

leren. durven. doen.



# *DATABANKEN*

*C# programmeur*

*T-SQL*

*SQL als*

*volwaardige taal*

# INHOUD

## Stored functions

- Built-in functions
- Udf = User defined functions

## SQL server Transactions

## Triggers

# **SQL als volwaardige taal**

## **Stored Functions**

# Built- in Stored Functions in T-SQL

- **standard SQL functions:**  
**min,max,sum,avg,count**
- **non-standard built-in functions:**  
SQL Server: **datediff, substring, len,**  
**round, ...**

<http://technet.microsoft.com/en-us/library/ms174318.aspx>

- **user defined functions**

# User defined function

```
ALTER FUNCTION fn_GeefJarenVerschil
(
    @beginDatum AS DATE,
    @eindDatum AS DATE
)
RETURNS INT
AS
BEGIN
RETURN DATEDIFF(year, @beginDatum, @eindDatum)
- CASE WHEN 100 * MONTH(@eindDatum) +
DAY(@eindDatum) < 100 * MONTH(@beginDatum) +
DAY(@beginDatum)
    THEN 1
    ELSE 0
END;
END;
```

# Gebruik van User defined function

```
select
```

```
lastname,firstname,birthdate,hiredate,  
dbo.fn_GeefJarenVerschil(birthdate,getdate(  
) ) as leeftijd,  
dbo.fn_GeefJarenVerschil(hiredate,getdate()  
) as dienstjaren,  
dbo.fn_GeefJarenVerschil(birthdate,hiredate  
) as LeeftijdIndiensttreding  
from Employees
```

# Oefening User defined function 1

Schrijf een udf `fn_TakeRightChars(Tekst, x)` die in een gegeven tekst (max 1000) karakters de x rechtse karakters teruggeeft

Test je functie uit via de volgende instructie:

```
Use BierenDb;
```

```
Go
```

```
select [dbo].[fn_TakeRightChars](Naam, 3) as Rechtse3Chars from  
Bieren
```



# Oefening User defined function 2

Schrijf een udf `fn_StripAdditionalSpaces` die in een gegeven tekst (max 1000) karakters alle spaties die méér dan 1 keer na elkaar voorkomen vervangt door 1 enkele spatie

Test je functie uit via de volgende instructie:

```
PRINT [dbo].[fn_StripAdditionalSpaces>(' test ing ')
```

print in het message venster:

```
test ing
```

# Waarom PSM (SP en UDF) gebruiken ?

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

- 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 kan men 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

# T-SQL Advanced

## Transacties

# Wat zijn transacties?

Een transactie is een enkele eenheid van werk.

- Als een transactie succesvol is, worden alle gegevenswijzigingen die tijdens de transactie zijn doorgevoerd, vastgelegd en worden deze een permanent onderdeel van de database. (COMMIT)
- Als een transactie fouten aantreft en moet worden geannuleerd of teruggedraaid, worden alle gegevenswijzigingen gewist. (ROLLBACK)

```
USE tempdb;
GO
CREATE TABLE ValueTable ([value] int);
GO
DECLARE @TransactionName varchar(20) = 'Transaction1';
BEGIN TRAN @TransactionName
    INSERT INTO ValueTable VALUES(1), (2);
ROLLBACK TRAN @TransactionName;
    INSERT INTO ValueTable VALUES(3),(4);
SELECT [value] FROM ValueTable;
DROP TABLE ValueTable;
```

# Wat zijn transacties?

**Oefening Transactie:**

**GitHub Les 10**

**Oefening-Transactie-RekSaldo-27-10-2020.pdf**



# T-SQL Advanced

## Triggers

# procedurele database objecten

## Overzicht

soort	batches	opgeslaan als	uitvoering	ondersteunt parameters
script	meerdere	apart bestand	client tool (bv. Management Studio)	nee
stored procedure	1	database object	via applicatie of SQL script	ja
user defined function	1	database object	via applicatie of SQL script	ja
trigger	1	database object	via DML statement	nee

# Wat is een trigger?

- speciaal type SP
- **DML**, DDL, login/logoff triggers
- stuk code (~procedure) dat **automatisch wordt uitgevoerd** als een neveneffect van een actie op een tabel
  - geen parameters
  - kan niet expliciet aangeroepen worden  
wanneer het triggering event zich voordoet zal de trigger uitgevoerd worden ('firing' van de trigger)
- Aka **ECA** rules
  - **E**vent **C**ondition **A**ction

# Waarvoor triggers gebruiken?

- validatie van data en complexe constraints
- automatische generatie van waarden
- ondersteuning voor alerts  
bv automatisch een e-mail sturen wanneer een werknemer uit de tabel Employees wordt gewist
- bijhouden van audits en history van gegevens  
automatisch bijhouden wie wat doet en met welke tabel
- replicatie en gecontroleerd bijhouden van redundante data  
automatisch opvullen van datawarehouse-tabellen voor rapportering

# Create Trigger

```
CREATE TRIGGER trigger_name  
ON table_name  
BEFORE | AFTER | INSTEAD OF {[INSERT],[UPDATE],[DELETE]}  
[NOT FOR REPLICATION]  
AS  
{sql_statements}
```

- **triggering event**

- bepaalt bij welke gebeurtenis de trigger zal geactiveerd worden:

- insert**

- delete**

- update**

- bij update kan eventueel gespecificeerd worden voor welke kolommen de triggering event gegenereerd wordt

- update of ...**

# MS SQL Server:trigger test tabellen

## 2 tijdelijke tabellen

### —**deleted** tabel

- bevat kopies van de gewijzigde (update) of verwijderde (delete) rijen


tijdens de update of delete worden rijen gekopieerd naar de **deleted tabel**

### —**inserted** tabel

- bevat kopies van gewijzigde (update) of ingevoegde (insert) rijen

tijdens een update of insert wordt een kopie van elke rij die gewijzigd of toegevoegd wordt geplaatst in de **inserted tabel**

# Create Trigger – Timing: Before




```
CREATE TRIGGER [schema_name.]trigger_name  
ON table_name  
BEFORE | AFTER | INSTEAD OF {[INSERT],[UPDATE],[DELETE]}  
[NOT FOR REPLICATION]  
AS  
{sql_statements}
```

## – before

evaluatie van de trigger conditie en eventuele uitvoering van trigger actie gebeurt op de toestand van de DB zoals deze is **vóór** de triggering event zelf wordt afgehandeld

# Create Trigger – Timing: After




```
CREATE TRIGGER [schema_name.]trigger_name  
ON table_name  
BEFORE | AFTER | INSTEAD OF {[INSERT],[UPDATE],[DELETE]}  
[NOT FOR REPLICATION]  
AS  
{sql_statements}
```

## – after

evaluatie van de trigger conditie en eventuele uitvoering van trigger actie gebeurt op de toestand van de DB zoals deze is **nadat** de triggering event zelf wordt afgehandeld



# Create Trigger – Timing: Instead of



```
CREATE TRIGGER [schema_name.]trigger_name  
ON table_name  
BEFORE | AFTER | INSTEAD OF {[INSERT],[UPDATE],[DELETE]}  
[NOT FOR REPLICATION]  
AS  
{sql_statements}
```

## –instead of

Trigger wordt uitgevoerd ter vervanging van het DML statement

# Uitvoering van een trigger

## Volgorde bij een true-conditie:

1. uitvoering van **before statement level triggers** op de tabel
2. voor elke rij geaffecteerd door de trigger
  - a) uitvoering van **before triggers**
  - b) uitvoering van de **triggering event** (i.e. update/delete/insert)
  - c) toepassen van **referentiële constraints**
  - d) uitvoering van **after triggers**
3. uitvoering van **after statement level triggers** op de tabel

# Voordelen en nadelen

Grootste voordeel:

–mogelijkheid om **functionaliteit in de DB** op te slaan en **consistent uit te voeren** bij elke wijziging aan de DB

Dus:

–geen redundante code

functionaliteit zit op 1 plaats in db, niet in verschillende applicaties

–wijzigingen aanbrengen wordt eenvoudig

written & tested 'once' door een ervaren DBA

–veiligheid

Triggers zitten in DB dus kunnen alle beveiligingsregels volgen

–meer processing power

voor DBMS en DB

–past in client-server model

1 aanroep naar db-server waar veel kan gebeuren zonder dat verdere communicatie vereist is

# Voordelen en nadelen

## Nadelen

### —complexiteit

DB ontwerp, implementatie en onderhoud wordt complexer door het verschuiven van functionaliteit van de applicatie naar de DB

- **zeer moeilijk te debuggen!!!**

### —verborgen functionaliteit

- de gebruiker kan geconfronteerd worden met **onverwachte neveneffecten van de trigger**, mogelijks ongewenst
- triggers **kunnen cascaderen**, bij het ontwerp van de trigger is dit niet altijd duidelijk te voorspellen

### —performantie

- bij **elke wijziging aan de DB moet de triggerconditie geëvalueerd** worden

### —portabiliteit

- je pint je vast op het **dialect** van het gebruikte **DBMS**

# Voorbeeld trigger

```
CREATE TRIGGER tr_docentenHistory
```

```
ON Docenten
```

```
AFTER INSERT, DELETE
```

```
AS
```

```
BEGIN
```

```
    INSERT INTO DocentenHistory(Docent_ID, Voornaam, FamilieNaam, email, Datum)
```

```
    SELECT i.Docent_ID, i.Voornaam, i.Familienaam, i.email, getDate()
```

```
    FROM inserted i
```

```
    UNION
```

```
    SELECT d.Docent_ID, d.Voornaam, d.Familienaam, d.email, getDate()
```

```
    FROM deleted d
```

```
END
```

```
GO
```

Tabel Docenten



```
INSERT INTO Docenten (Voornaam, FamilieNaam, email)
VALUES('Jos', 'De Klos', 'jos.deklos@gmail.com');
```

Docent_ID	Voornaam	Familienaam	Email
1	Helena	Coppieters	hcoppieters@hotmail.com
2	Filip	Van Oosten	filip.test@com
3	Lowie	Delneste	lDeneste@test.com

Tabel DocentenHistory



Docent_ID	Voornaam	Familie naam	Email	Datum	HistId
4	Jos	De Klos	jos.deklos@gmail.com	2020-10-25	1
4	Jos	De Klos	Jos.deklos@hotmail.com	2020-10-25	2

# Oefening Trigger

## Oefening Trigger Database BierenDb

Maak een lege kopie **BierenHistory** van de tabel Bieren (gebruik `Insert * into BierenHistory From Bieren where 1=2`)

Voeg aan **BierenHistory** een nieuwe kolom toe **HistID** van type `int IDENTITY(1,1)`

En **verzet** de **PK** op **HistID**

Voeg een nieuwe kolom toe met naam **VerwijderDatum** van het type `datetime`

Schrijf een **AFTER DELETE trigger** op de tabel **Bieren** die een `INSERT` uitvoert op de Tabel **BierenHistory** waarin alle gegevens staan van de verwijderde rij uit Bieren.

Gebruik voor de invulling van de gegevens de trigger tabel `deleted`

Gebruik voor de waarde van kolom **VerwijderDatum** `getDate()`

# VRAGEN?

# REFERENTIES

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