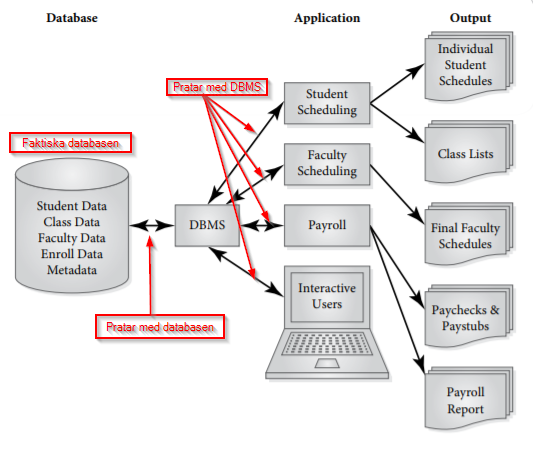
Individuell inlämning 1

**Kort historisk aspekt av databaslagring**

Idén att lagra och hantera information har funnits i tusentals år, men det var först i slutet på 1800-talet som starten kom till vad som kallas databas idag. Då börjades det använda hålkort för att permanent lagra olika sorts data. Under tidsperioden tills dagens läge uppfanns nya lösningar på samma problem - hur data lagras effektiv och pålitligt. Lösnigar såsom magnetiska band och tillslut modernare hårddiskar.   
  
**Integrerad databas**  
I dess mening inom teknikens värld syftar termen *databas* på en samling data som är logiskt sammanhängande. Data ska kunna hämtas, placeras eller lagras på ett säkert sätt.   
Vanligtvis talar man dock om en *integrerad databas* och syftar då på en databas som är kopplad till ett system som bestämmer tillgången till databasen, hur data hämtas och skrivs (t.ex. ett företag har roller som hanterar databasen eller utvecklar program till användning av databasen).  
  
Den integrerade databasen löser många problem som tidigare fanns t.ex. *delning av data*, *redundant data*, *konsekvent data*, *datasäkerhet* m.flr. Detta uppnås bland annat med ett extra lager som agerar som en proxy till databasen som heter *Database Management System (DBMS)* och all åtkomst och *queries* (förfrågningar, exempel på SQL-queries: SELECT FROM, SELECT WHERE osv.) hanteras av DBMS som är den enda entiteten i systemet som agerar direkt på databasen. DBMS möjliggör därför bl.a. *concurrency*, *backup/återhämtning av data*, *autentisering och auktorisering* till databasen.

  
Bild 1 från *Database Illuminated* sid.8

**Big data**

Inom kontexten databas syftar *Big data* på att samla in, hantera och massiv mängd data som kommit in från flera olika källor (övervakningskameror, datorer, mobiltelefoner genererar alla en ström av data som kan lagras). I praktiken innebär detta att t.ex. ett företag eller organisation samlar in enorma mängder data i hopp om att kunna använda information till en mängd olika ändamål med idén om att kvantitet av data inte ska vara en flaskhals. Big data syftar även på de olika teknologierna att hantera all insamlad data, “hur får man ut relevant data?”. *Data mining* är en något som applicerar *big data*.

## Modellera databassystem

**Entity Relationship modellen och Entity Relationship diagram**

När systemutvecklare ska designa och konstruera ett databassystem behövs en sorts översikt för att designern ska kunna förmedla konceptuella egenskaper (*relationer och attribut*) hos olika delar/enheter (*entities*). Med hjälp av *Entity Relationship Model* kan designer visualisera databassystemet, sambandet mellan *entities*. Modellen är tänkt att kunna skapa mening och struktur som därefter kan implementeras i en databas.

Entity

Modellen byggs till grunden upp av ett objekt vid namn *Entity*. Detta är en sorts datatyp som består av *attribut* och kan vara i relation till andra *entities*, samt att den kan urskiljas från andra entities. En *entity* ska ritas som en rektangel i E-R-diagrammet.

En *entity-instans* syftar på en egen uppsättning av dess attribut. (detta kan jämföras med objektorienteringens klass och objekt).  
- *Attribut* syftar på datan som utgör en entity, t.ex. ålder, namn, kurskod, bilmodell, personnummer, och ritas som en oval i E-R-diagrammet.  
- *Relation* syftar på hur de samverkar med andra entities. T.ex. en lärare är i relation till en student genom att studenten ingår i samma klass som läraren lär ut, och studenten är i relation på andra hållet. (“Teaches” och “Is taught by”). En relation ritas som en romb i E-R-diagrammet.

