INF1300

PENSUM

H2012

Pensum fra forelesnings-foilere

Joakim
Myrvoll
Johansen

Innhold

DATA

- Transiente data → data som slettes når programmet avslutte
- Persistente data → data som overlever mellom to programkjøringer (skrives til disk)
- Informasjon → data og regler for hvordan data skal tolkes

DATABASE OG DBMS

- Database → samling persistente data som fysisk bare kan aksesseres fra ett spesialprogram, kalt et *databasehåndteringssystem*, forkortet DBMS (DataBase Management System)
- Et slik program fungerer som en server, og vilkårlig mange klienter (programmer) kan aksessere dataene samtidig.

TRANSAKSJONER

- Transaksjon → en henvendelse fra en klient til DBMSet
- En transaksjon som bare leser data, kalles en lesetransaksjon ellen en spørring (query)
- En transaksjon som legger til ny data eller forandrer eksisterende data, kalles en skrivetransaksjon

INFORMASJONSMODELLER

- En fullstendig beskrivelse av interesseområdet (UoD = Universe of Discourse) kalles en informasjonsmodell
- Informasjonsmodellen uttrykkes gjerne i et modellspråk, som f.eks;
    - UML (Unified Modelling Languages)
    - ORM (Object-Role Modelling)
    - ER (Entity Relationship)

SKRANKER

- Forretningsregler → lovene som styrer virkeligheten
- Skranker → beskrivelsen av forretningsreglene
    - Statiske:
      beskriver begrensninger på mulige tilstander i interesseområdet
      (eks: en ferdig ORM-modell)
    - Dynamiske:
      beskriver begrensinger på mulige forandringer i interessemrådet

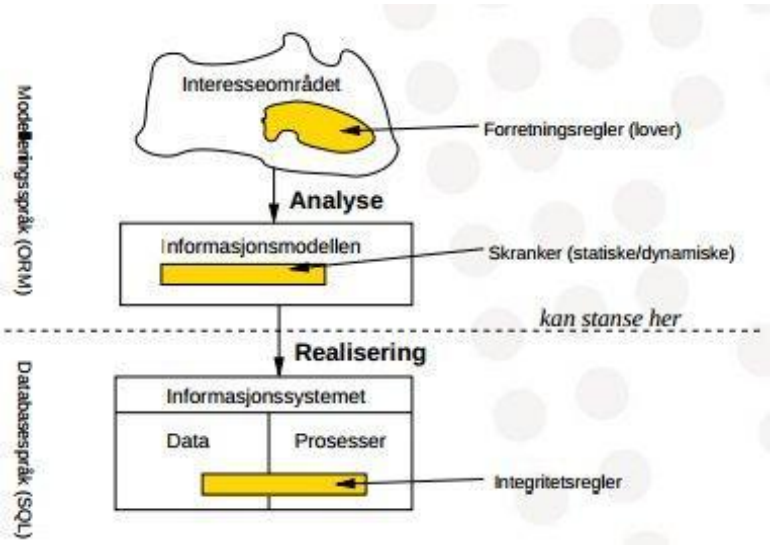
DET BEGREPSMESSIGE SKJEMA

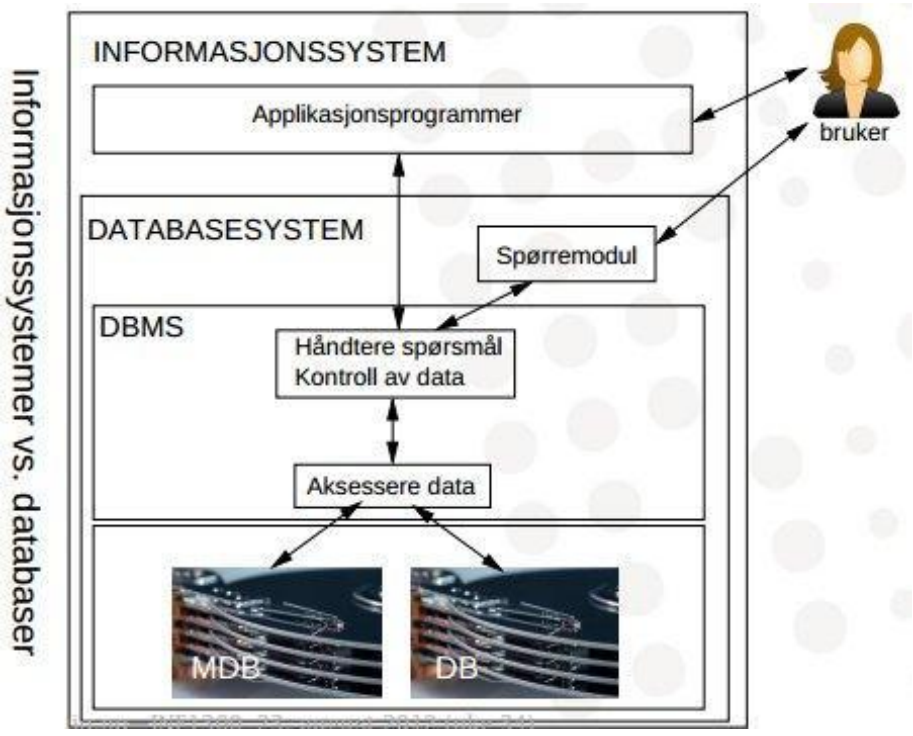
- Informasjonsmodellen brukt som regelverk (preskripsjon) for hvordan informasjonssystemet skal oppføre seg, kalles det begrepsmessige skjema
- Det begrepsmessige skjema uttrykkes i et språk som paser for den databaseteknologien vi skal bruke, f.eks;
    - SQL (Structured Query Language) for relasjonsdatabaser
    - ODL (Object Definition Language) for objektdatabaser

INTEGRITETSREGLER

- I det begrepsmessige skjemaet kaller vi skrankene for integritetsregler
- Integritetsreglene bestemmer
    - hva som er lovlig å lagre i informasjons-systemet
      (lovlige tilstander i databasen)
    - hva som er lovlige forandringer (lovlige transisjoner/transaksjoner)
- Det er umulig for en bruker av informasjons-systemet å gjøre noe som strider mot integritetsreglene

INFORMASJONSSYSTEMER





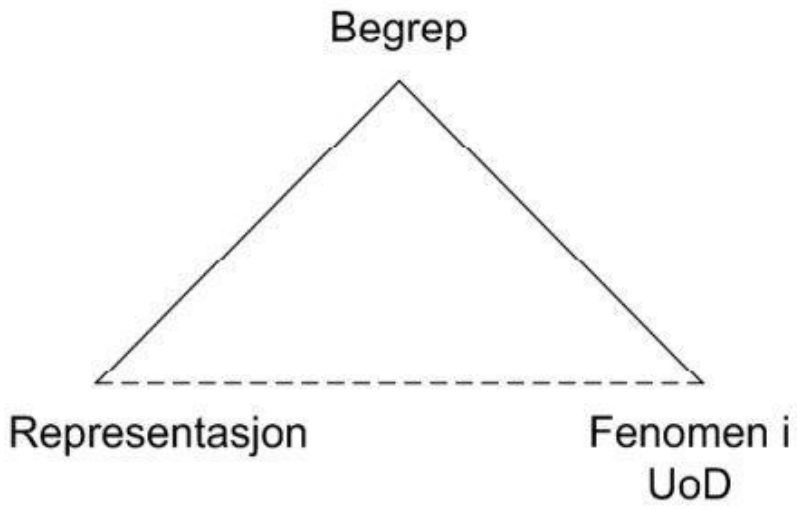
ORM - Object Role Modelling

REPRESENTASJON

- Vi kan ikke lagre f.eks. kyr i databasen, så vi (eller bonden som skal registrere melkeproduksjon) må finne en måte å lagre informasjon som identifiserer hver enkelt ku
- En slik identifikasjonsmåte kalles en *representasjon* eller *identifikator* for begrepet

ONTOLOGI

- Ontologi → vitenskapen om sammenhengen mellom virkelighet (UoD), begreper og representasjon
- Hovedpoenget med ontologi betegnes gjerne som Ogdens trekant



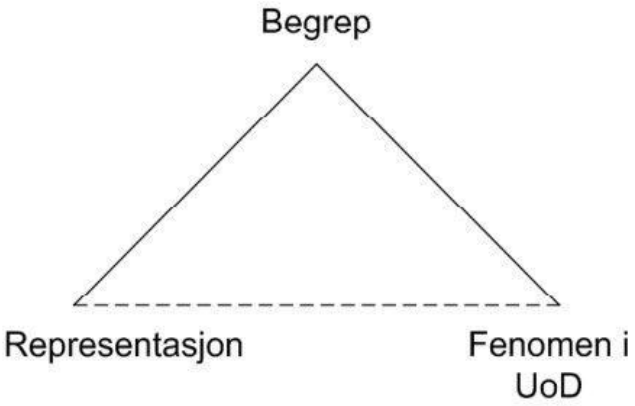
ELEMENTÆRE SETNINGER

- En setning som ikke kan deles opp uten å miste meningsinnhold, kalles elementær
    - Eksempel:
      – Per liker is og brus

  Denne Denne setningen er ikke elementær fordi den setningen er ikke elementær fordi den kan erstattes av de to elementære setningene
    - Per liker brus
    - Per liker is

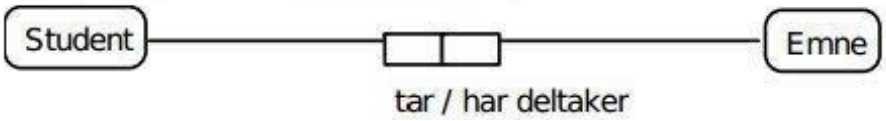
SETNINGER OG OGDENS TREKANT

- Se på setningen «Hanne tar INF1000»
- Det er mye informasjon som er underforstått:
    - Hanne er navn på en student
    - INF1000 er en emnekode
- Det vi mener er at;
  "*Studenten* med *navn* Hanne tar *emnet* med *emnekode* INF1000"
- «student» og «emne» er begreper
- «navn» og «emnekode» er deres representasjoner
- «Hanne» og «INF1000» er forekomster (data)



SETNINGER OG FAKTATYPER

- De to setningene under har samme meningsinnhold:
    - «Studenten med navn Hanne tar emnet med emnekode INF1000»
    - «Emnet med emnekode INF1000 har deltaker studenten med navn Hanne»
- Vi kan forme liknende fakta ved å bytte ut forekomstene:
    - «Studenten med navn Henrik tar emnet med emnekode INF1100»
      (eller: «Emnet med emnekode INF1100 har deltaker studenten med navn Henrik»)
- I ORM tegner vi denne faktatypen slik:



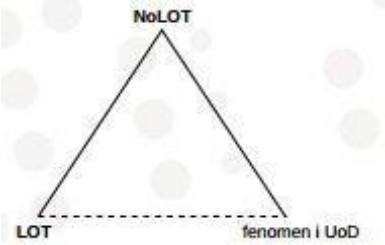
OGDENS TREKANT OG ORM

- I ORM tegner vi begreper og representasjoner som hhv. heltrukne og stiplede rektangler med runde hjørner
    - Eksempel:
      Begrepet Person og representasjonen Fødselsnummer tegnes slik:

- I ORM brukes ordet objekttyper som en felles betegnelse på begreper og representasjoner

ORM VS stORM

- Verktøyet stORM bruker en gammel ORM-dialekt
  (NIAM–Natural language Information Analysis Method)
- I stORM brukes sirkler i stedet for rektangler
- Et *begrep* kalles en **NoLOT**
  (Non Lexical Object Type)
- En *representasjon* kalles en **LOT**
  (Lexical Object Type)



SETNINGERS ARITET

- Den elementære setningen:
  Studenten med navn Anne fikk i emnet med emnekode INF1010 resultatet karakteren B
- Inneholder tre begreper: student, emne og resultat
- Antall begreper i en setning kalles setningens aritet
  (I dette eksempelet er ariteten 3)
- Setninger med aritet 1 kaller vi unære
- Setninger med aritet 2 kaller vi binære
- Setninger med aritet 3 kaller vi ternære
- Man kan konstruere elementære setninger med vilkårlig høy aritet
- Elementære setninger med aritet > 3 er sjeldne, så vi gir dem ikke egne navn (n-ære setninger)

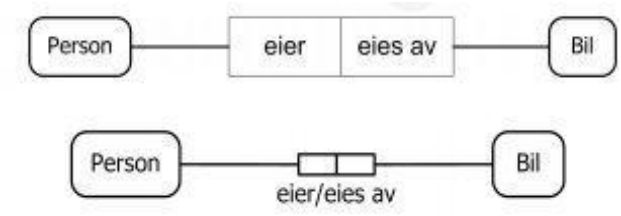
UNÆRE SETNINGER

- Disse er vanlige i dagligtalen, men sjeldne i en informasjonsmodell
- Eksempel: En person med navn Siri sover
- En unær setning kan alltid erstattes av en binær setning hvor ett av begrepene har boolsk representasjon
- Eksempel: En person med navn Siri har sovestatus med boolsk verdi true
- Unære setninger er tillatt i ORM, men ikke i modelleringsverktøyet stORM

BINÆRE SETNINGER

- Disse er vanlige i dagligtalen og enda vanligere i en informasjonsmodell
- Eksempel: En person med navn Siri eier en bil med reg.nr. DL12345
- Binære (og unære) setninger er alltid elementære
- En ORM informasjonsmodell baserer seg nesten i sin helhet på binære setninger

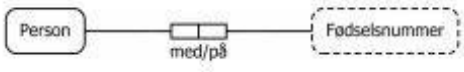
FAKTATYPER I ORM



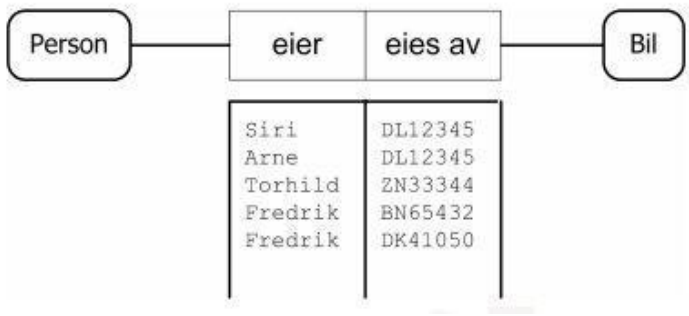
- Et eksempel på en **faktatype** mellom begrepene Person og Bil tegnes slik i ORM1 øverst, en mer kompakt skrivemåte ORM2 nederst.

BROER

- En **bro** er en forbindelse mellom et begrep og en representasjon

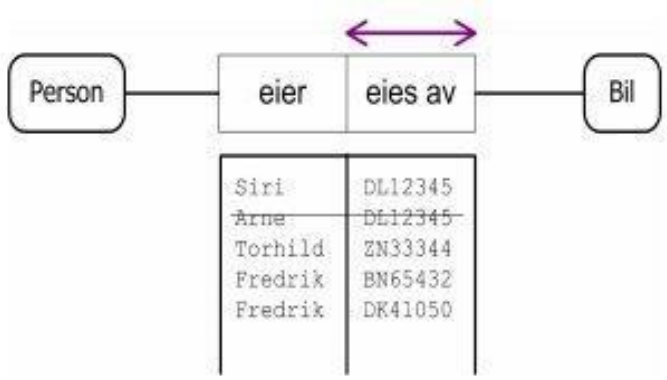


FOREKOMSTTABELL



- Legg merke til at en faktatype med forekomsttabell gir oss like mange fakta som det er linjer i forekomsttabellen

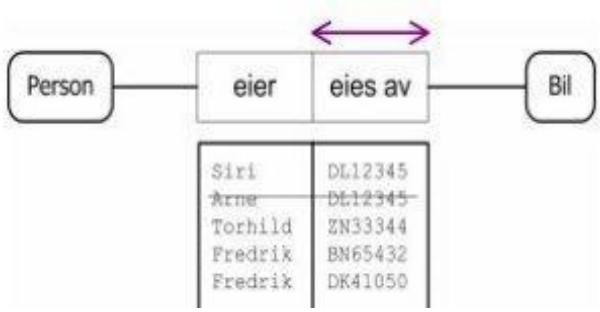
ENTYDIGHETSSKRANKER



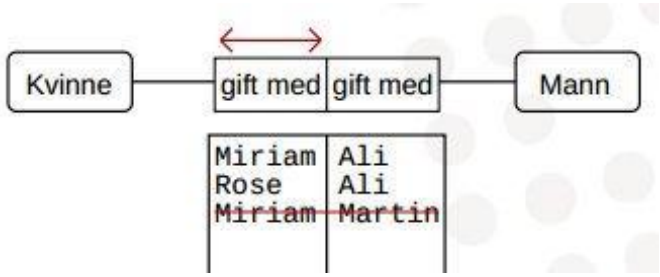
- I ORM-diagrammet setter vi en pil med spiss i begge ender over rollen hvor samme forekomst ikke kan gjentas i forekomsttabellen
- Pilen kalles en entydighetsskranke
- Entydighetsskranker kan gå over flere roller—da er det forekomstkombinasjonen i rollene som ikke kan gjentas

BRUK

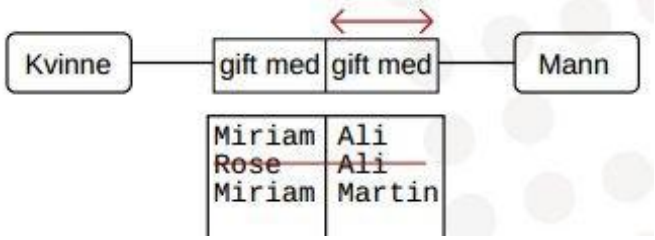
- 1:1



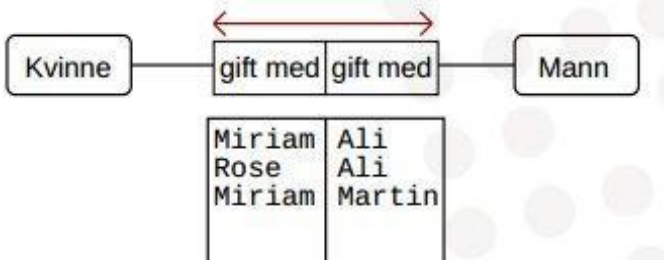
- n:1



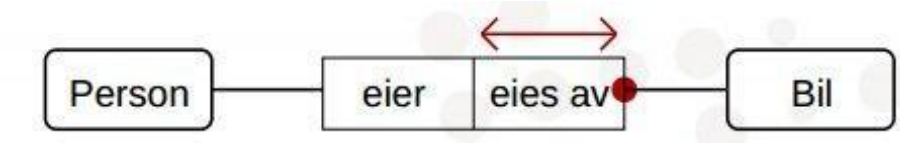
- 1:n



- m:n



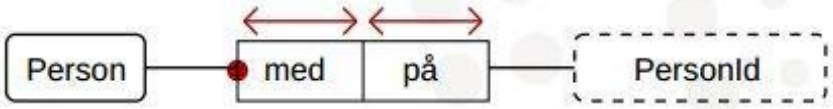
TOTALE ROLLER



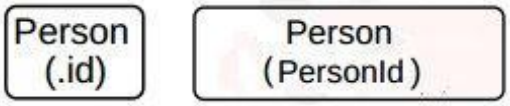
- Dersom alle biler har en eier, sier matematikerne at vi har en total funksjon fra Bil til Person (den er definert for alle forekomster av Bil)
- Vi sier at rollen eies av er en total rolle for Bil og markerer det med en kule (liten fylt sirkel) på rollen
- Merk: Det at rollen er total, gjør at hver gang vi legger inn en bil i databasen, må vi samtidig registrere hvem som eier bilen

PERFEKT BRO

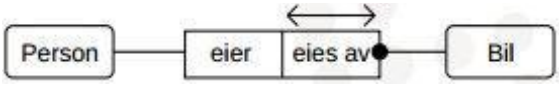
- En 1:1 bro der begrepsrollen er total, kalles en **perfekt bro**



- Kortform (ekvivalent):



FUNKSJONELLE AVHENGIGHETER (FD)

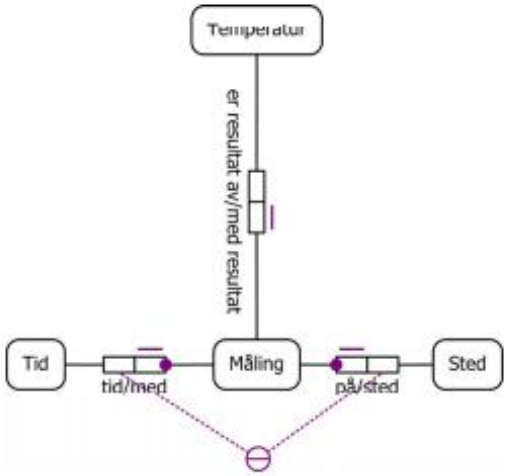


- Det at en bil alltid har én, og bare én, eier, gjør at hvis vi vet hvilken bil vi har (dvs. at vi kjenner bilens reg.nr), så vet vi også hvilken person som eier bilen
- Faktatypen fra Bil til Person definerer altså en funksjon fra forekomstene av Bil til forekomstene av Person som til en gitt bil gir oss eieren
- Vi sier at Person er funksjonelt avhengig av Bil, eller at vi har en FD (Functional Dependency) fra Bil til Person

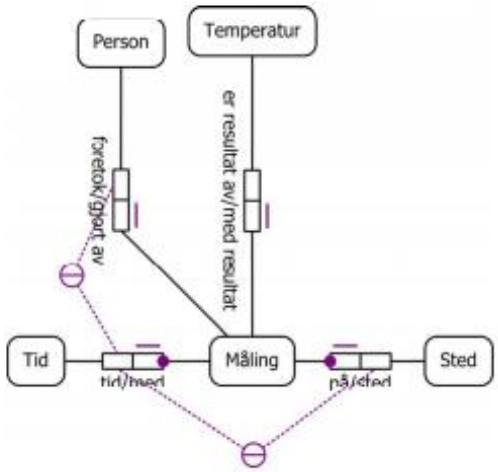
BEGREPSDANNELSE

- n-1-regelen:
    - En setning med aritet n er aldri elementær hvis det tilhørende forekomstdiagrammet har en entydighetsskranke som er kortere enn n-1
    - Hvis korteste entydighetsskranke har lengde n-1, er setningen elementær
    - Hvis korteste entydighetsskranke har lengde n, er setningen nesten alltid elementær
    - Setninger med aritet 1 eller 2 er alltid elementære
- Alle entydighetsskranker som som går over mer enn går over mer enn én rolle i en faktatype, skjuler et (nytt) begrep. Man skal alltid vurdere om det skal lages nye begreper når man får faktatyper med lange entydighetspiler.
- En faktatype med aritet 3 eller 4 (eller mer) kan gjøres om til binære setninger ved å lage ett eller eller flere nye begreper flere nye begreper.

EKSTERNE ENTYDIGHETSSKRANKER



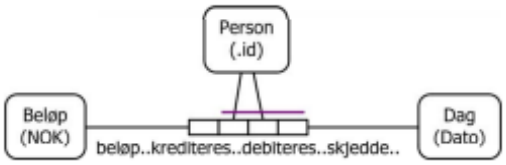
- Entydighet på tvers av faktatyper indikerer vi med en ekstern entydighetsskranke på de involverte rollene (her: tid og sted)



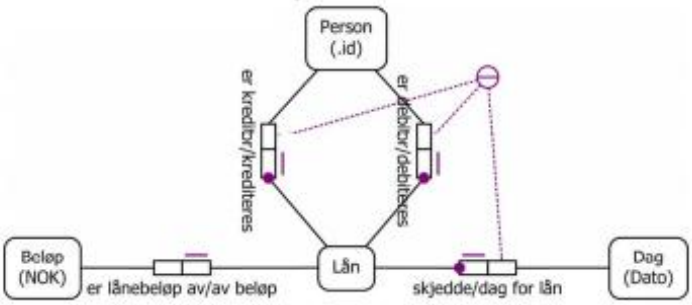
- Vi kan også uttrykke at en person ikke kan foreta mer enn én måling av gangen
- Dette gjøres med en ekstern entydighetsskranke på rollene foretok og tid

EKSTERNE ENTYDIGHETSSKRANKER FOR ARITET 4

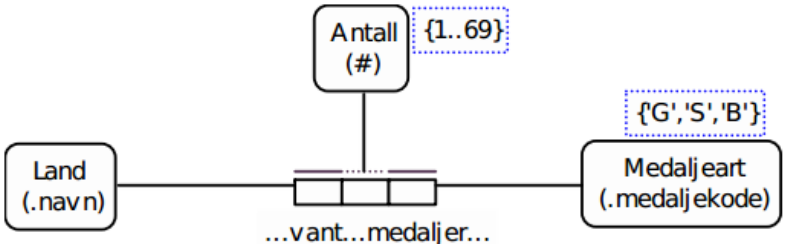
- På en gitt dag låner en person, kalt debitor, et beløp fra en annen person, kalt kreditor



- Det nye begrepet, *Lån*, består av én dag og to personer



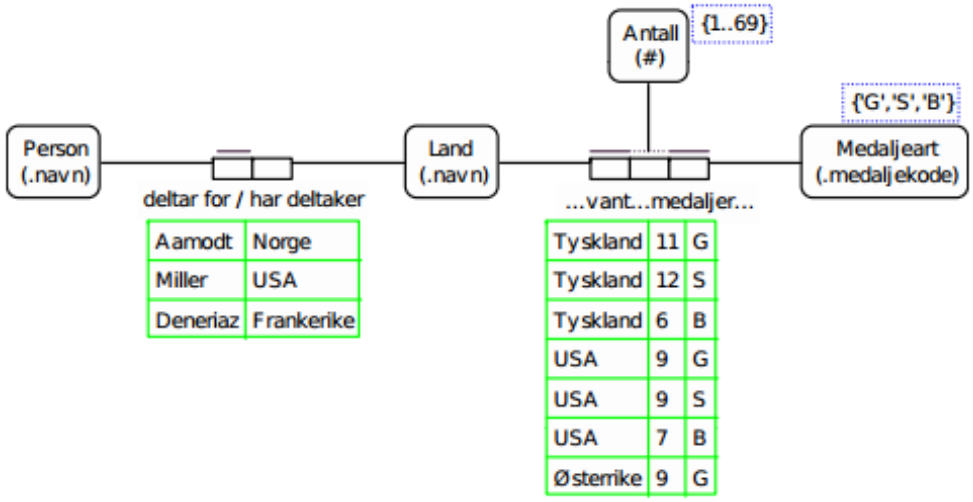
VERDISKRANKER



- Begrenser mulige forekomster av et begrep
- I praksis: Angir en mengde verdier som er lovlige representasjoner – f.eks. ved direkte opprams av verdiene, angivelse av en nedre og/eller øvre grense, innebygygde kontrollsifre (eks. fødselsnumre)

