finns värden i **båda** tabellerna. I exemplet nedan returneras alla rader för ID 1 (Kalle) och ID 2 (Olle) eftersom de har bilar och därför finns i båda tabellerna. Raden med ID 3 (Erik) returneras inte eftersom han inte finns i tabellen bilar.

OBS! Inner Join är detsamma som att bara skriva **Join**.

```
SELECT Owner.Fnamn, Owner.Enamn, Cars.Bilmärke, Cars.Bilmodell
FROM Owner
INNER JOIN Cars
ON Owner.ID = Cars.ID
```

Man använder punkt som i **Owner.Fnamn** så att databashanteraren
Vet vilken **tabell** (Owner) som **kolumnen** (Fnamn) tillhör.

Denna fråga returnerar de värden som är markerade med gult. I **SELECT** väljer vi ut de kolumner som ska visas. Eftersom vi inte har valt kolmnerna för **Owner.ID**, **Cars.ID** och **Cars.RegNr** (de i blått) kommer de **inte** med, men skulle kunnat vara med.

I resten av frågan väljer vi ut vilka rader vi vill visa.

Raden med **Erik** i **Owner** kommer **inte** alls med, eftersom han **inte** finns i båda tabellerna.

Owner

ID	Fnam	Enamn
1	Kalle	Ek
2	Olle	Karlsson
3	Erik	Nilsson

Cars

ID	RegNr	Bilmärke	Bilmodell
1	ABC 123	Mercedes	SLK
1	DXS 271	Ferrari	F40
2	GEW 377	Pontiac	Trans Am
1	EAS 174	Aston Martin	DBS

LEFT JOIN

Left Join – Returnerar alla rader i **vänstra tabellen** även om det **inte** finns motsvarande värde i **högra tabellen**.

```
SELECT Owner.Fnamn, Owner.Enamn, Cars.Bilmärke, Cars.Bilmodell
FROM Owner
LEFT JOIN Cars
ON Owner.ID = Cars.ID
```

Nu är även ID 3 (Erik) med eftersom han finns i vänstra tabellen

Owner

ID	Fnam	Enamn
1	Kalle	Ek
2	Olle	Karlsson
3	Erik	Nilsson

Cars

ID	RegNr	Bilmärke	Bilmodell
1	ABC 123	Mercedes	SLK
1	DXS 271	Ferrari	F40
2	GEW 377	Pontiac	Trans Am
1	EAS 174	Aston Martin	DBS

RIGHT JOIN

Right Join – Returnerar alla rader i **höger tabell** även om det **inte** finns motsvarande rader i **vänstra tabellen**.

```
SELECT Owner.Fnamn, Owner.Enamn, Cars.Bilmärke, Cars.Bilmodell
FROM Owner
RIGHT JOIN Cars
ON Owner.ID = Cars.ID
```

Hade vi nu haft kvar det som innan hade det inte spelat något roll eftersom vi redan returnerar allt från **höger tabell**. Så här har jag lagt till en bil som inte har någon ägare. Vi har lagt till **ID 8** men eftersom det inte finns någon ägare med **ID 8** så finns denne inte med i vänstra tabellen.

Nu kommer även Volvo med i resultatet eftersom vi får med allt i **högra tabellen** men inte Erik i **vänstra**.

Kanske har ägaren slutat i bilklubben men man har glömt att ta bort hans bil.

Owner

ID	Fnam	Enamn
1	Kalle	Ek
2	Olle	Karlsson
3	Erik	Nilsson

Cars

ID	RegNr	Bilmärke	Bilmodell
1	ABC 123	Mercedes	SLK
1	DXS 271	Ferrari	F40
2	GEW 377	Pontiac	Trans Am
1	EAS 174	Aston Martin	DBS
8	ABE 777	Volvo	V70

OBS! I vissa databaser kallas **RIGHT JOIN** för **RIGHT OUTER J**OIN.

FULL JOIN

Full Join – Returnerar allt i båda tabellerna.

TRANSACTIONS

- TRANSAKTIONS sätter samman flera uppgifter till en enhet.
- Om någon av uppgifterna misslyckas så genomförs ingen av uppgifterna. Därmed hindrar man skada på databasen.
- Man kan tänka på transaktioner som att kompilera flera programmerings-rader tillsammans istället för rad för rad.
- En transaktion börjar med att köra en SQL-fråga (**`UPDATE`**/**`INSERT`**/**`DELETE`**)
- Alla rader tills man når **COMMIT** eller **ROLLBACK** ingår i transaktionen. När man kör COMMIT eller ROLLBACK har transaktionen avslutats.
- **COMMIT** – Om alla rader är rätt i en transaktion sparas allt till databasen.
- **ROLLBACK** – Alla ändringar tas tillbaka och databasen förändras inte.

Normalisation

- Används för att förhindra att information finns på flera ställen i en databas.
- Normalisation gör det enklare att köra frågor och uppdatera databasen. Det gör det också enklare att upprätthålla säkerhet och integritet. Integritet kan innebära att om en ägare tas bort ur databasen så tas hans bilar också bort.

  Nackdelen är att det blir fler tabeller, så man ska inte normalisera för mycket.
- OBS! Att ha t.ex. student_ID i flera tabeller är inte onödig data. Det behövs för att kunna göra kopplingar (related dependency).

Varför ska man använda normalisation?

- Det reducerar (minskar) lagringsutrymmet som behövs för en databas och ser till att data är logiskt lagrat.

Normalform

- Ett sätt att organisera (strukturera) information så att man undviker onödiga data och inkonsistenta data (data som inte hänger ihop ungefär). Man använder det också för att göra databasen lättare att underhålla, lagra i och uppdatera.
- Det finns flera nivåer av normalisation:
  - De tre vanligaste är:
    - Första normalformen (1NF)
    - Andra normalformen (2NF)
    - Tredje normalformen (3NF)

1NF – Första Normalformen

- Ska bestå av grupper av poster (som tabellen bilar t.ex.)
- Varje post ska vara unik (som den blir genom primärnyckel)

- Varje kolumn får bara innehålla ett värde. T.ex. får man **inte** skriva två bilar i en kolumn t.ex. Ferrari, Jaguar.
- Man får inte ha flera kolumner av samma sort. T.ex. en kolumn för märket Volvo och en för märket SAAB.

Grundläggande regler för en databas:

- Ta bort upprepade kolumner (som i exemplet ovan med Volvo och SAAB)
- Skapa separata tabeller för varje grupp av data som hör ihop. (T.ex. en tabell för personer och en tabell för bilar).
- Identifiera varje rad med en unik kolumn (Primärnyckel).

2NF – Andra Normalformen

- Är hårdare än 1NF när det gäller att ta bort upprepade data.
- Måste vara i 1NF.
- Får inte finnas upprepade värden i onödan.

3NF – Tredje Normalformen

- Måste vara i 2NF.
- Tar bort kolumner som inte är beroende av primärnyckeln.
- En kolumn som inte är absolut nödvändig tas bort. T.ex. kan man ta bort kolumnen för skatt för den kan man räkna ut.

Index

Vad är ett index?

- Index är en lista på något:
  - Ett index för en telefonkatalog gör att det är lättare att hitta rätt namn.
  - Man använder ett index för att gå direkt till det ställe vi vill i en telefonkatalog (eller bok) och samma gäller för en databas.

- Ett index är en datastruktur.
  - En lista med nyckelord och data som hör till nyckelorden som pekar mot det ställe där det finns mer utförligt information.
  - Kan vara filer eller poster på en hårddisk t.ex. eller nycklar i en databas.

- Index gör det snabbare att söka.
- En nackdel är att det tar mer tid att skriva index och det tar större plats, precis som extra sidor behövs för ett register i en bok.
  - Utan index skulle det inte vara möjligt att söka på Internet.