Attribut

För varje attribut i en entity finns en sorts gräns på vilka värden den kan anta - detta kallas attributets *domän*. För ett heltal kan detta sättas i en sk. *range* där värden utanför inte tillåts. T.ex. en ålder på en entity som representerar en person ska inte kunna vara mindre än noll. Ett attributs domän kan ses som ett de set som innehåller subset enligt den kartesisk produkten.   
Ex. Låt carBrand = {“Volvo”, “Audi”, “Skoda”} och carColor = {“Blue”, “Red”, “Green”}

En databas som hanterar datatyper med dessa domäner kan endast innehålla alla subset av den *kartesiska produkten* carBrand X carColor, { {“Volvo”, “Blue”}, {“Volvo”, “Red”} …. {“Skoda”, “Green”} }

*Attribut* inom E-R-modellen kan finnas i olika former för att kunna modellera en effektiv och övergripligt.

* Om en entity kan ha flera uppsättningar av samma attribut ska detta kallas *Multivalued Attribute* och ritas med två linjer istället för en i ovalen.
* Om ett attribut kan delas upp i flera värden ska detta kallas för *Composite Attribute* och ritas som ett vanligt attribut, fast “under-attributen” ritas också in som vanliga attribut kopplade till *Composite attributet*.
* Om ett attribut inte behöver existera på egen hand utan beräknas eller grundas på ett annat attribut i samma entity ska denna kallas *Derived attribute* och ritas som en streckad oval. Ex. ett attribut age kan vara opraktiskt att lagra i en databas då de uppdateras hela tiden, utan istället kan beräknas via ett annat “vanligt” attribut dateOfBirth finns.

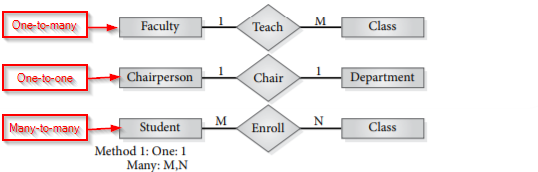
Nycklar

Översiktligt sett är *nycklar* som ett sak eller objekt som låter oss skilja på värden (såsom entities). Ofta används benämningen key-value som ett par där nyckeln används för att identifiera värdet. I databas kontexten är nycklas en uppsättning av attribut som identifierar entities.

* En *superkey* är en sorts nyckel som låter oss unikt identifiera en specifik entity. En sådan nyckel kan vara uppsatt av flera attribut.
* En *candidate key* är en nyckel som unikt identifierar en entity, men med så få attribut som möjligt. T.ex. en *superkey* identifierar en student med hjälp av stuID, firstName och lastName. Däremot kan varje student i detta fall identifieras med enbart stuID, så en kandidatnyckel skulle kunna vara enbart stuID (alltså en så lite *superkey* som möjligt).
* En *primary key* är en *candidate key* som valts ut av designern. T.ex. en student kan ha stuID och personalNumber, båda kan av sig själva vara kandidatnycklar, men designern vill att stuID ska användas för att identifiera en student, då blir stuId en *primärnyckel*.
* En *foreign key* är en *primärnyckel* av någon annan entity (eller samma vid rekursionssamband). T.ex. en student kan kopplas till att ha en parkeringsplats, då har studenten stuID som *primary key* och t.ex. *regNum* som en *foreign key* till sin bil (regNum måste då givetvis vara *primary key* till en bil-entity).

Samband (förhållande?)

Entities kan vara länkade med varandra i s.k. *samband*. T.ex. en student har ett *samband* med en lärare av typen “isTaughtBy”, och läraren har ett samband till studenten “teaches”. Samband ritas som en romb i E-R-diagram och binds med olika typer av linjer beroende på vilket samband det är.

* *Sambandsgrad* syftar på hur många entity set som länkas ihop. T.ex. en lärare och student i ovan nämnda samband “Teach” har två entities (lärare och student). En annan relation skulle kunna vara lärare-bok-kurs. Sambandsgraden hos ett samband ger den olika namn - 1:unary, 2:binary, 3:ternary osv.
* Ett sambands *kardinalitet* syftar på antalet entities som kan finnas i relation. T.ex. en lärare kan lära ut till flera studenter, men (som exempel) kan studenterna endast ha en lärare. I en databas skulle detta betyda att flera elever kan vara i samband (kopplade) till samma lärare, och en lärare kan vara i samband (kopplad) med flera olika studenter. *Kardinaliteten* är därmed av typen *one-to-many*. Det finns även *one-to-one* (en person gift med en annan person) och *many-to-many* (kurs och student)  
    
  Bild 2 från *Database Illuminated* sid. 72
* Ett sambands *deltagande* syftar på hur stor andel av alla instanser hos ett entity set som ska delta i ett samband. Detta kan förklaras bättre som “Om jag är en <entitytyp>, måste jag då delta i sambandet <samband>?”.   
  *Total participation* innebär att alla instanser av samtliga entity set i en relation *deltar* i relationen. T.ex. lärare och kurs, en lärare måste finnas i en kurs och lära ut den, annars är det inte en lärare.   
  *Partial participation* innebär att alla instanser av en entity inte måste delta i sambandet. T.ex. student och kurs, alla studenter måste inte finnas i en klass.
* Ett sambands *roller* syftar på varför en entity är i sambandet. Såsom student-lärare “isTaughtBy” eller lärare-student “teaches”. Olika entities kan alltså ha olika *roller* i samma samband.

