Transact-SQL – Data Definition Language

Stephan Karrer

Datenbankobjekte unter SQL Server

- Möglichst alle durch SQL Server verwalteten Einheiten werden als Datenbankobjekte präsentiert
- Es gibt somit eine Vielzahl von Objekten die mittels der DDL-Anweisungen (CREATE, ALTER, DROP) erzeugt, verändert und gelöscht werden können:
  - Tabellen
  - Views
  - Sequenzen
  - Indizes
  - Schemata
  - Datenbanken
  - ....

Erzeugen von Tabellen mit der CREATE-Anweisung

```
CREATE TABLE <tabellenname>
    ( <spaltenname> <spaltendefinition>
      [,<spaltenname> <spaltendefinition>]    )
```

```
CREATE TABLE cust_sample (
   cust_id             int      PRIMARY KEY,
   cust_name          varchar(50),
   cust_address          varchar(50),
   cust_credit_limit   money,
   CONSTRAINT chk_id CHECK (cust_id BETWEEN 0 and 10000 )
 );
```

Spezialfälle

```
CREATE TABLE T  ( c1 int IDENTITY (10, 1),
                  c2 varchar(20),
                  c3 AS 'Computed' + c2 PERSISTED,
                  c4 int DEFAULT SYSDATETIME(),
                  c5 int NULL,
                  c6 datetime NOT NULL);
```

- Identity-Spaltenc werden zur Schlüsselgenerierung benutzt, Angabe des Startwerts und der Schrittweite ist möglich
- Spalten können Default-Werte bekommen
- NULL-Werte können erlaubt (Standard) bzw. verhindert werden
- Berechnete Spalten können definiert werden mit Angabe, ob Ergebnis zu speichern oder nicht

Constraints: Bedingungen auf Tabellen- bzw. Spalten-Ebene

Folgende Constraints sind zulässig:

- NOT NULL: erlaubt keine NULL-Werte
- UNIQUE: erlaubt nur eindeutige oder NULL-Werte
- PRIMARY KEY: Kombination aus NOT NULL und UNIQUE
- FOREIGN KEY: legt eine Fremdschlüsselbeziehung fest
- CHECK: gibt eine/mehrere Bedingung(en) an, die erfüllt sein müssen

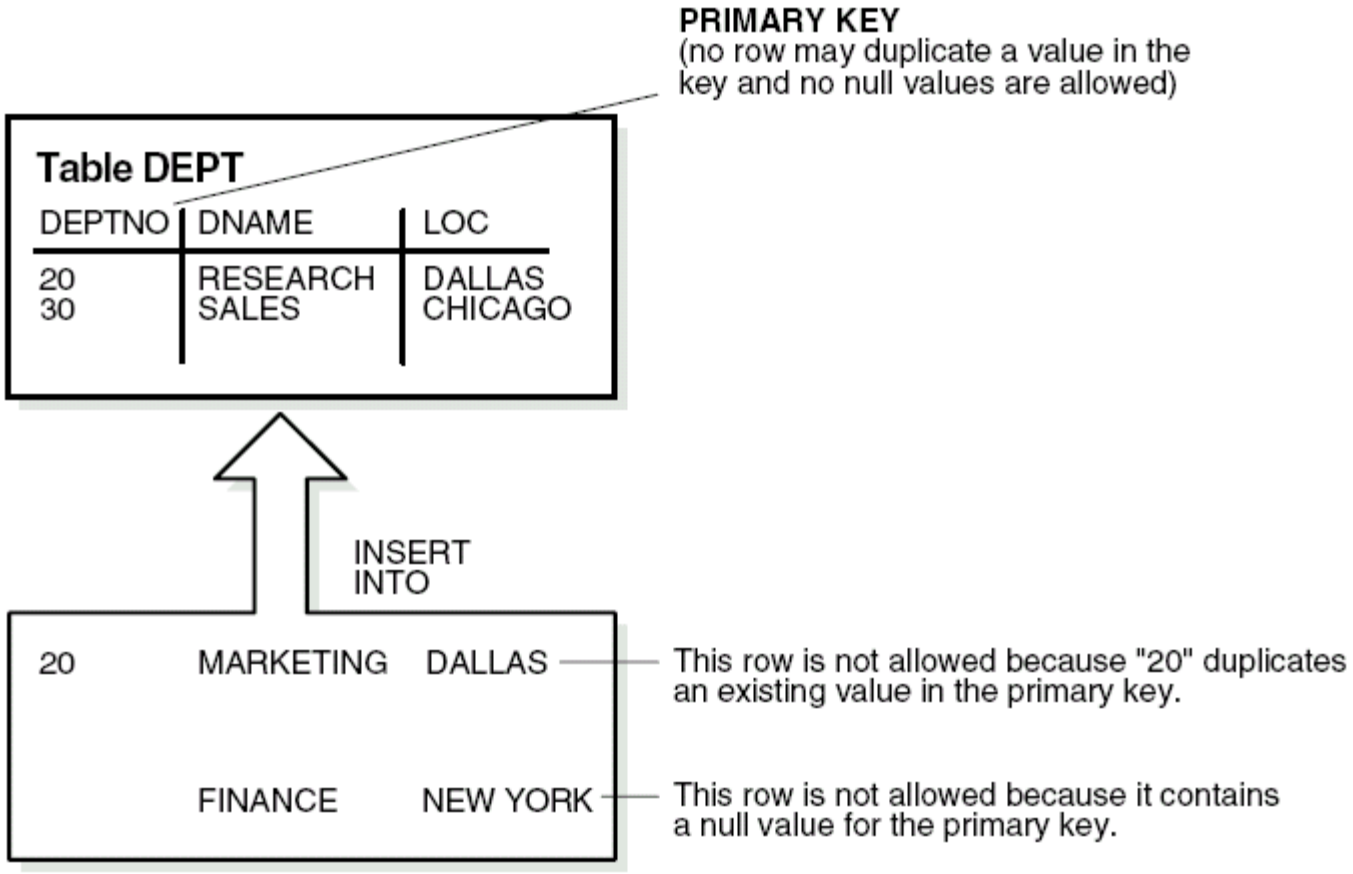
Constraints können entweder beim Anlegen mit CREATE TABLE
oder nachträglich über ALTER TABLE gesetzt werden.

