

IN2090 – Databaser og datamodellering

10 – Outer joins og mengdeoperatorer

Leif Harald Karlsen
leifhka@ifi.uio.no



Universitetet i Oslo

Aggregering og NULL

- ◆ Aggregering med `sum`, `min`, `max` og `avg` ignorerer `NULL`-verdier
- ◆ Det betyr også at dersom det kun er `NULL`-verdier i en kolonne blir resultatet av disse `NULL`
- ◆ `count(*)` teller med `NULL`-verdier
- ◆ Men dersom vi oppgir en konkret kolonne, f.eks. `count(product_name)` vil den kun telle verdiene som ikke er `NULL`
- ◆ For eksempel:

Name	Age
Per	2
Kari	4
Mari	NULL

```
SELECT min(Age) FROM Person;    --> 2
SELECT avg(Age) FROM Person;    --> 3
SELECT count(Age) FROM Person;  --> 2
SELECT count(*) FROM Person;    --> 3
```

```
SELECT sum(Age) FROM Person
WHERE Name = 'Mari';           --> NULL
```

```
SELECT count(Age) FROM Person
WHERE Name = 'Mari';          --> 0
```

Repetisjon: Inner joins

Hvilken kunde har kjøpt hvilket produkt?

```
SELECT ProductName, Customer
FROM products AS p INNER JOIN orders AS o
ON p.ProductID = o.ProductID
```

Resultat

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

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

Inner joins og manglende verdier

Hvilken kunde har kjøpt hvilket produkt?

```
SELECT ProductName, Customer
FROM products AS p INNER JOIN orders AS o
ON p.ProductID = o.ProductID
```

Resultat

products		
ProductID	Name	Price
0	TV 50 inch	8999
1	Laptop 2.5GHz	7499
2	Noise-amplifying Headphones	9999

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

Inner joins og manglende verdier med aggregater

Hvor mange har kjøpt hvert produkt?

```
SELECT ProductName, count(o.Customer) AS num
FROM products AS p INNER JOIN orders AS o
    ON p.ProductID = o.ProductID
GROUP BY p.ProductID
```

Resultat

products		
ProductID	Name	Price
0	TV 50 inch	8999
1	Laptop 2.5GHz	7499
2	Noise-amplifying Headphones	9999

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

Problemer med Indre joins

- ◆ I forige spørring fikk vi ikke opp at 0 kunder har kjøpt Noise-amplifying Headset
- ◆ Årsaken er at den ikke joiner med noe, og derfor forsvinner fra svaret
- ◆ For å få ønsket resultat trenger vi altså en ny type join
- ◆ De nye joinene som løser problemet vårt heter ytre joins, eller *outer join* på engelsk

Outer Joins

- ◆ Vi har flere varianter av ytre joins, nemlig
 - ◆ `left outer join`
 - ◆ `right outer join`
 - ◆ `full outer join`
- ◆ Brukes ved å bytte ut `INNER JOIN` med f.eks. `LEFT OUTER JOIN`
- ◆ Hovedidéen bak denne typen join er å bevare alle rader fra en eller begge tabellene i joinen
- ◆ Og så fylle inn med `NULL` hvor vi ikke har noen match

Left Outer Join

- ◆ I en *left outer join* vil alle rader i den venstre tabellen bli med i svaret
- ◆ Resultatet av a `LEFT OUTER JOIN` b `ON` (a.c1 = b.c2) blir
 - ◆ samme som a `INNER JOIN` b `ON` (a.c1 = b.c2),
 - ◆ men hvor alle rader fra a som ikke matcher noen i b
 - ◆ (altså hvor a.c1 ikke er lik noen b.c2)
 - ◆ blir lagt til resultatet, med `NULL` for alle bs kolonner

Eksempel: Left Outer Join

Left outer join mellom products og orders

```
SELECT *  
FROM products AS p LEFT OUTER JOIN orders AS o  
ON p.ProductID = o.ProductID;
```

Resultat

products

ProductID	Name	Price
0	TV 50 inch	8999
1	Laptop 2.5GHz	7499
2	Noise-amplifying Headphones	9999

orders

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

Eksempel: Left Outer Join

Hvor mange har kjøpt hvert produkt?

```
SELECT p.ProductName, count(o.Customer) AS num
FROM products AS p LEFT OUTER JOIN orders AS o
ON p.ProductID = o.ProductID
GROUP BY p.ProductID
```

Resultat

ProductID	Name	Price
0	TV 50 inch	8999
1	Laptop 2.5GHz	7499
2	Noise-amplifying Headphones	9999

