

<https://www.halvorsen.blog>

# Views, Stored Procedures and Triggers in SQL Server

Hans-Petter Halvorsen



# Contents

In this tutorial we will learn to create and use Views, Stored Procedures and Triggers in SQL Server.

- [Introduction](#)
- [Views](#)
- [Stored Procedures](#)
- [Triggers](#)

<https://www.halvorsen.blog>

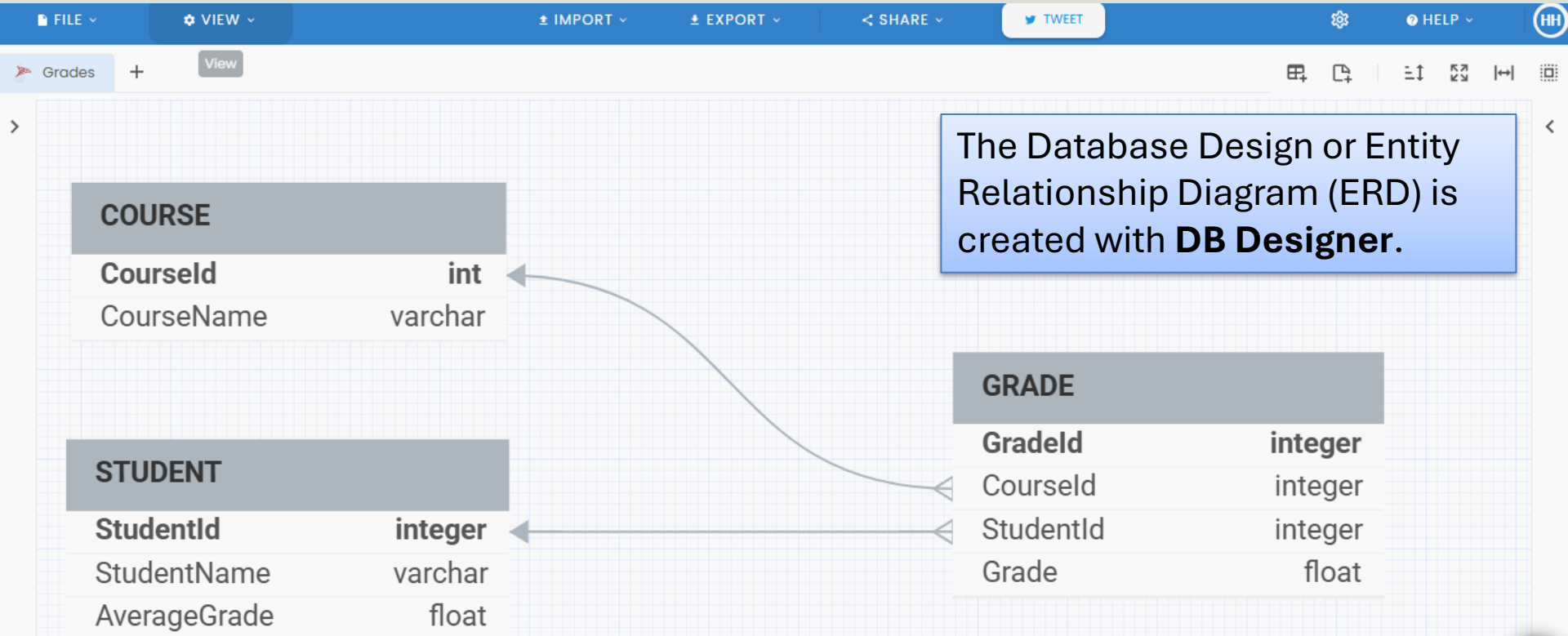
# Introduction

[Table of Contents](#)

Hans-Petter Halvorsen



# Database



This is a simplified ERD used to see how we can create and use Views, Stored Procedures and Triggers in SQL Server

# Table Script

```
CREATE TABLE [COURSE] (  
[CourseId] int IDENTITY(1,1) NOT NULL UNIQUE,  
[CourseName] nvarchar(50) NOT NULL UNIQUE,  
PRIMARY KEY ([CourseId])  
);
```

```
CREATE TABLE [STUDENT] (  
[StudentId] int IDENTITY(1,1) NOT NULL UNIQUE,  
[StudentName] nvarchar(50) NOT NULL,  
[AverageGrade] float(53),  
PRIMARY KEY ([StudentId])  
);
```

```
CREATE TABLE [GRADE] (  
[GradeId] int IDENTITY(1,1) NOT NULL UNIQUE,  
[CourseId] int NOT NULL,  
[StudentId] int NOT NULL,  
[Grade] float(1) NOT NULL,  
PRIMARY KEY ([GradeId])  
);
```

```
ALTER TABLE [GRADE] ADD CONSTRAINT [GRADE_fk1] FOREIGN KEY ([CourseId]) REFERENCES [COURSE]([CourseId]);
```

```
ALTER TABLE [GRADE] ADD CONSTRAINT [GRADE_fk2] FOREIGN KEY ([StudentId]) REFERENCES [STUDENT]([StudentId]);
```

