



INF115 Lecture 2: *Queries on One Table*

Adriaan Ludl
Department of Informatics
University of Bergen

Spring Semester

Practical Aspects:

❖ Join the INF115 Discord:

- <https://discord.gg/bcPFnmvD>

❖ Take the Quizzes in the Modules to get the lecture slides on mitt.uib :

- <https://mitt.uib.no/courses/33533/modules>

➤ ***Registration forms to **physically attend** lectures from **next week**.***

- *Form for Tuesday's lecture will be published on mitt.uib today at 12.00*

Chapter 2: Queries on a single table



Learning Goals:

- Important use cases of the **database language SQL**
- Be able to use SQL *selection queries* to **read out data from a table**
 - How to **select rows** and **columns**
 - How to **sort rows** with respect to certain columns
 - How to **aggregate** data (compute *sums, averages*, etc.)
 - How to use **functions** and **operators**

SQL use cases

- ❖ Create new tables
- ❖ Work with tables:
 - Insert new rows (**rader**)
 - Delete rows
 - Updates rows
- ❖ Query / **Spørre**
 - Filter
 - Sort
 - Group
 - Join
- ❖ User administration
- ❖ Optimise search



A word cloud of SQL operations. The words are arranged in a roughly circular shape. The words include: 'insert', 'create user', 'create index', 'select', 'delete', 'create table', and 'update'. The words are in various colors (dark red, orange, brown) and sizes, with 'select' being the largest and most prominent.

Database Table *Employee / Ansatt*

AnsattNr	Etternavn	Fornavn	AnsattDato	Stilling	Lønn
1	Veum	Varg	01.01.1996	Løpegutt	383 000.00
2	Stein	Trude	10.10.2004	DBA	470 700.00
3	Dudal	Inger-Lise	24.12.2012	Sekretær	499 000.00
4	Hansen	Hans	23.08.2010	Programmerer	525 000.00
5	Bjørnsen	Henrik	01.01.2014	Tekstforfatter	575 000.00
6	Gredelin	Sofie	18.05.2012	Underdirektør	825 850.00
7	Zimmermann	Robert	17.05.1999	Regnskapsfører	575 000.00
8	Nilsen	Lise	03.04.2016	Direktør	875 340.00
11	Fosheim	Katinka	13.09.2015	Selger	620 000.00
13	Lovløs	Ada	12.08.2009	Programmerer	584 250.00
16	Ibsen	Bjørnstjerne	02.01.2012	Tekstforfatter	546 000.00
17	Fleksnes	Marve	17.05.2013	Lagerleder	520 120.00
20	Felgen	Reodor	12.12.2005	Sykkelreparatør	479 500.00
23	Karius	Jens	13.12.2015	Salgssekretær	480 390.00
29	Wirkola	Gabriel	21.04.2018	Sekretær	455 000.00

