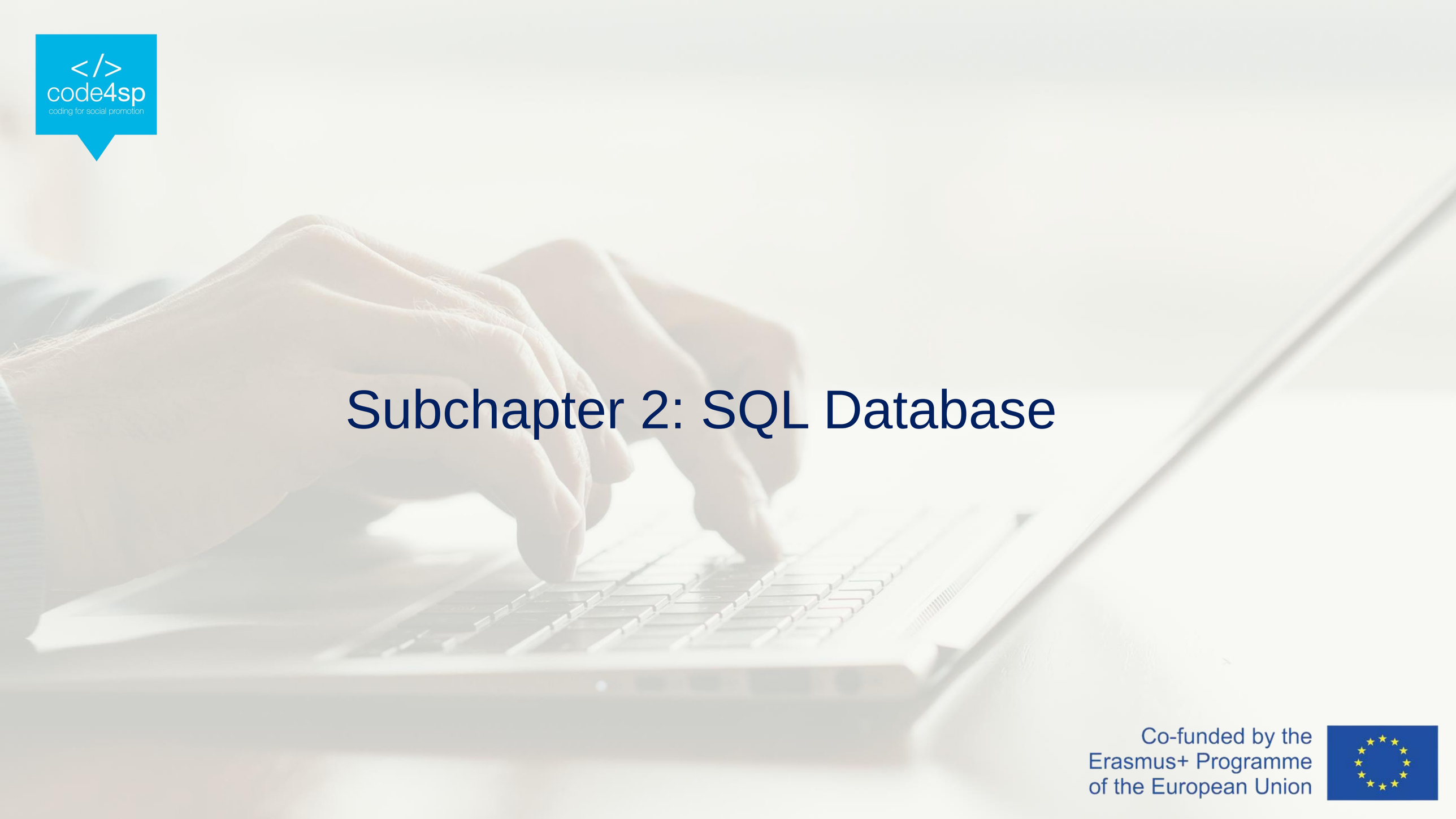
Co-funded by the
Erasmus+ Programme
of the European Union

