FB 4 – Wirtschaftsinformatik
Prof. Dr. Peter Zschockelt

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Kurzreferenz SQL-Server 2008/2012 (1)

1 Administration der persönlichen SQL Server 2008 Datenbank

Das DBMS MS SQL Server 2008 wie auch 2012 wird im DB-Labor des FB 4 der HTW proprietär mittels 'Microsoft SQL Server Management Studio' verwaltet.

Bei nicht sachgerechter Administration besteht die Gefahr, dass die Zugangsinformationen (Passwort und User-ID) auf dem Client (PC-Labor-Arbeitsplatz) aktiviert bleiben und der nachfolgende Nutzer mit dieser Identifikation weiter arbeitet.

Aus diesem Grund wird die nachfolgend beschriebene Konfigurationsvariante unbedingt empfohlen.

Starten Sie das Microsoft SQL Server Management Studio aus dem Programmfenster 'Datenbanken Programme' → 'Microsoft SQL Server Management Studio'.

Sofern das Fenster "Verbindung mit Server herstellen“ nicht automatisch startet, klicken Sie im Objekt-Explorer auf "Verbinden“

In dem sich dann öffnenden Anmeldefenster geben Sie Ihre Anmeldedaten für den Server ein:

Nach erfolgreicher Anmeldung klicken Sie im Verzeichnis "Datenbanken" des Objektexplorers auf den Namen Ihrer persönlichen Datenbank oder der von Ihnen zur Nutzung vorgesehenen öffentlichen Datenbank. Durch Klicken der Menütaste "Neue Abfrage" öffnen Sie ein Abfragefenster für die Eingabe von SQL-Anweisungen. Wird für eine Abfrage ein Wechsel zu einer anderen Datenbank gewünscht, können Sie dies durch die SQL-Anweisung Use *datenbankname* erreichen (z. B. für die Nutzung der Datenbank zb1w8999999: Use zb1w8999999).

2 SQL-Anweisungen

2.1 Notation und generelles Sprachformat

Als **Notationsform** wird eine Mischung zwischen SQL-Quelltext-Kommentaren und der für die Beschreibung von Programmiersprachen üblichen Symbolik verwendet. Im Einzelnen bedeuten:

- Zeichenfolgen nach **--** (2 Minuszeichen) SQL-Kommentare,
- **[** Variable/SQL-Worte in eckigen Klammern **]** wahlfreie Angaben,
  (Bitte verwechseln Sie dies nicht mit der für SQL Server generell möglichen Angabe von Namen in eckigen Klammern !)
- { Auswahltext in mehreren Zeilen } alternative Angaben (**eine** muss gewählt werden)
- Worte in Großbuchstaben SQL-Schlüsselworte,
- Worte in Kleinbuchstaben Variable.

Die allgemeine **Darstellungsform** von SQL ist formatfrei. Die einzelnen SQL-Schlüsselworte und/oder Variablen müssen jedoch immer vollständig in einer Zeile stehen und dürfen nicht abgeteilt werden.

Als **Trennzeichen** zwischen
- einzelnen Bezeichnern (SQL-Schlüsselworte, Variable, Literale) stehen 1 bis n Leerzeichen (Steht eine runde Klammer, ist kein Trennzeichen erforderlich.);
- einzelnen SQL-Anweisungen steht bei SQL-Server 2008 das Wort **GO** oder ein Semikolon;
- den Elementen einer Liste (z. B. Feldliste) stehen Kommas (**,**).

Datenbank-Objekt-Namen werden in Standard-SQL mit Buchstaben und Ziffern dargestellt (z. B. Create table *NeuTab*). Sie können in SQL-Server aber auch durch eckige Klammern eingeschlossen werden. Diese T-SQL-Erweiterung ist zum Beispiel nützlich, wenn Leer- oder Sonderzeichen in Spaltennamen enthalten sein sollen (z. B. Create table *[Neue Tab]*)

Generell können Datenbankobjekte mit dem Datenbanknamen, dem Schemanamen, Tabellennamen und ggf. Spaltennamen bezeichnet werden. Sofern Eindeutigkeit gegeben ist, können jedoch alle Angaben bis zum Tabellen- bzw. Spaltennamen entfallen.

Beispiele:	vollständig:	Select * from Hotel.dbo.Test
	bei Eindeutigkeit:	Select * from dbo.Test
	bzw.	Select * from Test

2.2 Anweisungen zur Strukturierung von Datenbanken und Tabellen

2.2.1 Erstellen und Löschen einer Datenbank

```
CREATE DATABASE datenbankname
```

erzeugt eine neue Datenbank.

```
DROP DATABASE datenbankname
```

löscht eine Datenbank.
Eine geöffnete (aktive) Datenbank kann nicht gelöscht werden. Deshalb ist vor dem Löschen ggf. mit USE *datenbankname* der Benutzerkontext auf eine andere Datenbank (z. B. use tempdb) zu ändern.

2.2.2 Erstellen, Ändern und Löschen einer Tabelle

```
CREATE TABLE [#[#]]tabellenname
(spaltenname datentyp [DEFAULT ... ][NOT NULL], ...)
```

ist die Grundform einer Tabellendefinition.
Die optionale (wahlfreie) Angabe DEFAULT bezeichnet einen Vorgabewert der Spalte für jede neu in die Tabelle aufgenommene Zeile. Möglich sind z. B. ein Literal (z. B. "Berlin"), NULL, Getdate() (aktuelles Datum), USER (Nutzername im System).
Die optionale Angabe NOT NULL legt fest, dass die Spalte unbedingt einen Wert enthalten muss.
Eine Tabelle kann temporär (nur für die Dauer einer Sitzung) definiert werden, indem dem Tabellennamen ein Zeichen # (lokal) bzw. zwei Zeichen ## (global) vorangestellt werden.

Das nachträgliche Ändern einer Tabellendefinition ist mit den Anweisungen

```
ALTER TABLE tabellenname
ADD neuer_spaltenname datentyp ...
```

für das Hinzufügen von Spalten, mit

```
ALTER TABLE tabellenname
DROP COLUMN spaltenname
```

für das Löschen von Spalten und mit

```

```

```
ALTER TABLE tabellenname
 ALTER COLUMN spaltenname datentyp
```

für das Modifizieren von Spalten möglich.

Eine wesentliche funktionale Erweiterung der Tabellendefinition erfolgt durch die Festlegung von Constraints (Prüfvorschriften). Generell sind wahlfrei drei unterschiedliche Constraint-Typen verfügbar.

Diese prüfen

- die Eindeutigkeit aller Zeilen (UNIQUE, PRIMARY KEY),
- die Bezugnahme auf eine referenzierte Tabelle (REFERENCES-Klausel),
- die Zulässigkeit der Werte einer Spalte (CHECK-Klausel).

Prinzipiell können sich Constraints nur auf eine Spalte (Spaltenconstraints) oder auf die gesamte Tabelle (Tabellenconstraints) beziehen. Die wichtigsten Tabellenconstraints sind:

Der **Eindeutigkeits-Constraint**:

```
CREATE TABLE tabellenname
(spaltenname datentyp [NOT NULL], ...
[ CONSTRAINT constraintname ]
{UNIQUE | PRIMARY KEY} (spaltenname, ...)
)
```

Nachträglich kann in einer bereits definierten Tabelle mit

```
ALTER TABLE tabellenname
ADD [ CONSTRAINT constraintname ]
  {UNIQUE | PRIMARY KEY} (spaltenname, ...)
```

der gleiche Effekt erzielt werden.

Die **referentiellen Abhängigkeit** zwischen Tabellen:

Mittels der ALTER TABLE-Anweisung formuliert man das Hinzufügen einer Referenz wie folgt:

```
ALTER TABLE tabellenname
ADD [CONSTRAINT constraintname ]
FOREIGN KEY (spaltenname, ...)
REFERENCES bezugstabellenname (bezugsspaltenname, ...)
```

Der **CHECK-Constraint** für den Inhalt einer Spalte:

Er kann ebenfalls bei Anlegen der Tabelle oder nachträglich mittels der ALTER TABLE-Anweisung wie folgt formuliert werden:

```
ALTER TABLE tabellenname
ADD [ CONSTRAINT constraintname ]
CHECK (bedingung)
```

Als Bedingung können unter anderem der Vergleich mit einer Konstanten (zum Beispiel: pnr BETWEEN 0000 AND 7999) oder der Vergleich von Spalten innerhalb einer Tabelle (zum Beispiel: ende > beginn) angegeben werden.

Zum Löschen eines Constraints wird innerhalb der ALTER TABLE-Anweisung an Stelle von ADD CONSTRAINT die Anweisungsspezifikation DROP CONSTRAINT constraintname benutzt.

```
DROP TABLE tabellenname
```

löscht eine Tabelle.

2.2.3 Umbenennen von Tabellen

Tabellen können auf SQL-Server 2008 nur indirekt über die vorgefertigte Stored Procedure

```
EXEC sp_rename 'alter_tabellenname', 'neuer_tabellenname'
```

umbenannt werden.

2.3 SQL-Anweisungen zum Manipulieren von Sätzen in einer Tabelle

2.3.1 INSERT INTO zur Neuaufnahme von Sätzen

```
INSERT INTO tabellenname [(spalte1, ... spaltex)]
            VALUES (wert1, ... wertx)
```

zur Übernahme in der INSERT-Anweisung explizit angegebener Daten oder

```
INSERT INTO tabellenname [(spalte1, ... spaltex)]
            SELECT-Anweisung
```

zur Übernahme von Daten aus einer anderen Tabelle. Diese Tabelle kann sich auch auf eine andere SQL-Server Datenbank der gleichen Instanz beziehen.

Bemerkung:

- Sollen nicht alle Felder der Tabelle ausgefüllt werden, kann nach *tabellenname* eine Liste der Form **(spalte1, spalte2, ... spaltex)** angegeben werden, z. B.:
  insert into buchung (rnr,pnr,ktnr,lnr,anmeldung) values ('1999-19243','0213','01',1,getdate())
- Es muss mindestens ein Wert angegeben sein (danach steht kein abschließendes Komma).
- CHAR- und Datums-Werte sind in Hochkommata (') zu setzen.
- SELECT ... ist entsprechend der Syntaxbeschreibung der SELECT-Anweisung zu formulieren.

Beispiel: INSERT INTO dozent VALUES ('2400',54.80, '1')

2.3.2 UPDATE zum Ändern von Sätzen (Zeilen)

```
UPDATE tabellenname
    SET spalte1 = wert1,  ... spaltex = wertx
    [WHERE bedingung ]
```

Bemerkung:

- Es muss mindestens ein Feld (entsprechend Wert) angegeben werden. Nach der letzten Angabe folgt kein abschließendes Komma.
- Die Angabe von WHERE bedingung ist optional.
- CHAR- und DATE-Werte sind in Hochkommata (') zu setzen.
- An Stelle eines Wertes (z. B. wert1) kann auch eine Unterselectanweisung stehen. Diese darf jedoch nur genau eine Zeile pro Aufruf als Ergebnis zurückliefern.

Beispiel: UPDATE person SET ort = 'Berlin' WHERE ort LIKE 'Berlin%'

2.3.3 DELETE FROM zum Löschen von Sätzen

```
DELETE FROM tabellenname WHERE bedingung
```

Bemerkung:

- Gelöscht wird der Inhalt einer Tabelle, nicht die Tabelle selbst.
- Die Angabe von WHERE bedingung ist optional. Wird sie nicht angegeben, so wird der gesamte Tabelleninhalt gelöscht.

Beispiel: DELETE FROM person WHERE famname = "Superuser"

2.4 SELECT-Anweisung zur Projektion und Selektion

```
SELECT [DISTINCT]
 spaltenname
 [[as]neuer_spaltenname],          { neuer-spaltenname ist optional }
                          { oder }
 (Unterselectanweisung)            { an Stelle eines spaltennamens }
                          { oder }
 *  { für die Gesamtheit aller Feldnamen }
FROM   tabellenangabe   { oder } (Unterselectanweisung)
   { danach folgen optional keine oder eine Kombination der nachfolgenden Klauseln }
[WHERE]                 {Bedingung}
[ORDER BY] spaltenname, ...
[GROUP BY] spaltenname, ...
[HAVING]                {Bedingung nach GROUP BY)
[UNION]      {Select-Anweisung}     { oder }
[INTERSECT]  {Select-Anweisung}     { oder }
[EXCEPT]     {Select-Anweisung}
```

Erklärung:

- Die Angabe DISTINCT bewirkt, dass mögliche Duplikate unter den selektierten Sätzen nicht ausgegeben werden.
- **spaltenname** kann (bzw. muss bei unterschiedlichen Tabellen) mit tabelle.spaltenname angegeben werden (qualifizierter Name) – siehe auch: S. 7. Datenbank-Objekt-Namen.
- für **tabellenangabe** steht eine Liste (Listenelemente durch Komma getrennt) von Tabellennamen und/oder von Joinings (siehe auch Anmerkung zum Join auf Seite 13).

  Beispiele: Select name, ort from adressen, bkopf where adressen.regnr=bkopf.regnr
  Select name, ort from adressen join bkopf on adressen.regnr=bkopf.regn
- Eine **Bedingung** kann unter anderem formuliert werden als

  ⇨ **einfache Bedingung** mit den Vergleichsoperatoren

=	gleich	<>	ungleich	!=	ungleich
>	größer	>=	größer gleich		
<	kleiner	<=	kleiner gleich.		

  ⇨ **Mustervergleich** in Character-Feldern mit dem Schlüsselwort LIKE 'maske'
  In der Maske können neben beliebigen Zeichen folgende Operatoren (Ersetzungszeichen) stehen:

%	ersetzt eine beliebige Folge von 0-n Zeichen (z. B. Feld1 LIKE 'A%');
__	ersetzt genau ein beliebiges Zeichen (z. B. Feld1 LIKE 'Artikel_');
[...]	gibt einen Wertebereich für genau ein Zeichen an. Möglich sind als Angabe (in Beispielform): [AZ] für Wert A oder Z (z. B. Tab1.Feld1 LIKE '[AaZz]%'), [A-Z] für alle Werte von A bis Z, [^A] für aller Werte außer A.

  ⇨ **Bereichsabfrage** mit der Angabe BETWEEN *von* AND *bis*, z. B.:
  select ktnr, lnr from veranstaltung where beginn between '01.11.98' and '31.03.99'

  ⇨ **Abfrage nach einem Null-Wert** mit der Angabe IS NULL, z. B.:
  select * from buchung where abmeldung IS NULL

  ⇨ **Mengenabfrage** mit der Angabe IN (menge), z. B.: Feld1 IN (1,2,5).
  menge kann auch die Ergebnisspalte einer Unterselectanweisung sein.

⇨ **Existenzabfrage** durch die Klausel EXISTS (SELECT ...).
Hier wird danach gefragt, ob die innere (in Klammern gesetzte) SELECT-Anweisung (Unterselectanweisung) wenigstens eine Ergebnis-Zeile liefert.

⇨ **erweiterte Bedingung** durch Verknüpfung der bisher genannten (einfachen) Bedingungen mit den Schlüsselworten AND, OR und NOT.

Eine **Unterselectanweisung** innerhalb einer SELECT-Anweisung muss in runden Klammern stehen. In der Unterselectanweisung können die in der übergeordneten SELECT-Anweisung aufgeführten Tabellen ebenfalls angesprochen werden
(z. B. Select * from bestellung where not exists (Select bnr from position where bestellung.bnr=position.bnr)).

- **ORDER BY** spaltenname$_1$,... spaltenname$_x$ wird verwendet zur sortierten Ausgabe des Ergebnisses entsprechend dem/den angegebenen Sortierfeld(ern). Bei fallender Sortierfolge ist die Option DESC (Standard: ASC) anzugeben.
- **GROUP BY** liefert Ergebnisse für Gruppen innerhalb einer Spalte oder einer Liste von Spalten (Listenelemente durch Komma getrennt). Eine Gruppe wird durch alle Sätze einer Spalte bzw. der Liste von Spalten gebildet, die den gleichen Inhalt haben (z. B. alle Artikelbezeichnungssätze mit der Artikelnummer 91280). Die Verwendung der GROUP-BY-Klausel ist nur in Verbindung mit Aggregaten (siehe nachfolgender Abschnitt '"Besonderheiten") sinnvoll.
- **HAVING** bedingung ist **nach** GROUP BY an Stelle der in dieser Konstellation nicht zulässigen WHERE-Klausel zu verwenden. Als Variable sind nur im Ergebnis der GROUP-BY-Operation berechnete Werte möglich.
- **UNION** select-anweisung leitet die Ausgabe aller mit UNION verknüpften Select-Anweisungen in eine gemeinsame Ausgabetabelle. Die Spaltennamen werden von der ersten Select-Anweisung übernommen. Die Spaltenanzahl aller durch UNION vereinigten Select-Anweisungen muss folglich gleich sein. Bei ORDER BY muss statt des Namens die Ordnungszahl der Sortierspalte angegeben werden.
- **INTERSECT** select-anweisung (logischer Durchschnitt der Resultate zweier Select-Anweisungen) schreibt nur die Zeilen in eine gemeinsame Ausgabetabelle, die im Resultat-Set beider Select-Anweisungen stehen. Das Ergebnis kann wiederum mit INTERSECT verbunden werden.
- **EXCEPT** select-anweisung (logische Mengendifferenz der Resultate zweier Select-Anweisungen) schreibt nur die Zeilen in eine gemeinsame Ausgabetabelle, die im Resultat-Set der ersten, aber nicht im Resultat-Set der zweiten Select-Anweisungen stehen. Das Ergebnis kann wiederum mit INTERSECT verbunden werden.

