

IN2090 – Databaser og datamodellering

06 – Enkele joins og nestede SELECT

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

- ◆ (Enkle) **SELECT**-spørringer har formen:

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

- ◆ (Enkle) **SELECT**-spørringer har formen:

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

- ◆ hvor <kolonner> er en liste med (uttrykk over) kolonne-navn,
- ◆ <tabeller> er en liste med tabell-navn
- ◆ <uttrykk> er et uttrykk over kolonne-navn som evaluerer

Enkle SELECT-spørringer i et nøtteskall

- ◆ `FROM`-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **WHERE**-klausulen velger ut hvilke rader som skal være med i svaret

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **WHERE**-klausulen velger ut hvilke rader som skal være med i svaret
 - ◆ Kolonnenavn brukes som variable som instansieres med radenes verdier

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **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**

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **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** og **OR** for å kombinere og paranteser for å gruppere uttrykk

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **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** og **OR** for å kombinere og paranteser for å gruppere uttrykk
 - ◆ Evaluerer til enten **TRUE**, **FALSE** eller **NULL** for hver rad

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **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** og **OR** for å kombinere og paranteser for å gruppere uttrykk
 - ◆ Evaluerer til enten **TRUE**, **FALSE** eller **NULL** for hver rad
 - ◆ Kun de som evaluerer til **TRUE** blir med i svaret

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **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** og **OR** for å kombinere og paranteser for å gruppere 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

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **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** og **OR** for å kombinere og paranteser for å gruppere 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.

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **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** og **OR** for å kombinere og paranteser for å gruppere 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

Enkle SELECT-spørringer i et nøtteskall

- ◆ **FROM**-klausulen sier hvilke tabell(er) som skal brukes for å besvare spørringen
- ◆ **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** og **OR** for å kombinere og paranteser for å gruppere 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

Eksempler: FilmDB

Finn alle Star Trek filmer [118 rader]

```
SELECT title
  FROM film
 WHERE title LIKE '%Star Trek%';
```

Finn fult navn på alle kvinner [462897 rader]

```
SELECT DISTINCT firstname || ' ' || lastname AS name
  FROM person
 WHERE firstname IS NOT NULL
       AND gender = 'F';
```

Kombinere informasjon fra flere tabeller

- ◆ Frem til nå har vi bare sett på spørringer over én og én tabell

Kombinere informasjon fra flere tabeller

- ◆ Frem til nå har vi bare sett på spørringer over én og én tabell
- ◆ Ofte ønsker vi å kombinere informasjon fra ulike tabeller

Kombinere informasjon fra flere tabeller

- ◆ Frem til nå har vi bare sett på spørringer 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

Spørringer over flere tabeller

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

To tabeller i **FROM**

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

Resultat

Spørringer 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

Spørringer over flere tabeller

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

To tabeller i **FROM**

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

Resultat

ProductID	ProductName	Price	OrderID	OrderedProduct	Customer
0	TV 50 inch	8999	0	1	John Mill
0	TV 50 inch	8999	1	1	Peter Smith
0	TV 50 inch	8999	2	0	Anna Consuma
0	TV 50 inch	8999	3	1	Yvonne Potter
1	Laptop 2.5GHz	7499	0	1	John Mill
1	Laptop 2.5GHz	7499	1	1	Peter Smith
1	Laptop 2.5GHz	7499	2	0	Anna Consuma
1	Laptop 2.5GHz	7499	3	1	Yvonne Potter

Spørringer 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

Spørringer over flere tabeller

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

To tabeller i **FROM**

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

Resultat – Fargekodet

ProductID	ProductName	Price	OrderID	OrderedProduct	Customer
0	TV 50 inch	8999	0	1	John Mill
0	TV 50 inch	8999	1	1	Peter Smith
0	TV 50 inch	8999	2	0	Anna Consuma
0	TV 50 inch	8999	3	1	Yvonne Potter
1	Laptop 2.5GHz	7499	0	1	John Mill
1	Laptop 2.5GHz	7499	1	1	Peter Smith
1	Laptop 2.5GHz	7499	2	0	Anna Consuma
1	Laptop 2.5GHz	7499	3	1	Yvonne Potter

Kryssprodukt

- ◆ Med flere tabeller i **FROM**-klausulen får vi alle mulige kombinasjoner av radene fra hver tabell

Kryssprodukt

- ◆ Med flere tabeller i **FROM**-klausulen får vi alle mulige kombinasjoner av radene fra hver tabell
- ◆ Dette kalles *kryssproduct* eller *Kartesisk produkt*