code4sp
coding for social promotion

SQL Trainer Materials
Subchapter 2 – SQL Databases

WP3: Code4SP Training Materials

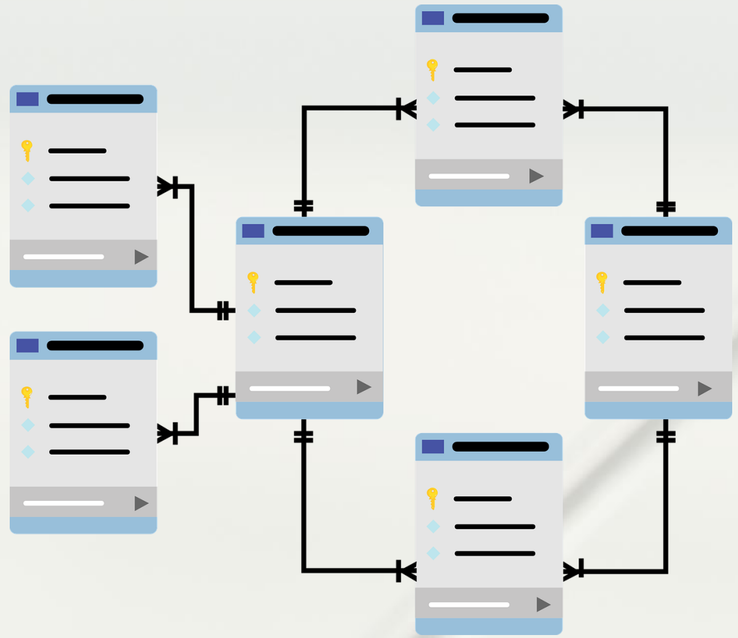
Prepared by: CITIZENS IN POWER



Subchapter 2: SQL Database

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Database Introduction



As we have mentioned in the previous subchapter that was dedicated to the basic statements used in SQL, this programming language is mainly used for relational databases.

Therefore, in this subchapter, we will learn how to create a database, modify it, and manipulate it with SQL.

Let’s start simple, and we will build into slightly more complicated statements.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Create DB

The CREATE DATABASE statement **creates a new SQL database.**

Syntax:

```
CREATE DATABASE DatabaseName;
```

Note: Always remember that the name of the database should be unique within the Relational Database Management System (RDMS) that you are using, and make sure that you have admin privileges before creating any database.

Let’s say you want to create a test database. You would use the following statement:

```
CREATE DATABASE testDB;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Drop DB

The DROP DATABASE statement **deletes an existing SQL database.**

Syntax:

DROP DATABASE DatabaseName;

*Before you delete the database, make sure that you don’t need any of the information that it contains because it completely deletes it.

Remember the database that we just created called “testDB”? Now we are going to delete it.

Example:

DROP DATABASE testDB;

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Backup DB

The BACKUP DATABASE statement does **a complete backup on an existing SQL database.**

To use this statement, you need to provide two things: the name of the database and the file path

Syntax:

```
BACKUP DATABASE DatabaseName
TO DISK = ‘filepath’
```

Example:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Backup DB

- To avoid technical problems, **it is better to back up the database to a different drive than the one the existing database is on.**

- There is also another option where you perform a differential backup based on changes that have been made since the last complete database backup. This type of backup also reduces the backup time.

Syntax:

```
BACKUP DATABASE DatabaseName
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

Example:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak'
WITH DIFFERENTIAL;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Create Table

The CREATE TABLE statement creates a new table in a database.

Syntax:

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

In this statement, you need to specify **the names of the columns** and the **type of data** that the column will contain.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Create Table

There are many data types such as integer, date or varchar. Depending on the type of data that you want to store, you choose the most suitable option. For instance, if you have a column named “Date of Birth”, then you would probably choose the Date as the data type.

