

<https://www.halvorsen.blog>



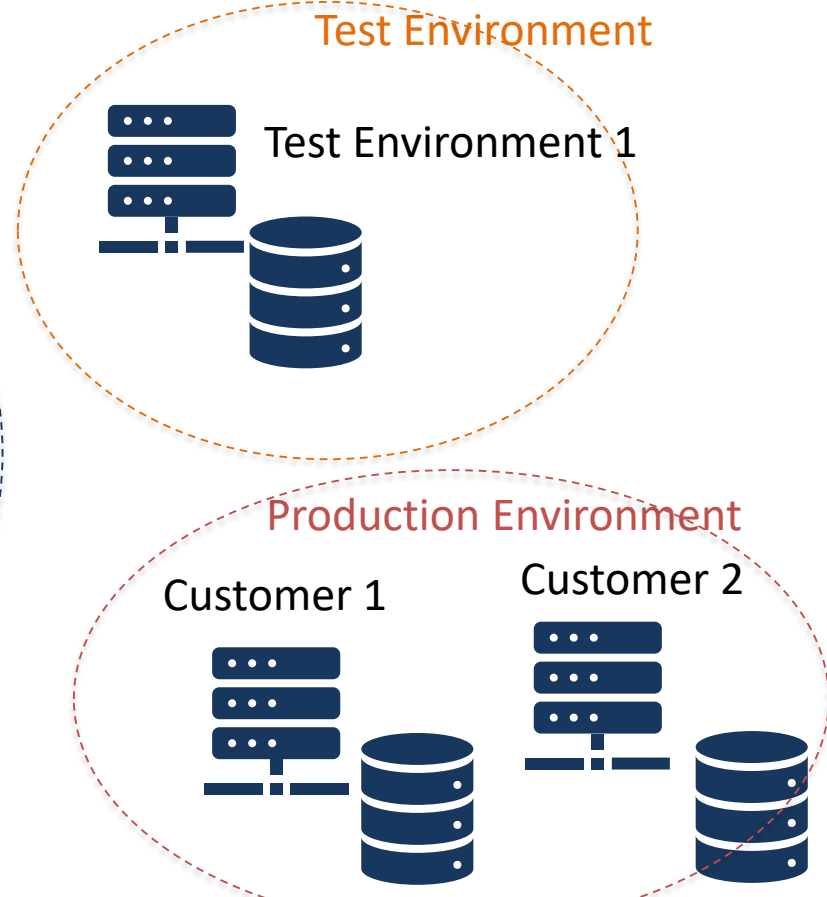
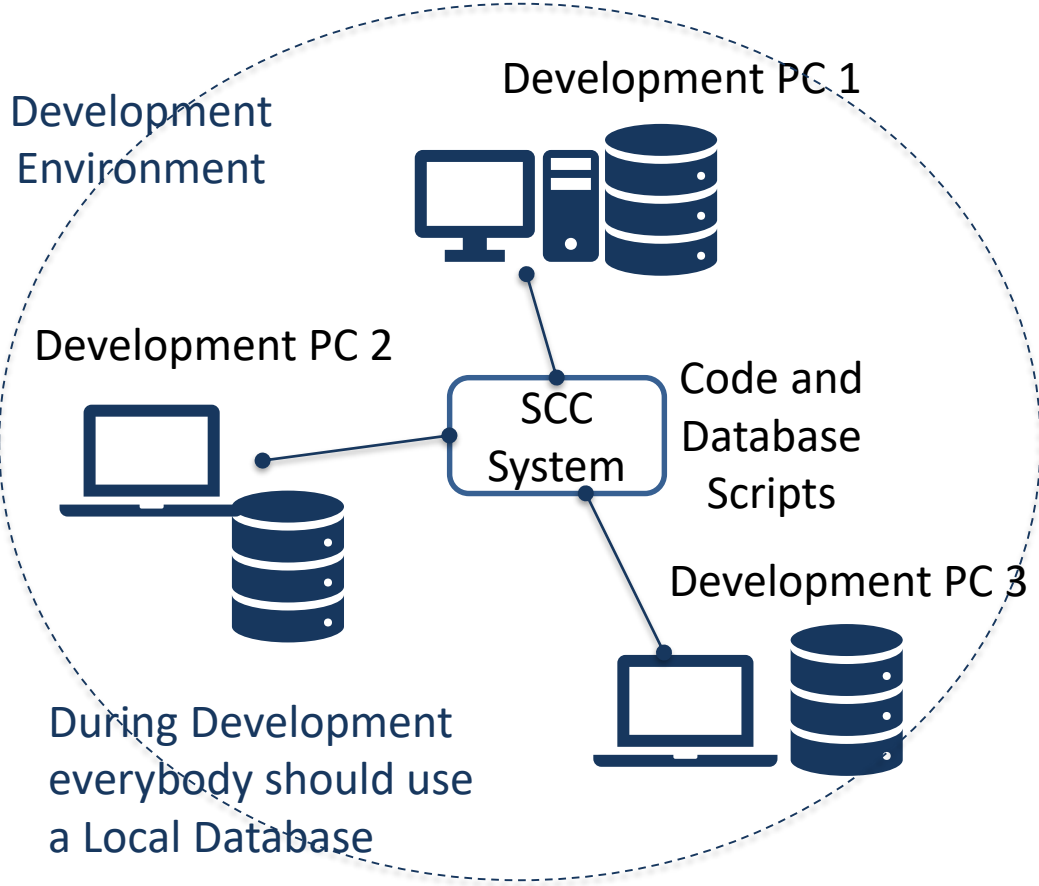
Database Scripts