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

Andre nyttige bruksområder for ytre joins

- ◆ Som vi ser er ytre joins nyttige når vi aggregerer, for å ikke miste resultater underveis
- ◆ Ytre joins kan også være nyttige for å kombinere ufullstendig informasjon fra flere tabeller
- ◆ For eksempel:

ID	Name
1	Per
2	Mari
3	Ida

ID	Phone
1	48123456
3	98765432

ID	Email
1	per@mail.no
2	mari@umail.net

```
SELECT p.Name, n.Phone, e.Email
FROM Persons AS p
     LEFT OUTER JOIN Numbers AS n
       ON (p.ID = n.ID)
     LEFT OUTER JOIN Emails AS e
       ON (p.ID = e.ID);
```

p.Name	n.Phone	e.Email
Per	48123456	per@mail.no
Mari	NULL	mari@umail.net
Ida	98765432	NULL

Andre ytre joins

- ◆ a `RIGHT OUTER JOIN` b `ON` (a.c1 = b.c2) er akkurat det samme som b `LEFT OUTER JOIN` a `ON` (b.c2 = a.c1)
- ◆ Altså, i en *right outer join* vil alle radene i den høyre tabellen være med i resultatet
- ◆ Vi har også en `FULL OUTER JOIN` som er en slags kombinasjon, her vil ALLE rader være med i svaret
- ◆ For eksempel:

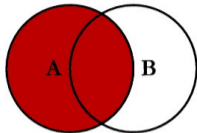
ID	Name
1	Per
2	Mari

ID	Phone
1	48123456
3	98765432

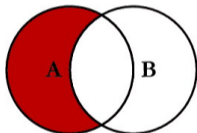
```
SELECT p.Name, n.Phone
FROM Persons AS p
FULL OUTER JOIN Numbers AS n
ON (p.ID = n.ID);
```

p.Name	n.Phone
Per	48123456
Mari	NULL
NULL	98765432

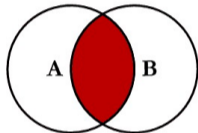
SQL JOINS



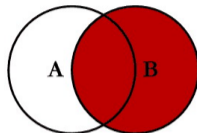
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



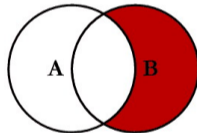
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL.
```



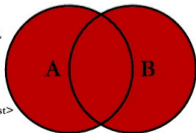
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



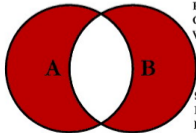
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL.
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL.
```

Ytre join-eksempel (1)

Finn antall bestillinger gjort av hver kunde

```
SELECT c.company_name, count(o.order_id) AS num_orders
FROM customers AS c
      LEFT OUTER JOIN orders AS o
          USING (customer_id)
GROUP BY c.customer_id;
```

Ytre join-eksempel (2)

Finn ut for hvor mye penger hver kunde har bestilt, for de som har færre en 5 bestillinger totalt

```
SELECT c.company_name ,
       sum(d.unit_price * d.quantity) AS sum_money
FROM   customers AS c
       LEFT OUTER JOIN orders AS o
           USING (customer_id)
       LEFT OUTER JOIN order_details AS d
           USING (order_id)
GROUP BY c.customer_id
HAVING count(DISTINCT o.order_id) < 5;
```

Merk: `count(DISTINCT o.order_id)` lar oss telle kun unike verdier! Nyttig snarvei.

Ytre join-eksempel (3)

Finn ut for hvor mye penger hver kunde har bestilt, for de som har færre en 100 produkter totalt

```
SELECT c.company_name ,
       sum(d.unit_price * d.quantity) AS sum_money
FROM   customers AS c
       LEFT OUTER JOIN orders AS o
           USING (customer_id)
       LEFT OUTER JOIN order_details AS d
           USING (order_id)
GROUP BY c.customer_id
HAVING sum(d.quantity) < 100 OR
       sum(d.quantity) IS NULL; -- MERK: NULL < 100 er NULL
```


Ytre join-eksempel (4)

Finn ut antall ganger hver ansatt har håndtert en ordre fra hver kunde

```
WITH
  all_combinations AS (
    SELECT e.employee_id,
           e.first_name || ' ' || e.last_name AS fullname,
           c.customer_id,
           c.company_name
    FROM employees AS e, customers AS c -- Kryssprodukt, alle kombinasjoner
  )
SELECT ac.fullname,
       ac.company_name,
       count(o.order_id) AS num_transactions
FROM all_combinations AS ac
     LEFT OUTER JOIN orders AS o
     ON (ac.employee_id = o.employee_id AND
         ac.customer_id = o.customer_id)
GROUP BY ac.customer_id, ac.company_name,
         ac.employee_id, ac.fullname;
```

Ytre join-eksempel (5)

Finn navnet på alle kunder som ikke har bestilt noe

```
SELECT c.company_name
       FROM customers as c
       LEFT OUTER JOIN orders as o
       USING (customer_id)
WHERE o.customer_id IS NULL;
```

Syntaks for joins

I stedet for

- ◆ `LEFT OUTER JOIN` kan man skrive `LEFT JOIN`
- ◆ `RIGHT OUTER JOIN` kan man skrive `RIGHT JOIN`
- ◆ `FULL OUTER JOIN` kan man skrive `FULL JOIN`
- ◆ `INNER JOIN` kan man skrive `JOIN`

Mengdeoperatorer

- ◆ Vi har nå et relativt uttrykingskraftig språk for å hente ut informasjon fra en database
- ◆ Men det er noen elementære ting vi fortsatt ikke kan gjøre
- ◆ F.eks. kombinere svar fra to spørringer til én tabell
- ◆ Eller trekke svarene fra en spørring fra en annen
- ◆ Husk at vi kan se på en svarene fra `SELECT` som en (multi-)mengde
- ◆ SQL tillater oss å bruke vanlige mengdeoperatorer (snitt, union, osv.)
- ◆ Ettersom SQLs tabeller er multimengder har vi to versjoner av hver operator:
 - ◆ én versjon som behandler resultatene som mengder (f.eks. `UNION`)
 - ◆ én versjon som behandler dem som multimengder (f.eks. `UNION ALL`)
- ◆ Disse mengdeoperatorene puttes *mellom* to spørringer

Mengdeoperatorene

- ◆ Vi har følgende mengdeoperatører:
 - ◆ Union – UNION
 - ◆ Snitt – INTERSECT
 - ◆ Differanse – EXCEPT
- ◆ For alle disse har vi i tillegg en variant med ALL etter seg som behandler resultatene som multimengder
- ◆ Antall ganger en rad er med i resultatet av $q1 \text{ UNION ALL } q2$ er det summen av antallet ganger raden er med i resultatene fra $q1$ og $q2$
- ◆ Antall ganger en rad er med i resultatet av $q1 \text{ INTERSECT ALL } q2$ er det minste antallet ganger raden er med i resultatene fra $q1$ og $q2$
- ◆ Antall ganger en rad er med i resultatet av $q1 \text{ EXCEPT ALL } q2$ er antallet ganger raden er med i resultatene $q1$ minus antallet ganger den er med i $q2$

Union-operatoren

persons

ID	Name	Phone	Email
1	Per	48123456	per@mail.no
2	Mari	NULL	mari@umail.net
3	Ola	NULL	NULL
4	Ida	98765432	NULL

```
(SELECT *  
  FROM persons  
  WHERE Phone IS NOT NULL)  
UNION  
(SELECT *  
  FROM persons  
  WHERE Email IS NOT NULL)
```

Resultat:

ID	Name	Phone	Email
1	Per	48123456	per@mail.no
4	Ida	98765432	NULL
2	Mari	NULL	mari@umail.net

```
(SELECT *  
  FROM persons  
  WHERE Phone IS NOT NULL)  
UNION ALL  
(SELECT *  
  FROM persons  
  WHERE Email IS NOT NULL)
```

Resultat:

ID	Name	Phone	Email
1	Per	48123456	per@mail.no
4	Ida	98765432	NULL
1	Per	48123456	per@mail.no
2	Mari	NULL	mari@umail.net

Union-kompatibilitet

- ◆ Hva skjer om vi tar unionen av to spørringer som returnerer forskjellig antall kolonner?

```
(SELECT Name, Phone
FROM person
WHERE Phone IS NOT NULL)
UNION
(SELECT Name, Phone, Email
FROM person
WHERE Email IS NOT NULL)
```

- ◆ Vi får en error! Spørringen gir ikke mening.
- ◆ For å ta unionen av to spørringer må de returnere like mange kolonner
- ◆ Kolonnene må også ha kompatible typer
- ◆ Kan f.eks. ta unionen av en kolonne med `integer` og `decimal`, får da en kolonne av typen `numeric`
- ◆ Alle mengdeoperatorer må ha union-kompatibilitet mellom tabellene

Eksempel: Union

Finn navn på alle produkter som enten kommer fra eller er solgt til Norge

```
(SELECT p.product_name
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)
WHERE c.country = 'Norway')
UNION
(SELECT p.product_name
FROM products AS p
    INNER JOIN suppliers AS s USING (supplier_id)
WHERE s.country = 'Norway');
```


Snitt-operatøren

persons

ID	Name	Country
1	Per	UK
2	Mari	Norway
3	Ola	Norway
4	Ida	Italy
5	Carl	USA

companies

ID	Name	Country
1	Per's company	Germany
2	Fish'n trolls	Norway
3	Matpakke AS	Norway
4	Big Burgers	USA
5	Ysteriet	Norway

```
(SELECT Country
 FROM persons)
INTERSECT
(SELECT Country
 FROM companies)
```

Resultat:

Country
Norway
USA

```
(SELECT Country
 FROM person)
INTERSECT ALL
(SELECT Country
 FROM companies)
```

Resultat:

Country
Norway
Norway
USA

Eksempel: Snitt

Finn navnet på alle sjefer som har håndtert bestillinger.

```
SELECT first_name, last_name
  FROM employees
 WHERE employee_id IN (
    (SELECT reports_to
     FROM employees
    INTERSECT
    (SELECT employee_id
     FROM orders)
  );
```

Differanse-operatoren

companies

ID	Name	Country
1	Per's company	Germany
2	Fish'n trolls	Norway
3	Matpakke AS	Norway
4	Big Burgers	USA
5	Ysteriet	Norway

persons

ID	Name	Country
1	Per	UK
2	Mari	Norway
3	Ola	Norway
4	Ida	Italy
5	Carl	USA

```
(SELECT Country
 FROM companies)
EXCEPT
(SELECT Country
 FROM persons)
```

Resultat:

Country
Germany

```
(SELECT Country
 FROM companies)
EXCEPT ALL
(SELECT Country
 FROM persons)
```

Resultat:

Country
Germnay
Norway

Eksempel: Differanse

Finn navnet på alle produkter som selges i flasker med som ikke er i kategorien *Beverages*

```
SELECT product_name
  FROM products
 WHERE product_id IN (
    (SELECT product_id
      FROM products
     WHERE quantity_per_unit LIKE '%bottles%')
 EXCEPT
 (SELECT p.product_id
  FROM products AS p
   INNER JOIN categories AS c
     USING (category_id)
  WHERE c.category_name = 'Beverages')
 );
```

Mengdeoperatorer – oppførsel

Gitt en tabell t . Er følgende riktig?

- ◆ $t \text{ UNION } t = t$? Nei, **UNION** fjerner alle duplikater
- ◆ $t \text{ UNION ALL } t = t$? Nei, vi får hver rad i t to ganger
- ◆ $t \text{ INTERSECT } t = t$? Nei, samme som for **UNION**
- ◆ $t \text{ INTERSECT ALL } t = t$? Ja!
- ◆ $t \text{ EXCEPT } t$ blir tomt? Ja!
- ◆ $t \text{ EXCEPT ALL } t$ blir tomt? Ja!

EXISTS

- ◆ Av og til er vi kun interessert i om en del spørring *har et svar*, og ikke svaret i seg selv
- ◆ Typisk er dette når vi er interessert i å hente ut objekter med en bestemt egenskap, men hvor egenskapen kan avgjøres med en delspørring
- ◆ I slike tilfeller kan vi bruke **EXISTS** før en delspørring i **WHERE**-klausulen
- ◆ **EXISTS** q er sann for en spørring q dersom q har minst ett svar
- ◆ Kan også bruke **NOT EXISTS** q for å finne ut om q ikke har noen svar

EXISTS-nøkkelordet

companies

ID	Name	Country
1	Per's company	Germany
2	Fish'n trolls	Norway
3	Matpakke AS	Norway
4	Big Burgers	USA
5	Ysteriet	Norway

persons

ID	Name	Country
1	Per	UK
2	Mari	Norway
3	Ola	Norway
4	Ida	Italy
5	Carl	USA

```
SELECT p.Name
FROM persons AS p
WHERE NOT EXISTS (
    SELECT * -- Kan bruke hva som helst her
    FROM companies AS c
    WHERE c.country = p.country
);
```

Resultat:

p.Name
Per
Ida

Eksempel: EXISTS (1)

Finn navnet til alle sjefer på laveste nivå (ikke sjef for en sjef)

```
SELECT DISTINCT boss.employee_id, boss.first_name, boss.last_name
FROM employees AS boss
WHERE boss.employee_id IN (SELECT reports_to FROM employees) AND
  NOT EXISTS (
    SELECT *
    FROM employees AS e2 INNER JOIN employees AS e
      ON (e2.reports_to = e.employee_id)
    WHERE e.reports_to = boss.employee_id
  );
```


Eksempel: EXISTS (2)

Finn alle par av kunder og kategorier slik at kunden aldri har kjøpt noe fra den kategorien

```
SELECT c.company_name, cg.category_name
FROM customers AS c, categories AS cg
WHERE NOT EXISTS (
  SELECT *
  FROM orders AS so
    INNER JOIN order_details AS sod USING (order_id)
    INNER JOIN products AS sp USING (product_id)
    INNER JOIN categories AS scg USING (category_id)
  WHERE so.customer_id = c.customer_id AND
    cg.category_id = scg.category_id);
```

Mange måter å gjøre det samme på

Finn ID på alle kunder som ikke har bestilt noe:

Med EXCEPT

```
(SELECT customer_id
 FROM customers)
EXCEPT
(SELECT customer_id
 FROM orders);
```

Med NOT IN

```
SELECT customer_id
FROM customers
WHERE customer_id NOT IN (
    SELECT customer_id
    FROM orders);
```

Med NOT EXISTS

```
SELECT c.customer_id
FROM customers AS c
WHERE NOT EXISTS (
    SELECT * FROM orders AS o
    WHERE o.customer_id = c.customer_id
);
```

Med LEFT OUTER JOIN

```
SELECT c.customer_id
FROM customers AS c
    LEFT OUTER JOIN orders AS o
    USING (customer_id)
WHERE o.customer_id IS NULL;
```

CASE-uttrykk

- ◆ Av og til er det nyttig å kunne bytte ut verdier
- ◆ I SQL kan man bruke **CASE**-uttrykk for dette
- ◆ **CASE**-uttrykk har formen

```
CASE
  WHEN <condition1> THEN <expression1>
  WHEN <condition2> THEN <expression2>
  ...
  ELSE <expressionN>
END
```

- ◆ For eksempel:

```
SELECT product_name,
       CASE
         WHEN unit_price = 0 THEN 'Free'
         WHEN unit_price < 30 THEN 'Cheap'
         ELSE 'Expensive'
       END AS expensiveness
FROM products;
```

Eksempel: CASE

Finn ut hvor mange bestillinger i gjennomsnitt personene i de tre kategoriene "sjefer", "eiere" og "andre" har behandlet

```
WITH
  new_titles AS (
    SELECT customer_id,
           CASE
             WHEN contact_title LIKE '%Manager%' THEN 'Manager'
             WHEN contact_title LIKE '%Owner%' THEN 'Owner'
             ELSE 'Other'
           END AS title
    FROM customers
  )
SELECT nt.title,
       count(o.order_id)::float/count(DISTINCT nt.customer_id) AS nr_orders
FROM new_titles AS nt
     LEFT OUTER JOIN orders AS o
     USING (customer_id)
GROUP BY nt.title;
```

Mye vi ikke har sett på

Følgende er nyttige ting vi ikke har sett på (og ikke del av pensum):

- ◆ Viduspørringer

<https://www.postgresql.org/docs/current/tutorial-window.html>

- ◆ Rekursive spørringer

<https://www.postgresql.org/docs/current/queries-with.html>

- ◆ Lateral join

Sek. 7.2.1.5 i

<https://www.postgresql.org/docs/current/queries-table-expressions.html>

- ◆ Triggere

<https://www.postgresql.org/docs/current/plpgsql-trigger.html>

Eksempler: Rekursive spørringer (Ikke pensum) (1)

Finn alle tall fra 1 til 100

```
WITH RECURSIVE
  numbers AS (
    (SELECT 1 AS n)
    UNION
    (SELECT n+1 AS n
     FROM numbers
     WHERE n < 100)
  )
SELECT * FROM numbers;
```

Eksempler: Rekursive spørringer (Ikke pensum) (2)

Finn alle Fibonacci-tall mindre enn 100000

```
WITH RECURSIVE
  fibs AS (
    (SELECT 1 AS n, 1 AS m)
    UNION
    (SELECT m AS n, n+m AS m
     FROM fibs
     WHERE m < 100000)
  )
SELECT n FROM fibs;
```

Eksempler: Rekursive spørringer (Ikke pensum) (3)

Finn ut alle *sjef-av*-relasjoner (hvor dersom a er sjef for b og b er sjef for c er også a sjef for c)

```
WITH RECURSIVE
  bossof AS (
    (SELECT employee_id, reports_to
     FROM employees)
    UNION
    (SELECT e.employee_id, b.reports_to
     FROM employees AS e INNER JOIN bossof AS b
     ON (e.reports_to = b.employee_id))
  )
SELECT * FROM bossof;
```