Example:

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

Empty table - CREATE TABLE Example

(Source: https://www.w3schools.com/sql/sql_create_table.asp)

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Create Table

You can also create a table by using another table and choosing which columns you want in the new table. Keep in mind that the data of the existing table will fill the entries of the new table.

Syntax:

```
CREATE TABLE new_table_name AS
SELECT column1, column2,...
FROM existing_table_name
WHERE ....;
```

Example:

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

As you have learnt in the previous section:

- SELECT specifies the columns from the existing table,
- FROM specifies the name of the existing table, and
- WHERE can be used if you want a set of records that fulfil a specified condition.

Co-funded by the Erasmus+ Programme of the European Union

SQL Drop Table

Similar to the DROP DATABASE statement that we saw earlier, the DROP TABLE statement deletes an existing table in a database.

Remember that you need to be sure that you do not need any of the information contained in a table before deleting it.

Syntax:

```
DROP TABLE TableName;
```

Example:

```
DROP TABLE Persons;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Drop Table

You can also choose to delete the data contained in a table, but not the table itself.

Maybe you created a new table from an existing table that has the structure that you want, but you want to add completely new entries. That is where TRUNCATE TABLE is useful.

Syntax:

```
TRUNCATE TABLE TableName;
```

Example:

```
TRUNCATE TABLE Persons;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Alter Table

The ALTER TABLE statement can add, delete or modify columns in an existing table. Also, it can be used to add and drop constraints on an existing table.

Syntax to add a column:

```
ALTER TABLE TableName
ADD column_name datatype;
```

This is familiar to how we created a table by specifying the name of the column and the type of data to be contained in that column.

Example:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Alter Table

To delete a column in a table, as we have seen before, you use the DROP statement.

Keep in mind that some database systems do not allow for users to delete a column.

Syntax:

```
ALTER TABLE TableName
DROP COLUMN ColumnName;
```

As an example, let's delete the column that we created:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Alter Table

To **change the data type of a column**, you can use the following statements depending on the RDBMS that you are using:

- ALTER COLUMN (for SQL Server/MS Access);
- MODIFY COLUMN (for My SQL/ Oracle prior to version 10G);
- MODIFY (for Oracle version 10G and later).

Syntax:

ALTER TABLE TableName

ALTER COLUMN ColumnName datatype;

* Note that the second statement is the one that changes depending on the RDBMS that you are using from ALTER COLUMN to MODIFY COLUMN or MODIFY. The rest stays the same.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Alter Table

Example: Add a column named “DateofBirth” in the Persons table

```
ALTER TABLE Persons
ADD DateofBirth date;
```

The new column that we added to the table has the data type of date, which means that it stores data in a date format. Underneath, you can see the table with the new column added.

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

ALTER TABLE Example

(Source: https://www.w3schools.com/sql/sql_alter.asp)

Co-funded by the Erasmus+ Programme of the European Union

SQL Alter Table

However, what if you changed your mind and wanted to change the data type of the new column, then you can use the ALTER COLUMN statement.

For example, we can change the newly added column’s type from date to year:

```
ALTER TABLE Persons
ALTER COLUMN DateofBirth year;
```

The year data type holds a year in two- or four-digits format.

To delete the column that we just altered, we use the DROP COLUMN statement.

```
ALTER TABLE Persons
DROP COLUMN DateofBirth;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Constraints

SQL Constraints are used when the table is created with the statement CREATE TABLE or after the table is created with the statement ALTER TABLE.

Constraints are used to **specify a set of rules and restrictions that apply to a column or a table.** They are used to ensure the integrity, accuracy, and reliability of the data. If the constraints are applied to a table, then all columns need to adhere to these constraints.

Syntax:

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

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Constraints

The following constraints are the ones that are most commonly used:

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

We will go through each of these constraints to explain their usage and syntax with examples.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Not Null

In SQL, columns can hold null values by default. The NOT NULL constraint is used to avoid null values in columns. This is particularly important to ensure that when a new entry is added to a table all the necessary fields are filled.

As an example, let's say that we want to create a table named "Persons" and we want to ensure that the columns "ID", "LastName", and "FirstName" do not hold any null values:

```
CREATE TABLE Persons (
        ID int NOT NULL,
         LastName varchar(255) NOT NULL,
        FirstName varchar(255) NOT NULL,
        Age int
);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Not Null

If for some reason, you want to alter an already existing table to add constraints, you can use the following statement:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Unique

The UNIQUE constraint is used to ensure that all values stored in a column are unique among the rows in a table. To make this clearer, think of the variable ID. You wouldn't want two people to have the same ID, therefore you would use the constraint UNIQUE on this occasion.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

My SQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

Co-funded by the Erasmus+ Programme of the European Union

SQL Unique

If you want to **name or define a UNIQUE constraint on multiple columns**, use the following:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Unique

You can also add a UNIQUE constraint after the table has been created by using the ALTER TABLE statement that we learnt earlier.

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

If you also want to **name and define a UNIQUE constraint on multiple already existing columns**, you use the following statement:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Persons UNIQUE (ID, LastName);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Primary Key

The PRIMARY KEY constraint is used to uniquely identify each row or record in a table. Note that primary keys must contain unique values, but cannot contain null values.

A table can only have **ONE** primary key and that primary key can consist of one or multiple columns.

SQL Server/Oracle/MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

Co-funded by the Erasmus+ Programme of the European Union

SQL Primary Key

The following example allows you to **name and define a PRIMARY KEY constraint on multiple columns:**

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

*** Note that the PRIMARY KEY is still one, but the value of the primary key encompasses two columns.**

Co-funded by the Erasmus+ Programme of the European Union

SQL Primary Key

You can also create a PRIMARY KEY constraint on an existing table by using the following statement:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

To add and define a PRIMARY KEY constraint on an existing table, use the following statement:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Persons PRIMARY KEY (ID, LastName);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Primary Key

To drop a PRIMARY KEY constraint, use the following statements according to your RDBMS.

MySQL:

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

SQL Server / Oracle / MS Access:

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

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Foreign Key

The FOREING KEY **represents the columns of a table that are linked to a primary key in another table.** The table that has a foreign key is called the child table, whereas the table that has the primary key is called the referenced or parent table.

This type of constraint is used to prevent any actions that would destroy links between parent and child tables. **Consider the following two tables: What do they have in common?**

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Persons table - FOREING KEY Example
(Source: https://www.w3schools.com/sql/sql_foreignkey.asp)

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Orders table - FOREING KEY Example
(Source: https://www.w3schools.com/sql/sql_foreignkey.asp)

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Foreign Key

These two tables are linked by the column “PersonID” that is found in both tables. Now, the primary key is located in the Persons table and the foreign key is the “PersonID” in the Orders table.

The FOREIGN KEY constraint works by preventing the input of invalid data in the foreign key column because it is linked with the parent table and its values need to be identical.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Foreign Key

My SQL:

```
CREATE TABLE Orders (
        OrderID int NOT NULL,
        OrderNumber int NOT NULL,
        PersonID int,
         PRIMARY KEY (OrderID),
         FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

This statement linked the Orders table to the Persons table with the FOREIGN KEY constraint based on PersonID column.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Check

The CHECK constraint is used to specify the values allowed in a column or in certain columns of a table based on values found in other columns of the same row.

Example of CHECK constraint on CREATE TABLE: Ensure that a person is not under the age of 18, so the CHECK constraint is added to the “Age” column.

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
      CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

Co-funded by the Erasmus+ Programme of the European Union

SQL Check

If you want to name a CHECK constraint and use the constraint on multiple columns, you can use the following statement:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City= ‘Sandnes’)
);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Check

Example of CHECK constraint on ALTER TABLE

To create a constraint on an already existing table, use the following statement.

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

To name a constraint and create it on multiple columns, you can use:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_Person CHECK (Age>=18 AND City= ‘Sandnes’);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Check

Example of DROP a CHECK constraint

To eliminate a CHECK constraint, you can use the following according to the RDMBS.

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Default

The DEFAULT constraint is used to **specify a default value for a column.** If there are no other values specified, the default value will be added to all new records.

*Example of DEFAULT constraint on CREATE TABLE: A*dds a default value to the City column when the Persons table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Default

This constraint can also be used to **insert system values with functions such as GETDATE():**

