https://www.halvorsen.blog

Views, Stored Procedures and Triggers in SQL Server

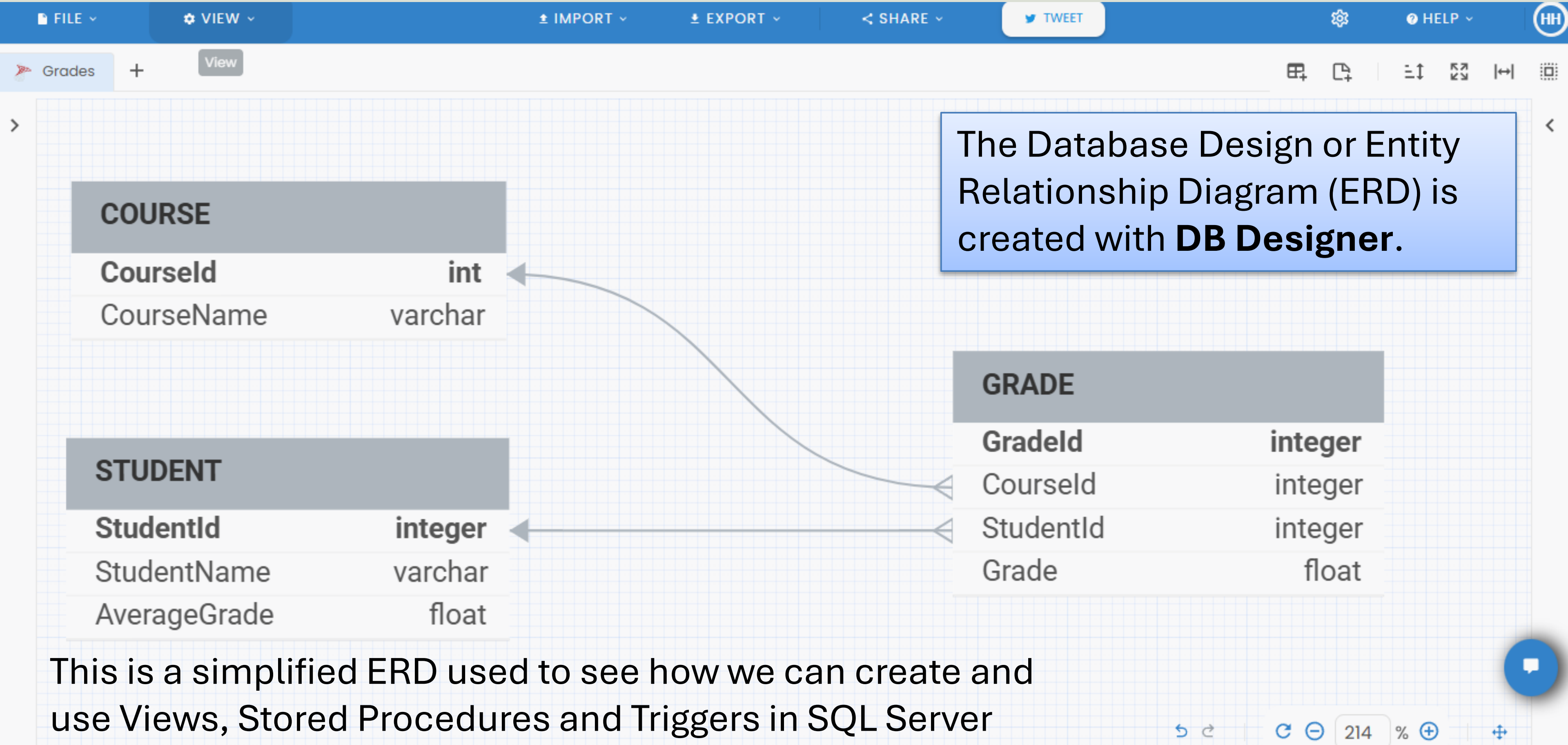
Hans-Petter Halvorsen

Contents

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

Introduction

Database



The Database Design or Entity Relationship Diagram (ERD) is created with **DB Designer**.

COURSE	
CourseId	**int**
CourseName	varchar

STUDENT	
StudentId	**integer**
StudentName	varchar
AverageGrade	float

GRADE	
GradeId	**integer**
CourseId	integer
StudentId	integer
Grade	float

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

Table Script

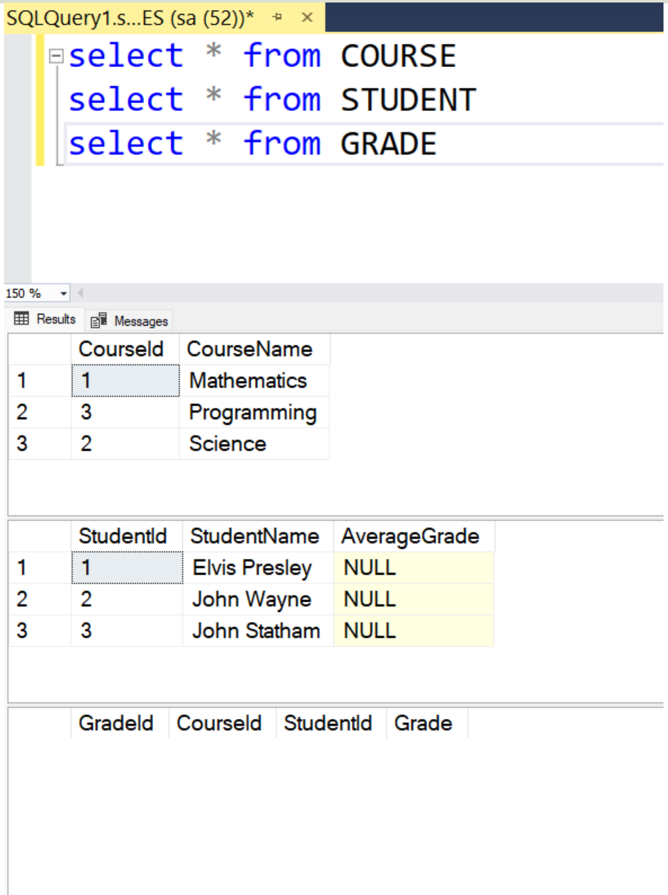
```
CREATE TABLE [COURSE] (
[CourseId] int IDENTITY(1,1) NOT NULL UNIQUE,
[CourseName] nvarchar(50) NOT NULL UNIQUE,
PRIMARY KEY ([CourseId])
);

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

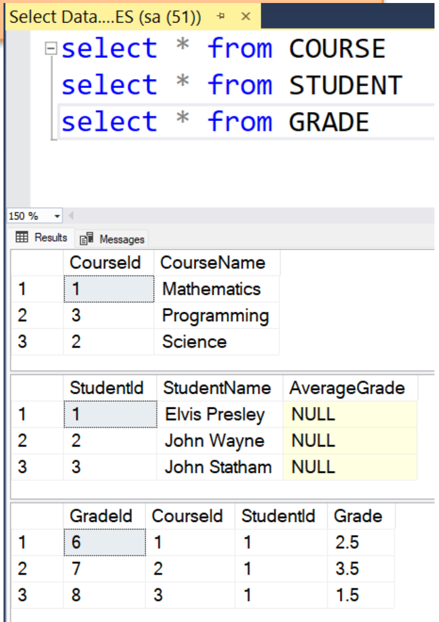
	GradeId	CourseId	StudentId	Grade

Insert Grades

```sql
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



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

	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

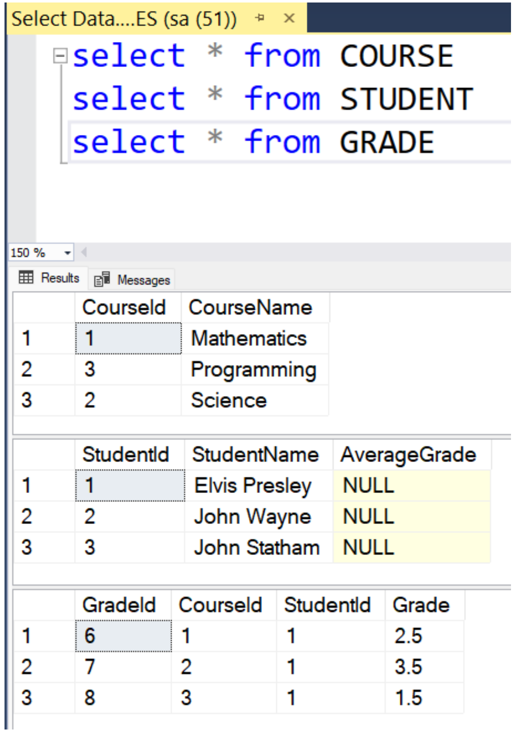
	GradeId	CourseId	StudentId	Grade
1	6	1	1	2.5
2	7	2	1	3.5
3	8	3	1	1.5

Views

Hans-Petter Halvorsen

Problem Description

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



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

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

150 %

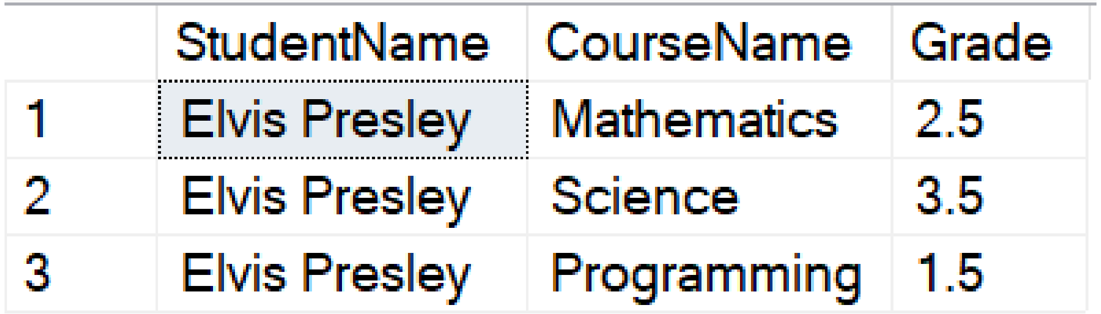
Results | Messages

	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

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

```sql
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
```

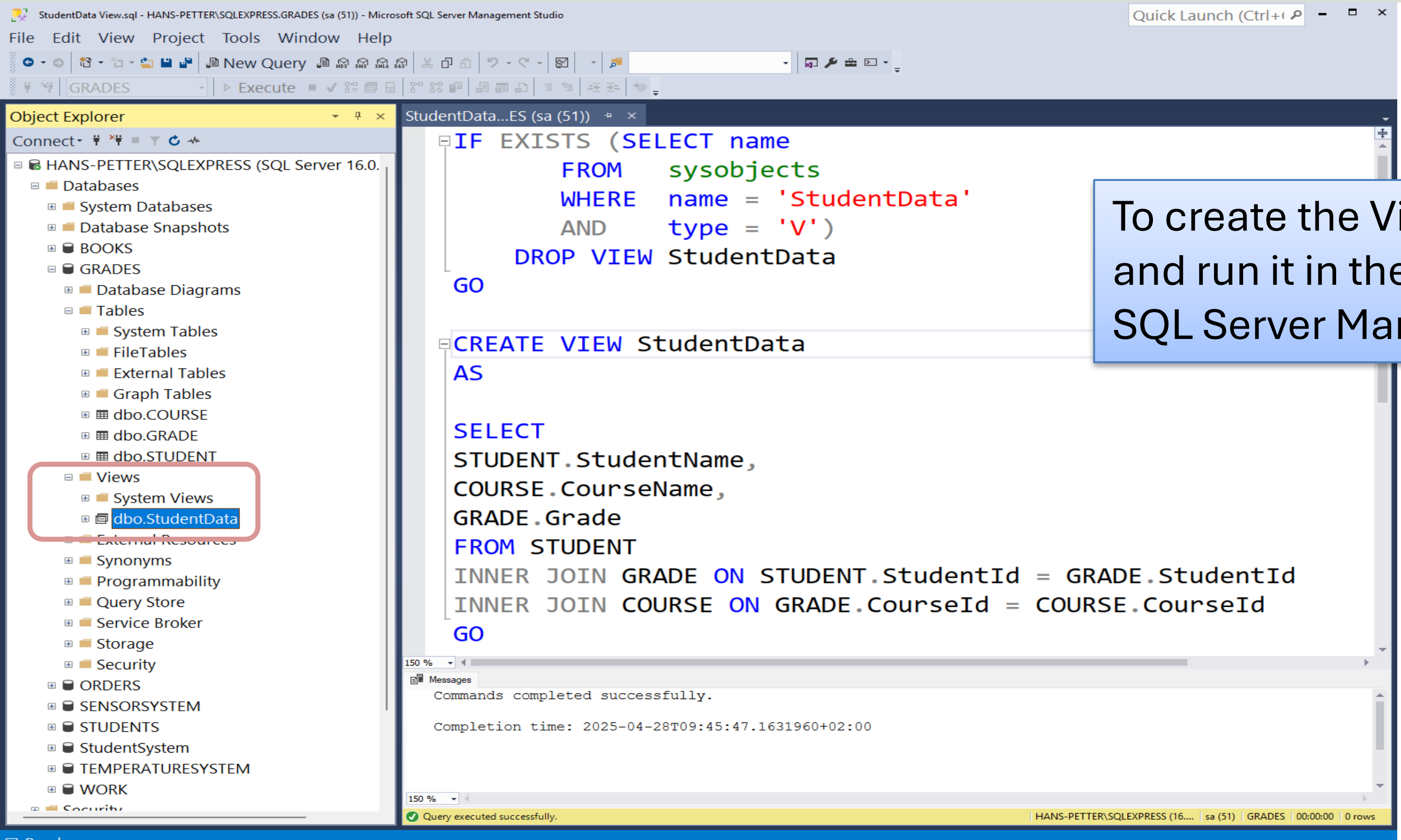
In a View we typically use "**INNER JOIN**" to join information stored in different Tables.

