CORRELAID

GOOD CAUSES. BETTER EFFECTS.

SQL Introduction

How experienced are you with SQL?

Please use the “**annotate**” function of Zoom / Teams to indicate your level of expertise with SQL

No experience

Expert

Contents

SQL lets you access and manipulate databases

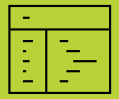
What is SQL?

- SQL stands for Structured Query Language
- SQL lets you **access** and **manipulate databases**

What can it do?

- With SQL you can **execute queries** and **retrieve data** from a database, **insert, update** or **delete records** in a database, **create** new **databases**, …

How does it work?



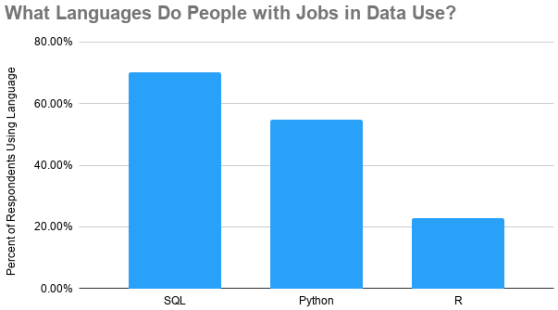
- SQL is basically a relational database management system (**RDBMS**) where **data** is **stored in tables** that can be accessed and manipulated with queries

SOURCE: https://www.w3schools.com/sql/sql_intro.asp

It is one of the most popular languages for work with data

Why you should learn SQL

SQL is still one of the **most used languages** to work with **databases** and used by most large companies (Uber, Netflix, Airbnb, …)



Data Source: StackOverflow 2020 Survey

SOURCE: https://www.dataquest.io/blog/why-sql-is-the-most-important-language-to-learn/

Contents

Let's have a look at some SQL basics

First, **open** the **w3 schools SQL editor** using the link below or google w3 SQL editor
w3schools.com/sql/trysql.asp?filename=trysql_op_in

Your Database:

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Restore Database

The example database contains 8 tables with 3 – 518 rows

SQL Statement:

```
SELECT * FROM Customers;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 91

CustomerID	CustomerName	ContactName	Address
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312

You can write your statements into the editor

Results will be shown below after you "Run SQL"

In SQL most actions are performed with so called statements

SQL Statements

An SQL **statement** is text that the database recognizes as a valid command

The **structure** of SQL statements **vary**. The **number of lines** used **does not matter**. A statement can be written all on one line, or split up across multiple lines if it makes it easier to read

Example

```
SELECT *
FROM Customers
WHERE XYZ;
```

SELECT and FROM are **clauses**. In SQL clauses perform specific actions and are written in capital letters[1]

This part refers to the result columns (separated by “,”, e.g. CustomerID, CustomerName). “*” means to return all columns in the table. You can also use AS to rename columns (e.g. CustomerName AS Name)

Refers to the table we want to access

WHERE clauses can be used to filter the output rows (e.g. WHERE country = “Germany”)

SQL statements always end with a “;”

Now try it out yourself: select the customer's name and address of all customers. What is the name of the 3rd customer? What categories are in the categories-table?

You can use clauses like “distinct”, “where”, etc. to tailor your data extraction

Important clauses	What they do	Example
SELECT DISTINCT	Return only distinct values	SELECT DISTINCT City FROM Customers;
WHERE	Filter rows	SELECT * FROM Customers WHERE Country='Mexico';
AND, OR, NOT	Combine multiple WHERE conditions	SELECT * FROM Customers WHERE Country=‘Mexico’ OR Country=‘Germany’;
ORDER BY	Order results ASC (default) or DESC	SELECT * FROM Customers ORDER BY City DESC;
LIKE	Filter rows based on patterns. “%” represents 0-x characters, “_” just one	SELECT * FROM Orders WHERE OrderID LIKE "102%" OR OrderID LIKE "1035_";
BETWEEN	Selects values in a range of numbers, text or dates	SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;
IN	Selects rows based on matches with a list of following values	SELECT * FROM Products WHERE CategoryID IN (1,2,6);

When using multiple clauses in a statement you need to place them in the correct order

Syntax order of SELECT clauses[1]

SELECT

FROM

WHERE

GROUP BY (you will learn about this later)

ORDER BY

1: selected clauses

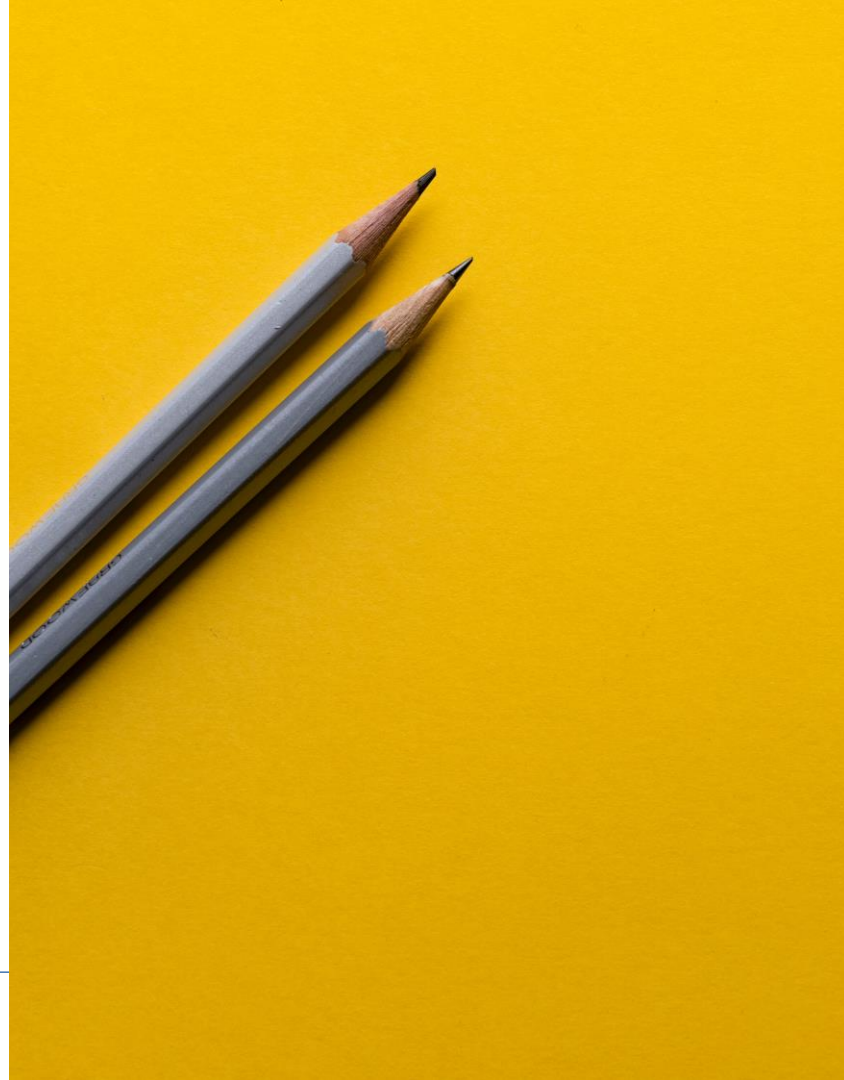
Now try it out

Exercise

1. Try out the different clauses and examples and see what they do

2. Return the customer name, contact name and address of all customers in Germany where the postal code starts with a 1 in descending order by the customer name

3. Return a list of employee ids (in ascending order) that are present in the order table and are associated with orders in 1997



