3. Die Struktur der Datenbank ändern

Gegeben in: Woche 5

Abgabe:

Woche 6 – Teil I (5 Prozeduren + RollbackProzeduren)

Woche 7 – Teil II

Die Endnote wird nur in Woche 7 berechnet.

Lese erstmal die Theorie durch und dann die Aufgabe!

Theorie

Syntax:

```
CREATE PROCEDURE <Name> [@param1 type1, ...] AS
    -- Sequenz von SQL Befehele
GO
```

Eine gespeicherte Prozedur wird **mit EXEC ausgeführt:**

```
EXEC <Name>
```

Eine Prozedur ändern:

ALTER PROCEDURE .....

Prozedur mit Output Parameter:

```
CREATE PROCEDURE getKursTitel(@Kredite int, @Number int output)
AS
     SELECT @Number = COUNT(*)
     FROM Kurse
     WHERE ECTS = @Kredite
GO
```

Prozedur mit Output ausführen:

```
DECALRE @Nr int
SET @Nr = 0
exec getKursTitel 6, @Number=@Nr output
print @Nr
```

Dynamische Execution

Syntax:

```
EXEC (command)
```

Beispiel:

- ```
  EXEC ('SELECT * FROM Studenten WHERE age>20')
  GO
  ```

oder

- ```
  DECLARE @query as varchar(MAX)
  SET @query = 'SELECT * FROM Studenten WHERE age>20'
  EXEC(@query)
  GO
  ```

Nachteile von EXEC:

- kann schlecht für Performance sein (es kann sein dass SQL Server für dynamisches SQL den Ausführungsplan jedes Mal neu erstellen muss)
- potentielle Sicherheit Probleme (SQL Injection)

Alternative für EXEC → gespeicherte Prozedur **`sp_executesql`**:

- manchmal viel schneller als EXEC
- verhindert SQL Injection
- sie können Parameter nur dort verwenden, wo die SQL Syntax dies auch zulässt → es dürfen keine Parameter für Spalten oder Tabellennamen verwenden werden
- wenn dynamisches SQL regelmäßig verwendet wird, ist **`sp_executesql`** die bessere Wahl, da der Abfrageplan/ Ausführungsplan wiederverwendet werden kann

```
DECLARE @query as nchar(50)
SET @query = 'SELECT * FROM Studenten WHERE age>@Nr'
EXECUTE sp_executesql @query, N'@Nr int', @Nr=20;
```

Bemerkung!

1. Wenn Parameter auch für Spalten oder Tabellennamen verwendet werden müssen, funktioniert das nur mit EXEC
2. Wenn Parameter nur für Werte gebraucht werden, dann benutzt man EXEC nicht
3. Wenn die Abfrage als Verkettung von Strings aufgebaut wird, passt auf wo man Leerzeichen braucht (mit **PRINT** die Abfrage überprüfen)

Beispiel von einer Prozedur mit dynamischer Execution

Eine Prozedur, die eine Indexstruktur löscht:

```
create procedure deleteIndex (@tableName varchar(10),
                              @indexName varchar(10))
 as
   begin
   declare @sqlQuery as varchar(MAX)
   set @sqlQuery = 'drop index ' +@tableName +'.'+ @indexName

   print(@sqlQuery)
   exec(@sqlQuery)
   end
 go
```

Die Prozedur ausführen:

```
   exec deleteIndex 'table1', 'index1'
```

Aufgabe

Manchmal müssen wir die Struktur der Datenbank, nachdem diese erstellt wurde, ändern. Aber solche Änderungen sind nicht immer korrekt, also es muss eine Möglichkeit geben zu einer vorherigen Version zurückzukehren. Eure Aufgabe ist es, ein Versioning Mechanismus zu erstellen, der einem Benutzer erlaubt von einer Version zu einer anderen zu wechseln.

Teil I (2.5p)

Schreibe **generische (parametrisierte) gespeicherte Prozeduren** (wenn die SQL Anweisungen generieret wurden müssen diese genau erklärt werden können), die folgende Operationen implementieren (mit entsprechenden Parametern):

a) den Typ einer Spalte (Attribut) ändert (modify type of column)
b) ein default Constraint erstellt
c) eine neue Tabelle erstellt (create table)
d) eine neue Spalte für eine Tabelle erstellt (add a column)
e) eine Referenz-Integritätsregel erstellt (foreign key constraint)

Für jede dieser Operationen schreibe eine neue Prozedur die den **Rollback/Revert** implementiert (das umgekehrte der Operation), oder erkläre warum eine neue Prozedur nicht notwendig ist.

Teil II (6.5p)

Erstelle eine neue Tabelle, die nur die aktuelle Version der Datenbank speichert (eine Spalte mit einer einzigen Nummer = aktuelle Version).

Erstelle eine zusätzliche Tabelle für die Versionen. Für jede neue Version muss man Folgendes speichern:

- welche neue Prozedur wurde ausgeführt (um zu dieser Version zu gelangen)
- mit welchen Parametern (wenn nötig)

Schreibe eine gespeicherte Prozedur, die eine Versionnummer als Parameter kriegt und die Datenbank zu dieser Version bringt.

Bemerkungen.

1. Wenn man eine der 5 Prozeduren ausführt, dann sollte automatisch die neue Version in der entsprechender Tabelle eingefügt werden und die Versionnummer inkrementiert werden.
2. Die Prozedur welche die Version ändert arbeitet mit der Tabelle von Versionen.
3. Keine Version wird gelöscht, man kann z.B. von Version 5 zu Version 2, und dann von Version 2 zurück zu Version 4 gehen.
4. Die Prozeduren können in beliebiger Reihenfolge ausgeführt werden, um Versionen zu erstellen, solange die Parameter korrekt sind.

Für mehrere Informationen über Transact-SQL (declare, begin, end, set, usw.) und gespeicherte Prozeduren:

- Seminar 3
- https://msdn.microsoft.com/de-de/library/ms190487.aspx
- https://msdn.microsoft.com/de-de/library/ms188927.aspx
- https://msdn.microsoft.com/de-de/library/ms189484.aspx
- https://msdn.microsoft.com/de-de/library/ms178642.aspx
- https://msdn.microsoft.com/de-de/library/ms188001.aspx
- https://msdn.microsoft.com/de-de/library/ms190782.aspx