Hans-Petter Halvorsen

Database Scripts

- Tables
- Views
- Stored Procedures
- Triggers
- Default Data

Why make Database Scripts?



Why make and use Database Scripts?

- It is easy to just add a new table or column directly into the database
 - then later you probably forgot that you did it
 - If everybody just add tables and columns or change data types when they feel for it, It will sooner or later become a total mess
- All changes in the database need to properly documented
- It makes it easy to install a new database or update an existing database – you just execute the script
- If there are multiple developers that have their local development database, they can just run the latest database script in order to update their database
- You may have customers running different versions of your software (and again will need to run different version of the database)
- Typically you want to include the Database Script into a setup to make it easy to install your software.
- Assume you have thousands of different customers that need to install your software on their local server.

Local Database

Why should each Developer have their own personal Database?

- They may work on different branches or different releases of the software that requires different databases
- You don't need internet access
- You may want to put lots of data into the database when developing and testing the software. This data may not be interesting or relevant for others than you
- You will be able to test the Database Scripts
- Etc.

Tables

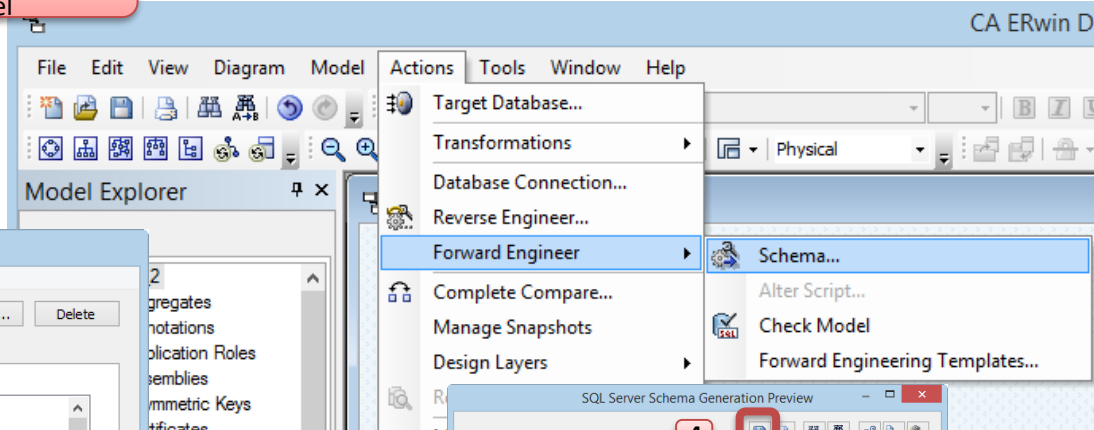
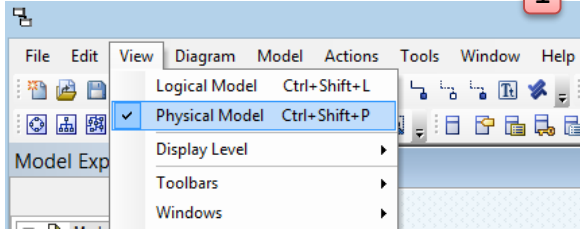
Create SQL Script using ERwin

2

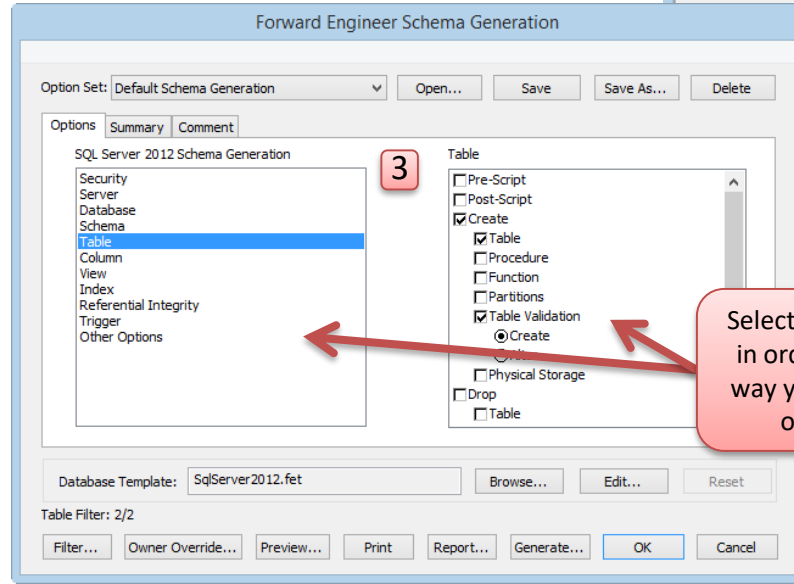
Select "Forward Engineering" and "Schema..."

1

Make sure you are using the Physical Model

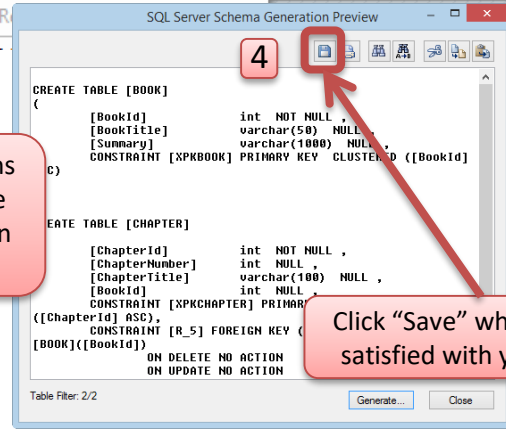


3



Select/Deselect different Options in order to make your script the way you want. Click "Preview" in order to see the results.

4



Click "Save" when you are satisfied with your Script

Example

```
if not exists (select * from dbo.sysobjects where id = object_id(N'[AUTHOR]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [AUTHOR]
(
    [AuthorId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [AuthorName] [varchar](50) NOT NULL UNIQUE,
)
GO
```

```
if not exists (select * from dbo.sysobjects where id = object_id(N'[PUBLISHER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [PUBLISHER]
(
    [PublisherId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [PublisherName] [varchar](50) NOT NULL UNIQUE,
)
GO
```

```
if not exists (select * from dbo.sysobjects where id = object_id(N'[CATEGORY]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [CATEGORY]
(
    [CategoryId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [CategoryName] [varchar](50) NOT NULL UNIQUE,
    [Description] [varchar](1000) NULL,
)
GO
```

```
if not exists (select * from dbo.sysobjects where id = object_id(N'[BOOK]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [BOOK]
(
    [BookId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [Title] [varchar](50) NOT NULL UNIQUE,
    [ISBN] [varchar](20) NOT NULL,
    [PublisherId] [int] NOT NULL FOREIGN KEY REFERENCES [PUBLISHER] ([PublisherId]),
    [AuthorId] [int] NOT NULL FOREIGN KEY REFERENCES [AUTHOR] ([AuthorId]),
    [CategoryId] [int] NOT NULL FOREIGN KEY REFERENCES [CATEGORY] ([CategoryId]),
)
GO
```



SQL Server Management Studio

The screenshot displays the Microsoft SQL Server Management Studio interface. The left pane shows the Object Explorer with the 'LIBRARY' database selected. The main window shows a SQL script with the following content:

```
if not exists (select * from dbo.sysobjects where id = object_id(N'[AUTHOR]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [AUTHOR]
(
    [AuthorId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [AuthorName] [varchar](50) NOT NULL UNIQUE,
    [Address] [varchar](50) NULL,
    [Phone] [varchar](50) NULL,
    [PostCode] [varchar](50) NULL,
    [PostAddress] [varchar](50) NULL,
)
GO

if not exists (select * from dbo.sysobjects where id = object_id(N'[PUBLISHER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [PUBLISHER]
(
    [PublisherId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [PublisherName] [varchar](50) NOT NULL UNIQUE,
    [Description] [varchar](1000) NULL,
    [Address] [varchar](50) NULL,
    [Phone] [varchar](50) NULL,
    [PostCode] [varchar](50) NULL,
    [PostAddress] [varchar](50) NULL,
    [EMail] [varchar](50) NULL,
)
GO

if not exists (select * from dbo.sysobjects where id = object_id(N'[CATEGORY]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [CATEGORY]
(
    [CategoryId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [CategoryName] [varchar](50) NOT NULL UNIQUE,
    [Description] [varchar](1000) NULL,
)
GO

if not exists (select * from dbo.sysobjects where id = object_id(N'[BOOK]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [BOOK]
(
    [BookId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [Title] [varchar](50) NOT NULL UNIQUE,
    [ISBN] [varchar](20) NOT NULL,
    [PublisherId] [int] NOT NULL FOREIGN KEY REFERENCES [PUBLISHER] ([PublisherId]),
    [AuthorId] [int] NOT NULL FOREIGN KEY REFERENCES [AUTHOR] ([AuthorId]),
    [CategoryId] [int] NOT NULL FOREIGN KEY REFERENCES [CATEGORY] ([CategoryId]),
    [Description] [varchar](1000) NULL,
    [Year] [date] NULL,
    [Edition] [int] NULL,
    [AverageRating] [float] NULL,
)
GO
```

The status bar at the bottom indicates 'Connected. (1/1)' and '0 rows'.

Execute the Table Script from SQL Server Management Studio

```
if not exists (select * from dbo.sysobjects where id = object_id(N'[CUSTOMER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE CUSTOMER
(
    CustomerId int PRIMARY KEY,
    CustomerNumber int NOT NULL UNIQUE,
    LastName varchar(50) NOT NULL,
    FirstName varchar(50) NOT NULL,
    AreaCode int NULL,
    Address varchar(50) NULL,
    Phone varchar(50) NULL,
)
GO
```

SQL Script Example that has been generated with ERwin but has been modified in SQL Server Management Studio for more robustness. The Script handles that tables may already exist, etc.

```
if exists(select * from dbo.syscolumns where id = object_id(N'[CUSTOMER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1 and name = 'CustomerId')
ALTER TABLE CUSTOMER ALTER COLUMN CustomerId int
Else
ALTER TABLE CUSTOMER ADD CustomerId int
GO
```

```
if exists(select * from dbo.syscolumns where id = object_id(N'[CUSTOMER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1 and name = 'CustomerNumber')
ALTER TABLE CUSTOMER ALTER COLUMN CustomerNumber int
Else
ALTER TABLE CUSTOMER ADD CustomerNumber int
GO
...
```

Check if Columns already exists
If Exists -> Modify
If not Exists -> Add

```
if not exists (select * from dbo.sysobjects where id = object_id(N'[CUSTOMER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
```

```
CREATE TABLE CUSTOMER
```

```
(  
    CustomerId int PRIMARY KEY,  
    CustomerNumber int NOT NULL UNIQUE,  
    LastName varchar(50) NOT NULL,  
    FirstName varchar(50) NOT NULL,  
    AreaCode int NULL,  
    Address varchar(50) NULL,  
    Phone varchar(50) NULL,  
)
```

```
GO
```

```
if exists(select * from dbo.syscolumns where id = object_id(N'[CUSTOMER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1 and name = 'CustomerId')
```

```
ALTER TABLE CUSTOMER ALTER COLUMN CustomerId int
```

```
Else
```

```
ALTER TABLE CUSTOMER ADD CustomerId int
```

```
GO
```

```
if exists(select * from dbo.syscolumns where id = object_id(N'[CUSTOMER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1 and name = 'CustomerNumber')
```

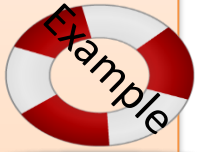
```
ALTER TABLE CUSTOMER ALTER COLUMN CustomerNumber int
```

```
Else
```

```
ALTER TABLE CUSTOMER ADD CustomerNumber int
```

```
GO
```

```
...
```



<https://www.halvorsen.blog>

Views

Hans-Petter Halvorsen

1

Creating Views using SQL code

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

This part is not necessary – but if you make any changes, you need to delete the old version before you can update it

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:

2

```
select * from CourseData
```

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

View Template

Copy to SQL Server Management Studio, save as a SQL File (.sql) as the same name as the View you are going to create. Store all your files on your hard drive.