Let's see if you got it right

Exercise	Solution
1. Try out the different clauses and examples and see what they do	-
2. Return the customer name, contact name and address of all customers in Germany where the postal code starts with a 1 in descending order by the customer name	SELECT CustomerName, ContactName, Address FROM Customers WHERE Country = "Germany" AND PostalCode LIKE "1%" ORDER BY CustomerName DESC;
3. Return a list of employee ids (in ascending order) that are present in the order table and are associated with orders in 1997	SELECT DISTINCT EmployeeID FROM Orders WHERE OrderDate BETWEEN "1997-01-01" AND "1997-12-31" ORDER BY EmployeeID;

Aggregation comes in handy to summarize results

Important clauses	What they do	Example
MIN / MAX	Return MIN / MAX Value	SELECT MAX(Quantity) FROM OrderDetails;
COUNT, AVG, SUM	Returns the number of rows / avg / sum of the column	SELECT SUM(Quantity) FROM OrderDetails;
LIMIT	Limits the output rows to a specified number	SELECT * FROM Customers LIMIT 5;
GROUP BY	Groups together rows, can be used together with COUNT, AVG, MAX, …	SELECT CategoryID, MAX(Price) AS MaxPrice FROM Products GROUP BY CategoryID;

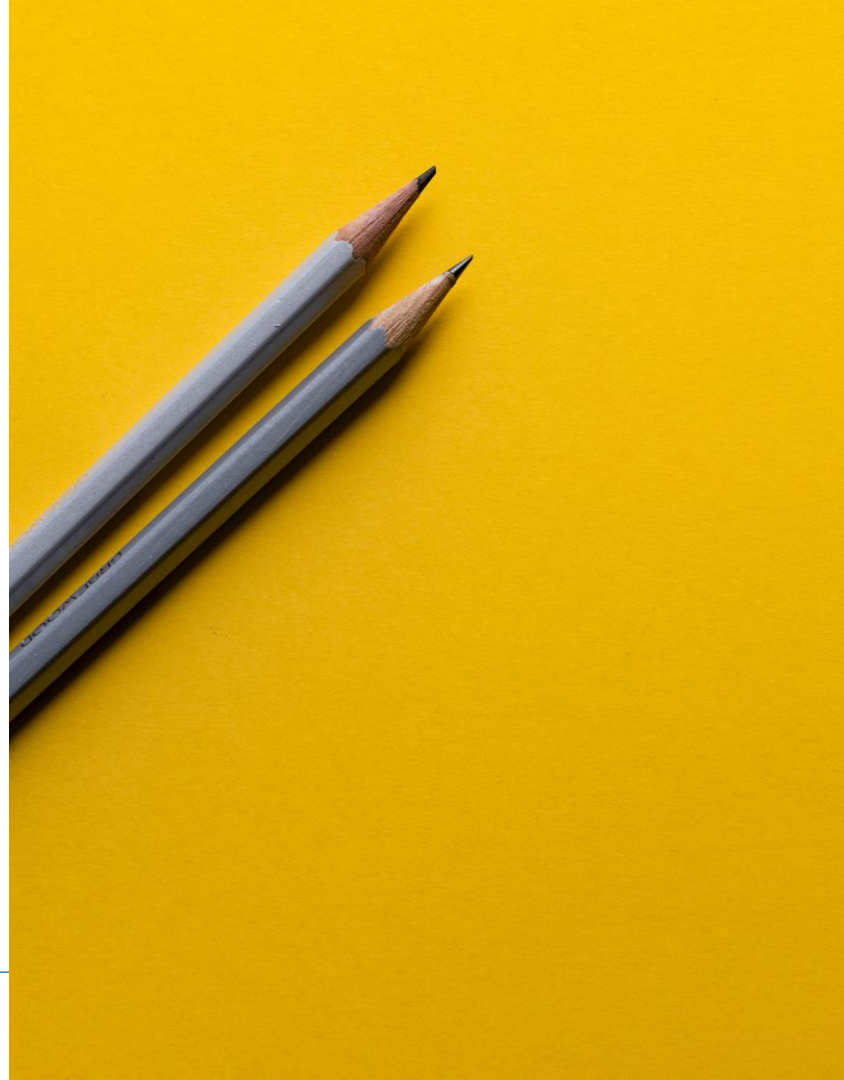
Now try it out

Exercise

1. Try out the different clauses and examples and see what they do

2. Find out the maximum and minimum price of products with the category IDs 2 and 4

3. Return the top 5 products by their ordered quantity (Hint: use the table OrderDetails)



Let's see if you got it right

Exercise	Solution
1. Try out the different clauses and examples and see what they do	-
2. Find out the maximum and minimum price of products with the category IDs 2 and 4	SELECT MAX(Price) AS MaxPrice, MIN(Price) AS MinPrice FROM Products WHERE CategoryID IN (2,4);
3. Return the top 5 products by their ordered quantity (Hint: use the table OrderDetails)	SELECT ProductID, SUM(Quantity) AS OrderedQuantity FROM OrderDetails GROUP BY ProductID ORDER BY OrderedQuantity DESC LIMIT 5;

Using multiple tables

Cross-Join

A	B
1	ab
2	cd

x

C	D
3	ef
4	gh

=

A	B	C	D
1	ab	3	Ef
1	ab	4	Gh
2	cd	3	Ef
2	cd	4	gh

SQL Example

```
SELECT *
FROM Customers, Orders;
```

Now try it out yourself: for example, SELECT * FROM Shippers, Categories;

Using multiple tables for a query

Example

```
SELECT Customers.CustomerID, Orders.OrderID
FROM Customers, Orders
WHERE Customers.CustomerID=
Orders.CustomerID;
```

Use the dot-operator to specify from which table the columns are from

Specify the tables you want to access

Using multiple tables for a query

For lazy programmers

```
SELECT DISTINCT C.CustomerID
FROM Customers AS C, Orders AS O,
OrderDetails AS OD
WHERE C.CustomerID=O.CustomerID AND
O.OrderID=OD.OrderID AND OD.Quantity>80;
```

Specify the tables you want to access

AS can also be used to rename tables

Use the dot-operator to specify from which table the columns are from

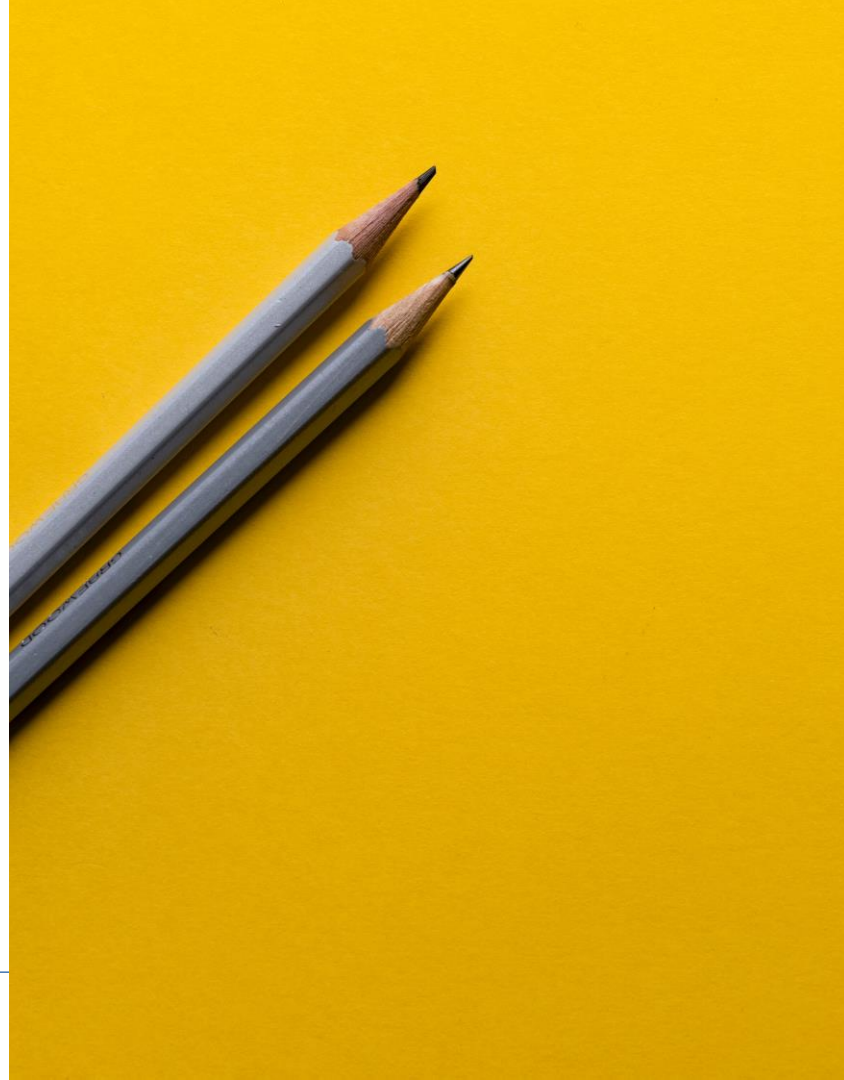
Now try it out

Exercise

1. Try out the different clauses and examples and see what they do

2. Find the names of all employees who worked on an order from “Ernst Handel”

3. Return the average number of ordered products by category

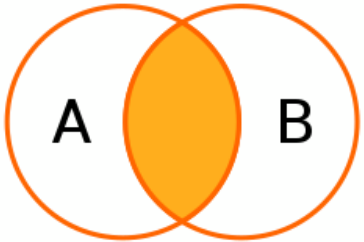


Let's see if you got it right

Exercise	Solution
1. Try out the different clauses and examples and see what they do	-
2. Find the names of all employees who worked on an order from “Ernst Handel”	SELECT DISTINCT E.FirstName FROM Employees AS E, Orders AS O, Customers AS C WHERE E.EmployeeID=O.EmployeeID AND O.CustomerID=C.CustomerID AND C.CustomerName="Ernst Handel";
3. Return the average number of ordered quantity by category	SELECT C.CategoryName, avg(OD.Quantity) AS AvgQuantity FROM OrderDetails AS OD, Categories AS C, Products AS P WHERE OD.ProductID=P.ProductID AND P.CategoryID=C.CategoryID GROUP BY C.CategoryName;

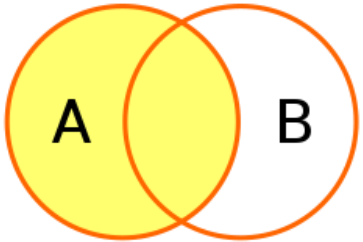
In SQL there are basically 4 main types of joins you can use to combine different tables

Inner join



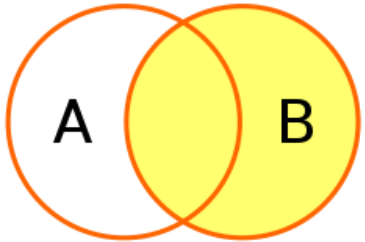
Matching values in both tables

Left (outer) join



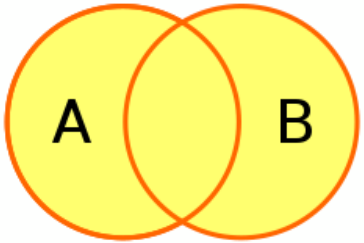
All records from the left table, and the matched records from the right table

Right (outer) join



All records from the right table, and the matched records from the left table

Full (outer) join



All records when there is a match in either left or right table

SOURCE: Wikipedia

The dot-operator can be used to refer to columns in individual tables

The Syntax of a join

```
SELECT Customers.CustomerName,
Orders.OrderID, Orders.OrderDate
FROM Orders INNER JOIN Customers ON
Orders.CustomerID=Customers.CustomerID
ORDER BY CustomerName;
```

Use the dot-operator to specify the columns that should be selected

Specify the tables to join after FROM and INNER JOIN (here Orders and Customers)

The columns to join on are specified after the clause ON, again using the dot-operator

Remarks: if you do not specify any columns to join on, SQL will generate the cross product

You can also join multiple tables in one join

Example: joining multiple tables