Create the View

```sql
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



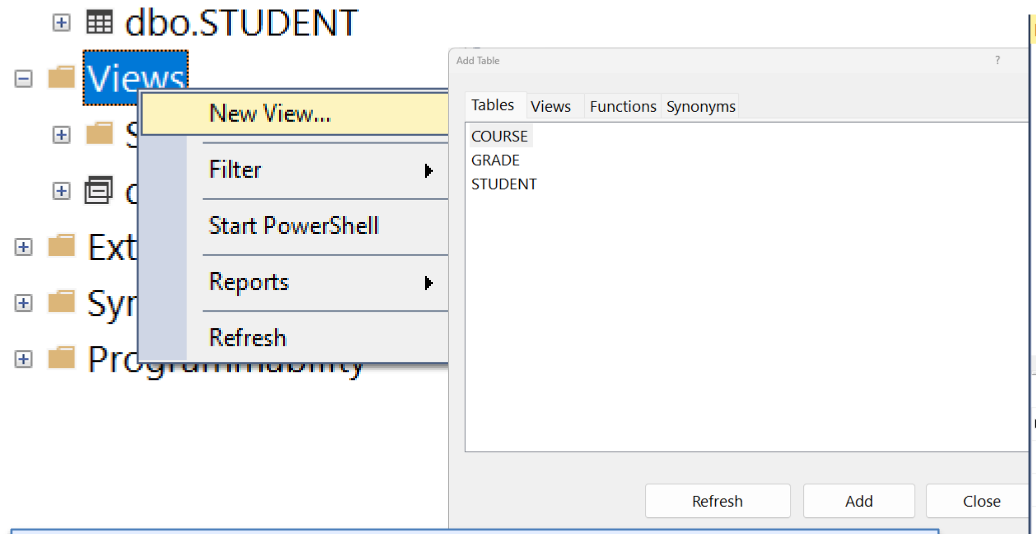
```
IF EXISTS (SELECT name
        FROM    sysobjects
        WHERE   name = 'StudentData'
        AND     type = 'V')
    DROP VIEW StudentData
GO

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
```

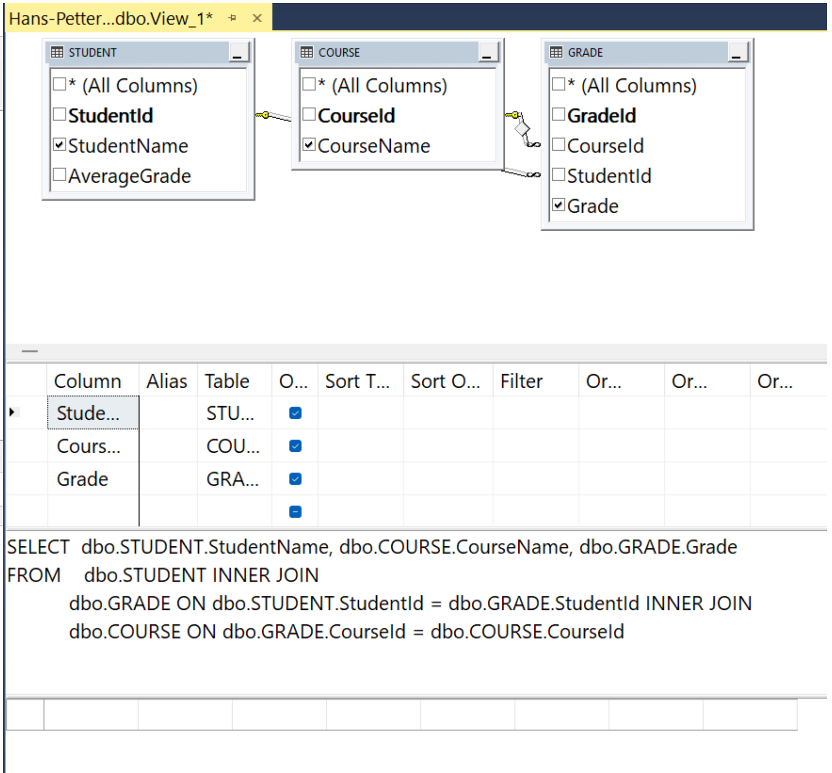
```
Commands completed successfully.

Completion time: 2025-04-28T09:45:47.1631960+02:00
```

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



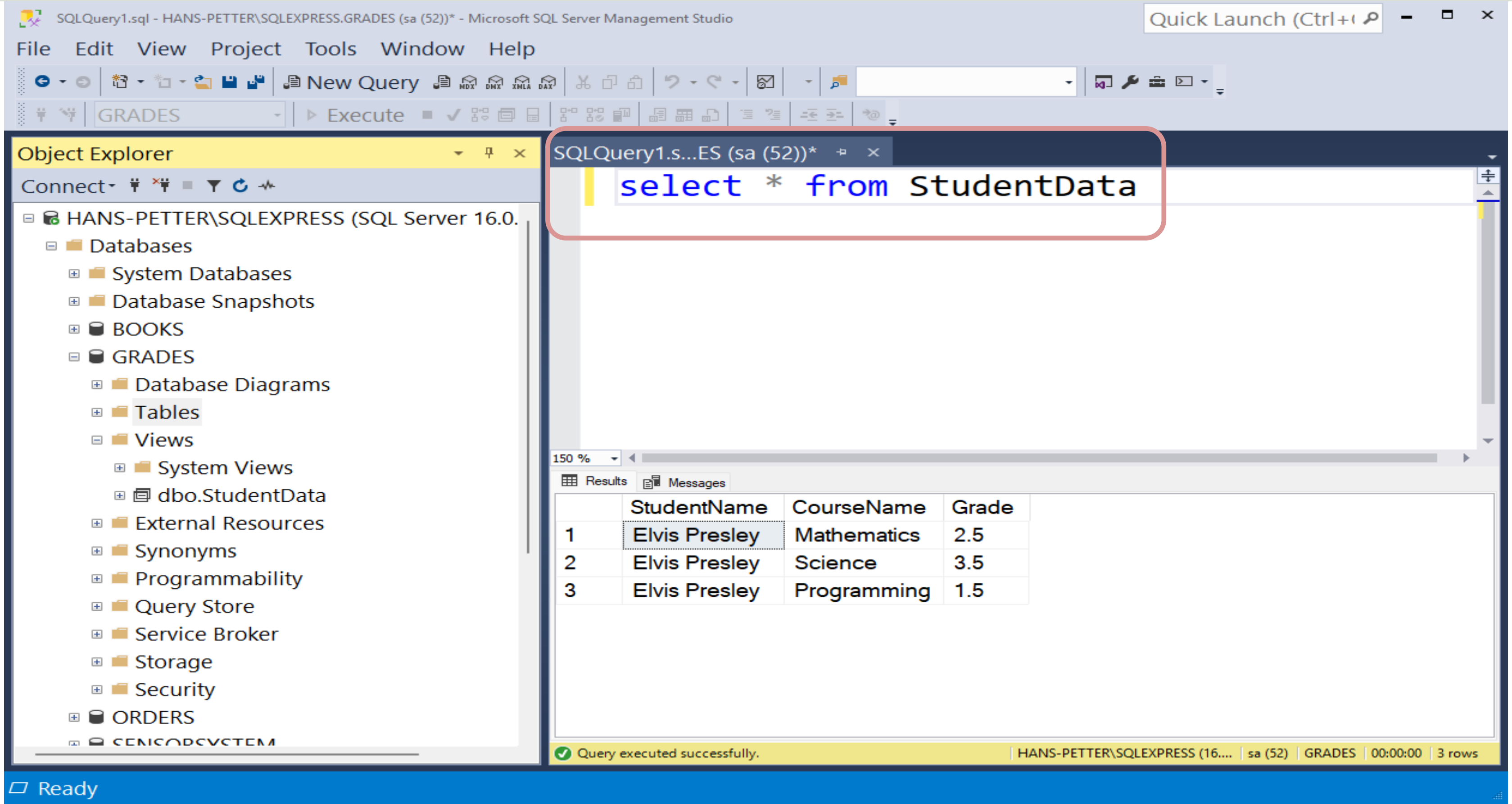


Column	Alias	Table	O...	Sort T...	Sort O...	Filter	Or...	Or...	Or...
Stude...		STU...	✓						
Cours...		COU...	✓						
Grade		GRA...	✓						
			✓						

```
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
```

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.

Using the View



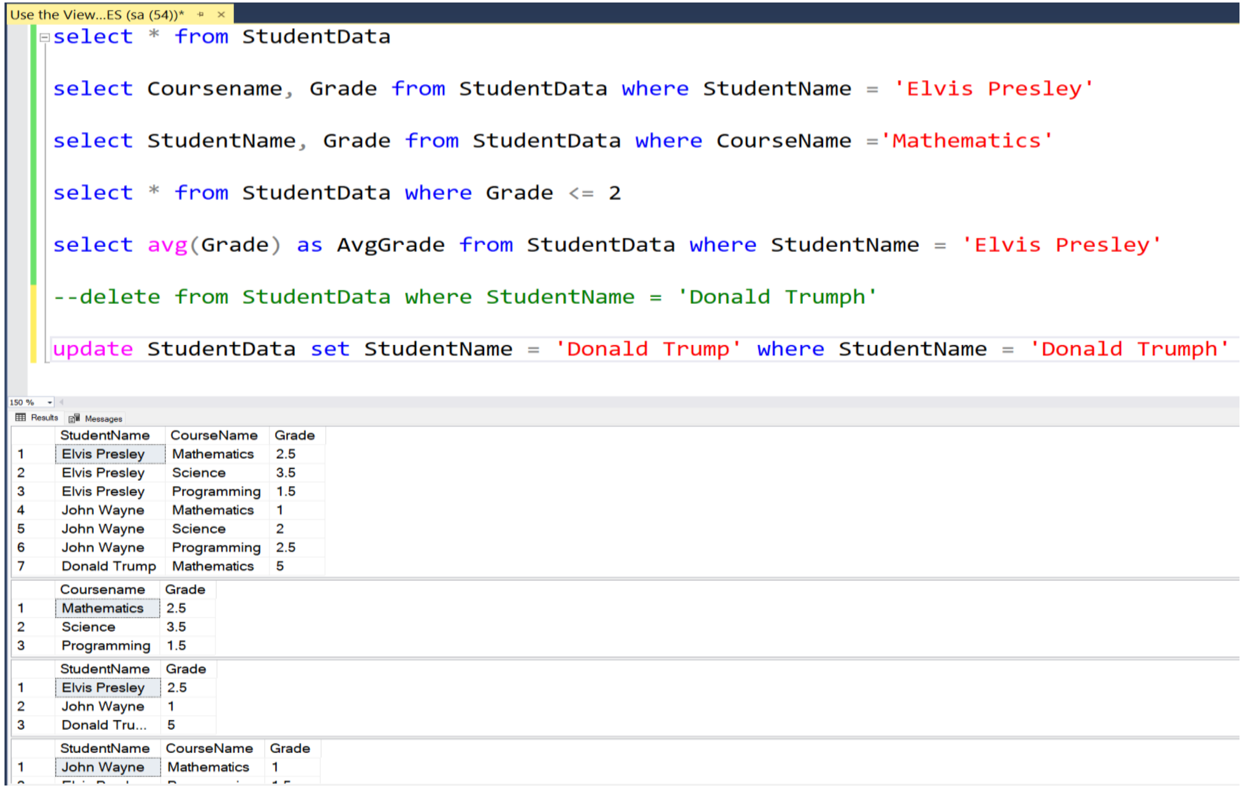
```
select * from StudentData
```

	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:



```
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 Trumph'

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

	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

Stored Procedures

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

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

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

```sql
AS

DECLARE
@StudentId int,
@CourseId int
```

Internal variables

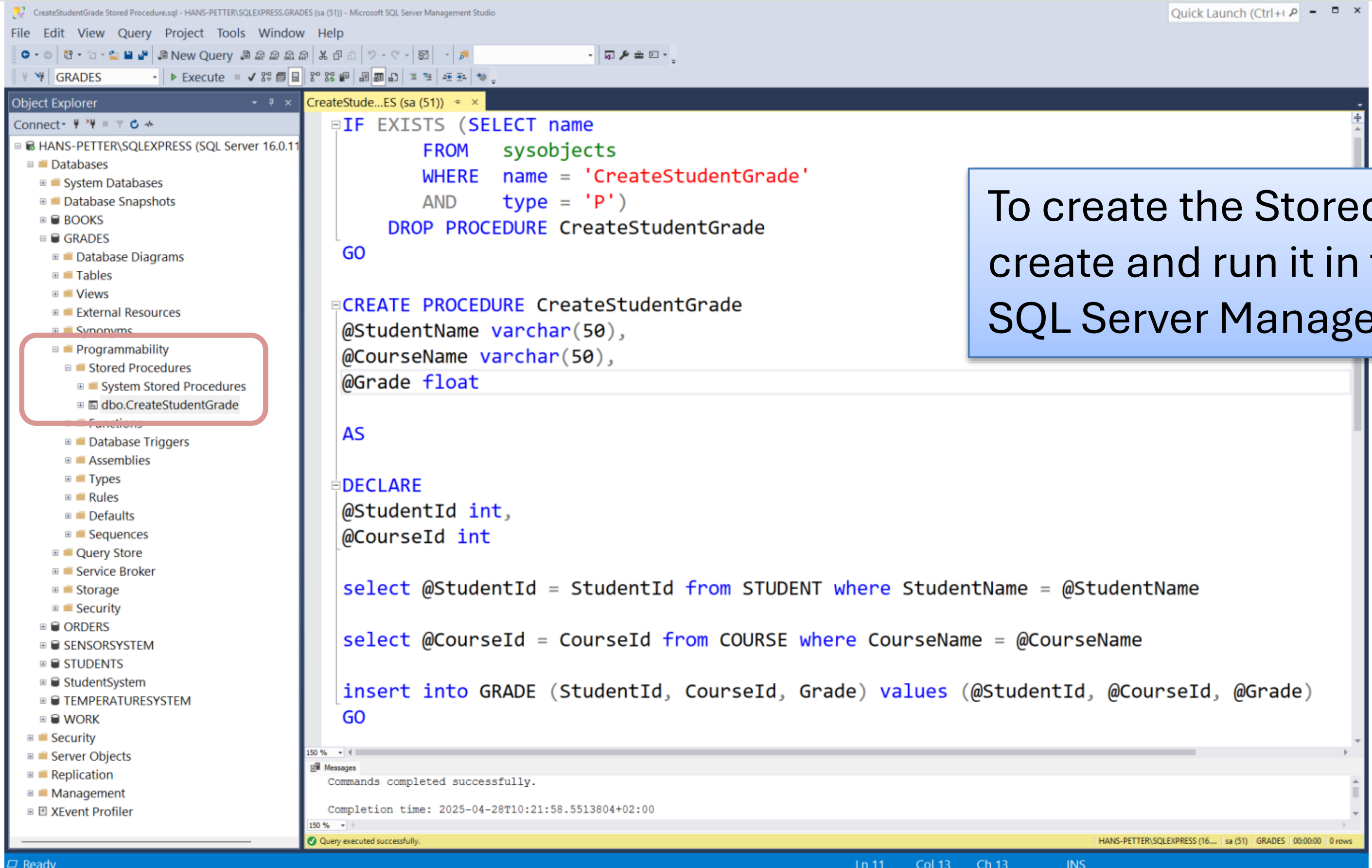
```sql
select @StudentId = StudentId from STUDENT where StudentName = @StudentName

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

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

Create the Stored Procedure

```sql
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



```sql
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
```

```
Commands completed successfully.

Completion time: 2025-04-28T10:21:58.5513804+02:00
```

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

Using the Stored Procedure

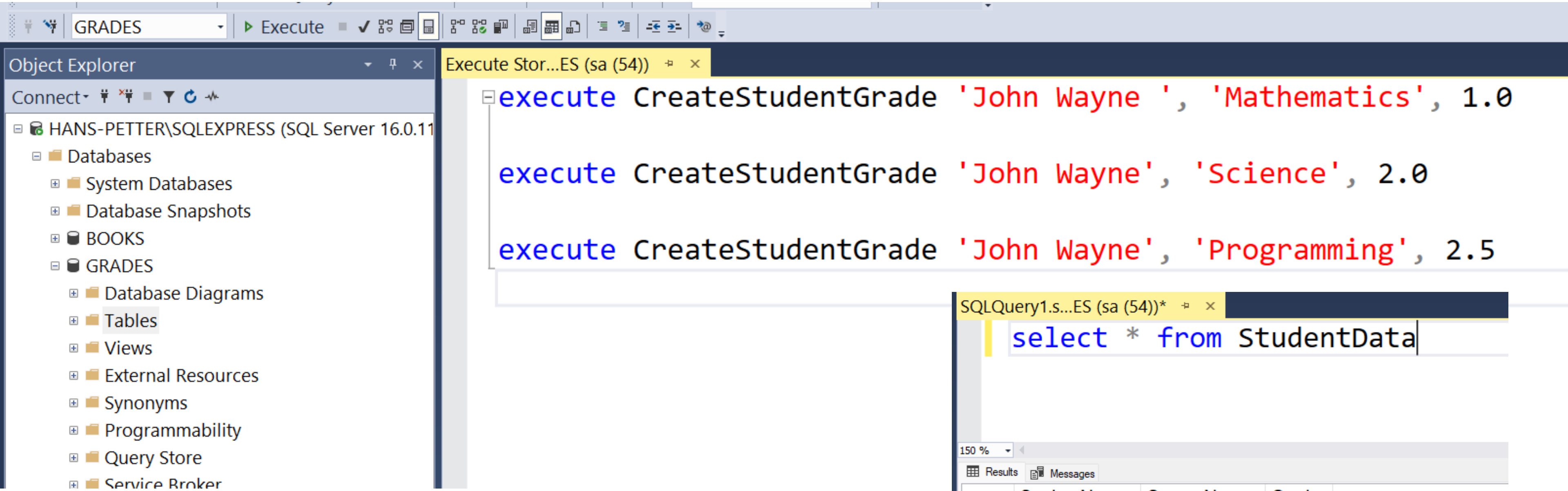
```
insert into GRADE (CourseId, StudentId, Grade) values (1, 1, 2.5)

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

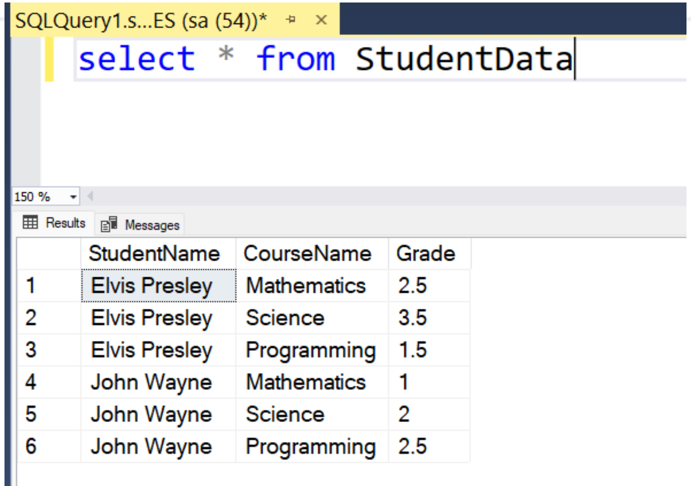
We can now insert Grades using the Stored Procedure:



```
HANS-PETTER\SQLEXPRESS (SQL Server 16.0.11
  Databases
    System Databases
    Database Snapshots
    BOOKS
    GRADES
      Database Diagrams
      Tables
      Views
      External Resources
      Synonyms
      Programmability
      Query Store
      Service Broker
```

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

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

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

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

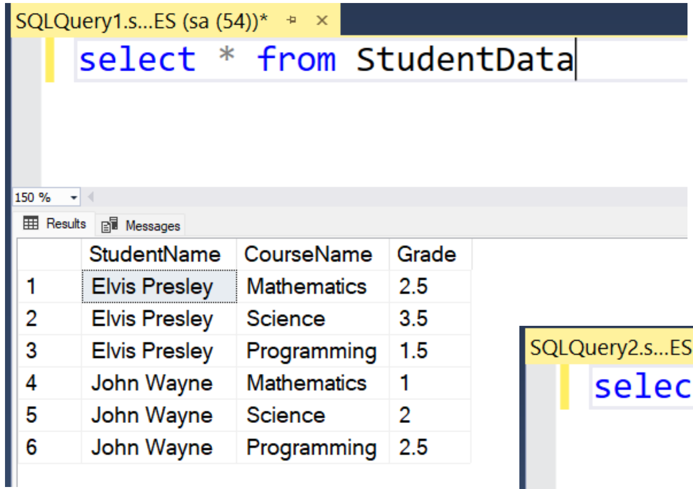


```sql
select * from StudentData
```

	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

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))*

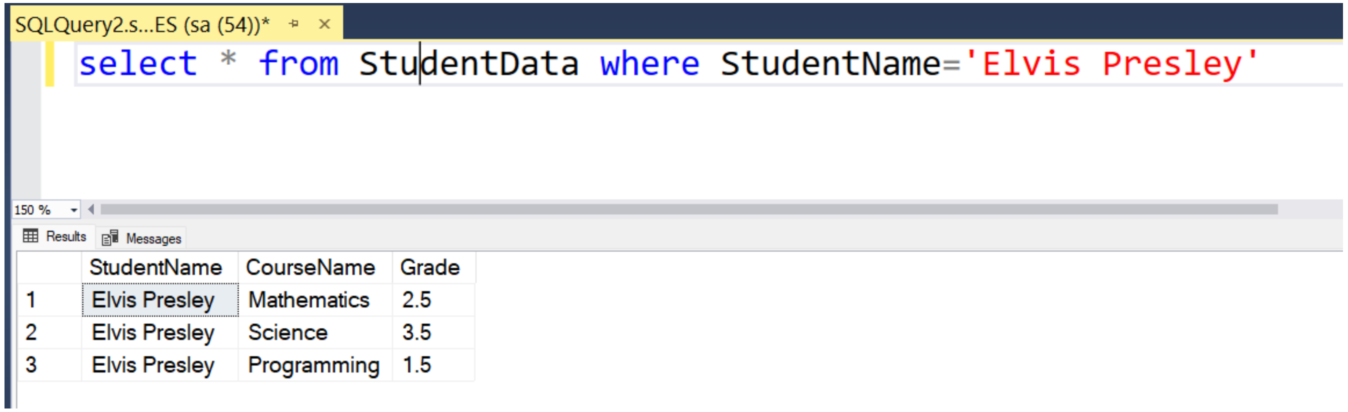
```
select * from StudentData
```

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 %

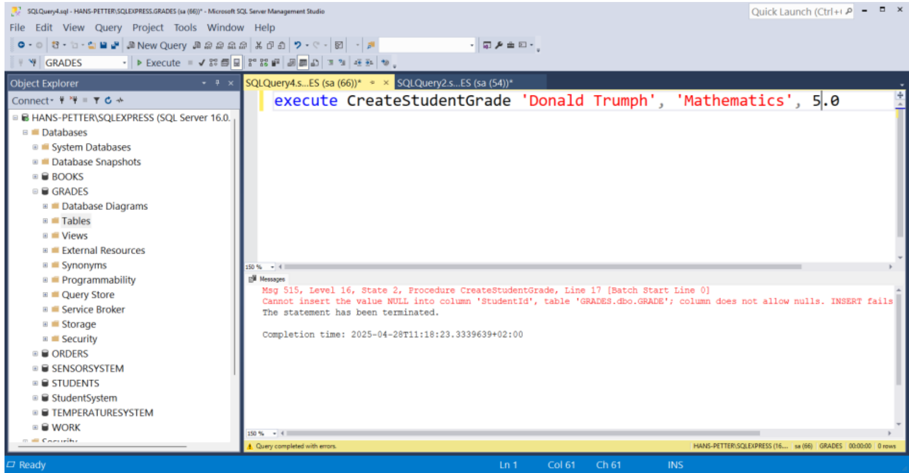
Results Messages

	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 Trumph” does not exists and we get an error message:



```
execute CreateStudentGrade 'Donald Trumph', 'Mathematics', 5.0
```

```
Msg 515, Level 16, State 2, Procedure CreateStudentGrade, Line 17 [Batch Start Line 0]
Cannot insert the value NULL into column 'StudentId', table 'GRADES.dbo.GRADE'; column does not allow nulls. INSERT fails
The statement has been terminated.

