Aggregate functions and set operators

- During this week we will learn:
  - How to use aggregate functions `COUNT`, `SUM`, `AVG`, `MIN` and `MAX`
  - How to use the `GROUP BY` statement with aggregate functions
  - How to combine result tables with `UNION`, `INTERSECT` and `UNION` set operators

Aggregate functions

- Performing some calculation for multiple rows so that the end result is a *single value* is a common query problem
- Example of such query is calculating the count of rows in a certain table
- For example, how can we calculate the number of courses in the `Course` table?
- Functions that perform such operations are referred to as *aggregate functions*

The COUNT aggregate function

- The `COUNT` aggregate function returns the *total number of rows* that match the specified criteria:

```
-- what's the number of courses in the Course table?
SELECT COUNT(*) as number_of_courses FROM Course
```

- The result table contains a single row:

number_of_courses
7

The COUNT aggregate function

- We can also filter the rows the aggregate function operates on using the `WHERE` clause:

```
-- what's the number of courses with more than 3 credits?
SELECT COUNT(*) as number_of_courses FROM Course
WHERE credits > 3
```

- The result table contains a single row:

number_of_courses
2

The SUM aggregate function

- The `SUM` aggregate function takes the name of a column as an argument and returns the *sum of all the values* in that column:

```
-- what's the sum of credits in the Course table?
SELECT SUM(credits) as sum_of_credits FROM Course
```

- The result table contains a single row:

sum_of_credits
24

The AVG aggregate function

- The `AVG` aggregate function returns the *average value* in a column:

```
-- what's the average grade from course with code "a730"?
SELECT AVG(grade) as average_grade FROM CourseGrade
WHERE course_code = 'a730'
```

- The result table contains a single row:

average_grade
3

The MIN aggregate function

- The `MIN` function returns the *smallest value* in a column

```
-- what's the lowest grade from course with code "a730"?
SELECT MIN(grade) as lowest_grade FROM CourseGrade
WHERE course_code = 'a730'
```

- The result table contains a single row:

lowest_grade
1

The MAX aggregate function

- The `MAX` function returns the *largest value* in a column

```
-- what's the highest grade from course with code "a730"?
SELECT MAX(grade) as highest_grade FROM CourseGrade
WHERE course_code = 'a730'
```

- The result table contains a single row:

highest_grade
5

Multiple aggregate functions in a single query

- We can have multiple aggregate functions in the same query:

```
-- what's the highest and lowest grade from course with code "a730"?
SELECT MAX(grade) as highest_grade, MIN(grade) as lowest_grade FROM CourseGrade
WHERE course_code = 'a730'
```

- The result table contains a single row with two columns:

highest_grade	lowest_grade
5	1

Grouping the aggregated rows

- So, an aggregate function performs a calculation for multiple rows so that the end result is a single value
- If the result table always contains just a single row, how can we write a query such as, *what's the average grade from each course?*
- To achieve this, we need to *group* the rows and perform the aggregate function for each group separately
- This can be done using the `GROUP BY` statement

The GROUP BY statement

- The `GROUP BY` statement uses a column or a group of columns to form groups of rows which the aggregate function operators on:

```
-- what's the average grade from each course?
SELECT course_code, AVG(grade) as average_grade FROM CourseGrade
-- form the groups using the course_code
GROUP BY course_code
```

The GROUP BY statement

- The result table will a row for each group having the aggregation function result. In the example's case the average grade for each course code:

course_code	average_grade
a290	2
a450	3
a480	2
a730	3

The GROUP BY statement

- It is worth noting that in the `SELECT` statement we can only select columns that are either aggregate functions or columns used in the `GROUP BY` statement:

```sql
-- ❌ student_number is not an aggreagate function, nor it is in the GROUP BY statement.
-- This will lead into an error
SELECT course_code, student_number, AVG(grade) as average_grade FROM CourseGrade
GROUP BY course_code
```

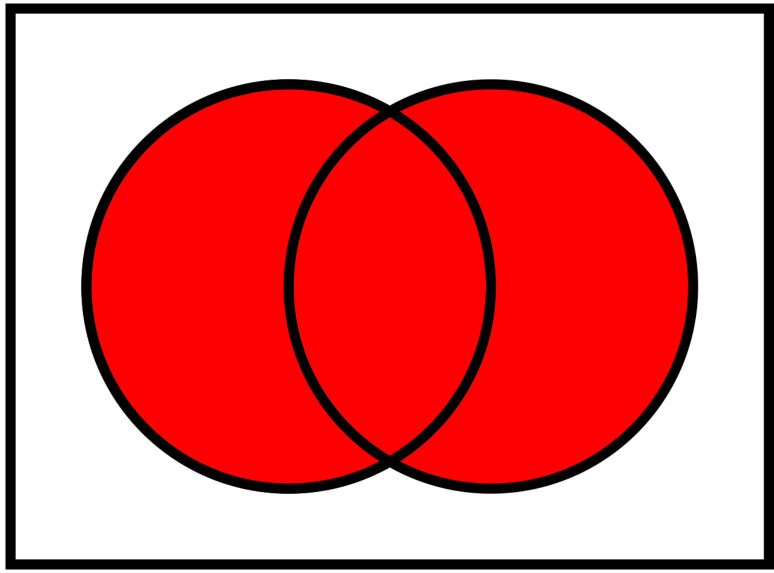
- This causes the following error:

> Column 'CourseGrade.student_number' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause

Combining results tables with set operators

- We can use the results from *multiple result tables* using the `UNION`, `EXCEPT`, and `INTERSECT` *set operators*
- For example, the `UNION` operator returns *all* the rows from two or more result tables *without duplicate values*:

```sql
-- What are all the surnames among teachers and students?
SELECT surname FROM Teacher
UNION
SELECT surname FROM Student
```



The UNION operator

```
SELECT surname FROM Teacher
```

surname
Huhta
Hellerus

```
SELECT surname FROM Student
```

surname
Kokki
Kuikka

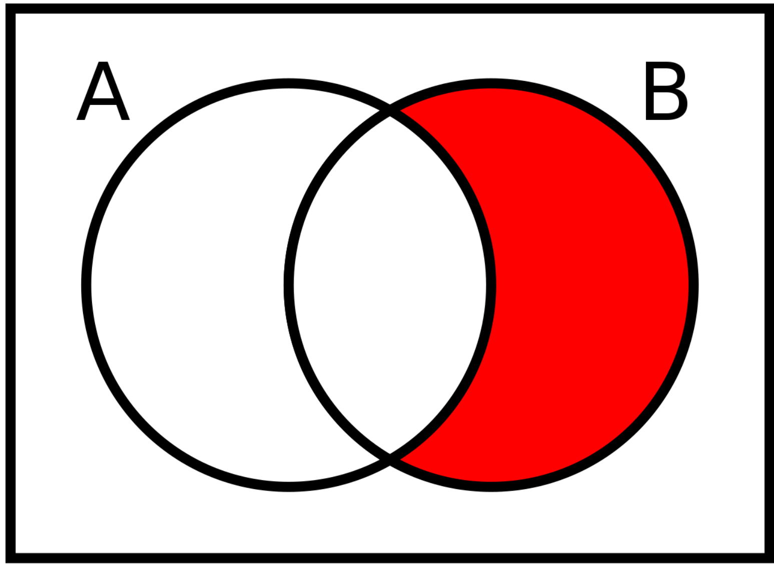
```
SELECT surname FROM Teacher
UNION
SELECT surname FROM Student
```

surname
Huhta
Hellerus
Kokki
Kuikka

The EXCEPT operator

- The `EXCEPT` operator returns only the rows from the first result table that are *not included* in the second result table

```
-- What are the campus cities that no student lives in?
SELECT city FROM Campus
EXCEPT
SELECT city FROM Student
```



The EXCEPT operator

```
SELECT city FROM Campus
```

city
Helsinki
Vantaa

```
SELECT city FROM Student
```

city
Helsinki
Espoo

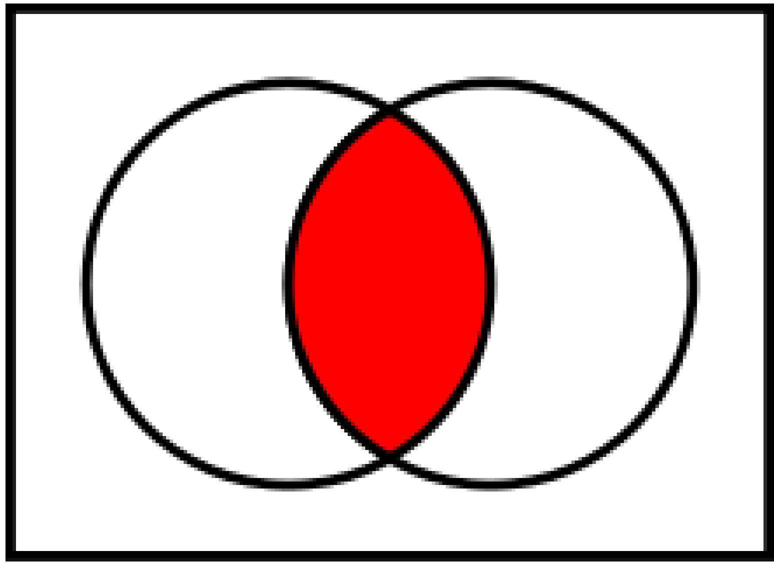
```
SELECT city FROM Campus
EXCEPT
SELECT city FROM Student
```

city
Vantaa

The INTERSECT operator

- The `INTERSECT` operator returns only the rows that *exist in both* result tables

```
-- What are the campus cities that have students living in them?
SELECT city FROM Campus
INTERSECT
SELECT city FROM Student
```



The EXCEPT operator

```
SELECT city FROM Campus
```

city
Helsinki
Vantaa

```
SELECT city FROM Student
```

city
Helsinki
Espoo

```
SELECT city FROM Campus
INTERSECT
SELECT city FROM Student
```

city
Helsinki

The set operators

- ⚠️ With set operators, the column names and data types of each `SELECT` statement *must match*:

```
-- ❌ first_name column is missing from the latter SELECT statement.
-- This will lead into an error.
SELECT surname, first_name FROM Teacher
UNION
SELECT surname FROM Student
```