POPULASJONER

- **Populasjon for en rolle:**
  Hvis *r* er en rolle, betegner *pop(r)* mengden av forekomster i kolonnen for *r* i forekomsttabellen
- **Populasjon for et begrep:**
  Begreper har egentlig ikke forekomster løsrevet fra roller, men vi definerer likevel populasjonen til et begrep A som har roller r_1, r_2, ..., r_n ved
  $$pop(A) = pop(r_1) \cup pop(r_2) \cup ... \cup pop(r_n)$$
- **Merk:**
  Populasjonen til en rolle/et begrep varierer med tiden
- **Eksempel:**



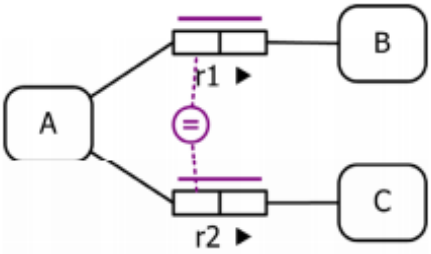
pop(Land **som** vant) = {Tyskland, USA, Østerrike}
pop(Land **som** har_deltaker) = {Norge, USA, Frankrike}
pop(Land) = {Tyskland, USA, Østerrike, Norge, Frankrike}

MENGDESKRANKER

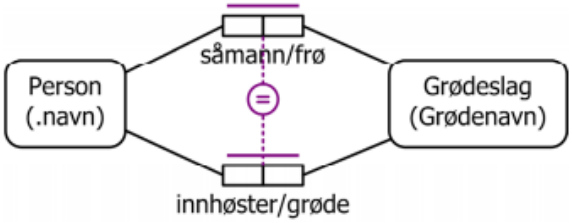
- Mengdeskrankene begrenser mengden av forekomster i en eller flere roller i forhold til forekomstene i andre roller
- Finnes i følgende varianter:
    - Likhetsskranke
    - Ulikhetsskranke
    - Delmengdeskranke

LIKHETSSKRANKE

- A skal ha rollen r_1 hvis og bare hvis A har rollen r_2.
  $pop(r_1) = pop(r_2)$ for alle tilstander

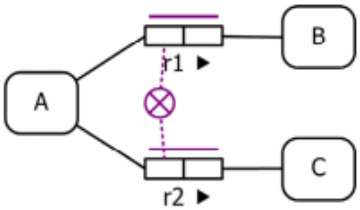


- Likhetsskranke over flere roller er vanligvis ikke lov

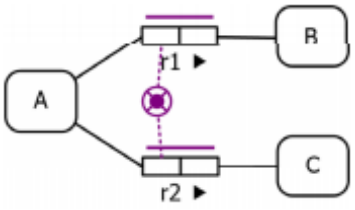


ULIKHETSSKRANKE

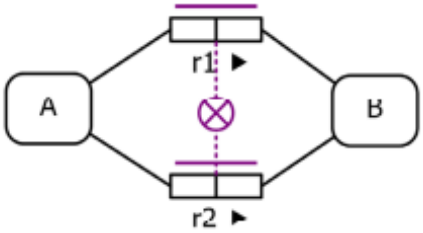
- A skal ikke ha både rollen r_1 og r_2.
  $pop(r_1) \cap pop(r_2) = \emptyset$ for alle tilstander
- A skal ikke ha både rollen r_1 og r_2. Det kan være forekomster av A som verken er i r_1 eller r_2



- A skal ha en og bare en av rollene r_1 og r_2



- Det skal ikke være forekomster av A og B som er relatert gjennom begge faktatypene



  - Eksempel:

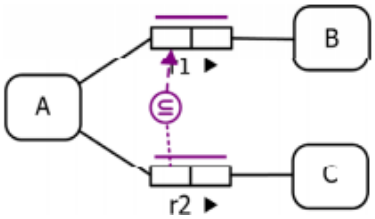
"Sag ikke av en gren når du sitter på en annen gren"

"Sag ikke av en gren som noen andre sitter på"

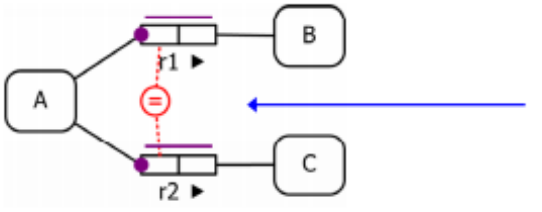
"Sag ikke av den grenen du selv sitter på"

DELMENGDESKRANKE

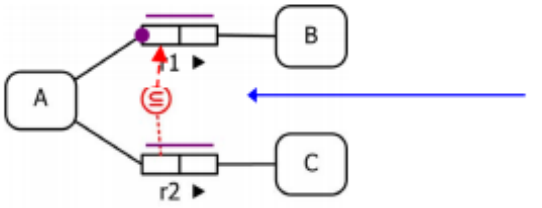
- Hvis Hvis A har rollen r_2, så skal A også ha rollen r_1
  $pop(r_2) \subseteq pop(r_1)$ for alle tilstander



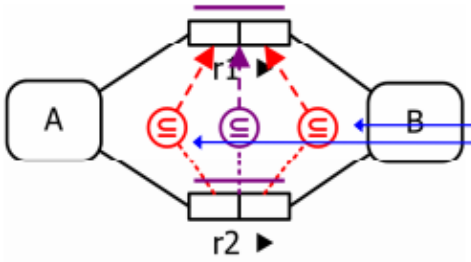
IMPLISERTE SKRANKER



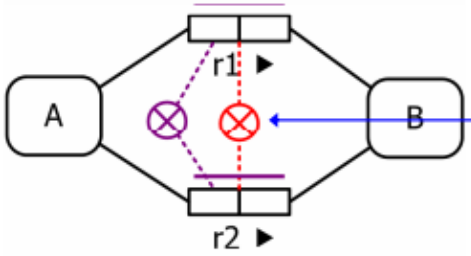
Implisert likhetsskranke skal ikke tegnes



Implisert delmengde-skranke skal ikke tegnes



Impliserte delmengde-skranker skal ikke tegnes



Implisert ulikhetsskranke skal ikke tegnes

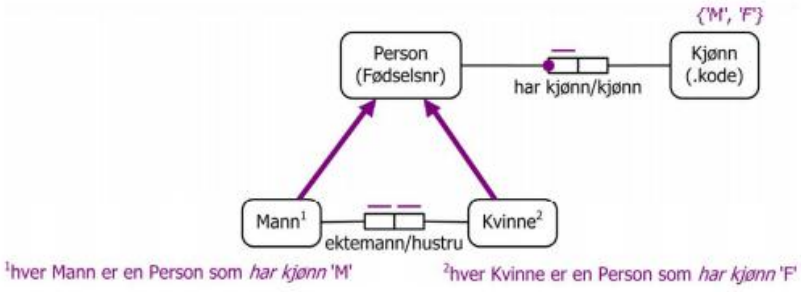
UNDERBEGREPER

- B er et underbegrep av A hvis vi alltid har at *pop(B) ⊆ pop(A)*

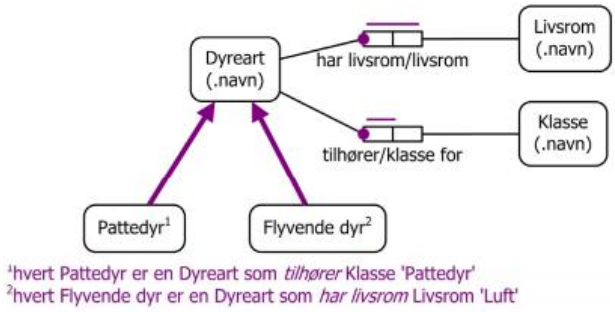
  Notasjon:

- Underbegreper arver representasjon og roller fra superbegrepet. I tillegg har de sine egne roller
- Underbegrepsskranker brukes til å bestemme hvilket underbegrep hver enkelt forekomst tilhører
- Underbegreper kan *overlappe* eller være *disjunkte*
- Underbegrepene kan, men må ikke, være *uttømmende* mhp. sitt superbegrep
- Resonnementer over entydighetsskranker, totale roller og underbegrepsskrankene avslører om underbegrepene er overlappende og/eller uttømmende

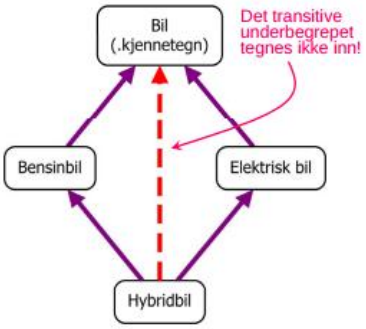
UNDERBEGREPSSKRANKE



EKSEMPEL PÅ OVERLAPPENDE OG IKKE-UTTØMMENDE UNDERBEGREPER

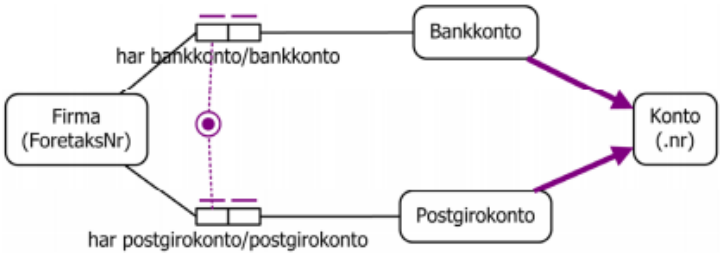


UNDERBEGREPER I FLERE NIVÅER



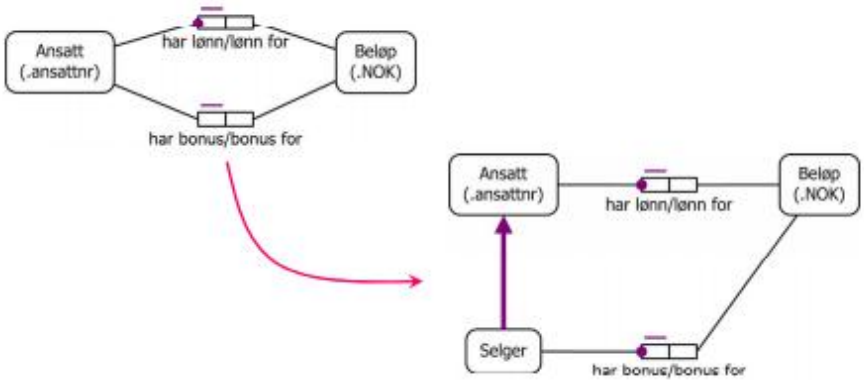
KOMBINERT TOTAL ROLLE OG UNDERBEGREP

- A skal ha enten rollen r_1 eller rollen r_2.
  $pop(r_1) \cup pop(r_2) = pop(A)$ for alle tilstander

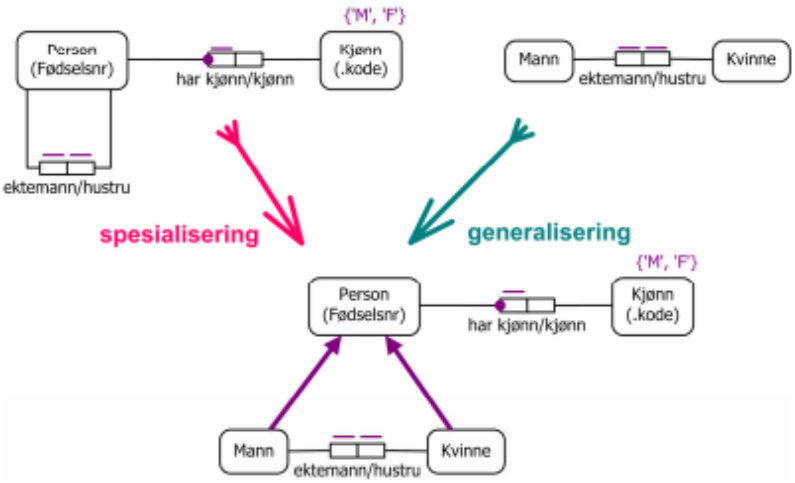


MANGEL AV TOTALE ROLLER

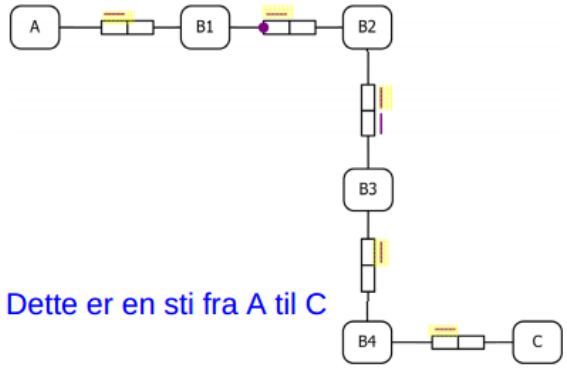
- Mangel på totale roller kan indikere et underbegreper



