

leren. durven. doen.



DATABANKEN

C# programmeur

INHOUD

- Variabelen
- PSM : stored procedures (SP) & stored functions

PMS: PERSISTENT STORED MODULES

C# programmeur

Persistent Stored Modules (PSM)

Oorspronkelijke SQL was geen volwaardige programmeertaal

- SQL/PSM, een volwaardige programmeertaal:
variabelen, constanten, datatypes, operatoren,
controlestructuren: if, case, while, for, ...
- procedures, functies
- exception handling

PSM = storedprocedures en storedfunctions

• Voorbeelden

–SQL Server: Transact SQL (t-SQL) of .NET-talen

–Oracle: PL/SQL of Java

–DB2: SQL PL (leunt aan bij SQL/PSM-standaard)of Java

–mySQL: eigen taal, leunt aan bij SQL/PSM-standaard

SQL als volwaardige taal

Stored Procedures

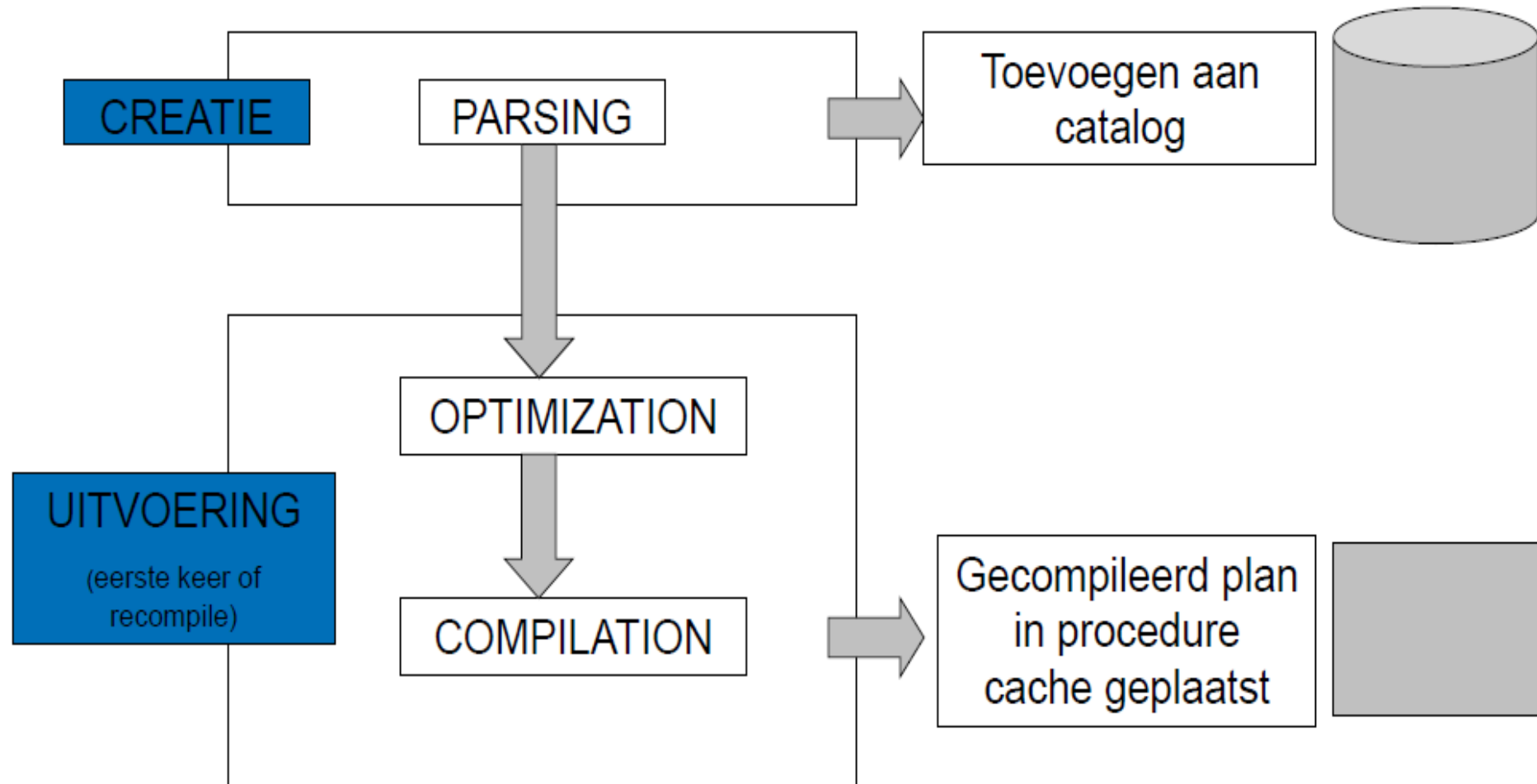
Stored Procedure

wat?

Definitie: een stored procedure is een benoemde verzameling SQL en control-of-flow opdrachten(programma) die opgeslagen wordt als een database object

- analoog aan procedures uit andere talen
- kan worden **aangeroept vanuit een programma, trigger of stored procedure**
- wordt opgeslagen in de **catalogus**
- accepteert in-en uitvoer **parameters**
- **retourneert status informatie** over de al dan niet correcte uitvoering van de stored procedure
- bevat taken die vaak worden uitgevoerd

Stored Procedure (SQL server)



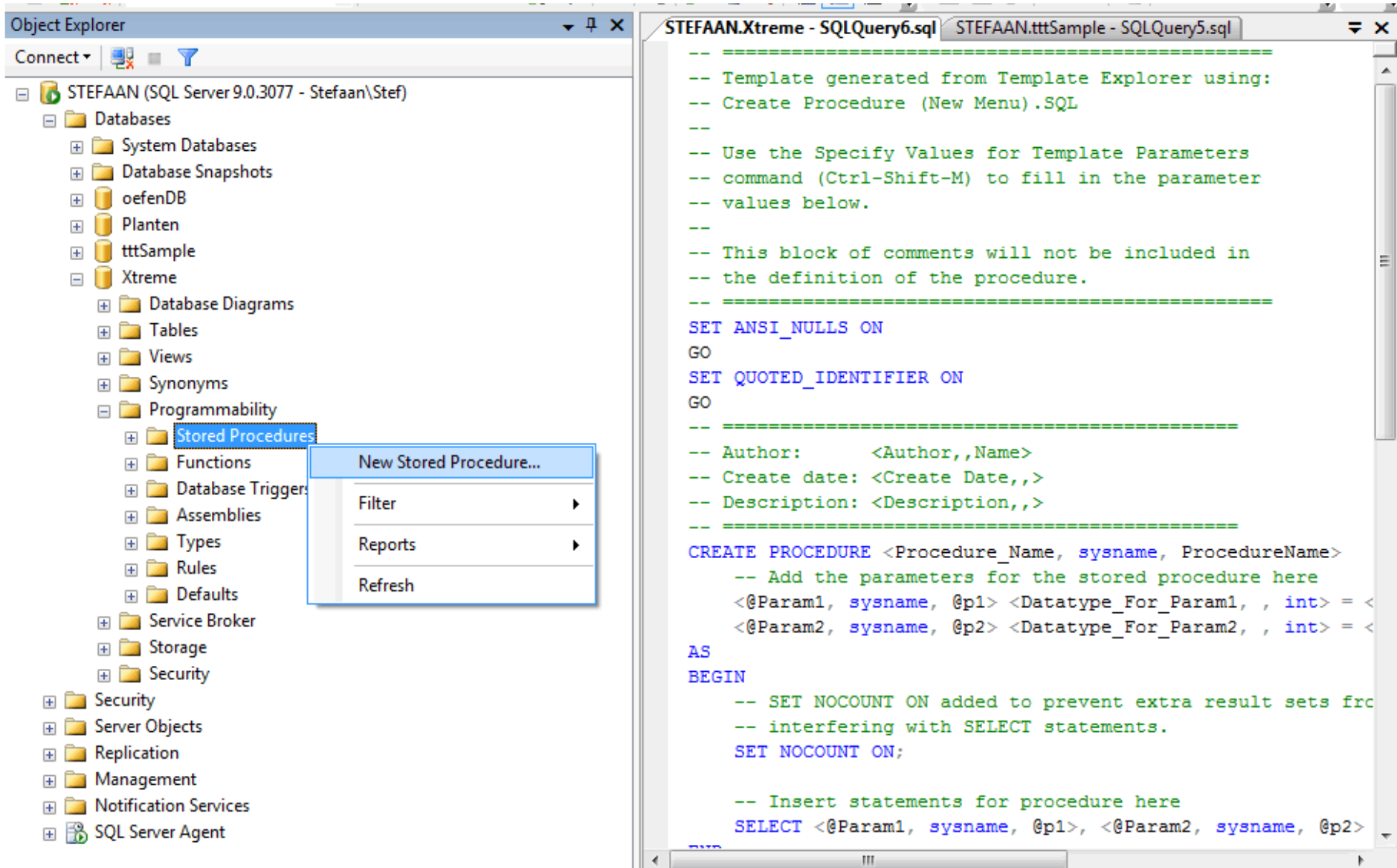
Creëren van Stored Procedure

```
CREATE PROCEDURE <proc_name> [parameter declaratie]  
AS  
<sql_statements>
```

- aanmaken db-object: via DDL instructie
- Voorbeeld:

```
CREATE PROCEDURE uspOrdersSelectAll  
AS  
select * from orders
```


Creëren van Stored Procedure via SQL Server Management Studio



The screenshot displays the SQL Server Management Studio (SSMS) interface. On the left, the Object Explorer shows the server 'STEFAAN (SQL Server 9.0.3077 - Stefaan\Stef)'. The 'Databases' folder is expanded, and the 'Programmability' folder is selected, showing a context menu with options: 'New Stored Procedure...', 'Filter', 'Reports', and 'Refresh'. The 'New Stored Procedure...' option is highlighted. On the right, the SQL query window 'STEFAAN.Xtreme - SQLQuery6.sql' contains a template for creating a stored procedure. The template includes comments for author, create date, and description, followed by the SQL code for creating the procedure with parameters and a SELECT statement.

Object Explorer

Connect

STEFAAN (SQL Server 9.0.3077 - Stefaan\Stef)

- Databases
 - System Databases
 - Database Snapshots
 - oefenDB
 - Planten
 - tttSample
 - Xtreme
 - Database Diagrams
 - Tables
 - Views
 - Synonyms
 - Programmability
 - New Stored Procedure...**
 - Filter
 - Reports
 - Refresh
 - Functions
 - Database Triggers
 - Assemblies
 - Types
 - Rules
 - Defaults
 - Service Broker
 - Storage
 - Security
- Security
- Server Objects
- Replication
- Management
- Notification Services
- SQL Server Agent

STEFAAN.Xtreme - SQLQuery6.sql

```
-- =====
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- =====

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- =====
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =====

CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
    -- Add the parameters for the stored procedure here
    <@Param1, sysname, @p1> <Datatype_For_Param1, , int> = <
    <@Param2, sysname, @p2> <Datatype_For_Param2, , int> = <

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
```

Aanroepen van Stored Procedure

```
EXECUTE <proc_name> [parameters]
```

EXEC(UTE) uspOrdersSelectAll

- **bij eerste uitvoering**
 - compilatie en optimalisatie
- **Hercompilatie forceren**
 - wenselijk bij wijzigingen aan structuur databank

execute uspOrdersSelectAll **with recompile**

execute sp_recompile uspOrdersSelectAll

De return waarde van een SP

- bij uitvoering keert een SP een **return waarde** terug
 - deze waarde is een **int**
 - de default return waarde is 0
- **return** statement
 - uitvoering van de SP wordt gestopt
 - laat toe om de return waarde te bepalen

De return waarde van een Stored Procedure

```
CREATE PROCEDURE usp_OrdersSelectAllAS  
select * fromorders  
return @@ROWCOUNT
```

Voorbeeld van gebruik return waarde:

```
DECLARE @returnCode int  
EXEC @returnCode = usp_OrdersSelectAll  
PRINT 'Er zijn ' + str(@returnCode) + '  
records.'
```

Voorbeeld gebruik van @@IDENTITY in stored procedure

@@IDENTITY geeft de laatst gegenereerde waarde van een auto-increment kolom

```
USE [xtreme5]
GO
CREATE PROCEDURE usp_Customer_Insert
AS
INSERT INTO Customer CustomerName)
VALUES('Syntra West')

SELECT customerID= @@IDENTITY

GO
EXEC usp_Customer_Insert
```

Oefeningen creëren van Stored Procedure: database xtreme5

1. Open SSMS en open een nieuw Query window op de database **xtreme5**
2. Maak een stored procedure met naam ***GetAllProductClasses*** die productClassID en ProductClassName van alle product klassen ophaalt, gesorteerd op productClassName
3. Test deze stored procedure door hem uit te voeren via:
EXEC GetAllProductClasses

Wijzigen en verwijderen van Stored Procedure

```
ALTER PROCEDURE <proc_name> [parameter declaratie]  
AS  
<sql_statements>
```

```
ALTER PROCEDURE uspOrdersSelectAll  
AS  
SELECT * FROM orders
```

```
DROP PROCEDURE <proc_name>
```

```
DROP uspOrdersSelectAll
```

Stored Procedure met parameters

soorten parameters:

- via een **input** parameter geef je een waarde door aan de SP
- via een **output** parameter geef je *eventueel* een waarde door aan de SP en krijg je een waarde terug van de SP

Voorbeeld:

```
CREATE PROCEDURE
```

```
usp_OrdersSelectAllForCustomer
```

```
@customerID nchar(5),
```

```
@count int OUTPUT
```

```
AS
```

```
SELECT @count = count(*)
```

```
FROM orders
```

```
WHERE customerID= @customerID
```


Oefeningen creëren van Stored Procedure: database xtreme5

1. Maak een stored procedure met naam **GetOrdersForCustomer** die alle orders van een op te geven klant (tabel Orders) ophaalt via de CustomerID.

Test deze stored procedure via

EXEC GetOrdersForCustomer 64

2. Maak een stored procedure met naam **NewProductClass** voor het toevoegen van een productClass. De naam van de nieuwe Productclass is een input parameter

Retourneer de gegenereerde nieuwe waarde van ProductClassID (auto-increment kolom). Test deze stored procedure en kijk na of de returnwaarde juist wordt teruggegeven

3. Pas NewProductClass aan zodat @@IDENTITY met RETURN Instructie wordt teruggegeven

4. Pas NewProductClass aan zodat @@IDENTITY wordt teruggegeven in een OUTPUT parameter

Stored Procedure met optionele parameters

@CustomerID is
een optionele
parameter

```
CREATE PROCEDURE usp_OrdersSelectAllForCustomer  
@customerID nchar(5) = 'ALFKI'  
AS  
SELECT count(*) AS numberOfOrders  
FROM orders WHERE customerID= @customerID
```

Aanroepen van SP met optionele parameters

aanroepen van de SP

- voorzie steeds keyword OUTPUT voor output parameters
- 2 manieren om actuele parameters door te geven
 1. gebruik formele parameter naam (*volgorde onbelangrijk*)
 2. Positioneel

Voorbeeld1 aanroepen stored procedure

usp_OrdersSelectAllForCustomer:

```
DECLARE @aantal int
EXECUTE usp_OrdersSelectAllForCustomer
@customerID= 'ALFKI',
@count= @aantal OUTPUT
PRINT @aantal
```

Voorbeeld 2 aanroepen stored procedure

usp_OrdersSelectAllForCustomer:

```
DECLARE @aantalint
EXEC usp_OrdersSelectAllForCustomer 'ALFKI', @aantal OUTPUT
PRINT @aantal
```

Oefeningen aanpassen van Stored Procedure & optionele param:

1. Pas de stored procedure met naam ***GetOrdersForCustomer*** aan, zodat input parameter CustomerID optioneel is en default waarde 1 is
Test deze stored procedure via
EXEC GetOrdersForCustomer

Oefening verwijderen van Stored Procedure:

Verwijder de stored procedure met naam ***GetOrdersForCustomer***

Error handling

@@error is een systeemfunctie die het foutnummer bevat van de laatst uitgevoerde opdracht
de waarde 0 wijst op succesvolle uitvoering

```
CREATE PROCEDURE usp_ProductsInsert
@productName varchar(40),
@categoryID int,
@unitprice money
AS
INSERT INTO products(productname, categoryID, unitprice)
VALUES (@productname, @categoryID, @unitprice)
IF @@error = 515
    PRINT 'ERROR! productnameis NULL.'
ELSE IF @@error = 547
    PRINT 'ERROR! CategoryIDis not in CATEGORY table.'
ELSE PRINT 'ERROR! Unable to add new product.'
RETURN @@error
```

Error handling

alle foutboodschappen zitten in de systeemtabel **sysmessages**

```
SELECT * FROM master.dbo.sysmessages  
WHERE error = @@ERROR
```

eigen fouten genereren kan via **raiserror**
raiserror(msg, severity, state)

- msg-de foutboodschap
- severity-waarde tussen 0 en 18
- state -waarde tussen 1 en 127

voorbeelden andere systeemfunctie:

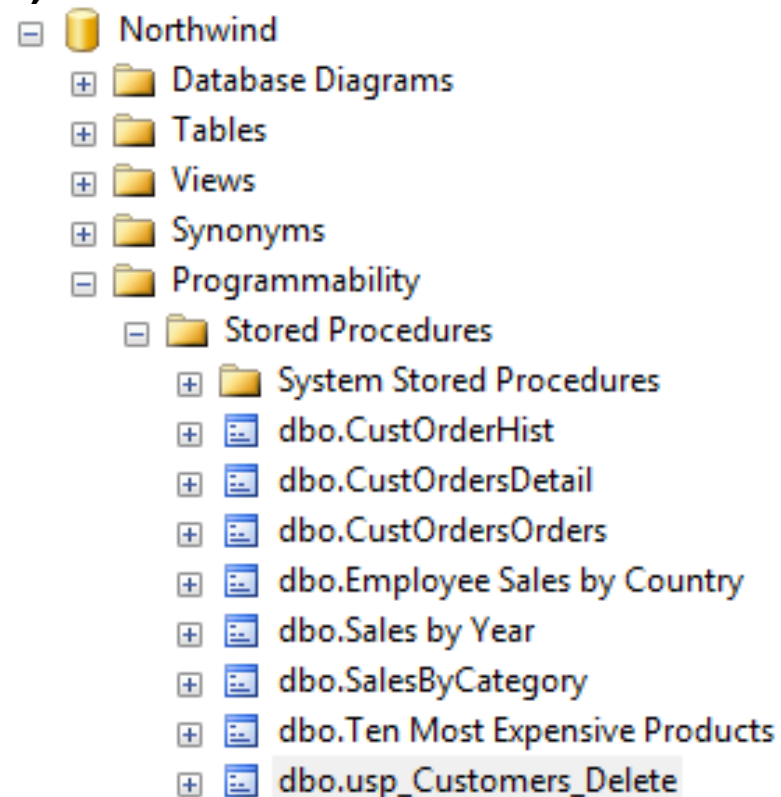
@@rowcount

=aantal aangepaste/geselecteerde rijen door de laatst uitgevoerde instructie

Stored Procedure : Voorbeeld RAISERROR

Voorbeeld in Northwind database (MS SQL Server):

```
CREATE procedure usp_Customers_Delete
@custno nchar(5) = NULL
AS
IF @custno IS NULL
BEGIN
RAISERROR('customerID is NULL', 10, 1)
RETURN
END
IF NOT EXISTS (SELECT * FROM customers
WHERE customerid = @custno)
BEGIN
RAISERROR ('Klant bestaat niet', 10, 1)
RETURN
END
IF EXISTS (SELECT * FROM orders
WHERE customerid = @custno)
BEGIN
RAISERROR ('Klant heeft orders', 10, 1)
RETURN
END
DELETE FROM customers WHERE customerid = @custno
```



Waarom PSM gebruiken ?

PSM vs. 3GL (Java, .NET, C++, Cobol...)
(SQL Server, ook eerdere Oracle-versies):
query-optimalisatie en execution plan caching & reuse, vooral bij PSM

- **Vroeger:** SQL uitvoeren via PSM was performanter
- **Nu:** +/-zelfde optimalisatie, ongeacht hoe query aankomt bij databank
- toch wordt performantievaak nog (ten onrechte) als argument pro PSM gebruikt.

PSM: voordelen

- **code modularisatie**

reduceren redundante code: veel-gebruikte query's in SP en hergebruiken in 3GL

- minder onderhoud bij schema-wijzigingen
- vaak voor CRUD-operaties

- **security**

- rechtstreekse query's op tabellen uitsluiten
- via SP's vastleggen wat kan en wat niet
- vermijd SQL-injection door gebruik input-parameters

- **centrale administratie** van (delen van)
DB-code

PSM-nadelen

Beperkte schaalbaarheid: business logica en db-verwerking op zelfde server, kan tot bottle-necks leiden.

Vendor lock-in:

- **syntax = geen standaard:** porteren van bijv. MS SQL Server naar Oracle zeer moeilijk
- maar portabiliteit heeft ook zijn prijs (vb. built-in functies dan niet gebruiken)

Twee programmeertalen:

1. JAVA/.NET/.....
 2. SP / UDF
- **Twee debugomgevingen**
 - **SP/UDF: beperkte OO-ondersteuning**

Vuistregels

- vermijd PSM voor grotere business logica
- gebruik PSM vooral voor technische zaken:
 - logging/auditing/validatie
- maak keuze portabiliteit / vendor lock-in in overleg met
 - business
 - corporate IT policies

VRAGEN?

REFERENTIES

<https://www.w3schools.com/sql>