# Create Courses and Students

Let's create some default data in our tables:

```
insert into COURSE (CourseName) values ('Mathematics')
insert into COURSE (CourseName) values ('Science')
insert into COURSE (CourseName) values ('Programming')

insert into STUDENT (StudentName) values ('Elvis Presley')
insert into STUDENT (StudentName) values ('John Wayne')
insert into STUDENT (StudentName) values ('John Statham')
```

# Courses and Students

SQLQuery1.s...ES (sa (52))\*

```
select * from COURSE
select * from STUDENT
select * from GRADE
```

150 %

Results Messages

	CourseId	CourseName
1	1	Mathematics
2	3	Programming
3	2	Science

	StudentId	StudentName	AverageGrade
1	1	Elvis Presley	NULL
2	2	John Wayne	NULL
3	3	John Statham	NULL

GradeId	CourseId	StudentId	Grade
---------	----------	-----------	-------

# Insert Grades

```
insert into GRADE (CourseId, StudentId, Grade) values (1, 1, 2.5)
```

```
insert into GRADE (CourseId, StudentId, Grade) values (2, 1, 3.5)
```

```
insert into GRADE (CourseId, StudentId, Grade) values (3, 1, 1.5)
```

Here student “Elvis Presley” (StudentId=1) gets the following grades in the different courses:

- “Mathematics” (CourseId=1) => Grade = 2.5
- “Science” (CourseId=2) => Grade = 3.5
- “Programming” (CourseId=3) => Grade = 1.5

```
Select Data...ES (sa (51))  
select * from COURSE  
select * from STUDENT  
select * from GRADE
```

150 %

Results		Messages	
	CourseId	CourseName	
1	1	Mathematics	
2	3	Programming	
3	2	Science	

	StudentId	StudentName	AverageGrade
1	1	Elvis Presley	NULL
2	2	John Wayne	NULL
3	3	John Statham	NULL

	GradeId	CourseId	StudentId	Grade
1	6	1	1	2.5
2	7	2	1	3.5
3	8	3	1	1.5



<https://www.halvorsen.blog>

# Views

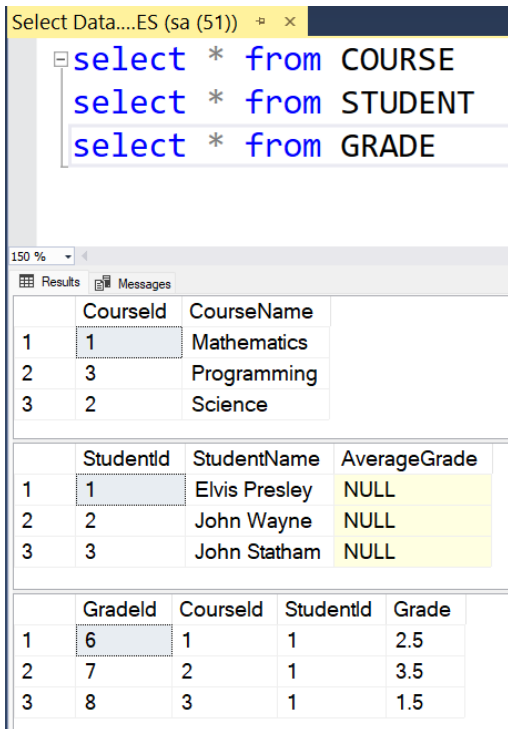
Hans-Petter Halvorsen



[Table of Contents](#)

# Problem Description

We create and use the following SQL queries to get information:



The screenshot shows a SQL IDE with three queries in the editor and their results in the Results pane.

Query 1:

```
select * from COURSE
```

Query 2:

```
select * from STUDENT
```

Query 3:

```
select * from GRADE
```

Results:

	CourseId	CourseName
1	1	Mathematics
2	3	Programming
3	2	Science

	StudentId	StudentName	AverageGrade
1	1	Elvis Presley	NULL
2	2	John Wayne	NULL
3	3	John Statham	NULL

	GradeId	CourseId	StudentId	Grade
1	6	1	1	2.5
2	7	2	1	3.5
3	8	3	1	1.5

But we want to get information like this:

	StudentName	CourseName	Grade
1	Elvis Presley	Mathematics	2.5
2	Elvis Presley	Science	3.5
3	Elvis Presley	Programming	1.5

But it is not possible because the information is stored in 3 different tables.

=> The solution is to create and use a **View**.

# Views

- A View is a “virtual” table that can contain data from multiple tables.
- Basically, a View is a SQL query that links 2 or more tables together making it possible to get data from these tables in a single query.

# View Example

```
CREATE VIEW StudentData  
AS
```

```
SELECT  
STUDENT.StudentName,  
COURSE.CourseName,  
GRADE.Grade  
FROM STUDENT  
INNER JOIN GRADE ON STUDENT.StudentId = GRADE.StudentId  
INNER JOIN COURSE ON GRADE.CourseId = COURSE.CourseId  
GO
```

In a View we typically use  
“**INNER JOIN**” to join information  
stored in different Tables.

# Create the View

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

```
CREATE VIEW StudentData
AS
```

```
SELECT
STUDENT.StudentName,
COURSE.CourseName,
GRADE.Grade
FROM STUDENT
INNER JOIN GRADE ON STUDENT.StudentId = GRADE.StudentId
INNER JOIN COURSE ON GRADE.CourseId = COURSE.CourseId
GO
```

To create the View, we just create and run it in the Query Editor in SQL Server Management Studio

# Create the View

The screenshot displays the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer shows the database structure for 'HANS-PETTER\SQLEXPRESS (SQL Server 16.0.)'. The 'Views' folder under the 'dbo' schema is highlighted with a red rectangle, and 'dbo.StudentData' is selected. The main Query Editor window shows the following SQL script:

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

CREATE VIEW StudentData
AS

SELECT
STUDENT.StudentName,
COURSE.CourseName,
GRADE.Grade
FROM STUDENT
INNER JOIN GRADE ON STUDENT.StudentId = GRADE.StudentId
INNER JOIN COURSE ON GRADE.CourseId = COURSE.CourseId
GO
```

At the bottom, the Messages pane shows the execution results:

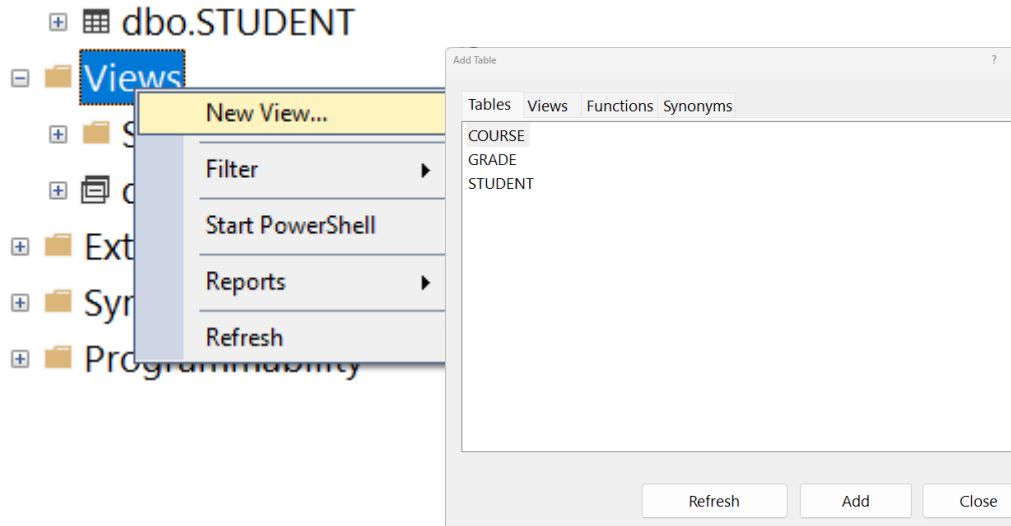
```
Commands completed successfully.
Completion time: 2025-04-28T09:45:47.1631960+02:00
```

The status bar at the bottom indicates 'Query executed successfully.' and 'HANS-PETTER\SQLEXPRESS (16.0.0.0) | sa (51) | GRADES | 00:00:00 | 0 rows'.

To create the View, we just create and run it in the Query Editor in SQL Server Management Studio

# View Designer

To can also use the “View Designer” in SQL Server Management Studio



Either you create the View using the “View Designer” or create the Views manually in SQL Server Management Studio, I advice you to save your View as a SQL Script file (file with extension .sql) so you can use it later, insert it on other databases, etc.

Hans-Petter...dbo.View\_1\*

The View Designer shows three tables: STUDENT, COURSE, and GRADE. The STUDENT table has columns: \* (All Columns), StudentId (checked), AverageGrade. The COURSE table has columns: \* (All Columns), CourseId (checked), CourseName. The GRADE table has columns: \* (All Columns), GradeId, CourseId, StudentId, Grade (checked). Relationships are shown between STUDENT.StudentId and GRADE.StudentId, and between COURSE.CourseId and GRADE.CourseId.

Column	Alias	Table	O...	Sort T...	Sort O...	Filter	Or...	Or...	Or...
Stude...		STU...	■						
Cours...		COU...	■						
Grade		GRA...	■						

```
SELECT  dbo.STUDENT.StudentName, dbo.COURSE.CourseName, dbo.GRADE.Grade
FROM    dbo.STUDENT INNER JOIN
        dbo.GRADE ON dbo.STUDENT.StudentId = dbo.GRADE.StudentId INNER JOIN
        dbo.COURSE ON dbo.GRADE.CourseId = dbo.COURSE.CourseId
```

# Using the View

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to 'HANS-PETTER\SQLEXPRESS.GRADES (sa (52))'. The menu bar includes File, Edit, View, Project, Tools, Window, and Help. The toolbar contains icons for various database operations. The Object Explorer on the left shows the database structure, with 'dbo.StudentData' selected under the 'Views' folder. The query editor in the center contains the SQL statement: `select * from StudentData`. The Results pane at the bottom displays the output of the query as a table with three columns: StudentName, CourseName, and Grade. The table contains three rows of data. The status bar at the bottom indicates that the query was executed successfully and returned 3 rows.

SQLQuery1.sql - HANS-PETTER\SQLEXPRESS.GRADES (sa (52))\* - Microsoft SQL Server Management Studio

File Edit View Project Tools Window Help

Object Explorer

Connect

HANS-PETTER\SQLEXPRESS (SQL Server 16.0.)

- Databases
  - System Databases
  - Database Snapshots
  - BOOKS
  - GRADES
    - Database Diagrams
    - Tables
    - Views
      - System Views
      - dbo.StudentData
    - External Resources
    - Synonyms
    - Programmability
    - Query Store
    - Service Broker
    - Storage
    - Security
  - ORDERS
  - SENSORSYSTEM

SQLQuery1.s...ES (sa (52))\*

```
select * from StudentData
```

Results

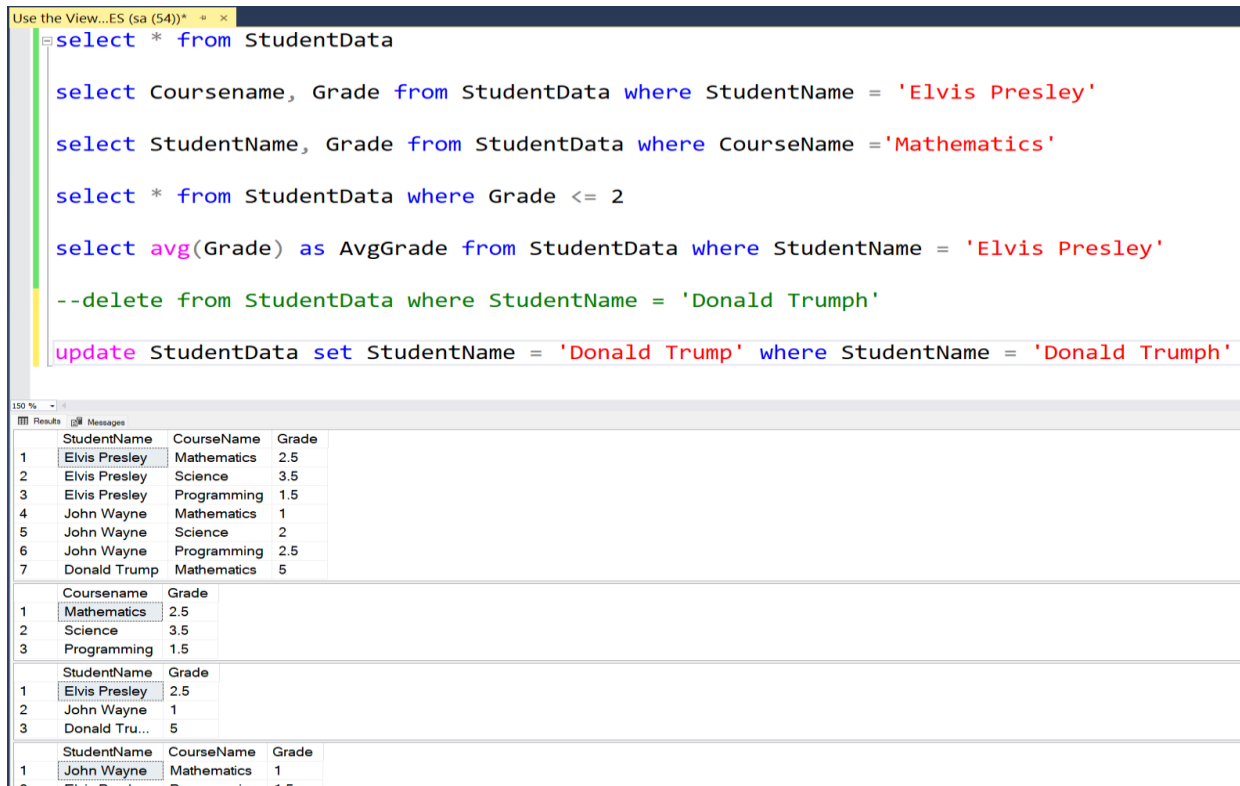
	StudentName	CourseName	Grade
1	Elvis Presley	Mathematics	2.5
2	Elvis Presley	Science	3.5
3	Elvis Presley	Programming	1.5

Query executed successfully. | HANS-PETTER\SQLEXPRESS (16... | sa (52) | GRADES | 00:00:00 | 3 rows



# Views Queries

You can use Views almost as you use Tables. Here are some examples:



The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL statements:

```
select * from StudentData

select CourseName, Grade from StudentData where StudentName = 'Elvis Presley'

select StudentName, Grade from StudentData where CourseName = 'Mathematics'

select * from StudentData where Grade <= 2

select avg(Grade) as AvgGrade from StudentData where StudentName = 'Elvis Presley'

--delete from StudentData where StudentName = 'Donald Trump'

update StudentData set StudentName = 'Donald Trump' where StudentName = 'Donald Trump'
```

The results pane displays the output of the queries. It shows a table of StudentData, a table of CourseName and Grade, a table of StudentName and Grade, and a table of StudentName, CourseName, and Grade.

StudentName	CourseName	Grade	
1	Elvis Presley	Mathematics	2.5
2	Elvis Presley	Science	3.5
3	Elvis Presley	Programming	1.5
4	John Wayne	Mathematics	1
5	John Wayne	Science	2
6	John Wayne	Programming	2.5
7	Donald Trump	Mathematics	5

CourseName	Grade	
1	Mathematics	2.5
2	Science	3.5
3	Programming	1.5

StudentName	Grade	
1	Elvis Presley	2.5
2	John Wayne	1
3	Donald Tru...	5

StudentName	CourseName	Grade	
1	John Wayne	Mathematics	1

But you typically cannot  
use Delete

<https://www.halvorsen.blog>

# Stored Procedures

[Table of Contents](#)

Hans-Petter Halvorsen



# Problem Description

To create/insert Grades we need to create and execute queries like this:

```
insert into GRADE (CourseId, StudentId, Grade) values (1, 1, 2.5)
```

```
insert into GRADE (CourseId, StudentId, Grade) values (2, 1, 3.5)
```

```
insert into GRADE (CourseId, StudentId, Grade) values (3, 1, 1.5)
```

The “drawback” is that we need to remember the CourseIds and the StudentIds, typically we only remember and want to use their names.

=> The solution is to create and use a **Stored Procedure**.

# Stored Procedures

- A Stored Procedure is very similar as a Method/Function in C# or Python - it is a piece of code with SQL commands that do a specific task – and you can reuse it.
- A Stored Procedure can have Input Arguments and Return values (just like a Method/Function).
- It also adds a layer of security, because you can do a lot of harm by creating the wrong queries. In that way you can create a set of Stored Procedures that is well implemented and tested properly.
- Stored Procedures can also prevent “SQL Injection” used by “hackers”, etc.

# Stored Procedure Example

```
CREATE PROCEDURE CreateStudentGrade
```

```
@StudentName varchar(50),  
@CourseName varchar(50),  
@Grade float  
AS
```

} Input Arguments. Note the “@” before the variable names.

```
DECLARE  
@StudentId int,  
@CourseId int
```

} Internal variables

```
select @StudentId = StudentId from STUDENT where StudentName = @StudentName
```

```
select @CourseId = CourseId from COURSE where CourseName = @CourseName
```

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

# Create the Stored Procedure

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

```
CREATE PROCEDURE CreateStudentGrade
@StudentName varchar(50),
@CourseName varchar(50),
@Grade float
```

AS

```
DECLARE
@StudentId int,
@CourseId int
```

```
select @StudentId = StudentId from STUDENT where StudentName = @StudentName
```

```
select @CourseId = CourseId from COURSE where CourseName = @CourseName
```

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

To create the Stored Procedure, we just create and run it in the Query Editor in SQL Server Management Studio

# Create the Stored Procedure

The screenshot displays the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer shows the database structure for 'HANS-PETTER\SQLEXPRESS (SQL Server 16.0.11058.1)'. The 'Programmability' folder is expanded, and the 'Stored Procedures' sub-folder is highlighted with a red rectangle. The main Query Editor window shows the following SQL script:

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

CREATE PROCEDURE CreateStudentGrade
    @StudentName varchar(50),
    @CourseName varchar(50),
    @Grade float
AS
DECLARE
    @StudentId int,
    @CourseId int

select @StudentId = StudentId from STUDENT where StudentName = @StudentName

select @CourseId = CourseId from COURSE where CourseName = @CourseName

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

At the bottom, the Messages pane shows the execution results:

```
Commands completed successfully.
Completion time: 2025-04-28T10:21:58.5513804+02:00
Query executed successfully.
```

To create the Stored Procedure, we just create and run it in the Query Editor in SQL Server Management Studio

# Using the Stored Procedure

```
insert into GRADE (CourseId, StudentId, Grade) values (1, 1, 2.5)
```

```
insert into GRADE (CourseId, StudentId, Grade) values (2, 1, 3.5)
```

```
insert into GRADE (CourseId, StudentId, Grade) values (3, 1, 1.5)
```



```
execute CreateStudentGrade 'John Wayne ', 'Mathematics', 1.0
```

```
execute CreateStudentGrade 'John Wayne', 'Science', 2.0
```

```
execute CreateStudentGrade 'John Wayne', 'Programming', 2.5
```



# Using the Stored Procedure

We can now insert Grades using the Stored Procedure:

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the Object Explorer displays the database structure for 'HANS-PETTER\SQLEXPRESS (SQL Server 16.0.11)', including 'Databases', 'System Databases', 'Database Snapshots', 'BOOKS', 'GRADES', 'Database Diagrams', 'Tables', 'Views', 'External Resources', 'Synonyms', 'Programmability', 'Query Store', and 'Service Broker'. The 'GRADES' database is selected. The main window shows the execution of a stored procedure named 'CreateStudentGrade' with three rows of data: 'John Wayne' in 'Mathematics' with a grade of 1.0, 'John Wayne' in 'Science' with a grade of 2.0, and 'John Wayne' in 'Programming' with a grade of 2.5. Below the execution window, a query window shows the SQL query 'select \* from StudentData'. The results of the query are displayed in a table with columns 'StudentName', 'CourseName', and 'Grade'.

```
execute CreateStudentGrade 'John Wayne ', 'Mathematics', 1.0  
execute CreateStudentGrade 'John Wayne', 'Science', 2.0  
execute CreateStudentGrade 'John Wayne', 'Programming', 2.5
```

```
select * from StudentData
```

	StudentName	CourseName	Grade
1	Elvis Presley	Mathematics	2.5
2	Elvis Presley	Science	3.5
3	Elvis Presley	Programming	1.5
4	John Wayne	Mathematics	1
5	John Wayne	Science	2
6	John Wayne	Programming	2.5

Then we can use the View to see the grades for the different students in the different courses:

# Using the Stored Procedure

Then we can use the View to see the grades for the different students in the different courses:

SQLQuery1.s...ES (sa (54))\*

```
select * from StudentData
```

150 %

Results Messages

	StudentName	CourseName	Grade
1	Elvis Presley	Mathematics	2.5
2	Elvis Presley	Science	3.5
3	Elvis Presley	Programming	1.5
4	John Wayne	Mathematics	1
5	John Wayne	Science	2
6	John Wayne	Programming	2.5

If we only want to see the grades for a specific Student , we can do like this:

SQLQuery2.s...ES (sa (54))\*

```
select * from StudentData where StudentName='Elvis Presley'
```

150 %

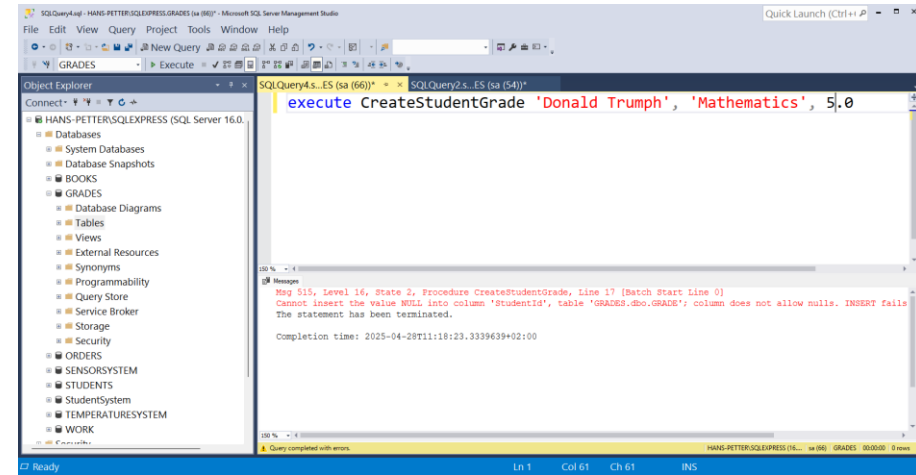
Results Messages

	StudentName	CourseName	Grade
1	Elvis Presley	Mathematics	2.5
2	Elvis Presley	Science	3.5
3	Elvis Presley	Programming	1.5

# Updated version

- Assume we use a StudentName or a CourseName that do not exist in the database.
- Or that the Grade already exists?

In this case the student “Donal Triumph” does not exists and we get an error message:



# Updated version #1

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

```
CREATE PROCEDURE CreateStudentGrade
@StudentName varchar(50),
@CourseName varchar(50),
@Grade float
```

AS

DECLARE

```
@StudentId int,
@CourseId int
```

```
if exists (select * from STUDENT where StudentName = @StudentName)
select @StudentId = StudentId from STUDENT where StudentName = @StudentName
```

```
if exists (select * from COURSE where CourseName = @CourseName)
select @CourseId = CourseId from COURSE where CourseName = @CourseName
```

```
if (@StudentId is not null and @CourseId is not null)
insert into GRADE (StudentId, CourseId, Grade) values (@StudentId, @CourseId, @Grade)
else
print 'Student or Course do not exist'
GO
```

Now the Stored Procedure checks if the Student or Course exist and if not, no data is inserted, and you get a message saying “Student or Course do not exist”.

# Updated version #2

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

CREATE PROCEDURE CreateStudentGrade
@StudentName varchar(50),
@CourseName varchar(50),
@Grade float
AS

DECLARE
@StudentId int,
@CourseId int

if not exists (select * from STUDENT where StudentName = @StudentName)
insert into STUDENT (StudentName) values (@StudentName)

select @StudentId = StudentId from STUDENT where StudentName = @StudentName

if not exists (select * from COURSE where CourseName = @CourseName)
insert into COURSE (CourseName) values (@CourseName)

select @CourseId = CourseId from COURSE where CourseName = @CourseName

if (@StudentId is not null and @CourseId is not null)
insert into GRADE (StudentId, CourseId, Grade) values (@StudentId, @CourseId, @Grade)
else
print 'Something went wrong...'
GO
```

Now the Stored Procedure checks if the Student or Course exist and if not, the Student and/or Course is/are created.

<https://www.halvorsen.blog>

# Triggers

Hans-Petter Halvorsen



[Table of Contents](#)

# Problem Description

Select Data....ES (sa (51))

```
select * from COURSE
select * from STUDENT
select * from GRADE
```

150 %

Results Messages

	Courseld	CourseName
1	1	Mathematics
2	3	Programming
3	2	Science

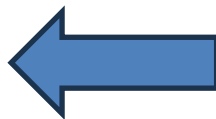
  

	StudentId	StudentName	AverageGrade
1	1	Elvis Presley	NULL
2	2	John Wayne	NULL
3	3	John Statham	NULL

	GradeId	Courseld	StudentId	Grade
1	6	1	1	2.5
2	7	2	1	3.5
3	8	3	1	1.5

```
execute CreateStudentGrade 'John Wayne ', 'Mathematics', 1.0
execute CreateStudentGrade 'John Wayne', 'Science', 2.0
execute CreateStudentGrade 'John Wayne', 'Programming', 2.5
```



We want to automatically update the “AverageGrade” for each student when inserting, updating or deleting Grades for a specific Student in a specific Course.

=> The solution is to create and use a **Trigger**.

# Triggers

- A Trigger is executed when you insert, update or delete data in a Table specified in the Trigger.
- A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs.
- A Trigger is attached to a specific Table.
- You can use a Trigger to change data in the same table or in other tables.



# Trigger Example

```
CREATE TRIGGER CalcAvgGrade ON GRADE
FOR INSERT, UPDATE, DELETE
AS
```

```
DECLARE
@StudentId int,
@AverageGrade float
```

```
select @StudentId = StudentId from INSERTED
```

```
select @AverageGrade = AVG(Grade) from GRADE where StudentId = @StudentId
```

```
update STUDENT set AverageGrade = @AverageGrade where StudentId = @StudentId
```

```
GO
```

You need to specify which Table the Trigger shall be attached to.

Note! “INSERTED” is a temporarily table containing the latest inserted data, and it is very handy to use inside a trigger.

