Primär- und Fremdschlüssel

Michael Dienert

10. Oktober 2017

1 Primärschlüssel

In der Theorie der Entity-Relationship-Modellierung (ERM) heisst es, dass jede Entität einen eindeutigen **Primärschlüssel** besitzen muss.
In der Sprache SQL gibt es hierzu passend die Angabe `PRIMARY KEY`. Hier ein Beispiel für die Verwendung von `PRIMARY KEY`:

```
CREATE TABLE ort(
   ort_id INTEGER NOT NULL,
   ort TEXT,
   PRIMARY KEY (ort_id)
);
```

Das darf man auch so schreiben:

```
CREATE TABLE ort(
   ort_id INTEGER PRIMARY KEY,
   ort TEXT
);
```

Die erste Schreibweise ist universeller, da man damit auch sog. *zusammengesetzte Schlüssel* erzeugen kann: alle Attribute, die den zusammengesezten Schlüssel bilden, werden zwischen den runden Klammern aufgeführt. Hier wieder ein Beispiel:

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

Die Angabe von `PRIMARY KEY` in SQL ist aber nur ein sog. **Constraint**, was auf Deutsch nichts anderes heisst wie 'Einschränkung'.

> Die Angabe von `PRIMARY KEY (nummer)` stellt die folgenden beiden Einschränkungen her:
>
> 1. '`nummer`' muss einen Wert haben, d.h. das Feld darf **nicht leer** sein
> 2. '`nummer`' muss **eindeutig** sein, d.h. es darf keine doppelten Werte geben
>
> Das ist alles!
> Keinesfalls werden mit `PRIMARY KEY` irgendwelche Beziehungen zu Schlüsseln in anderen Tabellen aufgebaut!

1.1 Aufgabe

1.1.1 Vorarbeiten

Für die folgenden Aufgaben soll eine Datei mit sql-Befehlen erstellt werden. Als Editor soll das Programm **gedit** verwendet werden. Dieses wird von einem Terminal innerhalb der grafischen Oberfläche (X11) so gestartet:

```
gedit schluessel.sql &
```

Ist die Datei gespeichert, kann man sie mit folgendem Kommando von der Shell aus zum DB-Server schicken:

```
psql -f schluessel.sql postgres
```

Vorher muss man sich natuerlich wieder selbst als berechtigten Datenbankbenuzter anlegen (`su postgres .... createuser <meinname> exit`).
Damit die Datenbank bei jedem Einspielen komplett neu angelegt wird, soll der Anfang der Datei so aussehen:

```
DROP DATABASE schluessel;
CREATE DATABASE schluessel;
\connect schluessel

--hier folgen jetzt die CREATE-befehle:
```

1.1.2 Tabellen erzeugen

Erzeuge nun mit dem sql-Befehl `CREATE TABLE` die Tabellen `'ort_mit'` und `'ort_ohne'` einmal **mit** und einmal **ohne** Angabe von PRIMARY KEY. Beispiele für `CREATE TABLE` stehen oben.

1.1.3 PRIMARY KEY testen

Versuche nun mit dem sql-Befehl `INSERT INTO TABLE` jeweils Datensätze mit fehlender oder doppelter `ort_id` einzufügen. Die Syntax für den `INSERT`-Befehl sieht so aus:
Wenn man alle Attribute in der Reihenfolge der Deklaration eingeben möchte:

```
INSERT INTO ort_mit VALUES (10,'freiburg');
```

Wenn man nur einen Teil der Attribute eingeben möchte, muss man angeben, welche:

```
INSERT INTO ort_mit (ort) VALUES ('freiburg');
```

Erweitere die sql-Datei noch um eine dritte Tabellendefinition:

```
CREATE TABLE ort(
   ort_id INTEGER UNIQUE NOT NULL,
   ort TEXT
);
```

Unterscheidet sich die Kombination von `UNIQUE` und `NOT NULL` von `PRIMARY KEY` ?

2 Fremdschlüssel und referentielle Integrität

Entsprechend zu den Fremdschlüsseln in einem ERM gibt es in SQL auch die Angabe 'FOREIGN KEY'.

Auch diese Angabe stellt wieder nur eine Einschränkung (Constraint) für die Werte da, die in einer Spalte stehen dürfen.

> Ein `FOREIGN KEY` Constraint legt fest, dass **alle** Werte in der Fremdschlüssel-Spalte einer Tabelle auch in einer Primärschlüssel-Spalte einer anderen Tabelle vorhanden sein müssen.
> Diese Bedingung wird **Referentielle Integrität** zwischen zwei Tabellen genannt.
> Referentielle Integrität bedeutet:
>
> 1. Man kann keinen Fremdschlüssel-Wert vergeben, der nicht schon in der zugehörigen Tabelle als Primärschlüssel auftaucht.
> 2. Man kann keinen Primärschlüssel löschen, wenn es hierzu in einer anderen Tabelle noch einen Fremdschlüssel gibt.

Hier ein Beispiel für die Vergabe eines Fremdschlüssel-Constraints:

```
CREATE TABLE plz_ort(
   plz text,
   ort_schluessel INTEGER NOT NULL,
   PRIMARY KEY (plz),
   FOREIGN KEY (ort_schluessel) REFERENCES ort (ort_id)
);
```

2.1 Aufgaben

Erweitere die Datei aus Kap. 1 um die Definition einer Postleitzahl-Tabelle ohne Fremdschlüssel:

```
CREATE TABLE plz_ort(
   plz text,
   ort_schluessel INTEGER,
   PRIMARY KEY (plz)
);
```

Füge einige Datensätze ein, bei denen `ort_schluessel` auf einen Wert von `ort_id` in der Ortetabelle zeigen.

Versuche anschliessend, die Datensätze aus der Ortetabelle zu löschen oder auf einen neuen Wert zu setzen.

Das Ändern auf einen neuen Wert geht so:

```
UPDATE ort_mit SET ort_id=666 WHERE ort_id=10;
```

Wiederhole das Löschen/Ändern in der Ortetabelle, nachdem Du in der Postleitzahlentabelle ein FOREIGN KEY - Constraint vergeben hast:

```
CREATE TABLE plz_ort(
   plz text,
   ort_schluessel INTEGER,
   PRIMARY KEY (plz),
   FOREIGN KEY (ort_schluessel) REFERENCES ort (ort_id)
);
```

3 Lösch- und Änderungsweitergabe

3.1 Aufgabe

Erweitere die Tabellendefinition von `plz_ort`:

```
CREATE TABLE plz_ort(
   plz text,
   ort_schluessel INTEGER,
   PRIMARY KEY (plz),
   FOREIGN KEY (ort_schluessel) REFERENCES ort (ort_id)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
```

Was passiert nun, wenn ich in der Tabelle ort einen Datensatz lösche, von dem noch ein Datensatz mit Fremdschlüssel abhängt? Was passiert, wenn ich einen Primärschlüssel ändere?