Example query / Eksempelspørring

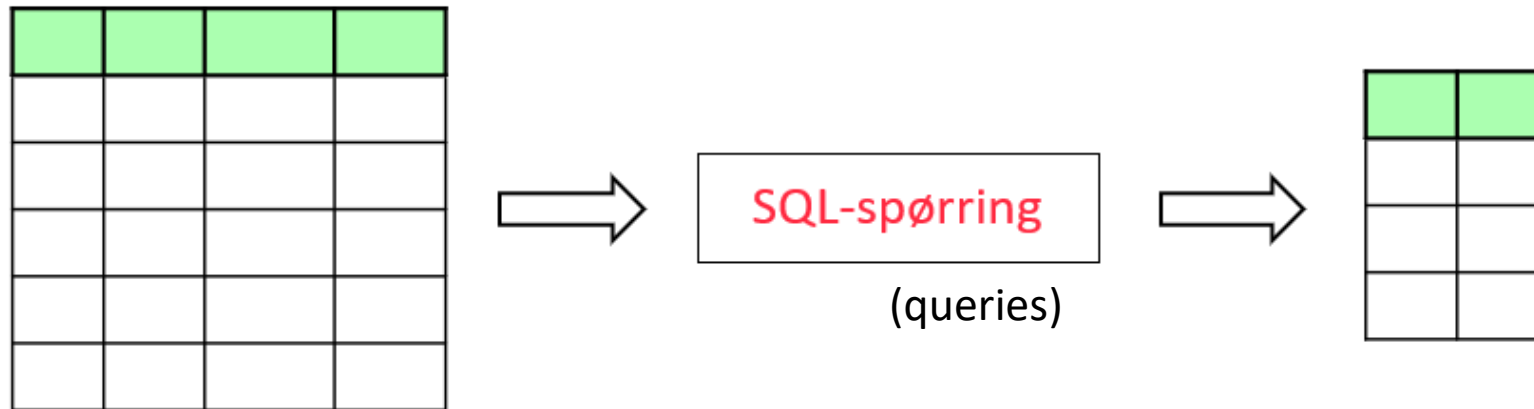
```
SELECT AnsattNr, Etternavn, Lønn  
FROM    Ansatt  
WHERE   Lønn < 480000
```

Query result:

AnsattNr	Etternavn	Lønn
1	Veum	kr 383 000.00
2	Stein	kr 470 700.00
20	Felgen	kr 479 500.00
29	Wirkola	kr 455 000.00

From table(s) to query results


- A **selection query** (**utvalgsspørring**) takes **tables** as *input* and gives as *output* a **query result** (**spørreresultat**) which is **also a table**.
- **First** we will look at **queries with 1 table**:



- How are selection queries performed by the DBMS ?

Performing selection queries / utvalgsspørringer

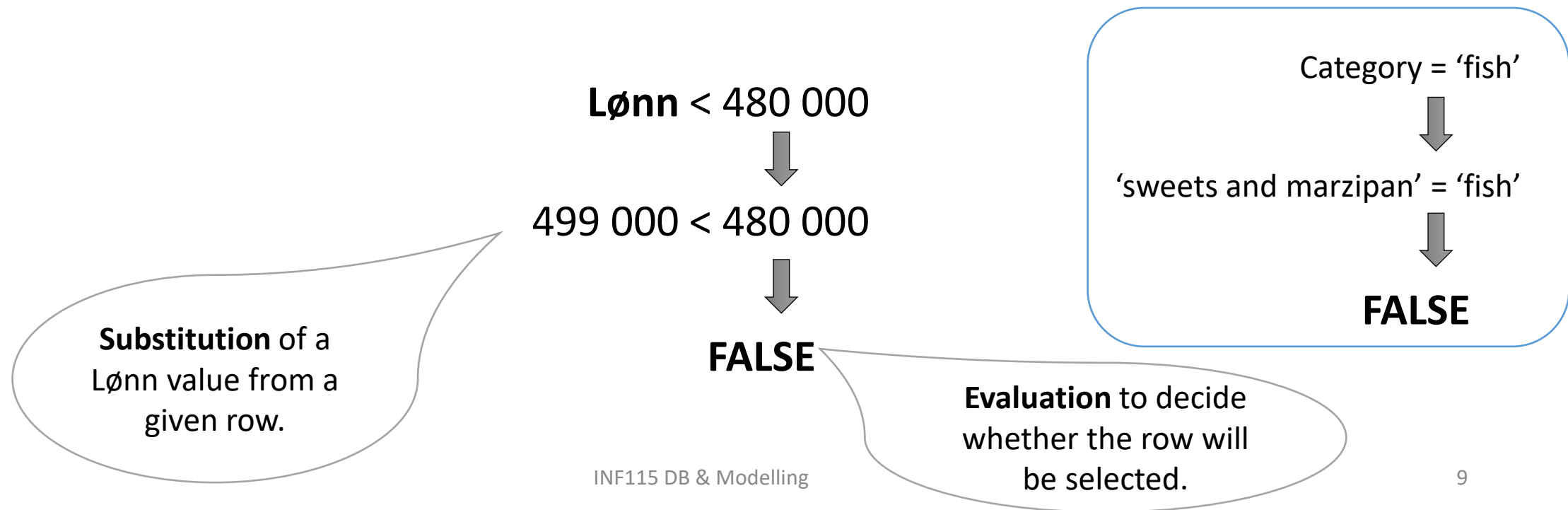
```
SELECT AnsattNr, Etternavn, Lønn  
FROM   Ansatt  
WHERE  Lønn < 480000
```