Svaga entities

Det finns entities som endast har mening om de är i samband med andra entities - sambandet är som ett nödvändigt komplement. När en entity är beroende av ett samband (alltså även en annan entity) för att få mening kallas denna för *svag entity*. T.ex. vid avslut av en kurs lämnar studenter en bedömning som lagras i en databas. Denna bedömning måste kopplas till en kurs, annars saknar den helt mening. Bedömningen är därmed en *svag entity*. Svaga entities ritas med dubbla linjer.

**Extended Entity Relationship modellen**

Den vanliga E-R-modellen visade sig under senare tid inte vara tillräckligt deskriptiv för att modeller mer avancerade applikationer såsom sökmotorer, data mining m.fl. utan krävde ytterligare funktionalitet. *Extended E-R* gör detta genom bl.a. generalisering och, motparten, specialisering.

* *Specialisering* syftar på att skapa ytterligare entities eller attribut som är till för att lägga till funktionalitet (Likt som subklasser gör för superklasser i objektorienterad programmering). EE-R kan alltså modellera arv och därmed kan underGrad eller graduate vara subtyp till supertypen student (sambandet är kallas *isa-samband*).
* *Generalisering* syftar på att hitta gemensam funktionalitet hos entities. T.ex. att inse att både lärare och student är personer, och båda bör ha t.ex. firstName och lastName.

Union

EE-R lade inte endast till arv-samband, utan det går även att kategorisera entities med s.k. *unioner* (kallas även *kategorier*). Tanken är att en entity som är med i en *union* kan anta en och endast en av subtyperna inom unionen. T.ex. en sponsor till något event kan vara av olika typer, såsom företag eller avdelning. Men en enskild sponsor kan aldrig vara flera olika subtyper inom unionen (Precis som en union fungerar i programmeringsspråket C/C++).

**Relationsmodellen**

I början av databassystem fanns många problem med strukturen i hur data lagras. Om data ändrades förändrades även strukturen av databasen. Nackdelen är uppenbar - om program var byggda för den gamla strukturen var de tvungna att skrivas om. *Relationsmodellen* löste detta genom att göra databasen *dataoberoende* och program som använder data i databasen behöver därmed inte känna till något som struktur av hur data faktiskt lagras.   
  
Modellens datastruktur kallas för *table* och är enkelt sett en tvådimensionell matris (rader och kolumner). En databas består alltså av *Tables* som håller all information om alla objekt. Ett *tables* rader visar på instanser av vad *tablet* ska visa, och kolumnerna visar på vilka attribut som finns i den instansen.  
Ex.1. En *table* Students har kolumner för stuID, firstName lastName, och varje rad representerar olika instanser av student.   
Ex.2 En *table* Teach har kolumner för stuID och teacherID, och varje rad representerar en “instans” av relationen “Teach”.

Modellen följer även strikta regler för att uppnå *dataoberoende* såsom:

* Varje cell i en *table* har ett och endast ett värde
* Varje kolumn bestämmer vilket attribut varje cell i kolumnen beskriver (detta innebär också att varje cells värde ingår i samma *domän*)
* Varje rad är en distinkt uppsättning av alla kolumners attribut (inga rader är likadana), däremot spelar inte radernas ordning någon roll

Matematiskt perspektiv

Varje rad är en uppsättning av de olika attributens (kolumnerna) värden inom deras domän. Detta gör att det finns en sorts “gräns” för hur många instanser (rader) ett *table* kan ha. Ett *table* kan alltså innehålla alla subset som finns i den kartesiska produkten av attributen (denna produkt är förklarad tidigare i sammanfattningen). Relationsmodellens sätt att strukturera data är alltså grundat på denna matematiska beräkning där varje relation i modellen ses som ett subset av den kartesiska produkten.

Översättning från ER-modellen

ER-modellen visar bara på en översiktlig modell för att en designer ska konceptuellt kunna visualisera en tänkt struktur i en databas. Relationsmodellen kan byggas upp med hjälp av en ER-modell och man behöver då “översätta” ER-modellen till relationsmodellen. För varje koncept i ER-modellen finns sätt att implementera detta i relationsmodellen. Några exempel:

* *Sambandets kardinalitet* representeras som antalet rader i ett *table*
* *Sambandsgrad* representeras som antalet kolumner i ett *table*
* *Nycklar* är värden i celler under nyckelattibutskolumnen, t.ex. kolumn 1 stuID

# Uppgift 1

