

Tomas Sandnes  
[tomas.sandnes@kristiania.no](mailto:tomas.sandnes@kristiania.no)

Week 4

# DB1102 / PGR 111 – DATABASES



# Today's topics

*(Today's chapters: 4 in Norwegian book, 4.1 in English book)*

- Exercises walkthrough, selected tasks
- More on dates
- JOIN:
  - Cartesian product
  - (Standard) JOIN
  - LEFT JOIN and RIGHT JOIN
  - (Some info on outer and natural join)



# Exercises walkthrough

# Exercises walkthrough, selected tasks

- On students' request (*thank you!*), we're looking into tasks:
  - 7, 8, 11, and 13 (from last week).
- These are about:
  - Creating a table containing a foreign key.
  - Copying contents from table `person` to table `email`.
  - The `DATE_FORMAT` method.
  - The `information_schema` metadata.

# More on dates

# Dates in MySQL

- Internally MySQL stores dates on the format: 'YYYY-MM-DD'. Example: '1814-12-24'.

```
SELECT *  
FROM person;
```

	ID	Ssn	Name	DateOfBirth
▶	1	12345678901	Ola Nordmann	1912-02-20
	2	98765432109	Unknown	1990-01-01
	3	10101010101	Rollo Tomasi	1999-12-31
*	NULL	NULL	NULL	NULL

- If we want to insert date values, we need to enclose them in apostrophes (as for text).
  - If we don't, MySQL interprets it as a mathematical expression: *YYYY minus MM minus DD*.

```
UPDATE person  
SET DateOfBirth = '1990-01-01'  
WHERE Ssn = '98765432109';
```

# Dates in MySQL – cont.

- We can write dates in other formats by using:

- `STR_TO_DATE(...)` for input.

```
UPDATE person
SET DateOfBirth = STR_TO_DATE('20.02.1912', '%d.%m.%Y')
WHERE Name = 'Ola Nordmann';
```

Can write date with day first (not year), if we specify this for MySQL.

- `DATE_FORMAT(...)` for output.

```
SELECT Name, DATE_FORMAT(DateOfBirth, '%d-%m-%Y') AS BirthDate
FROM person;
```

	name	BirthDate
▶	Ola Nordmann	20-02-1912
	Unknown	01-01-1990
	Rollo Tomasi	31-12-1999

- For details on how to use these functions and their date formatting, see:
  - [STR TO DATE\(...\)](#) and [DATE FORMAT\(...\)](#)

# JOIN



# SQL: Querying multiple tables

- What to do if we want to query multiple tables at once?

owner

Id	Name
1	Per Persen
2	Ola Olsen
3	Kari Nordmann

car

RegNr	Model	Owner_Id
EL22222	Tesla Model S	1
KH33333	Ferrari Spider	3
ZX22222	Volvo V70	NULL

- Example:
  - *"I want to retrieve **carowners' name** & **carowners' registered cars** (their RegNr + Model)."*

# Kartesisk produkt og SQL

- **NOT** ok syntax:
  - Returns way to many rows in the answer!
- This is called **cartesian product**.

```
SELECT *  
FROM owner, car;
```

	Id	Name	RegNr	Model	Owner_Id
▶	3	Kari Nordmann	EL22222	Tesla Model S	1
	2	Ola Olsen	EL22222	Tesla Model S	1
	1	Per Persen	EL22222	Tesla Model S	1
	3	Kari Nordmann	KH33333	Ferrari Spider	3
	2	Ola Olsen	KH33333	Ferrari Spider	3
	1	Per Persen	KH33333	Ferrari Spider	3
	3	Kari Nordmann	ZX22222	Volvo V70	NULL
	2	Ola Olsen	ZX22222	Volvo V70	NULL
	1	Per Persen	ZX22222	Volvo V70	NULL

# Cartesian product

- The Cartesian product operation gives as output the amount that combines each and every row ("tuple") in Table R with each and every row in Table S.

R	S
a	1
b	2

– Formula:  $R \times S$

- What will be the Cartesian product ( $R \times S$ )?

$R \times S$	
a	1
b	1
a	2
b	2

- Cartesian product:
  - We add columns to each other, multiply rows with each other.

# JOIN

- Correct SQL for joining two tables:

```
SELECT *  
FROM owner  
JOIN car ON Id = Owner_Id;
```

	Id	Name	RegNr	Model	Owner_Id
▶	1	Per Persen	EL22222	Tesla Model S	1
	3	Kari Nordmann	KH33333	Ferrari Spider	3

- We merge the tables based on the PK  $\leftrightarrow$  FK relationship.
  - We only retrieve the rows that have a common id.
  - SQL syntax for this: JOIN ... ON

# JOIN – cont.

- **JOIN** gives us all columns (R + S) by default, but only for the relevant, connected rows from both tables.
  - We can manually pick the columns we want to display. For example, retrieve *"owner's name, car's registration number, car's model"*:

```
SELECT Name, RegNr, Model  
FROM owner  
JOIN car ON Id = Owner_Id;
```

	Name	RegNr	Model
▶	Per Persen	EL22222	Tesla Model S
	Kari Nordmann	KH33333	Ferrari Spider

# LEFT JOIN

- So what if we also want to retrieve people who do not own a car?
  - Then we do a **LEFT JOIN**: (with owner as the "left" table)

```
SELECT Id, Name, RegNr, Model
FROM owner
LEFT JOIN car ON Id = Owner_Id;
```

	Id	Name	RegNr	Model
▶	1	Per Persen	EL22222	Tesla Model S
	2	Ola Olsen	NULL	NULL
	3	Kari Nordmann	KH33333	Ferrari Spider

- **LEFT JOIN** takes all rows from the **first ("left") table** (owner in this example), and then includes relevant data from the right table (car in this example) for rows that have a connection between the tables.
  - With **LEFT JOIN** we get Ola Olsen, even though he doesn't own a car.

# RIGHT JOIN

- What if we rather would have all cars, even those without an owner?
  - Then we use **RIGHT JOIN**: (still with owner as left table)

```
SELECT Id, Name, RegNr, Model
FROM owner
RIGHT JOIN car ON Id = Owner_Id;
```

	Id	Name	RegNr	Model
▶	1	Per Persen	EL22222	Tesla Model S
	3	Kari Nordmann	KH33333	Ferrari Spider
	NULL	NULL	ZX22222	Volvo V70

- **RIGHT JOIN** takes all rows from the **last ("right") table** (car in this example), and then includes relevant data from the left table (owner in this example) for rows that have a connection between the tables.
  - With **RIGHT JOIN** we get ZX22222 Volvo V70, even though it doesn't have an owner.

# World db, query on multiple tables

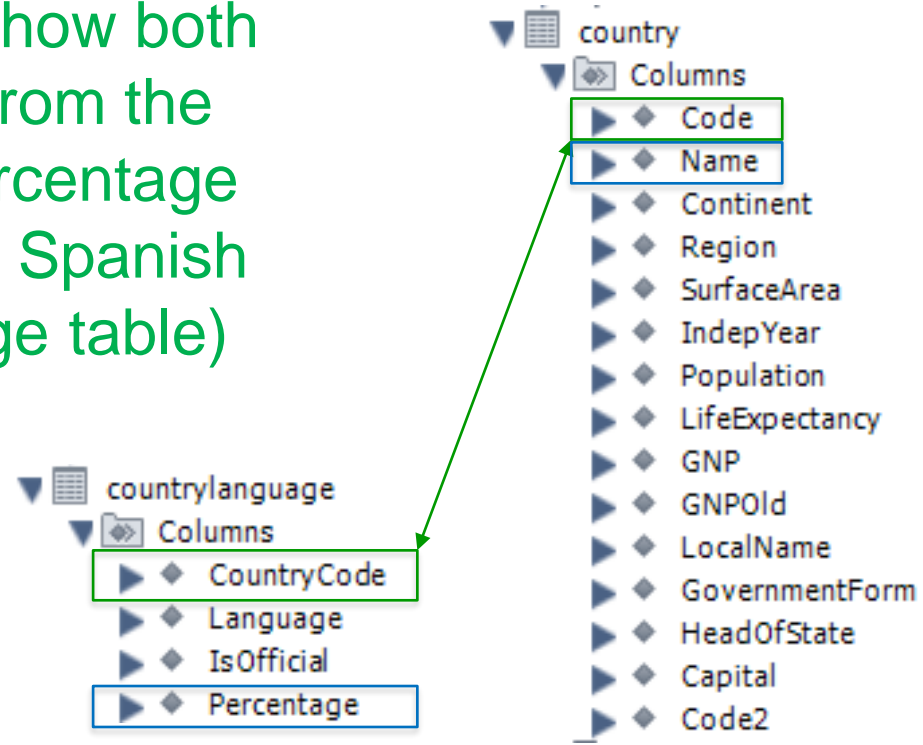
- Let's do an example from the `world` database:
  - We want to find which countries have Spanish as their official language.
- We have the knowledge to do this for the `countrylanguage` table by itself.
  - However, we won't get names of countries, only their countrycodes:

```
SELECT CountryCode, Percentage
FROM countrylanguage
WHERE Language = 'spanish' AND IsOfficial = 'T'
ORDER BY CountryCode;
```



# World db, query on multiple tables – cont.


- What to do if we want to show both the name of the country (from the Country table) and the percentage of the country that speaks Spanish (from the CountryLanguage table) at the same time?
- We use JOIN! :-)



# World db, query on multiple tables – cont.

- Example:

```
SELECT Name, Percentage
FROM country
JOIN countrylanguage ON Code = CountryCode
WHERE Language = 'spanish' AND IsOfficial = 'T'
ORDER BY Name;
```



- Note: We do *not* need to specify table names in the SELECT or ON sections, as long as the columns have unique names across the involved tables.
  - But **if there are equal column names**, we need to **add table name and a dot** in front of column names, like this: `SELECT country.Name, city.Name`

# World db, another JOIN example


- Example:
  - We want to retrieve the countries that begin with 'An%', and any cities in them.

```
SELECT country.Name, city.Name
FROM Country
JOIN City ON Code = CountryCode
WHERE country.Name like 'An%'
ORDER BY country.Name;
```

- But: *Does this retrieve all countries starting with 'An%'?*
  - Time will show!
  - (Well, actually, next slide will show. O\_o )

# World db, another JOIN example – cont.

- Antarctica is missing because it does not contain any cities!
  - To include Antarctica, we need to do a LEFT JOIN:

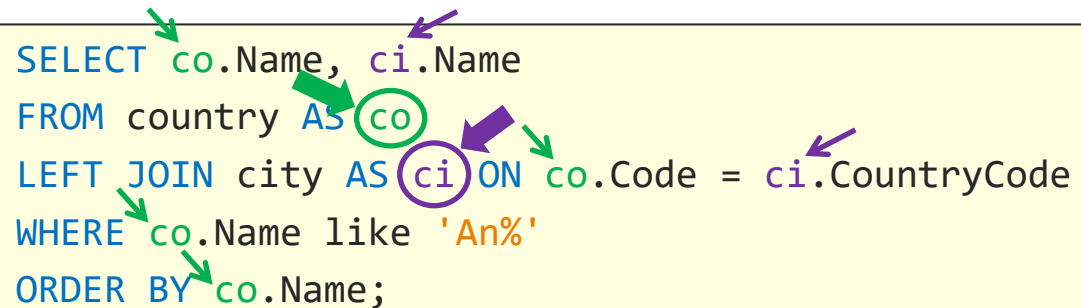


```
SELECT country.Name, city.Name
FROM Country
LEFT JOIN City ON Code = CountryCode
WHERE country.Name like 'An%'
ORDER BY country.Name;
```

- As we know: **LEFT** JOIN includes rows from the **first / left** table (Country) that are not in the last / right table (City).

# Some tips regarding JOIN

- We need to use table prefixes to separate identical column names over multiple tables.
  - *Note: We're allowed to use aliases here as well:*



```
SELECT co.Name, ci.Name
FROM country AS co
LEFT JOIN city AS ci ON co.Code = ci.CountryCode
WHERE co.Name like 'An%'
ORDER BY co.Name;
```

The diagram illustrates the use of table prefixes and aliases in a SQL query. It shows a query that joins a 'country' table (aliased as 'co') with a 'city' table (aliased as 'ci'). The query selects the 'Name' column from both tables, joins them on 'co.Code = ci.CountryCode', filters by 'co.Name like 'An%', and orders by 'co.Name'. Annotations include green arrows pointing to 'co' in the SELECT, FROM, WHERE, and ORDER BY clauses, and purple arrows pointing to 'ci' in the SELECT and JOIN clauses. The aliases 'co' and 'ci' are circled in green and purple respectively.

- A table prefix (ex: co.Code) is not required if the column is only defined in one of the tables. But you might get better performance with it.

# JOIN types and naming

- These JOINS have a longer name as well:

We have written		Could also have written
JOIN	→	INNER JOIN
LEFT JOIN	→	LEFT OUTER JOIN
RIGHT JOIN	→	RIGHT OUTER JOIN

- *Note:* There is a couple other types of JOINS as well:
  - FULL OUTER JOIN ← *Not supported by MySQL.*
  - NATURAL JOIN ← *Can omit ON when PK & FK have same name.*

# Another way to combine tables

- It's actually possible to combine tables through the comma separated setup that gave a cartesian product at the start of this lesson.
  - If we do that syntax, then we need to include the PK  $\leftrightarrow$  FK connection (the ON-part) in a where clause instead:

```
SELECT City.Name, City.Population, Country.Name  
FROM Country  
JOIN City ON Code = CountryCode  
WHERE Continent = 'Europe';
```

```
SELECT City.Name, City.Population, Country.Name  
FROM Country, City  
WHERE Code = CountryCode  
AND Continent = 'Europe';
```



# Today's exercises & next lesson

- Now: 2 hours of exercises.
- Exercises are found on Canvas. Short summary:
  - Queries (SELECT operations) using **JOIN**.
- Main contents for the next lesson:
  - Walkthrough of some of this week's JOIN tasks, using VIEWS and using subqueries (has *nothing* to do with U-boats O\_o ).



The

