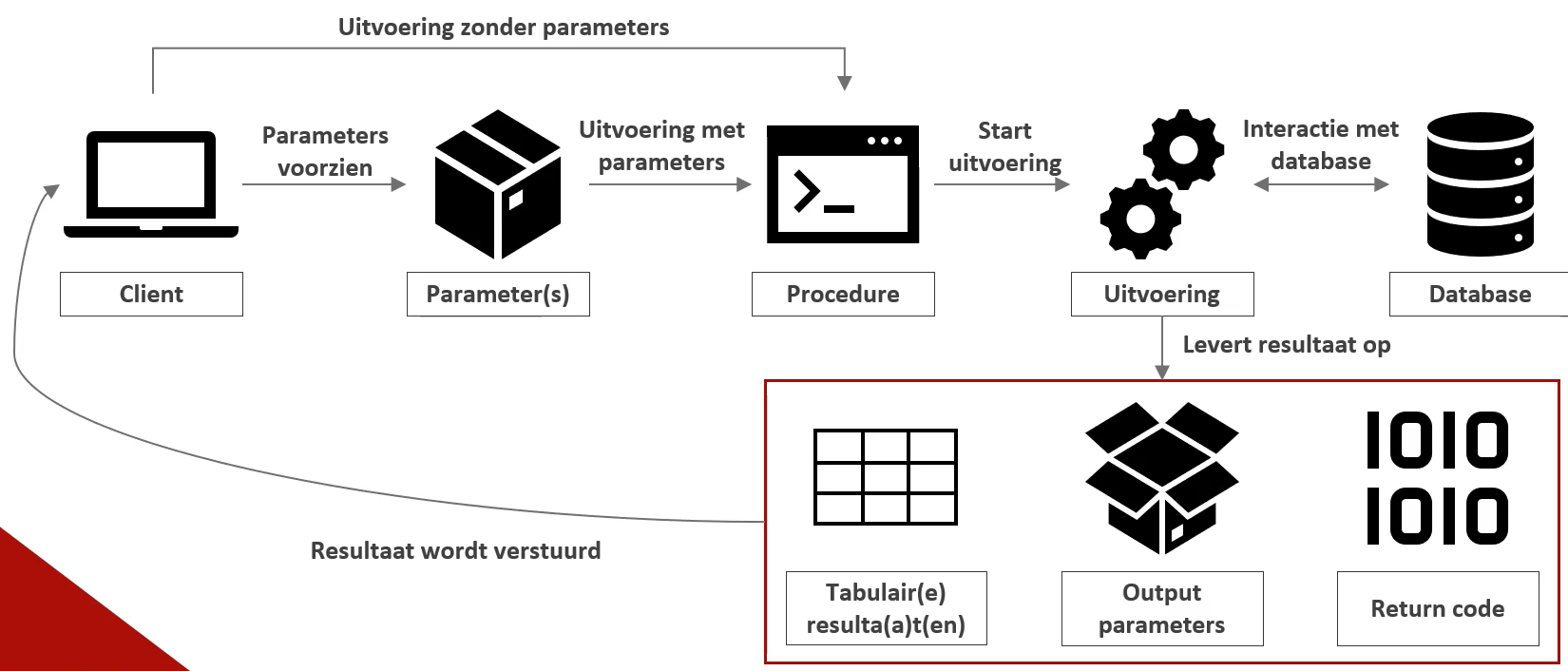
**Deel 8 - SQL procedures en functions**

**SQL Stored Procedures**

**Concept**

* Aantal instructies die bij elkaar horen
* Naast standaard SQL ook mogelijkheid tot programmatie
* Bestaat uit
  + Procedurenaam
  + In-en out parameters
  + Body (instructies)
  + Return waarde
  + Resultaat
* Uitvoerbaar maar niet benaderbaar als (virtuele) tabel
* Uitvoeren mbv exec instructie (call instructie in MySQL)



* Voordelen
  + Betere performantie owv execution plan dat wordt bewaard bij bewaren procedure
  + Security: vermijden van SQL injection
  + Onafhankelijk van de programmeertaal
* Nadelen
  + Introduceren van “business logica” in de datalaag

**Creatie**

CREATE OR ALTER { PROC | PROCEDURE }  
 [schema\_name.] procedure\_name

[ { @parameter data\_type }

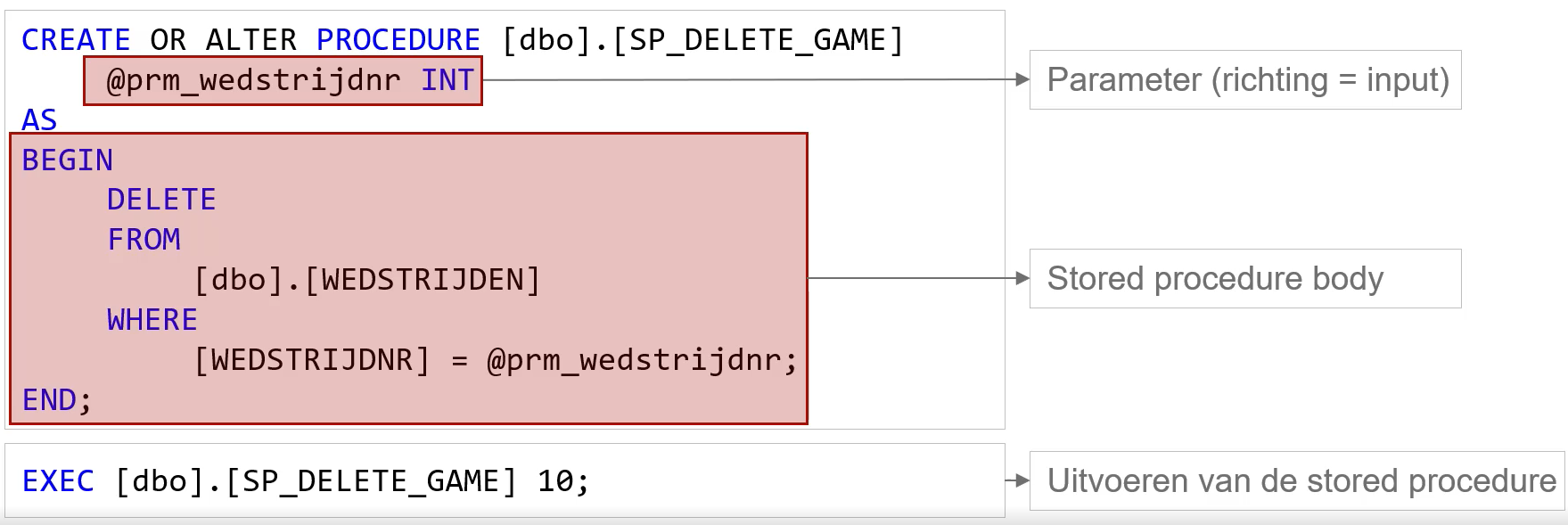
[ = default ] [ OUT | OUTPUT | [READONLY]

] [ ,...n ]

AS { [ BEGIN ] sql\_statement [;] [ ...n ] [ END ] }

[;]

**Voorbeelden**

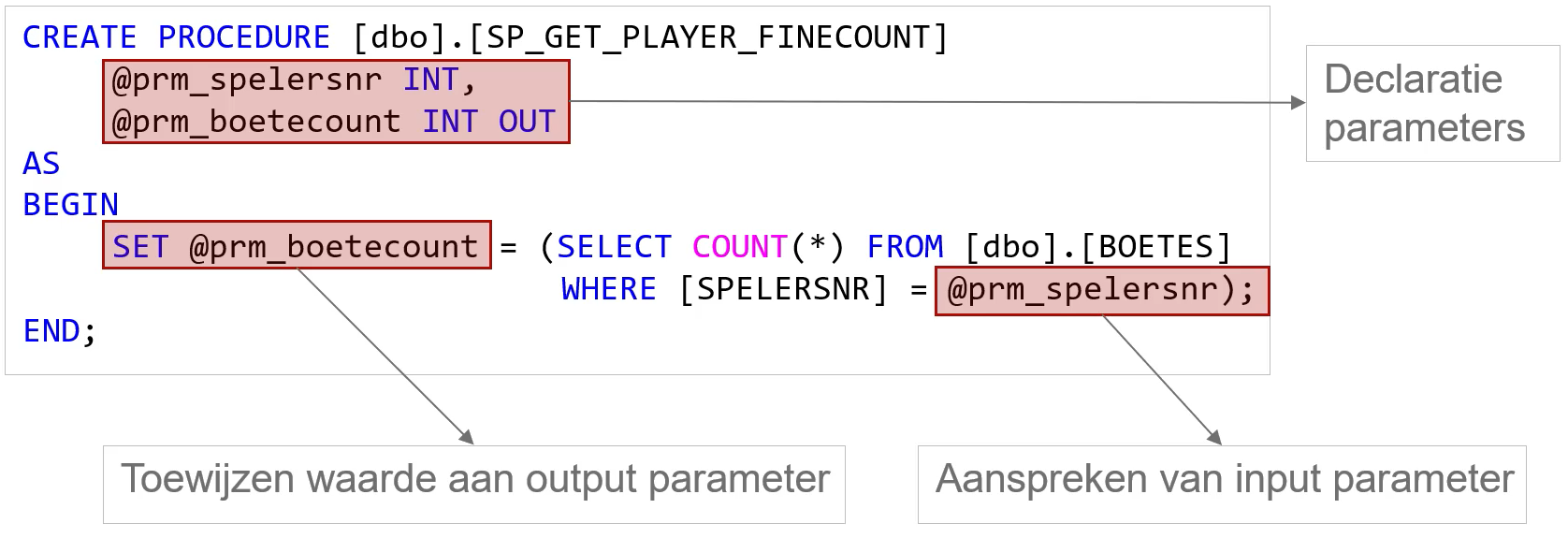


**Aandachtspunten**

* Wees consistent en duidelijk qua naamgeving!
* Gebruik een prefix bv. “SP\_”
* Geef altijd de actie aan van de stored procedure
  + GET
  + DELETE
  + INSERT
  + UPDATE
  + SAVE
* Eindig met iets dat duidelijk beschrijft waarvoor de procedure dient
  + Bv. SP\_SAVE\_PLAYER, SP\_DELETE\_PLAYER,...

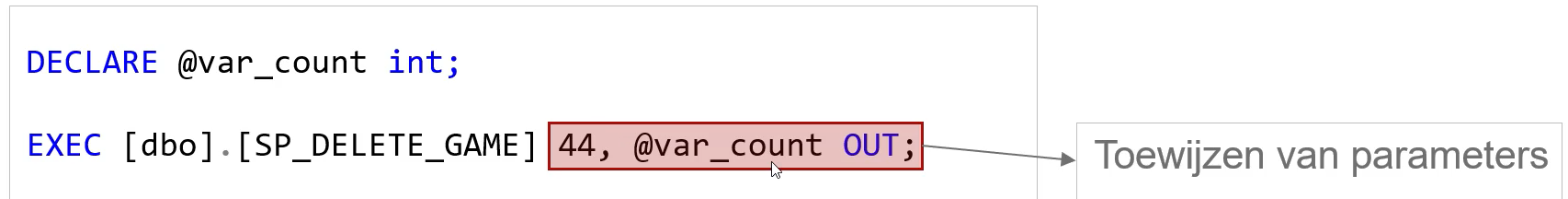
**Parameters**

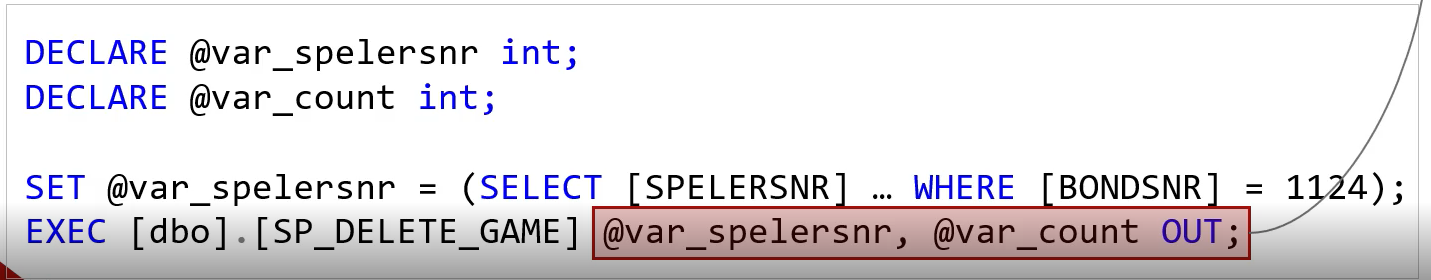
* Container waarlangs waarden kunnen meegegeven en/of uitgelezen worden
* Hebben een datatype dat ingesteld moet worden
* Waarden kunnen gebruikt worden binnen stored procedure
* Waarden kunnen opgevuld en teruggeven worden (output parameters)
* Te gebruiken om procedure dynamisch te maken
* **Input parameters**
  + Om waarden naar de procedure te brengen
  + Standaard van toepassing
* **Output parameters**
  + Om waarden vanuit de procedure naar buiten te brengen
  + Variabele declareren buiten procedure (met DECLARE keyword)
  + Bij toewijzen variabele aan procedure ook output keyword voorzien
  + TOewijzen waarde aan parameter (en variabelen) met SET keyword
* (Return code: standaard 0 maar kan gevuld worden vanuit procedure)



**Vb:**

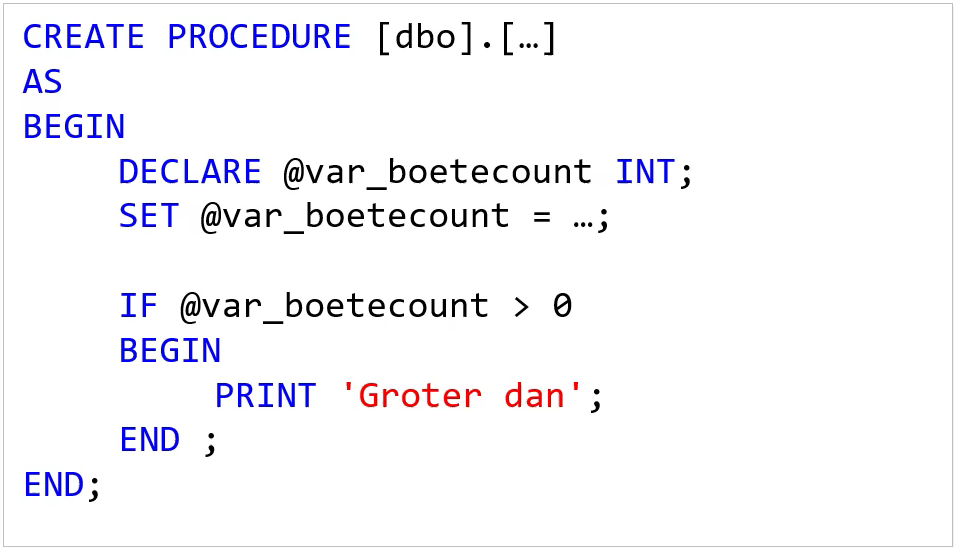
* Uitvoeren stored procedure met hardcoded waarde



* Uitvoeren stored procedure gebruik makend van variabele

**Body**

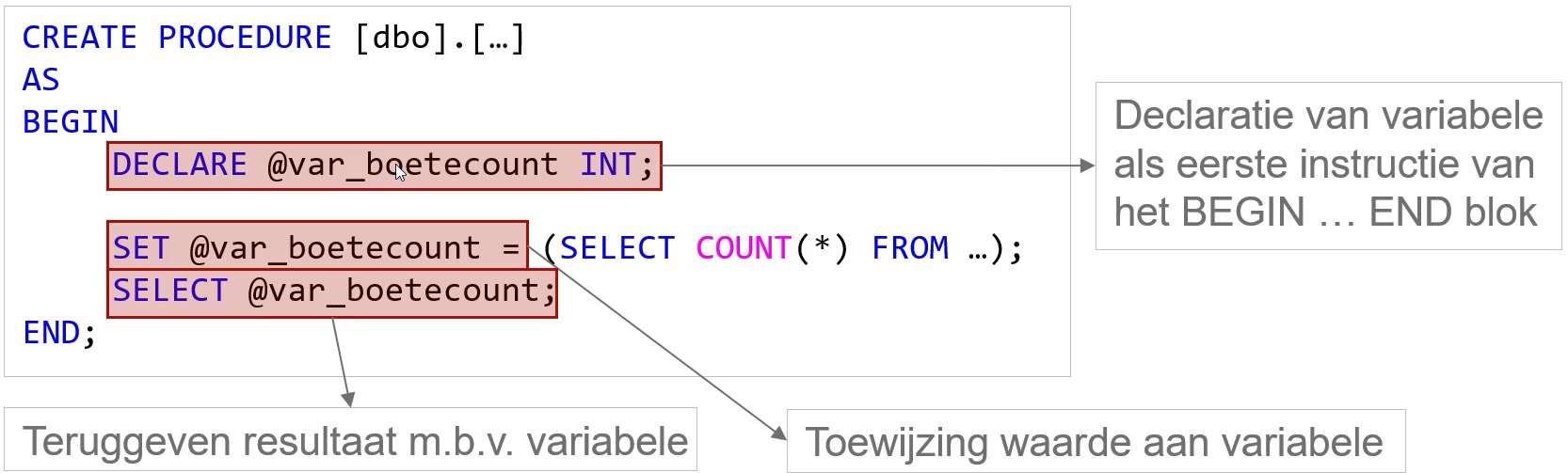
* Effectieve uitwerking van de logica
* Kan bestaan uit
  + SQL
  + Controlestructuren
  + Iteraties
  + Cursor
* TUssen BEGIN en END keyword plaatsen
* Meerdere BEGIN en END blokken mogelijk



**Vb:**

**Variabelen**

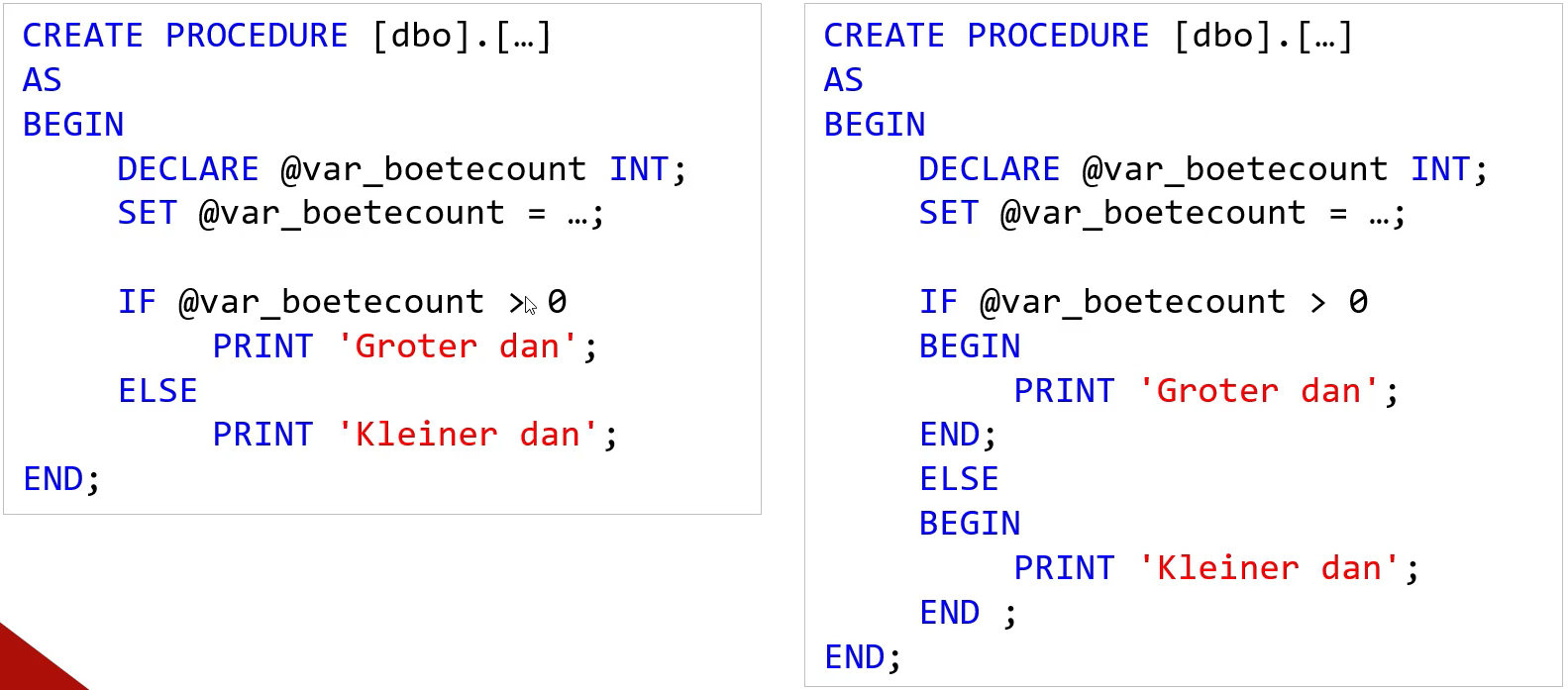
* Altijd als eerste instructie in het BEGIN … END blok
* Definiëren met een DECLARE keyword
* Naam van variabele vooraf laten gaan door @
* Datatype opgeven
* Niet toegelaten datatypes zijn:
  + TEXT
  + NTEXT
  + Image datatypes
* Waarde toekennen met behulp van het SET keyword



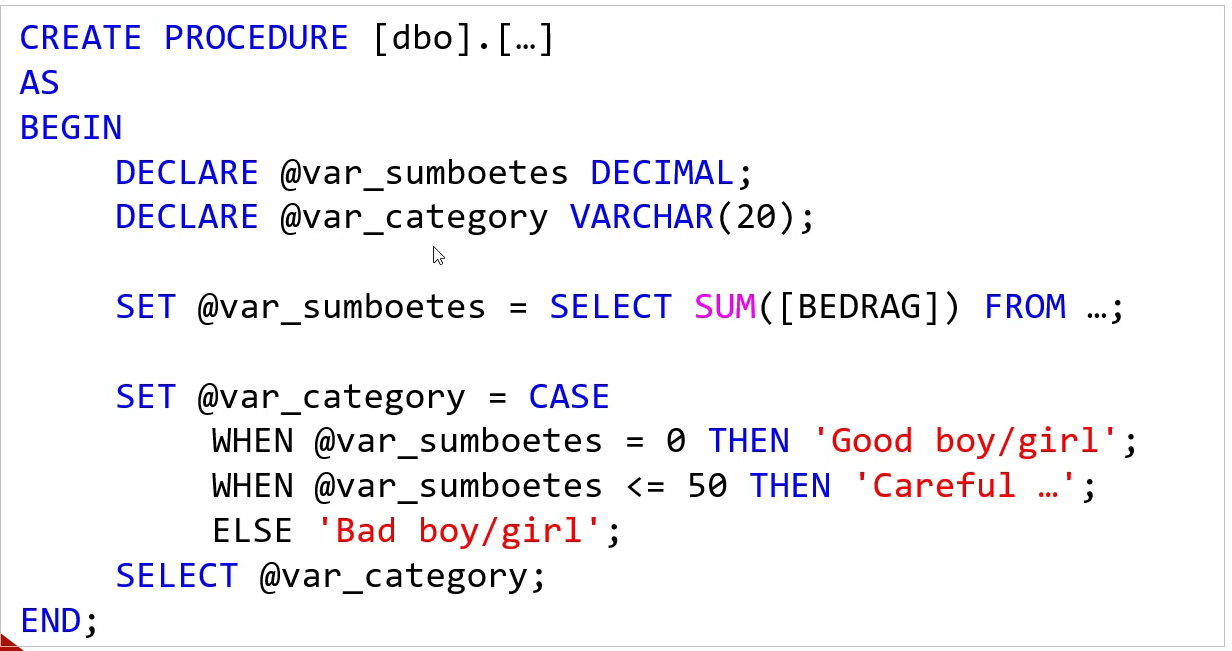
**Vb:**

**Controlestructuren**

* De **IF** controlestructuur
  + Enkel IF en ELSE keywords
  + Nesting mogelijk
  + Gebruik BEGIN … END wanneer meerdere instructies nodig zijn in een tak



* De **CASE** controlestructuur
  + Niet bruikbaar in stored procedures als flow control (in SQL Server)
  + Wel toepasbaar bij toekennen waarde aan variabele/parameter

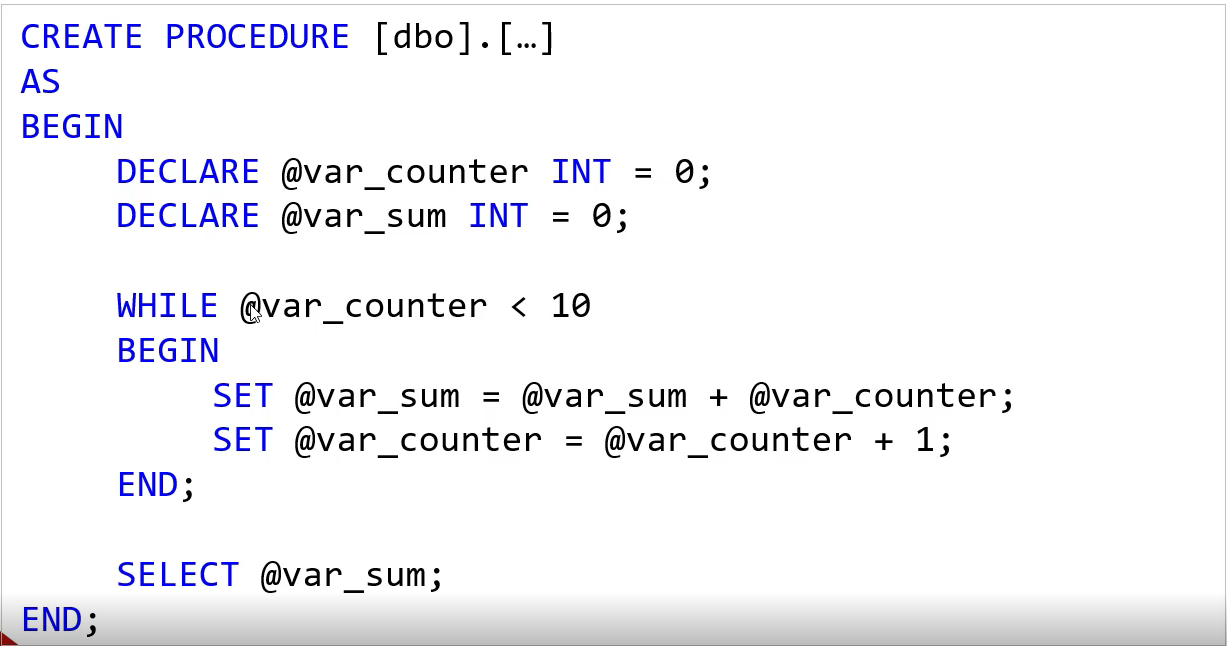


**Vb:**

**Iteraties**

* De **WHILE** iteratie
  + Voorwaarde die resulteert in een waar of onwaar
  + Instructies worden doorlopen zolang aan de voorwaarde is voldaan
  + Uitvoering van instructies binnenin de iteratie kan gestuurd worden met
    - BREAK: Beëindigd de (binnenste) iteratie
    - CONTINUE: Slaat instructies na CONTINUE over en herstart de iteratie

(MySQL kent ook nog andere iteraties zoals REPEAT en LOOP)



**Vb:**

**Foutafhandeling**

* SQL Server genereert een exception wanneer iets fout gaat
* Bevat specifieke informatie omtrent de fout die optrad
* Elke exception omvat een aantal eigenschappen
  + Error number => Identificatie nummer vd exception
  + Error line => Lijn in code waarop de exception optrad
  + Error message => Toelichting omtrent de exception die optrad
  + Severity => Categorie van “zwaarte” waaronder de exception valt
  + State => Nummer om te helpen bepalen waar de fout optrad
* Exceptions opvangen in BEGIN TRY … END TRY blok
* Exceptions verwerken in BEGIN CATCH … END CATCH blok
* Beiden moeten uitgewerkt worden
* Exception die wordt opgevangen wordt niet meer doorgegeven naar boven
* In geval dit nodig is kan dit door
  + Exception opnieuw throwen met de THROW instructie
* Informatie van exception opvragen met behulp van volgende functies
  + ERROR\_NUMBER()
  + ERROR\_SEVERITY()
  + ...
* Enkel bruikbaar binnen het CATCH blok
* Foutafhandeling gebeurt in veel gevallen in aanroepende code van een app
* Voorbeeldscenario: insert van relationele data waarbij
  + Een expliciete transactie wordt opgestart
  + Een child record wordt ingevoegd met een foreign key waarde waarvoor geen parent bestaat
  + Fout opvangen en rollback doen van transactie

**Cursor**

* Een object dat de mogelijkheid geeft om een resultaatset rij per rij te doorlopen
* Voorbeeld: hangkalender met een vakje dat je kan verschuiven

**Bekijk powerpoint voor voorbeeld werking van CURSOR**

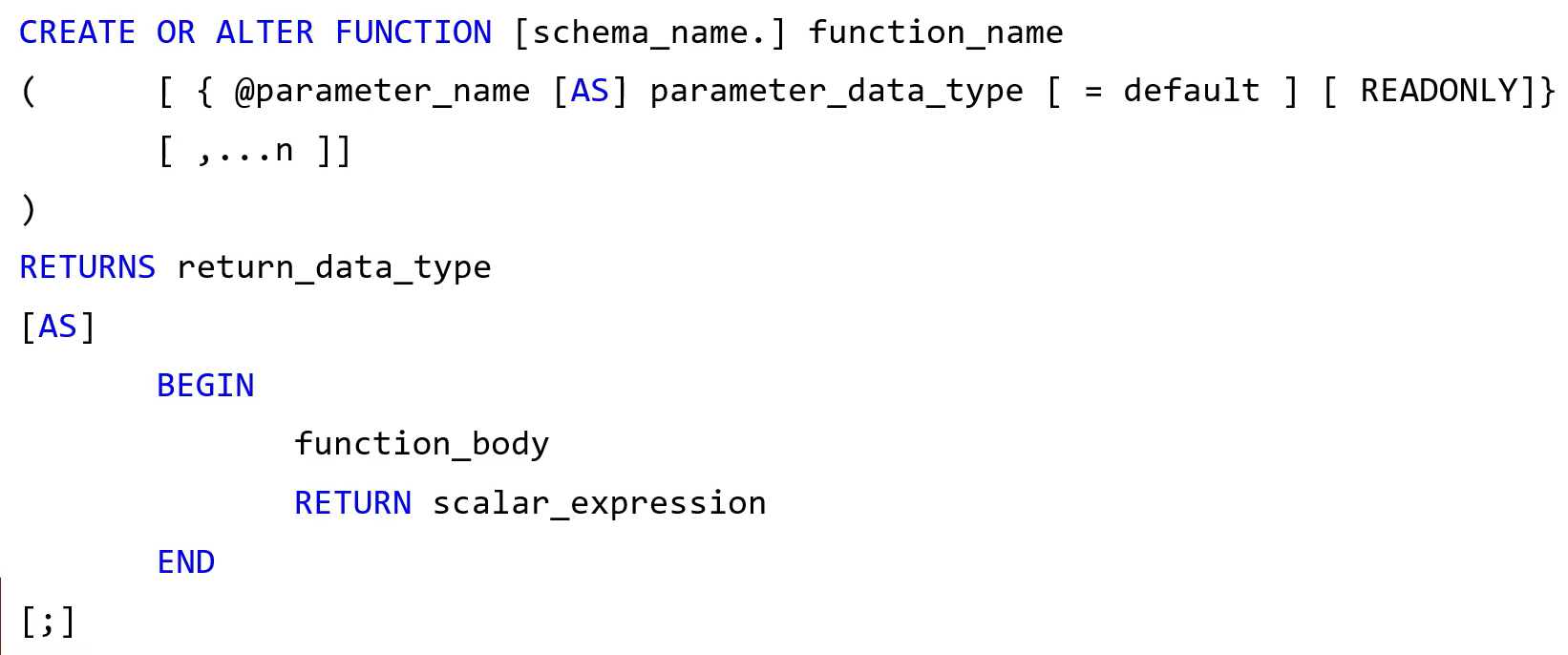
**User-defined scalar functions**

*Door gebruiker uitgewerkte functies waarbij het gaat over scalaire functies (die 1 enkele waarde teruggeven)*

**Concept**

* Kunnen gebruikt worden binnen elke scalaire expressie
  + Analoog aan functies zoals LOWER, UPPER, LEFT, RIGHT
* Omvat volgende onderdelen
  + Naam
  + Parameters
  + Return waarde
  + Implementatie

**Creatie**

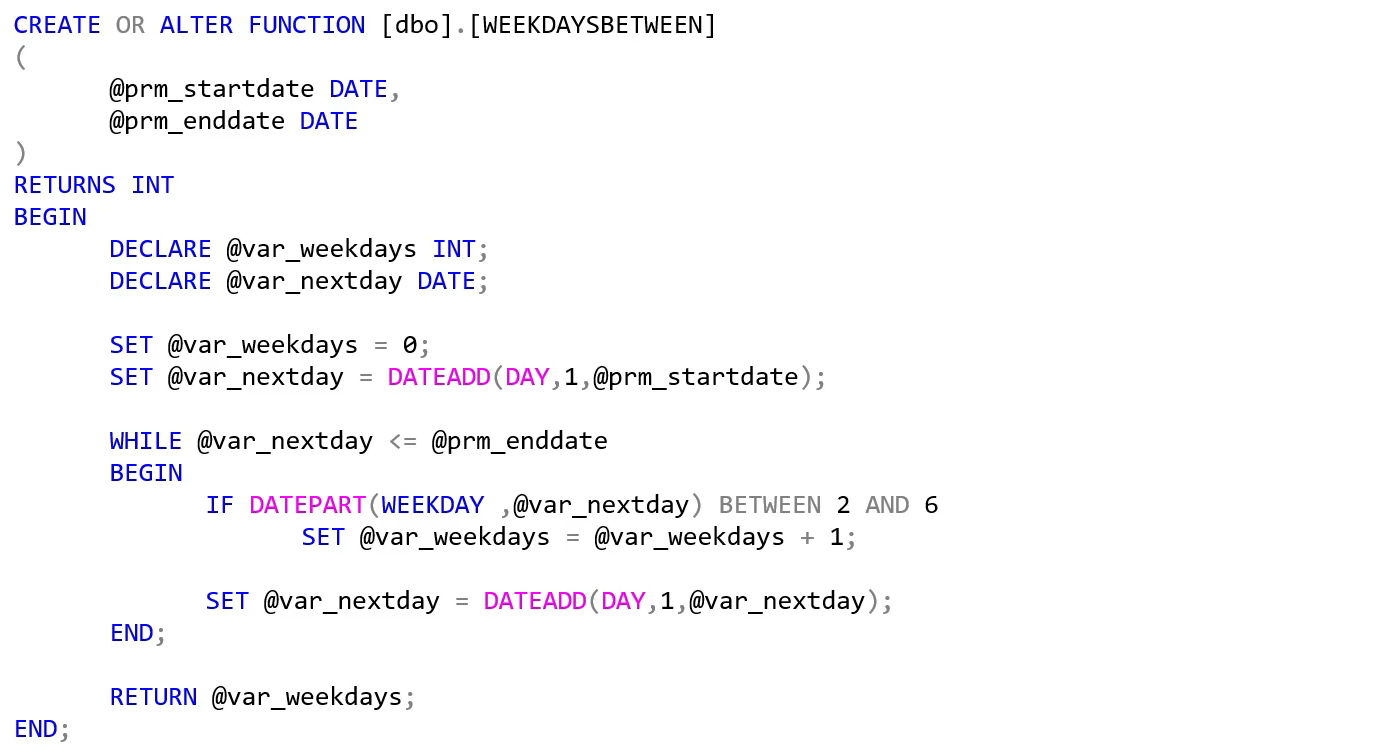
****

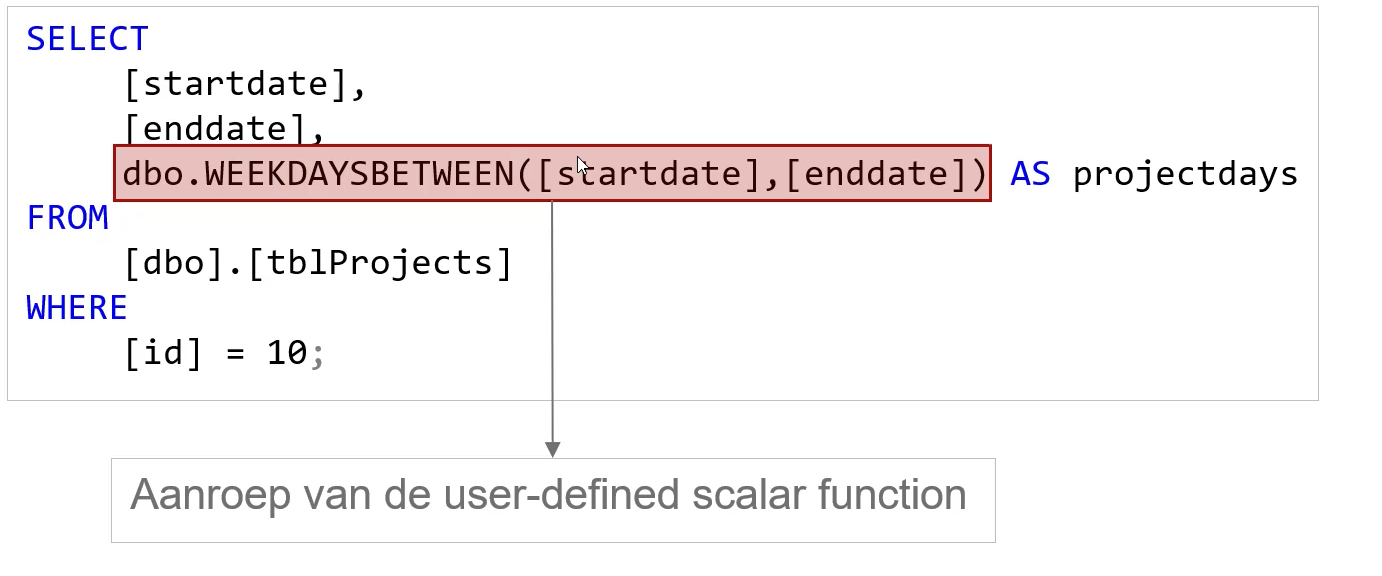
**Procedure =>** Resultaatset teruggeven of output parameter opvullen

**Functie =>** Scalaire waarde als resultaat teruggeven

**Voorbeeld**

* Een functie die het aantal weekdagen tussen een tijdspanne berekent
* Twee datums als input (begin- en einddatum)
* Geeft een geheel getal terug





**Functie gebruiken =>**