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

- ◆ 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
- ◆ Med to tabeller med

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
- ◆ Med to tabeller med
 - ◆ c_1 (f.eks. 3) og c_2 (f.eks. 5) antall kolonner, og

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
- ◆ Med to tabeller med
 - ◆ c_1 (f.eks. 3) og c_2 (f.eks. 5) antall kolonner, og
 - ◆ r_1 (f.eks. 32) og r_2 (f.eks. 74) antall rader,

Kryssprodukt

- ◆ Med flere tabeller i **FROM**-klausulen får vi alle mulige kombinasjoner av radene fra hver tabell
- ◆ Dette kalles *kryssprodukt* eller *Kartesisk produkt*
- ◆ Altså, det som var \times i relasjonsalgebraen
- ◆ Med to tabeller med
 - ◆ c_1 (f.eks. 3) og c_2 (f.eks. 5) antall kolonner, og
 - ◆ r_1 (f.eks. 32) og r_2 (f.eks. 74) antall rader,
 - ◆ så vil kryssproduktet bli en tabell med $c_1 + c_2$ kolonner (f.eks. $3 + 5 = 8$) og $r_1 \times r_2$ rader (f.eks. $32 \times 74 = 2368$).

Kryssprodukt

- ◆ Med flere tabeller i **FROM**-klausulen får vi alle mulige kombinasjoner av radene fra hver tabell
- ◆ Dette kalles *kryssprodukt* eller *Kartesisk produkt*
- ◆ Altså, det som var \times i relasjonsalgebraen
- ◆ Med to tabeller med
 - ◆ c_1 (f.eks. 3) og c_2 (f.eks. 5) antall kolonner, og
 - ◆ r_1 (f.eks. 32) og r_2 (f.eks. 74) antall rader,
 - ◆ så vil kryssproduktet bli en tabell med $c_1 + c_2$ kolonner (f.eks. $3 + 5 = 8$) og $r_1 \times r_2$ rader (f.eks. $32 \times 74 = 2368$).
- ◆ Med tre tabeller

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
- ◆ Med to tabeller med
 - ◆ c_1 (f.eks. 3) og c_2 (f.eks. 5) antall kolonner, og
 - ◆ r_1 (f.eks. 32) og r_2 (f.eks. 74) antall rader,
 - ◆ så vil kryssproduktet bli en tabell med $c_1 + c_2$ kolonner (f.eks. $3 + 5 = 8$) og $r_1 \times r_2$ rader (f.eks. $32 \times 74 = 2368$).
- ◆ Med tre tabeller
 - ◆ c_1 , c_2 , og c_3 antall kolonner, og

Kryssprodukt

- ◆ Med flere tabeller i **FROM**-klausulen får vi alle mulige kombinasjoner av radene fra hver tabell
- ◆ Dette kalles *kryssprodukt* eller *Kartesisk produkt*
- ◆ Altså, det som var \times i relasjonsalgebraen
- ◆ Med to tabeller med
 - ◆ c_1 (f.eks. 3) og c_2 (f.eks. 5) antall kolonner, og
 - ◆ r_1 (f.eks. 32) og r_2 (f.eks. 74) antall rader,
 - ◆ så vil kryssproduktet bli en tabell med $c_1 + c_2$ kolonner (f.eks. $3 + 5 = 8$) og $r_1 \times r_2$ rader (f.eks. $32 \times 74 = 2368$).
- ◆ Med tre tabeller
 - ◆ c_1, c_2 , og c_3 antall kolonner, og
 - ◆ r_1, r_2 og r_3 antall rader

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
- ◆ Med to tabeller med
 - ◆ c_1 (f.eks. 3) og c_2 (f.eks. 5) antall kolonner, og
 - ◆ r_1 (f.eks. 32) og r_2 (f.eks. 74) antall rader,
 - ◆ så vil kryssproduktet bli en tabell med $c_1 + c_2$ kolonner (f.eks. $3 + 5 = 8$) og $r_1 \times r_2$ rader (f.eks. $32 \times 74 = 2368$).
- ◆ Med tre tabeller
 - ◆ c_1, c_2 , og c_3 antall kolonner, og
 - ◆ r_1, r_2 og r_3 antall rader
 - ◆ vil kryssproduktet bli en tabell med $c_1 + c_2 + c_3$ kolonner og $r_1 \times r_2 \times r_3$ rader.

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

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

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

Eksempel spørring med flere tabeller

Hvilken kunde har kjøpt hvilket produkt?

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

Resultat

ProductID	ProductName	Price	OrderID	OrderedProduct	Customer
0	TV 50 inch	8999	0	1	John Mill
0	TV 50 inch	8999	1	1	Peter Smith
0	TV 50 inch	8999	2	0	Anna Consuma
0	TV 50 inch	8999	3	1	Yvonne Potter
1	Laptop 2.5GHz	7499	0	1	John Mill
1	Laptop 2.5GHz	7499	1	1	Peter Smith
1	Laptop 2.5GHz	7499	2	0	Anna Consuma
1	Laptop 2.5GHz	7499	3	1	Yvonne Potter

Eksempel spørring med flere tabeller

Hvilken kunde har kjøpt hvilket produkt?

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

Resultat

ProductID	ProductName	Price	OrderID	OrderedProduct	Customer
0	TV 50 inch	8999	0	1	John Mill
0	TV 50 inch	8999	1	1	Peter Smith
0	TV 50 inch	8999	2	0	Anna Consuma
0	TV 50 inch	8999	3	1	Yvonne Potter
1	Laptop 2.5GHz	7499	0	1	John Mill
1	Laptop 2.5GHz	7499	1	1	Peter Smith
1	Laptop 2.5GHz	7499	2	0	Anna Consuma
1	Laptop 2.5GHz	7499	3	1	Yvonne Potter

Eksempel spørring med flere tabeller

Hvilken kunde har kjøpt hvilket produkt?

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

Resultat

ProductID	ProductName	Price	OrderID	OrderedProduct	Customer
0	TV 50 inch	8999	0	1	John Mill
0	TV 50 inch	8999	1	1	Peter Smith
0	TV 50 inch	8999	2	0	Anna Consuma
0	TV 50 inch	8999	3	1	Yvonne Potter
1	Laptop 2.5GHz	7499	0	1	John Mill
1	Laptop 2.5GHz	7499	1	1	Peter Smith
1	Laptop 2.5GHz	7499	2	0	Anna Consuma
1	Laptop 2.5GHz	7499	3	1	Yvonne Potter

Eksempel spørring med flere tabeller

Hvilken kunde har kjøpt hvilket produkt?

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

Resultat

ProductName	Customer
TV 50 inch	Anna Consuma
Laptop 2.5GHz	John Mill
Laptop 2.5GHz	Peter Smith
Laptop 2.5GHz	Yvonne Potter

Joins

- ◆ Spørringer over flere tabeller kalles *joins*,

Joins

- ◆ Spørringer over flere tabeller kalles *joins*,
- ◆ Mange måter å relatere tabeller på, altså mange mulige joins, f.eks.

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

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

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
```

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
```

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 <theta> er en eller annen relasjon (f.eks. <, =, !=, [LIKE](#)) eller mer komplisert uttrykk

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 <theta> er en eller annen relasjon (f.eks. <, =, !=, [LIKE](#)) eller mer komplisert uttrykk

- ◆ *Equi-join* er en spesiell type *Theta-join*

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 <theta> er en eller annen relasjon (f.eks. <, =, !=, **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?

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

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

Problemer med spørring over flere tabeller

Hvilken kunde har kjøpt hvilket produkt?

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

Resultat

ProductID	ProductName	Price	OrderID	ProductID	Customer
0	TV 50 inch	8999	0	1	John Mill
0	TV 50 inch	8999	1	1	Peter Smith
0	TV 50 inch	8999	2	0	Anna Consuma
0	TV 50 inch	8999	3	1	Yvonne Potter
1	Laptop 2.5GHz	7499	0	1	John Mill
1	Laptop 2.5GHz	7499	1	1	Peter Smith
1	Laptop 2.5GHz	7499	2	0	Anna Consuma
1	Laptop 2.5GHz	7499	3	1	Yvonne Potter

Problemer med spørring over flere tabeller

Hvilken kunde har kjøpt hvilket produkt?

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

Resultat

ProductID	ProductName	Price	OrderID	ProductID	Customer
0	TV 50 inch	8999	0	1	John Mill
0	TV 50 inch	8999	1	1	Peter Smith
0	TV 50 inch	8999	2	0	Anna Consuma
0	TV 50 inch	8999	3	1	Yvonne Potter
1	Laptop 2.5GHz	7499	0	1	John Mill
1	Laptop 2.5GHz	7499	1	1	Peter Smith
1	Laptop 2.5GHz	7499	2	0	Anna Consuma
1	Laptop 2.5GHz	7499	3	1	Yvonne Potter

- ◆ Når vi har flere tabeller i samme spørring kan vi få flere kolonner med likt navn

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

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
- ◆ F.eks. `products.ProductID` og `orders.OrderID`

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
- ◆ F.eks. `products.ProductID` og `orders.OrderID`

Hvilken kunde har kjøpt hvilket produkt?

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
- ◆ F.eks. `products.ProductID` og `orders.OrderID`

Hvilken kunde har kjøpt hvilket produkt?

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

- ◆ Det er ofte nyttig å kunne gi en tabell et nytt navn

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

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)

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;
```

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

Spørring som finner navnene på alle par av kunder og leverandører som er i samme by [14 rader]

Eksempler på joins: Northwind-databasen

Spørring som finner navnene på alle par av kunder og leverandører som er i samme by [14 rader]

```
SELECT c.company_name, s.company_name  
FROM customers AS c, suppliers AS s  
WHERE c.city = s.city
```

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)

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;
```

- ◆ SQL-spørringene med joins kan også oversettes til relasjonsalgebra

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>
```


$$\pi_{\langle \text{columns} \rangle}(\sigma_{\langle \text{condition} \rangle}(\langle \text{t1} \rangle \times \langle \text{t2} \rangle \times \dots \times \langle \text{tN} \rangle))$$

Egen notasjon for joins

- ◆ SQL har en egen notasjon for joins

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

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
```

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
```

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

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

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

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]

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

Self-joins

- ◆ Av og til ønsker man å kombinere informasjon fra rader i samme tabell
- ◆ Dette kalles en *self-join*

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

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

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
.
.
.

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
.
.
.

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
.
.
.

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

P2.Name	P2.Price
TV 50 inch	8999
TV 48 inch	11999

Fler join eksempler: Fra Northwind-DB

Spørring som finner "høflighets-tittel" og jobbtittel på alle sjefer [2 rader]

Fler join eksempler: Fra Northwind-DB

Spørring som finner "høflighets-tittel" og jobbtittel på alle sjefer [2 rader]

```
SELECT DISTINCT e2.title_of_courtesy, e2.title
FROM employees AS e1 INNER JOIN employees AS e2
ON (e1.reports_to = e2.employee_id);
```

Naturlig Join

- ◆ Vi joiner ofte på de kolonnene som har likt navn

Naturlig Join

- ◆ Vi joiner ofte på de kolonnene som har likt navn
- ◆ F.eks. `categories.category_id` med `products.category_id`

Naturlig Join

- ◆ Vi joiner ofte på de kolonnene som har likt navn
- ◆ F.eks. `categories.category_id` med `products.category_id`
- ◆ Dette kan gjøres enklere med *naturlig join*

Naturlig Join

- ◆ Vi joiner ofte på de kolonnene som har likt navn
- ◆ F.eks. `categories.category_id` med `products.category_id`
- ◆ Dette kan gjøres enklere med *naturlig join*
- ◆ Naturlig join joiner (med likhet) automatisk på alle kolonner med likt navn

Naturlig Join

- ◆ Vi joiner ofte på de kolonnene som har likt navn
- ◆ F.eks. `categories.category_id` med `products.category_id`
- ◆ Dette kan gjøres enklere med *naturlig join*
- ◆ Naturlig join joiner (med likhet) automatisk på alle kolonner med likt navn
- ◆ I tillegg projiserer den vekk de dupliserte kolonnene

Naturlig Join

- ◆ Vi joiner ofte på de kolonnene som har likt navn
- ◆ F.eks. `categories.category_id` med `products.category_id`
- ◆ Dette kan gjøres enklere med *naturlig join*
- ◆ Naturlig join joiner (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)

Naturlig Join: Eksempel

Finn navnet på alle drikkevarer [12 rader]

Naturlig Join: Eksempel

Finn navnet på alle drikkevarer [12 rader]

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

Naturlig Join: Uønsket resultat

MERK: Naturlig join kan gi uønsket resultat da ALLE kolonner med likt navn joines

Finn productnavn og antall bestilte varer for alle bestillinger

Naturlig Join: Uønsket resultat

MERK: Naturlig join kan gi uønsket resultat da ALLE kolonner med likt navn joines

Finn productnavn og antall bestilte varer for alle bestillinger

```
SELECT product_name, quantity  
FROM order_details NATURAL JOIN products
```

[1493 rader]

