SQL – Exploring data

SQL, Standard Query Language, is a language constructed to query databases. In my mind, working with SQL really appeals to the journalist in me: As a journalist I ask questions – and with this language I can ask questions also to data.

It's good to know that there are dialects of the language – a query in PostgreSQL can sometimes be different than MySQL. Or a function in SQL Server might not exist in SQLite. I use Google for that. There are tons of pages with examples of queries and I usually find the right wording – or syntax there.

Another thing to understand is datatypes. If you are familiar with Excel you know that there are no limitations to what kind of data you put in a column. That column can contain numbers, text and dates – and usually Excel seems to handle each sort of data separately. SQL databases are more finicky – you have to determine whether a column – or field – will contain numbers or text or dates. And that means that if you have a column defined as text, you will not be able to sum the numbers put in that column. And if you have a column defined as number, it is not possible to put any text in it.

Some examples of data types:

Integer and real:	Integer can't have any decimals, if they have they are real
Char var and text:	All kinds of text
Date and time:	Dates and times, so that we can calculate time difference.

One other thing that differs from Excel is that we work in a database – and a database can contain one or more tables – and the tables can interact, be cross-matched with each other. So, we have to start by creating our database:

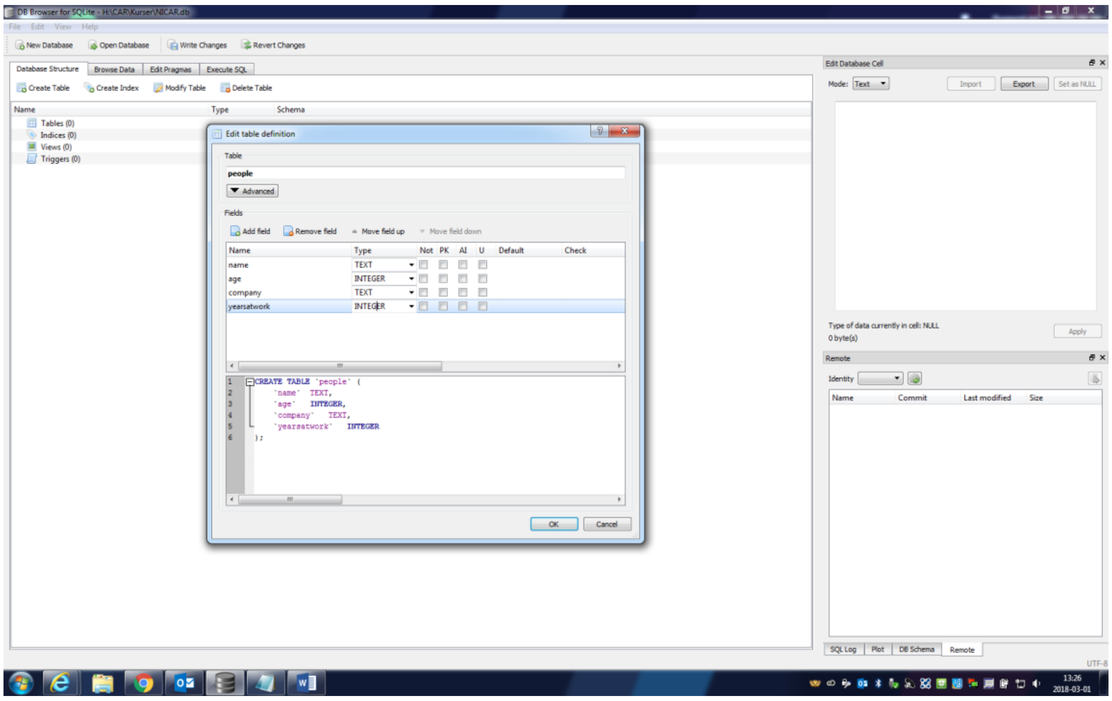
PEOPLE

Click on New database and fill in the name of the database – we can call it DATACLASS.

After that we can either upload a table – or create a table from scatch. The program is asking us to create our first table. Fill in the empty box, either by writing the following – or by using the tool to click on Add field for each field:

```
CREATE TABLE people (
   name text,
   age integer,
   company text,
   yearsatwork integer
);
```

Click OK once you are done.



Edit table definition

Table: people

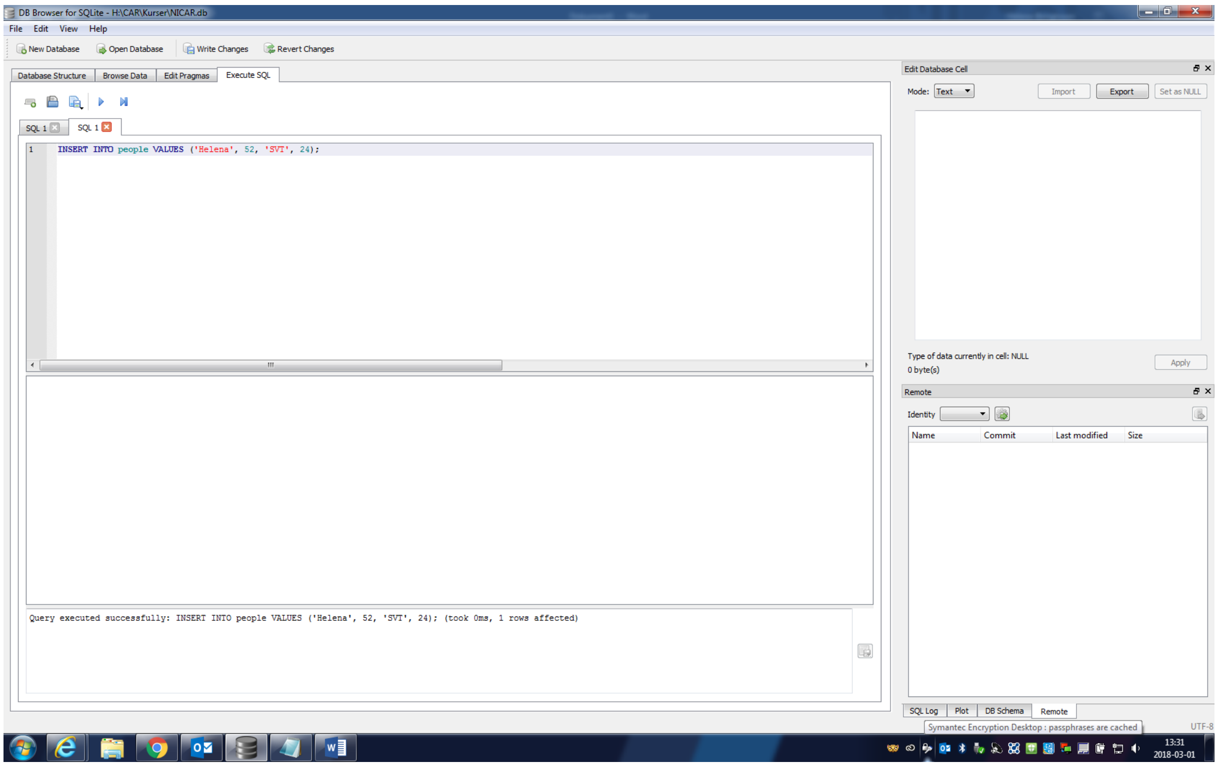
Name	Type	Not	PK	AI	U	Default	Check
name	TEXT	☐	☐	☐	☐		
age	INTEGER	☐	☐	☐	☐		
company	TEXT	☐	☐	☐	☐		
yearsatwork	INTEGER	☐	☐	☐	☐		

```
CREATE TABLE `people` (
	`name`	TEXT,
	`age`	INTEGER,
	`company`	TEXT,
	`yearsatwork`	INTEGER
);
```

Now let's fill the database. Click on the Execute SQL tab and enter this in the box:

INSERT INTO people VALUES ('Helena', 52, 'SVT', 24);

Click on the play button above the box.



```
INSERT INTO people VALUES ('Helena', 52, 'SVT', 24);
```

```
Query executed successfully: INSERT INTO people VALUES ('Helena', 52, 'SVT', 24); (took 0ms, 1 rows affected)
```

You can then use the sentence and just change the values to enter more people into your table. Just remember to press "play" between each person.

Once we have some people in our database we can start to ask questions to our data. Delete the INSERT-command and enter instead:

SELECT * FROM people;

And press play – this is your data. Note that the program tells you how many rows you have in the bottom of the screen.

You can also select just some of the columns – very useful if you have a wide table with a lot of columns – or fields.

SELECT name, age FROM people;

We can also sort our data. Delete the semi colon and add "ORDER BY name;" to your query:

SELECT name, age FROM people ORDER by name;

How would you change the sentence to sort by age instead? What if we want the ages sorted descending, the oldest first?

SELECT * FROM people ORDER by age DESC;

What if we only wanted people from Chicago Tribune? Then we'll need to add a condition – a WHERE statement. The only tricky thing is that you have to remember that SELECT comes first, then WHERE and then ORDER BY. So, let's change the query one more time:

SELECT * FROM people WHERE company='Chicago Tribune';

Press play to see the rows – note again that the number of rows are listed at the end of the page. You can of course use WHERE on numbers as well:

SELECT * FROM people WHERE age > 40;

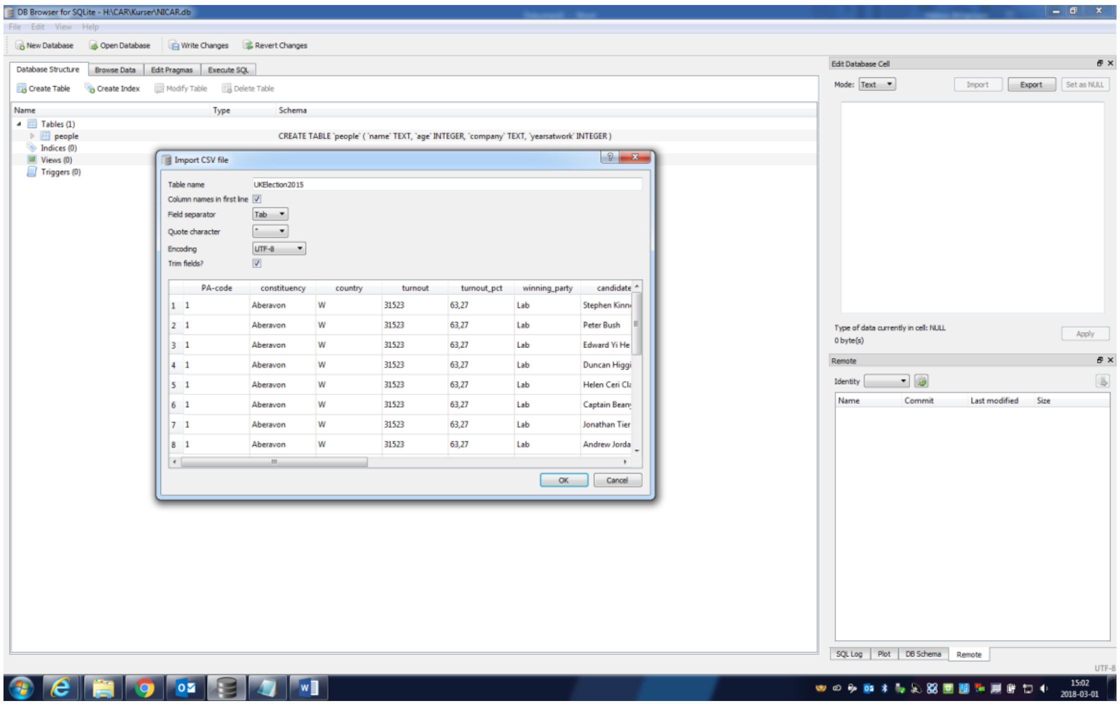
Note that you use quotation marks for text, but no quotes for numbers. So, what if you don't know exactly how people spelled their workplace? Then you have to change the syntax a little:

SELECT * FROM people WHERE company like '%Tribune%';

Note that we change the equal sign to a "like" and then included a percent sign before and after the text.

UK ELECTION

Now, let's look at some real data. I've prepared a file with the result of the 2015 UK Election. The UK election system works so that you have 650 constituencies and in each constituency, you can have only one person on the ballot for each party. So, the file contains all the candidates in each constituency and there is a column called 2015_winner that determines whether the candidate won the seat or not. Let's add this table to our DATACLASS database by choosing File in the menu and then Import and Table from CSV file. Go find the **UKElection2015**.csv file and click OK to add the file. We have to change the separator to Tab and click to make sure that the first row contains column names.



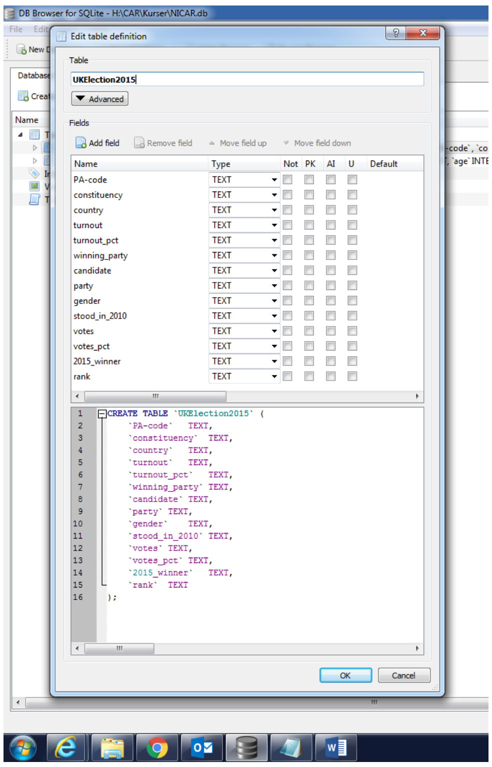
CREATE TABLE `people` (`name` TEXT, `age` INTEGER, `company` TEXT, `yearsatwork` INTEGER)

Import CSV file

Table name	UKElection2015
Column names in first line	✓
Field separator	Tab
Quote character	"
Encoding	UTF-8
Trim fields?	✓

	PA-code	constituency	country	turnout	turnout_pct	winning_party	candidate
1	1	Aberavon	W	31523	63,27	Lab	Stephen Kinn
2	1	Aberavon	W	31523	63,27	Lab	Peter Bush
3	1	Aberavon	W	31523	63,27	Lab	Edward Yi He
4	1	Aberavon	W	31523	63,27	Lab	Duncan Higgi
5	1	Aberavon	W	31523	63,27	Lab	Helen Ceri Cl
6	1	Aberavon	W	31523	63,27	Lab	Captain Bean
7	1	Aberavon	W	31523	63,27	Lab	Jonathan Tier
8	1	Aberavon	W	31523	63,27	Lab	Andrew Jorda

Click OK to import the table. But, we also have to make sure that the configuration is correct. Click on UKElection2015 and then click Modify table:

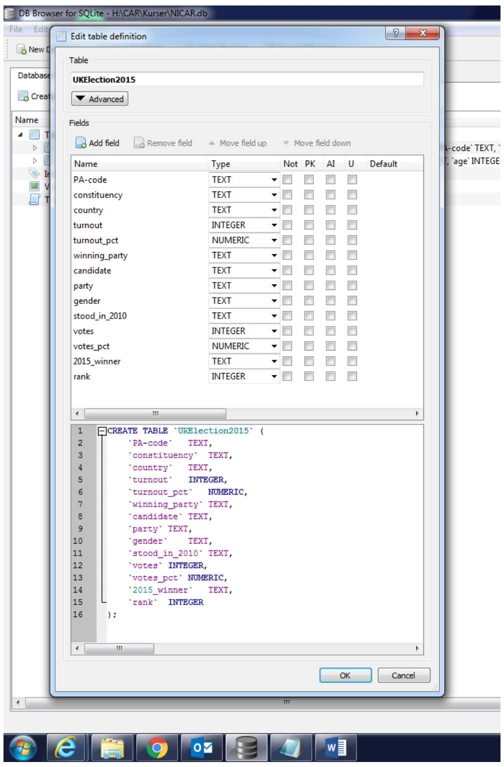


Edit table definition

Table: UKElection2015

Name	Type	Not	PK	AI	U	Default
PA-code	TEXT	☐	☐	☐	☐	
constituency	TEXT	☐	☐	☐	☐	
country	TEXT	☐	☐	☐	☐	
turnout	TEXT	☐	☐	☐	☐	
turnout_pct	TEXT	☐	☐	☐	☐	
winning_party	TEXT	☐	☐	☐	☐	
candidate	TEXT	☐	☐	☐	☐	
party	TEXT	☐	☐	☐	☐	
gender	TEXT	☐	☐	☐	☐	
stood_in_2010	TEXT	☐	☐	☐	☐	
votes	TEXT	☐	☐	☐	☐	
votes_pct	TEXT	☐	☐	☐	☐	
2015_winner	TEXT	☐	☐	☐	☐	
rank	TEXT	☐	☐	☐	☐	

```
CREATE TABLE `UKElection2015` (
    `PA-code`   TEXT,
    `constituency`  TEXT,
    `country`   TEXT,
    `turnout`   TEXT,
    `turnout_pct`   TEXT,
    `winning_party` TEXT,
    `candidate` TEXT,
    `party` TEXT,
    `gender`    TEXT,
    `stood_in_2010` TEXT,
    `votes` TEXT,
    `votes_pct` TEXT,
    `2015_winner`   TEXT,
    `rank`  TEXT
);
```



Edit table definition

Table: UKElection2015

Name	Type	Not	PK	AI	U	Default
PA-code	TEXT	☐	☐	☐	☐	
constituency	TEXT	☐	☐	☐	☐	
country	TEXT	☐	☐	☐	☐	
turnout	INTEGER	☐	☐	☐	☐	
turnout_pct	NUMERIC	☐	☐	☐	☐	
winning_party	TEXT	☐	☐	☐	☐	
candidate	TEXT	☐	☐	☐	☐	
party	TEXT	☐	☐	☐	☐	
gender	TEXT	☐	☐	☐	☐	
stood_in_2010	TEXT	☐	☐	☐	☐	
votes	INTEGER	☐	☐	☐	☐	
votes_pct	NUMERIC	☐	☐	☐	☐	
2015_winner	TEXT	☐	☐	☐	☐	
rank	INTEGER	☐	☐	☐	☐	

```
CREATE TABLE `UKElection2015` (
    `PA-code`   TEXT,
    `constituency`  TEXT,
    `country`   TEXT,
    `turnout`   INTEGER,
    `turnout_pct`   NUMERIC,
    `winning_party` TEXT,
    `candidate` TEXT,
    `party` TEXT,
    `gender`    TEXT,
    `stood_in_2010` TEXT,
    `votes` INTEGER,
    `votes_pct` NUMERIC,
    `2015_winner`   TEXT,
    `rank`  INTEGER
);
```

Note that we change the datatypes for some of the fields: turnout, votes and rank are INTEGER and turnout_pct and votes_pct are numeric.

Click ok and now we can start to ask questions to the data. Let's start by checking that the data is correct:

SELECT * FROM UKElection2015 WHERE 2015_winner='Y';

How many rows do you get? So how many consistencies did the Labour Party win? Their abbreviation in Lab. But, in order to find this out, you realize that you need two conditions, just doing WHERE winning_party='Lab' gives you a strange result. Why?

So, we need to combine the two conditions:

SELECT * FROM UKElection2015 WHERE 2015_winner='Y' AND winning_party='Lab';

And we can do the same for the conservative party:

SELECT * FROM UKElection2015 WHERE 2015_winner='Y' AND winning_party='Con';

But, what if we want to check two parties at once – instead of combining with AND, we want to use a OR-statement:

SELECT * FROM UKElection2015 WHERE party='Lab' OR party='Soc Lab';

And then we can continue our analysis. So this is some of the questions that come to mind:

- Where were the turnout highest? Lowest?
- Who won with the biggest share of the votes?
- How many women won their seats? How many men?
- What other parties are there, besides Con, Lab, SNP and LD?

How would you construct this last question – there are several ways. One way is to add conditions one after another, but that can be a little clumsy and hard to understand. Instead there is a very useful way to express called IN which can be used like this:

SELECT * FROM UKElection2015 WHERE rank=1 AND winning_party NOT IN ('Lab','Con','LD','SNP') ORDER BY party;

In order to count the number of women who won their seat, it's of course easy to just do a SELECT-statement and see how many rows you get:

SELECT * FROM UKElection2015 WHERE rank=1 AND gender='F';

But there is also another way – to use a function, just as you would in Excel:

SELECT count(candidate) FROM UKElection2015 WHERE rank=1 AND gender='F';

We can even use calculations:

SELECT count(candidate)/650.00 FROM UKElection2015 WHERE rank=1 AND gender='M';

Let's say we want to see which parties that won the election – we can then use a SELECT DISTINCT, where you only get each name once:

```
SELECT DISTINCT party FROM UKElection2015 WHERE rank=1;
```

So, what if we want to know how many constituencies each party won? And we don’t want to run a SELECT-statement 12 times – so how can we do this? This is when you have to group your candidates by party. Compare this to sitting at your kitchen table organizing documents in different piles. You want to take each winner, and put all the Conservatives in one pile, all the Labour candidates in another pile, and then you count each pile.

```
SELECT party, count(candidate) FROM UKElection2015 WHERE rank=1 GROUP by party;
```

Every column that you include in your SELECT has to be added to the GROUP BY, unless you use the column for calculations. You can of course combine this with ORDER BY – and the order is important:

```
SELECT party, count(candidate) FROM UKElection2015 WHERE rank=1 GROUP by party ORDER BY
count(candidate);
```

In SQL you have to but everything in just the correct order. Otherwise it will not work:

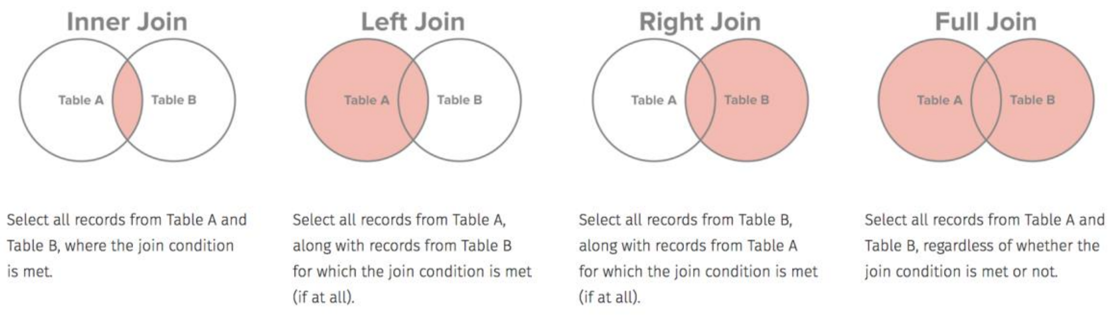
```
SELECT
FROM
WHERE
GROUP BY
ORDER BY
```

On top of this you can add AND, OR, LIKE and NOT LIKE to your WHERE-clause, and you can use functions in your SELECT-clause, like count(), but there is also sum() and avg():

```
SELECT party, count(party), avg(votes_pct)
FROM UKElection2015
WHERE rank = 1
GROUP BY party
ORDER BY count(party) desc ;
```

We can use an alias to make it easier to understand the result – and we can also limit the number of rows we get in our result:

```
SELECT party, count(party), avg(votes) as average
FROM UKElection2015
WHERE rank = 1
GROUP BY party
ORDER BY avg(votes) desc
LIMIT 5;
```



Select all records from Table A and Table B, where the join condition is met.

Select all records from Table A, along with records from Table B for which the join condition is met (if at all).

Select all records from Table B, along with records from Table A for which the join condition is met (if at all).

Select all records from Table A and Table B, regardless of whether the join condition is met or not.

Let's add in a second table to our database – the contributions to the candidates. Remember how we did that? Go to the File-menu and choose Import and then Table from CSV file. Just as the last time, we have to fix the variables once we have uploaded the table.

Make sure that VALUE has the datatype REAL, otherwise we can't calculate the amounts that the candidates received while campaigning. Let's explore the donations data:

How many candidates have donations?

```
SELECT CleanName, count(CleanName) as count
FROM UK15_donations
GROUP BY CleanName
ORDER BY  count desc;
```

Per candidate, how many donations, sum and average

```
SELECT CleanName, count(Value), sum(Value), avg(VALUE)
FROM UK15_donations
GROUP BY CleanName
ORDER BY count(Value) desc;
```

As you can see some candidates have several donations – the relationship between the results table and the donations table is a one-to-many relation. One row in the results table relates to many rows in the donations table. This is something we need to consider when we join on the name. Also, we can add an alias to a table just the way that we had an alias for a variable earlier. This saves time when writing your queries:

```
SELECT  CleanName, candidate
FROM UK15_donations a
JOIN UKElection2015 b
ON a.CleanName = b.candidate;
```

So, what we create when we do a join is really one large table that contains the columns (variables) we selected from both tables. And what happens if there are several rows in one table that relates to several rows in the other table? As you can see candidates name from the result table is repeated for all the rows in the donations table. That means that we can do anything to this large table – as we could to the single table we worked on before. Using functions like SUM, for instance:

```
SELECT  CleanName, candidate, sum(Value) as total
FROM UK15_donations a
JOIN UKElection2015 b
ON a.CleanName = b.candidate
GROUP BY CleanName
ORDER BY total desc;
```

What if we want to know who got donations and who was left out? Then we need to use a left join instead – which means that we get all data from the left table – and only the data that matches from the right table. So, there will be rows where there is no data in the columns from the right table.

```
SELECT  CleanName, candidate, sum(Value) as total
FROM UK15_donations a
LEFT JOIN  UKElection2015 b
ON a.CleanName = b.candidate
GROUP BY CleanName
ORDER BY total desc;
```

Look through the list and see how there are names on one side not on the other. Compare this to the result you got earlier. Can we add in a condition to see how many winners that were not getting any donations?

Lastly, we can also do a group by for party to see the total amount raised by each party, and by constituency:

```
SELECT  party, sum(Value) as total
FROM UK15_donations a
JOIN  UKElection2015 b
ON a.CleanName = b.candidate
GROUP BY party
ORDER BY total desc;

SELECT  constituency, sum(Value) as total
FROM UK15_donations a
JOIN  UKElection2015 b
ON a.CleanName = b.candidate
GROUP BY constituency
ORDER BY total desc;
```

I hope that it's become clear that SQL really is a way to interview data – and just as when you are interviewing people you have to ask the right questions and listen thoroughly to the answer in order to ask the correct follow up question.

A tip is to keep a log book of your queries – I copy and paste every statement, including the CREATE TABLE and INSERT-text into a text document and add in comments and other things so that I can easily go back and re-create what I once did. Also, great to have when your editor wants to know how you reached a certain conclusion.

The data: http://www.helenabengtsson.se/material/

Commonly used SQL commands: https://www.codecademy.com/articles/sql-commands