```
CREATE TABLE Orders (
        ID int NOT NULL,
        OrderNumber int NOT NULL,
        OrderDate date DEFAULT GETDATE()
);
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Default

Example of DEFAULT constraint on ALTER TABLE

In this example, the column “City” is used to create a DEFAULT constraint when we are altering an already existing table.

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQL Server:

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT ‘Sandnes’;
```

Oracle:

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Sandnes';
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Default

Example of DROP a DEFAULT constraint

It is used to delete the default constraint on the already existing table

MySQL:

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

SQL Server / Oracle / MS Access:

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

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Index

The CREATE INDEX statement **creates an index on a table**. Indexes are useful when you want to retrieve data more quickly.

To CREATE INDEX on a table where duplicate values are allowed, use the following syntax:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

To CREATE UNIQUE INDEX on a table where duplicate values are not allowed, use the following syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

- ✓ Please note that tables with indexes take more time to update in comparison to tables without. Therefore, **it is suggested to only create indexes on columns that are frequently searched.**
- ✓ Keep in mind **that creating indexes varies from database to database,** so always check the syntax to create one in your database.

Co-funded by the Erasmus+ Programme of the European Union

SQL Index

Examples of CREATE INDEX

In this example, we are **creating an index on the LastName column** by specifying the name idx_lastname:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

To **create an index on a combination of columns,** use the following statement:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

If you want, you can add more columns in the parenthesis.

Co-funded by the Erasmus+ Programme of the European Union

SQL Index

Examples of DROP INDEX

If you want to delete an index, use the following statement according to your RDBMS.

MS Access:

```
DROP INDEX index_name ON table_name;
```

SQL Server:

```
DROP INDEX table_name.index_name;
```

DB2/Oracle:

```
DROP INDEX index_name;
```

MySQL:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Auto Increment

Auto-increment is used to generate unique numbers automatically when a new record is entered into a table. This is usually used on the primary key field in order to ensure that no one person has the same ID.

This feature uses different syntax in MySQL, SQL Server, Access and Oracle. Therefore, we will be going through each of these to explain how to use Auto-Increment.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Auto Increment

MySQL:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

In MySQL, **AUTO_INCREMENT** adds the auto-increment feature and by default, **the value set is 1 and it goes up by 1 each time.**

If you would like the **sequence to start from a different value**, use the following statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Auto Increment

If you enter a new record into the Persons table, you will not have to specify a value for the “PersonID” column since it will be generated automatically:

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen');
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Auto Increment

SQL Server

We are following the same example as in the previous slides of MySQL, where we use the “Personsid” column as the primary key in the Persons table:

```
CREATE TABLE Persons (
    Personid int IDENTITY(1,1) PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

In SQL Server, the auto-increment feature uses the keyword IDENTIFY to be activated. The two values in the parenthesis indicate (starting value, adding value for each new record). It will start at 1 and go up by 1 each time a new record is entered.

If you wanted to change the starting value to 10 and to add 5 each time a new record is added, you would write it like this IDENTIFY (10,5).

When entering new records, you do not need to specify the Personsid. It will be automatically generated.

