

## Modul Datenbanksysteme (DBS)

## Übung DI: Datenintegrität

**1. Selbststudium**

☞ Lesen Sie im Buch von Meier & Kaufmann (2016) die Kapitel 2.3.3 und 3.7

☞ Beantworten Sie folgende Fragen:

- ? Welche 3 Arten von strukturellen Integritätsbedingungen gibt es?  
siehe oben
- ? Wie lautet die Definition für den Begriff „Referenzielle Integrität“?  
jede Referenzierung muss wirklich existieren
- ? Welche 8 verschiedenen Arten von deklarativen Integritätsbedingungen gibt es? Zählen sie die 8 Begriffe auf.  
siehe oben
- ? Können Sie diese 8 deklarativen Integritätsbedingungen aus Kap. 3.7 zu den 3 strukturellen Integritätsbedingungen aus Kap. 2.3.3 zuordnen?  
siehe oben

**2. Referenzielle Integrität in SQL**

Informieren Sie sich auf der PostgreSQL-Webseite, wie sie bei bestehenden Tabellen referenzielle Integritätsbedingungen einfügen können (ALTER TABLE ...)

<http://www.postgresql.org/docs/9.4/static/sql-altertable.html>

Definieren Sie alle referentiellen Constraints als Primär- und Fremdschlüssel für die bestehende Uni-Datenbank auf *PostgreSQL*. Sorgen Sie zudem dafür, dass bei Änderung der Primärschlüssel alle Fremdschlüssel entsprechend aktualisiert werden: Bei Löschen falls möglich auf Null setzen, sonst löschen; bei Veränderung übernehmen.

**3. Statische Integrity Constraints in SQL**

```

ALTER TABLE tabellennamen
ADD CONSTRAINT fk_someName
FOREIGN KEY (sender)
REFERENCES auth_user(column_referenced_name)
ON UPDATE cascade
ON DELETE [set null, delete];
  
```

Setzen Sie folgende Integritätsbedingungen in SQL auf der bestehenden Uni-Datenbank auf *PostgreSQL* mit ALTER TABLE um:

- `ALTER TABLE professoren ADD CONSTRAINT constraint_name check rang in ('R3', 'R4')`  
Professoren können nur die Ränge ‚C3‘ und ‚C4‘ haben.
- `ALTER TABLE professoren ADD CONSTRAINT constraint_name UNIQUE(Raum)`  
Professoren haben Einzelbüros.
- `ALTER TABLE pruefen ADD CONSTRAINT constraint_name check Note between 1 and 5`  
Die Noten dürfen nur zwischen 1.0 und 5.0 sein.
- Die Namen der Studenten, Professoren und Assistenten dürfen nicht leer sein.  
`ALTER TABLE studenten ALTER COLUMN name SET NOT NULL`  
`ALTER TABLE professoren ALTER COLUMN name SET NOT NULL`  
`ALTER TABLE assistenten ALTER COLUMN name SET NOT NULL`

#### 4. Trigger (1)

Professoren sollen immer nur einen Rang höher kommen, aber nie degradiert werden. Der Rang eines Professors ist dabei immer C3 oder C4. Implementieren Sie dazu in der *Postgresql* den folgenden Trigger:

```
CREATE OR REPLACE FUNCTION checkDegradierung ()
RETURNS TRIGGER AS $$ BEGIN
    IF OLD.Rang IS NULL
    OR NEW.Rang = 'C3' AND OLD.RANG <> 'C4'
    OR NEW.Rang = 'C4' THEN RETURN NEW;
    ELSE RAISE EXCEPTION
        'Degradierender Rang --> %', NEW.rang;
    RETURN OLD; END IF;
END; $$ LANGUAGE 'plpgsql';

CREATE TRIGGER keineDegradierung
BEFORE UPDATE ON professoren
FOR EACH ROW EXECUTE PROCEDURE checkDegradierung();
```

Testen Sie den Trigger: Schreiben Sie ein Update Statement, welches alle Professoren die im Rang C4 sind, auf C3 degradiert, und schauen Sie sich die Response des Datenbankservers an.

#### 5. Trigger (2)

- ☞ Informieren Sie sich auf der Webseite von Postgresql, wie bzw. mit welcher Syntax Trigger-Programme implementiert werden können.
- ☞ Schreiben Sie einen Trigger in *PostgreSQL*, der prüft, ob Studenten für die Prüfungen alle Vorbedingungen erfüllen, d.h. ob sie alle Vorgänger der zu prüfenden Vorlesung mit genügender Note absolviert haben ( $\leq 3.0$ ).

## 6. Abgabe der Übung

- Ergänzen Sie ihr Dokument mit den Namen der Teammitglieder, welche zur Lösung der Aufgabe beigetragen haben.
- Erstellen Sie ein PDF mit den Lösungen zu den Aufgaben: Übung\_DI\_Gruppe\_<XY>.pdf
- Laden Sie die Datei als PDF auf ILIAS in den Briefkasten DI
- Abgabetermin: Siehe Semesterplan Detail (auf ILIAS > Organisatorisches)