Erzeugen von Tabellen mit der CREATE-Anweisung

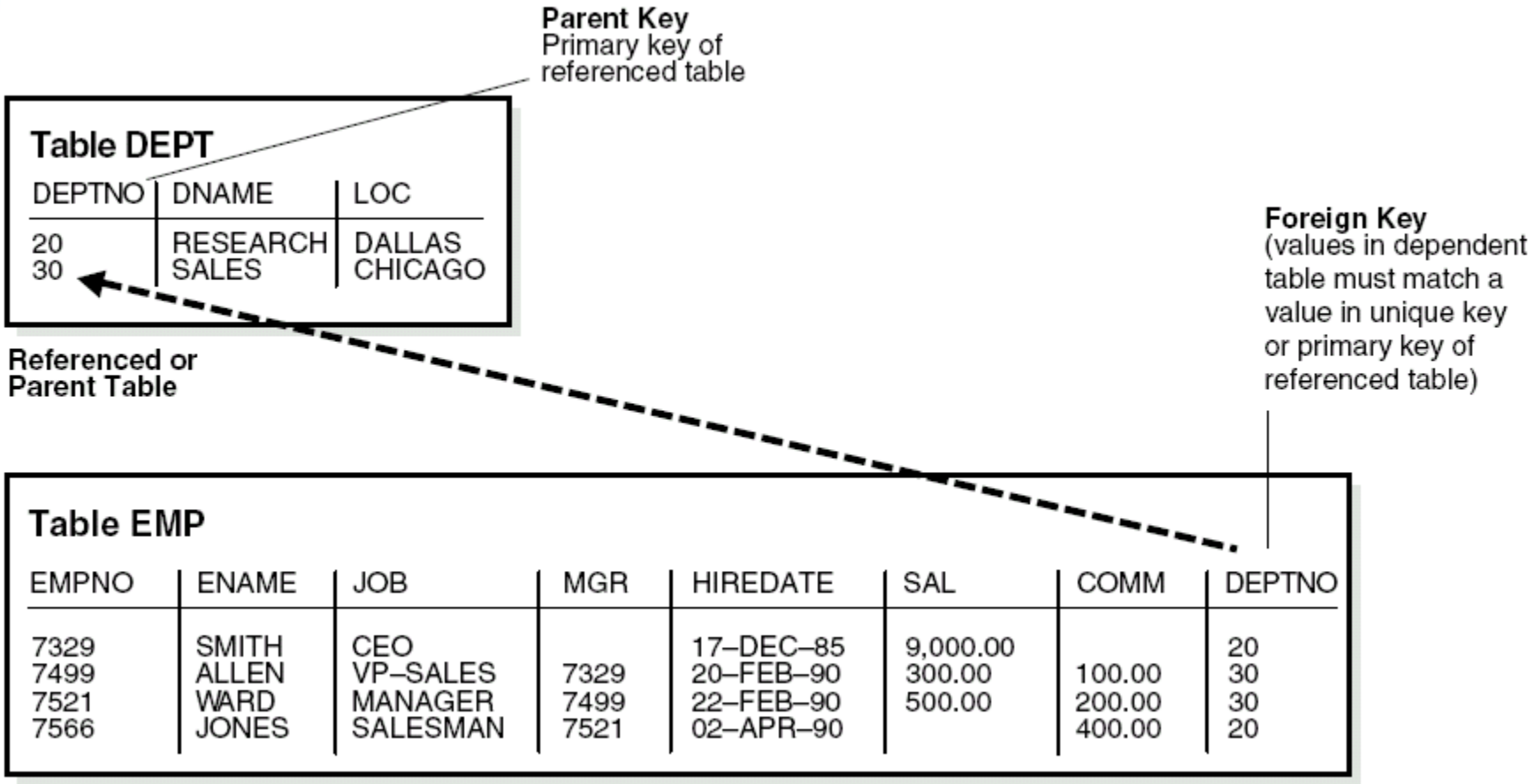
```
CREATE TABLE cust_sample (
   cust_id              int,
   cust_name            varchar(50) CHECK (len(cust_name)>2),
   cust_address         varchar(50),
   cust_credit_limit    money,
   CONSTRAINT chk_id CHECK (cust_id BETWEEN 0 and 10000 ),
   CONSTRAINT prk_id PRIMARY KEY (cust_id)
 );
```