AnsattNr	Etternavn	Fornavn	AnsattDato	Stilling	Lønn
✓ 1	Veum	Varg	01.01.1996	Løpegutt	383 000.00
✓ 2	Stein	Trude	10.10.2004	DBA	470 700.00
3	Dudal	Inger-Lise	24.12.2012	Sekretær	499 000.00
...
✓ 20	Felgen	Reodor	12.12.2005	Sykkelreparatør	479 500.00
23	Karius	Jens	13.12.2015	Salgssekretær	480 390.00
✓ 29	Wirkola	Gabriel	21.04.2018	Sekretær	455 000.00

Substitution and Evaluation

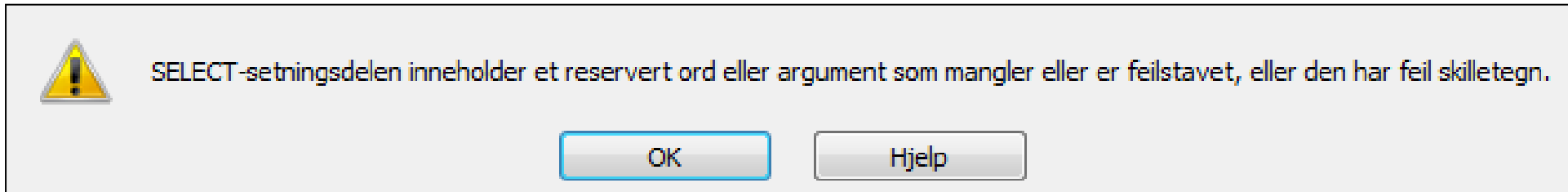
- The DBMS executes queries by going through the table **row by row**.
- For every row the *actual values* are **substituted** for the *column name*,
- And then the *conditional expression* is **evaluated** (as True or False).



Computers and formal language

SQL is a **formal** language :

- Precise rules describe how to form **allowed «sentences»**.
- Small **typos** lead to **error messages** or **unexpected results**



➤ Read the error messages !

- You need to **practice to interpret error messages**.
- Error messages do not always identify the error directly.

➤ Check that the query result makes sense:

- Queries can be **logically wrong** even you do not get an error message !



Building blocks in the query language SQL

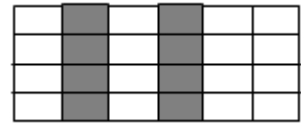
SQL-queries are built from:

- *Reserved words* (e.g. **SELECT** and **FROM ...**)
- *Names of tables and columns* (e.g. **Ansatt**, **Etternavn**, *CamelCase ...*)
- *Functions* (e.g. **UPPER ...**) and **operators** (e.g. **<, =, AND ...**)
- *Punctuation* (e.g. **comma, quotes ...**)

```
SELECT AnsattNr, Etternavn, Lønn
FROM   Ansatt
WHERE  Lønn < 480000 AND
      UPPER(Stilling) = 'SEKRETÆR'
```

Selection queries in SQL (SELECT)

- **First, simple queries on 1 table:**
 - Choose columns,
 - Choose rows,
 - Sort rows with respect to a column,
 - Insert computed columns,
 - Group and aggregate data (sum, average, ...).



Choose columns

- When we only want to see some of the columns:

```
SELECT AnsattNr, Etternavn  
FROM Ansatt
```

- When we want **all columns**:

```
SELECT *  
FROM Ansatt
```

- When we have selected a column, we can get many *repeated values*, such **duplicates** can easily be **removed**:

```
SELECT DISTINCT stilling  
FROM Ansatt
```



Choose rows

- We want to retrieve the rows that satisfy a given condition:

```
SELECT *  
FROM Ansatt  
WHERE Lønn < 480000
```

- A **condition** (betingelse) is an *expression* that is either **true** or **false**.



*Such **conditions** can be quite **complicated**.*



Quizz on SQL (part 1)

Please answer the practice quizz on mitt.uib now 😊
(you can take it again later if you want)

Link:

➤ <https://mitt.uib.no/courses/33533/quizzes/>

Logical operators: AND, OR, NOT

Used to build **compound conditions**,
examples:

(Lønn > 480000) **AND**
(Stilling = 'Sekretær')

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

(Lønn < 500000) **OR**
(Lønn > 700000)

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

NOT (Lønn <= 500000)

NOT	TRUE	FALSE
	FALSE	TRUE

Logical operators: AND, OR, NOT

Used to build **compound conditions**,
examples:

(Lønn > 480000) AND
(Stilling = 'Sekretær')

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

(Lønn < 500000) **OR**
(Lønn > 700000)

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

NOT (Lønn <= 500000)

NOT	TRUE	FALSE
	FALSE	TRUE

Logical operators: AND, OR, NOT

Used to build **compound conditions**,
examples:

(Lønn > 480000) AND
(Stilling = 'Sekretær')

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

(Lønn < 500000) OR
(Lønn > 700000)

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

NOT (Lønn <= 500000)

NOT	TRUE	FALSE
	FALSE	TRUE

Use parentheses!

We want to find the sellers/secretaries who earn more than 480.000 kroner per year.

```
SELECT *  
FROM   Ansatt  
WHERE  Lønn > 480000  
AND    Stilling = 'Selger'  
OR     Stilling = 'Sekretær'
```

- **Can the query be interpreted in several ways ?**
- *How many ways to place the parentheses ?*
- What shall be evaluated **first**: **AND** or **OR** ?
- ***What does the DBMS do if we do not place any parentheses ?***



Operator precedence (prioritet) rules

1. $-$ (unær minus, f.eks. -3)
2. $*$ $/$ $\%$
3. $+$ $-$ (binære operatorer, f.eks. $2+2$)
4. $<$ $<=$ $>$ $>=$ $=$ $<>$
5. **NOT**
6. **AND**
7. **OR**

Use parentheses!

We want to find the sellers/secretaries who earn more than 480.000 kroner per year.

```
SELECT *  
FROM   Ansatt  
WHERE  Lønn > 480000  
AND   Stilling = 'Selger'  
OR    Stilling = 'Sekretær'
```

- **Can the query be interpreted in several ways ?**
- *How many ways to place the parentheses ?*
- What shall be evaluated **first**: **AND** or **OR** ?
- ***What does the DBMS do if we do not place any parentheses ?***



Use parentheses!

We want to find the sellers/secretaries who earn more than 480.000 kroner per year.

Possible
but not as
desired !:

```
SELECT *  
FROM   Ansatt  
WHERE  ( Lønn > 480000  
        AND   stilling = 'Selger' )  
OR     stilling = 'Sekretær'
```

- Can the query be interpreted in several ways ?
- *How many ways to place the parentheses ?*
- What shall be evaluated **first**: **AND** or **OR** ?
- ***What does the DBMS do if we do not place any parentheses ?***



Use parentheses!

We want to find the sellers/secretaries who earn more than 480.000 kroner per year.

Correct:

```
SELECT *  
FROM   Ansatt  
WHERE  Lønn > 480000  
AND    ( stilling = 'selger'  
        OR      stilling = 'sekretær' )
```

- Can the query be interpreted in several ways ?
- *How many ways to place the parentheses ?*
- What shall be evaluated **first**: **AND** or **OR** ?
- ***What does the DBMS do if we do not place any parentheses ?***



Sorting and Comparisons

- We need to **compare in order to be able to sort** items.
 - **Numbers:** ... -2 < -1 < 0 < 1 < 2 < 3 < ...
 - **Letters:** a < b < c < ...
 - **Texts:** adam < anna < anne < anneli < david
 - **Dates:** 22.03.1982 < 07.02.1996 < 31.12.2018
- *Sorting in SQL* means **sorting rows** with respect to a given *column*.
 - Example below: some **rows** from the table *Ansatt* **sorted by income** (**Lønn**).

AnsattNr	Etternavn	Fornavn	AnsattDato	Stilling	Lønn
1	Veum	Varg	01.01.1996	Løpegutt	383 000.00
2	Stein	Trude	10.10.2004	DBA	470 700.00
20	Felgen	Reodor	12.12.2005	Sykkelreparatør	479 500.00
6	Gredelin	Sofie	18.05.2012	Underdirektør	825 850.00

Examples of Sorting in SQL

Sorted list of names:

```
SELECT AnsattNr, Etternavn  
FROM Ansatt  
ORDER BY Etternavn
```

Sorting using several criteria:

```
SELECT Etternavn, Stilling, Lønn  
FROM Ansatt  
ORDER BY Stilling ASC, Lønn DESC
```

- ❖ **ASC** gives **increasing** (or ascending) order,
- ❖ **DESC** gives **decreasing** (or descending) order.
- **ASC is the default !** So the first example above gives a list sorted by *increasing order*.

Examples of Sorting in SQL

Sorted list of names:

```
SELECT AnsattNr, Etternavn  
FROM Ansatt  
ORDER BY Etternavn
```

Sorting using several criteria:

```
SELECT Etternavn, Stilling, Lønn  
FROM Ansatt  
ORDER BY Stilling ASC, Lønn DESC  
                                  primary                                  secondary
```

Ordering Criteria

- ❖ **ASC** gives **increasing** (or ascending) order,
- ❖ **DESC** gives **decreasing** (or descending) order.
- **ASC is the default !** So the first example above gives a list sorted by *increasing order*.

15 minute break!
Lecture resumes at 11:...

Ordering example

Sorting using several criteria:

```
SELECT Etternavn, Stilling, Lønn  
FROM Ansatt  
ORDER BY Stilling ASC, Lønn DESC
```

AnsattNr	Etternavn	Fornavn	Stilling	Lønn
1	Veum	Varg	Selger	300 000
2	Stein	Trude	Programmerer	400 000
3	Dudal	Inger-Lise	Sekretær	300 000
4	Hansen	Hans	Programmerer	500 000
5	Bjørnsen	Henrik	Selger	500 000
6	Gredelin	Sofie	Direktør	800 000
7	Zimmermann	Robert	Programmerer	600 000

Sorting using several criteria:

```
SELECT Etternavn, Stilling, Lønn  
FROM Ansatt  
ORDER BY Stilling ASC, Lønn DESC
```

Table *Ansatt* in original order:

sorting

New ordering after sorting:

AnsattNr	Etternavn	Fornavn	Stilling	Lønn
1	Veum	Varg	Selger	300 000
2	Stein	Trude	Programmerer	400 000
3	Dudal	Inger-Lise	Sekretær	300 000
4	Hansen	Hans	Programmerer	500 000
5	Bjørnsen	Henrik	Selger	500 000
6	Gredelin	Sofie	Direktør	800 000
7	Zimmermann	Robert	Programmerer	600 000

AnsattNr	Etternavn	Fornavn	Stilling	Lønn
6	Gredelin	Sofie	Direktør	800 000
7	Zimmermann	Robert	Programmerer	600 000
4	Hansen	Hans	Programmerer	500 000
2	Stein	Trude	Programmerer	400 000
3	Dudal	Inger-Lise	Sekretær	300 000
5	Bjørnsen	Henrik	Selger	500 000
1	Veum	Varg	Selger	300 000

Wildcard operator % and Interval Search

Find all employees whose lastname begins with «F»:

```
SELECT *  
FROM   Ansatt  
WHERE  Etternavn >= 'F'  
AND    Etternavn <  'G'
```

In this case the **wildcard operator** (jokernotasjon) is simpler

(and more elegant):

```
SELECT *  
FROM   Ansatt  
WHERE  Etternavn LIKE 'F%'
```

- Why is it not meaningful to require **Etternavn = 'F%'** ?
- Find all employees whose lastname ends with «sen» !

Expressions in SELECT statements

Such expressions are allowed in SELECT statements:

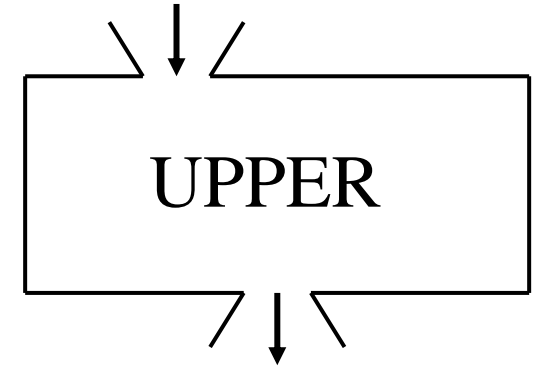
```
SELECT AnsattNr, Etternavn,  
       Lønn/12 AS LønnPrMåned  
FROM   Ansatt
```

- *Note:* Here it is not necessary to store both the yearly and monthly salary !
- **Lønn/12** is an **expression**.
- We give a **readable name** to this column in the output using the **AS** keyword.

Functions

- For example a name (or text) can be ***changed into capital letters***:

```
SELECT AnsattNr, UPPER(Etternavn)  
FROM   Ansatt
```



- **UPPER** is a **function** (in MySQL).
- A *function* can have ***several arguments***, but only **one value**.
- There are a large number of other functions, you can find them in the Appendix of the textbook and the online documentation of MySQL.

Try out some functions in MySQL

Experiment with functions to understand what they do! **Learning by doing !**

For example by writing and running a query without a FROM section:

```
SELECT UPPER('Hansen')
```

The answer is:

```
HANSEN
```

You can use MySQL as a basic calculator:

```
SELECT 2+2
```



Ongoing Experiments on
On Tables !

Operators and Expressions

Operators

- Arithmetic: `*, /, +, -`
- Comparisons: `>, <, =, >=, <=, <>`
- Wildcard: `LIKE`
- Test for null values: `IS NULL`
- Boolean: `NOT, AND, OR`
- Interval test: `BETWEEN ... AND ...`

We can **build expressions** from *concrete values*, *column names*, *functions* and *operators*:

`(stilling LIKE 'S%') AND ((Lønn/12)>40000)`

Date and Time

Find the duration of employment (ansettelsesforhold) in days:

```
SELECT Etternavn,  
       YEAR(AnsattDato),  
       CURDATE() - AnsattDato AS AntDager  
FROM Ansatt
```

Functions:

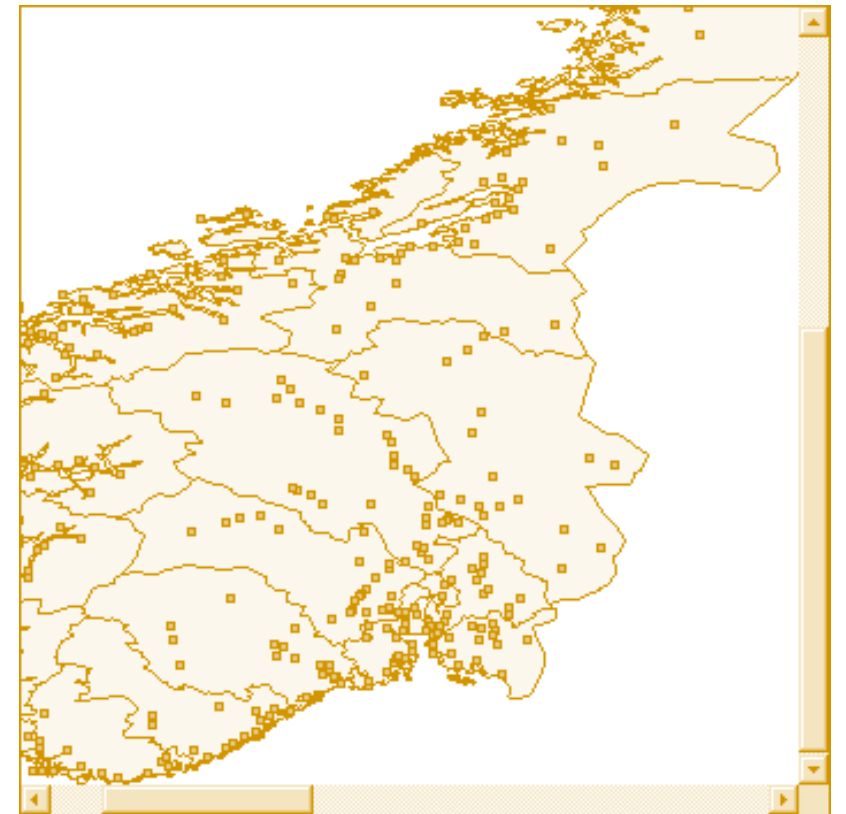
- Current time (now): **CURDATE**
- Extract parts of a date: **YEAR, MONTH, HOUR ...**
- **Operators** to compare dates or times: <, >, <=, >=, =

Note: *Names of functions may depend on the database system (DBS) used.*

Group and Aggregate Functions

Ola Nordmann lives in Hansegata 3, Andebu and earns ca. 320.000 kr. per year.

- Zooming into the map, at high resolution
We can see **attributes** of Ola.
- **Zooming out** we want to see **aggregated data**:
 - *Total income of a city or region,*
 - *Average income by gender or age,*
 - *etc ...*



Two kinds of columns

AnsattNr	Etternavn	Fornavn	Stilling	Lønn
1	Veum	Varg	Selger	300 000
2	Stein	Trude	Programmerer	400 000
3	Dudal	Inger-Lise	Sekretær	300 000
4	Hansen	Hans	Programmerer	500 000
5	Bjørnsen	Henrik	Selger	500 000
6	Gredelin	Sofie	Direktør	800 000
7	Zimmermann	Robert	Programmerer	600 000

- For the **Lønn** column (**numerical**): *compute sums, averages ...*
- For the **Stilling** column (**text**): *make groupings:*
 - for example we can compute an average of **Lønn** for each job category (**group**).

Simple Functions and Aggregate Functions

Operations on **single values** e.g.:

- UPPER: change text to capital letters
- YEAR: extract the year from a date

Operations on **collections of values**:

- **AVG**(col.name) average
- **SUM**(col.name) sum
- **COUNT**(*) number of rows
- **MIN**(col.name) minimum
- **MAX**(col.name) maximum

Quizz on SQL (part 2)

Please answer the practice quizz on mitt.uib now 😊
(you can take it again later if you want)

Link:

➤ <https://mitt.uib.no/courses/33533/quizzes/>

Aggregate Functions

- Average salary (snittlønn):

```
SELECT AVG(Lønn) AS SnittLønn  
FROM Ansatt
```




SnittLønn
485 714

1 row!

1 column!

- What about **total salary expenses** ?
- Or the **average salary for programmers** ?