```
SELECT Customers.CustomerName, Orders.OrderID, Orders.OrderDate, OrderDetails.ProductID,
OrderDetails.Quantity
FROM ((Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID)
LEFT JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID);
```

Now try it out yourself! What does this query return?

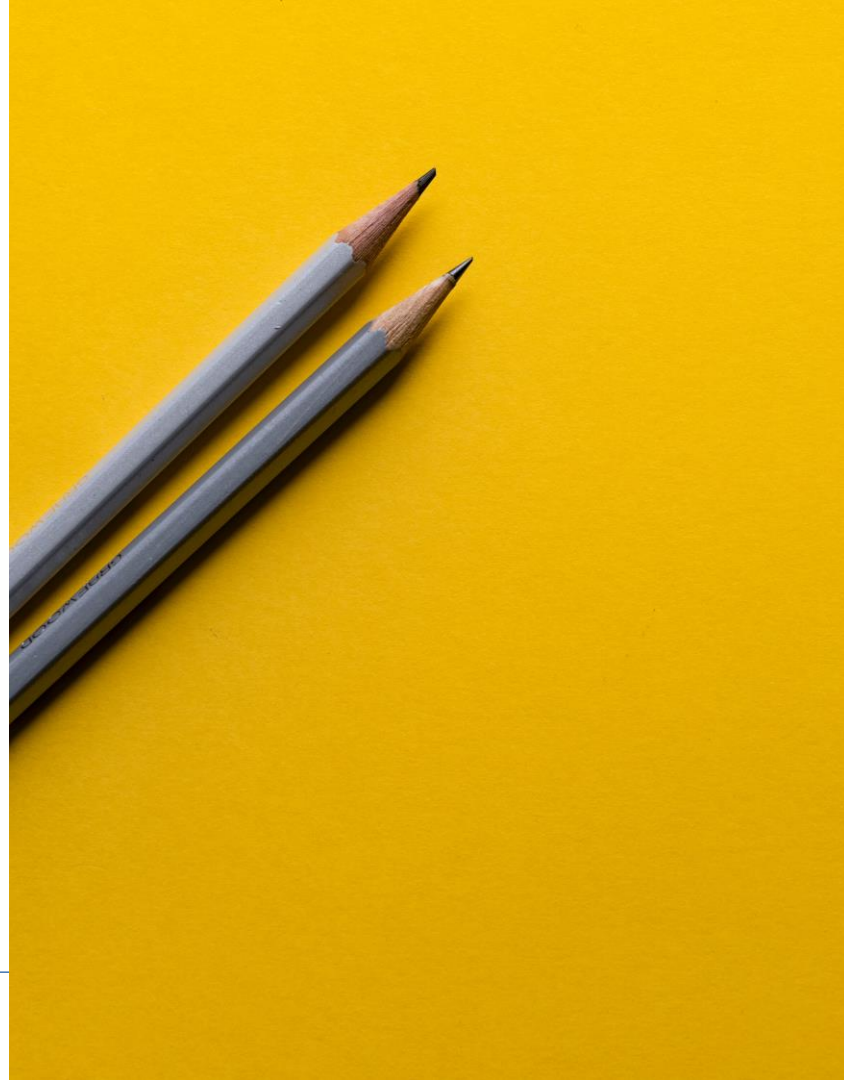
Now try it out

Exercise

1. Try out the different joins and see what they do

2. Return the top 5 customers by their ordered quantity (total)

3. How many customers have placed orders in 1997?



Let's see if you got it right (1/2)

Exercise	Solution
1. Try out the different clauses and examples and see what they do	-
2. Return the top 5 customers by their ordered quantity (total)	SELECT Customers.CustomerName, SUM(OrderDetails.Quantity) AS TotalOrderQuantity FROM ((Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID) LEFT JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID) GROUP BY Customers.CustomerName ORDER BY TotalOrderQuantity DESC LIMIT 5;

Lets see if you got it right (2/2)

Exercise

3. How many customers have placed orders in 1997?

Solution