- Primary-Key und Check-Constraints können Inline oder am Ende (offline) angegeben werden.
  Inline wird automatisch ein Name vergeben, offline ist der Name setzbar

Primärschlüssel-Beziehung



Fremdschlüssel-Beziehung (referentielle Integrität)



Primär- und Fremdschlüssel festlegen

```
/* Definition auf Tabellenebene (outline) */
CREATE TABLE emp (empno int,
                  ename varchar(10),
                  deptno int,
                  CONSTRAINT pk_emp PRIMARY KEY(empno),
                  CONSTRAINT fk_deptno FOREIGN KEY(deptno)
                      REFERENCES dept(deptno)
                  );
```

```
/* Definition auf Spaltenebene (inline) */
CREATE TABLE emp (empno int PRIMARY KEY,
                  ename varchar(10),
                  deptno int FOREIGN KEY(deptno)
                      REFERENCES dept(deptno)
                  );
```

Referentielle Integrität: Verhalten bei UPDATE oder DELETE

```
CREATE TABLE order_part
      (order_nmbr      int,
      part_nmbr      int
         FOREIGN KEY REFERENCES part_sample(part_nmbr)
            ON DELETE NO ACTION,
      qty_ordered      int);
```

Folgende Optionen existieren für die ON DELETE oder ON UPDATE-Klausel:

- NO ACTION
- CASCADE
- SET NULL
- SET DEFAULT

Zusammengesetzte UNIQUE- bzw. PRIMARY KEY-Werte

```
CREATE TABLE T1    (
      C1 varchar(10) NOT NULL
     ,C2 int NOT NULL
     …
     ,CONSTRAINT PK_T1 PRIMARY KEY(C1, C2)    )
```

```
CREATE TABLE T2    (
      C1 varchar(10) NOT NULL
     ,C2 int NOT NULL
     …
     ,CONSTRAINT UK_T2 UNIQUE(C1, C2)    )
```

Beispiel für temporäre Tabelle

```
-- lokale temporäre Tabelle
CREATE TABLE #MyTempTable1 (cola INT PRIMARY KEY);
INSERT INTO #MyTempTable1 VALUES (1);

-- globale temporäre Tabelle
CREATE TABLE ##MyTempTable2 (cola INT PRIMARY KEY);
```

- Global:
  Tabelle ist temporär und für alle Sessions mit entspr. Privilegien sichtbar
- Lokal:
  nur in der aktuellen Session verfügbar
- Temporäre Tabellen werden am Ende einer Session wieder gelöscht

ALTER TABLE: Spalten hinzufügen, ändern, löschen

```
CREATE TABLE dbo.doc_exa (column_a INT) ;
GO

ALTER TABLE dbo.doc_exa ADD column_b VARCHAR(20) NULL
    CONSTRAINT exb_unique UNIQUE ;
GO

ALTER TABLE dbo.doc_exa ADD AddDate smalldatetime NULL
    CONSTRAINT AddDateDflt DEFAULT GETDATE() WITH VALUES ;
GO

ALTER TABLE dbo.doc_exa ALTER COLUMN column_a DECIMAL (5, 2) ;
GO

ALTER TABLE dbo.doc_exa DROP COLUMN column_b ;
GO
```

Constraints hinzufügen und löschen

```
CREATE TABLE dbo.doc_exz ( column_a INT, column_b INT NULL) ;
GO

INSERT INTO dbo.doc_exz (column_a)VALUES ( -7 ) ;
GO

ALTER TABLE dbo.doc_exz
           ADD CONSTRAINT col_b_def DEFAULT 50 FOR column_b ;
GO

ALTER TABLE dbo.doc_exd WITH NOCHECK
           ADD CONSTRAINT exd_check CHECK (column_a > 1) ;
GO
```

```
CREATE TABLE dbo.doc_exc ( column_a INT
                           CONSTRAINT my_constraint UNIQUE) ;
GO

ALTER TABLE dbo.doc_exc DROP CONSTRAINT my_constraint ;
GO
```

Constraints können deaktiviert/aktiviert werden

```
CREATE TABLE dbo.cnst_example
(id INT NOT NULL,
 name VARCHAR(10) NOT NULL,
 salary MONEY NOT NULL
    CONSTRAINT salary_cap CHECK (salary < 100000)
);

-- Valid inserts
INSERT INTO dbo.cnst_example VALUES (1,'Joe Brown',65000);
INSERT INTO dbo.cnst_example VALUES (2,'Mary Smith',75000);

-- This insert violates the constraint.
INSERT INTO dbo.cnst_example VALUES (3,'Pat Jones',105000);

-- Disable the constraint and try again.
ALTER TABLE dbo.cnst_example NOCHECK CONSTRAINT salary_cap;
INSERT INTO dbo.cnst_example VALUES (3,'Pat Jones',105000);

-- Re-enable the constraint and try another insert; this will fail.
ALTER TABLE dbo.cnst_example CHECK CONSTRAINT salary_cap;
INSERT INTO dbo.cnst_example VALUES (4,'Eric James',110000) ;
```

TRUNCATE: Löschen aller Zeilen

```
TRUNCATE TABLE copy_emp;
```

- Entfernt alle Zeilen aus der Tabelle
- Ist effizienter als das Löschen aller Zeilen mit DELETE
- Tabellenstruktur verbleibt im Data Dictionary
- Constraints werden beachtet, sofern sie aktiviert sind
- Es ist kein Rollback möglich

DROP: Löschen von Tabellen

```
DROP TABLE list_customers ;
```

- Alle Daten und die Struktur der Tabelle werden gelöscht
- Alle noch offenen Transaktionen werden festgeschrieben
- Alle Indizes für die Tabelle werden gelöscht
- Alle Constraints werden gelöscht
- Es ist kein Rollback möglich
- Falls die Tabelle eine Fremdschlüsselbeziehung besitzt, muss zuerst die referenzierte Tabelle gelöscht werden

Sichten (Views)

EmployeeMaster-Tabelle

EmployeeID	FirstName	AddressID	ShiftID	LastName	MiddleName	SSN	▪ ▪ ▪
1	Sheri	1	1	Nowmer	E	245797967	▪ ▪ ▪
2	Derrick	2	1	Whelply	R	509647174	▪ ▪ ▪
3	Michael	3	1	Spence	C	42487730	▪ ▪ ▪
4	Maya	4	1	Gutierrez	Y	56920285	▪ ▪ ▪
5	Roberta	5	1	Damstra	B	695256908	▪ ▪ ▪

Sicht

FirstName	LastName	Description
Sheri	Nowmer	Engineering
Derrick	Whelply	Engineering
Michael	Spence	Engineering

Department-Tabelle

DepartmentID	Description	rowguid
1	Engineering	3FFD2603-EB6E-43B2-A8EF-C4F5C3064026
2	Tool Design	AE948718-D4BF-40E0-8ECD-2D9F4A0B211E
3	Sales	702C0EE3-03E6-4F95-9AB8-99F4F25921F3
4	Marketing	3E3C4476-B9EC-43CB-AA12-1E7A140A71A4
5	Purchasing	D6C63691-93B5-4F43-AD88-34B6B9A3C4A3

Eigenschaften von Views

- Views sind gespeicherte SQL-Anweisungen
- Können komplexe Abfragen verkörpern
- Unterliegen als DB-Objekte eigener Zugriffskontrolle
- Views können auf Basis anderer Views erstellt werden
- DML-Anweisungen sind mit Einschränkungen möglich
- Wesentliche Einschränkungen:
  - Views können nur in der aktuellen DB erstellt werden
  - Name des View muss sich von den Tabellennamen unterscheiden
  - Views können nur INSTEAD OF-Trigger zugewiesen werden
  - Temporäre Views oder Views basierend auf temporären Tabellen sind nicht möglich