Completion time: 2025-04-28T11:18:23.3339639+02:00
```

Updated version #1

```sql
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

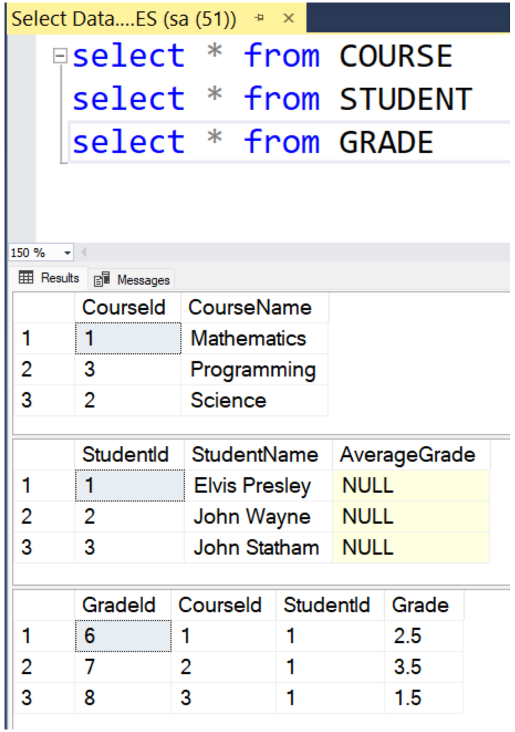
```sql
IF EXISTS (SELECT name
   FROM   sysobjects
   WHERE  name = 'CreateStudentGrade'
   AND   type = 'P')
DROP PROCEDURE CreateStudentGrade
GO

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.

Triggers

Hans-Petter Halvorsen

Problem Description



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

```sql
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

```sql
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

```sql
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
```

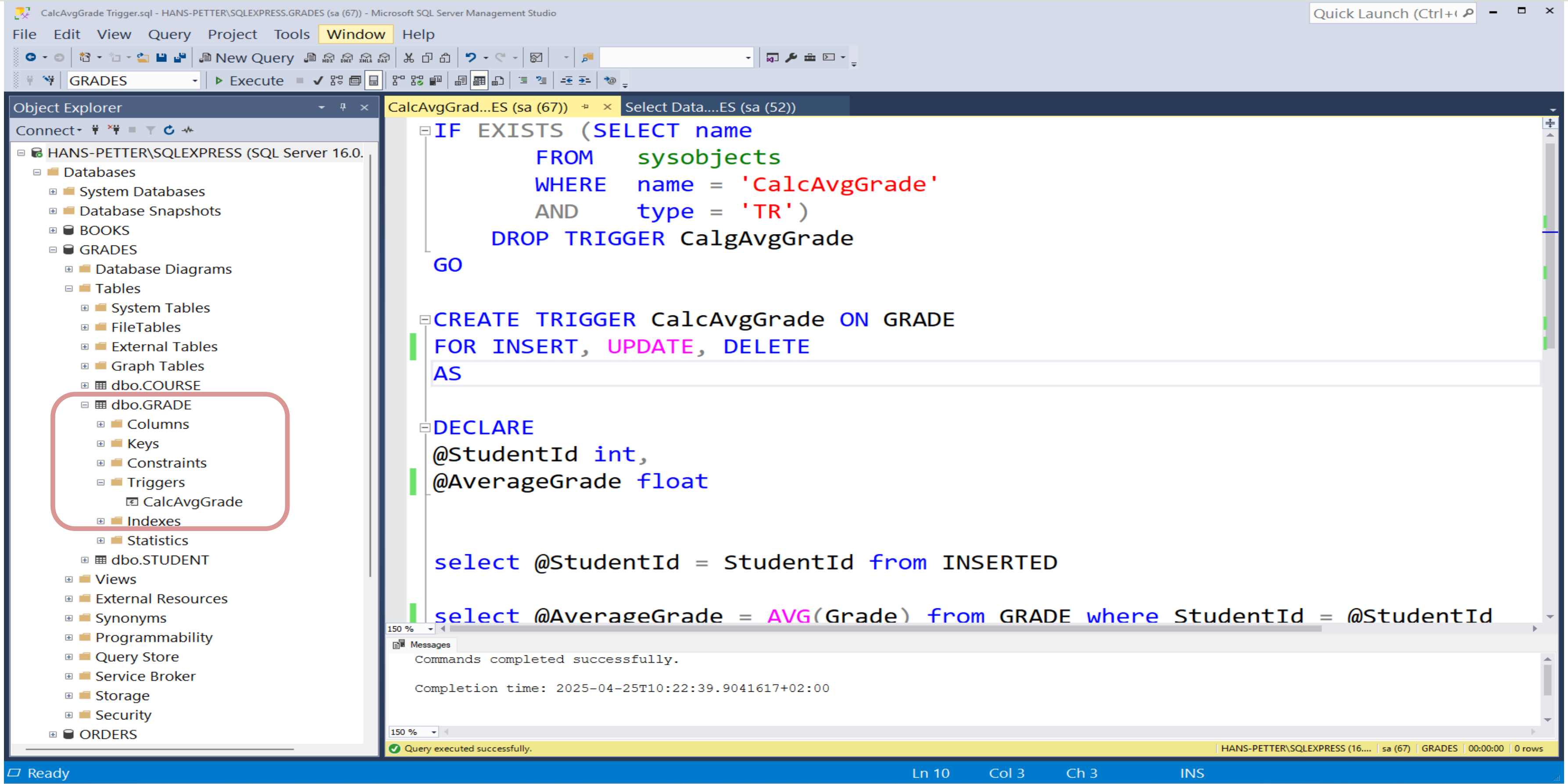
You need to specify which Table the Trigger shall be attached to.

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 CalgAvgGrade
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
```

