

Thomas Studer
Relationale Datenbanken:
Von den theoretischen Grundlagen
zu Anwendungen mit PostgreSQL
2. Auflage
Springer Vieweg, 2019
ISBN 978-3-662-58975-5

Dieser Foliensatz darf frei verwendet werden unter der Bedingung, dass diese Titelfolie nicht entfernt wird.

Datenbanken



SQL zur Datendefinition und -manipulation

Thomas Studer

Institut für Informatik
Universität Bern

INSERT

Gegebenes Schema:

```
Autos = (Marke, Farbe, Baujahr, FahrerId) .
```

```
INSERT INTO Autos  
VALUES ('Audi', 'silber', 2008, 3)
```

```
INSERT INTO Autos  
VALUES ('Skoda', 'silber', 2009, Null)
```

```
INSERT INTO Autos (Baujahr, Farbe, Marke)  
VALUES (2009, 'silber', 'Skoda')
```

INSERT mehrere Tupel

```
INSERT INTO Autos
VALUES ('Audi', 'silber', 2008, 3) ,
      ('Skoda', 'silber', 2009, Null)
```

fügt beide Tupel in die Relation Autos ein.

INSERT mit SQL Abfrage

```
INSERT INTO Autos
  SELECT 'Audi', 'silber', 2008, PersId
  FROM Personen
  WHERE Vorname = 'Eva' AND Nachname = 'Meier'
```

fügt das Tupel

```
( 'Audi', 'silber', 2008, 3 )
```

in die Relation Autos ein, wenn Eva Meier die PersId 3 hat.

Sequenzen für Primärschlüssel

```
CREATE SEQUENCE PersIdSequence START 10
```

```
SELECT nextval('PersIdSequence')
```

```
INSERT INTO Personen
```

```
VALUES ( nextval('PersIdSequence'),  
        'Bob',  
        'Müller' )
```

```
DROP SEQUENCE PersIdSequence
```

DELETE

Eine *Löschanweisung* in SQL hat folgende Form:

```
DELETE FROM <Relation>
WHERE <Prädikat>
```

Eine DELETE Anweisung wird in zwei Schritten ausgeführt:

- 1 Markiere zuerst alle Tupel in <Relation>, auf die <Prädikat> zutrifft.
- 2 Lösche die markierten Tupel aus der Relation <Relation>.

```
DELETE FROM Autos
WHERE FahrerId IN
  ( SELECT PersId
    FROM Personen
    WHERE Vorname = 'Eva' AND Nachname = 'Meier' )
```

Beispiel für das Zwei-Schritt-Vorgehen

<u>VaterSohn</u>	
<u>Vater</u>	<u>Sohn</u>
Bob	Tom
Bob	Tim
Tim	Rob
Tom	Ted
Tom	Rik
Ted	Nik

```
DELETE FROM VaterSohn
WHERE Vater IN
( SELECT Sohn
  FROM VaterSohn )
```

<u>VaterSohn</u>	
<u>Vater</u>	<u>Sohn</u>
Bob	Tom
Bob	Tim

UPDATE

```
UPDATE Autos  
SET Farbe = 'himmelblau'  
WHERE Farbe = 'blau' AND Baujahr > 2010
```

```
UPDATE Autos  
SET Baujahr = Baujahr - 1
```

Bemerkung: UPDATE Anweisungen werden ebenfalls in 2 Schritten ausgeführt (wie DELETE).

CREATE TABLE

```
CREATE TABLE TabellenName (  
    Attribut_1 Domäne_1,  
    ...  
    Attribut_n Domäne_n )
```

Einige PostgreSQL Datentypen:

integer	Ganzzahlwert zwischen -2^{31} und $2^{31} - 1$
serial	Ganzzahlwert mit Autoinkrement
boolean	Boolscher Wert
char	Einzelnes Zeichen
char(<i>x</i>)	Zeichenkette mit der Länge <i>x</i> , gegebenenfalls wird sie mit Leerzeichen aufgefüllt
varchar(<i>x</i>)	Zeichenkette mit <i>maximaler</i> Länge <i>x</i>
text	Zeichenkette mit beliebiger Länge

DEFAULT

```
CREATE TABLE Autos (  
    Marke varchar(10),  
    Farbe varchar(10) DEFAULT 'schwarz' )
```

Damit fügt die Anweisung

```
INSERT INTO Autos (Marke) VALUES ('Audi')
```

das Tupel ('Audi', 'schwarz') in die Tabelle Autos ein.

