LU06b - SQL-DQL: Select from one table

The simplest SELECT is reading from a table, as we don't have to deal with the connection between other tables. Let's have a quick look on the different parts of an average SELECT statement. Therefore, we will use our Customer Table as practical example for demonstration purposes.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

SELECT clause

Souce: W3Schools | SELECT clause

We can use the **SELECT** in different ways, e.g

- **SELECT ***
  - * (asterix) stands here for all columns in the table
  - the entire content of the table, columns as created, sorted by the primary key in ascending order.
- **SELECT myColumn1, myColumn3, myColumn5**
  - By specifying the columns we want to retrieve, we can reduce the output according to our needs.
  - The target columns are to seperated by commas
- **SELECT myColumn3, myColumn5, myColumn1**
  - We can also change the order in which the columns are to be displayed

FROM clause

Our first SQL statement is almost complete, for we need to name the source from which we want to retrieve the data. In our case it is the table **Customer**. Our SELECT including the FROM clause would look like:

```
SELECT *
FROM Customers;
```

WHERE clause

Source: W3Schools | WHERE clause

By adding the WHERE clause to the base SQL statement, we can reduce our output. In other words, we filter our output according to the defined criterias. E.g. if we only want to select data from one particular postal code 05023, we simply add that in our WHERE clause as shown below:

```
SELECT *
FROM Customers
WHERE PostalCode ='05023';
```

ORDER BY clause

Source: W3Schools | ORDER BY clause

In modern web applications, it is common for us to be able to choose how we want to retrieve the data, as the usual sorting criteria are name, date of birth or social security number. We realize this by the adding the keywords ORDER BY to our SQL statement, followed by the keywords ASC or DESC. Our SQL statement hen would look like:

```
SELECT  PostalCode, CustomerName, ContactName
FROM Customers
ORDER BY PostalCode DESC ;
```

In this case we call up a list of customers (Customername, ContanctName, PostalCode), which is ordered descending by the zip code. Hence, the resultset looks like:

```
SELECT  PostalCode, CustomerName, ContactName
FROM Customers
ORDER BY PostalCode DESC ;
```

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

Run SQL »

Result:

Number of Records: 91

PostalCode	CustomerName	ContactName
WX3 6FW	Eastern Connection	Ann Devon
WX1 6LT	Consolidated Holdings	Elizabeth Brown
WA1 1DP	Around the Horn	Thomas Hardy
V3F 2K1	Laughing Bacchus Wine Cellars	Yoshi Tannamuri
T2F 8M4	Bottom-Dollar Marketse	Elizabeth Lincoln
SW7 1RZ	North/South	Simon Crowther
S-958 22	Berglunds snabbköp	Christina Berglund
S-844 67	Folk och fä HB	Maria Larsson

Operators

Source: W3Schools | SQL Operators

It might be necessary to get data only to one specific person or product, or we want to know which products are on stock. For such cases SQL offers, as many programming languages, a variety of Operators which help to optimize our result set. Relevant for our use are the folling operators:

Arithmetical Operators

Operator	Description	Example	Result
+	Addition	SELECT 30 + 20 + 10;	60
*	Subtracting	SELECT 30 + -10 - 40;	-2
*	Multiplication	SELECT 1 * 2 * 3 * 4;	24
/	Division	SELECT 4 % 3	1.33333...
%	Modulo division, integer rest of a division	SELECT 17 % 5 ;	2
DIV	Integer division, diggits before the coma.	SELECT 17 DIV 5 ;	3

SQL Comparison Operators

Operator	Description	Example
=	Equal to	SELECT * FROM Products WHERE Price = 18;
>	Greater than	SELECT * FROM Products WHERE Price > 30;
<	Less than	SELECT * FROM Products WHERE Price < 30;
>=	Greater than or equal to	SELECT * FROM Products WHERE Price >= 30;
< =	Less than or equal to	SELECT * FROM Products WHERE Price < = 30;

Operator	Description	Example
<>	Not equal to	SELECT * FROM Products WHERE Price <> 18;

SQL Logical Operators

Operator	Description	Example
AND	TRUE if all the conditions separated by AND is TRUE	SELECT * FROM Customers WHERE City = „London“ AND Country = „UK“;
BETWEEN	TRUE if the operand is within the range of comparisons	SELECT * FROM Products WHERE Price BETWEEN 50 AND 60;
LIKE	TRUE if the operand matches a pattern	SELECT * FROM Customers WHERE City LIKE 's%';
NOT	Displays a record if the condition(s) is NOT TRUE	SELECT * FROM Customers WHERE City NOT LIKE 's%';
OR	TRUE if any of the conditions separated by OR is TRUE	SELECT * FROM Customers WHERE City = „London“ OR Country = „UK“;

Vocabulary

English	German
clause	Abschnitt
to ascend	aufsteigen
to descend	absteigend
to retrieve	abrufen
regarding	bezüglich
according to	gemäss