Using Comments

Using comments in you SQL script is important to make the script easier to read and understand.
In SQL we can use 2 different kinds of comments:
- Single-line comment
- Multiple-line comment

Single-line comment

We can comment one line at the time using "**--**" before the text you want to comment out.
Syntax:

```
-- text_of_comment
```

Multiple-line comment

We can comment several line using "**/***" in the start of the comment and "***/**" in the end of the comment.
Syntax:

```
/*
text_of_comment
text_of_comment

*/
```

Variables

The ability to using variables in SQL is a powerful feature. You need to use the keyword **DECLARE** when you want to define the variables. Local variables must have the symbol "@" as a prefix. You also need to specify a data type for your variable (int, varchar(x), etc.).

Syntax for declaring variables:

```
declare @local_variable data_type
```

If you have more than one variable you want to declare:

```
declare
@myvariable1 data_type,
@myvariable2 data_type,
…
```

When you want to assign values to the variable, you must use either a **SET** or a **SELECT** statement.
Example:

```
declare @myvariable int
set @myvariable=4
```

If you want to see the value for a variable, you can e.g., use the **PRINT** command like this:

```
declare @myvariable int
set @myvariable=4
print @myvariable
```

The following will be shown in SQL Server:

Built-in Global Variables

SQL have lots of built-in variables that are very useful to use in queries and scripts.

@@IDENTITY

After an INSERT, SELECT INTO, or bulk copy statement is completed, @@IDENTITY contains the last identity value that is generated by the statement. If the statement did not affect any tables with identity columns, @@IDENTITY returns NULL. If multiple rows are inserted, generating multiple identity values, @@IDENTITY returns the last identity value generated.

Example:

Given to tables; SCHOOL and COURSE:

SCHOOL table: COURSE table:

	SchoolId	SchoolName	Description	Address	Phone	PostCode	PostAddress
1	1	TUC	NULL	NULL	NULL	NULL	NULL
2	2	NTNU	NULL	NULL	NULL	NULL	NULL

	CourseId	CourseName	SchoolId	Description
1	1	SCE2006	1	NULL
2	2	SCE1106	1	NULL
3	3	SCE4206	1	NULL
4	4	SCE4106	1	NULL

We want to insert a new School into the SCHOOL table and we want to insert 2 new Courses in the COURSE table that belong to the School we insert. To find the "SchoolId" we can use the @@IDENTITY variable:

```
declare @SchoolId int
-- Insert Data into SCHOOL table
insert into SCHOOL(SchoolName) values ('MIT')
select @SchoolId = @@IDENTITY
-- Insert Courses for the specific School above in the COURSE table
insert into COURSE(SchoolId,CourseName) values (@SchoolId, 'MIT-
101')
insert into COURSE(SchoolId,CourseName) values (@SchoolId, 'MIT-
201')
```

The result becomes:

SCHOOL table:

	SchoolId	SchoolName	Description	Address	Phone	PostCode	PostAddress
1	1	TUC	NULL	NULL	NULL	NULL	NULL
2	2	NTNU	NULL	NULL	NULL	NULL	NULL
3	16	MIT	NULL	NULL	NULL	NULL	NULL

COURSE table:

	CourseId	CourseName	SchoolId	Description
1	1	SCE2006	1	NULL
2	2	SCE1106	1	NULL
3	3	SCE4206	1	NULL
4	4	SCE4106	1	NULL
5	5	MIT-101	16	NULL
6	6	MIT-201	16	NULL

Flow Control

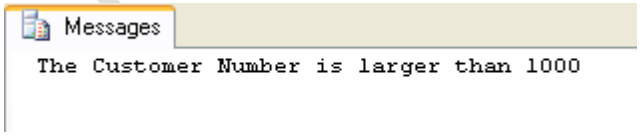
As with other programming languages you can use different kind of flow control, such as IFELSE,
WHILE, etc, which is very useful.

8.4.1 IF – ELSE

The IF-ELSE is very useful. Below we see an example:

```
declare @customerNumber int

select @customerNumber=CustomerNumber from CUSTOMER
where CustomerId=2
if @customerNumber > 1000
print 'The Customer Number is larger than 1000'
else
print 'The Customer Number is not larger than 1000'
```



BEGIN…END:

If more than one line of code is to be executed within an IF sentence you need to use **BEGIN**…**END**.
Example:

```
select @customerNumber=CustomerNumber from CUSTOMER where
CustomerId=2
if @customerNumber > 1000
begin
```

```
print 'The Customer Number is larger than 1000'
update CUSTOMER set AreaCode=46 where CustomerId=2
end
else
print 'The Customer Number is not larger than 1000'
```

WHILE

We can also use WHILE, which is known from other programming languages.