```
SELECT COUNT(Customers.CustomerName) AS
CustomersWithOrdersIn1997
FROM Customers
INNER JOIN Orders ON Customers.CustomerID =
Orders.CustomerID
WHERE Orders.OrderDate BETWEEN "1997-01-
01" AND "1997-12-31";
```

Contents

You can create databases and tables (within the databases) easily

Creating a database

Databases are easily created using the statement CREATE Database

```
CREATE DATABASE testDatabase;
```

Creating a table

Tables are created using the CREATE TABLE clause

```
CREATE TABLE table_name (
   column1 datatype(size),
   column2 datatype(size),
   column3 datatype(size),
  ....
);
```

In SQL, many datatypes are available, e.g. CHAR, DATE, INT, FLOAT etc. In brackets you can specify the size of the data field (e.g. VARCHAR(255))

A full list can be found here: https://www.w3schools.com/sql/sql_datatypes.asp

You can use SQL to add, remove and update rows in the tables

Important clauses	What they do	Example
INSERT INTO	Inserts a new record into a table	INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
DELETE	Deletes existing records in a table	DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
UPDATE	Modifies existing records in a table	UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' WHERE CustomerID = 1;
DROP TABLE	Deletes a table within a database	DROP TABLE Shippers;
DROP DATABASE	Deletes a database	DROP DATABASE testDB;

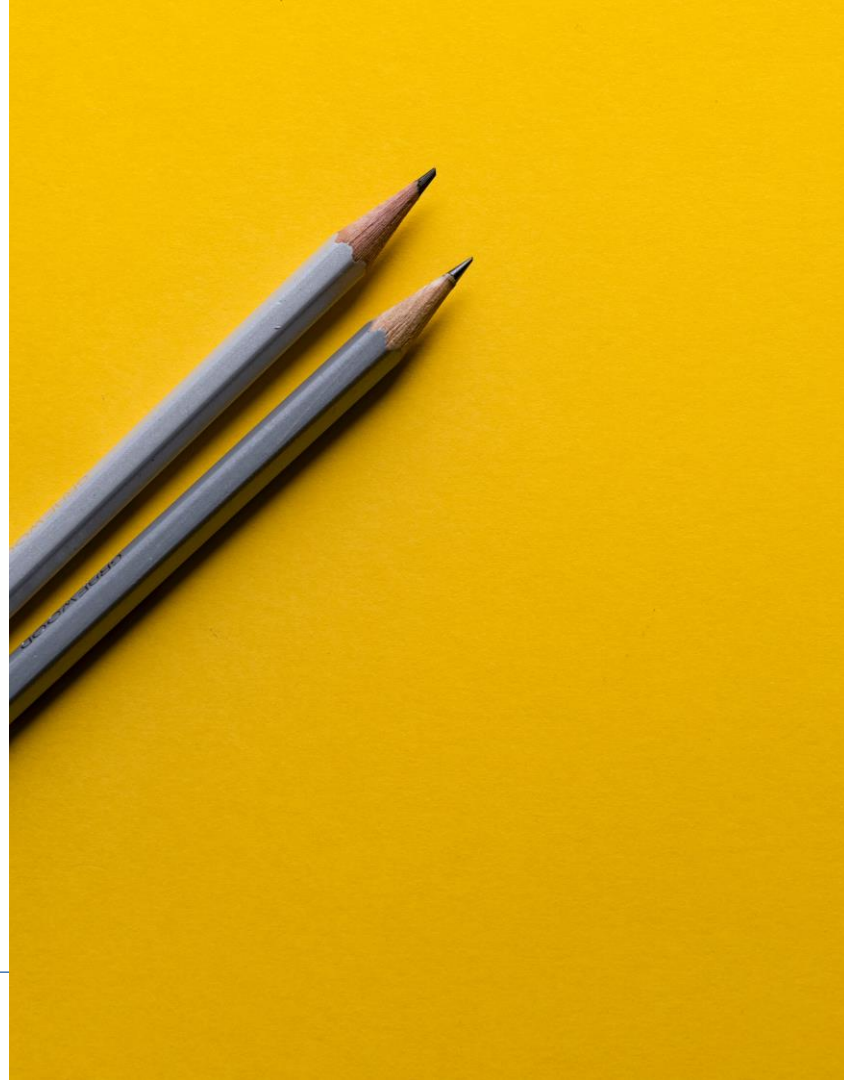
Now try it out

Exercise

1. Try out the different clauses and see what they do

2. Create a new table called “warehouses” with 3 columns (*warehouseID* (integer), *location* (text / varchar) and *storageCapacity* (integer))

3. Add 2 records to the warehouses table (feel free to chose the data yourself)



Let's see if you got it right

Exercise	Solution
1. Try out the different clauses and see what they do	-
2. Create a new table called “warehouses” with 3 columns (*warehouseID* (integer), *location* (text / varchar) and *storageCapacity* (integer))	CREATE TABLE warehouses (warehouseID int, location varchar(255), storageCapacity int);
3. Add 2 records to the warehouses table (feel free to chose the data yourself)	INSERT INTO warehouses VALUES (1, "Bielefeld", 20000); INSERT INTO warehouses VALUES (2, “Munich", 60000);

Contents

You can easily integrate SQL with Python and R

- There are multiple packages available to connect python with SQL databases
- E.g. if you use SQLite (which is probably the most used SQL system) you can use the sqlite3 package as shown below[1]