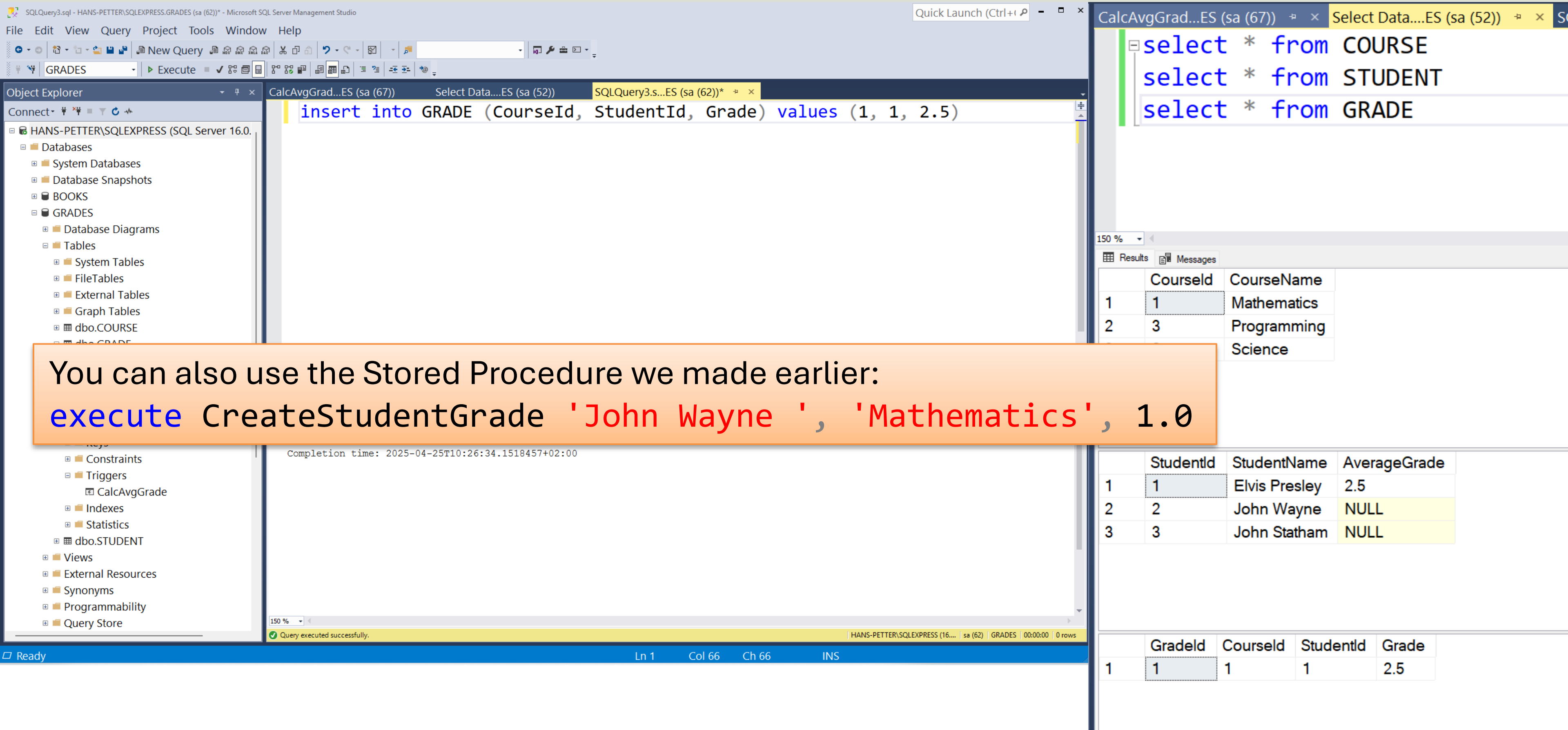
SQL Server Management Studio



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

```
Commands completed successfully.

Completion time: 2025-04-25T10:22:39.9041617+02:00
```

Insert Grades



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

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

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

	CourseId	CourseName
1	1	Mathematics
2	3	Programming
		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	1	1	1	2.5

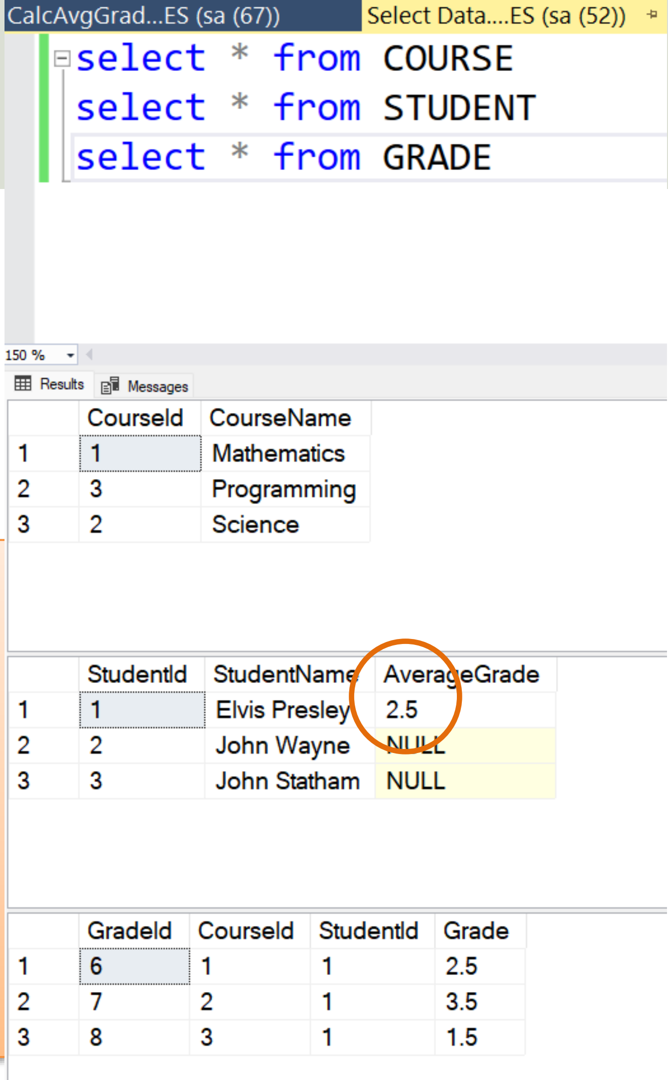
You can also use the Stored Procedure we made earlier:

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

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)
```



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

	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

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog