IN2090 – databaser og datamodellering

Repetisjon: Normalformer og SQL

Mathias Stang og Stein Michael Storleer

21. november 2018

Agenda

Normalformer

- Funksjonelle avhengigheter
- Nøkler
- Finne hvilke normalformer relasjoner tilfredsstiller

SQL

- CREATE TABLE, VIEW, INSERT/UPDATE/DELETE
- Alle viktige delene av et SELECT-uttrykk
- JOINs
- Aggregering
- Subspørringer, strategier for å sette sammen komplekse spørringer
- Mengdeoperasjoner

Første normalform – 1NF

- **Definisjon**: Relasjonen inneholder kun atomære verdier
- Atomære verdier er verdier «vi ikke kan dele opp»

- Er «adresse» atomær, eller kan vi dele den opp i postnummer/sted?
- Er en string atomær, eller kan vi dele den opp i flere chars?

- Egentlig mener vi med atomære verdier at «for vår datamodell, er dette en verdi uten bestanddeler vi bryr oss om»
- Alle relasjoner vi har jobbet med i IN2090 har vært på 1NF!

Relasjoner som bryter 1NF

filmid	genre	title
85908	Action, Sci-Fi, Thriller	The Matrix
26103	Action, Sci-Fi	Planet of the Apes
1320611	Action, Sci-Fi	Planet of the Apes

Bryter 1NF fordi *genre* er en liste av verdier

Hvert element i *genre* kan altså skilles ut som hver sin verdi

Variant som tilfredsstiller 1NF

filmid	genre	title
85908	Action	The Matrix
85908	Sci-Fi	The Matrix
85908	Thriller	The Matrix
26103	Action	Planet of the Apes
26103	Sci-Fi	Planet of the Apes
1320611	Action	Planet of the Apes
1320611	Sci-Fi	Planet of the Apes

Funksjonelle avhengigheter (FD-er)

- Hvis verdiene i ett attributt **A** er bestemt av verdiene i et annet attributt **X**, har vi en **funksjonell avhengighet** X → A
- Vi sier «X bestemmer A», eller «A er avhengig av X»

- Eksempel: Person(<u>pid</u>, postnr, poststed)

  FD: **postnr → poststed**

  Fordi vi med et gitt postnummer alltid kan avgjøre hva poststed er
  (postnr 0373 er alltid i Oslo, 7491 alltid Trondheim)

- I tillegg:
  pid → postnr
  pid → poststed

  Fordi med en gitt primærnøkkel, kan vi bestemme de andre verdiene

- Men vi har IKKE en FD posted → postnr, fordi med poststed «Oslo», så kan det finnes flere mulige postnumre (0373, 0130, 0010 osv).

Nøkler

- **Supernøkkel**: En mengde attributter der verdiene i disse attributtene er unike (distinkte) i en relasjon
- **Kandidatnøkkel**: En *minimal* mengde attributter der verdiene er unike. Dvs at ved å fjerne et attributt, er det ikke lenger noen supernøkkel
- **Primærnøkkel**: En spesielt utpekt kandidatnøkkel der ingen verdier kan være NULL
- **Nøkkelattributt**: Attributter som er med i en kandidatnøkkel

Se forelesning 26. sep og 24. okt for flere detaljer.
+ ukeoppgaver uke 6 og 10

Eksempel:

Filmgenre(<u>filmid, genre</u>, title)

filmid	genre	title
85908	Action	The Matrix
85908	Sci-Fi	The Matrix
85908	Thriller	The Matrix
26103	Action	Planet of the Apes
26103	Sci-Fi	Planet of the Apes
1320611	Action	Planet of the Apes
1320611	Sci-Fi	Planet of the Apes

Supernøkler: {filmid, genre},
{filmid, genre, title}

Kandidatnøkkel: {filmid, genre}

Primærnøkkel: {filmid, genre}

Nøkkelattributter: filmid og genre

Normalformer og FD-er

- For normalformene 2NF, 3NF og BCNF er det FD-ene vi ser på
- Gitt en relasjon R, med en mengde FD-er på formen X → A, der X og A er en mengde attributter, kan vi avgjøre normalformen
- Eksempel: Relasjonen R er følgende relasjon:
  Person(<u>pid</u>, postnr, poststed)
- FD-ene på formen X → A:
  - postnr → poststed *(X er postnr, A er poststed)*
  - pid → postnr *(X er pid, A er postnr)*
  - pid → poststed *(X er pid, A er postnr)*

