SQL(Structured Query Language)

1. **What is SQL**

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

2. **What can SQL can do?**

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

3. **What is RDBMS**

RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables.

A table is a collection of related data entries and it consists of columns and rows.

4. **SQL Syantax**

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and five columns (P_Id, LastName, FirstName, Address, and City).

5. SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

The following SQL statement will select all the records in the "Persons" table:

Select * from persons

SQL Select Statements

The SELECT statement is used to select data from a database.

The result is stored in a result table, called the result-set.

SQL SELECT Syntax

```
SELECT column_name(s)
FROM table_name
```

and

```
SELECT * FROM table_name
```

An SQL SELECT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City

1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select the content of the columns named "LastName" and "FirstName" from the table above.

We use the following SELECT statement:

```
SELECT LastName,FirstName FROM Persons
```

The result-set will look like this:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

6. The SQL SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax

```
SELECT DISTINCT column_name(s)
FROM table_name
```

SELECT DISTINCT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the distinct values from the column named "City" from the table above.

We use the following SELECT statement:

```
SELECT DISTINCT City FROM Persons
```

The result-set will look like this:

City
Sandnes
Stavanger

The WHERE clause is used to filter records.

7. The WHERE Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

SQL WHERE Syntax

```
SELECT column_name(s)
FROM table_name
```

WHERE column_name operator value

WHERE Clause Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the persons living in the city "Sandnes" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City='Sandnes'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

8. The AND & OR Operators

The AND operator displays a record if both the first condition and the second condition is true.

The OR operator displays a record if either the first condition or the second condition is true.

AND Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the persons with the first name equal to "Tove" AND the last name equal to "Svendson":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

OR Operator Example

Now we want to select only the persons with the first name equal to "Tove" OR the first name equal to "Ola":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Tove'
OR FirstName='Ola'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Combining AND & OR

You can also combine AND and OR (use parenthesis to form complex expressions).

Now we want to select only the persons with the last name equal to "Svendson" AND the first name equal to "Tove" OR to "Ola":

We use the following SELECT statement:

```
SELECT * FROM Persons WHERE
LastName='Svendson'
AND (FirstName='Tove' OR FirstName='Ola')
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

9. The ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

ORDER BY Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

ORDER BY DESC Example

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName DESC
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

10. The INSERT INTO Statement

The INSERT INTO statement is used to insert a new row in a table.

SQL INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...)
```

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

SQL INSERT INTO Example

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City

1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to insert a new row in the "Persons" table.

We use the following SQL statement:

```
INSERT INTO Persons
VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger')
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

11. The UPDATE Statement

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

SQL UPDATE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

Now we want to update the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

SQL UPDATE Warning

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
```

The "Persons" table would have looked like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Nissestien 67	Sandnes
2	Svendson	Tove	Nissestien 67	Sandnes
3	Pettersen	Kari	Nissestien 67	Sandnes
4	Nilsen	Johan	Nissestien 67	Sandnes
5	Tjessem	Jakob	Nissestien 67	Sandnes

12. The DELETE Statement

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax

```
DELETE FROM table_name
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

SQL DELETE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Now we want to delete the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
DELETE FROM Persons
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

The "Persons" table will now look like this:

P_Id	**LastName**	**FirstName**	**Address**	**City**
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name
```

or

```
DELETE * FROM table_name
```

Note: Be very careful when deleting records. You cannot undo this statement!

13. The TOP Clause

The TOP clause is used to specify the number of records to return.

The TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

Note: Not all database systems support the TOP clause.

SQL Server Syntax

```
SELECT TOP number|percent column_name(s)
FROM table_name
```

SQL SELECT TOP Equivalent in MySQL and Oracle

MySQL Syntax

```
SELECT column_name(s)
FROM table_name
LIMIT number
```

Example

```
SELECT *
FROM Persons
LIMIT 5
```

Oracle Syntax

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number
```

Example

```
SELECT *
FROM Persons
WHERE ROWNUM <=5
```

SQL TOP Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select only the two first records in the table above.

We use the following SELECT statement:

```
SELECT TOP 2 * FROM Persons
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

SQL TOP PERCENT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select only 50% of the records in the table above.

We use the following SELECT statement:

```
SELECT TOP 50 PERCENT * FROM Persons
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

14. The LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

SQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

LIKE Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select the persons living in a city that starts with "s" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE 's%'
```

The "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Next, we want to select the persons living in a city that ends with an "s" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%s'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Next, we want to select the persons living in a city that contains the pattern "tav" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%tav%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
3	Pettersen	Kari	Storgt 20	Stavanger

It is also possible to select the persons living in a city that NOT contains the pattern "tav" from the "Persons" table, by using the NOT keyword.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City NOT LIKE '%tav%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

15. SQL Wildcards

SQL wildcards can substitute for one or more characters when searching for data in a database.

