

# Stored Procedures

A Stored Procedure is a precompiled collection of SQL statements. In a stored procedure you can use if sentence, declare variables, etc.

## Create Stored Procedure:

```
IF EXISTS (SELECT name
            FROM sysobjects
            WHERE name = 'StudentGrade'
            AND type = 'P')
    DROP PROCEDURE StudentGrade
GO

CREATE PROCEDURE StudentGrade
    @Student varchar(50),
    @Course varchar(10),
    @Grade varchar(1)
AS
DECLARE
    @StudentId int,
    @CourseId int

select StudentId from STUDENT where StudentName = @Student

select CourseId from COURSE where CourseName = @Course

insert into GRADE (StudentId, CourseId, Grade)
values (@StudentId, @CourseId, @Grade)
GO
```

A Stored Procedure is like Method in C#  
- it is a piece of code with SQL commands that do a specific task – and you reuse it

Procedure Name

Input Arguments

Internal/Local Variables

Note! Each variable starts with @

SQL Code (the “body” of the Stored Procedure)

## Using the Stored Procedure:

```
execute StudentGrade 'John Wayne', 'SCE2006', 'B'
```

## Syntax for creating a Stored Procedure:

```
CREATE PROCEDURE <ProcedureName>
@<Parameter1> <datatype>
...
declare
@myVariable <datatype>
... Create your Code here
```

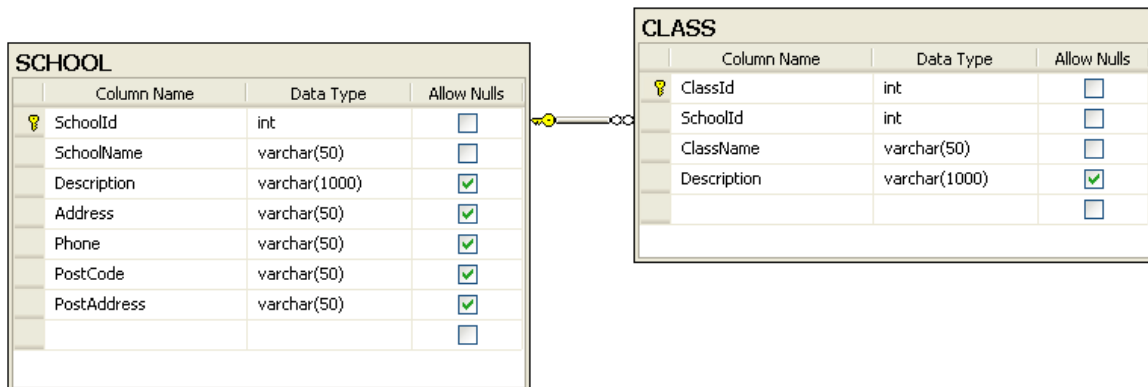
**Note!** You need to use the symbol “@” before variable names.

## Syntax for using a Stored Procedure:

```
EXECUTE <ProcedureName (...)>
```

## Example:

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



We create the Stored Procedure as follows:

```
CREATE PROCEDURE GetAllSchoolClasses
AS
select
SCHOOL.SchoolName,
CLASS.ClassName
from
SCHOOL
inner join CLASS on SCHOOL.SchoolId = CLASS.SchoolId
order by SchoolName, ClassName
```

When we have created the Stored Procedure we can run (or execute) the Stored procedure using the execute command like this:

```
execute GetAllSchoolClasses
```

	SchoolName	ClassName
1	NTNU	A1
2	NTNU	A2
3	TUC	PT1
4	TUC	PT2
5	TUC	SCE1
6	TUC	SCE2

We can also create a Store Procedure with input parameters.

**Example:**

We use the same tables in this example (SCHOOL and CLASS) but now we want to list all classes for a specific school.

The Stored Procedure becomes:

```

CREATE PROCEDURE GetSpecificSchoolClasses
@SchoolName varchar(50)
AS
select
SCHOOL.SchoolName,
CLASS.ClassName
from
SCHOOL
inner join CLASS on SCHOOL.SchoolId = CLASS.SchoolId
where SchoolName=@SchoolName
order by ClassName

```

We run (or execute) the Stored Procedure:

```
execute GetSpecificSchoolClasses 'TUC'
```

	SchoolName	ClassName
1	TUC	PT1
2	TUC	PT2
3	TUC	SCE1
4	TUC	SCE2

or:

```
execute GetSpecificSchoolClasses 'NTNU'
```

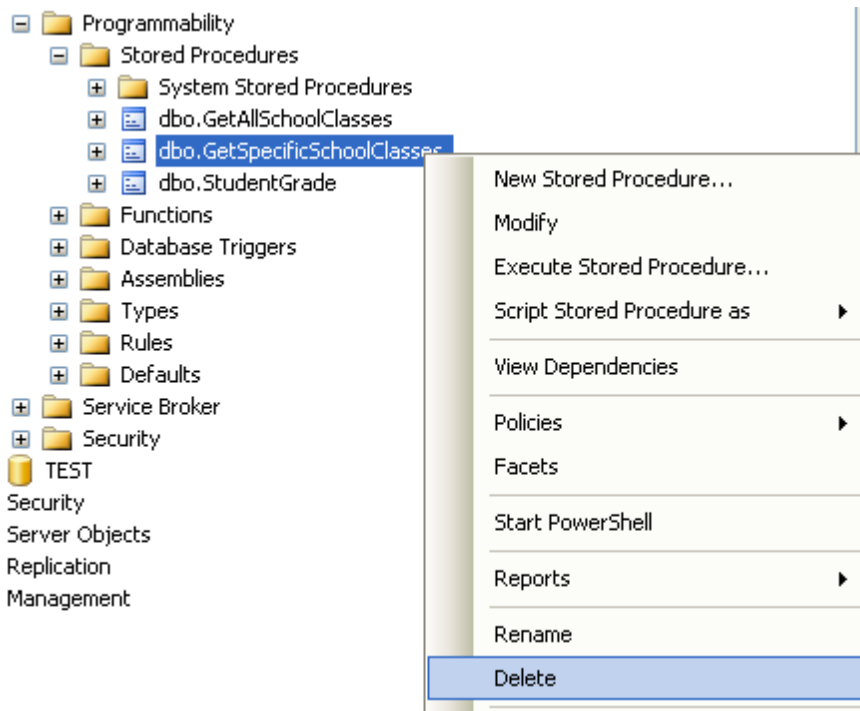
	SchoolName	ClassName
1	NTNU	A1
2	NTNU	A2

When we try to create a Stored Procedure that already exists we get the following error message:

There is already an object named 'GetSpecificSchoolClasses' in the database.

Then we first need to delete (or DROP) the old Stored Procedure before we can recreate it again.

We can do this manually in the Management Studio in SQL like this:



# Functions

With SQL and SQL Server you can use lots of built-in functions or you may create your own functions. Here we will learn to use some of the most used built-in functions and in addition we will create our own function.

## Built-in Functions

SQL has many built-in functions for performing calculations on data.

We have 2 categories of functions, namely **aggregate** functions and **scalar** functions.

Aggregate functions return a single value, calculated from values in a column, while scalar functions return a single value, based on the input value.

**Aggregate** functions - examples:

- ☐ **AVG()** - Returns the average value
- ☐ **STDEV()** - Returns the standard deviation value
- ☐ **COUNT()** - Returns the number of rows
- ☐ **MAX()** - Returns the largest value
- ☐ **MIN()** - Returns the smallest value
- ☐ **SUM()** - Returns the sum
- ☐ etc.

**Scalar** functions - examples:

- ☐ **UPPER()** - Converts a field to upper case
- ☐ **LOWER()** - Converts a field to lower case

- ☐ **LEN()** - Returns the length of a text field
- ☐ **ROUND()** - Rounds a numeric field to the number of decimals specified
- ☐ **GETDATE()** - Returns the current system date and time
- ☐ etc.

## String Functions

Here are some useful functions used to manipulate with strings in SQL Server:

### CHAR

- ☐ CHARINDEX
- ☐ REPLACE
- ☐ SUBSTRING
- ☐ LEN
- ☐ REVERSE
- ☐ LEFT
- ☐ RIGHT
- ☐ LOWER
- ☐ UPPER
- ☐ LTRIM
- ☐ RTRIM

Read more about these functions in the SQL Server Help.

## Date and Time Functions

Here are some useful Date and Time functions in SQL Server:

- ☐ DATEPART
- ☐ GETDATE
- ☐ DATEADD
- ☐ DATEDIFF
- ☐ DAY
- ☐ MONTH
- ☐ YEAR
- ☐ ISDATE

Read more about these functions in the SQL Server Help.

## Mathematics and Statistics Functions

Here are some useful functions for mathematics and statistics in SQL Server:

- ☐ COUNT
- ☐ MIN, MAX
- ☐ COS, SIN, TAN

- ☐ SQRT
- ☐ STDEV
- ☐ MEAN
- ☐ AVG

## AVG()

The AVG() function returns the average value of a numeric column.

Syntax:

```
SELECT AVG(column_name) FROM table_name
```

Example:

Given a GRADE table:

	Column Name	Data Type	Allow Nulls
▶	GradeId	int	<input type="checkbox"/>
	StudentId	int	<input type="checkbox"/>
	CourseId	int	<input type="checkbox"/>
	Grade	float	<input type="checkbox"/>
	Comment	varchar(1000)	<input checked="" type="checkbox"/>

We want to find the average grade for a specific student:

```
select AVG(Grade) as AvgGrade from GRADE where StudentId=1
```

	AvgGrade
1	4,5

## COUNT()

The COUNT() function returns the number of rows that matches a specified criteria.

The COUNT(column\_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name
```

The COUNT(\*) function returns the number of records in a table:

```
SELECT COUNT(*) FROM table_name
```

We use the CUSTOMER table as an example:

	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

```
select COUNT(*) as NumbersofCustomers from CUSTOMER
```

	NumbersofCustomers
1	3

## The GROUP BY Statement

Aggregate functions often need an added GROUP BY statement.

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

Example:

We use the CUSTOMER table as an example:

	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

If we try the following:

```
select FirstName, MAX(AreaCode) from CUSTOMER
```

We get the following error message:

Column 'CUSTOMER.FirstName' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

The solution is to use the GROUP BY:

```
select FirstName, MAX(AreaCode) from CUSTOMER
group by FirstName
```

	FirstName	(No column name)
1	John	32
2	Smith	45

## The HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

**Syntax:**

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

```
select * from GRADE
```

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

First we use the GROUP BY statement:

```
select CourseId, AVG(Grade) from GRADE
group by CourseId
```

	CourseId	(No column name)
1	1	4,5
2	3	2,666666666666667

While the following query:

```
select CourseId, AVG(Grade) from GRADE
group by CourseId
having AVG(Grade) > 3
```

	CourseId	(No column name)
1	1	4,5

**VERY IMPORTANT FOR INTERVIEW:**

Stored Procedures vs. Functions:

- ☐ Only functions can return a value (using the RETURN keyword).
- ☐ Stored procedures can use RETURN keyword but without any value being passed[1]
- ☐ Functions could be used in SELECT statements, provided they don't do any data



manipulation and also should not have any OUT or IN OUT parameters.

- ☐ Functions must return a value, but for stored procedures this is not compulsory.
- ☐ A function can have only IN parameters, while stored procedures may have OUT or IN OUT parameters.
- ☐ A function is a subprogram written to perform certain computations and return a single value.
- ☐ A stored procedure is a subprogram written to perform a set of actions, and can return multiple values using the OUT parameter or return no value at all.

For Kritiya Solutions