Normalformene

Gitt en relasjon R, med en mengde FD-er på formen $X \rightarrow A$,
der X og A er en mengde attributter

1NF:

- R inneholder kun atomære verdier/attributter

2NF:

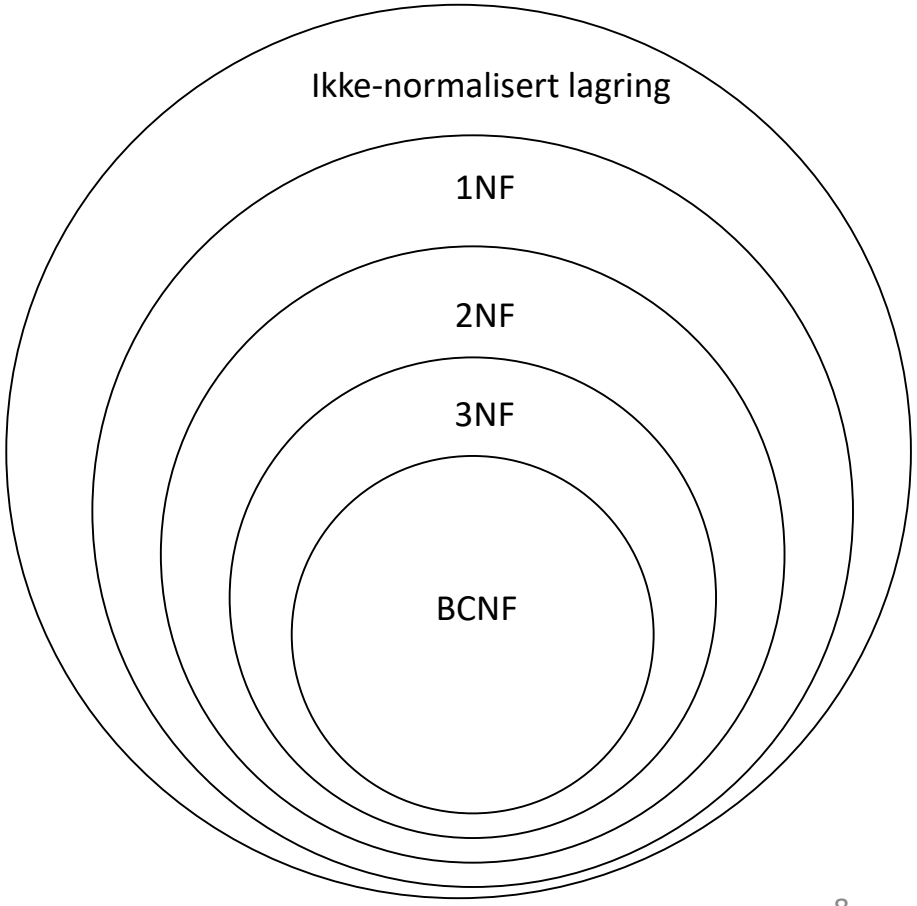
- X er en supernøkkel i R, eller
- A er et nøkkelattributt, eller
- X er *ikke* en delmengde av noen nøkler i R

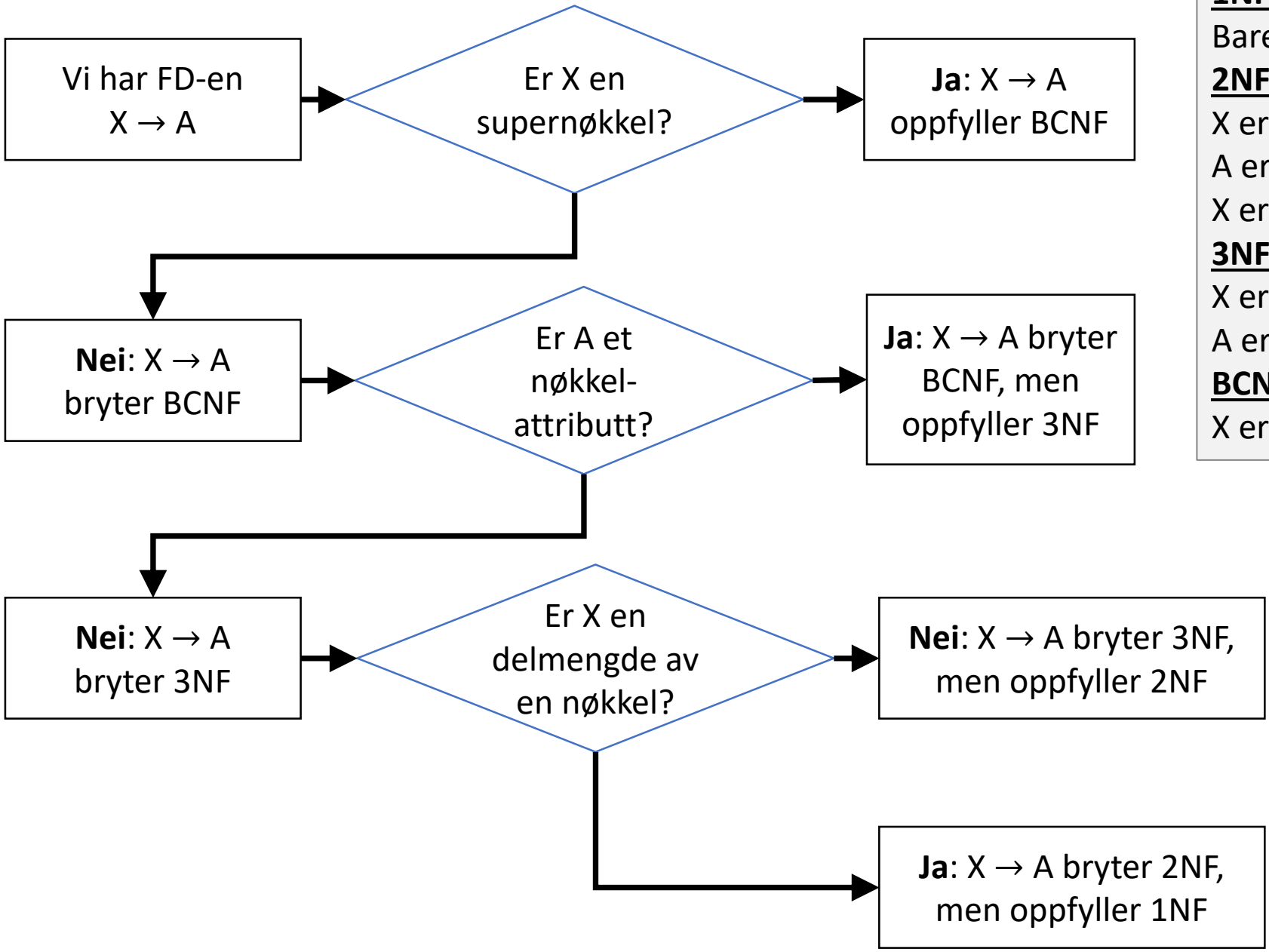
3NF:

- X er en supernøkkel i R, eller
- A er et nøkkelattributt

BCNF:

- X er en supernøkkel i R





1NF:
Bare atomære verdier/attributter
2NF:
X er en supernøkkel i R, eller
A er et nøkkelattributt, eller
X er *ikke* en delmengde av noen nøkler i R
3NF:
X er en supernøkkel i R, eller
A er et nøkkelattributt
BCNF:
X er en supernøkkel i R

Eksempel

pid
pid, postnr
pid, poststed
pid, postnr, poststed
} super-nøkkel

- Relasjon R: Person(<u>pid</u>, postnr, poststed)
- FD-ene på formen X → A:
  - postnr → poststed 2NF
  - pid → postnr BCNF
  - pid → poststed BCNF

- Avgjør normalformen for hver FD

- FD-en med den laveste normalformen gir oss normalformen til relasjonen 2NF

<u>1NF:</u>
Bare atomære verdier/attributter
<u>2NF:</u>
X er en supernøkkel i R, eller
A er et nøkkelattributt, eller
X er *ikke* en delmengde av noen nøkler i R
<u>3NF:</u>
X er en supernøkkel i R, eller
A er et nøkkelattributt
<u>BCNF:</u>
X er en supernøkkel i R

Eksempel 2

Student_emne(<u>id, emnekode</u>, bnavn, karakter)

FD-er:

- id, emnekode → bnavn, karakter
- bnavn, emnekode → id, karakter
- id → bnavn
- bnavn → id

~~id, emnekode → bnavn~~ triviell FD!

Hvilken normalform er relasjonen på?

<u>1NF:</u>
Bare atomære verdier/attributter
<u>2NF:</u>
X er en supernøkkel i R, eller
A er et nøkkelattributt, eller
X er *ikke* en delmengde av noen nøkler i R
<u>3NF:</u>
X er en supernøkkel i R, eller
A er et nøkkelattributt
<u>BCNF:</u>
X er en supernøkkel i R

Rettelser (grønt):
id, emnekode → bnavn er <u>IKKE</u> en triviell FD.

Kravet for en triviell FD er at gitt FD-en X → A, så kan ikke A være en delmengde av X.
Det ble sagt feil i opptaket.

Eksempel 3

Timeliste(<u>ansattnr, uke, år</u>, navn, timer)

FD-er:

- ansattnr, uke, år → navn, timer BCNF
- ansattnr → navn 1NF

Hvilken normalform er relasjonen på? 1NF

Flere oppgaver om normalformer: ukeoppgaver uke 12

1NF:
Bare atomære verdier/attributter
2NF:
X er en supernøkkel i R, eller
A er et nøkkelattributt, eller
X er *ikke* en delmengde av noen nøkler i R
3NF:
X er en supernøkkel i R, eller
A er et nøkkelattributt
BCNF:
X er en supernøkkel i R

SQL

CREATE TABLE

- CREATE TABLE
- Datatyper: text/varchar, int, date, time, char(n), boolean…
- PRIMARY KEY på én eller flere verdier
- NOT NULL
- UNIQUE (for kandidatnøkler)
- Fremmednøkler på én eller flere verdier
- CHECK

Flere detaljer i forelesning 26. sep

CREATE TABLE: Eksempel

Tliste(timelistenr, [status], [beskrivelse])
Tlistelinje(timelistenr, linjenr, timeantall, [beskrivelse], [kumulativt_timeantall])

```
CREATE TABLE tliste (
    timelistenr int primary key,
    status text check (status = 'aktiv' or status = 'levert' or status = 'utbetalt'),
    beskrivelse text
);

CREATE TABLE tlistelinje (
    timelistenr int references tliste(timelistenr),
    linjenr int,
    timeantall int not null,
    beskrivelse text unique,
    kumulativt_timeantall int,
    primary key(timelistenr, linjenr)
);
```

CREATE VIEW

- **`CREATE VIEW`** `navn` **`AS`** `(select …);`

- Et view lagrer ikke tabellen. I stedet lagrer vi bare spørringen, og hver gang vi ønsker å bruke et view i en spørring, kjøres spørringen til viewet

Flere detaljer i forelesning 17. okt

INSERT INTO

- **INSERT INTO** <tabell> **VALUES** (…)
- **INSERT INTO** <tabell> (feltnavn…) **VALUES** (…)
- **INSERT INTO** <tabell> **SELECT** … **FROM** …

UPDATE, DELETE

- **UPDATE** <tabell> **SET** kolonne1 = verdi, kolonne2 = v… **WHERE** …
- **DELETE FROM** <tabell> **WHERE** …

SELECT

SELECT [**DISTINCT**] [* | uttrykk [[**AS**] alias] [, ...]]
[**FROM** tabelluttrykk]
[**WHERE** betingelse]
[**GROUP BY** grupperingsattributt [, …]]
[**HAVING** betingelse]
[{ **UNION** | **INTERSECT** | **EXCEPT** } [**ALL**] select…]
[**ORDER BY** uttrykk [**ASC** | **DESC**]]
[**LIMIT** antall]
[**OFFSET** start]

Uttrykk i SELECT 1

- Uttrykket i SELECT er en liste av uttrykk som spesifiserer hvordan radene skal se ut
- Uttrykket kan inneholde attributter fra en tabell:
  `SELECT timelistenr, beskrivelse FROM tliste;`

timelistenr	beskrivelse
1	HMS
2	Test
3	Virksomhet

Uttrykk i SELECT 2

- Vi kan bruke statiske verdier:

```
SELECT timelistenr, 'Hei', 123, null FROM tliste;
```

timelistenr	?column?	?column?	?column?
1	Hei	123	*null*
2	Hei	123	*null*
3	Hei	123	*null*

- Vi kan spesifisere hva kolonnenavnet skal være:

```
SELECT timelistenr AS nr, 'Hei' AS hilsen,
   123 AS tall, null AS "ingenting her" FROM tliste;
```

nr	hilsen	tall	ingenting her
1	Hei	123	*null*
2	Hei	123	*null*
3	Hei	123	*null*

Uttrykk i SELECT 3

- Vi kan også kombinere disse med funksjonskall, sammensatte uttrykk osv:

```
SELECT timelistenr * 10 AS nr,
       'Beskr.: ' || beskrivelse,
       upper(beskrivelse)
FROM tliste;
```

nr	?column?	upper
10	Beskr.: HMS	HMS
20	Beskr.: Test	TEST
30	Beskr.: Virksomhet	VIRKSOMHET

Andre funksjoner

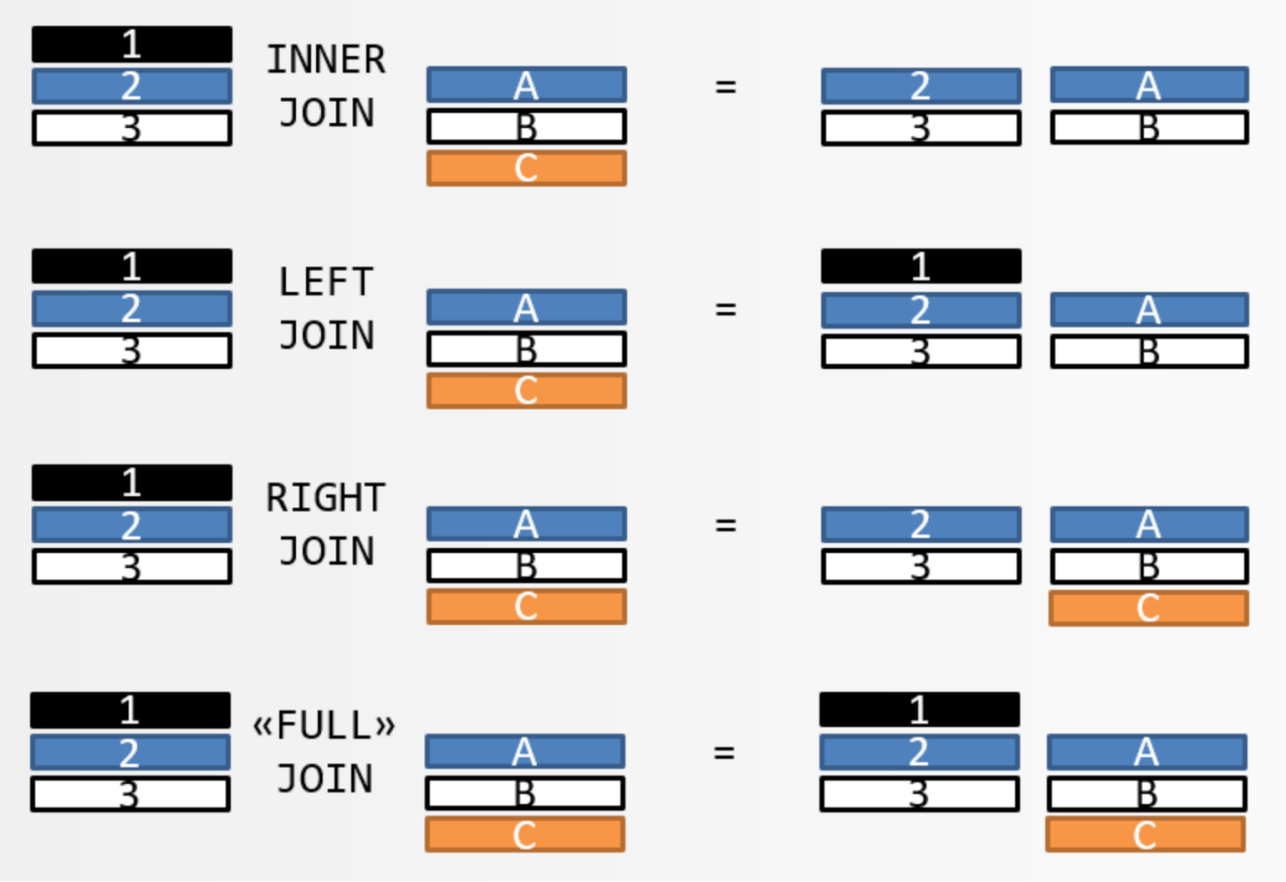
- Dere trenger ikke kunne funksjoner som SUBSTR, LOWER, UPPER
- **Men:** Dere må kunne bruke dem om funksjonen blir definert i oppgaveteksten

- **Eksempeloppgave:** Skriv ut alle **`firstname`** i tabellen **`person`** med store bokstaver
  Tilleggsinformasjon: Funksjonen `upper(string)` gjør alle bokstaver i en string om til store bokstaver (f.eks. gir `upper('Ola')` resultatet `OLA`)

- Da må dere kunne skrive løsningen:
  ```
  SELECT upper(firstname) FROM person;
  ```

- Unntak: Dere må kunne aggregeringsfunksjoner og stringkonkatenering (enten med operatoren || eller funksjonen CONCAT)

JOIN


1
2
3
INNER JOIN
A
B
C
=
2
3
A
B
1
2
3
LEFT JOIN
A
B
C
=
1
2
3
A
B
1
2
3
RIGHT JOIN
A
B
C
=
2
3
A
B
C
1
2
3
«FULL» JOIN
A
B
C
=
1
2
3
A
B
C

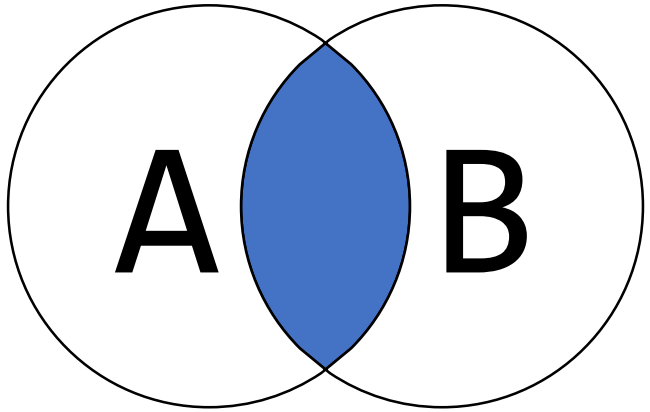
[INNER] JOIN

A

id	navn
1	Martin
2	**Ida**
3	Hanne
4	**Ole**

B

id	navn
1	**Ida**
2	Mathias
3	**Ole**
4	Øyvind




```
SELECT *
FROM A INNER JOIN B
     ON A.navn = B.navn;
```

A.id	A.navn	B.id	B.Navn
2	Ida	1	Ida
4	Ole	3	Ole

Vi får bare med rader der betingelsen slår til for både tabell A og tabell B.

LEFT [OUTER] JOIN

A

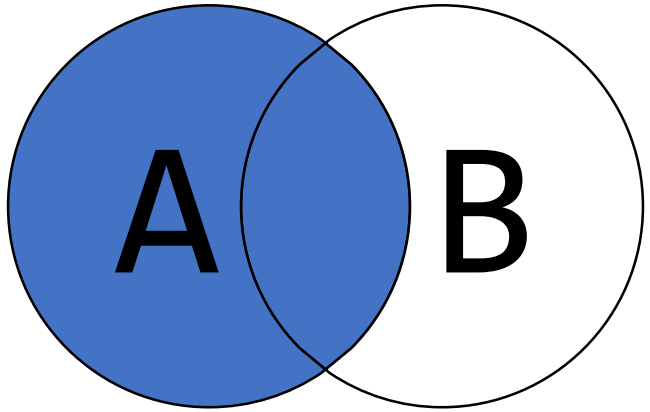
id	navn
1	Martin
2	**Ida**
3	Hanne
4	**Ole**

B

id	navn
1	**Ida**
2	Mathias
3	**Ole**
4	Øyvind

```
SELECT *
FROM A LEFT JOIN B
     ON A.navn = B.navn;
```

A.id	A.navn	B.id	B.Navn
1	Martin	*null*	*null*
2	Ida	1	Ida
3	Hanne	*null*	*null*
4	Ole	3	Ole



Vi tar med alle rader fra tabell A, samt radene fra tabell B som matcher betingelsen

RIGHT [OUTER] JOIN

A

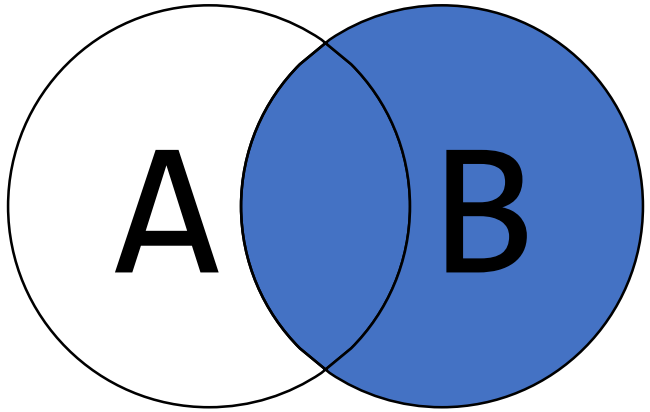
id	navn
1	Martin
2	**Ida**
3	Hanne
4	**Ole**

B

id	navn
1	**Ida**
2	Mathias
3	**Ole**
4	Øyvind

```
SELECT *
FROM A RIGHT JOIN B
     ON A.navn = B.navn;
```

A.id	A.navn	B.id	B.Navn
2	Ida	1	Ida
null	*null*	2	Mathias
4	Ole	3	Ole
null	*null*	4	Øyvind



Motsatt av LEFT JOIN:
Vi tar med alle rader fra tabell B, samt radene fra tabell A som matcher betingelsen

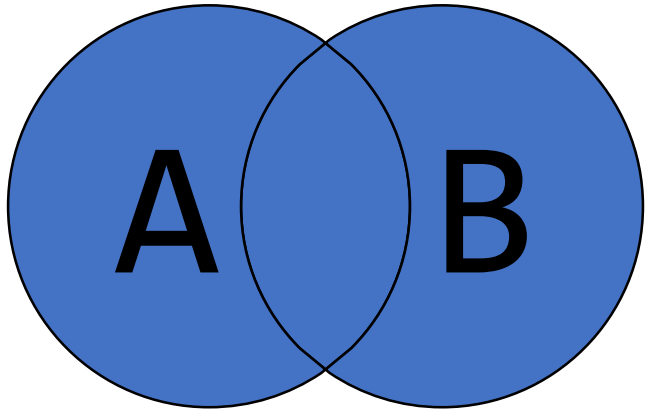
FULL [OUTER] JOIN

A

id	navn
1	Martin
2	**Ida**
3	Hanne
4	**Ole**

B

id	navn
1	**Ida**
2	Mathias
3	**Ole**
4	Øyvind




```
SELECT *
FROM A FULL OUTER JOIN B
     ON A.navn = B.navn;
```

A.id	A.navn	B.id	B.Navn
1	Martin	*null*	*null*
2	Ida	1	Ida
3	Hanne	*null*	*null*
4	Ole	3	Ole
null	*null*	2	Mathias
null	*null*	4	Øyvind

Vi tar med alle rader fra tabell A og B. Der betingelsen matcher, har radene data fra begge. Hvis det ikke er noen match, blir dataene på manglende side satt til *null*.

CROSS JOIN

A

id	navn
1	Martin
2	**Ida**
3	Hanne
4	**Ole**

B

id	navn
1	**Ida**
2	Mathias
3	**Ole**
4	Øyvind

Gir oss kartesisk produkt.

Vi kan etterpå filtrere resultatene, f.eks:

```
SELECT *
FROM TabellA, TabellB;
WHERE TabellA.navn = TabellB.navn;
```

(implisitt join, tilsvarer INNER JOIN)

```
SELECT *
FROM TabellA CROSS JOIN TabellB;
```

Gir samme resultat som:

```
SELECT *
FROM TabellA, TabellB;
```

A.id	A.navn	B.id	B.Navn
1	Martin	1	Ida
1	Martin	2	Mathias
1	Martin	3	Ole
1	Martin	4	Øyvind
2	Ida	1	Ida
2	Ida	2	Mathias
2	Ida	3	Ole
2	Ida	4	Øyvind
3	Hanne	1	Ida
3	Hanne	2	Mathias
3	Hanne	3	Ole
3	Hanne	4	Øyvind

NATURAL JOIN

A

id	navn
1	Martin
2	**Ida**
3	Hanne
4	**Ole**

B

id	navn
1	**Ida**
2	Mathias
3	**Ole**
4	Øyvind

```
SELECT *
FROM TabellA NATURAL JOIN TabellB;
```

id	navn

(0 rader)

Det finnes ingen rader i A og B der både navn og id er like

NATURAL JOIN

A

nr	navn
1	Martin
2	**Ida**
3	Hanne
4	**Ole**

B

id	navn
1	**Ida**
2	Mathias
3	**Ole**
4	Øyvind

```
SELECT *
FROM TabellA NATURAL JOIN TabellB;
```

A.nr	B.id	Navn
2	1	Ida
4	3	Ole

Nesten identisk med INNER JOIN

MEN: Merk antall kolonner i resultatet!

Operatorer i betingelser (WHERE, HAVING)

- `>  <  >=  <=  !=  <>  =`
- `+  *  -  /`
- `AND, OR`
- `LIKE` med wildcards `%` og `_`
- `IS NULL, IS NOT NULL`
- `[NOT] IN (a, b, c)`
- `[NOT] IN (SELECT …)`
- `[NOT] EXISTS (SELECT …)`

Også kjekt å vite om:

- `BETWEEN x AND y`
- `ANY` og `ALL`

GROUP BY og aggregering

- Dere må kunne aggregeringsfunksjonene min, max, avg, sum, count.
- Forskjellen på `count(*)`, `count(attr)` og `count(distinct attr)`
- Forskjellen på WHERE og HAVING
- Gruppere på ett eller flere attributer

For flere detaljer, se forelesning 19. sep

Subspørringer

Vi kan generelt sette inn subspørringer 3 steder:

- Der vi vanligvis skriver en tabell:

```
SELECT * FROM tabell a;
SELECT * FROM (SELECT x, y FROM tabell) a;
```

- I stedet for en liste, kan vi lage en subspørring som returnerer 1 kolonne:

```
SELECT * FROM tabell WHERE attr IN (1, 2, 3);
SELECT * FROM tabell WHERE attr IN (SELECT tall FROM …);
```

- I stedet for en verdi, kan vi lage en subspørring som returnerer 1 kolonne med 1 rad:

```
SELECT * FROM tabell WHERE rating = 9;
SELECT * FROM tabell WHERE rating = (SELECT MAX(…) FROM t);
```

EXISTS vs IN

- EXISTS bruker vanligvis korrelerte subspørringer

Bruk av WITH

- I stedet for å bruke views, kan man bruke WITH på lignende måte:

```
WITH egensporring AS (SELECT …)
SELECT * FROM egensporring …
```

Mengdeoperasjoner

- UNION [ALL] → viktig
- INTERSECT
- EXCEPT

} kan erstattes med andre ting

- Mengde- vs bagoperasjoner