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

CREATE VIEW <ViewName>
AS

SELECT
<TableName>.<ColumnName>,
<TableName>.<ColumnName>,
<TableName>.<ColumnName>,
<TableName>.<ColumnName>,
<TableName>.<ColumnName>

FROM
<TableName1>
INNER JOIN <TableName2> ON <TableName1>.<PrimKeyColumnName1> = <TableName2>.<PrimKeyColumnName2>
GO
```

Example

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

```
CREATE VIEW GetBookData
AS
```

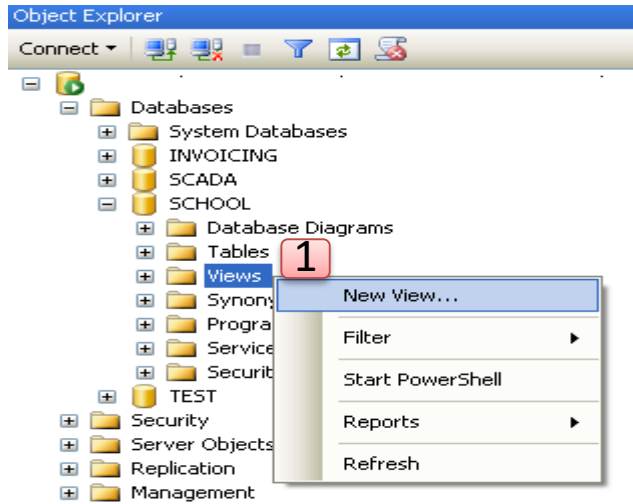
```
SELECT
BOOK.BookId,
BOOK.Title,
BOOK.ISBN,
PUBLISHER.PublisherName,
AUTHOR.AuthorName,
CATEGORY.CategoryName
```

```
FROM BOOK
INNER JOIN AUTHOR ON BOOK.AuthorId = AUTHOR.AuthorId
INNER JOIN PUBLISHER ON BOOK.PublisherId = PUBLISHER.PublisherId
INNER JOIN CATEGORY ON BOOK.CategoryId = CATEGORY.CategoryId
```

```
GO
```



Creating Views using the Editor



3

Select necessary columns

Graphical Interface where you can select columns you need

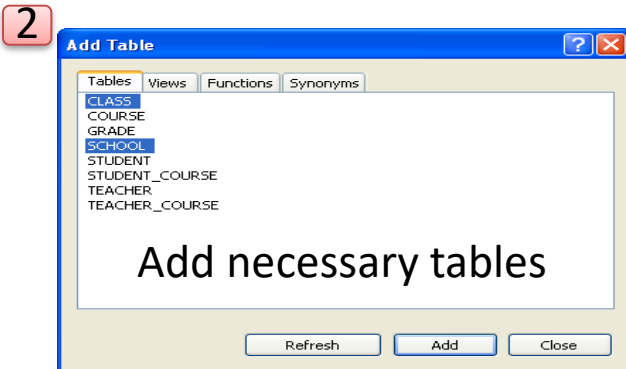
Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...	Or...	Or...
SchoolName		SCHOOL	<input checked="" type="checkbox"/>						
ClassName		CLASS	<input checked="" type="checkbox"/>						

```
SELECT dbo.SCHOOL.SchoolName, dbo.CLASS.ClassName  
FROM dbo.SCHOOL INNER JOIN  
      dbo.CLASS ON dbo.SCHOOL.SchoolId = dbo.CLASS.SchoolId
```

The Code is automatically generated

SchoolName	ClassName
TUC	SCE1
TUC	SCE2
TUC	PT1
TUC	PT2

Show the results



4

Copy the SQL Code and Create a New Script in the Management Studio

Stored Procedures

1 Create Stored Procedure:

Stored Procedure

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

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

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

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

This part is not necessary – but if you make any changes, you need to delete the old version before you can update it

Procedure Name

Input Arguments

Internal/Local Variables
Note! Each variable starts with @

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

2 Using the Stored Procedure:

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

Stored Procedure Template

```
IF EXISTS (SELECT name
           FROM   sysobjects
           WHERE  name = '<StoredProcedureName>'
           AND    type = 'P')
DROP PROCEDURE <StoredProcedureName>
```

```
GO
```

```
CREATE PROCEDURE <StoredProcedureName>
@<InputVariable1> <DataType>,
@<InputVariable2> <DataType>
AS
```

```
DECLARE
@<InternalVariable1> <DataType>,
@<InternalVariable2> <DataType>
```

```
select @<InternalVariable1> = <ColumnName> from <TableName> where <ColumnName> =
@<InputVariable1>
```

```
insert into <TableName> (<ColumnName1>, <ColumnName2>, ...) values (@<InternalVariable1>,
@<Inputvariable1>, ...)
GO
```

Copy to SQL Server Management Studio, save as a SQL File (.sql) as the same name as the SP you are going to create. Store all your files on your hard drive.

Example

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

CREATE PROCEDURE CreateBook

```
@Title varchar(50),
@Isbn varchar(20),
@PublisherName varchar(50),
@AuthorName varchar(50),
@CategoryName varchar(50)
AS
```

```
if not exists (select * from CATEGORY where CategoryName = @CategoryName)
INSERT INTO CATEGORY (CategoryName) VALUES (@CategoryName)
```

```
if not exists (select * from AUTHOR where AuthorName = @AuthorName)
INSERT INTO AUTHOR (AuthorName) VALUES (@AuthorName)
```

```
if not exists (select * from PUBLISHER where PublisherName = @PublisherName)
INSERT INTO PUBLISHER (PublisherName) VALUES (@PublisherName)
```

```
if not exists (select * from BOOK where Title = @Title)
INSERT INTO BOOK (Title, ISBN, PublisherId, AuthorId, CategoryId)
VALUES
(
@Title,
@ISBN,
(select PublisherId from PUBLISHER where PublisherName=@PublisherName),
(select AuthorId from AUTHOR where AuthorName=@AuthorName),
(select CategoryId from CATEGORY where CategoryName=@CategoryName)
)
```

```
GO
```



<https://www.halvorsen.blog>

Triggers

Hans-Petter Halvorsen

Triggers

A Trigger is executed when you insert, update or delete data in a Table specified in the Trigger.

Trigger Example:

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

This part is not necessary – but if you make any changes, you need to delete the old version before you can update it

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

Name of the Trigger

Specify which Table the Trigger shall work on

```
DECLARE
@StudentId int,
@AvgGrade float
```

Specify what kind of operations the Trigger shall act on

Internal/Local Variables

```
select @StudentId = StudentId from INSERTED
select @AvgGrade = AVG(Grade) from GRADE where StudentId = @StudentId
update STUDENT set TotalGrade = @AvgGrade where StudentId = @StudentId
GO
```

Inside the Trigger you can use ordinary SQL statements, create variables, etc.

SQL Code
(The “body”
of the Trigger)

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

Trigger Template

Copy to SQL Server Management Studio, save as a SQL File (.sql) as the same name as the Trigger you are going to create. Store all your files on your hard drive.

```
IF EXISTS (SELECT name
           FROM   sysobjects
           WHERE  name = '<TriggerName>'
           AND    type = 'TR')
    DROP PROCEDURE <TriggerName>
GO

CREATE TRIGGER <TriggerName>
FOR UPDATE, INSERT, DELETE --Delete the ones not needed
AS

DECLARE
@<InternalVariable1> <DataType>,
@<InternalVariable2> <DataType>

select @Variable1 = Column1 from INSERTED
select @Variable2 = AVG(Column2) from TABLE where Column1 = @Variable1
update TABLE set Column3= @Variabl2e where Column1= @Variable1

GO
```



Example

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

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

```
AS
DECLARE
@StudentId int,
@AvgGrade float
```

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

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

```
update STUDENT set TotalGrade = @AvgGrade where StudentId = @StudentId
```

```
GO
```


<https://www.halvorsen.blog>

Default Data Scripts

Hans-Petter Halvorsen

Default Data Script

- Typically we need to have some data in the Database, typically some information needed by our Software Program
- It could be e.g., some “Schools”, etc. that our Software System need to run properly
- All these “Default Data” can be entered in a Script



Example

```
--CATEGORY -----
INSERT INTO CATEGORY (CategoryName) VALUES ('Science')
GO
INSERT INTO CATEGORY (CategoryName) VALUES ('Programming')
GO
INSERT INTO CATEGORY (CategoryName) VALUES ('Novel')
GO

--AUTHOR -----
INSERT INTO AUTHOR (AuthorName) VALUES ('Knut Hamsun')
GO
INSERT INTO AUTHOR (AuthorName) VALUES ('Gilbert Strang')
GO
INSERT INTO AUTHOR (AuthorName) VALUES ('J.R.R Tolkien')
GO
INSERT INTO AUTHOR (AuthorName) VALUES ('Dorf Bishop')
GO

--PUBLISHER -----
INSERT INTO PUBLISHER (PublisherName) VALUES ('Prentice Hall')
GO
INSERT INTO PUBLISHER (PublisherName) VALUES ('Wiley')
GO
INSERT INTO PUBLISHER (PublisherName) VALUES ('McGraw-Hill')
GO

--BOOK -----
INSERT INTO BOOK (Title, ISBN, PublisherId, AuthorId, CategoryId)
VALUES
(
'Introduction to Linear Algebra',
'0-07-066781-0',
(select PublisherId from PUBLISHER where PublisherName='Prentice Hall'),
(select AuthorId from AUTHOR where AuthorName='Gilbert Strang'),
(select CategoryId from CATEGORY where CategoryName='Science')
)
GO

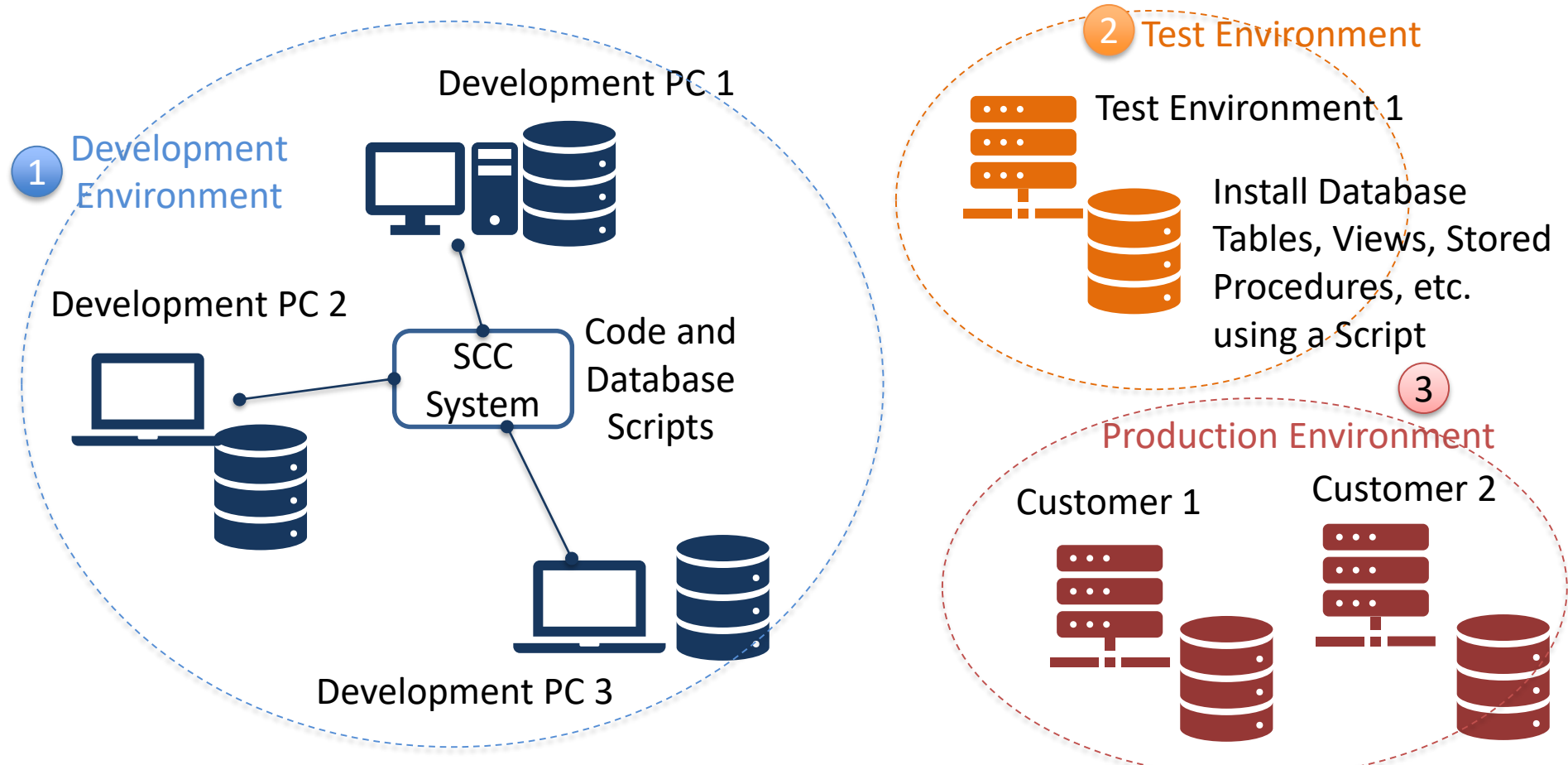
INSERT INTO BOOK (Title, ISBN, PublisherId, AuthorId, CategoryId)
VALUES
(
'Modern Control System',
'1-08-890781-0',
(select PublisherId from PUBLISHER where PublisherName='Wiley'),
(select AuthorId from AUTHOR where AuthorName='Dorf Bishop'),
(select CategoryId from CATEGORY where CategoryName='Programming')
)
GO
```

<https://www.halvorsen.blog>

Database Script Generator

Hans-Petter Halvorsen

Database Script Deployment

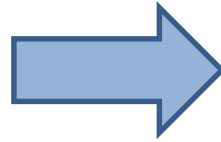


Database Script Generator

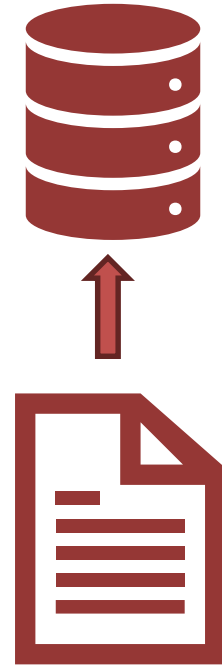
Development Environment



Test or Production Environment



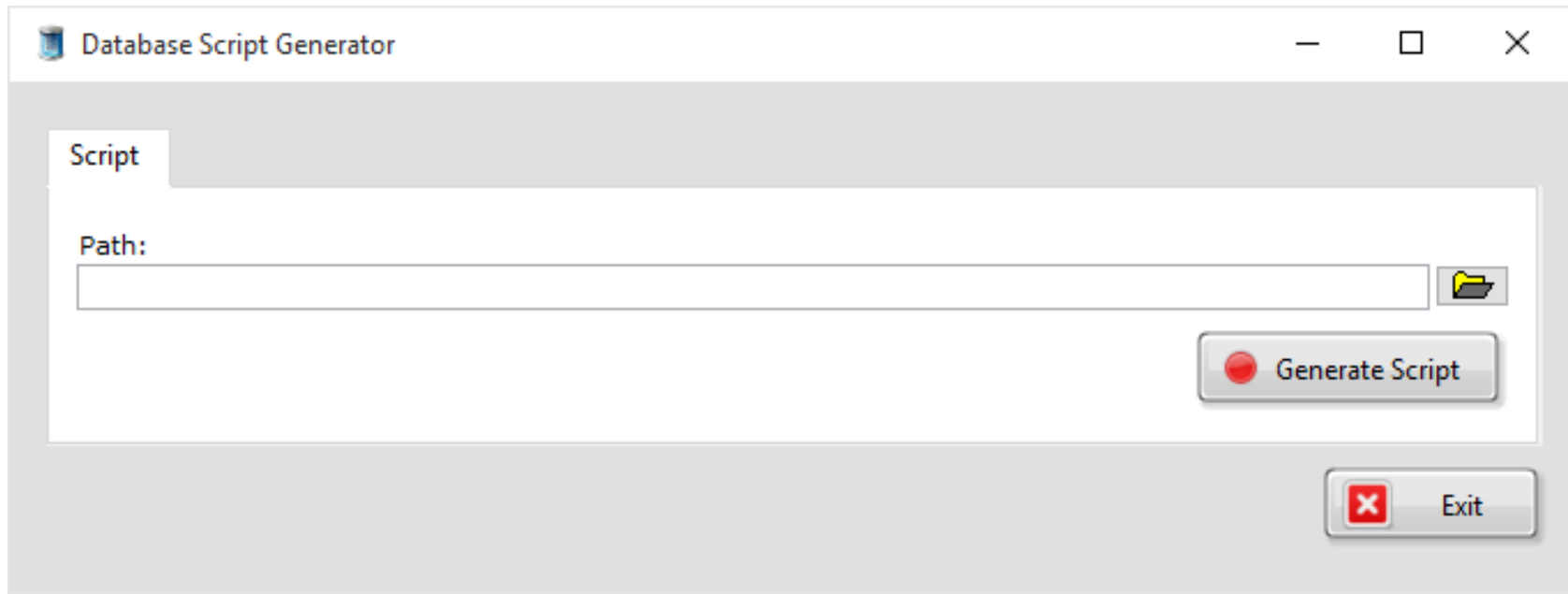
Create one Database Script which contains all Tables, Views, Stored Procedures, Triggers, etc.



Multiple Scripts, One or more Table Scripts, and Scripts for each View, Stored Procedure, Trigger, etc.

Database Script Generator

- Functions
- Scripts
- Stored Procedures
- Tables
- Triggers
- Views



The screenshot shows a window titled "Database Script Generator" with standard Windows window controls (minimize, maximize, close). The window contains a tab labeled "Script". Below the tab is a "Path:" label followed by a text input field and a folder selection icon. A "Generate Script" button with a red circular icon is positioned to the right of the path field. At the bottom right of the window is an "Exit" button with a red 'X' icon.

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

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

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

