SQL Queries

SQL (Structured Query Language) can be used to create and modify databases, but here we will only focus on requesting data from databases. This is the first step in the data science workflow, and it is an important skill for data analysts and scientists.

As an example, we will work with the following simple database, that contains a table called admissions:

patientID	**age**	**admission_date**	**discharge_date**
10021	78	01/02/2020	10/02/2020
10024	50	01/02/2020	03/02/2020
10027	63	06/02/2020	13/02/2020

SELECT

To select data from a table with SQL, we use the SELECT keyword. For example, to select the age column from the admissions table:

```
SELECT age
FROM admissions
```

This will give us the following data:

age
78
50
63

The keywords SELECT and FROM are not case-sensitive, so using “select” would also work. However, it is good practice to keep SQL keywords upper case, so you can distinguish them from the rest of the query.

To select all the data from a table, use a star:

```
SELECT *
FROM admissions
```

This will give us the entire admissions table.

If you want to select multiple columns, add their names after the SELECT keyword separated by commas:

SELECT admission_date, discharge_date
FROM admissions

This will give us:

admission_date	discharge_date
01/02/2020	10/02/2020
01/02/2020	03/02/2020
06/02/2020	13/02/2020

To select only unique values of a column, use DISTINCT:

SELECT DISTINCT admission_date
FROM admissions

admission_date
01/02/2020
06/02/2020

Filtering

We've seen how to select entire columns of data, but what if we only want some of the rows? For this we use the WHERE keyword. For example, if we want the admission and discharge dates only for patients older than 70:

SELECT admission_date, discharge_date
FROM admissions
WHERE age > 70

This will give us only one row, as only the first patient is older than 70:

admission_date	discharge_date
01/02/2020	10/02/2020

Make sure you add the WHERE statement after the FROM statement in your SQL query.

The available operators you can use in WHERE statements are:

=	equal
<>	not equal
<	less than
>	more than
<=	less or equal
>=	more or equal

You can include multiple conditions using the AND keyword. For example, if we wanted the patient IDs for patients between 60 and 70:

```
SELECT patientID
FROM admissions
WHERE age > 60 AND age < 70
```

patientID
10027

To include multiple conditions where some but not all the conditions need to be met, use OR. If we wanted the patient IDs who were discharged either on 10/02/2020 or 13/02/2020:

```
SELECT patientID
FROM admissions
WHERE discharge_date = ‘10/02/2020’ OR discharge_date = ‘13/02/2020’
```

This gives us the first and third patient:

patientID
10021
10027

If you are looking for values in a range, you can also use the BETWEEN keyword. Let’s redo the query asking for patient IDs between 60 and 70 years old:

```
SELECT patientID
FROM admissions
WHERE age BETWEEN 60 AND 70
```

patientID
10027

Aggregate Functions

These functions do a calculation on a group of values:

- Count: total number of rows that match the specified criteria

What is the total number of admissions where the patient’s age was over 50?

```
SELECT count(*)
FROM admissions
WHERE age > 50
```

2

You can use a column name of the table instead of *, e.g. **count(age)**. This will not count missing (NULL) values in that column.

- Sum: sum of all the values in a particular column
- Max/Min/Avg: maximum, minimum or average value in a particular column

What is the oldest age in the admissions dataset?

SELECT max(age)
FROM admissions

78

Group by

This is a clause that groups observations by identical values in one or more columns. It is often used in combination with aggregate functions to query information of similar observations.

Here we want to count the number of admissions per admission date:

SELECT admission_date, count(*)
FROM admissions
GROUP BY admission_date

admission_date	
01/02/2020	2
06/02/2020	1

Multiple Tables

Data is often spread across multiple tables, and we need to combine them to get the information we want. Let's add a second table to our previous one:

admissions

patientID	age	admission_date	discharge_date
10021	78	01/02/2020	10/02/2020
10024	50	01/02/2020	03/02/2020
10027	63	06/02/2020	13/02/2020

addresses

patientID	postcode
10021	AB46 1JQ
10024	AB58 2XC
10025	AB23 5RS
10027	AB58 2XC

The second table includes the postcode of the patients, and the same patientID.

patientID is the **primary key** in both tables: it is a unique value (no two rows in a single table share the same patientID), and cannot be NULL. It is used to join the tables:

- INNER JOIN: This is the default join (so can also be written simply as JOIN) and it returns results from more than one table by joining them together based on common column values specified using an ON clause.