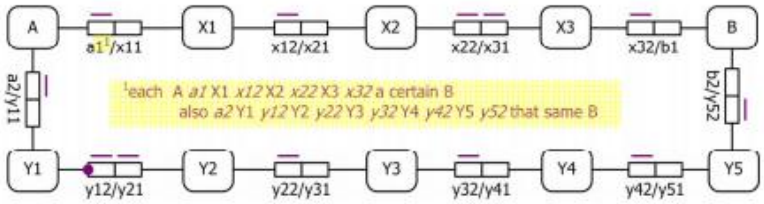
SPESIALISERING OG GENERALISERING



STIER

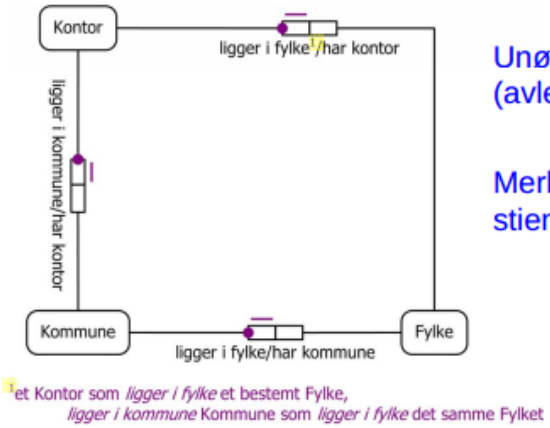


EKVIVALENTE STIER



- Dersom vi har to stier fra A til B som er slik at hvis vi starter med én forekomst i A, så skal vi komme til samme forekomst forekomst i B uavhengig av hvilken av de to stiene vi følger, så kaller vi de to stiene ekvivalente

AVLEDBARE DATA



Unødvendig (avledbar) informasjon

Merk de ekvivalente stiene!

JOINSKRANKER

- Ekvivalente stier er et viktig eksempel på det vi kaller *joinskranker*
- For å håndheve disse skrankene kan det være nødvendig å foreta en join mellom tabeller
- Alle mengdeskranker kan opptre som joinskranker
- Ekvivalente stier er et spesialtilfelle av en joinlikhetsskranke

AVANSERTE SKRANKER

- Alle skranker vi ikke har grafiske symboler for, kalles avanserte skranker
- Disse skrives på ORM-diagrammet som tekst
- Lovlig språk er førsteordens logikk og vanlig aritmetikk (regneformler)
- Lovlige variable er konstanter, roller, stier og inverse stier
- ORM 2 har et eget språk for avanserte skranker

BEHANDLING AV TID

VERSJONERING

- Hvis vi ønsker at databasen skal vise historiske opplysninger, lagrer vi *tidsstemplede versjoner* av informasjonen
- Med en versjon mener vi her et øyeblikksbilde av all informasjon
- De tidsstemplede versjonene kan ordnes langs en tidsakse

HVA ER ET TIDSPUNKT

- Tidsaksen består i praksis alltid av tidsintervaller i informasjonsmodellen
- Granulariteten til intervallene avhenger av behovet for nøyaktighet.
  Granularitet avgjør:
    - hvordan tidsintervallene skal representeres
        - Hvert tidsintervall identifiseres ved et tidsstempel, f.eks. år+måned, år+ukenummer, år+mnd+dag, år+mnd+dag+time+minutt
    - hva som er «samtidig»
        - Hendelser innen samme tidsintervall kan ikke skilles i tid
- En informasjonsmodell kan ha flere tidsakser med ulik oppdeling og granularitet
    - det er ikke alltid mulig å bestemme samtidighet på tvers av ulike tidsakser

HVA SKAL TIDSSTEMPELET REFLEKTERE?

1 Når en hendelse faktisk inntraff?
2 Når versjonen ble lagt inn?
3 Når versjonen skal tre i kraft?
4 Når versjonen ble ugyldig?
5 ...

Maks én av disse! (men neppe nr. 4)

Merk:

- De fleste modeller ligger etter virkeligheten, f.eks. nr.1 (og nr. 2):
  Det tar tid før en hendelse i virkeligheten kan gjenfinnes som en versjon i databasen (mikrosekunder til dager, avhengig av registreringsprosess)
- Noen modeller må ligge «foran» virkeligheten, f.eks. nr.3

TIDSMESSIG KONTINUITET

- Det er maksimalt én versjon pr. mulig tidsstempel
- Dersom det legges inn færre enn en versjon pr. mulig tidsstempel, så må det være mulig å avlede *ikke-materialiserte versjoner* for de tidsstemplene som ikke har en tilhørende versjon

FILM- OG LYSBILDEPRINSIPPET

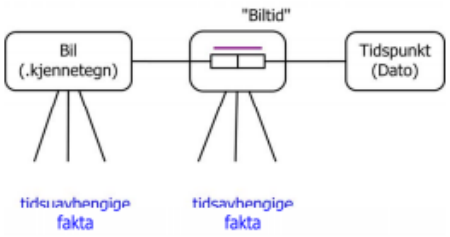
- Filmprinsippet:
    - Én ny versjon for hvert nytt mulig tidsstempel
    - Trenger mye lagerplass ved fin granularitet
- Lysbildeprinsippet:
    - Observerer og registrerer virkeligheten bare av og til
    - Bygg inn nok kunnskap til at de ikke-materialiserte versjonene kan utledes

ELEMENTÆRE SETNINGER OG TID

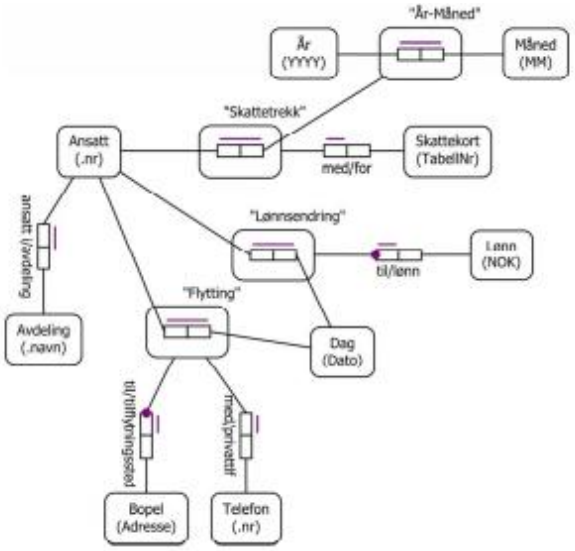
- Tre typer elementære setninger:
    - De som ikke har noen tidsdimensjon
    - De hvor vi bare ønsker å ta vare på siste aktuelle verdi

    - De hvor vi ønsker å modellere en tidsdimensjon
- Virkeligheten har to typer endringer:
    - Kontinuerlige
    - Sprangvise
- Virkelighet kontra modell:
    - Versjonene endrer seg alltid i rykk og napp
    - Versjonene kan være tidsmessig forskjøvet i forhold til virkeligheten

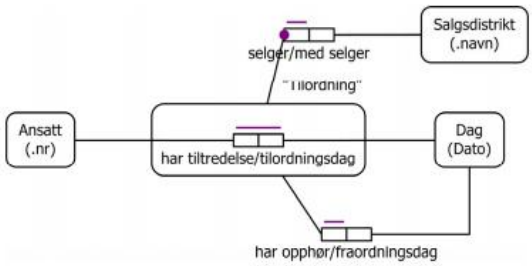
BEGREPSDANNELSE MED TIDSAKSEN



- Eksempel:



- Eksempel; angivelse av opprør



REPRESENTASJONER

- Alle begreper må kunne representeres
    - Begrepsforekomster kan ikke lagres;
      det vi lagrer, er *representasjonsforekomster*
- Skal vi kunne realisere modellen som en relasjonsdatabase, må vi representere alle begrepene entydig

VALG AV REPRESENTASJON

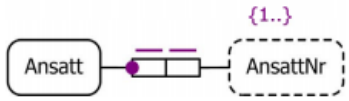
- Valg av representasjon:
    - entydig bro mellom en forekomst av en representasjon og forekomsten av det tilhørende begrepet
    - helst uforanderlig
    - støtte utveksling av informasjon mellom systemer
- *Identifikator* == representasjon hvor det er en uforanderlig en-til-en-bro mellom begrep og representasjon

REPRESENTASJONSTYPER

- navn, koder, forkortelser
- boolske verdier
- tellbare størrelser
- tids- og romlige verdier
- fritekst
- representasjoner av bilde og lyd

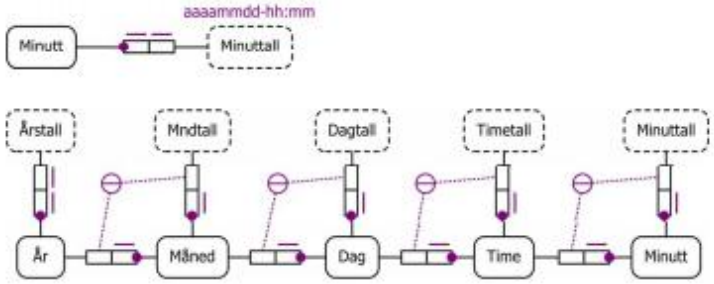
IKKE-INFORMASJONSBÆRENDE REPRESENTASJONER

- *Representasjonen* til begrepet identifiserer en forekomst av begrepet
- Det fins ingen innkodet informasjon i representasjonen



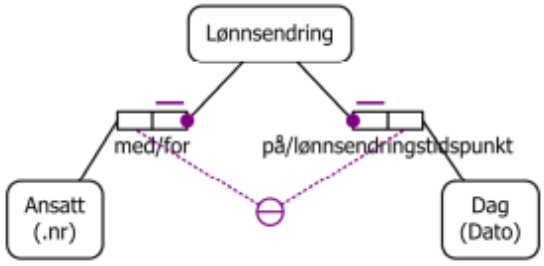
DELVIS INFORMASJONSBÆRENDE REPRESENTASJONER

- Deler av representasjonen til et begrep identifiserer en forekomst av et annet begrep. Dette kan, men behøver ikke, være synlig i modellen



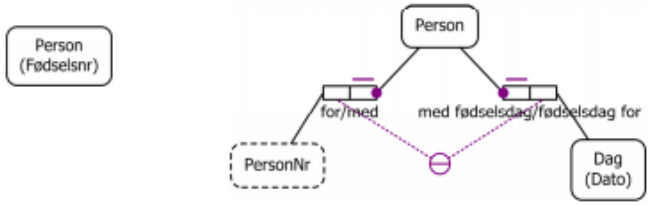
TOTALT INFORMASJONSBÆRENDE REPRESENTASJONER

- Representasjonen til begrepet består av en samling elementer der hvert element identifiserer en forekomst av et annet begrep



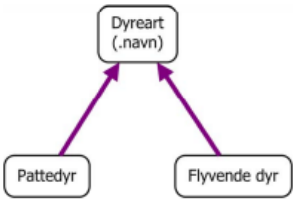
Synliggjøring eller ikke av informasjonsbærende representasjon i modellen?

- Hvis det er en mulighet for at brukeren etterspør denne informasjonen, bør den vises i modellen



REPRESENTASJON VIA SUPERBEGREP

- Underbegreper arver alltid representasjonen til sitt superbegrep



REPRESENTASJON VIA EN-TIL-EN-FAKTATYPE

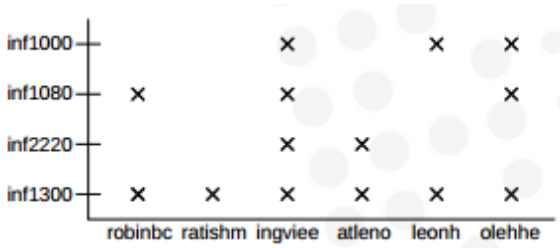
- Et begrep med en påkrevd rolle i en en-til-enfaktatype til et annet begrep kan identifiseres indirekte gjennom det andre begrepet
  Eksempel:
  En hovedstad kan identifiseres med det det landet den er hovedstad i landet den er hovedstad i

RELASJONER OG RELASJONSDATABASER

Personale

Ans#	Navn	Fdato	Pers#	Avd
10	Iziz	290264	39201	**nil**
9	Ehab	131172	35797	Knøttene
8	Bjørn	150571	34322	Tintin
12	Liv	031079	39201	**nil**

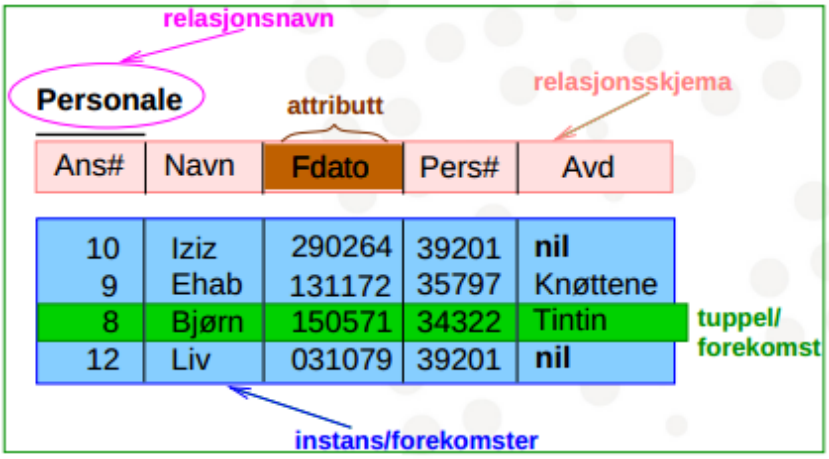
- Relasjon: Et matematisk begrep som kan tolkes som en tabell med verdier
- Relasjonsdatabase: En samling relasjoner
- nil indikerer at ingen verdi ligger lagret i denne posisjonen



- En matematisk relasjon er en mengde av ordnede tupler. I diagrammet over utgjør x-ene en matematisk relasjon. I dette tilfellet består den av (14) 2-tupler og kalles en binærrelasjon. Den kan også skrives f.eks. slik:

  {<robinbc, inf1080>, <robinbc, inf1300>, <ratishm, inf1300>,
  <ingviee, inf1000>, ..., <olehhe, inf1300>, }

RELASJONER - TERMINOLOGI



- *dom„* (Fdato…) *ƒ* = {sekssifrede tall med begrensninger på hvilke tall som er lovlige datoer}
- *dom*(„ Avd)…*ƒ* {Knøttene, Tintin, Tommeliten, Trollungene}

FORMELLE DEFINISJONER

- **Domene**: En mengde atomære verdier. (At elementenei et domene er atomære, betyr at elementene ikke selv kan være mengder.)
- **Attributt**: Et navn på en rolle spilt av et domene («kolonnenavn»). Hvis A er et attributt, skriver vi *dom*(„ A) =…*ƒ* D for å uttrykke at A er en rolle spilt av domenet D.
- **Relasjonsskjema** R„ (A_1, A_2, …, A_n)…: En navngitt mengde attributter R = {A_1, A_2, …, A_n} der R er relasjonsnavnet. *n* kalles relasjonens *grad* eller *aritet*.
- **Instans** av et relasjonsskjema R(„ A_1, A_2, …, A_n):
  En mengde {t_1, t_2, …, t_m} («rader») der hver t_k er et n-tuppel av verdier fra domenene til A_1, A_2, …, A_n.
  (Noen av verdiene kan være nil, f.eks. fordi verdien for et attributt ikke er lagt inn ennå, fordi verdien er ukjent eller fordi den ikke er relevant.)

- Dersom *t* er et tuppel i en instans av R(„ A_1, A_2, …, A_n) og *t* = *f* < v_1, v_2, …, v_n >, så er f.eks. *t*[A_2]‡ = *f* < v_2 > og *t*[† A_3, A_1, A_5] = ‡ *f* < v_3, v_1, v_5 >.
- **Relasjon**: Et relasjonsskjema med en tilhørende instans.

Relasjonsskjemaet kalles relasjonens *intensjon*. Instansen kalles relasjonens *ekstensjon*.

MERK

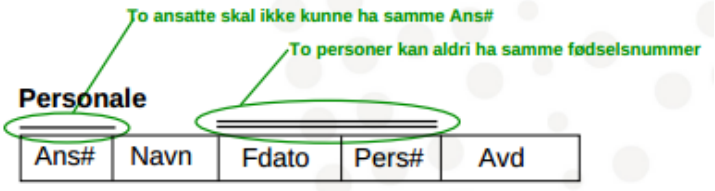
- Tuplenes rekkefølge i en instans er vilkårlig
- Verdienes rekkefølge i et tuppel er i utgangspunktet ikke vilkårlig (dette er mest for at notasjonen skal bli enklere)
- I en instans kan det ikke finnes to like tupler
- Et domene kan være endelig eller uendelig
- To attributter i et relasjonsskjema kan ha samme domene, men ikke samme navn

NØKLER OG NØKKELATTRIBUTTER

Personale

Ans#	Navn	Fdato	Pers#	Avd
10	Iziz	290264	39201	**nil**
9	Ehab	131172	35797	Knøttene
8	Bjørn	150571	34322	Tintin
12	Liv	031079	39201	**nil**

- Vi ønsker ikke at to ansatte skal kunne ha samme Ans#
- To personer kan aldri ha samme fødselsnummer = Fdato + Pers#
- **Supernøkkel**:
  En kombinasjon (delmengde) *X* av attributtene {A_1, A_2, …, A_n} som er slik at hvis *t* og *u* er to tupler hvor *t* !=*f* *u*, så er *t*† [*X*]‡ != *u*[† *X*]‡ .
  Merk:
  Relasjonsskjemaet er alltid selv en supernøkkel
- **Kandidatnøkkel**:
  En *minimal* supernøkkel. Dvs: Fjerning av et hvilket som helst attributt fører til at de gjenværende attributtene ikke lenger utgjør en supernøkkel.
- Supernøkler benyttes til å uttrykke integritetsregler
- **Primærnøkkel**: En utvalgt blant kandidatnøklene. Alle relasjoner skal ha nøyaktig én primærnøkkel.
- **Nøkkelattributt**: Attributt som er med i (minst) en kandidatnøkkel.



- Primærnøkkelen blir gjerne markert med én strek
- Andre kandidatnøkler er i dette tilfellet markert med to streker

- Sammenlign kandidatnøkler og entydighetsskranker i ORM: Begge angir at forekomster under skranken bare kan forekomme én gang

FUNKSJONELLE AVHENGIGHETER

Personale

Ans#	Navn	Fdato	Pers#	Avd

- Det at en person har høyst ett Ans#, gjør at hvis vi vet hvilken person det er snakk om (dvs. vi kjenner personens Ans#), så vet vi også navnet, fødselsnummeret og avdelingen til personen
- Primærnøkkelen definerer altså en funksjon fra forekomstene av Ans# til forekomstene av Navn, Fdato, Pers# og Avd
    - Det samme gjelder andre kandidatnøkler: Hvis vi kjenner forekomstene for attributtene Fdato og Pers#, så har vi bare én mulig verdi for hver av Ans#, Navn og Avd.
- Vi sier at Navn, Fdato, Pers#, Avd er **funksjonelt avhengig** av Ans#, eller at vi har en **FD** (Functional Dependency) fra Ans# til Navn, Fdato, Pers#, Avd
- Den vanlige notasjonen for en FD er: Ans# → Navn, Fdato, Pers#, Avd

FREMMEDNØKLER

Personale

Ans#	Navn	Fdato	Pers#	Avd
10	Iziz	290264	39201	nil
9	Ehab	131172	35797	Knøttene
8	Bjørn	150571	34322	Tintin
12	Liv	031079	39201	nil

Barn

Løpe#	Navn	Fdato	Avd	TilknPers
2	Lisa	180507	Tintin	8
5	Trym	030208	Knøttene	nil
4	Adnan	300308	Tommeliten	nil
7	Adnan	151207	Knøttene	9

Fremmednøkkel: Ett eller flere attributter som **peker ut/refererer** et tuppel i en annen relasjon.

- Fremmednøkkelen må ha samme antall attributter som primærnøkkelen i den relasjonen den peker ut, og attributtene må ha parvis samme domener. (Noen databasesystemer tillater også fremmednøkler til kandidatnøkler som ikke er primærnøkler.)
- Korresponderende attributter behøver ikke å ha samme navn
- Det er lov å ha fremmednøkler til «seg selv»
- Fremmednøkler benyttes til å uttrykke integritetsregler

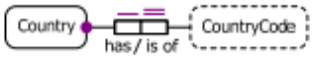
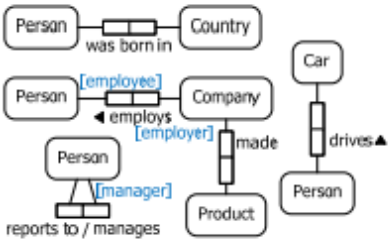
PÅKREVDE INTEGRITETSREGLER I RELASJONSDATABASER

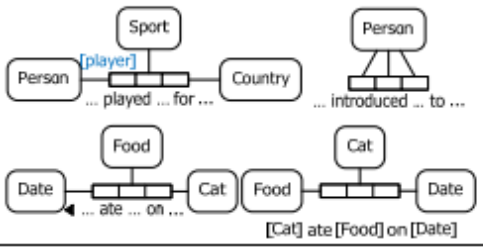
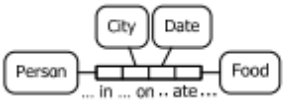
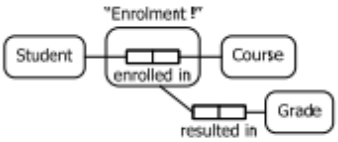
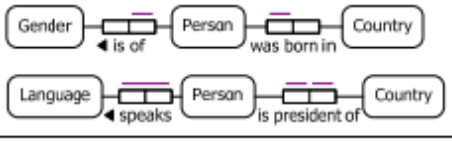
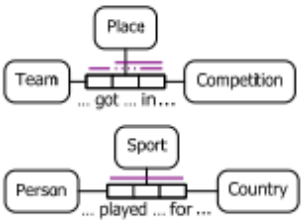
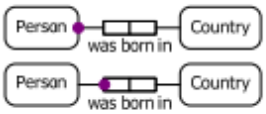
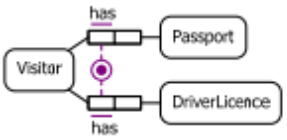
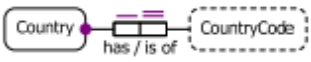
- **Entitetsintegritet**: Alle relasjonsskjemaer skal ha en og bare en primærnøkkel. Ingen av attributtene i primærnøkkelen får være nil
- **Referanseintegritet**: Hvis fremmednøkkelen ikke er nil, så skal det finnes et tuppel i den refererte relasjonen hvor primærnøkkelen har samme verdi som fremmednøkkelen (dvs. at det refererte tuppelet skal eksistere)

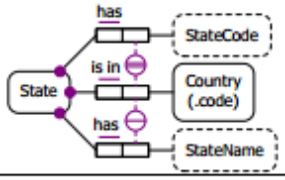
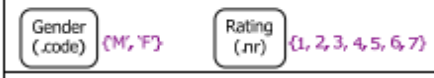
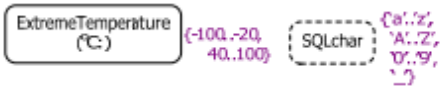
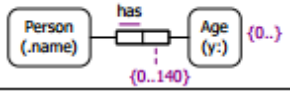
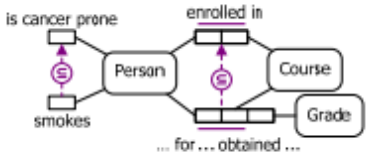
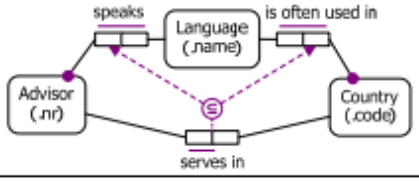
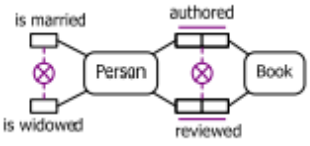
Merk: I forskjellige relasjonsskjemaer kan attributtnavn gjenbrukes. Notasjon for å skille mellom attributter med like navn: $R.A_i$.
I tillegg kan databasen ha andre integritetsregler, foreksempel kandidatnøkler utover primærnøklene.