Nach Ausführen des Statements

```
INSERT INTO Autos VALUES ('VW', null)
```

hat die Tabelle Autos folgende Form:

Autos	
Marke	Farbe
Audi	schwarz
VW	-

SERIAL

```
CREATE TABLE Personen (  
    PersId integer DEFAULT nextval('PersIdSequence'),  
    Vorname varchar(10),  
    Nachname varchar(10)    )
```

Wir können Bob Müller nun ganz einfach mit

```
INSERT INTO Personen (Vorname,Nachname)  
VALUES ('Bob','Müller')
```

Das Statement

```
CREATE TABLE TabellenName (  
    AttributName serial    )
```

ist äquivalent zu

```
CREATE SEQUENCE TabellenName_AttributName_seq;  
CREATE TABLE TabellenName (  
    AttributName integer DEFAULT  
        nextval('TabelleName_AttributName_seq')  
);
```

Constraints

```
CREATE TABLE TabellenName (  
    Attribut_1 Domäne_1,  
    ...  
    Attribut_n Domäne_n,  
    Constraint_1,  
    ...  
    Constraint_m )
```

Eine Ausnahme bilden NOT NULL Constraints, welche wir direkt hinter die entsprechenden Attribute schreiben.

Beispiel

```
CREATE TABLE Autos (  
    Marke varchar(10),  
    Farbe varchar(10),  
    Baujahr integer NOT NULL,  
    FahrerId integer,  
    PRIMARY KEY (Marke, Farbe) )
```

```
CREATE TABLE Personen (  
    PersId integer PRIMARY KEY,  
    Vorname varchar(10),  
    Nachname varchar(10) )
```

Fremdschlüssel

```
CREATE TABLE Autos (  
    Marke varchar(10),  
    Farbe varchar(10),  
    Baujahr integer NOT NULL,  
    FahrerId integer,  
    PRIMARY KEY (Marke, Farbe),  
    FOREIGN KEY (FahrerId) REFERENCES Personen )
```

```
CREATE TABLE Autos (  
    Marke varchar(10),  
    Farbe varchar(10),  
    Baujahr integer NOT NULL,  
    FahrerId integer REFERENCES Personen ,  
    PRIMARY KEY (Marke, Farbe) )
```

UNIQUE

```
CREATE TABLE Personen (  
    PersId integer PRIMARY KEY,  
    Vorname varchar(10),  
    Nachname varchar(10),  
    UNIQUE (Vorname, Nachname) )
```

```
CREATE TABLE Autos (  
    Marke varchar(10),  
    Farbe varchar(10),  
    Baujahr integer NOT NULL,  
    FahrerVorname varchar(10),  
    FahrerNachname varchar(10),  
    PRIMARY KEY (Marke, Farbe),  
    FOREIGN KEY (FahrerVorname, FahrerNachname)  
    REFERENCES Personen(Vorname, Nachname) )
```

Fremdschlüssel

```
CREATE TABLE S (  
  Id integer PRIMARY KEY )
```

<u>S</u>
Id
1
2

```
DELETE FROM S  
WHERE Id = 1
```

```
CREATE TABLE R (  
  Fr integer REFERENCES S )
```

<u>R</u>
Fr
1
2

```
UPDATE S  
SET Id = 3  
WHERE Id = 1
```

Fehlermeldung:

update or delete on table "S" violates foreign key constraint.

Kaskadierung bei DELETE

```
CREATE TABLE R (  
    Fr integer REFERENCES S ON DELETE CASCADE )
```

```
DELETE FROM S  
WHERE Id = 1
```

<u>R</u>	<u>S</u>
Fr	Id
<u>2</u>	<u>2</u>

Kaskadierung bei UPDATE

```
CREATE TABLE R (  
    Fr integer REFERENCES S ON UPDATE CASCADE )
```

```
UPDATE S  
SET Id = 3  
WHERE Id = 1
```

<u>R</u>	<u>S</u>
Fr	Id
3	3
2	2

Kaskade

```
CREATE TABLE S (  
    Id integer PRIMARY KEY )
```

```
CREATE TABLE R (  
    RId integer PRIMARY KEY,  
    FrS integer REFERENCES S ON DELETE CASCADE)
```

```
CREATE TABLE Q (  
    FrR integer REFERENCES R ON DELETE CASCADE)
```

<u>Q</u>	<u>R</u>	<u>S</u>
FrR	RId FrS	Id
A	A 1	1
B	B 2	2

Kasade 2

Nach Ausführen der Anweisung

```
DELETE FROM S
WHERE Id = 1
```

erhalten wir folgende Instanzen:

<u>Q</u>
<u>FrR</u>
B

<u>R</u>
<u>RId FrS</u>
B 2

<u>S</u>
<u>Id</u>
2

SET NULL

```
CREATE TABLE S (  
    Id integer PRIMARY KEY )
```

```
CREATE TABLE R (  
    RId integer PRIMARY KEY,  
    Fr integer REFERENCES S ON DELETE SET NULL  
                        ON UPDATE SET NULL )
```

<u>R</u>		<u>S</u>
<u>RId</u>	<u>Fr</u>	<u>Id</u>
A	1	1
B	2	2
C	3	3

SET NULL 2

```
DELETE FROM S
WHERE Id = 1
```

R		S
<u>RId</u>	<u>Fr</u>	<u>Id</u>
A	-	2
B	2	3
C	3	

```
UPDATE S
SET Id = 4
WHERE Id = 2
```

R		S
<u>RId</u>	<u>Fr</u>	<u>Id</u>
A	-	4
B	-	3
C	3	

CHECK Constraints

```
CREATE TABLE Autos (  
    Marke varchar(10),  
    Farbe varchar(10),  
    Baujahr integer,  
    FahrerId integer,  
    CHECK ( Farbe IN ('blau', 'rot')) )
```

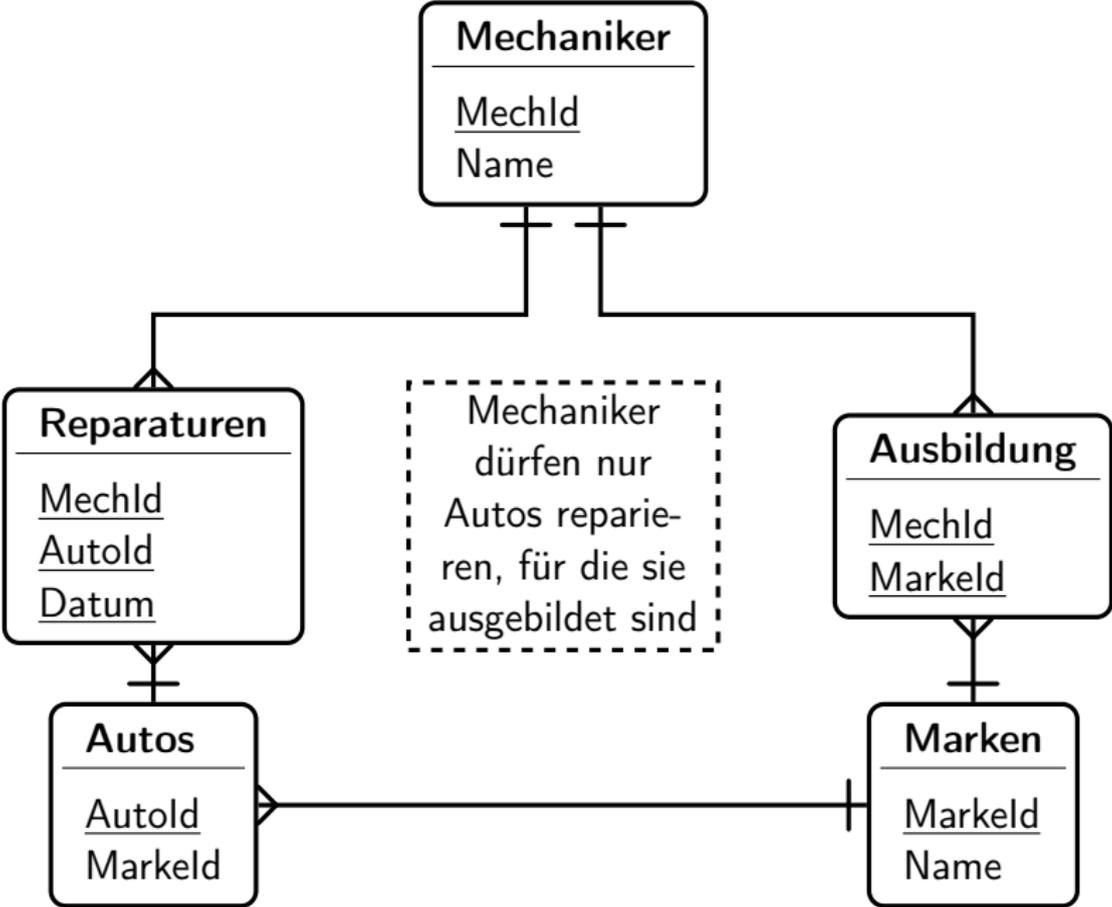
```
CREATE TABLE Autos (  
    Marke varchar(10),  
    Farbe varchar(10),  
    Mietbeginn integer,  
    Mietende integer,  
    FahrerId integer,  
    CHECK (Mietbeginn < Mietende) )
```

Beispiel

Anforderungen:

Eine Datenbank soll Autos und ihre Automarken verwalten. Zusätzlich werden noch Mechaniker verwaltet. Es wird abgespeichert, welche Mechaniker für welche Autos ausgebildet sind. Weiter wird abgespeichert welche Mechaniker wann welche Autos repariert haben. Ein Mechaniker kann ein Auto mehrfach (an verschiedenen Daten) reparieren. Er darf aber nur Autos reparieren, für deren Marken er ausgebildet wurde.

DB Schema



Tabellen

```
CREATE TABLE Mechaniker (  
    MechId integer PRIMARY KEY,  
    Name varchar(10) );
```

```
CREATE TABLE Marken (  
    MarkeId integer PRIMARY KEY,  
    Name varchar(10) );
```

```
CREATE TABLE Autos (  
    AutoId integer PRIMARY KEY,  
    MarkeId integer NOT NULL REFERENCES Marken );
```

```
CREATE TABLE Ausbildung (  
    MechId integer REFERENCES Mechaniker,  
    MarkeId integer REFERENCES Marken,  
    PRIMARY KEY (MechId, MarkeId) );
```

Tabellen 2

```
CREATE TABLE Reparaturen (  
    MechId integer REFERENCES Mechaniker,  
    AutoId integer REFERENCES Autos,  
    Datum integer,  
    PRIMARY KEY (MechId, AutoId, Datum) )
```

Gerne würden wir auch folgenden Constraint hinzufügen:

```
CHECK (  
    EXISTS (  
        SELECT *  
        FROM (Autos INNER JOIN Ausbildung  
            USING (MarkeId)) AS Tmp  
        WHERE Tmp.MechId = Reparaturen.MechId AND  
            Tmp.AutoId = Reparaturen.AutoId ) )
```

Jedoch unterstützt PostgreSQL keine Subqueries in CHECK Constraints.

ALTER TABLE

```
ALTER TABLE Marken RENAME TO Automarken
```

```
ALTER TABLE Automarken RENAME Name TO Bezeichnung
```

```
ALTER TABLE Autos ADD Baujahr INTEGER
```

```
ALTER TABLE Automarken ADD UNIQUE(Bezeichnung)
```

```
ALTER TABLE Autos ADD CHECK(Baujahr > 2010)
```

Andere Syntax für NOT NULL Constraints, da diese nur ein Attribut betreffen:

```
ALTER TABLE Mechaniker ALTER Name SET NOT NULL
```

Views

Sichten (oder *Views*) sind virtuelle Tabellen, die mittels SQL Abfragen definiert werden.

Eine Sicht kann einen Ausschnitt aus einer Tabelle präsentieren, aber auch einen Verbund von mehreren Tabellen. Somit können Views:

- 1 komplexe Daten so strukturieren, dass sie einfach lesbar sind,
- 2 den Zugriff auf Daten einschränken, so dass nur ein Teil der Daten lesbar ist (anstelle von ganzen Tabellen),
- 3 Daten aus mehreren Tabellen zusammenfassen, um Reports zu erzeugen.

CREATE VIEW

```
CREATE VIEW AlteAutos AS
  SELECT *
  FROM Autos
  WHERE Baujahr <= ALL (
    SELECT Baujahr
    FROM Autos)
```

```
SELECT Vorname, Nachname
FROM Personen INNER JOIN AlteAutos ON (PersId = FahrerId)
```

Materialisierte Views

Materialisierte Views sind Views, die nicht bei jeder Verwendung neu berechnet werden, sondern nur einmal zum Zeitpunkt ihrer Kreation.

Dies bedeutet natürlich einen Performancegewinn bei Abfragen, da auf das vorberechnete Resultat der View zurückgegriffen werden kann.

Dafür können materialisierte Views temporär nicht-aktuelle Daten enthalten.

CREATE MATERIALIZED VIEW

```
CREATE MATERIALIZED VIEW AlteAutos AS
  SELECT *
  FROM Autos
  WHERE Baujahr <= ALL (
    SELECT Baujahr
    FROM Autos)
```

```
INSERT INTO Autos VALUES ('Audi', 'silber', 2008, 3)
```

```
REFRESH MATERIALIZED VIEW AlteAutos
```