Besonderheiten:

☞ Für numerische Spalten können an Stelle von ***spaltenname*** Rechenoperationen angegeben werden, die sich auf den Spalteninhalt beziehen. Möglich sind u. a.

- die vier Grundrechenarten + - * /
  (z. B. SELECT artikel, preis Netto, preis * 1.19 AS brutto FROM liste)
- die Funktionen ROUND(wert, stelligkeit), CAST(wert AS typ) u. a.
  (z.B. ROUND(2345.9809, 2) → 2345.9800
  CAST(2345.9809 AS NUMERIC (10,2)) → 2345.98)
- die Angabe von **Aggregaten** (Berechnungsfunktionen für eine Gruppe bei Verwendung von GROUP BY bzw. für die gesamte Tabelle, wenn keine GROUP-BY-Klausel angegeben ist). Als Aggregate sind folgende Funktionen definiert:

  - COUNT(*) ermittelt die Anzahl der Sätze einer Tabelle bzw. Gruppe.
    (z. B. SELECT COUNT(*) FROM person)
  - SUM (spaltenname) Summe der Werte des angegebenen Feldes
    (z. B. SELECT rnr, sum(betrag) Gesamt FROM zahlung GROUP BY rnr)
  - AVG (spaltenname) Durchschnittswert der Werte des angegebenen Feldes
  - MAX (spaltenname) höchster Wert des angegebenen Feldes
  - MIN (spaltenname) niedrigster Wert des angegebenen Feldes

  Beispiel:
  SELECT AVG(groesse) Durchschnitt, MIN(groesse) Kleinste, MAX(groesse) Groesste FROM person

☞ Für alphanumerische Spalten können an Stelle von ***spaltenname*** Verknüpfungen des Inhalts mehrerer Zeichenketten-Spalten mittels des Verknüfungsoperators + (nicht verwechseln mit dem Additionsoperator für numerische Spalten) angegeben werden.
(z.B. 'Datum: ' + Beginn ergibt, sofern der Inhalt der Spalte Beginn der 31.12.2008 ist, den Ausdruck "Datum: 31.12.2008").

☞ Für Datumsfelder können Teile des Datums oder das gesamte Datum u. a. durch folgende Datum-Funktionen bezeichnet werden:

- DAY(datumsfeldname) gibt den Tag an,
- MONTH(datumsfeldname) gibt den Monat an,
- YEAR(datumsfeldname) gibt das Jahr an,
- GetDate() oder `CURRENT_TIMESTAMP` gibt das aktuelle Systemdatum (Datetime) an.
- DATEDIFF(datepart,startdate,enddate) berechnet Datumsdifferenzen wie folgt:
  datepart benennt die zu berechnende Zeiteinheit z.B. yy oder yyyy für Jahreszahl, mm für Monatszahl, dd für Tage, q für Quartale, hh für Stunden, mi für Minuten, ss für Sekunden, ms für Milisekunden.
  startdate und *enddate* legen den Beginn und das Ende der Zeitperiode fest.
- DATEADD(datepart,number,date) berechnet ein neues Datum nach Addition eines Datumsteils wie folgt:
  datepart benennt die zu berechnende Zeiteinheit (siehe Funktion DATEDIFF)
  number gibt die Anzahl der zu addierenden Zeiteinheiten an (kann auch negativ sein)
  date ist das Ausgangsdatum für die Berechnung.

  Hinweis: Die Addition einer ganzen Zahl wird immer als Datumsteil "Tag" gewertet. Zur Tagesaddition/-subtraktion ist also die Funktion DATEADD nicht erforderlich.

Datumsangaben werden generell als Zeichenketten dargestellt. In Abhängigkeit von der Serverkonfiguration wird im deutschsprachigen Raum in der Regel die Form 'TT.MM.JJJJ' (für den Datentyp DateTime ergänzt um 'HH:MM:SS.MS') verwendet. Generell kann aber auch die "englische" Form 'YYYY-MM-DD hh:mi:ss.ms' benutzt werden.
z. B.: select rnr,anmeldung from buchung where anmeldung >= '01.10.2008'
select rnr,anmeldung from buchung where anmeldung >= '2008-07-06 10:38:29.999'

Zur **Konvertierung** von Zeichenkettendaten in DateTime-Typen kann die Funktion CAST benutzt werden, z. B. CAST('01.12.2004 09:02' AS smalldatetime). Ebenso kann ein Datumswert in eine Zeichenkette konvertiert werden, z. B. convert(varchar,CURRENT_TIMESTAMP,104).

Anmerkung zum Join:

Die SQL-92 konforme Form des Joinings als Bestandteil der FROM-Klausel einer Select-Anweisung ist in SQL Server 2008 wie folgt definiert:

`tabellenname [ART] JOIN tabellenname ON bedingung.`

Für **ART** steht

- INNER für den natürlichen Join (Standard, falls *ART* nicht angegeben),
- LEFT [OUTER] für den linken Outer-Join),
- RIGHT [OUTER] für den rechten Outer-Join oder
- FULL [OUTER] für den "beidseitigen" Outer-Join.

Unter **bedingung** wird formuliert, welche Spalten der beiden am Joining beteiligten Tabellen einen übereinstimmenden Wert aufweisen müssen.

Beispiel: Select adressen.regnr, name, angebot.anr, aname, preis
from adressen INNER JOIN angebot ON angebot.regnr = adressen.regnr
INNER JOIN artikel ON angebot.anr = artikel.anr

Natürlich ist auch die generell gültige Form der logischen Formulierung des natürlichen Joins Für SQL Server 2008 wie für jedes andere relationale DBMS möglich, im Beispiel:

```
Select adressen.regnr, name, angebot.anr, aname, preis
  from adressen, angebot, artikel
  Where angebot.regnr = adressen.regnr AND angebot.anr = artikel.anr
```

2.5 Anweisungen zum Erzeugen und Löschen von Views

```
CREATE VIEW viewname (viewspalte1, ... viewspaltex)
AS SELECT ... [ WITH CHECK OPTION ]
```

erzeugt einen View.

Bemerkung:

- SELECT ist entsprechend der Syntaxbeschreibung der SELECT-Anweisung zu formulieren. Die in der SELECT-Anweisung genannten Spalten werden in der angegebenen Reihenfolge den nach viewname in Klammern angegebenen Viewspalten (viewspalte1,... viewspaltex) zugeordnet. Die Angaben ORDER BY und INTO TEMP sind nicht zulässig.
- Die WITH CHECK OPTION-Angabe ist optional. Sie bewirkt, dass bei Neuaufnahme von Sätzen in einen View geprüft wird, ob diese der in der SELECT-Anweisung formulierten Bedingung genügen.

Beispiel: CREATE VIEW frauen AS SELECT pnr, famname, vorname
FROM person WHERE geschl='w'

```
DROP VIEW viewname
```

löscht einen View.

3. Wichtige Datentypen

Ganze Zahlen: Bigint (8 Byte), int (4 Byte), smallint (2 Byte), tinyint (1 Byte)

Dezimalzahlen: Decimal (p[, s])] oder numeric[(p[, s])]

Währung: money (8 Byte), smallmoney (4 Byte)

Gebrochene Zahlen: real, float

Datums- und Uhrzeitangaben: Date, datetime, datetime2, smalldatetime, time

Zeichenfolgen: Char(n), varchar(n), Text

Binärdaten: Binary, varbinary, image

Eine fortlaufende Zahl für eine Spalte (entspricht dem konzeptionellen Datentyp SERIAL) wird mit dem Attribut IDENTITY(1,1) vereinbart, z. B.
CREATE TABLE [Fortlaufende Zahl] (nummer int IDENTITY(1,1) NOT NULL) ON primary.
Dabei bezeichnet der erste numerische Wert in der Klammer nach Identity den Anfangswert und der zweite Wert die Schrittweite.