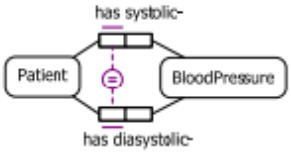
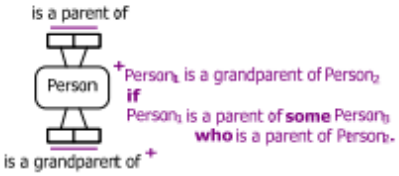
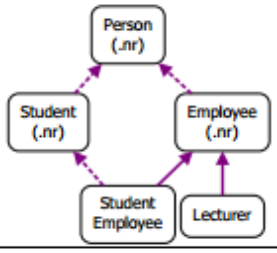
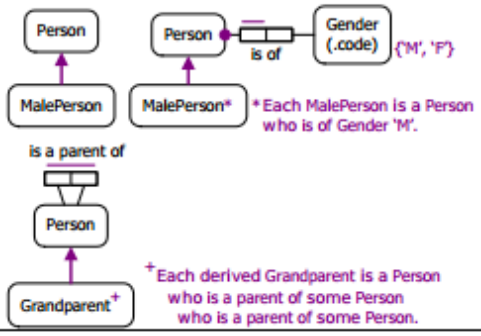
RELASJONSDATABASER - DEFINISJONER

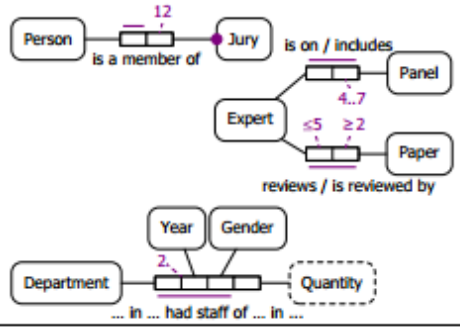
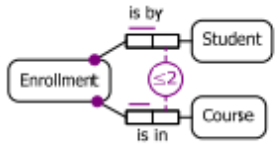
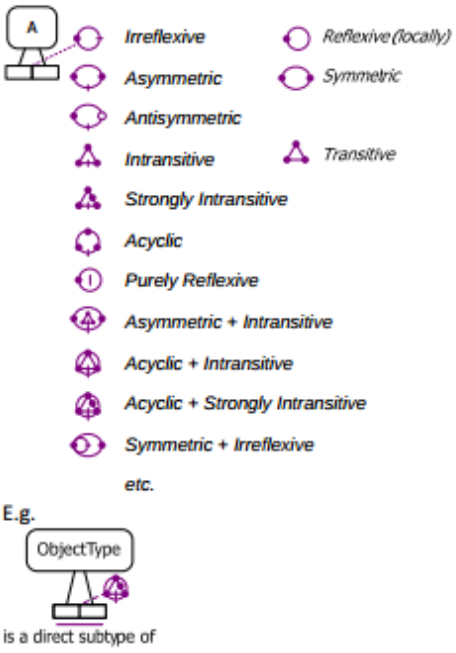
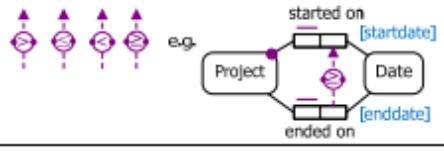
- **Relasjonsdatabaseskjema**: Samling av relasjonsskjemaer + integritetsregler
- **Relasjonsdatabaseinstans**: Samling av relasjonsinstanser
- **Relasjonsdatabase** = Relasjonsdatabaseskjema + relasjonsdatabaseinstans

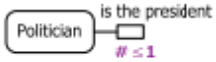
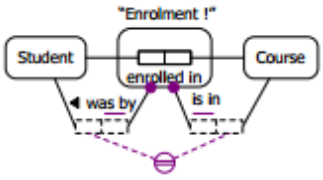
Construct	*Examples*	*Description/Notes*
Entity Type	Country or Country or Country	Named soft rectangle, named hard rectangle, or named ellipse. The soft rectangle shape is the default.
Value Type	CountryCode or CountryCode or CountryCode	Named, dashed, soft rectangle (or hard rectangle or ellipse).
Entity type with popular reference mode	Country (.code) Course (.code) Company (.name) Building (.nr)	Abbreviation for injective reference relationship to value type, e.g. 
Entity type with unit-based reference mode	Height (cm:) Mass (kg:) Salary (USD:) Price (EUR:) Height (cm: Length) Salary (USD: Money) Price (EUR: Money)	Abbreviation for reference type, e.g.  Optionally, unit type may be displayed.
Entity type with general reference mode	Book (ISBN) Website (URL) WebLink (URL)	Abbreviation for reference type, e.g. 
Independent Object Type	Country ! CountryCode !	Instances of the type may exist, without playing any elementary fact roles
External Object Type	Address^	This notation is tentative (yet to be finalized)
Predicate (unary, binary, ternary, etc.)	smokes was born in ... speaks ... very well ... played ... for ... ... in ... on ... ate ...	Ordered set of 1 or more role boxes with at least one predicate reading in mixfix notation. If shown, object placeholders are denoted by "...". If placeholders are not shown, unaries are in prefix and binaries are in infix notation.
Duplicate type or predicate shape	Person StateCode was born in	If an object type or predicate shape is displayed more than once (on the same page or different pages) it is shadowed.
Unary fact type	Person smokes	The smokes role may be played by instances of the Person object type
Binary fact type		By default, predicate readings (binary or longer) are read left-to-right or top-to-bottom. An arrow-tip is used to display a different reading direction. Role names may be displayed in square brackets beside their role. Forward and inverse readings for binaries may be shown together, separated by "/".

Construct	Examples	Description/Notes
Ternary fact type		Role names may be added in square brackets. Arrow-tips are used to reverse the default left-right or top-down reading order. Reading orders other than forward and reverse are shown using named placeholders.
Quaternary fact type		The above notes for the ternary case apply here also. Fact types of higher arity (number of roles) are also permitted.
Objectification (a.k.a. nesting)		The enrolment fact type is objectified as an entity type whose instances can play roles. In this example, the objectification type is independent, so we can know about an enrolment before the grade is obtained.
Internal uniqueness constraint (UC) on unaries		These are equivalent (by default, predicates are assumed to be populated with sets, so no whole fact may be duplicated).
Internal UC on binaries		The examples show the 4 possible patterns: 1:*n* (one-to-many); *n*:1 (many-to-one); *m*:*n* (many-to-many); 1:1 (one-to-one0
Internal UC on ternaries. For *n*-aries (n > 1) each UC must span at least *n*-1 roles		The first example has two, 2-role UCs: the top UC forbids ties; the other UC ensures that each team gets only place per competition (a dotted line excludes its role from the UC). The second example has a spanning UC (many-to-many-to-many).
Simple mandatory role constraint		The example constraint means that each person was born in some country. The mandatory role dot may be placed at either end of the role connector.
Inclusive-or constraint (disjunctive mandatory role)		The constraint is displayed as a circled dot connected to the constrained roles. The example constraint means that each visitor referenced in the model must have a passport or a driver licence (or both).
Preferred internal UC		A double bar on a UC indicates it underlies the preferred reference scheme.

Construct	*Examples*	*Description/Notes*
External UC (double-bar indicates preferred identifier)		Here, each state is primarily identified by combining its country and state code. Each combination of country and state name also applies to only one state.
Object Type Value Constraint		*Enumerations*
		*Ranges* are inclusive of end values by default. Round brackets are used to exclude an end value. Square brackets may be added to explicitly declare inclusion, e.g. the constraint on PositiveScore may also be specified as {(0..100]}.
		Multiple combinations are allowed.
Role value constraint		As for object type value constraints, but connected to the constrained role. Here, an age of a person must be at most 140 years.
Subset constraint		The arrow points from the subset end to the superset end (e.g. if a person smokes then that person is cancer prone). The role sequences at both ends must be compatible. A connection to the junction of 2 roles constrains that role pair.
Join subset constraint		The constrained role pair at the superset end is projected from a role path that involves a conceptual join on Language. The constraint declares that if an advisor serves in a country then that advisor must speak a language that is often used in that country.
Exclusion constraint		These constraints mean that no person is both married and widowed, and no person reviewed and authored the same book. Exclusion may apply between 2 or more compatible role sequences, possibly involving joins.
Exclusive-or constraint		An exclusive-or constraint is simply the conjunction of an inclusive-or constraint and an exclusion constraint. Also known as an xor constraint.

Construct	*Examples*	*Description/Notes*
Equality constraint		This constraint means that a patient's systolic BP is recorded if and only if his/her diastolic BP is recorded. An equality constraint may apply between 2 or more compatible role sequences, possibly involving joins.
Derived fact type, and derivation rule		A fact type is either asserted, derived, or semiderived. A derived fact type is marked with an asterisk "*". A derivation rule is supplied. A double asterisk "**" indicates derived and stored (eager evaluation).
Semiderived fact type, and derivation rule		A fact type is semiderived if some of its instances may be derived, and some of its instances may be simply asserted. It is marked by "+" (half an asterisk). "++"indicates semiderived and stored (eager evaluation for derived instances).
Subtyping		All subtypes are proper subtypes. An arrow runs from subtype to supertype. A solid arrow indicates a path to the subtype's preferred identifier (e.g. here, student employees are primarily identified by their employee number). A dashed arrow indicates the supertype has a different preferred identifier.
Subtyping constraints		A circled "X" indicates the subtypes are mutually exclusive. A circled dot indicates the supertype equals the union of the subtypes. The combination (xor constraint) indicates the subtypes partition the supertype (exclusive and exhaustive).
Subtype derivation status		A subtype may be • asserted, • derived (denoted by "*"), • or semiderived (denoted by "+"). If the subtype is asserted, it has no mark appended and has no derivation rule. If the subtype derived or semiderived, a derivation rule is supplied.

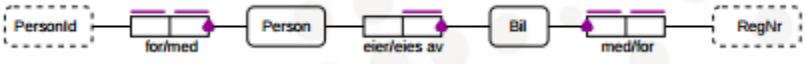
Construct	*Examples*	*Description/Notes*
Internal frequency constraint		This constrains the number of times an occurring instance of a role or role sequence may appear in each population. Here: each jury has exactly 12 members; each panel that includes an expert includes at least 4 and at most 7 experts; each expert reviews at most 5 papers; each paper that is reviewed is reviewed by at least 2 experts; and each department and year that has staff numbers recorded in the quaternary appears there twice (once for each gender).
External frequency constraint		The example constraint has the following meaning. In this context, each combination of student and course relates to at most two enrolments (i.e. a student may enroll at most twice in the same course)
Ring constraints		A ring predicate *R* is locally reflexive if and only if, for all *x* and *y*, *xRy* implies *xRx*. E.g. "knows" is locally but not globally reflexive. Reflexive, symmetric and transitive properties may also be enforced using semiderivation rather than by constraining asserted fact types. The example constrains the subtyping relationship in ORM to be both acyclic (no cycles can be formed by a chain of subtyping connections) and strongly intransitive (no object type *A* can be both a direct subtype of another type *B* and an indirect subtype of *B*, where indirect subtyping means there is a chain of two or more subtyping relationships that lead from *A* to *B*). Ring constraints may be combined only if they are compatible, and one is not implied by the other. ORM tools ensure that only legal combinations are allowed.
Value-comparison constraints		The example constraint verbalizes as: **For each** Project, **existing** enddate >= startdate.

Construct	*Examples*	*Description/Notes*
Object cardinality constraint		The example constraints ensure there is exactly one president and at most 100 senators (at any given time),
Role cardinality constraint		The example constraint ensures that at most one politician plays the role of president (at any given time).
Deontic constraints	*Uniqueness* *Mandatory* *Subset, Equality, Exclusion* *Frequency* ≤f *Irreflexive* *Acyclic* *Asymmetric* *Asym-Intrans* *Intransitive* *Acyclic-Intrans* *Antisymmetric* *Symmetric* *Strongly Intransitive* etc. e.g. 	Unlike alethic constraints, deontic constraint shapes are colored blue rather than violet. Most include "o" for "obligatory". Deontic ring constraints instead use dashed lines. In the parenthood example, the alethic frequency constraint ensures that each person has at most two parents, the alethic ring constraint ensures that parenthood is acyclic, and the deontic ring constraint makes it obligatory for parenthood to be strongly intransitive.
Textual constraints	 [1] **Each** Employee **who** has Rank 'NonExec' uses **at most one** CompanyCar. [2] **Each** Employee **who** has Rank 'Exec' uses **some** CompanyCar.	First-order constraints with no graphic notation may be expressed textually in the FORML 2 language. These examples use footnoting to capture a restricted uniqueness constraint and a restricted mandatory role constraint.
Objectification display options: link fact types, and compact display.	 	Internally, link fact types connect objectified associations to their component object types. By default, display of link fact types is suppressed. If displayed, link predicate shapes use dashed lines instead of solid lines. Objectification object types may also be displayed without their defining components, using an object type shape containing a small predicate shape, as shown in this Enrolment example.

REALISERINGSALGORITMEN

FRA ORM-DIAGRAM TIL RELASJONSDATABASESKJEMA

UNDERLIGGENDE IDÉ (forenklet)

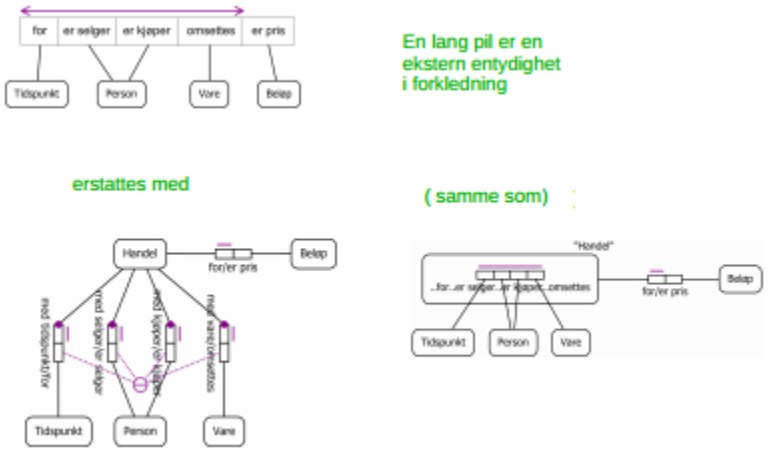


- Lag en tabell for hvert *begrep*: Person( ), Bil( )
- Lag en tabell for hver *faktatype*: eier/eies_av(,)
- Perfekte broer brukes til å bestemme hvordan begrepene skal representeres: Person(PersonId), Bil(RegNr), eier/eies_av(PersonId, RegNr)
- Entydighetspiler brukes til å bestemme primærnøkler: Person(PersonId), Bil(RegNr), eier/eies_av(PersonId, RegNr)
- For å få en "penere" database: slå sammen tabeller med samme primærnøkkel: Person(PersonId), Bil(RegNr, PersonId)

FORUTSETNINGER/FORBEREDELSER

- Alle lange piler må gjøres til gjenstand for begrepsdannelse (og gis et navn)
- ORM-diagrammet må være refererbart
- Diagrammet må ikke inneholde synonyme broer: Alle broer må ha en entydig begrepsrolle

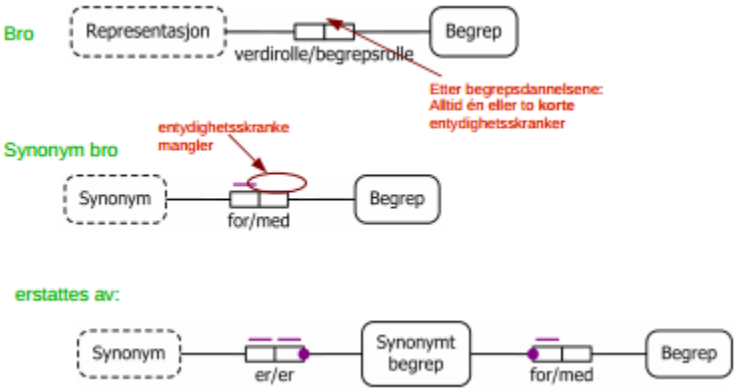
A) BEGREPSDANNELSE AV "LANGE PILER"



B) REFERERBARE ORM-DIAGRAMMER

- Intuitivt er et ORM-diagram refererbart hvis alle begreper kan representeres entydig (via perfekte broer)

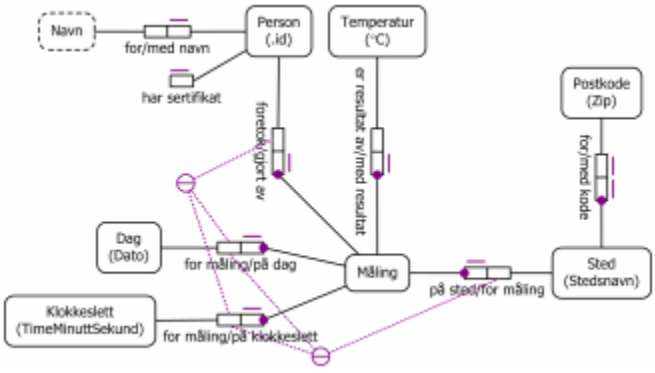
C) ELIMINISJON AV SYNONYME BROER



REALISERINGSALGORITMEN

1. Hvert begrep gir opphav til en relasjon (basistabell) med samme navn som begrepet
2. Finn referansemåte for alle begreper og marker alle prefererte referansetyper som brukt (referansemåtene blir primærnøkler)
3. Grupper resterende broer til sine respektive begreper (hver bro gir ett attributt i tabellen)
4. Grupper resterende faktatyper (hver faktatype blir en fremmednøkkel)
5. Overfør skrankene til relasjonsskjemaet
6. Bestem hvilke referanserelasjoner som skal fjernes

EKSEMPEL



SQL - STRUCTURED QUERY LANGUAGE

- SQL er et deklarativt språk for spørringer mot relasjonsdatabaser
- SQL inneholder også konstruksjoner for å definere nye relasjonsdatabaser, for å legge inn og endre data i databasen, mm

SQL-STANDARDER

- Flere standarder:
    - ANSI SQL (1986)
    - SQL2 (SQL-92)
    - SQL3 (SQL:1999) = SQL2 + objekt-relasjonelle egenskaper mm
- Mange dialekter:
    - Hvert DBMS har sine særegenheter. Sjekk den aktuelle
    - DBMSens dokumentasjon!
    - Alle oppfyller ANSI SQL, stort sett også SQL2
    - Noen deler av SQL er ikke veldefinert (selv ikke i ANSI SQL) og behandles derfor potensielt forskjellig i forskjellige DBMSer

SQLs BESTANDDELER

SQL består av en samling konstruksjoner som funksjonelt,
men ikke syntaktisk, kan deles opp slik:

- SDL: Storage Definition Language
  3-skjema-arkitekturens fysiske lag
- **DDL: Data Definition Language**
  3-skjemaarkitekturens konseptuelle lag
- VDL: View Definition Language
  3-skjemaarkitekturens presentasjonslag
- **DML: Data Manipulation Language**
  innlegging, endring og sletting av data
- **DQL: Data Query Language—spørrespråk**
- DCL: Data Control Language—integritet og sikkerhet

SQLs DDL - DATA DEFINITION LANGUAGE

- **create**: Opprette tabell
- **drop**: Fjerne tabell
- **alter table**: Endre tabell, herunder:
    - Legge til eller fjerne kolonner
    - Legge til, fjerne eller endre integritetsregler (constraints)

CREATE

create table R
$(A_1\ D_1\ [S_1],$
$\ldots$
$A_n\ D_n\ [S_n],$
[*liste av skranker*]
);

R er navnet på relasjonen
A_i er et atributt
D_j er et domene
S_k er en skranke

[ ] betyr at dette leddet er en valgfri del av setningen

- Eksempel:
  Ansatt(AId, Navn, Tittel, Fdato, Pnr, AnsDato)
  - Vi ønsker ikke at to ansatte skal kunne ha samme AId
  - To personer kan aldri ha samme fødselsnummer = Fdato + Pnr
  - Dermed er både AId og (Fdato, Pnr) kandidatnøkler. Vi velger AId som primærnøkkel og markerer (Fdato, Pnr) som kandidatnøkkel.

```
create table Ansatt (
    AId         int primary key,
    Navn        varchar(40) not null,
    Tittel      varchar(15) not null,
    Fdato       char(6) not null,
    Pnr         char(5) not null,
    AnsDato     date,
unique (Fdato, Pnr)
);
```

DATATYPER I PostgreSQL

datatype	*forklaring*
integer	heltall
real	flyttall
numeric(n,d)	n desimale sifre, d etter desimal-punktum
char(n)	tekst med fast lengde
varchar(n)	tekst med variabel lengde
boolean	boolsk verdi
date	dato
time	klokkeslett
timestamp	dato og klokkeslett
bit(n)	bitstreng med fast lengde
bit varying(n)	bitstreng med variabel lengde

PRIMÆRNØKLER

- Kan, dersom attributtet utgjør primærnøkkelen alene, deklareres som i eksempelet over. Den kan også deklareres seperat:

```
create table Ansatt (
    AId         int,
    ...
primary key (AId)
);
```

- **Regler**:
    - Maks én primærnøkkeldeklarasjon pr. relasjon
    - Konsekvenser av deklarasjonen:
        - To tupler i relasjonen får ikke stemme overens i alle attributtene i primærnøkkelen. Forsøk på brudd ved **insert** eller **update** skal avvises av DBMSet
        - Attributtene i primærnøkkelen får ikke inneholde **null**
    - Dette må sjekkes av systemet ved hver **insert** og hver **update**.

KANDIDATNØKLER

- Kan deklareres som i eksempelet, eller sammen med nøkkelattributtet (hvis dette attributtet utgjør nøkkelen alene)

```
create table Ansatt (
    ...
    Fnr     char(11) not null unique,
    ... );
```

- **Regler**:
    - Flere kandidatnøkkeldeklarasjoner er tillatt pr. relasjon
    - Konsekvenser av deklarasjonen:
        - To tupler i relasjonen får ikke stemme overens i alle attributtene i kandidatnøkkelen
        - Kan brytes hvis ett eller flere av attributtene i kandidatnøkkelen inneholder **null**
    - Dette må sjekkes av systemet ved hver **insert** og hver **update**.

SKRANKER

- Skranke på et attributt eller et tuppel i en relasjon:
  create table R (... **check** ...);
    - Sjekkes ved **insert** og **update** av R
- Skranke på tvers av relasjoner:
  create trigger T ...;
    - Triggere «vekkes» når en hendelse (typisk insert, update, delete på en relasjon) skal til å inntreffe
    - Når triggerne «vekkes», eksekveres en tilhørende metode
- **Skranke på ett attributt**:
    - not null
        - **create table** Ansatt (... Fdato int **not null**, ..);
        - Konsekvenser:
            - Kan ikke sette inn tuppel med verdien null i attributtet
            - Kan ikke endre verdien til null senere

    - check
        - **create table** Ansatt (
          . . .
          T i t t e l **varchar**(15)
          check (T i t t e l= ’ Selger ’ or T i t t e l= ’ Direktør ’ or . . .) ,
          . . .) ;
    - Angir en betingelse på attributtet. Sjekkes ved hver endring av attributtets verdi

FREMMEDNØKLER

- Deklarasjon av fremmednøkler:

```
create table Timeliste (
    AId      int references Ansatt (AId),
    ...);
```

- Alternativt:

```
create table Timeliste (
    AId      int,
    ...
    foreign key (AId) references Ansatt (AId)
    ...);
```

- **Regler**:
    - Konsekvenser av deklarasjonen:
        - De refererte attributtene må være deklarert **unique** eller **primary key**
        - Verdier (!=" **null**) som opptrer i fremmednøkkelens refererende attributter *må* opptre i de refererte attributtene
    - Dette må sjekkes av systemet både ved **insert**, **update** og **delete**

DROP, ALTER

drop table R;
alter table R **add** A_x D_y ;
alter table R **drop** A_x;

- R er et relasjonsnavn
- A_x er et attributt
- D_y er et domene

INSERT (FRA SQLs DML)

- **insert**: Innsetting av nye data

insert into R„ (A_1, A_2, …, A_k)…
values (v_1, v_2, …, v_k..);
Attributtlisten kan sløyfes hvis den dekker samtlige attributter i *R* og følger attributtenes default rekkefølge

SQLs DQL - DATA QUERY LANGUAGE

select [**distinct**] ATTRIBUTTLISTE
from NAVNELISTE
[**where** WHERE-BETINGELSE]
[**group by** GRUPPERINGSATTRIBUTTER
[**having** HAVING-BETINGELSE]]
[**order by** ATTRIBUTT [**asc** | **desc**]
[, ATTRIBUTT [**asc** | **desc**]] ...];

[ ] betyr at dette leddet er en valgfri del av setningen

SELECT-SETNINGENS ENKELTDELER

- **select**
  Angir hvilke attributter som skal vises i svaret (* = alle)
- **distinct**
  Fjerner flerforekomster (duplikater) av svartuplene
- **from**
  Navn på de relasjonene spørringen refererer til
- **where**
  Seleksjonsbetingelse (kan inneholde en eller flere join-betingelser)
- **group by, having**
  Angir grupperingsattributter til bruk ved aggregering og betingelser på resultatet av grupperingen
- **order by**
  Ordner tuplene i henhold til angitte kriterier (**desc** = descending)
- Merknader:
  - select (SQL) skiller ikke mellom store og små bokstaver, unntatt i tekststrenger
  - select beregner bager (med unntak av noen av operatorene)

  En bag er en mengde med tupler der samme tuppel kan forekomme flere ganger. (Like tupler fjernes eventuelt ved å bruke distinct.)

AGGREGERINGSFUNKSJONER

SQL har fem aggregeringsfunksjoner

navn	*virkning (returnerer)*
count	teller antall
min	finner minste verdi
max	finner største verdi
sum	summerer verdier
avg	finner gjennomsnitt av verdier

SELEKSJONS- OG JOIN-BETINGELSER

- **Seleksjonsbetingelser**:
  Eks: navn **like** 'joakim'
  % matcher en vilkårlig string
  _ matcher ett vilkårlig tegn
- **Joinbetingelser**:
  eks: A.id = B.id
  (**natural join**, etc)

WHERE-BETINGELSER

- { =*f* ; <; >; <=*f* ; =>*f* ; <>; **like**}
  (like er bare lov ved en konstant tekststreng)
- null-test: **is null**, **is not null**
- relasjonssammenlikning:

i SQL-2	*betyr*
exists R	at R har minst én forekomst
not exists R	at R ikke har noen forekomster
in R	$\in$ R
not in R	$\notin$ R
any R	en vilkårlig verdi i R
all R	alle verdier i R

WHERE vs. HAVING

- **where**-betingelsen velger ut de tuplene som skal danne datagrunnlaget for grupperingen
- **having**-betingelsen plukker ut de tuplene fra det ferdig-grupperte resultatet som skal med i det endelige svaret
- **having**-betingelsen kan inneholde aggregatfunksjoner, men det kan ikke where-betingelsen

- Siden having håndterer en (mye) mindre datamengde enn where, er det i kompliserte spørringer lurt å legge så mye som mulig av logikken inn i having-betingelsen

DATO OG TIDSPUNKTER

- Dato: **date** 'yyyy-mm-dd'
- Tidspunkt: **time** 'hh:mm', **time** 'hh:mm:ss'
- Tidspunkt med finere gradering enn sekund:
  time 'hh:mm:ss.ccc'
- Tidspunkt før GMT: **time** 'hh:mm:ss+hh'
- Tidspunkt etter GMT: **time** 'hh:mm:ss-hh'
- Dato og tid: **timestamp** 'yyyy-mm-dd hh:mm:ss'

ALIASER

- **as** → brukes til å navngi relasjoner med aliaser
  eks: firstname || '_' || lastname **as** name

SUB-QUERIES

- Det er lov å ha sub-queries (indre select-setninger) i:
  - **from**-klausulen
  - **where**-klausulen
  - **having**-klausulen
- SQL-standarden inneholder ingen øvre grense for antall sub-queries i et query
- Sub-queries skal alltid være omsluttet av parenteser

NESTEDE SPØRSMÅL

- Gitt tabellen Ansatt(anr, navn, lonn, avd)
  Finn antall ansatte ved Ifi som tjener mer enn det dobbelte av gjennomsnittslønna i administrasjonen

```
select count (*)
from  Ansatt
where avd = 'ifi' and
    lonn > ( select 2 * avg(lonn)
             from Ansatt
             where avd = 'adm');
```

  Merk: En select inne i where-betingelsen må være omsluttet av parenteser

VIEWS

- Et view er en tenkt relasjon som vi bruker som mellomresultat i kompliserte SQL-beregninger
- Det er to måter å lage view på:
    - **create view** navn(attributtliste) as **select** ...
    - **create view** navn as **select** ...
- I det første tilfellet må det være like mange navn i attributtlisten som det er attributter i **select**-setningen
- I det andre tilfellet arver viewet attributtnavnene fra **select**-setningen
- Når man har laget et view, kan det brukes som en vanlig tabell i senere select-setninger
- Eks:
  Prosjektplan(pnr, anr, timer)

```
create view Innsats as
            select   anr, sum(timer) as timer
            from     Prosjektplan
            group by anr;


create view Bonus(anr, bonusbeløp) as
            (select  anr, 3000
            from     Innsats
            where    timer >= 50 )
       union
            (select  anr, 1500
             from    Innsats
             where   timer >= 15 and timer < 50 );
```

HENGETUPLER

- Når vi joiner to tabeller, kaller vi et tuppel som ikke har noen match i den andre relasjonen, et hengetuppel
- Hengetupler blir ikke med i resultatet av en (vanlig) join, også kalt en inner join
- Hvis vi ønsker å gjøre en join hvor vi beholder hengetuplene fra en eller begge tabellene, bruker vi en outer join

LEFT OUTER JOIN

- Syntaks for en **left outer join** er slik:

```
select <svar-attributter>
from   tabell-1 left outer join tabell-2
               on join-betingelse;
```

- Resultatet blir en join av tabell-1 og tabell-2, pluss en linje for hvert hengetuppel i tabell-1 der alle svar-attributtene fra tabell-2 er **null**
- Eventuelle hengetupler fra tabell-2 blir ikke med i resultatet

RIGHT OUTER JOIN

- Syntaks for en **right outer join** er slik:

```
select <svar-attributter>
from   tabell-1 right outer join tabell-2
               on join-betingelse;
```

- Resultatet blir en join av tabell-1 og tabell-2, pluss en linje for hvert hengetuppel i tabell-2 der alle svar-attributtene fra tabell-1 er **null**
- Eventuelle hengetupler fra tabell-1 blir ikke med i resultatet

FULL OUTER JOIN

- Syntaks for en **full outer join** er slik:

```
select <svar-attributter>
from   tabell-1 full outer join tabell-2
               on join-betingelse;
```

- Resultatet blir en join av tabell-1 og tabell-2, pluss en linje for hvert hengetuppel i tabell-1 der alle svar-attributtene fra tabell-2 er null og en linje for hvert hengetuppel i tabell-2 der alle svar-attributtene fra tabell-1 er **null**

OPERATORER

Unionkompabilitet:
R og S må ha like mange attributter og attributtene må parvis ha identiske domener. Eks:
Ansatt(navn,tlf,adr,aid) ∪ Medlem(mid,navn,adr,tlf)

UNION

- For at R ∪ S skal være definert, må R og S være *unionkompatible*
- Eksempel på vanlig mengdeunion:
  {a, b, c} ∪ {b, c, d, e} = {a, b, c, d, e}
- Eksempel på vanlig bagunion:
  {a, a, b, c, c} ∪ {a, c, c, c, d} = {a, a, a, b, c, c, c, c, c, d}

SNITT

- For at R ∩ S skal være definert, må R og S være *unionkompatible*
- Eksempel på vanlig mengdesnitt:
  {a, b, c} ∩ {b, c, d, e} = {b, c}

Eksempel på vanlig bagunion:
{a, a, b, c, c} ∩ {a, c, c, c, d} = {a, c, c}

DIFFERANSE

- For at R\S skal være definert, må R og S være *unionkompatible*
- Eksempel på vanlig mengdedifferanse:
  {a, b, c} \ {b, c, d, e} = {a}
- Eksempel på vanlig bagdifferanse:
  {a, a, b, c, c} \ {a, c, c, c, d} = {a, b}

SELEKSJON

- σ_C („ R)…er relasjonen som fås fra R ved å velge ut de tuplene i R som tilfredsstiller betingelsen C
- C er et vilkårlig boolsk uttrykk bygget opp fra atomer på formen $\mathbf{op}_1$ Θ $\mathbf{op}_2$ der operandene $\mathbf{op}_1$ og $\mathbf{op}_2$ er
  - enten to attributter i R med samme domene
  - eller ett atributt i R og en konstant fra dette attributtets domene
- operatoren Θ ∈ {**=**, **≠**, **<**, **>**, **<=**, **>=**, **like**}
- Bagseleksjon:
  - Hvis R er en bagrelasjon, er σ_C „ (R)…bagrelasjonen som fås fra R ved å anvende C på hvert enkelt tuppel individuelt og velge ut de tuplene i R som tilfredsstiller betingelsen C

PROJEKSJON

- π_L („ R)…hvor R er en relasjon og L er en liste av attributter i R, er relasjonen som fås fra R ved å velge ut kolonnene til attributtene i L
- Relasjonen har et skjema med attributtene i L
- Ingen tupler skal forekomme flere ganger i π_L„ (R…)
- bagprojeksjon:
  - Hvis R er en bagrelasjon og L er en (ikketom) liste av attributter, er π_L („ R)… bagrelasjonen som fås fra R ved å velge ut kolonnene til attributtene i L
  - π_L („ R)…har like mange tupler som R

RENAVNING

- $\rho_{S(„ A1, A2, …,An)}$ (…„ R)…renavner R til en relasjon med navn S og attributter A_1, A_2, …, A_n
- ρ_S (R…) renavner R til en relasjon med navn S
  Attributtnavnene fra R beholdes

DIVISJON

- Gitt to relasjoner
  $R(\underline{A_1, A_2, \ldots, A_n}, B_1, B_2, \ldots, B_m)$…og
  $S(\underline{B_1, B_2, \ldots, B_m})$
- R **div** S er en relasjon $Q(\underline{A_1, A_2, \ldots, A_n})$…som inneholder et tuppel *t* hvis og bare hvis det for hvert eneste tuppel *u* i S, fins et tuppel *v* i R slik at
  $\square_{A1,A2,\ldots,An}(\underline{v})$…= *t* og $\square_{B1,B2,\ldots,Bm}(v)$…= *f u*
- Hvis vi lar *tu* betegne sammensetningen av to tupler *t* og *u* til ett tuppel, skal altså *t* være med i svaret når vi for hver eneste *u* i S har at *tu* er med i R

SQLs DML - DATA MANIPULATION LANGUAGE

INSERT

insert into $R(\underline{A_1, A_2, \ldots, A_k})$…
values $(v_1, v_2, \ldots, v_k)$;

insert into $R(A_1, A_2, \ldots, A_k)$…
select-setning;

- Attributtlisten kan sløyfes hvis den dekker samtlige attributter i *R* og følger attributtenes default rekkefølge
- **NB**—optimaliseringer i DBMSet kan medføre at tuplene legges inn etterhvert som de beregnes i **select**-setningen. Dette kan ha sideeffekter på beregningen av **select**-setningen

UPDATE, DELETE

update R
set $A_1 = f\ E_1, \ldots, A_k\ f = E_k$
[**where** C];

delete from R
[**where** C];

- *R* er en relasjon, A_i er attributter (kolonnenavn) og E_j uttrykk. [ ] betyr at dette leddet er en valgfri del av setningen

SQLs SDL - STORAGE DEFINITION LANGUAGE: INDEKSER

create index *X* on R(„ A_1, …, A_k)…;

drop index X;

Valg av indekser må gjøres med omhu.
Indekser gjør at

- spørringer mot vedkommende attributt(er) går mye fortere
- innsetting, sletting og oppdatering blir mer komplisert

IMPLEMENTASJON

- Implementasjon av indekser:
    - B+-trær
    - Hash
    - …

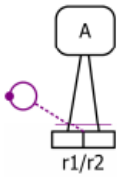
INDEKSER PÅ KANDIDATNØKLER

- DBMSer bygger vanligvis indekser automatisk på primærnøklene
- For hver kandidatnøkkel må man vurdere spesielt om det bør deklareres indeks på nøkkelen. Syntaks avhenger av DBMSet Noen SQL-implementasjoner tillater deklarasjon av kandidatnøkkel + indeks i en og samme setning:

    create unique index FnrIndex
    on Ansatt (Fdato , Pnr) ;
- **I Postgres bygges automatisk en unique index på kandidatnøkler**
- Hvis det er opprettet indeks på en nøkkel, benyttes denne under sjekk av flerforekomster. Ellers: Må i verste fall søke gjennom hele relasjonen

RINGSKRANKER

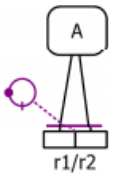
REFLEKSIV SKRANKE



$x \in pop(r1) \cup pop(r2) \Rightarrow (x,x) \in pop(r1,r2)$

r1	r2
a	b
a	a
b	b

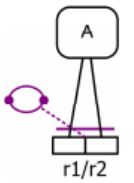
IRREFLEKSIV SKRANKE



$(x,x) \notin pop(r1,r2)$

r1	r2
a	b
a	a
b	b

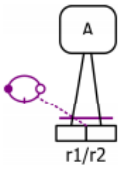
SYMMETRISKRANKE



$(x,y) \in pop(r1,r2) \Rightarrow (y,x) \in pop(r1,r2)$

r1	r2
a	b
b	a

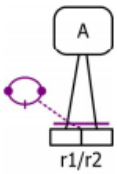
ANTISYMMETRISKRANKE



$(x,y) \in pop(r1,r2) \wedge x \neq y \Rightarrow (y,x) \notin pop(r1,r2)$

r1	r2
a	b
a	a
b	a

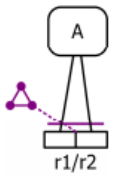
ASYMMETRISKRANKE



= antisymmetri + irrefleksivitet

r1	r2
a	b
b	a
a	a
b	b

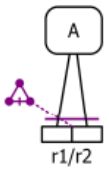
TRANSITIV SKRANKE



$(x,y) \in pop(r1,r2) \wedge (y,z) \in pop(r1,r2) \Rightarrow$
$(x,z) \in pop(r1,r2)$

r1	r2
a	b
b	c
a	c

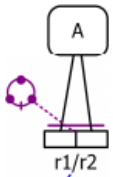
INTRANSITIV SKRANKE



$(x,y) \in pop(r1,r2) \wedge (y,z) \in pop(r1,r2) \Rightarrow$
$(x,z) \notin pop(r1,r2)$

r1	r2
a	b
b	c
a	c

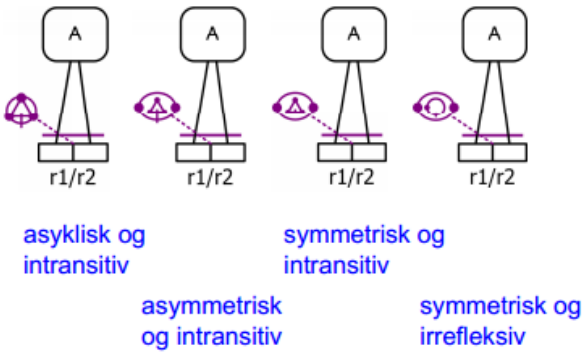
ASYKLISK SKRANKE



$(x_1,x_2) \in pop(r1,r2) \wedge \ldots \wedge (x_{n-1}, x_n) \in pop(r1,r2) \Rightarrow$
$(x_n, x_1) \notin pop(r1,r2)$

r1	r2
a	b
b	c
c	d
b	a
c	a
c	b
d	a
d	b
d	c

KOMBINERTE RINGSKRANKER



NORMALFORM

- Problem: Hvordan vurdere objektivt om en samling relasjoner er god/dårlig?
- **Normalformer** er et uttrykk for hvor godt vi har lykkes i en dekomposisjon
- At et skjema er på en normalform, sikrer at visse typer dobbeltlagring ikke forekommer
- Jo høyere normalform, jo mindre dobbeltlagring