COUNT and Null values

- Number of all employees (= number of rows):
`SELECT COUNT(*) FROM Ansatt`

- Number of employees with a salary (Lønn):
`SELECT COUNT(Lønn) FROM Ansatt`

***COUNT(Lønn)** does not count rows with **Null values** in the **Lønn** column.*

- Below is an **incorrect try** to compute the average:
`SELECT SUM(Lønn)/COUNT(*) FROM Ansatt`
➤ Use **AVG(Lønn)** instead !



Grouping

Find the average salary for each job category.

```
SELECT stilling, AVG(Lønn)  
FROM Ansatt  
GROUP BY stilling
```

- **Stilling only takes a few different values.**

- Group columns must appear in the result.
- Grouping is often used together with aggregate functions.

Query results

1. To find the **average salary for each job category**, first have to build **groups**:
seller, programmer ...
2. Then we compute the **average salary for each group**.
3. **The result has 4 rows** because there are 4 different values in *Stilling*.

Stilling	Lønn
Sekretær	300 000
Programmerer	500 000
Selger	400 000
Direktør	800 000

Group Conditions (HAVING)

- What will the result be here ?

```
SELECT stilling, AVG(Lønn), COUNT(*)  
FROM Ansatt  
GROUP BY stilling  
HAVING AVG(Lønn) > 380000
```

- **HAVING** on groups corresponds to **WHERE** on rows.
- *Note:* it is not allowed to use aggregate functions in WHERE conditions.



Conditions on rows and on groups:

```
SELECT stilling, AVG(Lønn), COUNT(*)  
FROM Ansatt  
WHERE Year(AnsattDato) > 1990  
GROUP BY stilling  
HAVING AVG(Lønn) > 380000
```

- How does the DBMS execute this query ?
- Why can these two conditions not be combined ?

Summary: Building selection queries



Selection queries (**SELECT**) on **1 table** follow the following pattern:

SELECT	<i>which columns do we want?</i>
FROM	<i>which table are we looking at?</i>
WHERE	<i>which rows do we want?</i>
GROUP BY	<i>Make groups based on which columns?</i>
HAVING	<i>which groups do we want?</i>
ORDER BY	<i>sort according to which columns?</i>

- **We can choose to use only some of these parts in a query.**
- We will learn about other reserved words soon.
- **SELECT is one of many SQL commands ...**



Practical Aspects:

❖ Join the INF115 Discord:

- <https://discord.gg/bcPFnmvD>

❖ Take the Quizzes in the Modules to get the lecture slides on mitt.uib :

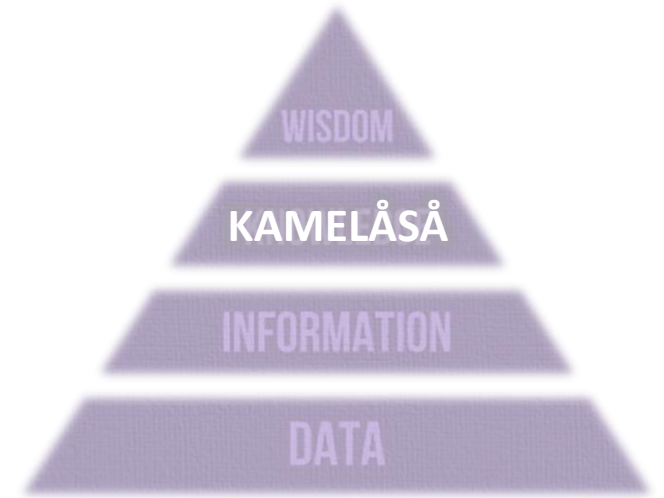
- <https://mitt.uib.no/courses/33533/modules>

➤ ***Registration forms to **physically attend** lectures from **next week**.***

- *Form for Tuesday's lecture will be published on mitt.uib today at 12.00*



Questions ?



- **Next lecture next Tuesday**
- **Register for physical attendance using the link published in the announcement on mitt.uib !**