

# IN2090 – Databaser og datamodellering

## 09 – Aggregering og sortering

Leif Harald Karlsen  
leifhka@ifi.uio.no



Universitetet i Oslo

## Enklere syntaks for joins

---

- ◆ Jeg lærte noe nytt forige uke!

# Enklere syntaks for joins

---

- ◆ Jeg lærte noe nytt forige uke!
- ◆ Man kan bruke `USING (<kolonne>)` fremfor `ON (a.<kolonne> = b.<kolonne>)`

# Enklere syntaks for joins

---

- ◆ Jeg lærte noe nytt forige uke!
- ◆ 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);
```

# Enklere syntaks for joins

---

- ◆ Jeg lærte noe nytt forige uke!
- ◆ 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 delspørringer (1)

---

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

## Eksempel: Variable i delspørringer (1)

---

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);
```

## Eksempel: Variable i delspøringer (1)

---

Merk: Man kan bruke variabler fra en spørring i dens delspøringer

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);
```



## Eksempel: Variable i delspørringer (2)

---

Finn antall produkter for hver kategori

## Eksempel: Variable i delspørringer (2)

---

### Finn antall produkter for hver kategori

```
SELECT c.categoryname,  
       (SELECT count(*)  
        FROM products AS p  
        WHERE p.category_id = c.category_id) AS nr_products  
FROM categories AS c
```

## Eksempel: Variable i delspørringer (2)

---

### Finn antall produkter for hver kategori

```
SELECT c.categoryname,  
       (SELECT count(*)  
        FROM products AS p  
        WHERE p.category_id = c.category_id) AS nr_products  
FROM categories AS c
```

- ◆ Denne spørringen utfører en aggregering per kategori.

## Eksempel: Variable i delspørringer (2)

---

### Finn antall produkter for hver kategori

```
SELECT c.categoryname,  
       (SELECT count(*)  
        FROM products AS p  
        WHERE p.category_id = c.category_id) AS nr_products  
FROM categories AS c
```

- ◆ Denne spørringen utfører en aggregering per kategori.
- ◆ Den lager altså ett aggregat per *gruppe* av rader

## Eksempel: Variable i delspørringer (2)

---

### Finn antall produkter for hver kategori

```
SELECT c.categoryname ,
       (SELECT count(*)
        FROM products AS p
        WHERE p.category_id = c.category_id) AS nr_products
FROM categories AS c
```

- ◆ Denne spørringen utfører en aggregering per kategori.
- ◆ Den lager altså ett aggregat per *gruppe* av rader
- ◆ Senere i dag skal vi se en egen syntaks for å gjøre dette litt enklere

# Sortering

---

- ◆ For å sortere radene i resultatet fra en `SELECT`-spørring, kan vi bare legge `ORDER BY` <kolonner> på slutten av spørringen

# 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

# 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 drikkevarer etter pris:

```
SELECT p.product_name, p.unit_price
      FROM products AS p INNER JOIN categories AS c
      ON (p.categoryid = c.categoryid)
      WHERE c.categorydescription = 'Beverages'
      ORDER BY p.unit_price
```



# 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 drikkevarer etter pris:

```
SELECT p.product_name, p.unit_price
      FROM products AS p INNER JOIN categories AS c
      ON (p.categoryid = c.categoryid)
      WHERE c.categorydescription = 'Beverages'
      ORDER BY p.unit_price
```

- ◆ Sorteringen er gjort i henhold til typens naturlige ordning

# 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 drikkevarer etter pris:

```
SELECT p.product_name, p.unit_price
      FROM products AS p INNER JOIN categories AS c
      ON (p.categoryid = c.categoryid)
      WHERE c.categorydescription = 'Beverages'
      ORDER BY p.unit_price
```

- ◆ Sorteringen er gjort i henhold til typens naturlige ordning
  - ◆ Tall: verdi

# 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 drikkevarer etter pris:

```
SELECT p.product_name, p.unit_price
      FROM products AS p INNER JOIN categories AS c
      ON (p.categoryid = c.categoryid)
      WHERE c.categorydescription = 'Beverages'
      ORDER BY p.unit_price
```

- ◆ Sorteringen er gjort i henhold til typens naturlige ordning
  - ◆ Tall: verdi
  - ◆ Tekst: alfabetisk

# 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 drikkevarer etter pris:

```
SELECT p.product_name, p.unit_price
      FROM products AS p INNER JOIN categories AS c
      ON (p.categoryid = c.categoryid)
      WHERE c.categorydescription = 'Beverages'
      ORDER BY p.unit_price
```

- ◆ Sorteringen er gjort i henhold til typens naturlige ordning
  - ◆ Tall: verdi
  - ◆ Tekst: alfabetisk
  - ◆ Tidspunkter: kronologisk

# 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 drikkevarer etter pris:

```
SELECT p.product_name, p.unit_price
      FROM products AS p INNER JOIN categories AS c
      ON (p.categoryid = c.categoryid)
      WHERE c.categorydescription = 'Beverages'
      ORDER BY p.unit_price
```

- ◆ Sorteringen er gjort i henhold til typens naturlige ordning
  - ◆ Tall: verdi
  - ◆ Tekst: alfabetisk
  - ◆ Tidspunkter: kronologisk
  - ◆ osv.

# 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 drikkevarer etter pris:

```
SELECT p.product_name, p.unit_price
      FROM products AS p INNER JOIN categories AS c
      ON (p.categoryid = c.categoryid)
      WHERE c.categorydescription = 'Beverages'
      ORDER BY p.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

## 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



## 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 raden, så ihht. den andre kolonnen for de med like verdier på den første, osv.

## 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 raden, så ihht. den andre kolonnen for de med like verdier på den første, osv.
- ◆ For eksempel, for å sortere drikkevarer først på pris, og så på antall på lager, begge i nedadgående rekkefølge:

```
SELECT p.product_name, p.units_in_stock, p.unit_price
FROM products AS p INNER JOIN categories AS c
ON (p.category_id = c.category_id)
WHERE c.category_name = 'Beverages'
ORDER BY p.unit_price DESC,
         p.units_in_stock DESC;
```

## Begrense antall rader i resultatet

---

- ◆ Når vi gjør spørringer mot store tabeller får vi ofte mange svar

## Begrense antall rader i resultatet

---

- ◆ Når vi gjør spørringer mot store tabeller får vi ofte mange svar
- ◆ Av og til er vi ikke interessert i alle svarene

## Begrense antall rader i resultatet

---

- ◆ Når vi gjør spørringer 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`

## Begrense antall rader i resultatet

---

- ◆ Når vi gjør spørringer 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;
```

## Begrense antall rader i resultatet

---

- ◆ Når vi gjør spørringer 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 1: Finn navn og pris på produktet med lavest pris (1)

---

Ved `min`-aggregering og tabell-delspørring



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

---

Ved `min`-aggregering og tabell-delspørring

```
SELECT p.product_name, p.unit_price
FROM products AS p,
      (SELECT min(unit_price) AS minprice
       FROM products) AS h
WHERE p.unit_price = h.minprice;
```

## Eksempel 1: Finn navn og pris på produktet med lavest pris (2)

---

Ved `min`-aggregering og verdi-delspørring

## Eksempel 1: Finn navn og pris på produktet med lavest pris (2)

---

Ved `min`-aggregering og verdi-delspørring

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

## Eksempel 1: Finn navn og pris på produktet med lavest pris (3)

---

Ved `ORDER BY` og `LIMIT 1`

## Eksempel 1: Finn navn og pris på produktet med lavest pris (3)

---

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

# Hoppe over rader

---

- ◆ Av og til ønsker man å hoppe over rader
- ◆ Dette er nyttig dersom man ønsker å presentere resultater i grupper

# Hoppe over rader

---

- ◆ Av og til ønsker man å hoppe over rader
- ◆ Dette er nyttig dersom man ønsker å presentere resultater i grupper
- ◆ F.eks. slik som søkeresultater fra en søkemotor, man får presentert én og én side med resultater



# Hoppe over rader

---

- ◆ Av og til ønsker man å hoppe over rader
- ◆ Dette er nyttig dersom man ønsker å presentere resultater i grupper
- ◆ F.eks. slik som søkeresultater fra en søkemotor, man får presentert én og én side med resultater
- ◆ Dette gjøres med `OFFSET`-klausulen

## Hoppe over rader

---

- ◆ Av og til ønsker man å hoppe over rader
- ◆ Dette er nyttig dersom man ønsker å presentere resultater i grupper
- ◆ F.eks. 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.
```

# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner

# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner
- ◆ Og tidligere i dag, hvordan man kan regne ut aggregater over grupper

# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner
- ◆ Og tidligere i dag, hvordan man kan regne ut aggregater over grupper
- ◆ Det finner derimot en egen klausul for å gruppere før man aggregerer

# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner
- ◆ Og tidligere i dag, hvordan man kan regne ut aggregater over grupper
- ◆ Det finner derimot en egen klausul for å gruppere før man aggregerer
- ◆ Nemlig `GROUP BY <kolonner>`

# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner
- ◆ Og tidligere i dag, hvordan man kan regne ut aggregater over grupper
- ◆ Det finner derimot en egen klausul for å gruppere 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

# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner
- ◆ Og tidligere i dag, hvordan man kan regne ut aggregater over grupper
- ◆ Det finner derimot en egen klausul for å gruppere 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



# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner
- ◆ Og tidligere i dag, hvordan man kan regne ut aggregater over grupper
- ◆ Det finner derimot en egen klausul for å gruppere 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

# Aggregere i grupper

---

- ◆ Vi har sett hvordan vi kan aggregere over hele kolonner
- ◆ Og tidligere i dag, hvordan man kan regne ut aggregater over grupper
- ◆ Det finner derimot en egen klausul for å gruppere 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 i `SELECT`

# Aggregere i grupper: Eksempel

---

Finn gjennomsnittsprisen for hver kategori

```
SELECT Category, avg(Price) AS Averageprice
FROM Products
GROUP BY Category
```

Resultat

# 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: Velg ut kolonner og grupper ihht. Categories

Price	Category
8999	Televisions
11999	Televisions
7499	Computers
6999	Computers
5195	Computers
4999	Speakers

# 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

# 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

Category	Averageprice
Televisions	10499
Computers	6731
Speakers	4999



# Aggregering i grupper: Eksempel 1

---

Finn antall produkter per bestilling

# Aggregering i grupper: Eksempel 1

---

## Finn antall produkter per bestilling

```
SELECT o.order_id, sum(d.quantity) AS nr_products
  FROM orders AS o INNER JOIN order_details AS d
    USING (order_id)
GROUP BY o.order_id;
```

## Aggregering i grupper: Eksempel 2

---

Finn navn på ansatte og antall bestillinger den ansatte har håndtert, sortert etter antall bestillinger fra høyest til lavest

## Aggregering i grupper: Eksempel 2

---

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(o.order_id) AS num_orders
FROM   orders AS o INNER JOIN employees AS e
       USING (employee_id)
GROUP BY e.employee_id
ORDER BY num_orders DESC
```

# Hvorfor fungerte forrige spørring?

---

```
SELECT format('%s %s', e.first_name, e.last_name) AS emp_name,
       count(o.order_id) AS num_orders
FROM   orders AS o INNER JOIN employees AS e
       USING (employee_id)
GROUP BY e.employee_id
ORDER BY num_orders DESC
```

## Hvorfor fungerte forrige spørring?

---

```
SELECT format('%s %s', e.first_name, e.last_name) AS emp_name,
       count(o.order_id) AS num_orders
FROM   orders AS o INNER JOIN employees AS e
       USING (employee_id)
GROUP BY e.employee_id
ORDER BY num_orders DESC
```

- ◆ Har e.first\_name og e.last\_name i **SELECT**, men grupperer på e.employee\_id

# Hvorfor fungerte forrige spørring?

---

```
SELECT format('%s %s', e.first_name, e.last_name) AS emp_name,  
       count(o.order_id) AS num_orders  
FROM orders AS o INNER JOIN employees AS e  
   USING (employee_id)  
GROUP BY e.employee_id  
ORDER BY num_orders DESC
```

- ◆ Har e.first\_name og e.last\_name i **SELECT**, men grupperer på e.employee\_id
- ◆ Har tidligere sagt at det kun gir mening å ha kolonnene man grupperer på utenfor aggregater i **SELECT**

# Hvorfor fungerte forrige spørring?

---

```
SELECT format('%s %s', e.first_name, e.last_name) AS emp_name,
       count(o.order_id) AS num_orders
FROM   orders AS o INNER JOIN employees AS e
       USING (employee_id)
GROUP BY e.employee_id
ORDER BY num_orders DESC
```

- ◆ Har `e.first_name` og `e.last_name` i `SELECT`, men grupperer på `e.employee_id`
- ◆ Har tidligere sagt at det kun gir mening å ha kolonnene man grupperer på utenfor aggregater i `SELECT`
- ◆ Siden `employee_id` er primærnøkkel for `employee` vil man for hver `employee_id` ha kun ett `first_name` og ett `last_name`



## Hvorfor fungerte forrige spørring?

---

```
SELECT format('%s %s', e.first_name, e.last_name) AS emp_name,
       count(o.order_id) AS num_orders
FROM   orders AS o INNER JOIN employees AS e
       USING (employee_id)
GROUP BY e.employee_id
ORDER BY num_orders DESC
```

- ◆ Har `e.first_name` og `e.last_name` i `SELECT`, men grupperer på `e.employee_id`
- ◆ Har tidligere sagt at det kun gir mening å ha kolonnene man grupperer på utenfor aggregater i `SELECT`
- ◆ Siden `employee_id` er primærnøkkel for `employee` vil man for hver `employee_id` ha kun ett `first_name` og ett `last_name`
- ◆ Derfor kan Postgres utlede hvilket `first_name` og `last_name` som hører til hver gruppe

## Hvorfor fungerte forrige spørring?

---

```
SELECT format('%s %s', e.first_name, e.last_name) AS emp_name,
       count(o.order_id) AS num_orders
FROM   orders AS o INNER JOIN employees AS e
       USING (employee_id)
GROUP BY e.employee_id
ORDER BY num_orders DESC
```

- ◆ Har `e.first_name` og `e.last_name` i `SELECT`, men grupperer på `e.employee_id`
- ◆ Har tidligere sagt at det kun gir mening å ha kolonnene man grupperer på utenfor aggregater i `SELECT`
- ◆ Siden `employee_id` er primærnøkkel for `employee` vil man for hver `employee_id` ha kun ett `first_name` og ett `last_name`
- ◆ Derfor kan Postgres utlede hvilket `first_name` og `last_name` som hører til hver gruppe
- ◆ Dette vil f.eks. ikke fungere om man heller grupperer på (kun) `e.last_name`, da dette ikke er en primærnøkkel

## Hvorfor fungerte forrige spørring?

---

```
SELECT format('%s %s', e.first_name, e.last_name) AS emp_name,
       count(o.order_id) AS num_orders
FROM   orders AS o INNER JOIN employees AS e
       USING (employee_id)
GROUP BY e.employee_id
ORDER BY num_orders DESC
```

- ◆ Har `e.first_name` og `e.last_name` i `SELECT`, men grupperer på `e.employee_id`
- ◆ Har tidligere sagt at det kun gir mening å ha kolonnene man grupperer på utenfor aggregater i `SELECT`
- ◆ Siden `employee_id` er primærnøkkel for `employee` vil man for hver `employee_id` ha kun ett `first_name` og ett `last_name`
- ◆ Derfor kan Postgres utlede hvilket `first_name` og `last_name` som hører til hver gruppe
- ◆ Dette vil f.eks. ikke fungere om man heller grupperer på (kun) `e.last_name`, da dette ikke er en primærnøkkel
- ◆ Ikke alle RDBMSer støtter dette

## Aggregering i grupper: Eksempel 3

---

Finn beskrivelsen av den kategorien med høyest gjennomsnittspris

## Aggregering i grupper: Eksempel 3

---

Finn beskrivelsen av den kategorien med høyest gjennomsnittspris

```
SELECT c.description, a.avgprice
FROM (SELECT category_id, avg(unit_price) AS avgprice
      FROM products
      GROUP BY category_id
     ) AS a
INNER JOIN categories AS c
ON (c.category_id = a.category_id)
ORDER BY a.avgprice DESC
LIMIT 1;
```

## Aggregering i grupper: Eksempel 4

---

Finn navn og total regning for hver kunde

## Aggregering i grupper: Eksempel 4

---

### Finn navn og total regning for hver kunde

```
SELECT sum(d.unit_price * d.quantity * (1 - d.discount)) AS customertotal,  
       c.company_name  
FROM customers AS c  
     INNER JOIN orders AS o  
         ON (c.customer_id = o.customer_id)  
     INNER JOIN order_details AS d  
         ON (o.order_id = d.order_id)  
GROUP BY c.customer_id
```

## Gruppere på flere kolonner

---

- ◆ Vi kan også gruppere på flere kolonner



## 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

## 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

# 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
    USING (category_id)
GROUP BY c.category_id, p.discontinued;
```

## Filtrere på aggregat-resultat

---

- ◆ I mange tilfeller ønsker vi å velge ut rader basert på verdien av aggregatet

## Filtrere på aggregat-resultat

---

- ◆ I mange tilfeller ønsker vi å velge ut rader basert på verdien av aggregatet
- ◆ F.eks. dersom man vil vite kategorinavn og antall produkter på de kategoriene som har flere enn 10 produkter

## Filtrere på aggregat-resultat

---

- ◆ I mange tilfeller ønsker vi å velge ut rader basert på verdien av aggregatet
- ◆ 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 USING (category_id)
  GROUP BY c.category_id) AS t
WHERE nr_products > 10;
```

## Filtrere på aggregat-resultat

---

- ◆ I mange tilfeller ønsker vi å velge ut rader basert på verdien av aggregatet
- ◆ 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 USING (category_id)
  GROUP BY c.category_id) AS t
WHERE nr_products > 10;
```

- ◆ Men det finnes en egen klausul for å begrense på uttrykkene i `SELECT`, slik som aggregater

# Filtrere på aggregat-resultat

---

- ◆ 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 USING (category_id)
GROUP BY c.category_id
HAVING count(*) > 10;
```



# Filtrere på aggregat-resultat

---

- ◆ 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 USING (category_id)
GROUP BY c.category_id
HAVING count(*) > 10;
```

- ◆ Merk: Kan ikke bruke navnene vi gir i **SELECT**

# Filtrere på aggregat-resultat

---

- ◆ 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 USING (category_id)
GROUP BY c.category_id
HAVING count(*) > 10;
```

- ◆ Merk: Kan ikke bruke navnene vi gir i **SELECT**
- ◆ **HAVING** blir altså evaluert på hver gruppe

# Filtrere på aggregat-resultat

---

- ◆ 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 USING (category_id)
GROUP BY c.category_id
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

# Oversikt over SQLs SELECT

---

- ◆ Vi har nå sett mange nye klausuler
- ◆ Generelt ser våre SQL-spørringer nå slik ut:

```
    WITH <navngitte-spørringer>  
    SELECT <kolonner>  
    FROM <tabeller>  
    WHERE <uttrykk>  
    GROUP BY <kolonner>  
    HAVING <uttrykk>  
    ORDER BY <kolonner> [DESC]  
    LIMIT <N>  
    OFFSET <M>
```

# Oversikt over SQLs SELECT

---

- ◆ Vi har nå sett mange nye klausuler
- ◆ Generelt ser våre SQL-spørringer nå slik ut:

```
    WITH <navngitte-spørringer>
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)

# Oversikt over SQLs SELECT

---

- ◆ Vi har nå sett mange nye klausuler
- ◆ Generelt ser våre SQL-spørringer nå slik ut:

```
    WITH <navngitte-spørringer>
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

---

Generelt kan et navn kun brukes i den spørringen den introduseres i og den spørringens delspørringer, ihht. følgende begrensninger:



# Hvor kan ulike navn brukes

---

Generelt kan et navn kun brukes i den spørringen den introduseres i og den spørringens delspørringer, ihht. følgende begrensninger:

- ◆ Navnene vi lager med **AS** i **WITH**-klausulen kan kun brukes i alle de etterfølgende spørringene

# Hvor kan ulike navn brukes

---

Generelt kan et navn kun brukes i den spørringen den introduseres i og den spørringens delspørringer, ihht. følgende begrensninger:

- ◆ Navnene vi lager med `AS` i `WITH`-klausulen kan kun brukes i alle de etterfølgende spørringene
- ◆ Navnene fra `SELECT` kan kun brukes i `SELECT`- og `ORDER BY`-klausulene

# Hvor kan ulike navn brukes

---

Generelt kan et navn kun brukes i den spørringen den introduseres i og den spørringens delspørringer, ihht. følgende begrensninger:

- ◆ Navnene vi lager med **AS** i **WITH**-klausulen kan kun brukes i alle de etterfølgende spørringene
- ◆ Navnene fra **SELECT** kan kun brukes i **SELECT**- og **ORDER BY**-klausulene
- ◆ Navnene fra **FROM** kan kun brukes i alle klausuler utenom samme **FROM**-klausul

## Eksempel 1: Implisitt informasjon om kategorier

---

Finn høyeste, laveste, og gjennomsnittspris på produktene i hver kategori

## Eksempel 1: Implisitt informasjon om kategorier

---

Finn høyeste, laveste, og gjennomsnittspris på produktene i hver kategori