SQL wildcards must be used with the SQL LIKE operator.

With SQL, the following wildcards can be used:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character
[charlist]	Any single character in charlist
[^charlist]	Any single character not in charlist

or [!charlist]	

SQL Wildcard Examples

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Using the % Wildcard

Now we want to select the persons living in a city that starts with "sa" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE 'sa%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Next, we want to select the persons living in a city that contains the pattern "nes" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%nes%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Using the _ Wildcard

Now we want to select the persons with a first name that starts with any character, followed by "la" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName LIKE '_la'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

Next, we want to select the persons with a last name that starts with "S", followed by any character, followed by "end", followed by any character, followed by "on" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName LIKE 'S_end_on'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

Using the [charlist] Wildcard

Now we want to select the persons with a last name that starts with "b" or "s" or "p" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName LIKE '[bsp]%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Next, we want to select the persons with a last name that do not start with "b" or "s" or "p" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName LIKE '[!bsp]%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

16. The IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

IN Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select the persons with a last name equal to "Hansen" or "Pettersen" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName IN ('Hansen','Pettersen')
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

17. The BETWEEN Operator

The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

SQL BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

BETWEEN Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City

1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select the persons with a last name alphabetically between "Hansen" and "Pettersen" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

Note: The BETWEEN operator is treated differently in different databases!

In some databases, persons with the LastName of "Hansen" or "Pettersen" will not be listed, because the BETWEEN operator only selects fields that are between and excluding the test values.

In other databases, persons with the LastName of "Hansen" or "Pettersen" will be listed, because the BETWEEN operator selects fields that are between and including the test values.

And in other databases, persons with the LastName of "Hansen" will be listed, but "Pettersen" will not be listed (like the example above), because the BETWEEN operator selects fields between the test values, including the first test value and excluding the last test value.

Therefore: Check how your database treats the BETWEEN operator.

Example 2

To display the persons outside the range in the previous example, use NOT BETWEEN:

```
SELECT * FROM Persons
WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

18. SQL Alias

You can give a table or a column another name by using an alias. This can be a good thing to do if you have very long or complex table names or column names.

An alias name could be anything, but usually it is short.

SQL Alias Syntax for Tables

```
SELECT column_name(s)
FROM table_name
AS alias_name
```

SQL Alias Syntax for Columns

```
SELECT column_name AS alias_name
FROM table_name
```

Alias Example

Assume we have a table called "Persons" and another table called "Product_Orders". We will give the table aliases of "p" and "po" respectively.

Now we want to list all the orders that "Ola Hansen" is responsible for.

We use the following SELECT statement:

```
SELECT po.OrderID, p.LastName, p.FirstName
FROM Persons AS p,
Product_Orders AS po
WHERE p.LastName='Hansen' AND p.FirstName='Ola'
```

The same SELECT statement without aliases:

```
SELECT Product_Orders.OrderID, Persons.LastName, Persons.FirstName
FROM Persons,
Product_Orders
WHERE Persons.LastName='Hansen' AND Persons.FirstName='Ola'
```

As you'll see from the two SELECT statements above; aliases can make queries easier to both write and to read.

19. SQL JOIN

The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

Tables in a database are often related to each other with keys.

A primary key is a column (or a combination of columns) with a unique value for each row. Each primary key value must be unique within the table. The purpose is to bind data together, across tables, without repeating all of the data in every table.

Look at the "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Note that the "P_Id" column is the primary key in the "Persons" table. This means that **no** two rows can have the same P_Id. The P_Id distinguishes two persons even if they have the same name.

Next, we have the "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Note that the "O_Id" column is the primary key in the "Orders" table and that the "P_Id" column refers to the persons in the "Persons" table without using their names.

Notice that the relationship between the two tables above is the "P_Id" column.

Different SQL JOINs

Before we continue with examples, we will list the types of JOIN you can use, and the differences between them.

- **JOIN**: Return rows when there is at least one match in both tables
- **LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table
- **RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table
- **FULL JOIN**: Return rows when there is a match in one of the tables

SQL INNER JOIN Keyword

The INNER JOIN keyword return rows when there is at least one match in both tables.

SQL INNER JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

PS: INNER JOIN is the same as JOIN.

SQL INNER JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons with any orders.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
INNER JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456

Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

The INNER JOIN keyword return rows when there is at least one match in both tables. If there are rows in "Persons" that do not have matches in "Orders", those rows will NOT be listed.

SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

SQL LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

PS: In some databases LEFT JOIN is called LEFT OUTER JOIN.

SQL LEFT JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3

2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons and their orders - if any, from the tables above.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
LEFT JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	**FirstName**	**OrderNo**
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	

The LEFT JOIN keyword returns all the rows from the left table (Persons), even if there are no matches in the right table (Orders).

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all the rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

SQL RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
RIGHT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

PS: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

SQL RIGHT JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the orders with containing persons - if any, from the tables above.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
RIGHT JOIN Orders
```

```
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
		34764

The RIGHT JOIN keyword returns all the rows from the right table (Orders), even if there are no matches in the left table (Persons).

SQL FULL JOIN Keyword

The FULL JOIN keyword return rows when there is a match in one of the tables.

SQL FULL JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
FULL JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

SQL FULL JOIN Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City

1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons and their orders, and all the orders with their persons.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
FULL JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

Svendson	Tove	
		34764

The FULL JOIN keyword returns all the rows from the left table (Persons), and all the rows from the right table (Orders). If there are rows in "Persons" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Persons", those rows will be listed as well.

20. The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax

```
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2
```

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

SQL UNION ALL Syntax

```
SELECT column_name(s) FROM table_name1
UNION ALL
SELECT column_name(s) FROM table_name2
```

PS: The column names in the result-set of a UNION are always equal to the column names in the first SELECT statement in the UNION.

SQL UNION Example

Look at the following tables:

"Employees_Norway":

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

"Employees_USA":

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

Now we want to list **all the different** employees in Norway and USA.

We use the following SELECT statement:

```
SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA
```

The result-set will look like this:

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally

Kent, Clark
Scott, Stephen

Note: This command cannot be used to list all employees in Norway and USA. In the example above we have two employees with equal names, and only one of them will be listed. The UNION command selects only distinct values.

SQL UNION ALL Example

Now we want to list **all** employees in Norway and USA:

```
SELECT E_Name FROM Employees_Norway
UNION ALL
SELECT E_Name FROM Employees_USA
```

Result

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Svendson, Stephen
Scott, Stephen

21. The CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.

SQL CREATE TABLE Syntax

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete Data Types reference.

CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns: P_Id, LastName, FirstName, Address, and City.

We use the following CREATE TABLE statement:

```
CREATE TABLE Persons
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The P_Id column is of type int and will hold a number. The LastName, FirstName, Address, and City columns are of type varchar with a maximum length of 255 characters.

The empty "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City

The empty table can be filled with data with the INSERT INTO statement.

22. SQL Constraints

Constraints are used to limit the type of data that can go into a table.

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

We will focus on the following constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

SQL NOT NULL Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a database table.

The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
UNIQUE (P_Id)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
)
```

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (P_Id)
```

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
```

To DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

MySQL:

```
ALTER TABLE Persons
DROP INDEX uc_PersonID
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT uc_PersonID
```

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain unique values.

A primary key column cannot contain NULL values.

Each table should have a primary key, and each table can have only ONE primary key.

SQL PRIMARY KEY Constraint on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (P_Id)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
```

Note: In the example above there is only ONE PRIMARY KEY (pk_PersonID). However, the value of the pk_PersonID is made up of two columns (P_Id and LastName).

SQL PRIMARY KEY Constraint on ALTER TABLE

To create a PRIMARY KEY constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD PRIMARY KEY (P_Id)
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
```

Note: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

To DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT pk_PersonID
```

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents that invalid data form being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders
(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders
(
```

```
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
)
```

SQL FOREIGN KEY Constraint on ALTER TABLE

To create a FOREIGN KEY constraint on the "P_Id" column when the "Orders" table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

To DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

MySQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY fk_PerOrders
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
```

DROP CONSTRAINT fk_PerOrders

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK Constraint on CREATE TABLE

The following SQL creates a CHECK constraint on the "P_Id" column when the "Persons" table is created. The CHECK constraint specifies that the column "P_Id" must only include integers greater than 0.

My SQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CHECK (P_Id>0)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
)
```

SQL CHECK Constraint on ALTER TABLE

To create a CHECK constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (P_Id>0)
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
```

To DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT chk_Person
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

SQL DEFAULT Constraint on CREATE TABLE

The following SQL creates a DEFAULT constraint on the "City" column when the "Persons" table is created:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
)
```

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
OrderDate date DEFAULT GETDATE()
)
```

SQL DEFAULT Constraint on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'SANDNES'
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT 'SANDNES'
```

To DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

MySQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT
```