Co-funded by the Erasmus+ Programme of the European Union

SQL Auto Increment

MS Access

```
CREATE TABLE Persons (
        Personid AUTOINCREMENT PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

MS Access uses AUTOINCREMENT keyword to activate the auto-increment feature. Similar to the other two, the starting value is one and it adds up by one each time a record is added.

You can specify different values such as 10 for starting value and 5 for each addition with AUTOINCREMENT(10,5).

Again, note that each time we add a new record, we do not need to specify the Personid value. It is generated automatically.

Co-funded by the Erasmus+ Programme of the European Union

SQL Auto Increment

Oracle

In Oracle, the code is a bit trickier. To create an auto-increment field, you need to create a sequence of numbers:

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10;
```

This sequence creates a sequence object named “seq_person”, sets the minimum value to start from (which is 1 in this instance), then specifies the increment by 1. The cache specifies how many sequence values should be stored in memory for faster access.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Auto Increment

Unlike the previous examples, to enter a new record into the Persons table, you need to use the nextval function. This function is used to retrieve the next value from the sequence object that we created.

```
INSERT INTO Persons (Personid,FirstName,LastName)
VALUES (seq_person.nextval,'Lars','Monsen');
```

Here, we can see that the Personid column is selected to be assigned the next number from the sequence object that we created called “seq_person”.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Dates

One of the most challenging parts when working with dates is to ensure that the format of the date you are trying to enter is the same with the format of the date column in the database.

It is important to note that data that contains only date portions will work as expected in queries. However, if there is a time portion, things get a bit more complicated.

Date Data types found in MySQL:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Co-funded by the Erasmus+ Programme of the European Union

SQL Dates

Date Data types found in SQL Server:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Keep in mind that the data types are chosen when you are creating a new table in your database.

We will use the Orders table in our example to select the records with an OrderDate of “2008-11-11”.

Example:

```
SELECT *
FROM Orders
WHERE OrderDate='2008-11-11’;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Dates

Note that two dates can be easily compared when there is no time stamp involved.

Suppose that you have the Orders table, but with a timestamp in the OrderDate column.

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

Orders table with timestamp - Dates Example

(Source: https://www.w3schools.com/sql/sql_dates.asp)

- If you attempted to use the same query as the one we used in the previous slide, you would get no result. Why? Because the query is not taking into account the time stamp.

It is recommended to not use timestamps unless you absolutely have to.

Co-funded by the Erasmus+ Programme of the European Union

SQL Views

In SQL, a view is a **virtual table of a result-set created from a specific query.** A view is useful when you want to view and present data through a combination of tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

*** Note that a view always shows up-to-date data since the database recreates the virtual table, every time users query it.**

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Views

Example to create a view that selects every product in the Products table with a price that is higher than the average price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

To query the view above:

```
SELECT * FROM [Products Above Average Price];
```

Example to query all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

To query the view:

```
SELECT * FROM [Brazil Customers];
```

Co-funded by the Erasmus+ Programme of the European Union

SQL Views

To update a view, use the CREATE OR REPLACE VIEW statement:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

To add the “City” column to the Brazil Customer view that we created earlier:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil’;
```

To delete a view, use the DROP VIEW statement:

```
DROP VIEW view_name;
```

To delete the “Brazil customers” view:

```
DROP VIEW [Brazil Customers];
```

Co-funded by the Erasmus+ Programme of the European Union

SQL Data Types

Generally, each column in a table requires a name and a data type.

An SQL developer will need to decide the type of data that will be stored inside each column when creating a table. The data type is used for SQL to understand the data that will be contained in each column and also how it will interact with the data.

*** Please keep in mind that data types might have different names in different databases. Always check the documentation even if the name is the same because other details might be different like the size.**

- For more information on different data types in different RDBMS, visit the following website: https://www.w3schools.com/sql/sql_datatypes.asp

Co-funded by the Erasmus+ Programme of the European Union

Let’s practice

You have learnt a lot of new things by now, so it is time to put what we have learnt into practice!

To do this, click here.

Co-funded by the
Erasmus+ Programme
of the European Union

code4sp
coding for social promotion

THANK YOU!

NEXT CHAPTER: SQL References

For more information, visit here

Co-funded by the
Erasmus+ Programme
of the European Union