```
SELECT p.product_name, o.quantity  
FROM order_details AS o INNER JOIN products AS p  
ON (o.product_id = p.product_id)
```

[2155 rader]

Naturlig Join: Uønsket resultat

MERK: Naturlig join kan gi uønsket resultat da ALLE kolonner med likt navn joines

Finn productnavn og antall bestilte varer for alle bestillinger

```
SELECT product_name, quantity  
FROM order_details NATURAL JOIN products
```

[1493 rader]

```
SELECT p.product_name, o.quantity  
FROM order_details AS o INNER JOIN products AS p  
ON (o.product_id = p.product_id)
```

[2155 rader]

Hvorfor er de forskjellige? `unit_price` er en kolonne i begge, men betyr forskjellige ting: `products.unit_price` er nåværende pris, mens `order_details.unit_price` er prisen den ble solgt for

Aggregering

- ◆ En aggregeringsfunksjon er en funksjon som returnerer en enkel verdi fra en samling verdier

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.

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)

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

Aggregering: Sum

- ◆ For å summere en hel kolonne, kan vi putte `sum(<column>)` i `SELECT`-klausulen
- ◆ For eksempel, for å finne antall varer som er solgt:

```
SELECT sum(quantity) AS totalquantity  
FROM order_details
```

Aggregering: Sum

- ◆ For å summere en hel kolonne, kan vi putte `sum(<column>)` i `SELECT`-klausulen
- ◆ For eksempel, for å finne antall varer som er solgt:

```
SELECT sum(quantity) AS totalquantity  
FROM order_details
```

- ◆ Tilsvarende har vi:

Aggregering: Sum

- ◆ For å summere en hel kolonne, kan vi putte `sum(<column>)` i `SELECT`-klausulen
- ◆ For eksempel, for å finne antall varer som er solgt:

```
SELECT sum(quantity) AS totalquantity
FROM order_details
```

- ◆ Tilsvarende har vi:
 - ◆ `avg` – gjennomsnitt

Aggregering: Sum

- ◆ For å summere en hel kolonne, kan vi putte `sum(<column>)` i `SELECT`-klausulen
- ◆ For eksempel, for å finne antall varer som er solgt:

```
SELECT sum(quantity) AS totalquantity
FROM order_details
```

- ◆ Tilsvarende har vi:
 - ◆ `avg` – gjennomsnitt
 - ◆ `max` – maksimum

Aggregering: Sum

- ◆ For å summere en hel kolonne, kan vi putte `sum(<column>)` i `SELECT`-klausulen
- ◆ For eksempel, for å finne antall varer som er solgt:

```
SELECT sum(quantity) AS totalquantity
FROM order_details
```

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

Aggregering: Sum

- ◆ For å summere en hel kolonne, kan vi putte `sum(<column>)` i `SELECT`-klausulen
- ◆ For eksempel, for å finne antall varer som er solgt:

```
SELECT sum(quantity) AS totalquantity
FROM order_details
```

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

Kombinere aggregering og andre kolonner

- ◆ En aggregeringsfunksjon returnerer én enkel verdi

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

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 unit_price,                                -- ERROR!  
       sum(quantity) AS totalquantity  
FROM   order_details
```

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 unit_price,                                -- ERROR!  
       sum(quantity) AS totalquantity  
FROM order_details
```

- ◆ Merk at man derimot kan kombinere flere aggregater i samme `WHERE`-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 det totale antallet tilbehørsprodukter (eng.: "accessories"):

```
SELECT count(*) AS numberofaccessories
  FROM products AS p INNER JOIN categories AS c
    ON (p.category_id = c.category_id)
 WHERE c.category_name = 'Accessories'
```

Aggregering: Count

- ◆ For eksempel, for å finne det totale antallet tilbehørsprodukter (eng.: "accessories"):

```
SELECT count(*) AS numberofaccessories
  FROM products AS p INNER JOIN categories AS c
    ON (p.category_id = c.category_id)
 WHERE c.category_name = 'Accessories'
```

- ◆ Merk at `count(*)` gir samme svar som `count(product_id)`, altså, vi teller antall rader

Aggregering: Count

- ◆ For eksempel, for å finne det totale antallet tilbehørsprodukter (eng.: "accessories"):

```
SELECT count(*) AS numberofaccessories
  FROM products AS p INNER JOIN categories AS c
    ON (p.category_id = c.category_id)
 WHERE c.category_name = 'Accessories'
```

- ◆ Merk at `count(*)` gir samme svar som `count(product_id)`, altså, vi teller antall rader
- ◆ Merk at det kan være duplikater i svaret

Aggregering: Count

- ◆ For eksempel, for å finne det totale antallet tilbehørsprodukter (eng.: "accessories"):

```
SELECT count(*) AS numberofaccessories
  FROM products AS p INNER JOIN categories AS c
    ON (p.category_id = c.category_id)
 WHERE c.category_name = 'Accessories'
```

- ◆ Merk at `count(*)` gir samme svar som `count(product_id)`, altså, vi teller antall rader
- ◆ Merk at det kan være duplikater i svaret
- ◆ Skal straks se hvordan man kan telle kun unike svar

- ◆ Husk at tingene i en **FROM**-klausul er tabeller

Delspørringer

- ◆ Husk at tingene i en **FROM**-klausul er tabeller
- ◆ Husk også at resultatet av en **SELECT**-spørring er en tabell

Delspørringer

- ◆ 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!

Delspørringer

- ◆ 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-delspørringer

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

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

Ekempel-delspøringer

- ◆ F.eks., 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 beveragesorder_details AS d  
ON (beverages.product_id = d.product_id)
```

Ekempel-delspøringer

- ◆ F.eks., 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 delspøringer som tabeller må gis et navn

Delspørringer som verdier

- ◆ En aggregatfunksjon over en kolonne returnerer én enkelt verdi

Delspørringer som verdier

- ◆ En aggregatfunksjon over en kolonne returnerer én enkelt verdi
- ◆ Vi kan derfor bruke den som en verdi i **WHERE**-klausulen

Delspøringer 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)
```

Delspørringer 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

Delspørringer 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 nøkkelorder **IN**

Delspørringer 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 nøkkelorder **IN**
- ◆ Kan ofte brukes i stedet for joins

Delspøringer 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 nøkkelorder **IN**
- ◆ Kan ofte brukes i stedet for joins
- ◆ F.eks. 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

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

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?

Hva er den største differansen mellom prisen på laptopper?

```
SELECT max(l1.Price - l2.Price) AS diff
FROM (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS l1,
      (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS l2
```

Hva er den største differansen mellom prisen på laptopper?

```
SELECT max(l1.Price - l2.Price) AS diff
FROM (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS l1,
     (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS l2
```

- ◆ 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(l1.Price - l2.Price) AS diff
FROM laptops AS l1, laptops AS l2
```

Hva er den største differansen mellom prisen på laptopper?

```
SELECT max(l1.Price - l2.Price) AS diff
FROM (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS l1,
     (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS l2
```

- ◆ 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(l1.Price - l2.Price) AS diff
FROM laptops AS l1, laptops AS l2
```

- ◆ Dette er både enklere å lese, lettere å vedlikeholde, og mer effektivt (slipper å kjøre laptops-spørringen to ganger)

Hva er den største differansen mellom prisen på laptopper?

```
SELECT max(l1.Price - l2.Price) AS diff
FROM (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS l1,
     (SELECT Price FROM products WHERE Name LIKE '%Laptop%') AS l2
```

- ◆ 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(l1.Price - l2.Price) AS diff
FROM laptops AS l1, laptops AS l2
```

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

Komplisert eksempel

Finn kundenavn og productnavn på alle kunder som har bestilt en drikkevare som ikke lenger selges ("discontinued") [230 rader]

Komplisert eksempel

Finn kundenavn og productnavn på alle kunder som har bestilt en drikkevare som ikke lenger selges ("discontinued") [230 rader]

```
SELECT DISTINCT c.company_name, p.product_name
FROM products AS p
    INNER JOIN order_details AS d ON (p.product_id = d.product_id)
    INNER JOIN orders AS o ON (o.order_id = d.order_id)
    INNER JOIN customers AS c ON (c.customer_id = o.customer_id)
    INNER JOIN categories AS g ON (g.category_id = p.category_id)
WHERE p.discontinued = 1 AND
    g.category_name = 'Beverages';
```


Komplisert eksempel

Finn navnet på alle drikkevarer som aldri har blitt solgt for lavere enn gjennomsnittsprisen for alle salg [2 rader]

Komplisert eksempel

Finn navnet på alle drikkevarer som aldri har blitt solgt for lavere enn gjennomsnittsprisen for alle salg [2 rader]

```
SELECT p.product_name
FROM products AS p INNER JOIN categories AS c
    ON (p.category_id = c.category_id)
WHERE c.category_name = 'Beverages' AND
    (SELECT avg(unit_price)
     FROM order_details)
    <
    (SELECT min(d.unit_price)
     FROM order_details AS d
     WHERE p.product_id = d.product_id);
```