

# Introduction to SQL

Leif Harald Karlsen  
leifhka@ifi.uio.no

19.10.22



# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases

# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases
- ◆ The most common query language for such databases

# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases
- ◆ The most common query language for such databases
- ◆ Used to formulate queries, i.e. questions to a database

# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases
- ◆ The most common query language for such databases
- ◆ Used to formulate queries, i.e. questions to a database
- ◆ SQL is also used to manipulate the database

# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases
- ◆ The most common query language for such databases
- ◆ Used to formulate queries, i.e. questions to a database
- ◆ SQL is also used to manipulate the database
  - ◆ To create tables,

# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases
- ◆ The most common query language for such databases
- ◆ Used to formulate queries, i.e. questions to a database
- ◆ SQL is also used to manipulate the database
  - ◆ To create tables,
  - ◆ insert data,

# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases
- ◆ The most common query language for such databases
- ◆ Used to formulate queries, i.e. questions to a database
- ◆ SQL is also used to manipulate the database
  - ◆ To create tables,
  - ◆ insert data,
  - ◆ delete data,



# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases
- ◆ The most common query language for such databases
- ◆ Used to formulate queries, i.e. questions to a database
- ◆ SQL is also used to manipulate the database
  - ◆ To create tables,
  - ◆ insert data,
  - ◆ delete data,
  - ◆ ...

# SQL: Structured Query Language

---

- ◆ SQL is a query language for relational databases
- ◆ The most common query language for such databases
- ◆ Used to formulate queries, i.e. questions to a database
- ◆ SQL is also used to manipulate the database
  - ◆ To create tables,
  - ◆ insert data,
  - ◆ delete data,
  - ◆ ...
- ◆ Made in 1974, but first standard appeared in 1986

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )
  - ◆ “For every element in the list  $L$  print the value of the element”  
(`for e in L: print(e)`)

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )
  - ◆ “For every element in the list  $L$  print the value of the element”  
(`for e in L: print(e)`)



# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )
  - ◆ “For every element in the list  $L$  print the value of the element”  
(`for e in L: print(e)`)
- ◆ A query language is a precise language for expressing *questions to a database*

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )
  - ◆ “For every element in the list  $L$  print the value of the element”  
(`for e in L: print(e)`)
- ◆ A query language is a precise language for expressing *questions to a database*
- ◆ Such questions are often called a *query*

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )
  - ◆ “For every element in the list  $L$  print the value of the element”  
(`for e in L: print(e)`)
- ◆ A query language is a precise language for expressing *questions to a database*
- ◆ Such questions are often called a *query*
- ◆ SQL is declarative in nature, e.g.:

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )
  - ◆ “For every element in the list  $L$  print the value of the element”  
(`for e in L: print(e)`)
- ◆ A query language is a precise language for expressing *questions to a database*
- ◆ Such questions are often called a *query*
- ◆ SQL is declarative in nature, e.g.:
  - ◆ “Which elements have a name starting with 'P'?”

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )
  - ◆ “For every element in the list  $L$  print the value of the element”  
(`for e in L: print(e)`)
- ◆ A query language is a precise language for expressing *questions to a database*
- ◆ Such questions are often called a *query*
- ◆ SQL is declarative in nature, e.g.:
  - ◆ “Which elements have a name starting with 'P'?”
  - ◆ “Let 'Parents' be all elements having a 'hasChild'-related element”

# Python vs. SQL

---

- ◆ A programming language (e.g. Python) is a precise language for expressing *sequences of instructions to a computer*
- ◆ Python is imperative in nature, e.g.:
  - ◆ “Set the value of  $x$  to 2” ( $x = 2$ )
  - ◆ “Add  $x$  and  $y$  and assign the result to  $z$ ” ( $z = x+y$ )
  - ◆ “For every element in the list  $L$  print the value of the element”  
(`for e in L: print(e)`)
- ◆ A query language is a precise language for expressing *questions to a database*
- ◆ Such questions are often called a *query*
- ◆ SQL is declarative in nature, e.g.:
  - ◆ “Which elements have a name starting with 'P'?”
  - ◆ “Let 'Parents' be all elements having a 'hasChild'-related element”
  - ◆ “How many employees have a boss which earn more than 1000000 KR?”