```python
import sqlite3
from sqlite3 import Error

def create_connection(path):
    connection = None
    try:
        connection = sqlite3.connect(path)
        print("Connection to SQLite DB successful")
    except Error as e:
        print(f"The error '{e}' occurred")

    return connection
```

Code example

- Similarly to Python, there are also multiple R packages you can use to work with SQL databases
- For SQLite you can use the package RSQLite[2]

```r
mammals <- DBI::dbConnect(RSQLite::SQLite(), "data_raw/portal_mammals.sqlite")
```

```r
tbl(mammals, sql("SELECT year, species_id, plot_id FROM surveys"))
```

Code example

SOURCE: https://realpython.com/python-sql-libraries/#using-python-sql-libraries-to-connect-to-a-database, https://dept.stat.lsa.umich.edu/~jerrick/courses/stat701/notes/sql.html

1. Full tutorial here: https://realpython.com/python-sql-libraries/#using-python-sql-libraries-to-connect-to-a-database
2. Full tutorial here: https://datacarpentry.org/R-ecology-lesson/05-r-and-databases.html

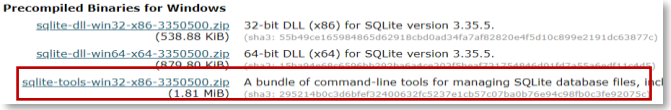
Contents

- What is SQL
- SQL Basics
- Working with SQL
  - Integration into R and Python
  - **How to build your own database**

You can use SQLite to create your SQL database

Example: Creating your fist database with windows

1. Download precompiled binaries from sqlite https://www.sqlite.org/download.html



2. Create a folder (e.g. SQLite) and unzip the files in the folder (there should be 3 files: sqldiff.exe, sqlite3.exe and sqlite3_analyzer.exe)

3. Open CMD and navigate to the new folder. To create a new database type “sqlite3 testDB.db”. This will create a new database file in the folder

4. You can now use SQL statements to interact with the database. E.g. create new tables and add records

```
sqlite> CREATE TABLE warehouses (
   ...>     warehouseID int,
   ...>     location varchar(255),
   ...>     storageCapacity int
   ...> );
sqlite>
sqlite> INSERT INTO warehouses
   ...> VALUES (1, "Bielefeld", 20000);
sqlite>
sqlite> INSERT INTO warehouses
   ...> VALUES (2, "Munich", 60000);
sqlite>
sqlite> SELECT * FROM warehouses;
1|Bielefeld|20000
2|Munich|60000
sqlite>
```

SOURCE: https://www.sqlite.org/quickstart.html

Thank you!