Here we want the postcodes of all patients for whom we have demographic information:

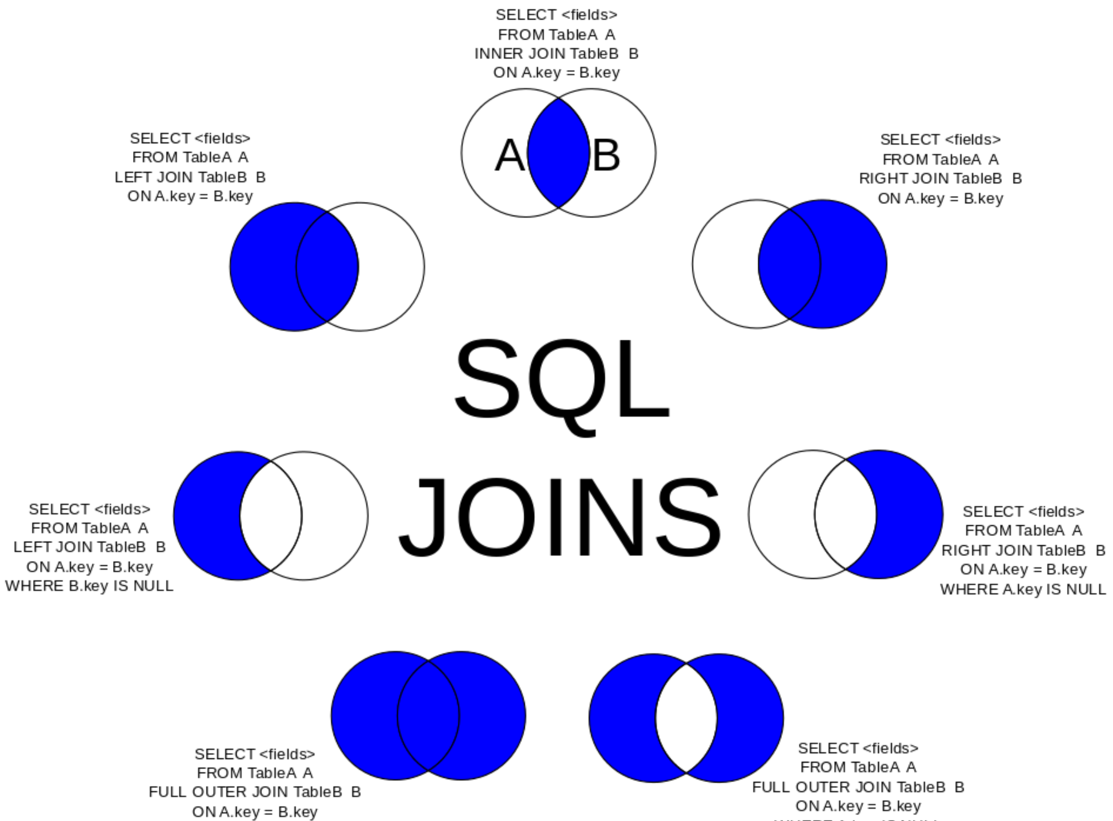
SELECT *
FROM addresses
JOIN admissions
ON addresses.patientID = admissions.patientID

patientID	postcode
10021	AB46 1JQ
10024	AB58 2XC
10027	AB58 2XC

Patient with patientID = 10025 does not exist in the demographics table, so they are not included in our results.

There are three other types of join:

- LEFT JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table


SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key
A
B
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
SQL
JOINS
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL