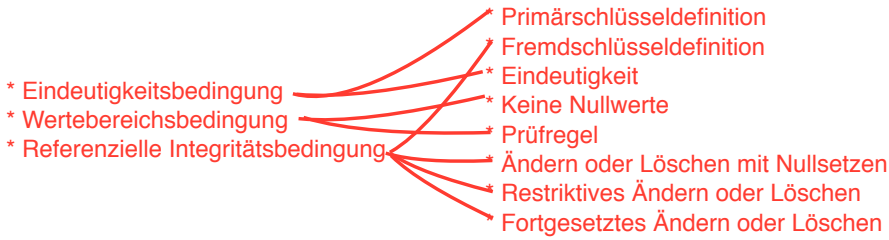
Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

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 deklariativen 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.

siehe letzte Seite

3. Statische Integrity Constraints in SQL

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

- Professoren können nur die Ränge ‚C3’ und ‚C4’ haben.
  ALTER TABLE professoren ADD CONSTRAINT constraint_name check rang in (‘R3’, ‘R4’)
- Professoren haben Einzelbüros.
  ALTER TABLE professoren ADD CONSTRAINT constraint_name UNIQUE(Raum)
- Die Noten dürfen nur zwischen 1.0 und 5.0 sein.
  ALTER TABLE pruefen ADD CONSTRAINT constraint_name check Note between 1 and 5
- 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();
```

```
ERROR:  Degradierender Rang --> C3
CONTEXT:  PL/pgSQL function checkdegradierung() line 5 at RAISE
SQL state: P0001
```

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 Postegresql, 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 (<= 3.0).

siehe nächste Folie

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)

```
CREATE OR REPLACE FUNCTION checkPruefungen()
   RETURNS TRIGGER AS $$ BEGIN

IF(SELECT COUNT(*) FROM (SELECT Note

FROM vorlesungen LEFT JOIN voraussetzen

ON vorlesungen.VorlNr = voraussetzen.Nachfolger

LEFT JOIN pruefen

ON voraussetzen.Vorgänger = pruefen.VorlNr

WHERE vorlesungen.VorlNr = NEW.VorlNr) as vornoten

WHERE Note > 3 OR Note is NULL) = 0 THEN RETURN NEW;

     ELSE RAISE EXCEPTION
          'Voraussetzungen nicht erfüllt';
     RETURN OLD; END IF;
   END; $$ LANGUAGE 'plpgsql';

CREATE TRIGGER checkPruefungen
   BEFORE INSERT ON pruefen
   FOR EACH ROW EXECUTE PROCEDURE checkPruefungen();
```

Aufgabe 2:

```
ALTER TABLE studenten ADD PRIMARY KEY (matrnr);
ALTER TABLE assistenten ADD PRIMARY KEY (persnr);
ALTER TABLE professoren ADD PRIMARY KEY (persnr);
ALTER TABLE vorlesungen ADD PRIMARY KEY (vorlnr);


ALTER TABLE hoeren ADD FOREIGN KEY (vorlnr) REFERENCES vorlesungen ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE hoeren ADD FOREIGN KEY (matrnr) REFERENCES studenten ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE pruefen ADD FOREIGN KEY (vorlnr) REFERENCES vorlesungen ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE pruefen ADD FOREIGN KEY (matrnr) REFERENCES studenten ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE pruefen ADD FOREIGN KEY (persnr) REFERENCES professoren ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE voraussetzen ADD FOREIGN KEY (vorgänger) REFERENCES vorlesungen(vorlnr) ON UPDATE CASCADE ON
DELETE CASCADE;
ALTER TABLE voraussetzen ADD FOREIGN KEY (nachfolger) REFERENCES vorlesungen(vorlnr) ON UPDATE CASCADE ON
DELETE CASCADE;

ALTER TABLE hoeren ADD PRIMARY KEY (vorlnr, matrnr);
ALTER TABLE pruefen ADD PRIMARY KEY (vorlnr, matrnr, persnr);
ALTER TABLE voraussetzen ADD PRIMARY KEY (vorgänger, nachfolger);
```