Erzeugen, Nutzen, Ersetzen und Löschen von Views

```
CREATE VIEW dept_sum_vu (name, minsal, maxsal, avgsal)
AS SELECT   d.department_name, MIN(e.salary),
            MAX(e.salary), AVG(e.salary)
   FROM     employees e JOIN departments d
   ON       (e.department_id = d.department_id)
   GROUP BY d.department_name;

SELECT * FROM dept_sum_view;

ALTER VIEW dept_sum_vu AS SELECT * FROM employees; --!!!!

DROP VIEW dept_sum_vu;
```

- Views können beliebig komplexe Queries verkörpern

Aktualisieren von Daten über Views

```
CREATE TABLE T1 ( column_1 int, column_2 varchar(30));
GO

CREATE VIEW V1 AS SELECT column_2, column_1 FROM T1;
GO

INSERT INTO V1 VALUES ('Row 1',1);
GO

SELECT column_1, column_2 FROM T1;
GO

SELECT column_1, column_2 FROM V1;
GO
```

Einschränkungen bei der Datenaktualisierung (INSERT, UPDATE, DELETE)

```
-- Diese Sicht ist nicht aktualisierbar
CREATE VIEW dept_sum_vu (name, minsal, maxsal, avgsal)
AS SELECT   d.department_name, MIN(e.salary),
           MAX(e.salary), AVG(e.salary)
   FROM     employees e JOIN departments d
   ON       (e.department_id = d.department_id)
   GROUP BY d.department_name;
```

- Alle Änderungen müssen sich auf Spalten aus lediglich einer Basistabelle beziehen
- Die betroffenen Spalten des Views müssen direkt aus die Basisspalten verweisen (keine Ableitung via Berechnung oder Aggregatfunktionen)
- Die betroffenen Spalten dürfen nicht von GROUP BY-, HAVING- oder DISTINCT-Klauseln betroffen sein
- Für die nicht sichtbaren Spalten müssen default-Werte existieren oder aber NULL-Werte erlaubt sein

View-Anomalie

```
CREATE VIEW clerk AS
    SELECT employee_id, last_name, department_id, job_id
        FROM employees
        WHERE job_id = 'PU_CLERK'
           or job_id = 'SH_CLERK'
           or job_id = 'ST_CLERK';

UPDATE clerk SET job_id = 'PU_MAN' WHERE employee_id = 118;

CREATE VIEW clerk AS
    SELECT employee_id, last_name, department_id, job_id
        FROM employees
        WHERE job_id = 'PU_CLERK' or job_id = 'SH_CLERK'
           or job_id = 'ST_CLERK'
        WITH CHECK OPTION ;
```

- Ein erfolgreich eingefügter oder veränderter Datensatz ist möglicherweise über den View nicht sichtbar!
- WITH CHECK OPTION verhindert diese Anomalie

CREATE/ALTER: WITH SCHEMABINDING-Option

- Stellt sicher, dass eine Veränderung der Struktur der Basistabellen via DDL nicht den View betrifft
- Bindet die Sicht an das Schema der Basistabellen

```
CREATE VIEW V1 WITH SCHEMABINDING
    AS SELECT ColA, ColB, ColC * (ColD + 10) AS ExpCol
            FROM dbo.TableT
```

Löschen von Views

```
USE AdventureWorks ;
GO

IF OBJECT_ID ('dbo.Reorder', 'V') IS NOT NULL
       DROP VIEW dbo.Reorder ;
GO
```

Informationen zu Views

- Informationen zu Views
  - sys.views
  - sys.columns
  - sp_helptext
- Abhängigkeiten von Views
  - sys.sql_expression_dependencies
  - sys.dm_sql_referenced_entities
  - sys.dm_sql_referencing_entities

Erzeugung und Löschen eines Index

SQL Server erzeugt automatisch einen Index für Primärschlüssel- Spalten bei der Tabellenerzeugung.

Der Benutzer kann zusätzliche Indizes auf Spalten erzeugen

```
CREATE INDEX  emp_last_name_idx
ON            employees(last_name);
```

```
DROP INDEX emp_last_name_idx;
```

Richtlinien für die Erzeugung eines Index

	Create an index when:
✓	A column contains a wide range of values
✓	A column contains a large number of null values
✓	One or more columns are frequently used together in a `WHERE` clause or a join condition
✓	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table
	Do not create an index when:
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression