Programmierung und Fehlerbehandlung

T-SQL-Batches

- Batches sind eine oder mehrere Anweisungen, welche eine Einheit bilden (Analyse, Optimierung, Ausführung)
- Schlüsselwort „GO“ kennzeichnet das Ende einer Batch
  - ➢ Batchtrennzeichen definierbar (Optionen, Abfrageausführung)
- Batches begrenzen den Gültigkeitsbereich von Variablen
- Einige Anweisungen (CREATE FUNCTION, CREATE PROCEDURE, ...) dürfen nicht mit anderen im gleichen Batch kombiniert werden, bzw. dürfen nur einmal pro Batch vorkommen

T-SQL-Batches II

- Batches werden als Einheit auf ihre Syntax analysiert; bei Syntaxfehlern wird der ganze Batch abgelehnt
- Batches können Code zur Fehlerbehandlung beinhalten

```
-- Gültiger Bereich
INSERT INTO dbo.t1 VALUES(1, 2, 'abc');
INSERT INTO dbo.t1 VALUES(1, 2, 'abc');
GO
- - Ungültiger Bereich
INSERT INTO dbo.t1 VALUE(1, 2, 'abc');
INSERT INTO dbo.t1 VALUES(1, 2, 'abc');
GO
```

Variablen

- Variablen sind Speicherobjekte, welche Werte aufnehmen können
  - Werden mit dem DECLARE-Schlüsselwort definiert
  - Seit SQL Server 2008 Deklaration und Initialisierung in einer Anweisung möglich
  - Gültigkeitsbereich auf die Batch beschränkt, in welcher sie definiert wurden

```
DECLARE @zahl1 AS INT;
SET @zahl1 = 3;

DECLARE @zahl2 INT = 1;

PRINT @zahl1 + @zahl2;
GO

PRINT @zahl1 * @zahl2;
```

@zahl1 deklariert und initialisiert (AS optional)

@zahl2 deklariert und initialisiert in einer Anweisung

Ausgabe hier nicht möglich, da @zahl1 und @zahl2 hier keine Gültigkeit mehr haben

Arbeiten mit Variablen

- Wertezuweisung mit SET oder SELECT
  - Mit SET kann man immer nur einer Variablen etwas zuweisen, mit SELECT kann man mehreren Variablen etwas zuweisen
  - Wenn ein Wert mit der SELECT-Anweisung zugewiesen wird, darf die Abfrage nur eine Zeile zurückgeben!

```
DECLARE @var1 NVARCHAR(max);
SELECT @var1 = lastname FROM Employees WHERE empid = 1;
DECLARE @var2 INT;
SET @var2 = 42;
```

Synonyme

- Alias oder Verknüpfung mit einem Objekt derselben SQL-Server-Instanz oder auf einem verknüpften Server
- Synonyme können auf Remoteobjekte verweisen
- Synonyme werden mit den Befehlen CREATE, ALTER und DROP erzeugt und verändert
- Synonyme können auf Tabellen, Sichten, Prozeduren und Funktionen zeigen

Beispiel

Synonym für Tabelle:

```
CREATE SYNONYM Tab FOR WAWI_FERTIG.dbo.tblPersonal;

SELECT * FROM Tab;
```

Synonym für Funktion:

```
CREATE SYNONYM NameITAx FOR fn_NameAnzeigen;

DECLARE @name NVARCHAR(30);
EXECUTE @name = NameITAx @var = 10;
SELECT @name;
```

```
CREATE FUNCTION fn_NameAnzeigen(@var INT)
RETURNS NVARCHAR(30)
AS
BEGIN
        DECLARE @var2 = NVARCHAR(30);
        SELECT @var2 = Vorname
        FROM ITAxTeilnehmer
        WHERE TeilnehmerID = @var;
        RETURN @var2;
END
GO
```

Sprachelemente zur Ablaufsteuerung

- Begin … END
  - Fassen Codeblöcke zusammen
- RETURNS / RETURN
  - Definieren bzw. Zurückgeben von Werten (auch Tabellenwerte, siehe Funktionen)
- IF … ELSE
  - Verzweigungen
- WHILE
  - Schleifen
- BEGIN TRY … END TRY BEGIN CATCH … END CATCH
  - Verhalten bei Fehlern

IF … ELSE

- Ausdruck nach IF wird ausgewertet
- Bei TRUE wird der Code aus dem IF-Block ausgeführt
- Bei FALSE oder UNKNOWN der Code aus dem ELSE-Block

```
DECLARE @var4 = NVARCHAR(30);

EXECUTE @var4 = nameITAx @var = 20;
IF @var4 = 'Gerwin'
        PRINT 'Dozent';
ELSE
        PRINT @var4;
GO
```

- Sehr Praktisch in Kombination mit dem EXISTS-Operator

WHILE

- Nach WHILE wird ein Ausdruck ausgewertet
- Bei TRUE wird der Code aus dem BEGIN … END Block ausgeführt
- Bei FALSE oder UNKNOWN wird die Schleife beendet
- Ausführung kann mit BREAK oder CONTINUE geändert werden

```
DECLARE @var1 INT = 10, @var4 NVARCHAR(30);
WHILE @var1 < 170
BEGIN
EXEC @var4 =  Vornameteilnehmer @var = @var1;
SET @var1 = @var1 + 10;
IF @var4 IS NULL
PRINT 'Keine Daten gefunden'
ELSE PRINT @var4;
END
GO
```

TRY ... CATCH

- Mithilfe von TRY ... CATCH kann auf Laufzeitfehler reagiert werden
  - TRY-Block ausführen und mögliche Fehler erfassen
  - Bei Auftreten eines Fehlers wird der CATCH-Block ausgeführt
  - Mit den bisher verarbeiteten Anweisungen kann man differenziert verfahren, z.B. Transaktion zurücksetzen, Fehler protokollieren u.v.m.
- Nicht alle Fehler lassen sich mit TRY ... CATCH ermitteln
  - Syntax- oder Kompilierungsfehler
  - Einige Fehler bei der Namensauflösung

Syntax

```
BEGIN TRY
        { sql_statement | statement_block }
END TRY
BEGIN CATCH
        [ {sql_statement | statement_block } ]
END CATCH [;]
```

- Auf einen TRY-Block muss ein CATCH-Block direkt folgen
- Ein TRY-CATCH – Konstrukt darf sich nicht über mehrere Batches erstrecken
- Ein TRY-CATCH – Konstrukt fängt alle Ausführungsfehler ab, deren Schweregrad größer als 10 ist und durch den die Datenbankverbindung nicht geschlossen wird
- Siehe: https://msdn.microsoft.com/de-de/library/ms164086.aspx

Eigenschaften und Funktionen

Eigenschaft	Abzufragende Funktion	Beschreibung
Nummer	ERROR_NUMBER	Eindeutige Nummer, welcher der Fehler zugewiesen wird
Nachricht	ERROR_MESSAGE	Text der Fehlermeldung
Schweregrad	ERROR_SEVERITY	Schweregrad (Klasse 1 bis 25)
Prozedurname	ERROR_PROCEDURE	Name des Verursachers (Prozedur oder Auslöser)
Zeilennummer	ERROR_LINE	Nummer der Zeile, welche den Fehler im Batch, in der Prozedur, in dem Auslöser oder in der Funktion verursacht hat

- Die zurückgegebenen Werte entsprechen der sys.messages-Ansicht
- Ab SQL Server 2012 THROW-Anweisung
  - ➢ Nachfolger der RAISERROR-Anweisung

THROW

- THROW bietet Auswahlmöglichkeiten beim Behandeln von Fehlern
  - ➢ im lokalen CATCH-Block behandeln oder an andere Prozesse übergeben
- THROW wird verwendet
  - ➢ Mit Parametern, um benutzerdefinierte Fehler zu übergeben
  - ➢ Ohne Parameter, um den ursprünglichen Fehler erneut auszulösen (muss in einem CATCH-Block sein)

```
THROW 55500, 'Blabla', 4711;
```

```
BEGIN TRY
        SELECT 1/0 AS Problem
END TRY
BEGIN CATCH
        PRINT 'ERROR: ' + CAST(ERROR_NUMBER() AS VARCHAR(255));
        THROW;
END CATCH
```