Databasteknik

föreläsare: Kjell Lindqvist

NADA

Vad är

data?

och

databaser?

och

vad är de bra för?

och

varför ska ni kunna något om dem?

Vad är ...

Data är en representation av information. I datorn endast ett bitmönster. För att återskapa informationen måste vi tolka data korrekt. Vi behöver alltså data om data och inte bara data.

En databas är en lagringsplats för data och ett programsystem för att hantera lagringsplatsen och dess data.

Vad är databaser bra till?

Alla organistaioner behöver spara information för senare användning.

Redovisning av verksamheten till berörda myndigheter (skattemyndigheter, tillsynsmyndigheter, et.c.).

Internt har man behov av att känna till sina anställda, betala deras löner, hålla reda på vad de sysslar med ...

Det betyder att

Vad är . . .

- man kan tvingas spara stora mängder data
- många olika användare måste kunna komma åt dem, ibland kanske samtidigt
- flera användare kan tänkas ändra i data (uppdatera, korrigera, lägga till, . . .)
- man måste ha ett uniformt (enkelt) sätt (språk) för att hantera data och för att ställa frågor om data så att alla behöriga kan göra det
- obehöriga inte ska kunna komma åt dem, m.m.

Varför använder man databaser? . . .

Moderna databaser, databassystem och datbashanteringssystem har alla egenskaper för att uppfylla kraven.

- Man kan hantera datamängder upp till PB-storlek (1 petabyte = 10^{15} tecken)
- Man har samtidighetskontroll
- Man har frågespråk (inte bara för frågor)
- De uppfyller höga krav på säkerhet.

Relationsdatabaser

Data organiseras (ytligt sett) som *tabeller* som kallas *relationer*. Kolumnrubrikerna kallas *attribut* och raderna kallas *tupler*. En relation *Gaturegister* kan se ut något i stil med:

Namn	Kartblad	Ruta	Offset-Ö	Offset-S
Ahlströmska skolan	38	B5	5	12
Karlaplan	38	B5	24	13
Karlavägen	37	F4	12	17
. . .	. . .	. . .	. . .	. . .

Relationsdatabaser . . .

SQL (structured query language) har blivit en defacto standard och bygger på relationsmodellen. Det är inte svårt ens för den oinvigde att förstå enkla frågor, t.ex.:

```
SELECT Namn FROM Gaturegister WHERE Kartblad = 38;
```

som går igenom alla rader i tabellen ”Gaturegister”, plockar ut de rader som stämmer med villkoret och plockar ut namnen ur dessa.

I de praktiska systemen måste man kunna ställa frågor som är mer komplicerade också som ”icke-experter”. Därför finns det möjlighet till optimering av frågorna.

Ner- och uppskalning

Från början var databassystem förunnat endast stora företag eftersom programmen var stora och dyra. Idag får man rum med en databashanterare och ett stort antal databaser på en ordinär PC.

Men de stora databaserna blir större och större. De allra största, de som har med stjärnhimlen att göra, har med temporala och spatiala aspekter. De växer med c:a 1–10 TB per dygn.

Det är svårt att tänka sig databaser som är mer än en *petabyte* (10^5 kB) stora, men det finns alltså så stora databaser. Normala lagringsmedia räcker inte för att hålla dem ”online”.

Varför måste ni kunna något om dem?

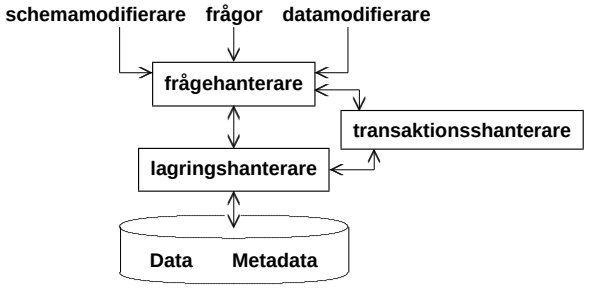
Databaser är viktiga i all slags ingenjörsverksamhet idag.

Fler och fler områden datoriseras.

Nästan alla kommer i kontakt med dem under sin yrkesverksamhet.

Hur är en databashanterare uppbyggd?

Man kan ha många aspekter av en databashanterares uppbyggnad. Man kan fokusera på funktionalitet.



Hur är en databashanterare . . .

Frågehanteraren **Vi har (generellt sett) tre olika metoder för input till databashanteraren**

1. **Frågor (queries) som ställs avseende data. Dessa kan hanteras via något generellt frågehanteringsgränssnitt eller genom ett applikationsprogramgränssnitt.**
2. **Ändringar (modifications). Operationer för att uppdatera innehållet i befintliga databaser. Kan liksom frågorna hanteras antingen via något generellt frågehanteringsgränssnitt eller genom ett applikationsprogramgränssnitt.**
3. **Schemaändringar. Operationer för att skapa nya databaser eller för att förändra strukturen hos befintliga. Sker endast genom ett specialgränssnitt som bara den s.k. databasadministratören får använda.**

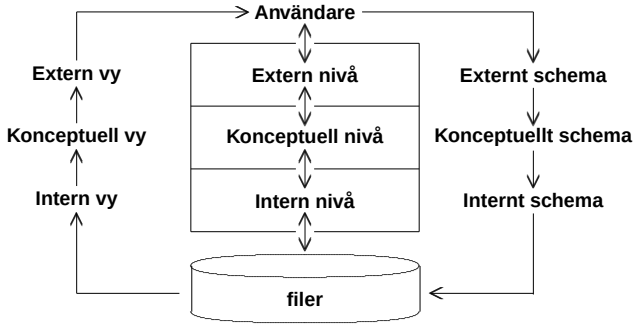
Hur är en databashanterare . . .

Transaktionshanteraren **Har till uppgift att se till att varje förändring av databasen har s.k. *ACID*-egenskaper:**

1. **Atomicitet (*atomicity*). Varje operation ska utföras som en odelbar enhet, fullständigt eller inte alls. Man accepterar inte att en del utförs men inte resten.**
2. **Konsistens (*consistency*). Det får inte finnas motsägelser i datamängden.**
3. **Isolering (*isolation*). Två transaktioner som utförs parallellt måste ge samma resultat som om de utförts i sekvens.**
4. **Beständighet (*durability*). Utförda förändringar får aldrig gå om intet (oplanerat), inte ens om systemet kraschar.**

Hur är en databashanterare . . .

Man skulle också kunna fokusera på abstraktion.



Databasmodellering

1. **När vi studerar verkligheten för att beskriva (en del av) den i ett system (inte bara datorer), måste vi göra en modell.**

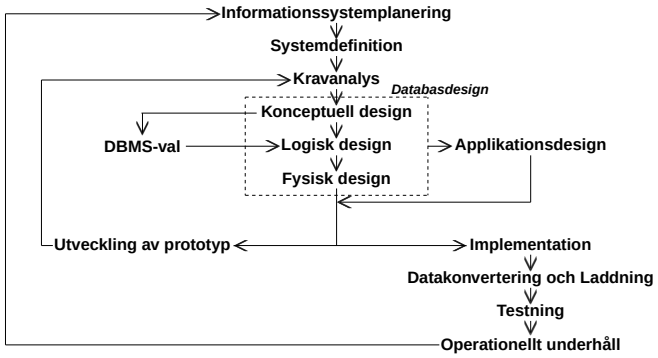
2. **Modelleringsverktyget ska ha egenskaper som gör att vi:**
   - **inte behöver kunskap om de verktyg som senare krävs för att realisera systemet.**
   - **lätt får överblick över modellen.**
   - **har tillgång till detaljer när vi vill**
   - **kan bortse från detaljer om vi vill**
   - **slipper beroende av den dator som senare skall användas.**
   - **lätt kommer vidare från modell till databasstruktur.**

Databasmodellering . . .

Databasen är endast en del av det system som krävs för att täcka *informationsbehovet* hos en organistation. Normalt måste man undersöka och analysera alla informationssystemets komponenter:

- **Applikationsprogram**
- **Databashanteringssystem**
- **Databas**
- **Hårdvara**
- **Personal**

Databasmodellering . . .



Databasmodellering . . .

Vi kommer bara bekymra oss om den streckade boxen med titeln *Databasdesign* och anta att det övriga är klart.

Praktiskt blir det lite annorlunda:

I en labb kommer vi att testa SQL – det språk som blivit en ”de facto” standard för att dels bygga databaser, dels undersöka och ändra innehållet i en databas.

Objekt – objektklasser

Alla modelleringsverktyg för relationsdatabaser bygger på att man identifierar mängder av intressanta objekt i den del av världen man vill avbilda. Dessa brukar man kalla objektklasser. Det man avbildar i modellen är ”möjligheter”.

Det *kan* finnas sådana objekt som beskrivs i objektklasserna. Det är *möjligt* och *troligt*, men *inte säkert*.

Grafiskt brukar man markera objektklasserna som rektanglar med klassens namn i. Namnet är i singularis och *ska* vara beskrivande.

Person

Objektklasser . . .

Intressanta objektklasser(objektmängder) brukar vara:

- ***Objekt i egentlig mening:*** **Flygplan, tåg, människor, hus, bilar, husgeråd, kontor, fabriker, et.c.**
- ***Roller:*** **Läkare, lärare, husägare, bilist, hyresgäst, handledare, köpare, säljare, et.c.**
- ***Händelser:*** **Händelse, olycka, systemkrasch, födelsedag, et.c.**
  (!! kanske !!)
- ***Specifikationer:*** **Modellbeskrivningar, transaktionstyper, m.m.**

Objekt . . .

Ett objekt är en *instans* av en objektklass, och är entydigt identifierbar genom en *I-term (identifikationsterm)*.

Informellt: ett objekt är en förekomst i en tabell med ett entydigt nyckelvärde som identifierar objektet.

Ex.:

Person

Personnr	Namn	Gatuadress	Gatunr
640111-0010	Kalle	Karlavägen	12
640212-0028	Mona	Månvägen	7
640313-0046	Vera	Verdandig	3

Objekt och objektklasser . . .

Objektklassens egenskaper beskrivs av ett antal *attribut*.

För ett enskilt objekt har ett av dessa attribut, I-termen, ett för tabellkolumnen unikt värde ur motsvarande domän, medan övriga attribut (*E-termer, egenskapstermer*) har ett värde ur attributets domän eller är värdet *null*.

Alla attribut redovisas i en s.k. *egenskapsmatris*

Klass	Namn	I-termer	E-termer
Obj	Person	Personnr	Namn, Gatuadress, Gatunr
. . .	. . .	. . .	. . .

Databasmodellering . . .

Modellen kommer att bestå av ett antal objektklasser, deras egenskaper (attribut) och dessutom de *associationer* (*samband*) som *kan finnas* mellan objekten i objektklasserna.

Sambanden samlas i sambandsklasser som utgör ”kopplingen” mellan objektklasserna. Så kommer hela modellen att utgöra en sammanhållen graf.

Samband och sambandsklasser

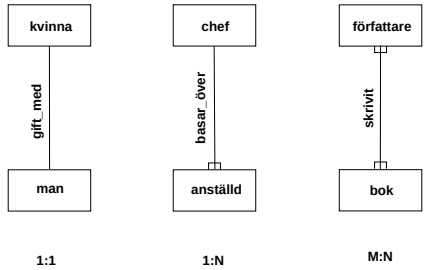
- **Allt som utgör en association mellan objektklasser: Försäljning, Lån (i bank eller på bibliotek), boende, flight, resa, m.m. (!! det här är inte alltid så säkert !!).**
- **Allmänt: sambandsklasser definierar ett beroende mellan objektklasser.**
- **Identifieras med i-termerna från de objektklasser de associerar till varandra.**
- **Ett samband utgörs då av en förekomst i en tabell och identifieras entydigt av en nyckel bestående av flera i-termer.**
- **Samband samlas alltså i tabeller som motsvarar sambandsklassen.**
- **Sambandsklasser ritas som namngivna linjer dragna till de objektklasser som relateras till varandra.**
- **Namnet skall helst vara ett substantiv i singularis.**

Sambandsklassers grad . . .

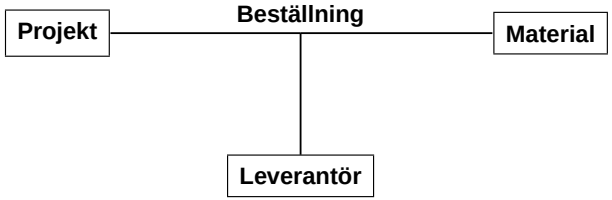
Graden (eller ordningen) på en sambandsklass är antalet objektklasser som sambandsklassen knyter samman.

- 1:a gradens sambandsklasser finns naturligtvis inte.
- Sambandsklasser som associerar en objektklass med sig själv räknas också som 2:a gradens.
- Man kan lägga på restriktioner på 2:a gradens sambandsklasser.
  - ***1:1*** om ett objekt från den ena objektklassen associeras med *max ett* objekt från den andra och vice versa.
  - ***1:N*** om ett objekt från den ena objektklassen kan associeras med *mer än ett* objekt från den andra medan objekt från den andra klassen kan associeras med *max ett* objekt från den första.
  - ***M:N*** om ett objekt från den ena objektklassen kan associeras med *mer än ett* objekt från den andra och vice versa.

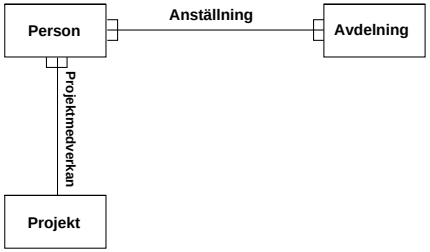
Samband och sambandsklasser . . .



Samband och sambandsklasser . . .

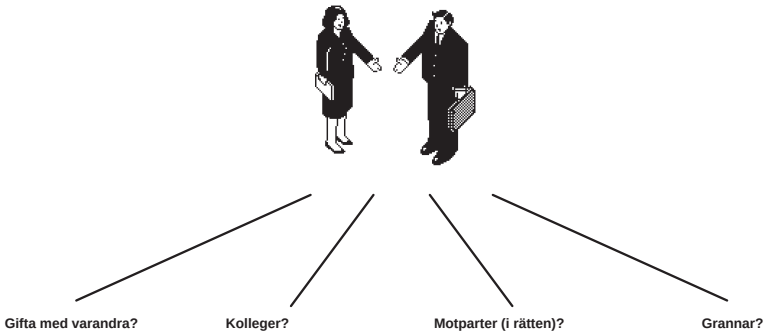


Samband och sambandsklasser . . .



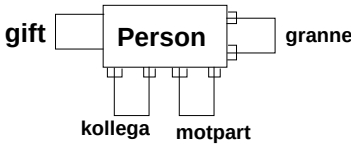
Samband och sambandsklasser . . .

Sambandsklasser kan vara lite knepiga:



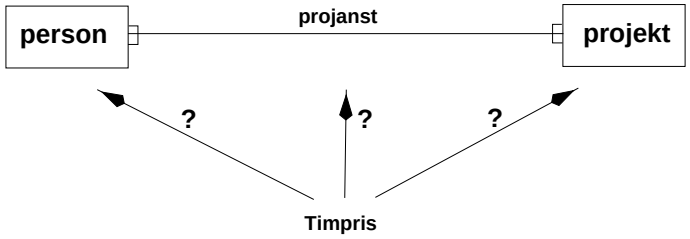
Samband och sambandsklasser . . .

Vi måste bestämma - situationen avgör.
Vad vill vi med modellen??
Två personer kan ha alla dessa samband med varandra.



Samband och sambandsklasser . . .

Fördelningen av egenskaper går oftast bra men det finns fall då man måste noga analysera egenskaper som knyts till sambandsklasser. Ex:



Samband och sambandsklasser . . .

Idén med informationssystemsmodeller är att beskriva de logiska elementen i den del av världen som vi vill representera, d.v.s. att finna alla kopplingar mellan alla objekt, (= alla samband mellan dem) och objektens alla egenskaper (attribut).

För att man initalt ska vara nöjd med en modell ska den till slut utgöra en graf utan disjunkta delar, d.v.s. en sammanhängande struktur utan några lösa "öar".

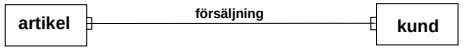
Samband och sambandsklasser . . .

Ett varuhus säljer en viss artikel.

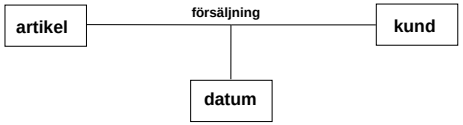
1. **Det finns en artikel**

2. **Det finns en kund**

3. **Det finns ett samband (försäljning)**

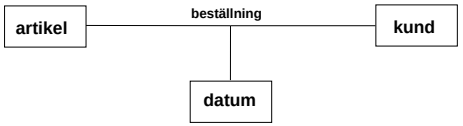


4. **Köpet ägde rum ett viss datum**



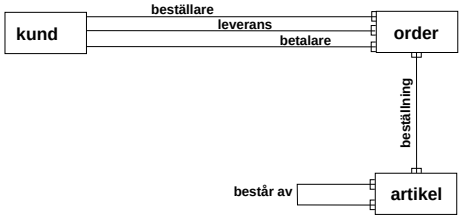
5. **eller . . .**

Det finns ett samband (beställning)



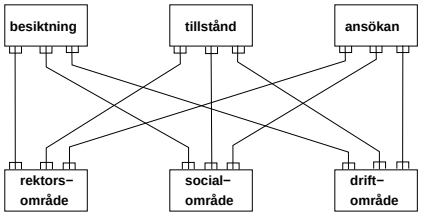
6. **eller . . .**

Kunden har gjort en beställning och har en viss rabatt
Leveransen kan gå till ngn annan än beställaren
Betalaren kan vara ngn annan
En artikel kan vara ett set av artiklar

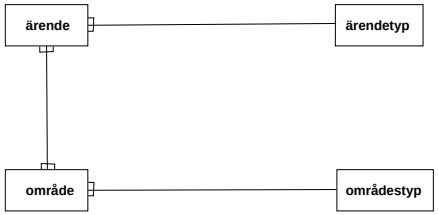


Hantera roller

Generalisera ibland



Generalisera ibland . . .



Ett lite större exempel

En ny hamburgerrestaurangkedja strävar efter att snabbt nå marknadsdominans genom att tillhandahålla produkter av färskaste kvalitet. Därför vill man att så lite som möjligt ligger kvar i lager vid stängningstid varje dag. Man försöker ta hem precis de mängder av allting som kommer att gå åt under dagen. Hänsyn tas till att åtgången varierar olika typer av dagar (fredagar, andra vardagar, lördagar och helgdagar). Personalpoolen måste vara flexibel. En person kan tjänstgöra i olika restauranger olika dagar beroende av var behovet är störst. Du får i uppdrag av ledningen för restaurangkedjan att, som ett led i den nödvändiga datoriseringen innefattande bl.a. skapandet av en databas, göra en informationsstruktur (datamodell) som representerar hela restaurangkedjan (ett tiotal restauranger), där följande data skall ingå:

Ett lite större exempel . . .

- Restaurangdata (antal sittplatser, adress mm.)
- Lagersaldon och priser för alla produkter
- Beräknad åtgång av produkter och personal olika typer av dagar för de olika restaurangerna
- Personaldata : namn, adress, lön etc, arbetad tid / restaurang (dvs hur mycket och var)
- Kalenderdata (typ för varje dag)
- osv

Indata till systemet blir:

- Sålda produkter (direkt via kassaterminal)
- Ändrade priser och lagersaldokorrektioner
- Ändrade arbetsscheman för personalen

- Kalenderdatainmatning (t.ex. att 17/5 är en helgdag)
- osv

Utdata:

- Beställningslistor och personalbesättning för nästa dag / restaurang
- Lönelistor
- osv

Börja med att hitta objekten

exempel . . .

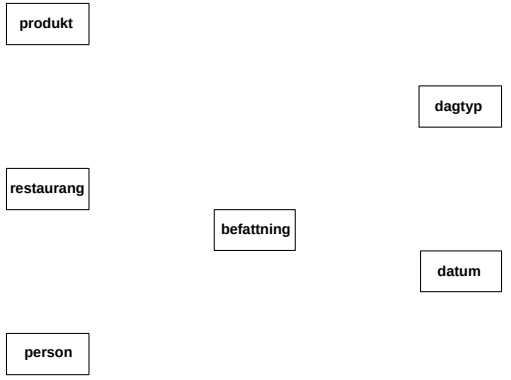
Personalpoolen måste vara flexibel. En person kan *tjänstgöra i olika restauranger olika dagar* beroende av var behovet är störst. Följande data skall ingå:

- *Restaurangdata* (antal sittplatser, adress mm.)
- Lagersaldon och priser för alla *produkter*
- Beräknad åtgång av produkter och personal olika *typer av dagar* för de olika restaurangerna
- *Personaldata*: namn, adress, lön etc, arbetad tid / restaurang (dvs hur mycket och var)
- *Kalenderdata* (typ för varje dag)
- osv

Indata till systemet blir:

- Sålda produkter (direkt via kassaterminal)
- Ändrade priser och lagersaldokorrektioner
- Ändrade arbetsscheman för personalen
- Kalenderdatainmatning (t.ex. att 17/5 är en helgdag)
- osv

exempel . . .



exempel . . .

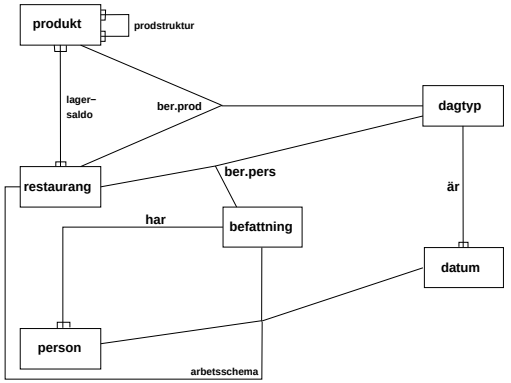
Personalpoolen måste vara flexibel. En person kan tjänstgöra i olika restauranger olika dagar beroende av var behovet är störst. Följande data skall ingå:

- Restaurangdata (antal sittplatser, adress mm.)
- *Lagersaldon* och priser för alla produkter
- *Beräknad åtgång av produkter* och *personal* olika typer av dagar för de olika restaurangerna
- Personaldata: namn, adress, *lön etc, arbetad tid* / *restaurang* (dvs hur mycket och var)
- Kalenderdata *(typ för varje dag)*
- osv

Indata till systemet blir:

- Sålda produkter (direkt via kassaterminal)
- Ändrade priser och lagersaldokorrektioner
- Ändrade arbetsscheman för personalen
- Kalenderdatainmatning (t.ex. att 17/5 är en helgdag)
- osv

exempel . . .

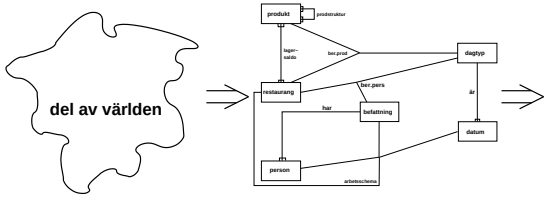


exempel . . .

Sedan måste man lägga till alla egenskaper. Det gör vi med en egenskapsmatris:

Typ	Namn	I-termer	E-termer
Obj	Produkt	Pnamn	Pris
Obj	Resturang	Rnamn	Platsantal,Adress,Telnr
Obj	Dagtyp	Typbet	
Obj	Personal	Pnr	Namn,Adress,Telnr
Obj	Datum	Datum	
Obj	Befattning	Befbet	Arbetsbeskrivning
Samb	Produktåtg	Pnamn,Rnamn,Typbet	Kvantitet
Samb	Personalåtg	Rnamn,Typbet,Befbet	Antal
Samb	Lagersaldo	Pnamn,Rnamn	Mängd
Samb	Arbetsschema	Rnamn,Pnr,Datum,Befbet	Starttid,Sluttid
Samb	Är	Typbet,Datum	
Samb	Produktstruktur	Pnamn-I,Pnamn-B	Kvantitet
Samb	Harbef	Pnr,Befbet	Lön

Hur får vi in detta i datorn?



Basrelationer

rel1 (a1, b1, c1, ..)
rel2 (a2, b2, c2, ..)
rel3 (a3, b3, c3, ..)
rel4 (a4, b4, c4, ..)
rel5 (a5, b5, c5, ..)
.....

Vad betyder detta?

BasrelationsNamn (primärNyckel, attributLista);

Där primärnyckeln består av ett eller flera attribut.

Modell⇒ Struktur

a. Objektklass som innehåller e-term(er) bildar en basrelation.
b. Objektklass som bara består av en i-term och finns på N-sidan av någon sambandsklass av typ 1:N bildar basrelation.
c. Sambandsklass av högre ordning än 2 bildar basrelation.
d. Sambandsklass av ordning 2 och typ M:N bildar basrelation.
e. Sambandsklass av ordning 2 och typ 1:N försvinner men I-termen i objektklassen som finns på 1-sidan blir E-term i objektklassen på N-sidan. Eventuella E-termer i sambandsklassen blir E-termer i objektet på N-sidan.
f. Sambandsklass av typ 1:1 behandlas som om den vore av typen 1:N med ”godtyckligt” val av ”N-sida” eller så försvinner de och objektklasserna de förbinder slås samman.

Modell⇒ Struktur . . .

Alla objektklasser som innehåller E-termer bildar en basrelation.

Produkt	(Pnamn, Pris)
Resturang	(Rnamn, Platsantal, Adress, Telnr)
Personal	(Pnr, Namn, Adress, Telnr)
Befattning	(Befbet, Arbetsbeskrivning)

Modell⇒ Struktur . . .

Objektklass som bara består av en i-term och finns på N-sidan av någon sambandsklass av typ 1:N bildar basrelation.

Produkt	(Pnamn, Pris)
Resturang	(Rnamn, Platsantal, Adress, Telnr)
Personal	(Pnr, Namn, Adress, Telnr)
Befattning	(Befbet, Arbetsbeskrivning)
Datum	(Datum)

Alltså försvinner

Dagtyp	(Typbet)

Modell⇒ Struktur . . .

Alla sambandsklasser av högre ordning än 2 bildar basrelationer.

Produkt	(Pnamn, Pris)
Resturang	(Rnamn, Platsantal, Adress, Telnr)
Personal	(Pnr, Namn, Adress, Telnr)
Befattning	(Befbet, Arbetsbeskrivning)
Datum	(Datum)
Produktåtg	(Pnamn, Rnamn, Typbet, Kvantitet)
Personalåtg	(Rnamn, Typbet, Befbet, Antal)
Arbetsschema	(Rnamn, Pnr, Datum, Befbet, Starttid, Sluttid)

Modell⇒ Struktur . . .

Alla binära sambandsklasser av typ M:N bildar basrelationer.

Produkt	**(Pnamn, Pris)**
Resturang	**(Rnamn, Platsantal, Adress, Telnr)**
Personal	**(Pnr, Namn, Adress, Telnr)**
Befattning	**(Befbet, Arbetsbeskrivning)**
Datum	**(Datum)**
Produktåtg	**(Pnamn, Rnamn, Typbet, Kvantitet)**
Personalåtg	**(Rnamn, Typbet, Befbet, Antal)**
Arbetsschema	**(Rnamn, Pnr, Datum, Befbet, Starttid, Sluttid)**
Lagersaldo	**(Pnamn, Rnamn, Mängd)**
Produktstruktur	**(Pnamn-I, Pnamn-B, Kvantitet)**

Modell⇒ Struktur . . .

Sambandsklasser av typ 1:N försvinner men I-termen i objektklassen som finns på 1-sidan blir E-term i basrelationen på N-sidan.

Produkt	**(Pnamn, Pris)**
Resturang	**(Rnamn, Platsantal, Adress, Telnr)**
Personal	**(Pnr, Namn, Adress, Telnr, *Befbet*, *Lön*)**
Befattning	**(Befbet, Arbetsbeskrivning)**
Datum	**(Datum, *Typbet*)**
Produktåtg	**(Pnamn, Rnamn, Typbet, Kvantitet)**
Personalåtg	**(Rnamn, Typbet, Befbet, Antal)**
Arbetsschema	**(Rnamn, Pnr, Datum, Befbet, Starttid, Sluttid)**
Lagersaldo	**(Pnamn, Rnamn, Mängd)**
Produktstruktur	**(Pnamn-I, Pnamn-B, Kvantitet)**

Modell⇒ Struktur . . .

Sambandsklass av typ 1:1 behandlas som om den vore av typen 1:N med ”godtyckligt” val av ”N-sida” eller så försvinner de och objektklasserna de förbinder slås samman.

KLART

Frågespråk

För att få kontakt med en databashanterare och ställa frågor till den avseende innehållet i en databas måste vi antingen nöja oss med vad andra har skrivit ihop eller måste vi kunna skriva program själva.

- **Kan vi använda vanligt språk?**

  Stort forskningsområde, men inga användbara resultat.

- **Kan vi använda vanliga programspråk?**

Frågespråk . . .

Inte utan ansträngning.
Vanliga programspråk ser poster i taget.
Databashanteraren ser mängder av mängder i taget.
I databasen har man ett deklarativt synsätt
”dessa egenskaper skall resultatet ha”.
I högnivåspråken har man ett imperativt synsätt
”låt oss göra så här och se vad vi får till resultat”.

- **Alltså: Nya språk är nödvändiga för kommunikationen. De kan byggas på utvidgningar och, i viss mån, specialisering av traditionell matematik.**

Frågespråk . . .

- **En databas kan ses som en mängd tabeller.**
- **En tabell kan ses som en mängd rader.**
- **En rad som en mängd värden.**

Vi kan då tänkas använda mängdalgebra eller predikatkalkyl.

Vad vet vi mer?

Frågespråk . . .

1. **Namnen på delmängderna som kallas *basrelationer* (tabeller).**
2. **Utseendet på elementen i dessa delmängder (raderna i tabellerna, *tupler*). De är i sig mängder.**
3. **Vi känner namnen på tuplernas (radernas) delar (attributen – kolumnnamnen), eller deras faktiska inbördes ordning.**
4. **Vi känner också attributens värdedomäner.**

Frågespråk . . .

Man kan definiera speciella operationer som en påbyggnad på mängdalgebra och kallas relationsalgebra som vi kommer att titta lite på.

Man kan också definera matematik som liknar predikatkalkyl med variabler som löper över tabellrader eller med variabler som löper över värdedomäner. Man får då tupelkalkyl eller domänkalkyl.

Språk finns som implementerar alla tre synsätten, men hybrider är vanligast.

Språken kallas frågespråk (Query Language) trots att frågandet är endast en liten del av språket, men mekanismerna används för andra uppgifter också.

Frågespråk . . .

Språken består av två delar: *DML* (=Data Manipulation Language) och *DDL* (= Data Definition Language).

Vi skall titta närmare på ett språk, *SQL* (Structured Query Language), som de facto blivit mer eller mindre standard på området och som delvis bygger på relationsalgebra.

Meningen med en fråga till databasen är att ur mängden av relationer, ta fram just de data i relationerna som svarar mot ett visst behov hos användaren.

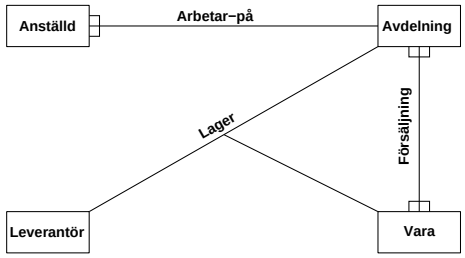
Dessa data behöver inte komma direkt ur en basrelation utan kan formas av fiktiva relationer som vi specificerar m.h.a. de som redan finns i databasen.

Frågespråk . . .

Vi kan alltså ur de existerande relationerna plocka bitar som vi sätter ihop till tupler i egendefinierade relationer.

En databas över ett fiktivt och mycket enkelt varuhus kan tjäna som bas för vissa exempel:

Varuhuset . . .



Varuhuset, databasstruktur . . .

Anställd	**(namn, lön, chef, avd)**
Försäljning	**(avd, varunr, volym)**
Leverantör	**(företag, adress)**
Lager	**(företag, avd, varunr, volym)**
Avdelning	**(avd, våning)**
Vara	**(varunr, typ)**

Relationsalgebra

Matematiken bakom frågespråken är viktig. Med relationsdatabaserna fick man för första gången produkter som grundades på sunda matematiska teorier och som visade sig hålla måttet.

Relationsalgebra bygger på mängdalgebra, men man behöver inte kunna mängdalgebran för att förstå.

För att ett databasspråk ska sägas vara matematiskt komplett måste man kunna extrahera alla enskilda värden och deras kombinationer enligt den givna modellen. Eftersom man har gedigen kunskap om de mängder man hanterar är det inte svårt att nå.

Det enda riktigt starka kravet för att matematiken ska fungera (utan utvidgningar) är kravet på *atomicitet*. De attributvärden man hanterar måste vara sådana att det finns *precis ett värde* för varje attribut och tupel.

Relationsalgebra . . .

Fem operationer behövs:

Varje operation ger en basrelation till resultat.

De mest grundläggande operatorerna hämtar rader resp kolumner från tabellerna. Genom en kombination av sådana operationer är det uppenbart att man kan nå varje enskilt värde i en tabell.

För att hitta rader med vissa egenskaper måste man kunna hantera villkor.

Selektion

om $\mathcal{R}$ är en basrelation och f en formel bestående av

1. **operander som är konstanter eller attributnamn,**
2. **de aritmetiska jämförelseoperatorerna, $<$, $=$, $>$, $\leq$, $\neq$, $\geq$,**
3. **de logiska operatorerna $\wedge$ (och), $\vee$ (eller) och $\neg$ (icke).**

Då är $\sigma_f(\mathcal{R})$ den mängd av tupler i $\mathcal{R}$ för vilka f är sann.

Resultatet har alltid samma attribut som $\mathcal{R}$.

Selektion (σ) . . .

Ex.

Finn alla anställda som tjänar mer än 24000

$\sigma_{\text{lön} > 24000}(\text{Anställd})$

Villkoret kan vara sammansatt:
Finn alla anställda som tjänar mer än 24000 och har Kalle till chef.

$\sigma_{\text{lön} > 24000 \wedge \text{chef} = \text{'Kalle'}}(\text{Anställd})$

eller

$\sigma_{\text{lön} > 24000}(\sigma_{\text{chef} = \text{'Kalle'}}(\text{Anställd}))$

Selektion ...

eller

$\sigma_{\text{chef} = \text{'Kalle'}}(\sigma_{\text{lön} > 24000}(\text{Anställd}))$

Operationen är kommutativ.

Finn alla som tjänar mer än 24000 eller har Kalle till chef.

$\sigma_{\text{lön} > 24000 \vee \text{chef} = \text{'Kalle'}}(\text{Anställd})$

Finn alla som tjänar mer än 24000 men inte har Kalle till chef.

$\sigma_{\text{lön} > 24000 \wedge \neg(\text{chef} = \text{'Kalle'})}(\text{Anställd})$

$\sigma_{\text{lön} > 24000 \wedge \text{chef} \neq \text{'Kalle'}}(\text{Anställd})$

Projektion (Π)

Idén är att vi tar bort och/eller arrangerar om attributen i en relation.

Ex: Finn namnen på alla företag

$\Pi_{\text{Företag}}(\text{Leverantör})$

Ex.: Finn alla löner

$\Pi_{\text{Lön}}(\text{Anställd})$

Selektion och projektion ...

Vi kan redan ställa relativt komplicerade frågor.

Ex.: Finn namn och lön för alla anställda som tjänar mer än 24000 och inte har Kalle som chef

$\Pi_{\text{Namn, Lön}}(\sigma_{\text{lön} > 24000 \wedge \text{chef} \neq \text{'Kalle'}}(\text{Anställd}))$

Unionskompatibilitet

De återstående operationerna bland de fem nödvändiga är sådana som vi finner i traditionell mängdlära, men vi måste på två av dem lägga vissa restriktioner.

Jag kommer i fortsättningen att då och då ange relationernas *schema*. Relationen $\mathcal{R}$ har schemat $\mathcal{X}$ skrives $\mathcal{R}(\mathcal{X})$ och betyder att $\mathcal{X}$ är en lista med attribut som anger vilka kolumnrubriker tabellen har. Attributen räknas upp i en viss ordning men den ordningen har ingen betydelse.

Unionskompatibilitet ...

Det finns olika def på unionskompatibilitet. Den vanligaste är (informellt):

Två relationer $\mathcal{R}(\mathcal{X})$ och $\mathcal{S}(\mathcal{Y})$ är unionskompatibla om det finns en avbildning från $\mathcal{X}$ till $\mathcal{Y}$ som avbildar attributen i $\mathcal{X}$ på attributen i $\mathcal{Y}$ och vice versa så att både namn och domän stämmer.

Nu är det klart för def av de tre övriga nödvändiga operationerna.

Union ($\cup$) ...

Kräver unionskompatibilitet.

Unionen mellan $\mathcal{R}$ och $\mathcal{S}$ utgörs av mängden av de tupler som finns antingen i $\mathcal{R}$ eller i $\mathcal{S}$ eller i båda.

Operationen är

kommutativ, d.v.s. $\mathcal{R} \cup \mathcal{S} \equiv \mathcal{S} \cup \mathcal{R}$ och

associativ, d.v.s. $\mathcal{R} \cup (\mathcal{S} \cup \mathcal{T}) \equiv (\mathcal{R} \cup \mathcal{S}) \cup \mathcal{T}$

$\boldsymbol{\sigma}_{\text{lön} > 24000 \vee \text{chef} = \text{'Kalle'}}(\textbf{Anställd}) \equiv$
$\boldsymbol{\sigma}_{\text{lön} > 24000}(\textbf{Anställd}) \cup \boldsymbol{\sigma}_{\text{chef} = \text{'Kalle'}}(\textbf{Anställd})$

Differens ($-$)

Kräver unionskompatibilitet.

Differensen mellan $\mathcal{R}$ och $\mathcal{S}$ utgörs av mängden av de tupler som finns i $\mathcal{R}$ *men inte* i $\mathcal{S}$, d.v.s att man operationellt kan tänka sig att man går igenom $\mathcal{S}$ och för varje tupel ser efter om den finns i $\mathcal{R}$ och i så fall tar bort den.

Operationen är *varken*

kommutativ, d.v.s. $\mathcal{R} - \mathcal{S} \not\equiv \mathcal{S} - \mathcal{R}$ *eller*

associativ, d.v.s. $\mathcal{R} - (\mathcal{S} - \mathcal{T}) \not\equiv (\mathcal{R} - \mathcal{S}) - \mathcal{T}$

Kartesisk produkt ($\times$)

Kräver inte unionskompatibilitet.

Den kartesiska produkten av $\mathcal{R}$ och $\mathcal{S}$ utgörs av mängden av de tupler som man konstruerar genom att ta varje tupel i $\mathcal{R}$ och ”slå ihop” med varje tupel i $\mathcal{S}$. Resultatrelationen får som attributmängd båda relationernas attribut. Så om vi har

$\mathcal{R}$		$\mathcal{S}$	
a	**b**	**c**	**d**
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2
a_3	b_3		

Kartesisk produkt ($\times$) ...

så blir $\mathcal{R} \times \mathcal{S}$

a	b	c	d
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2
a_2	b_2	c_1	d_1
a_2	b_2	c_2	d_2
a_3	b_3	c_1	d_1
a_3	b_3	c_2	d_2

Ex.: Vilka jobbar på första våningen? ***Här ser vi även hur namnkollisioner hanteras.*** **Jag kallar Anställd för $\mathcal{R}$ och Avdelning för $\mathcal{S}$.**

$\Pi_{\mathcal{R}.\text{Namn}}(\sigma_{\mathcal{R}.\text{avd}=\mathcal{S}.\text{avd} \wedge \mathcal{S}.\text{vån}=1}(\mathcal{R} \times \mathcal{S}))$

Relationsalgebra ...

Dessa fem räcker för att göra ”allt”. Vi kan plocka ut vilken del som helst av en tupel och vilka tupler som helst ur en relation.

Vi kan konstruera nya tupler och relationer.

Vi kan slå ihop eller dela upp relationer.

Men det blir inte lätt. Ett antal andra relationsoperatorer finns definierade.

De kan alla härledas ur de fem redan definierade operatorerna.

Skärning ($\cap$)

Kräver unionskompatibilitet.

$\mathcal{R} \cap \mathcal{S}$ är mängden av tupler som finns i *både $\mathcal{R}$ och $\mathcal{S}$*.

Kan uttryckas som $\mathcal{R} - (\mathcal{R} - \mathcal{S})$.

***Uttrycket går inte att förenkla*.**

theta-join ($\bowtie_\theta$)

Join kan ses som en kartesisk produkt följd av en selektion.

$\mathcal{R} \bowtie_{\mathcal{R}.\text{a}=\mathcal{S}.\text{b}} \mathcal{S} \equiv \sigma_{\mathcal{R}.\text{a}=\mathcal{S}.\text{b}}(\mathcal{R} \times \mathcal{S})$

Anställd $\bowtie_{\text{Anställd.avd} <> \text{Avdelning.avd}}$ Avdelning

En thetajoin med jämförelseoperatorn $=$ kallas vanligen för en ekvijoin.

Naturlig join ($\bowtie$)

Naturlig join av $\mathcal{R}$ och $\mathcal{S}$ skrivs $\mathcal{R} \bowtie \mathcal{S}$.

Det måste finnas åtminstone en kolumn i $\mathcal{R}$ och $\mathcal{S}$ med samma namn och domän. Enklaste beskrivnningen:

1. **Beräkna $\mathcal{R} \times \mathcal{S}$.**
2. **För varje attribut $\mathcal{A}_i$ i $\mathcal{R}$:**
   Om $\mathcal{A}_i$ är namnet på ett attribut även i $\mathcal{S}$ så selekteras de tupler i $\mathcal{R} \times \mathcal{S}$ där $\mathcal{R}.\mathcal{A}_i = \mathcal{S}.\mathcal{A}_i$.
3. **Slopa sedan de redundanta kolumnerna.**

Om det finns *en* kolumn med gemensamt namn är operationen ekvivalent med en ekvijoin över kolumnerna med samma namn.

Relationsvariabler

I vissa fall kan det vara viktigt att kunna döpa om attribut och vi skall införa en formalism för detta också.

Vi inför begreppet *relationsvariabel* som innebär att vi kan namnge resultatet av operationer och därmed även erhålla temporär lagring av resultat (finns inte i boken).

Ex: I stället för

$\sigma_{\text{lön} > 24000 \wedge \text{chef} = \text{'Kalle'}}(\text{Anställd})$ **eller**
$\sigma_{\text{lön} > 24000}(\sigma_{\text{chef} = \text{'Kalle'}}(\text{Anställd}))$

kan vi skriva:

Relationsvariabler ...

tmp $\leftarrow \sigma_{\text{chef} = \text{'Kalle'}}(\text{Anställd})$

resultat $\leftarrow \sigma_{\text{lön} > 24000}(\text{tmp})$

Då vi tilldelar en relationsvariabel kan vi samtidigt ge nya namn åt attributen (kolumnerna). Gör vi det måste *alla* attribut i resultatet få nya namn, inte bara något eller några.

Ex: Om vi vill, kan vi plocka fram alla anställda och deras löner genom

resultat(Anst, månadslön) $\leftarrow \Pi_{\text{namn, lön}}(\text{Anställd})$

Relationsvariabler ...

Enligt den givna definitionen är Avdelning och Vara inte unionskompatibla, men Avdelning är unionskompatibel med tmp(avd, våning) $\leftarrow \Pi_{\text{typ, varunr}}(\text{Vara})$ **och operationen är utförbar även om resultatet inte har någon rimlig användning.**

Försäljning är unionskompatibel med $\Pi_{\text{avd, varunr, volym}}(\text{Lager})$ **utan namngivning men (även här) – vad ska man ha resultatet till?**

Aggregat . . .

Ett aggregat eller en aggregatfunktion opererar på hela kolumner och utför någon form av beräkning. Standarduppsättningen är:

(SUM) Sum beräknar summan av alla värden i en kolumn.

(AVG) Average beräknar medelvärdet av alla värden i en kolumn.

(MAX) Maximum plockar ut det största av alla värden i en kolumn.

(MIN) Minimum plockar ut det minsta av alla värden i en kolumn.

(COUNT) Count räknar antalet tupler i en relation.

SQL

Frågor till databasen ställs med

select [distinct] *lista-av-attribut*

from *lista-av-relationer*

where *lista-av-predikat*

[group by *annan-lista-av-attribut*

[having *annan-lista-av-predikat*]]

[order by *tredje-lista-av-attribut*]

SQL . . .

Selektion:

select * from Anställd
skriv ut tabellen anställd

select * from Anställd where lön > 24000
skriv ut de anställda som tjänar mer än 24000

select * from Anställd where lön > 24000 and chef = 'Kalle'

select * from Anställd where lön > 24000 or chef = 'Kalle'

select * from Anställd where lön > 24000 and not chef = 'Kalle'
select * from Anställd where lön > 24000 and chef <> 'Kalle'

Tabellen anställd

Namn	Lön	Chef	Avd
Anna	26400	Lena	Mat
Bengt	23000	Lena	Mat
Bosse	23000	Kalle	Skor
Eva	25000	Kalle	Skor
Evert	24200	Lena	Mat
Jonas	44000	NULL	NULL
Kalle	36000	Jonas	Skor
Karin	24000	Lena	Mat
Laila	23600	Lena	Mat
Lena	36000	Jonas	Mat
Mia	24000	Kalle	Skor
Pelle	26000	Kalle	Skor
Pia	24200	Kalle	Skor

select * from Anställd where lön > 24000

Namn	Lön	Chef	Avd
Anna	26400	Lena	Mat
Eva	25000	Kalle	Skor
Evert	24200	Lena	Mat
Jonas	44000	NULL	NULL
Kalle	36000	Jonas	Skor
Lena	36000	Jonas	Mat
Pelle	26000	Kalle	Skor
Pia	24200	Kalle	Skor

select * from Anställd where lön > 24000 and chef = 'Kalle'

Namn	Lön	Chef	Avd
Eva	25000	Kalle	Skor
Pelle	26000	Kalle	Skor
Pia	24200	Kalle	Skor

select * from Anställd where lön > 24000 or chef = 'Kalle'

Namn	Lön	Chef	Avd
Anna	26400	Lena	Mat
Bengt	23000	Lena	Mat
Bosse	23000	Kalle	Skor
Eva	25000	Kalle	Skor
Evert	24200	Lena	Mat
Jonas	44000	NULL	NULL
Kalle	36000	Jonas	Skor
Lena	36000	Jonas	Mat
Mia	24000	Kalle	Skor
Pelle	26000	Kalle	Skor
Pia	24200	Kalle	Skor

select * from Anställd where lön > 24000 and chef <> 'Kalle'

Namn	Lön	Chef	Avd
Anna	26400	Lena	Mat
Bengt	23000	Lena	Mat
Evert	24200	Lena	Mat
Jonas	44000	NULL	NULL
Kalle	36000	Jonas	Skor
Lena	36000	Jonas	Mat

Projektion

Ger endast viss(-a) kolumn(-er) i en tabell eller ett tabellvärt uttryck.

select Namn from Leverantör

select Lön from Anställd

Projektion + selektion

select Namn,Lön from Anställd where lön > 24000 and chef <> 'Kalle'

Villkor

Test på mängdtillhörighet

Vad tjänar chefen/cheferna på skoavdelningen?

select lön from Anställd where namn in
(select chef from Anställd where avd = 'skor')

Även kostanta mängder kan användas

select lön from Anställd where avd in
(select avd from Avdelning where våning in (1, 2))

Relationsvariabler

create view tmp as select * from Anställd where chef = 'Kalle';

create view resultat as select * from tmp where lön > 24000;

Namngivning

create view resultat(anst, månadslön) as
select namn, lön from Anställd;

Union

select * from Anställd where lön > 24000 and chef = 'Kalle'
union
select * from Anställd where lön > 24000 and chef <> 'Kalle'

Differens

select * from Anställd where chef = 'Kalle'
except
select * from Anställd where lön <= 24000;

Kartesisk produkt

select * from Anställd, Avdelning;

select * from Anställd, Avdelning where våning = 2;

select * from Anställd, Avdelning where Avdelning.våning = 2;

Skärning

select * from Anställd where chef = 'Kalle'
intersect
select * from Anställd where lön > 24000;

Join

Har jag redan visat (kartesisk produkt):

select * from Anställd, Avdelning where våning = 2;

select * from Anställd, Avdelning where Avdelning.våning = 2;

select * from Anställd, Avdelning where Anställd.avd <> Avdelning.avd;

select * from Anställd, Avdelning where Avdelning.avd <> Anställd.avd;

Join (SQL2)

select * from Anställd, Avdelning where Anställd.avd <> Avdelning.avd;
kan skrivas
select * from Anställd join Avdelning on Anställd.avd <> Avdelning.avd;

Naturlig join (SQL2)

select * from Anställd natural join Avdelning

Aggregat

SUM(attr), AVG(attr), MAX(attr), MIN(attr), COUNT(distinct attr | *)

Finn totala lönekostnaden för alla anställda
select SUM(lön) from Anställd; ***ignorera NULL***

Finn totala lönekostnaden för alla anställda på skoavdelningen
select SUM(lön) from Anställd where avd='skor'; ***ignorera NULL***

Finn totala lönekostnaden och antalet anställda på skoavdelningen
select SUM(lön), COUNT(*) from Anställd where avd='skor';

Partitionering

Finn totala lönekostnaden och antal anställda per avdelning
select avd, SUM(lön), COUNT(*) from Anställd group by avd;
ignorera NULL

create view result(avd, avglön) as
select avd, AVG(lön) from Anställd group by avd;

create view result(avglön) as select AVG(lön) from Anställd;

Hur använder man distinct ? order by ?

Ex.: Vilka varor säljs på andra våningen?
select varunr from Försäljning where avd in
(select avd from Avdelning where våning = 2); *= ut med allt på skärmen*
(låt dubletter vara med)

select distinct varunr from Försäljning where avd in
(select avd from Avdelning where våning = 2); *= ta bort dubletter*

select varunr from Försäljning where avd in
(select avd from Avdelning where våning = 2)
order by varunr; *= sortera utmatningen*

distinct? order by? . . .

select distinct varunr from Försäljning where avd in
(select avd from Avdelning where våning = 2)
order by varunr; *ta bort dubletter och sortera utmatningen*

create view tmp as select varunr from Försäljning where avd in
(select avd from Avdelning where våning = 2); *inga dubletter! osorterad*

create view tmp as select varunr from Försäljning where avd in
(select avd from Avdelning where våning = 2)
order by varunr; *inga dubletter, sorterad*

Inslag från annan matematik än mängdalgebra

tupelvariabler (variabler som kan ‘”hålla i” en tabellrad)

Dessa får förekomma på snart sagt alla platser.
Kallas ofta för ”alias” då de (ytligt sett) kan betraktas som ett alias för en tabell.
Tillåter bindningar mellan olika nivåer i nästlade frågor.

tupelvariabler . . .

Hitta alla anställda som har kalle till chef och tjänar mer än 24000 =
hitta alla anställda som har kalle till chef **men inte**
de som tjänar mindre än eller lika med 24000

select * from Anställd U where chef = 'Kalle' and not exists
(select * from Anställd where lön <= 24000 and namn = U.namn
and lön = U.lön and chef = U.chef and avd = U.avd)

Skärning

select * from Anställd U where chef = 'Kalle' and exists
(select * from Anställd where lön > 24000 and namn = U.namn
and lön = U.lön and chef = U.chef and avd = U.avd)

tupelvariabler . . .

Frågan:

vilka företag leverar alla varor som finns till försäljning?

kan skrivas om till:

för vilka företag gäller att det **inte** ***finns varor som företaget*** **inte** ***levererar?***

Med hjälp av bindningar via tupelvariabler kan vi med nästling och dubbel negation få fram ett svar

select företag from Leverantör L where not exists
(select * from Vara V where not exists
(select * from Lager where L.företag = företag and V.varunr = varunr))

Jämförelse mellan element och mängder

I SQL tar man endast den aktiva domänen i beaktande.

select namn from Anställd L where lön > all
(select lön from Anställd where avd = 'fisk');

select namn from Anställd L where lön > any
(select lön from Anställd where avd = 'fisk');

Textmatchning

Like och Matches kan användas vid matchning av texter
select namn from Anställd L where namn like "A%"

select namn from Anställd L where namn matches "A??"

Like:	**%** ***följd av tecken***
	_ ***ett tecken***
	**** ***speciell mening med nästa tecken***
Matches:	***** ***följd av tecken***
	? ***ett tecken***
	[ ] ***ett tecken ut ett intervall***
	**** ***speciell mening med nästa tecken***

Normalisering

- **En databas konstrueras med en viss avsikt.**
- **Dess innehåll måste *naturligtvis* tolkas med samma avsikt.**
- **Det mest centrala är att skapa ett schema som avspeglar en ”uppenbar” avsikt och en ”lika uppenbar” tolkning.**
- **Ju bättre ett schema är konstruerat, desto enklare är det att förstå dess semantik.**

Varuhuset, databasstruktur

Anställd	**(namn, lön, *chef*, *avd*)**
Försäljning	**(*avd*, *varunr*, volym)**
Leverantör	**(företag, adress)**
Lager	**(*företag*, *avd*, *varunr*, volym)**
Avdelning	**(avd, våning)**
Vara	**(varunr, typ)**

Understrukna attribut utgör *tillsammans* primärnycklar i respektive relation.

Attribut i *kursiv stil* är *var för sig* främmande nycklar (ägda attribut), attribut som utgör primärnyckel i någon basrelation.

Normalisering . . .

Komplexa modeller blir lätt oöverskådliga och även i relativt små modeller händer det att man inte lyckas representera alla restriktioner som finns i den verklighet man vill avbilda i databasen.

Det leder till att den valda representationen kommer att tillåta lagring av redundanta data.

Normaliseringsprocessen har som mål att minimera redundansen, som ger upphov till:

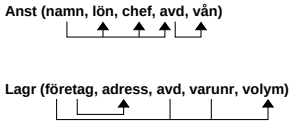
Normalisering . . .

Insättningsproblem. **När data sätts in måste de kanske sättas in i flera basrelationer (och tupler). Man måste identifiera alla platser för representation av den nya informationen.**

Uppdateringsproblem. **När data uppdateras måste de kanske uppdateras i flera basrelationer. Man måste identifiera alla platser där informationen redan är representerad.**

Borttagningsproblem. **När data tas bort måste de kanske tas bort i flera basrelationer (och tupler). Man måste identifiera alla platser där informationen redan är representerad och alla platser där existerande information inte längre är relevant.**

Normalisering . . .



Mera lagringsutrymme går åt eftersom värdet för *våning* upprepas varje gång samma värde för avd förekommer.

Mera lagringsutrymme går åt eftersom värdet för *adress* upprepas varje gång samma värde för företag förekommer.

Normalisering . . .

Uppdateringsproblem.

Om ett företag flyttar måste vi gå igenom hela tabellen *Lagr*

Insättningsproblem.

a. **Om en ny person anställs måste vi sätta null på både *avd* och *vån* i det fall avdelningstillhörighet inte bestämts ännu.**
b. **Om en ny avdelning skapas utan någon anställd måste värdet för alla termer utom *avd* och *vån* sättas till null.**

Normalisering . . .

Borttagningsproblem.

a. **Om en anställd som är ensam på en avdelning slutar så försvinner all information om avdelningen.**
b. **Om ett företag just nu inte har något lager på någon avdelning så har vi ingen information om företaget.**

Problem med nullvärden.

Om ett värde är null är det inte helt säkert hur vi skall tolka det.

***a*. Det ska finnas ett värde men det aktuella värdet är inte känt**
***b*. Det finns ett värde men det har inte förts in ännu**
***c*. Det finns inget relevant värde i aktuell situation**

Tre regler

1. **Design av basrelationsscheman skall väl avspegla den verklighet databasen avses motsvara. Varje schema skall vara lätt att förklara.**

2. **Design av basrelationsscheman skall göras så att inga uppdaterings-, borttagnings- eller insättningsproblem uppstår.**

3. **Försök, så mycket som möjligt, att placera attribut så att nullvärden inte behöver lagras.**

Ibland måste man göra våld på reglerna och då måste man noga bokföra sina avsteg så att man kan kompensera för dem.

Tre regler . . .

Det låter enkelt och låter sig kanske göras i små databaser, men i de riktigt stora? De med hundratals tabeller och ett femtiotal attribut per tabell i snitt? Hur håller man tungan rätt i munnen då?

Det behövs någon formell metod!

I fallen med *Anst* och *Lagr* kan man se två typer av problem – i *Anst* berodde problemet på ett beroende via en annan egenskap och i *Lagr* berodde ett attribut endast av en del av primärnyckeln.

Vi behöver något för att beskriva beroenden och regler för att bryta ner relationer i flera relationer.

Några definitioner

Att relationen $\mathcal{R}$ har schemat $\mathcal{X}$ skriver vi $\mathcal{R}(\mathcal{X})$.

En tupel i $\mathcal{R}$ har samma schema som $\mathcal{R}$, men vi kan projicera ut en del av en tupel genom t.ex. $t[\mathcal{Y}]$ om $\mathcal{Y} \subseteq \mathcal{X}$ och får då endast den del av tupeln som har schemat $\mathcal{Y}$.

OBS att attributens faktiska inbördes ordning inte har någon betydelse.

OBS också att man ofta förkortar $\mathcal{X} \cup \mathcal{Y}$ till $\mathcal{XY}$

Funktionellt berooende (FD)

I den sammansatta basrelationen Anställd (namn, lön, chef, avd, vån) förväntar vi oss att ett visst värde på *avd* alltid ger samma värde på *vån*. Detta kallar vi ett funktionellt beroende – funktionellt därför att det fungerar som en funktion gör.

Formellt definieras funktionellt beroende så:

Låt $\mathcal{X}$ vara schemat för en relation $\mathcal{R}$ och låt $\mathcal{Y} \subseteq \mathcal{X}$ och $\mathcal{Z} \subseteq \mathcal{X}$ vara attributmängder. $\mathcal{Y} \rightarrow \mathcal{Z}$ ($\mathcal{Z}$ är *funktionellt beroende* av $\mathcal{Y}$) om det för godtyckliga tupler t_1 och t_2 sådana att $t_1[\mathcal{Y}] = t_2[\mathcal{Y}]$ alltid gäller att $t_1[\mathcal{Z}] = t_2[\mathcal{Z}]$.

Fullständigt funktionellt beroende (FFD)

I basrelationen Lager(företag, avd, varunr, volym) gäller att (företag, avd, varunr) $\rightarrow$ volym men även att (företag, avd) $\not\rightarrow$ volym, (företag, varunr) $\not\rightarrow$ volym och (avd, varunr) $\not\rightarrow$ volym.

Då man vet att ett visst FD $\mathcal{X} \rightarrow \mathcal{Y}$ gäller samt att det *inte finns* någon äkta delmängd $\mathcal{Z} \subset \mathcal{X}$ sådant att $\mathcal{Z} \rightarrow \mathcal{Y}$ säges $\mathcal{Y}$ vara fullständigt funktionellt beroende (FFD) av $\mathcal{X}$ eller $\mathcal{X} \mapsto \mathcal{Y}$.

I exemplet ovan är volym FFD av (företag, avd, varunr) eller (företag, avd, varunr) $\mapsto$ volym.

Funktionellt beroende ...

- För varje attributmängd $\mathcal{X}$ gäller alltid att $\mathcal{X} \rightarrow \mathcal{Y}$ om $\mathcal{Y} \subseteq \mathcal{X}$. Detta kallas för trivialt beroende.
- Varje minimal attributmängd $\mathcal{X}$, sådan att det finns ett icketrivialt funktionellt beroende $\mathcal{X} \rightarrow \mathcal{Y}$ kallas för en *determinant*.

Nycklar och primattribut

- Varje mängd av attribut som entydigt bestämmer en tupel i en basrelation kallas för en *supernyckel*.
- Varje supernyckel som inte kan reduceras och fortfarande entydigt kan bestämma en tupel kallas för en *kandidatnyckel*.
- En av kandidatnycklarna väljs att bli *primärnyckel*. *OBS* att detta är ett subjektivt val. Ofta väljer man enligt något minimalitetskriterium, t.ex. minst antal attribut, kortast, eller liknande.
- Ett attribut som ingår i en kandidatnyckel kallas för *primattribut* eller *primt attribut*. Övriga attribut kallas ickeprima.

Regler för funktionella beroenden

Det finns ett antal regler som man kan använda för att dela upp relationer så att man undviker redundans.

$\mathcal{X} \rightarrow \mathcal{X}$	(Reflexivitet)
Om $\mathcal{X} \rightarrow \mathcal{YZ}$ och $\mathcal{Z} \rightarrow \mathcal{W}$ så $\mathcal{X} \rightarrow \mathcal{YZW}$	(Akumulering)
Om $\mathcal{X} \rightarrow \mathcal{YZ}$ så $\mathcal{X} \rightarrow \mathcal{Y}$ och $\mathcal{X} \rightarrow \mathcal{Z}$	(Projektion)

Transitiva höljen

Reglerna kan användas för att undersöka vilka attribut som bestäms av en viss attributmängd i en given omgivning.

Antag $\mathcal{F} = \{\mathcal{AB} \rightarrow \mathcal{E}, \mathcal{AG} \rightarrow \mathcal{J}, \mathcal{BE} \rightarrow \mathcal{I}, \mathcal{E} \rightarrow \mathcal{G}, \mathcal{GI} \rightarrow \mathcal{H}\}$

Uppgift: Hitta alla attribut som bestäms av $\mathcal{AB}$

1. Börja alltid med första regeln: $\mathcal{AB} \rightarrow \mathcal{AB}$
2. Upprepade gånger används andra regeln för att, med hjälp av de funktionella beroendena, utvidga högerledet tills inga fler attribut kan inkluderas.
   - $\mathcal{AB} \rightarrow \mathcal{AB}$, $\mathcal{AB} \rightarrow \mathcal{E}$ $\Rightarrow \mathcal{AB} \rightarrow \mathcal{ABE}$
   - $\mathcal{AB} \rightarrow \mathcal{ABE}$, $\mathcal{BE} \rightarrow \mathcal{I}$ $\Rightarrow \mathcal{AB} \rightarrow \mathcal{ABEI}$
   - $\mathcal{AB} \rightarrow \mathcal{ABEI}$, $\mathcal{E} \rightarrow \mathcal{G}$ $\Rightarrow \mathcal{AB} \rightarrow \mathcal{ABEIG}$
   - $\mathcal{AB} \rightarrow \mathcal{ABEIG}$, $\mathcal{AG} \rightarrow \mathcal{J}$ $\Rightarrow \mathcal{AB} \rightarrow \mathcal{ABEIGJ}$
   - $\mathcal{AB} \rightarrow \mathcal{ABEIGJ}$, $\mathcal{GI} \rightarrow \mathcal{H}$ $\Rightarrow \mathcal{AB} \rightarrow \mathcal{ABEIGJH}$

Transitiva höljen ...

Det vi nu erhållit ($\mathcal{ABEIGJH}$) kallas *det transitiva höljet* till $\mathcal{AB}$.

3. Avsluta med regel tre, som ger $\mathcal{AB} \rightarrow \mathcal{ABEIGH} \Rightarrow \mathcal{AB} \rightarrow \mathcal{GH}$
   Alltså gäller $\mathcal{AB} \rightarrow \mathcal{GH}$.

Transitiva höljen är användbara för att hitta kandidatnycklar. Vi behöver känna till *alla* kandidatnycklar.

I exemplet kan vi se att $\mathcal{AB}$ är en supernyckel eftersom alla attribut inkluderats. Men är det en kandidatnyckel?

Normalformer – 1NF

En basrelation är i första normalform (1NF) omm alla attributvärden är atomära (omm för varje tupel gäller att den innehåller exakt *ett* värde för varje attribut).

Annars:
Hitta alla kandidatnycklar genom att beräkna alla transitiva höljen. Välj en till primärnyckel.

Normalformer – 1NF ...

Djurägare

Namn	Djur
Kalle	Hund
	Katt
Anna	Katt
Johan	Kanin
	Mus

$\Longrightarrow$

Djurägare

Namn
Kalle
Anna
Johan

Har-djur

Namn	Djur
Kalle	Hund
Kalle	Katt
Anna	Katt
Johan	Kanin
Johan	Mus

Husdjur

Djur
Hund
Kanin
Katt
Mus

Normalformer – 2NF

En basrelation är i 2NF omm den är i 1NF och varje ickeprimt attribut är fullständigt funktionellt beroende av varje kandidatnyckel.

Skilj ut varje ickeprimt attribut som är funktionellt beroende av endast en del av *någon* nyckel och gör dem till en egen relation som får som i-term den del av nyckeln, i den ursprungliga relationen, som de är funktionellt beroende av.

Lagr(företag, avd, varunr, adress, volym)
$\Downarrow$
Lager(företag, avd, varunr, volym)
Leverantör(företag, adress)

Normalformer – 3NF

En basrelation är i 3NF omm det för varje icketrivialt funktionellt beroende $\mathcal{X} \rightarrow \mathcal{Y}$ gäller att dess determinant är en supernyckel eller att $\mathcal{Y}$ är ett primattribut. (d.v.s att man har 3NF omm man har 2NF och det inte finns några transitiva beroenden $\mathcal{X} \rightarrow \mathcal{Y} \rightarrow \mathcal{Z}$ där $\mathcal{Y}$ inte är ett primattribut).

Annars: Skilj ut varje icke-primt attribut som är transitivt beroende av nyckeln via något annat icke-primt attribut och låt dem bilda egna basrelationer där nyckeln är det/de attribut de beror av.

Anst(namn, lön, chef, avd, vån)
⇓
Anställd(namn, lön, chef, avd)
Avdelning(avd, vån)

Normalisering . . .

Detta brukar räcka om det paras med lite sunt förnuft (som man ju får med erfarenhet). Det finns algoritmer för att utföra normaliseringen, men man måste förstå vad de gör.

Man kan råka ut för fall då det här inte räcker och blir naturligtvis tvungen att studera högre normalformer för att klara av de problemen.

Varuhuset

Varuhus (avd, namn, lön, chef, företag, f_adress, varunr, typ, l_volym, f_volym, våning)

fd:

avd ⟶ våning
namn ⟶ lön, chef, avd
företag ⟶ f_adress
varunr ⟶ typ
avd, varunr ⟶ f_volym
avd, företag, varunr ⟶ l_volym

Varuhuset . . .

transitiva höljen:

{avd}$^+$ = {avd, våning}
{namn}$^+$ = {namn, lön, chef, avd, våning}
{företag}$^+$ = {företag, f_adress}
{varunr}$^+$ = {varunr, typ}
{avd, varunr}$^+$ = {avd, varunr, våning, typ, f_volym}
{avd, företag, varunr}$^+$ = {avd, företag, varunr, våning, f_adress, typ, f_volym, l_volym}

primärnyckel:

{namn, företag, varunr}$^+$ =
{namn, lön, chef, avd, våning, företag, f_adress, varunr, typ, f_volym, l_volym}

det finns inte några andra kandidatnycklar (alla kombinationer av attribut måste undersökas).

Varuhuset . . .

0NF → 1NF

(namn, företag, varunr, lön, chef, avd, våning, f_adress, typ, f_volym, l_volym)

För att komma till 2NF skall alla ickeprima attribut vara FFD av primärnyckeln:

{namn}$^+$ = {namn, lön, chef, avd, våning} vilket ger
(namn, lön, chef, avd, våning)
{företag}$^+$ = {företag, f_adress} vilket ger
(företag, f_adress)
{varunr}$^+$ = {varunr, typ} vilket ger
(varunr, typ)
{namn, varunr}$^+$ = {namn, lön, chef, avd, våning, varunr, typ, f_volym} vilket ger
(namn, varunr, f_volym)
Kvar i den ursprungliga relationen blir endast, förutom nyckeln, l_volym:
(namn, företag, varunr, l_volym)

Varuhuset . . .

→ 2NF

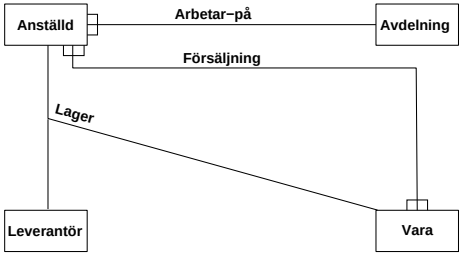
Anställd (namn, lön, chef, avd, våning)
Leverantör (företag, f_adress)
Vara (varunr, typ)
Lager (namn, företag, varunr, l_volym)
Försäljning (namn, varunr, f_volym)
→ 3NF

Anställd (namn, lön, chef, avd, våning)
är inte i 3NF eftersom namn → avd, avd → våning och avd är inte ett primattribut.

Anställd (namn, lön, chef, avd, våning)
ersätts av
Anställd (namn, lön, chef, avd) och
Avdelning (avd, våning)

Varuhuset . . .

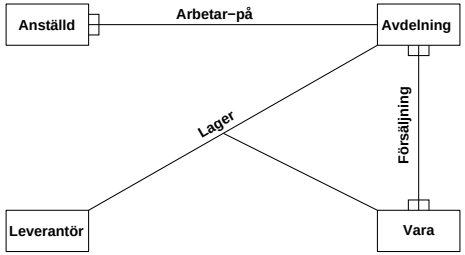
Rita modell:



Vi bestämmer att en anställd inte har ett lager av varor han säljer det är avdelningen där den anställde arbetar som har ett lager.

Varuhuset . . .

Rita om modellen:



Överför på nytt till databasstruktur (Arbetar_på försvinner och avd blir e_term i Anställd):

Varuhuset . . .

Anställd	**(namn, lön, chef, avd)**
Avdelning	**(avd, våning)**
Försäljning	**(avd, varunr, f_volym)**
Leverantör	**(företag, f_adress)**
Vara	**(varunr, typ)**
Lager	**(avd, företag, varunr, l_volym)**

Kontrollera att strukturen är i 3NF och om inte så upprepa hela proceduren!