# Python vs. SQL

---

- ◆ A Python-program tells the computer *how to compute* the answers you want

# Python vs. SQL

---

- ◆ A Python-program tells the computer *how to compute* the answers you want
- ◆ An SQL-query tells the computer *what to compute*,



# Python vs. SQL

---

- ◆ A Python-program tells the computer *how to compute* the answers you want
- ◆ An SQL-query tells the computer *what to compute*,
- ◆ and its up to the database to decide *how to find* the answers

# Types of SQL-queries

---

The first keyword in a query states what it does:

# Types of SQL-queries

---

The first keyword in a query states what it does:

**SELECT** retrieves information (answers a query)

# Types of SQL-queries

---

The first keyword in a query states what it does:

**SELECT** retrieves information (answers a query)

**CREATE** creates something (e.g. a new table)

# Types of SQL-queries

---

The first keyword in a query states what it does:

**SELECT** retrieves information (answers a query)

**CREATE** creates something (e.g. a new table)

**DROP** deletes something (e.g. a table)

# Types of SQL-queries

---

The first keyword in a query states what it does:

**SELECT** retrieves information (answers a query)

**CREATE** creates something (e.g. a new table)

**DROP** deletes something (e.g. a table)

**INSERT** inserts data into a table

# Types of SQL-queries

---

The first keyword in a query states what it does:

**SELECT** retrieves information (answers a query)

**CREATE** creates something (e.g. a new table)

**DROP** deletes something (e.g. a table)

**INSERT** inserts data into a table

**DELETE** deletes data from a table

# Types of SQL-queries

---

The first keyword in a query states what it does:

**SELECT** retrieves information (answers a query)

**CREATE** creates something (e.g. a new table)

**DROP** deletes something (e.g. a table)

**INSERT** inserts data into a table

**DELETE** deletes data from a table

We will only focus on **SELECT**.



## SELECT-queries

---

- ◆ (Simple) `SELECT`-queries have the form:

```
SELECT <columns>  
FROM <tables>
```

## SELECT-queries

---

- ◆ (Simple) `SELECT`-queries have the form:

```
SELECT <columns>  
FROM <tables>
```

- ◆ where `<columns>` is a list of column names,
- ◆ `<tables>` is a list of table names

## SELECT-queries

---

- ◆ (Simple) `SELECT`-queries have the form:

```
SELECT <columns>  
FROM <tables>
```

- ◆ where `<columns>` is a list of column names,
- ◆ `<tables>` is a list of table names

The result of such a query is a *new table* consisting of:

- ◆ the columns listed in `<columns>`,
- ◆ based on the rows from the tables in `<tables>`

# Select single column

---

Query retrieving all names in Patient-table

```
SELECT Name  
FROM Patient;
```

Answers

# Select single column

---

## Query retrieving all names in Patient-table

```
SELECT Name
FROM Patient;
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select single column

---

## Query retrieving all names in Patient-table

```
SELECT Name  
FROM Patient;
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select single column

---

## Query retrieving all names in Patient-table

```
SELECT Name  
FROM Patient;
```

## Answers

Name
Anna Consuma
Peter Young
Carla Smith
Sam Penny
John Mill
Yvonne Potter

# Select multiple columns

---

Query retrieving all names and date of birth pairs in Patient-table

```
SELECT Name, Birthdate  
FROM Patient;
```

Answers



# Select multiple columns

Query retrieving all names and date of birth pairs in Patient-table

```
SELECT Name, Birthdate  
FROM Patient;
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select multiple columns

Query retrieving all names and date of birth pairs in Patient-table

```
SELECT Name, Birthdate
FROM Patient;
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select multiple columns

---

Query retrieving all names and date of birth pairs in Patient-table

```
SELECT Name, Birthdate  
FROM Patient;
```

## Answers

Name	Birthdate
Anna Consuma	1978-10-09
Peter Young	2009-03-01
Carla Smith	1986-06-14
Sam Penny	1961-01-09
John Mill	1989-11-16
Yvonne Potter	1971-04-12

# Selecting all columns

---

Query retrieving all tuples in Patient-table

```
SELECT *  
FROM Patient;
```

Answers

# Selecting all columns

## Query retrieving all tuples in Patient-table

```
SELECT *  
FROM Patient;
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Selecting all columns

## Query retrieving all tuples in Patient-table

```
SELECT *  
FROM Patient;
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

- ◆ We will use DBFiddle to interact with SQL

# DBFiddle

---

- ◆ We will use DBFiddle to interact with SQL
- ◆ DBFiddle is a webpage giving SQL-access to a database



# DBFiddle

---

- ◆ We will use DBFiddle to interact with SQL
- ◆ DBFiddle is a webpage giving SQL-access to a database
- ◆ Mostly used for small examples or illustrating a point

# DBFiddle

---

- ◆ We will use DBFiddle to interact with SQL
- ◆ DBFiddle is a webpage giving SQL-access to a database
- ◆ Mostly used for small examples or illustrating a point
- ◆ Database created on the fly when you access webpage

# DBFiddle

---

- ◆ We will use DBFiddle to interact with SQL
- ◆ DBFiddle is a webpage giving SQL-access to a database
- ◆ Mostly used for small examples or illustrating a point
- ◆ Database created on the fly when you access webpage
- ◆ Supports all of SQL (queries are executed over real RDBMSs)

# DBFiddle

---

- ◆ We will use DBFiddle to interact with SQL
- ◆ DBFiddle is a webpage giving SQL-access to a database
- ◆ Mostly used for small examples or illustrating a point
- ◆ Database created on the fly when you access webpage
- ◆ Supports all of SQL (queries are executed over real RDBMSs)
- ◆ However, no security, no users, does not scale, etc.

# Exmples

SELECT

---

`https://dbfiddle.uk/Wu5i\_q6E?hide=2`

Find all observations in observation-table

# Exmples

## SELECT

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

Find all observations in observation-table

```
SELECT *  
FROM observation;
```

# Exmples

## SELECT

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

Find all observations in observation-table

```
SELECT *  
FROM observation;
```

Find genus and common name for all species

# Exmples

## SELECT

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

Find all observations in observation-table

```
SELECT *  
FROM observation;
```

Find genus and common name for all species

```
SELECT genus, common_name  
FROM species;
```



## Adding the `WHERE`-clause

---

- ◆ We often just want specific rows

## Adding the `WHERE`-clause

---

- ◆ We often just want specific rows
- ◆ we can then use a `WHERE`-clause to pick out the rows we want

## Adding the `WHERE`-clause

---

- ◆ We often just want specific rows
- ◆ we can then use a `WHERE`-clause to pick out the rows we want
- ◆ SQL-queries then have the form

```
SELECT <columns>  
FROM <tables>  
WHERE <condition>
```

## Adding the `WHERE`-clause

---

- ◆ We often just want specific rows
- ◆ we can then use a `WHERE`-clause to pick out the rows we want
- ◆ SQL-queries then have the form

```
SELECT <columns>  
FROM <tables>  
WHERE <condition>
```

- ◆ `<condition>` is an expression than can be true or false for each row

## Adding the `WHERE`-clause

---

- ◆ We often just want specific rows
- ◆ we can then use a `WHERE`-clause to pick out the rows we want
- ◆ SQL-queries then have the form

```
SELECT <columns>  
FROM <tables>  
WHERE <condition>
```

- ◆ `<condition>` is an expression than can be true or false for each row
- ◆ The result is now same as before, but contains only the rows where `<condition>` holds.

# Select specific values

---

Query retrieving birth date of patient with name John Mill

```
SELECT Birthdate
FROM Patient
WHERE Name = 'John Mill'
```

Answers

# Select specific values

Query retrieving birth date of patient with name John Mill

```
SELECT Birthdate
FROM Patient
WHERE Name = 'John Mill'
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select specific values

Query retrieving birth date of patient with name John Mill

```
SELECT Birthdate
FROM Patient
WHERE Name = 'John Mill'
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6



# Select specific values

Query retrieving birth date of patient with name John Mill

```
SELECT Birthdate
FROM Patient
WHERE Name = 'John Mill'
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select specific values

---

Query retrieving birth date of patient with name John Mill

```
SELECT Birthdate  
FROM Patient  
WHERE Name = 'John Mill'
```

Answers

Birthdate
1989-11-16

# Select range of values

---

Query for names of patients that have more than 10 treatments

```
SELECT Name  
  FROM Patient  
 WHERE NrTreatments > 10
```

Answers

# Select range of values

Query for names of patients that have more than 10 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments > 10
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select range of values

Query for names of patients that have more than 10 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments > 10
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select range of values

Query for names of patients that have more than 10 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments > 10
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select range of values

---

Query for names of patients that have more than 10 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments > 10
```

Answers

Name
Anna Consuma
Sam Penny

# Select with multiple restrictions

---

Query for birth dates and names of patients which have between 4 and 10 treatments

```
SELECT Birthdate, Name
FROM Patient
WHERE NrTreatments > 4 AND
      NrTreatments < 10
```

Answers



# Select with multiple restrictions

Query for birth dates and names of patients which have between 4 and 10 treatments

```
SELECT Birthdate, Name
FROM Patient
WHERE NrTreatments > 4 AND
      NrTreatments < 10
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select with multiple restrictions

Query for birth dates and names of patients which have between 4 and 10 treatments

```
SELECT Birthdate, Name
FROM Patient
WHERE NrTreatments > 4 AND
      NrTreatments < 10
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select with multiple restrictions

Query for birth dates and names of patients which have between 4 and 10 treatments

```
SELECT Birthdate, Name
FROM Patient
WHERE NrTreatments > 4 AND
      NrTreatments < 10
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select with multiple restrictions

Query for birth dates and names of patients which have between 4 and 10 treatments

```
SELECT Birthdate, Name
FROM Patient
WHERE NrTreatments > 4 AND
      NrTreatments < 10
```

## Answers

Birthdate	Name
1986-06-14	Carla Smith
1989-11-16	John Mill
1971-04-12	Yvonne Potter

## Select with restrictions on multiple columns

---

Query for Birthdate and number of treatments for patients which have less than or equal to 8 treatments and is born before 01.01.1988

```
SELECT Birthdate, NrTreatments
FROM Patient
WHERE NrTreatments <= 8 AND
      Birthdate < '1988-01-01'
```

Answers

# Select with restrictions on multiple columns

Query for Birthdate and number of treatments for patients which have less than or equal to 8 treatments and is born before 01.01.1988

```
SELECT Birthdate, NrTreatments
FROM Patient
WHERE NrTreatments <= 8 AND
      Birthdate < '1988-01-01'
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select with restrictions on multiple columns

Query for Birthdate and number of treatments for patients which have less than or equal to 8 treatments and is born before 01.01.1988

```
SELECT Birthdate, NrTreatments
FROM Patient
WHERE NrTreatments <= 8 AND
      Birthdate < '1988-01-01'
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select with restrictions on multiple columns

Query for Birthdate and number of treatments for patients which have less than or equal to 8 treatments and is born before 01.01.1988

```
SELECT Birthdate, NrTreatments
FROM Patient
WHERE NrTreatments <= 8 AND
      Birthdate < '1988-01-01'
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6



## Select with restrictions on multiple columns

Query for Birthdate and number of treatments for patients which have less than or equal to 8 treatments and is born before 01.01.1988

```
SELECT Birthdate, NrTreatments
FROM Patient
WHERE NrTreatments <= 8 AND
      Birthdate < '1988-01-01'
```

### Answers

Birthdate	NrTreatments
1986-06-14	8
1971-04-12	6

## Select with OR

---

Query for names of patients who have less than or equal to 5 treatments or greater than or equal to 15 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments <= 5 OR
      NrTreatments >= 15
```

Answers

# Select with OR

Query for names of patients who have less than or equal to 5 treatments or greater than or equal to 15 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments <= 5 OR
      NrTreatments >= 15
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select with OR

Query for names of patients who have less than or equal to 5 treatments or greater than or equal to 15 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments <= 5 OR
      NrTreatments >= 15
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select with OR

Query for names of patients who have less than or equal to 5 treatments or greater than or equal to 15 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments <= 5 OR
      NrTreatments >= 15
```

## Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

# Select with OR

Query for names of patients who have less than or equal to 5 treatments or greater than or equal to 15 treatments

```
SELECT Name
FROM Patient
WHERE NrTreatments <= 5 OR
      NrTreatments >= 15
```

## Answers

Name
Anna Consuma
Peter Young

## Select with both AND and OR

---

Query for names of patients who have between 5 and 15 treatments and is born after '2000-01-01'

```
SELECT Name FROM Patient
WHERE (NrTreatments <= 5 OR
      NrTreatments >= 15) AND
      Birthdate > '2000-01-01'
```

Answers

## Select with both AND and OR

Query for names of patients who have between 5 and 15 treatments and is born after '2000-01-01'

```
SELECT Name FROM Patient
WHERE (NrTreatments <= 5 OR
      NrTreatments >= 15) AND
      Birthdate > '2000-01-01'
```

### Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6



## Select with both AND and OR

Query for names of patients who have between 5 and 15 treatments and is born after '2000-01-01'

```
SELECT Name FROM Patient
WHERE (NrTreatments <= 5 OR
      NrTreatments >= 15) AND
      Birthdate > '2000-01-01'
```

### Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

## Select with both AND and OR

Query for names of patients who have between 5 and 15 treatments and is born after '2000-01-01'

```
SELECT Name FROM Patient
WHERE (NrTreatments <= 5 OR
      NrTreatments >= 15) AND
      Birthdate > '2000-01-01'
```

### Answers

PatientID	Name	Birthdate	NrTreatments
0	Anna Consuma	1978-10-09	19
1	Peter Young	2009-03-01	1
2	Carla Smith	1986-06-14	8
3	Sam Penny	1961-01-09	14
4	John Mill	1989-11-16	8
5	Yvonne Potter	1971-04-12	6

## Select with both AND and OR

---

Query for names of patients who have between 5 and 15 treatments and is born after '2000-01-01'

```
SELECT Name FROM Patient
WHERE (NrTreatments <= 5 OR
      NrTreatments >= 15) AND
      Birthdate > '2000-01-01'
```

### Answers

Name
Peter Young

# Exmples

WHERE

---

`https://dbfiddle.uk/Wu5i_q6E?hide=2`

Find date of all observations in Oslo

# Exmples

WHERE

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

Find date of all observations in Oslo

```
SELECT observed_time
FROM observation
WHERE location = 'Oslo';
```

# Exmples

WHERE

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

Find date of all observations in Oslo

```
SELECT observed_time
FROM observation
WHERE location = 'Oslo';
```

Find common name for all species that are blacklisted or have a global conservation between 3 and 5.

# Exmples

## WHERE

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

Find date of all observations in Oslo

```
SELECT observed_time
FROM observation
WHERE location = 'Oslo';
```

Find common name for all species that are blacklisted or have a global conservation between 3 and 5.

```
SELECT common_name
FROM species
WHERE blacklisted OR
      (global_conservation >= 3 AND
       global_conservation <= 5);
```

# Functions and operators

---

- ◆ Can also use all the normal mathematical operators on values



# Functions and operators

---

- ◆ Can also use all the normal mathematical operators on values
- ◆ I.e. +, -, \*, /, etc.

# Functions and operators

---

- ◆ Can also use all the normal mathematical operators on values
- ◆ I.e. +, -, \*, /, etc.
- ◆ E.g. if a treatment consists of taking 4 pills, how many pills have each patient taken?

# Functions and operators

---

- ◆ Can also use all the normal mathematical operators on values
- ◆ I.e. +, -, \*, /, etc.
- ◆ E.g. if a treatment consists of taking 4 pills, how many pills have each patient taken?

```
SELECT Name, NrTreatments * 4 AS NrPills  
FROM Patients;
```

# Functions and operators

---

- ◆ Can also use all the normal mathematical operators on values
- ◆ I.e. +, -, \*, /, etc.
- ◆ E.g. if a treatment consists of taking 4 pills, how many pills have each patient taken?

```
SELECT Name, NrTreatments * 4 AS NrPills  
FROM Patients;
```

- ◆ How old is each patient that have taken more than 10 pills?

# Functions and operators

---

- ◆ Can also use all the normal mathematical operators on values
- ◆ I.e. +, -, \*, /, etc.
- ◆ E.g. if a treatment consists of taking 4 pills, how many pills have each patient taken?

```
SELECT Name, NrTreatments * 4 AS NrPills
FROM Patients;
```

- ◆ How old is each patient that have taken more than 10 pills?

```
SELECT Name, current_date - Birthdate AS Age
FROM Patients
WHERE NrTreatments * 4 > 10;
```

## Aggregates: Sum, avg, min and max and count

---

- ◆ Sometimes we want to find the minimum, maximum, sum or average of all values in a column

## Aggregates: Sum, avg, min and max and count

---

- ◆ Sometimes we want to find the minimum, maximum, sum or average of all values in a column
- ◆ This can be done by using the *aggregate functions* `min`, `max`, `sum`, `avg`

## Aggregates: Sum, avg, min and max and count

---

- ◆ Sometimes we want to find the minimum, maximum, sum or average of all values in a column
- ◆ This can be done by using the *aggregate functions* `min`, `max`, `sum`, `avg`
- ◆ E.g. what is the average number of treatments?



## Aggregates: Sum, avg, min and max and count

---

- ◆ Sometimes we want to find the minimum, maximum, sum or average of all values in a column
- ◆ This can be done by using the *aggregate functions* `min`, `max`, `sum`, `avg`
- ◆ E.g. what is the average number of treatments?

```
SELECT avg(NrTreatments) AS avg_nr_treatments
FROM Patients;
```

- ◆ Can use `count(*)` to count the number of rows in the result of a query

## Aggregates: Sum, avg, min and max and count

---

- ◆ Sometimes we want to find the minimum, maximum, sum or average of all values in a column
- ◆ This can be done by using the *aggregate functions* `min`, `max`, `sum`, `avg`
- ◆ E.g. what is the average number of treatments?

```
SELECT avg(NrTreatments) AS avg_nr_treatments
FROM Patients;
```

- ◆ Can use `count(*)` to count the number of rows in the result of a query
- ◆ E.g. how many patients are there born after '1990-01-01'?

## Aggregates: Sum, avg, min and max and count

---

- ◆ Sometimes we want to find the minimum, maximum, sum or average of all values in a column
- ◆ This can be done by using the *aggregate functions* `min`, `max`, `sum`, `avg`
- ◆ E.g. what is the average number of treatments?

```
SELECT avg(NrTreatments) AS avg_nr_treatments
FROM Patients;
```

- ◆ Can use `count(*)` to count the number of rows in the result of a query
- ◆ E.g. how many patients are there born after '1990-01-01'?

```
SELECT count(*) AS avg_nr_treatments
FROM Patients
WHERE Birthdate > '1990-01-01';
```

# Exmples

WHERE

---

`https://dbfiddle.uk/Wu5i\_q6E?hide=2`

How old are the observations in Oslo?

# Exmples

WHERE

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

How old are the observations in Oslo?

```
SELECT current_date - observed_time AS age
FROM observation
WHERE location = 'Oslo';
```

# Exmples

WHERE

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

How old are the observations in Oslo?

```
SELECT current_date - observed_time AS age
FROM observation
WHERE location = 'Oslo';
```

What is the average local conservation for non-blacklisted species?

# Exmples

## WHERE

---

[https://dbfiddle.uk/Wu5i\\_q6E?hide=2](https://dbfiddle.uk/Wu5i_q6E?hide=2)

How old are the observations in Oslo?

```
SELECT current_date - observed_time AS age
FROM observation
WHERE location = 'Oslo';
```

What is the average local conservation for non-blacklisted species?

```
SELECT avg(local_conservation) AS avg_local
FROM species
WHERE NOT blacklisted;
```

## SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query



## SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names

## SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The **WHERE**-clause picks out which rows should be part of the answer

## SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The **WHERE**-clause picks out which rows should be part of the answer
  - ◆ Evaluates to either **true** or **false** for each row

## SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The **WHERE**-clause picks out which rows should be part of the answer
  - ◆ Evaluates to either **true** or **false** for each row
  - ◆ Similar to an expression in a Python's `if`-test

## SELECT in a nutshell

---

- ◆ The `FROM`-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The `WHERE`-clause picks out which rows should be part of the answer
  - ◆ Evaluates to either `true` or `false` for each row
  - ◆ Similar to an expression in a Python's `if`-test
  - ◆ Variables are column names denoting the row's value in that column

## SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The **WHERE**-clause picks out which rows should be part of the answer
  - ◆ Evaluates to either **true** or **false** for each row
  - ◆ Similar to an expression in a Python's `if`-test
  - ◆ Variables are column names denoting the row's value in that column
  - ◆ Use parenthesis to group statements

# SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The **WHERE**-clause picks out which rows should be part of the answer
  - ◆ Evaluates to either **true** or **false** for each row
  - ◆ Similar to an expression in a Python's `if`-test
  - ◆ Variables are column names denoting the row's value in that column
  - ◆ Use parenthesis to group statements
- ◆ The **SELECT**-clause selects which columns to be part of the answer

# SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The **WHERE**-clause picks out which rows should be part of the answer
  - ◆ Evaluates to either **true** or **false** for each row
  - ◆ Similar to an expression in a Python's **if**-test
  - ◆ Variables are column names denoting the row's value in that column
  - ◆ Use parenthesis to group statements
- ◆ The **SELECT**-clause selects which columns to be part of the answer
  - ◆ Can also reorder columns



# SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The **WHERE**-clause picks out which rows should be part of the answer
  - ◆ Evaluates to either **true** or **false** for each row
  - ◆ Similar to an expression in a Python's `if`-test
  - ◆ Variables are column names denoting the row's value in that column
  - ◆ Use parenthesis to group statements
- ◆ The **SELECT**-clause selects which columns to be part of the answer
  - ◆ Can also reorder columns
  - ◆ Use `*` to select all columns

# SELECT in a nutshell

---

- ◆ The **FROM**-clause states which table(s) should be used to answer the query
  - ◆ Just a list of table names
- ◆ The **WHERE**-clause picks out which rows should be part of the answer
  - ◆ Evaluates to either **true** or **false** for each row
  - ◆ Similar to an expression in a Python's **if**-test
  - ◆ Variables are column names denoting the row's value in that column
  - ◆ Use parenthesis to group statements
- ◆ The **SELECT**-clause selects which columns to be part of the answer
  - ◆ Can also reorder columns
  - ◆ Use **\*** to select all columns
  - ◆ Can use aggregates (**min**, **max**, **avg**, **sum** and **count**)

## Notes on writing SQL

---

SQL does not care about indent and newlines like Python, so

# Notes on writing SQL

---

SQL does not care about indent and newlines like Python, so

```
SELECT Birthdate  
FROM Patients  
WHERE NrTreatments > 5;
```

# Notes on writing SQL

---

SQL does not care about indent and newlines like Python, so

```
SELECT Birthdate  
  FROM Patients  
 WHERE NrTreatments > 5;
```

```
SELECT Birthdate FROM Patients  
 WHERE NrTreatments > 5;
```

# Notes on writing SQL

---

SQL does not care about indent and newlines like Python, so

```
SELECT Birthdate
  FROM Patients
 WHERE NrTreatments > 5;
```

```
SELECT Birthdate FROM Patients
 WHERE NrTreatments > 5;
```

```
SELECT Birthdate
FROM Patients WHERE NrTreatments > 5;
```

# Notes on writing SQL

---

SQL does not care about indent and newlines like Python, so

```
SELECT Birthdate
  FROM Patients
 WHERE NrTreatments > 5;
```

```
SELECT Birthdate FROM Patients
 WHERE NrTreatments > 5;
```

```
SELECT Birthdate
FROM Patients WHERE NrTreatments > 5;
```

```
SELECT Birthdate FROM Patients WHERE NrTreatments > 5;
```

are all allowed and represents the same query.

# Notes on writing SQL

---

- ◆ For SQL-keywords and names of tables and columns, SQL is case-insensitive



## Notes on writing SQL

---

- ◆ For SQL-keywords and names of tables and columns, SQL is case-insensitive
- ◆ That is, it does not distinguish between upper and lower case characters

# Notes on writing SQL

---

- ◆ For SQL-keywords and names of tables and columns, SQL is case-insensitive
- ◆ That is, it does not distinguish between upper and lower case characters
- ◆ So
  - ◆ `SELECT Name FROM Patients;`
  - ◆ `select name from patients;`are equivalent queries

# Notes on writing SQL

---

- ◆ For SQL-keywords and names of tables and columns, SQL is case-insensitive
- ◆ That is, it does not distinguish between upper and lower case characters
- ◆ So
  - ◆ `SELECT Name FROM Patients;`
  - ◆ `select name from patients;`are equivalent queries
- ◆ However, SQL is case-sensitive for all values
  - ◆ so 'Anna' and 'anna' are two different values

# Notes on writing SQL

---

- ◆ For SQL-keywords and names of tables and columns, SQL is case-insensitive
- ◆ That is, it does not distinguish between upper and lower case characters
- ◆ So

- ◆ `SELECT Name FROM Patients;`

- ◆ `select name from patients;`

are equivalent queries

- ◆ However, SQL is case-sensitive for all values
  - ◆ so 'Anna' and 'anna' are two different values
- ◆ Use -- (two dashes) to write a comment (ignored by the database), e.g.

```
SELECT Name --This is a comment
FROM Patients
```

## Translating a question into SQL

---

“What are the names of the patients that have more than 5 treatments?”

## Translating a question into SQL

---

“What are the names of the patients that have more than 5 treatments?”

“Select the names of the patients that have more than 5 treatments”

## Translating a question into SQL

---

“What are the names of the patients that have more than 5 treatments?”

“Select the names of the patients that have more than 5 treatments”

“Select the `Names` from the `Patients` table where `5 < NrTreatments`”

## Translating a question into SQL

---

“What are the names of the patients that have more than 5 treatments?”

“Select the names of the patients that have more than 5 treatments”

“Select the Names from the Patients table where 5 < NrTreatments”

“Select ~~the~~ Names from ~~the~~ Patients ~~table~~ where 5 < NrTreatments”



## Translating a question into SQL

---

“What are the names of the patients that have more than 5 treatments?”

“Select the names of the patients that have more than 5 treatments”

“Select the Names from the Patients table where 5 < NrTreatments”

“Select the Names from the Patients table where 5 < NrTreatments”

```
SELECT Name FROM Patients WHERE 5 < NrTreatments;
```

(See *SQL Queries for Mere Mortals* for more examples)

# Programs generating SQL

---

If one goes to `http://finn.no`'s "Bolig til salgs" and put:

and click on "Søk"

It will generate an SQL-query looking something like this:

```
SELECT *
  FROM boliger£\pause£
 WHERE (sted = 'Oslo'
        OR sted = 'Akershus')
        £\pause£AND pris <= 50
        £\pause£AND pris >= 30
        £\pause£AND ant_rom >=
```

# Programs generating SQL

---

If one goes to `http://finn.no`'s "Bolig til salgs" and put:

- ◆ Sted: Oslo eller Akershus

and click on "Søk"

It will generate an SQL-query looking something like this:

```
SELECT *
  FROM boliger£\pause£
 WHERE (sted = 'Oslo'
        OR sted = 'Akershus')
        £\pause£AND pris <= 50
        £\pause£AND pris >= 30
        £\pause£AND ant_rom >=
```

# Programs generating SQL

---

If one goes to `http://finn.no`'s "Bolig til salgs" and put:

- ◆ Sted: Oslo eller Akershus
- ◆ Makspris: 5,000,000,-

and click on "Søk"

It will generate an SQL-query looking something like this:

```
SELECT *
  FROM boliger£\pause£
 WHERE (sted = 'Oslo'
        OR sted = 'Akershus')
        £\pause£AND pris <= 50
        £\pause£AND pris >= 30
        £\pause£AND ant_rom >=
```

# Programs generating SQL

---

If one goes to <http://finn.no>'s "Bolig til salgs" and put:

- ◆ Sted: Oslo eller Akershus
- ◆ Makspris: 5,000,000,-
- ◆ Minste pris: 3,000,000,-

and click on "Søk"

It will generate an SQL-query looking something like this:

```
SELECT *
  FROM boliger£\pause£
 WHERE (sted = 'Oslo'
        OR sted = 'Akershus')
        £\pause£AND pris <= 50
        £\pause£AND pris >= 30
        £\pause£AND ant_rom >=
```

# Programs generating SQL

---

If one goes to `http://finn.no`'s "Bolig til salgs" and put:

- ◆ Sted: Oslo eller Akershus
- ◆ Makspris: 5,000,000,-
- ◆ Minste pris: 3,000,000,-
- ◆ Antall rom: 3

and click on "Søk"

It will generate an SQL-query looking something like this:

```
SELECT *
  FROM boliger£\pause£
 WHERE (sted = 'Oslo'
        OR sted = 'Akershus')
        £\pause£AND pris <= 50
        £\pause£AND pris >= 30
        £\pause£AND ant_rom >=
```

## CREATE and INSERT (not part of curriculum)

---

- ◆ SQL is used for all interaction with the database
- ◆ To create a table, we use the `CREATE`-command
- ◆ E.g. to create the `Patient`-table, we can write:

## CREATE and INSERT (not part of curriculum)

---

- ◆ SQL is used for all interaction with the database
- ◆ To create a table, we use the `CREATE`-command
- ◆ E.g. to create the Patient-table, we can write:

```
CREATE TABLE Patients(  
    PatientID int, Name text, Birthdate date, NrTreatments int  
);
```

- ◆ Similarly we can use `INSERT` to insert data into a table
- ◆ E.g. to add the data into the Patients-table, we can write:



# CREATE and INSERT (not part of curriculum)

---

- ◆ SQL is used for all interaction with the database
- ◆ To create a table, we use the `CREATE`-command
- ◆ E.g. to create the Patient-table, we can write:

```
CREATE TABLE Patients(  
    PatientID int, Name text, Birthdate date, NrTreatments int  
);
```

- ◆ Similarly we can use `INSERT` to insert data into a table
- ◆ E.g. to add the data into the Patients-table, we can write:

```
INSERT INTO Patients VALUES  
(0, 'Anna Consuma', '1978-10-09', 19),  
(1, 'Peter Young', '2009-03-01', 1),  
(2, 'Carla Smith', '1986-06-14', 8),  
(3, 'Sam Penny', '1961-01-09', 14),  
(4, 'John Mill', '1989-11-16', 8),  
(5, 'Yvonne Potter', '1971-04-12', 6);
```