```
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
FROM categories AS c
     INNER JOIN products AS p USING (category_id)
GROUP BY c.category_id;
```

## Eksempel 2: Implisitt informasjon om land

---

Finn de tre mest kjøpte produktene for hvert land

## Eksempel 2: Implisitt informasjon om land

### Finne de tre mest kjøpte produktene for hvert land

```
WITH
  sold_by_country AS (
    SELECT c.country, p.product_id, p.product_name, count(*) AS nr_sold
    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, product_id
  ),
  sold_ordered AS (
    SELECT * FROM sold_by_country ORDER BY nr_sold DESC
  )
SELECT c.country,
  (SELECT s.product_name FROM sold_ordered AS s
   WHERE s.country = c.country
   LIMIT 1) AS first_place,
  (SELECT s.product_name FROM sold_ordered AS s
   WHERE s.country = c.country
   OFFSET 1
   LIMIT 1) AS second_place,
  (SELECT s.product_name FROM sold_ordered AS s
   WHERE s.country = c.country
   OFFSET 2
   LIMIT 1) AS third_place
FROM (SELECT DISTINCT country FROM customers) 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 aggreger 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;
```

## Mer interessante views

---

- ◆ Aggregering i grupper, sortering og å begrense svaret er svært nyttig når man har store mengder data

## Mer interessante views

---

- ◆ Aggregering i grupper, sortering og å begrense svaret er svært nyttig når man har store mengder data
- ◆ Når vi grupperer lager vi oss på sett og vis nye objekter, eller gjør implisitte objekter eksplisitte

## Mer interessante views

---

- ◆ Aggregering i grupper, sortering og å begrense svaret er svært nyttig når man har store mengder data
- ◆ Når vi grupperer lager vi oss på sett og vis nye objekter, eller gjør implisitte objekter eksplisitte
- ◆ Aggregeringen over disse gruppene lar oss utlede informasjon om disse nye objektene

## Mer interessante views

---

- ◆ Aggregering i grupper, sortering og å begrense svaret er svært nyttig når man har store mengder data
- ◆ Når vi grupperer lager vi oss på sett og vis nye objekter, eller gjør implisitte objekter eksplisitte
- ◆ Aggregeringen over disse gruppene lar oss utlede informasjon om disse nye objektene
- ◆ Sortering og begrensnig lar oss hente ut de mest interessante objektene

## Mer interessante views

---

- ◆ Aggregering i grupper, sortering og å begrense svaret er svært nyttig når man har store mengder data
- ◆ Når vi grupperer lager vi oss på sett og vis nye objekter, eller gjør implisitte objekter eksplisitte
- ◆ Aggregeringen over disse gruppene lar oss utlede informasjon om disse nye objektene
- ◆ Sortering og begrensnig lar oss hente ut de mest interessante objektene
- ◆ Dette gjør at vi kan lage mer interessante views

## Mer interessante views

---

- ◆ Aggregering i grupper, sortering og å begrense svaret er svært nyttig når man har store mengder data
- ◆ Når vi grupperer lager vi oss på sett og vis nye objekter, eller gjør implisitte objekter eksplisitte
- ◆ Aggregeringen over disse gruppene lar oss utlede informasjon om disse nye objektene
- ◆ Sortering og begrensnig lar oss hente ut de mest interessante objektene
- ◆ Dette gjør at vi kan lage mer interessante views
- ◆ For eksempel vil de eksemplene vi nettopp så være interessante views