SQLite Foreign Keys erzwingen

Wechseln zu:Navigation, Suche

Inhaltsverzeichnis

1 Problem
2 Lösung
3 Online Generator für Fremdschlüssel
- 3.1 Quellen

1 Problem

SQLite unterstützt standartmäßig keine FOREIGN KEYS (Fremdschlüssel). Man kann diese zwar bei erstellen von Tabellen definieren, jedoch werden diese, wie gesagt, ignoriert.

2 Lösung

Es ist möglich, SQLite dazu zu zwingen, Fremdschlüssel zu akzeptieren. Um das zu erreichen werden TRIGGER verwendet. TRIGGER werden zum Einfügen, Löschen oder Ändern von Referenzdaten eingesetzt. Der TRIGGER wird ausgeführt, entweder bevor Änderungen an der Tabelle vorgenommen werden oder danach. TRIGGER können selbst Datensätze einfügen, ändern oder löschen und durch seine Ausführung weitere TRIGGER auslösen. Nun zum Beispiel: Nehmen wir an wie haben 2 Tabellen-Deklarationen:

```
CREATE TABLE kontakt (
  id INTEGER NOT NULL PRIMARY KEY
);

CREATE TABLE termin (
  id INTEGER NOT NULL PRIMARY KEY,
  kontakt_id INTEGER NOT NULL
         CONSTRAINT fk_kontakt_id REFERENCES a(id) ON DELETE CASCADE
);
```

Die Tabelle termin hat einen Fremdschlüssel, der auf die Primär-Schlüssel-Spalte in der kontakt-Tabelle verweist. Obwohl SQLite diese Syntax unterstützt, wird sie einfach ignoriert. Wenn man also eine Referenz erzwingen will, muss man TRIGGER erstellen, die das erledigen. Jede Bedingung muss einen eigenen TRIGGER haben, also einen für INSERT, DELETE und UPDATE. Der INSERT-TRIGGER sieht folgendermaßen aus:

```
CREATE TRIGGER fki_termin_kontakt_id
BEFORE INSERT ON termin
FOR EACH ROW BEGIN
  SELECT CASE
     WHEN ((SELECT id FROM kontakt WHERE id = NEW.kontakt_id) IS NULL)
     THEN RAISE(ABORT, ‘insert on table "termin" violates foreign key‘ ||
‘constraint "fk_kontakt_id"’)
  END;
END;
```

Wenn man die Fremdschlüssel-Spalte als NOT NULL definiert haben will/sollte, muss der INSERT-TRIGGER SELECT CASE ein wenig anders aussehen:

```
CREATE TRIGGER fki_termin_kontakt_id
BEFORE INSERT ON termin
FOR EACH ROW BEGIN
   SELECT CASE
     WHEN ((new.kontakt_id IS NOT NULL)
           AND ((SELECT id FROM kontakt WHERE id = new.kontakt_id) IS
NULL))
     THEN RAISE(ABORT, ‘insert on table "termin" violates foreign key‘ ||
‘constraint "fk_kontakt_id"’)
  END;
END;
```

Das UPDATE-Statement ist fast identisch, wenn die Fremdschlüssel-Spalte NOT NULL ist:

```
CREATE TRIGGER fku_termin_kontakt_id
BEFORE UPDATE ON termin
FOR EACH ROW BEGIN
   SELECT CASE
     WHEN ((SELECT id FROM kontakt WHERE id = new.kontakt_id) IS NULL))
     THEN RAISE(ABORT, ‘update on table "termin" violates foreign key‘ ||
‘constraint "fk_kontakt_id"’)
  END;
END;
```

und wenn NULL erlaubt ist:

```
CREATE TRIGGER fku_termin_kontakt_id
BEFORE UPDATE ON termin
FOR EACH ROW BEGIN
   SELECT CASE
     WHEN ((new.kontakt_id IS NOT NULL)
           AND ((SELECT id FROM kontakt WHERE id = new.kontakt_id) IS
NULL))
     THEN RAISE(ABORT, ‘update on table "termin" violates foreign key‘ ||
‘constraint "fk_kontakt_id"’)
  END;
END;
```

Der DELETE TRIGGER wird zur Primär-Sclüssel-Tabelle hinzugefügt und überwacht DELETEs auf der kontakt-Tabelle:

```
CREATE TRIGGER fkd_termin_kontakt_id
BEFORE DELETE ON kontakt
FOR EACH ROW BEGIN
  SELECT CASE
    WHEN ((SELECT kontakt_id FROM termin WHERE kontakt_id = OLD.id) IS
NOT NULL)
    THEN RAISE(ABORT, ‘delete on table "kontakt" violates foreign key‘ ||
‘constraint "fk_kontakt_id"’)
  END;
END;
```

Dieser TRIGGER verhindert Löschvorgänge in der kontakt-Tabelle, wenn es Fremdschlüssel gibt, die auf die termin-Tabelle verweisen. Eine weitere, etwas übersichtlichere Lösung:

```
CREATE TABLE departments (
    dept_id   INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    dept_name TEXT NOT NULL UNIQUE
);

CREATE TABLE employees (
    emp_id   INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    emp_name TEXT NOT NULL UNIQUE,
    dept_id  INTEGER NOT NULL,
    CONSTRAINT fkey_dept_id FOREIGN KEY (dept_id) REFERENCES departments
(dept_id)
);
```

und die zugehörigen TRIGGER:

```
CREATE TRIGGER fkey_dept_id_ins BEFORE INSERT ON employees
  FOR EACH ROW
    BEGIN
      SELECT
        RAISE(ABORT,
             'insert on table "employees" violates foreign key constraint
"fkey_dept_id"')
          WHERE (SELECT dept_id FROM departments
                   WHERE dept_id = new.dept_id) IS NULL;
    END;

CREATE TRIGGER fkey_dept_id_upd BEFORE UPDATE ON employees
  FOR EACH ROW
    BEGIN
      SELECT
        RAISE(ABORT,
             'update on table "employees" violates foreign key constraint
"fkey_dept_id"')
          WHERE (SELECT dept_id FROM departments
                   WHERE dept_id = new.dept_id) IS NULL;
    END;

CREATE TRIGGER fkey_dept_id_del BEFORE DELETE ON departments
  FOR EACH ROW
    BEGIN
      SELECT
        RAISE(ABORT,
             'delete on table "departments" violates foreign key
constraint "fkey_dept_id" on "employees"')
          WHERE (SELECT dept_id from employees
                   WHERE dept_id = old.dept_id) IS NOT NULL;
    END;
```

Die INSERT-Befehle und SELECT-Ausgaben sehen dann so aus:

```
INSERT INTO departments (dept_name) VALUES ('Marketing');
INSERT INTO departments (dept_name) VALUES ('MIS');
INSERT INTO departments (dept_name) VALUES ('Accounting');

SELECT * FROM departments;

1|Marketing
2|MIS
3|Accounting

INSERT INTO employees (emp_name, dept_id) VALUES ('John', 1);
INSERT INTO employees (emp_name, dept_id) VALUES ('Tim', 2);
INSERT INTO employees (emp_name, dept_id) VALUES ('Gene', 3);

SELECT emp_id, emp_name, dept_id FROM employees;

1|John|1
2|Tim|2
3|Gene|3

UPDATE employees set dept_id = 2 WHERE emp_id = 1;
UPDATE employees set dept_id = 3 WHERE emp_id = 2;

SELECT emp_id, emp_name, dept_id FROM employees;

1|John|2
2|Tim|3
3|Gene|3

DELETE FROM departments WHERE dept_id = 1;
DELETE FROM departments WHERE dept_id = 2;
// hier müsste nach dem 2. Statement ein Fehler auftreten, weil
Department mit ID=2 noch von einem Mitarbeiter belegt ist
// das ist völlig OK

SELECT * FROM departments;

2|MIS
3|Accounting
```

3 Online Generator für Fremdschlüssel

Für SQLite gibt es einen Online-Fremdschlüssel-Generator, der die eingegebnenen Tabellen-Deklarationen, wie im obrigen Beispiel erklärt, automatisch mitsamt dem dazugehörigen TRIGGER ausgibt. Zu finden ist der Generator unter RCS Computers.

3.1 Quellen

SQLite Homepage
Wikipedia: Datenbanktrigger
Online FOREIGN KEY Generator

Kategorien:
Glossar
Informatik