Übung 2

Aufgabe 2.1

Im Zusammenhang mit dem relationalen Datenmodell haben wir die Begriffe *Superschlüssel*, *Schlüsselkandidat* und *Primärschlüssel* kennengelernt.

a) Wie sind diese Begriffe formal definiert?

- Alle drei Begriffe bezeichnen Attribute/Attributkombinationen S, anhand derer man die Datensätze (=Tupel/Zeilen) in einer Relation eindeutig identifizieren kann.
- Formal:
  S hat Schlüsseleigenschaft wenn gilt: $\forall$ t1, t2 $\in$ r: t1 $\neq$ t2 $\Rightarrow$ t1[S] $\neq$ t2[S]
- Superschlüssel: Die Attributmenge S ist nicht unbedingt minimal, d.h. man kann ein Attribut A aus S weglassen und S\{A} hat immer noch die Schlüsseleigenschaft.
- Schlüsselkandidat: eine minimale Attributmenge, aus der man kein Attribut weglassen kann, ohne dass die Schlüsseleigenschaft verloren geht. In der Regel existieren mehrere Schlüsselkandidaten.
- Primärschlüssel: ein vom Schemadesigner ausgewählter Schlüsselkandidat. Zusätzliche Anforderungen. Attributwerte dürfen nicht NULL sein und müssen stabil sein (d.h. keine Veränderung über die Zeit).

b) Gegeben sei eine Relation

`Händler(Id, Name, Adresse, Telefonnummer, Branche, URL)`,

in der Informationen für ein Händlerverzeichnis gespeichert werden.
Welche Attribute oder Attributkombinationen sind Schlüsselkandidaten? Welcher der Schlüsselkandidaten bietet sich als Primärschlüssel an? Begründen Sie Ihre Wahl und machen Sie Ihre Annahmen über die Bedeutung und möglichen Ausprägungen der Attribute explizit.

Schlüsselkandidaten

- ID (künstlicher Schlüssel über eindeutige ID)
- Name, Adresse (Annahme: es gibt keine zwei Händler mit gleichem Namen und gleicher Adresse)
- Name, Telefonnummer (dito)
- URL (Annahme: URL zeigt nicht auf "Sammel"-Webseite für mehrere Händler)

→ ID ist einzig sinnvoller Primärschlüssel, da die anderen Attribute/Attributkombinationen sich über die Zeit ändern können (neuer Name nach Heirat, Umzug, neue Telefonnummer, neue Homepage)

Übung 2

Aufgabe 2.2

Geben Sie ein SQL-Script an, das die folgenden Tabellen anlegt:

- `person(nr, name, vorname, plz, strasse)`
- `plzort(plz, ort)`

Überlegen Sie sich geeignete Datentypen und legen Sie dabei folgende Integrity-Constraints an:

- das Feld *ort* in der Tabelle *plzort* darf nicht leer sein;
- der Wert *plz* in der Tabelle *person* muss in der Tabelle *plzort* vorkommen;
- wenn in *plzort* eine *plz* geändert wird, soll diese Änderung bei allen betroffenen *person*en automatisch nachgezogen werden;
- das Löschen eines Datensatzes aus *plzort* soll unterbunden werden, wenn in diesem *ort* noch *person*en wohnen;
- die Kombination *vorname, name* soll eindeutig sein;
- die *plz* darf nur aus Ziffern bestehen (Hinweis: SIMILAR TO[1]);

```
DROP TABLE IF EXISTS person, plzort;

CREATE TABLE plzort (
  plz VARCHAR(5) PRIMARY KEY CHECK(plz SIMILAR TO '\d{5}'),
--  Alternativ geht auch:              SIMILAR TO '[0-9]{5}'
-- Würde man die PLZ als NUMERIC definieren, so wären zwar auch nur Ziffern möglich,
-- aber PLZ, die mit 0 beginnen, könnten nicht gespeichert werden.

  ort VARCHAR(50) NOT NULL
);

CREATE TABLE person (
  nr INTEGER PRIMARY KEY,
  name VARCHAR(30),
  vorname VARCHAR(30),
  plz VARCHAR(5) REFERENCES plzort(plz) ON UPDATE CASCADE ON DELETE NO ACTION,
-- Andere Reaktionen wären:          ON UPDATE/ON DELETE SET NULL
  strasse VARCHAR(50),
  UNIQUE(name, vorname)
);
```

[1] Siehe: https://www.postgresql.org/docs/13/functions-matching.html

Übung 2

Aufgabe 2.3

Nehmen Sie an, dass die oben definierten Tabellen die folgenden Daten enthalten:

plzort

plz	ort
47805	Krefeld

person

nr	name	vorname	plz	strasse
4711	Gans	Gustav	47805	Kölner Straße 66

Geben Sie Beispiele für *INSERT*, *UPDATE* und *DELETE*-Befehle an, die aufgrund der in Aufgabe 2.2 definierten Constraints fehlschlagen. Welche Statements schlagen warum fehl?

```
-- Erstmal ein paar Datensätze einfügen
INSERT INTO plzort VALUES ('47805', 'Krefeld');
INSERT INTO person VALUES ('4711', 'Gans', 'Gustav', '47805', 'Kölner Strasse 66');
INSERT INTO person VALUES ('4712', 'Franz', 'Gustav', '47805', 'Kölner Strasse 64');

-- Jetzt ein paar nicht erlaubte INSERT, UPDATE, DELETE Statements
INSERT INTO plzort VALUES ('47453', NULL);
-- Schlägt fehl wegen NOT NULL-Constraint auf Ort

INSERT INTO person VALUES ('4713', 'Duck', 'Dagobert', '47453', 'Krefelder Strasse 11');
-- Schlägt fehl wegen FOREIGN KEY auf plzort.plz (Datensatz mit 47453 existiert nicht)

INSERT INTO person VALUES ('4711', 'Duck', 'Donald', '47805', 'Kölner Strasse 66');
-- Schlägt fehl wegen PRIMARY KEY auf person.nr

INSERT INTO person VALUES ('4714', 'Gans', 'Gustav', '47805', 'Reinarzstrasse 49');
-- Schlägt fehl wegen UNIQUE-Constraint auf (name, vorname)

UPDATE plzort SET plz = '47806’ WHERE plz = '47805'
-- Funktioniert trotz FK, weil Änderung an person propagiert wird (ON UPDATE CASCADE)

UPDATE plzort SET ort = NULL WHERE plz = '47806'
-- Schlägt fehl wegen NOT NULL-Constraint auf Ort

DELETE FROM plzort WHERE plz = '47806'
-- Schlägt fehl, weil es abhängige Datensätze in person gibt und ON DELETE NO ACTION
definiert wurde (dies ist die Standardeinstellung für Forein Key-Verletzungen
```

Aufgabe 2.4

Mit dem ALTER TABLE Befehl können Spalten zu einer Tabelle hinzugefügt oder entfernt werden:

- `ALTER TABLE table ADD COLUMN column type`
- `ALTER TABLE table DROP COLUMN column`
- `ALTER TABLE table RENAME COLUMN oldname TO newname`

Der Datentyp einer Spalte lässt sich in PostgreSQL wie folgt ändern:

- `ALTER TABLE table ALTER COLUMN column TYPE newtype [USING expression]`

Die optionale Klausel `USING expression` gibt dabei einen Ausdruck an, der die Werte vom alten Spaltentyp in den neuen Spaltentyp umwandelt. Wenn die USING-Klausel fehlt, versucht PostgreSQL, die alten Werte über implizites Casting in den neuen Datentyp zu wandeln.

a) Nehmen Sie an, dass Sie *person.plz* zunächst ungünstigerweise als *INTEGER* definiert hatten. Wie lautet der Befehl, um in PostgreSQL die Spalte *person.plz* von *INTEGER* in *VARCHAR(5)* zu ändern?

```
ALTER TABLE person
ALTER COLUMN plz TYPE VARCHAR(5)
USING CAST(plz as VARCHAR(5));
-- USING optional (ansonsten implizites Casting von Integer nach VARCHAR)
```

b) In manchen DBMS existiert kein eigener Befehl zum Ändern des Datentyps einer Spalte. Die Änderung eines Spaltentyps lässt sich dann nur über einen Umweg erreichen. Überlegen Sie, welcher Umweg das ist und geben Sie eine Folge von SQL-Statements für das obige Beispiel an.

Die Idee besteht darin, zunächst eine neue Spalte vom Zieltyp hinzuzufügen (ADD COLUMN), die Daten aus der alten Spalte mittels UPDATE mit der gewünschten Konvertierung, z.B. TO CHAR(plz, '00000'), in die neue Spalte zu kopieren, dann die alte Spalte zu löschen (DROP COLUMN) und die neue Spalte in den alten Namen umzubenennen (RENAME COLUMN).

Alternativ könnte man natürlich auch erst die alte Spalte umbenennen und die neue Spalte gleich mit dem richtigen Namen anlegen. Rest bleibt gleich.

Übung 2

Aufgabe 2.5

Geben Sie für folgende Datenmanipulationen an den obigen Tabellen SQL-Statements an:

a) Auflisten aller *nr* in *person*, bei denen mindestens eines der restlichen Felder leer ist;

```
SELECT nr
FROM person
WHERE name IS NULL OR vorname IS NULL OR plz IS NULL OR strasse IS NULL;
```

b) Anhängen des Strings 'bla' an jeden *namen* in p*erson* (Hinweis: der Operator für String-Konkatenation lautet in SQL ||);

```
update person set name=name||'bla';
```

c) Auflisten der Tupel in *person*, bei denen die *strasse* keine Hausnummer hat (Hinweis: regex mit SIMILAR TO);

```
select *
from person
where strasse NOT SIMILAR TO '%[0-9]+';
```

d) Auflisten aller *name*n und *vorname*n von *person*en zusammen mit dem *ort*, in dem sie wohnen;

```
-- Variante 1 mit JOIN .. ON ..
select name, vorname, ort
from person JOIN plzort on person.plz = plzort.plz;

-- Variante 2: NATURAL JOIN auch möglich, da das JOIN-Attribut in beiden Tabellen
-- gleich heisst (plz)
select name, vorname, ort
from person NATURAL JOIN plzort;

-- Variante 3: Old-school Notation mit JOIN-Bedingung in der WHERE-Klausel
select name, vorname, ort
from person, plzort
where person.plz = plzort.plz;
```