# Create the Trigger

```
IF EXISTS (SELECT name
FROM sysobjects
WHERE name = 'CalcAvgGrade'
AND type = 'TR')
DROP TRIGGER CalcAvgGrade
GO
```

```
CREATE TRIGGER CalcAvgGrade ON GRADE
FOR INSERT, UPDATE, DELETE
AS
```

```
DECLARE
@StudentId int,
@AverageGrade float
```

```
select @StudentId = StudentId from INSERTED
```

```
select @AverageGrade = AVG(Grade) from GRADE where StudentId = @StudentId
```

```
update STUDENT set AverageGrade = @AverageGrade where StudentId = @StudentId
```

```
GO
```

# SQL Server Management Studio

The screenshot displays the Microsoft SQL Server Management Studio (SSMS) interface. The title bar indicates the current file is 'CalcAvgGrade Trigger.sql' and the server is 'HANS-PETTER\SQLEXPRESS.GRADES (sa (67))'. The menu bar includes File, Edit, View, Query, Project, Tools, Window, and Help. The toolbar contains icons for New Query, Execute, and other standard SSMS functions. The Object Explorer on the left shows the server hierarchy: HANS-PETTER\SQLEXPRESS (SQL Server 16.0) > Databases > GRADES > Triggers > CalcAvgGrade. The main query editor window shows a SQL script with the following content:

```
IF EXISTS (SELECT name
           FROM sysobjects
           WHERE name = 'CalcAvgGrade'
           AND type = 'TR')
    DROP TRIGGER CalcAvgGrade
GO

CREATE TRIGGER CalcAvgGrade ON GRADE
FOR INSERT, UPDATE, DELETE
AS

DECLARE
    @StudentId int,
    @AverageGrade float

select @StudentId = StudentId from INSERTED

select @AverageGrade = AVG(Grade) from GRADE where StudentId = @StudentId
```

Below the query editor, the Messages pane shows the execution results: 'Commands completed successfully.' and 'Completion time: 2025-04-25T10:22:39.9041617+02:00'. The status bar at the bottom indicates 'Query executed successfully.' and provides details about the server, database, and query execution time.

# Insert Grades

SQLQuery3.sql - HANS-PETTER\SQLEXPRESS.GRADES (sa (62)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

GRADES Execute

Object Explorer

Connect: HANS-PETTER\SQLEXPRESS (SQL Server 16.0)

- Databases
  - System Databases
  - Database Snapshots
  - BOOKS
  - GRADES
    - Database Diagrams
    - Tables
      - System Tables
      - FileTables
      - External Tables
      - Graph Tables
      - dbo.COURSE
      - dbo.GRADES

CalcAvgGrad...ES (sa (67))

Select Data...ES (sa (52))

SQLQuery3.s...ES (sa (62))\*

```
insert into GRADE (CourseId, StudentId, Grade) values (1, 1, 2.5)
```

Completion time: 2025-04-25T10:26:34.1518457+02:00

Query executed successfully.

150 %

Results Messages

	CourseId	CourseName
1	1	Mathematics
2	3	Programming Science

You can also use the Stored Procedure we made earlier:

```
execute CreateStudentGrade 'John Wayne ', 'Mathematics', 1.0
```

	StudentId	StudentName	AverageGrade
1	1	Elvis Presley	2.5
2	2	John Wayne	NULL
3	3	John Statham	NULL

	GradeId	CourseId	StudentId	Grade
1	1	1	1	2.5

# Results

Here student “Elvis Presley” (StudentId=1) get his grades in the courses:

“Mathematics” (CourseId=1) => Grade = 2.5

“Science” (CourseId=2) => Grade = 3.5

“Programming” (CourseId=3) => Grade = 1.5

```
insert into GRADE (CourseId, StudentId, Grade)
values (1, 1, 2.5)
```

```
insert into GRADE (CourseId, StudentId, Grade)
values (2, 1, 3.5)
```

```
insert into GRADE (CourseId, StudentId, Grade)
values (3, 1, 1.5)
```

CalcAvgGrad...ES (sa (67))    Select Data....ES (sa (52))

```
select * from COURSE
select * from STUDENT
select * from GRADE
```

150 %

Results    Messages

	CourseId	CourseName
1	1	Mathematics
2	3	Programming
3	2	Science

	StudentId	StudentName	AverageGrade
1	1	Elvis Presley	2.5
2	2	John Wayne	NULL
3	3	John Statham	NULL

	GradeId	CourseId	StudentId	Grade
1	6	1	1	2.5
2	7	2	1	3.5
3	8	3	1	1.5

# Hans-Petter Halvorsen

University of South-Eastern Norway

[www.usn.no](http://www.usn.no)

E-mail: [hans.p.halvorsen@usn.no](mailto:hans.p.halvorsen@usn.no)

Web: <https://www.halvorsen.blog>

