

Aggregate functions and set operators

- The learning objectives for this week are:
 - Knowing what kind of query problems *aggregate functions* solve
 - Knowing how to use the aggregate functions `COUNT` , `SUM` , `AVG` , `MIN` and `MAX`
 - Knowing what kind of query problems *set operators* solve
 - Knowing how to combine result tables with `UNION` , `EXCEPT` and `INTERSECT` 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, or sum of a column values in a certain table
- For example, "what is the total number of courses?", or "what is the sum of male teachers' salaries"?
- Functions that perform such operations are referred to as *aggregate functions*

The COUNT aggregate function

```
COUNT ( * | { [ DISTINCT ] column_expression } )
```

- 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 COUNT aggregate function

- We can also provide a column name for the `COUNT` aggregate function in which case the function returns the number of the *non-null values* of the given column:

```
-- what's the number of students with an email address?  
SELECT COUNT(email) as number_of_students_with_email  
FROM Student
```

The COUNT aggregate function

| student_number | email |
|----------------|-----------------|
| o354 | 0354@takkula.fi |
| o410 | 0410@takkula.fi |
| o473 | NULL |

```
-- count all rows, total_number_of_students is 3
```

```
SELECT COUNT(*) as total_number_of_students
```

```
FROM Student
```

```
-- count rows with non-null email column value, number_of_students_with_email is 2
```

```
SELECT COUNT(email) as number_of_students_with_email
```

```
FROM Student
```

The SUM aggregate function

```
SUM ( [ DISTINCT ] column_expression )
```

- The `SUM` aggregate function returns the *sum of all the values* of a column:

```
-- what's the sum of salaries of female teachers?  
SELECT SUM(salary) as sum_of_salaries FROM Teacher  
WHERE gender = 'F'
```

- The result table contains a single row:

| sum_of_salaries |
|-----------------|
|-----------------|

| |
|-----------|
| 102273.00 |
|-----------|

The AVG aggregate function

```
AVG ( [ DISTINCT ] column_expression )
```

- The `AVG` aggregate function returns the *average value* of 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 AVG aggregate function

- Calculating the average includes a division operation, which can produce decimal numbers
- To avoid losing the decimal part of the average, we need to cast integer column values to a `DECIMAL` type:

```
-- multiplying an integer with 1.0 will end up with a DECIMAL type
SELECT AVG(grade * 1.0) as average_grade FROM CourseGrade
WHERE course_code = 'a730'
```

| average_grade |
|---------------|
| 3.333333 |

The AVG aggregate function

- We can limit the number of decimal places in the result by using casting the result to a `DECIMAL` type with specific precision (the total number of decimal digits stored) and scale (the number of decimal digits stored to the right of the decimal point):

```
-- use scale of 2 in the DECIMAL type to round to two decimals places
SELECT CAST(AVG(grade * 1.0) AS DECIMAL(9, 2)) as average_grade
FROM CourseGrade
WHERE course_code = 'a730'
```

| average_grade |
|---------------|
| 3.33 |

The MIN aggregate function

```
MIN ( column_expression )
```

- The `MIN` function returns the *smallest value* of 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

```
MAX ( column_expression )
```

- The `MAX` function returns the *largest value* of 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 |

Only operating on distinct values

- The `COUNT`, `SUM` and `AVG` aggregate functions support the `DISTINCT` keyword for only operating on *distinct values*:

```
-- how many different grades have been given?  
SELECT COUNT(DISTINCT grade) as number_of_different_grades FROM CourseGrade
```

Combining aggregate function and non-aggregate function columns

- If we use an aggregate function, we can't include non-aggregate function columns to the `SELECT` statement*:

```
-- ✅ only aggregate function columns, all good here  
SELECT COUNT(*) as number_of_courses FROM Course
```

```
-- ❌ combination of aggregate function and non-aggregate function columns, this won't work  
SELECT course_name, COUNT(*) as number_of_courses FROM Course
```

- * That is, unless the non-aggregate function columns are included in a `GROUP BY` clause, but we will cover that later

Combining aggregate function and non-aggregate function columns

- If it would be possible, how would the RDMS know, which `course_name` to display in the result table?

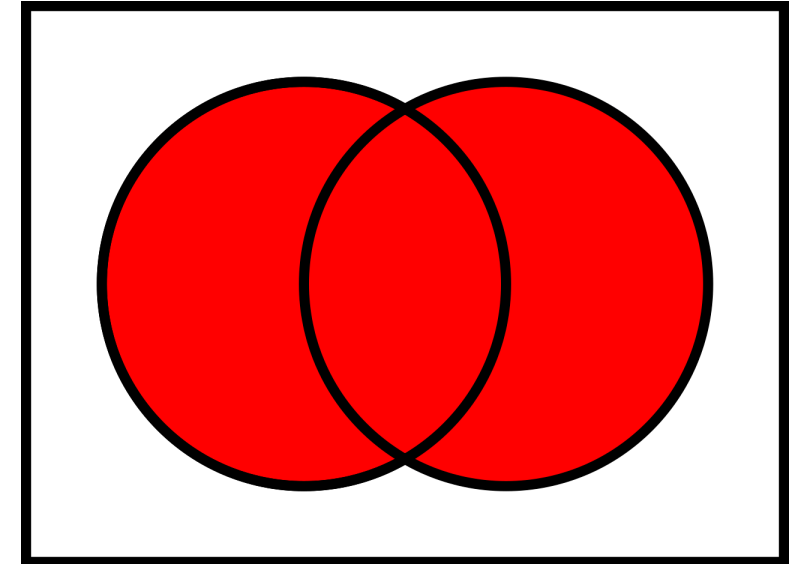
```
-- ✗ combination of aggregate function and non-aggregate function columns, this won't work  
SELECT course_name, COUNT(*) as number_of_courses FROM Course
```

| course_name | number_of_courses |
|-------------|-------------------|
| ? | 2 |

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*:

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



The UNION operator

"What are all the surnames among teachers and students?"

```
SELECT surname FROM Teacher
```

| surname |
|----------|
| Huhta |
| Hellerus |

```
SELECT surname FROM Student
```

| surname |
|---------|
| Kokki |
| Kuikka |

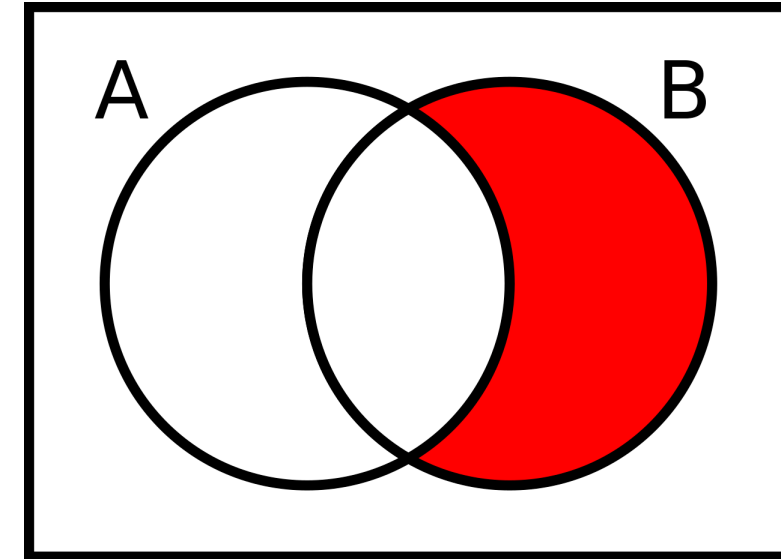
```
SELECT surname FROM Teacher  
UNION  
SELECT surname FROM Student
```

| 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 *without duplicate values*:

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



The EXCEPT operator

"What are the campus cities that no student lives in?"

```
SELECT city FROM Campus
```

| city |
|----------|
| Helsinki |
| Vantaa |

```
SELECT city FROM Student
```

| city |
|----------|
| Helsinki |
| Espoo |

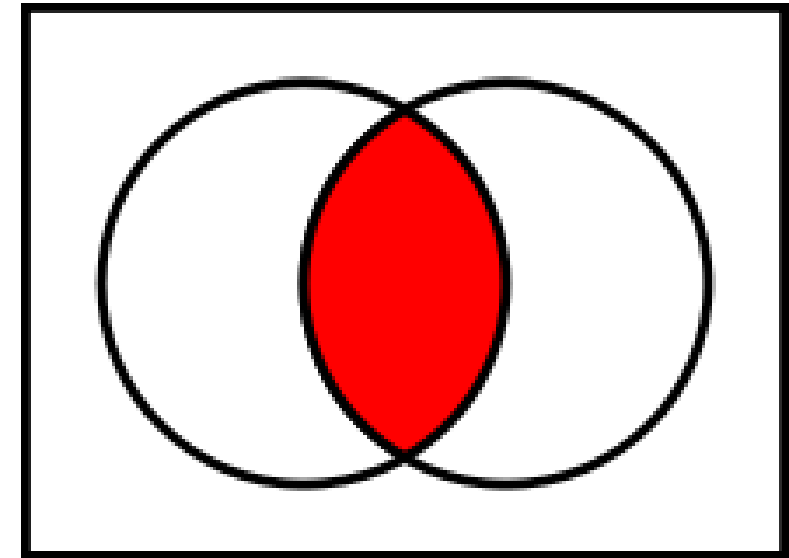
```
SELECT city FROM Campus  
EXCEPT  
SELECT city FROM Student
```

| city |
|--------|
| Vantaa |

The INTERSECT operator

- The `INTERSECT` operator returns only the rows that *exist in both result tables without duplicate values*:

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



The EXCEPT operator

"What are the campus cities that have students living in them?"

```
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
```

Summary

- We can perform calculations on multiple rows so that the end result is a single row using *aggregate functions*
- The `COUNT` aggregate function returns the *total number of rows*
- The `SUM` aggregate function returns the *sum of all the values* of a column
- The `AVG` aggregate function returns the *average of values* of a column
- The `MIN` aggregate function returns the *minimum value* of a column
- The `MAX` aggregate function returns the *maximum value* of a column
- The `UNION` , `EXCEPT` and `INTERSECT` *set operators* can be used to combine results of multiple result tables