Example:

We are using the CUSTOMER table:

	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

and the following query:

```
while (select AreaCode from CUSTOMER where CustomerId=1) < 20
begin
update CUSTOMER set AreaCode = AreaCode + 1

end
select * from CUSTOMER
```

	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	20	California	11111111
2	2	1001	Jackson	Smith	53	London	22222222
3	3	1002	Johnsen	John	40	London	33333333

As you can see the code inside the WHILE loop is executed as long as “AreaCode” for CustomerId=1 is less than 20. For each iteration is the “AreaCode” for that customer incremented with 1.

CASE

The CASE statement evaluates a list of conditions and returns one of multiple possible result expressions.

Example:

We have a “GRADE” table that contains the grades for each student in different courses:

```
select GradeId, StudentId, CourseId, Grade from GRADE
```

	GradeId	StudentId	CourseId	Grade
1	1	1	1	4
2	2	2	1	5
3	3	3	3	0
4	4	4	3	3
5	5	1	3	5

In the “GRADE” table is the grades stored as numbers, but since the students get grades with
the letters A..F (A=5, B=4, C=3, D=2, E=1, F=0), we want to convert the values in the table into letters using a CASE statement:

```
select
GradeId,
StudentId,
CourseId,
case Grade
when 5 then 'A'
when 4 then 'B'
when 3 then 'C'
when 2 then 'D'
when 1 then 'E'
when 0 then 'F'
else '-'
end as Grade
from GRADE
```

	GradeId	StudentId	CourseId	Grade
1	1	1	1	B
2	2	2	1	A
3	3	3	3	F
4	4	4	3	C
5	5	1	3	A

Views

Views are virtual table for easier access to data stored in multiple tables.

Create View:

```
IF EXISTS (SELECT name
              FROM   sysobjects
              WHERE  name = 'CourseData'
              AND       type = 'V')
           DROP VIEW CourseData
GO

CREATE VIEW CourseData
AS

SELECT
SCHOOL.SchoolId,
SCHOOL.SchoolName,
COURSE.CourseId,
COURSE.CourseName,
COURSE.Description

FROM
SCHOOL
INNER JOIN COURSE ON SCHOOL.SchoolId = COURSE.SchoolId
GO
```

A View is a "virtual" table that can contain data from multiple tables

The Name of the View

Inside the View you join the different tables together using the **JOIN** operator

You can Use the View as an ordinary table in Queries :

Using the View:

```
select * from CourseData
```

	SchoolId	SchoolName	CourseId	CourseName	Description
1	1	TUC	1	Industrial IT	The best course ever
2	1	TUC	2	Control with Implementation	Control Theory
3	1	TUC	3	Systems and Control Laboratory	Practical Lav course

Syntax for creating a View:

```
CREATE VIEW <ViewName>
AS
…
```

... but it might be easier to do it in the graphical view designer that are built into SQL Management Studio.

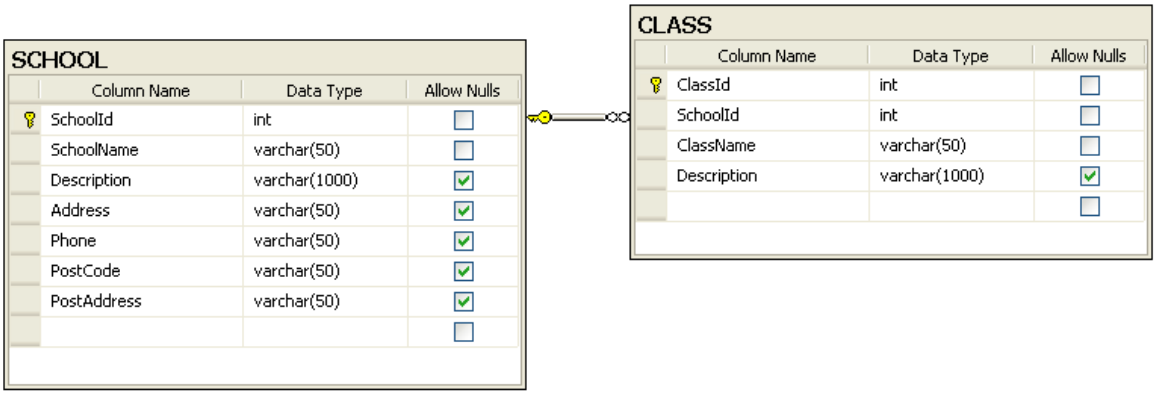
Syntax for using a View:

```
select * from <MyView> where …
```

As shown above, we use a VIEW just like we use an ordinary table.

Example:

We use the SCHOOL and CLASS tables as an example for our View. We want to create a View that lists all the existing schools and the belonging classes.



We create the VIEW using the CREATE VIEW command:

```
CREATE VIEW SchoolView
AS
SELECT
SCHOOL.SchoolName,
CLASS.ClassName
FROM
SCHOOL
INNER JOIN CLASS ON SCHOOL.SchoolId = CLASS.SchoolId
```

Note! In order to get information from more than one table, we need to link the tables together using a JOIN.