Online-Marktplatz

Mehmet Özer 7206358

Tobias Barthold 7209370

Yusuf Bas 7209349

```
5
6       DROP TABLE unternehmen CASCADE CONSTRAINTS  PURGE;
7       DROP TABLE kategorie CASCADE CONSTRAINTS  PURGE;
8       DROP TABLE einkaufswagen CASCADE CONSTRAINTS  PURGE;
9       DROP TABLE nutzer CASCADE CONSTRAINTS  PURGE;
0       DROP TABLE produkt CASCADE CONSTRAINTS  PURGE;
1       DROP TABLE einkaufswagen_produkt CASCADE CONSTRAINTS  PURGE;
2       DROP TABLE nutzer_einkaufswagen CASCADE CONSTRAINTS  PURGE;
3       DROP TABLE kategorie_business CASCADE CONSTRAINTS  PURGE;
4       DROP TABLE bestellung CASCADE CONSTRAINTS  PURGE;
5       DROP TABLE bestellung_produkt CASCADE CONSTRAINTS  PURGE;
```

Im ersten Block unseres Installationsskripts löschen wir alle Tabellen für den Fall, dass Sie noch da sind und geben den Speicher frei. Damit das auch mit Tabellen geht die zueinander in Beziehung stehen nutzen wir CASCADE CONSTRAINTS.

```
CREATE TABLE unternehmen (
    id INTEGER NOT NULL PRIMARY KEY,
    name varchar(24) NOT NULL,
    addresse varchar(24) NOT NULL,
    telefonnummer varchar(24) NOT NULL UNIQUE,
    ceo varchar(24) NOT NULL,
    iban varchar(22) NOT NULL UNIQUE
);
```

Beim Unternehmen haben wir bei der Telefonnummer und der IBAN die Integritätsbedingung UNIQUE implementiert, da diese beide einzigartig sind. Alle Attribute in dem Fall sind nicht NULLABLE, da Sie für die Identifikation für Transaktionen und ggf. für Rechtliches wichtig sind.

```
CREATE TABLE kategorie (
    id INTEGER NOT NULL PRIMARY KEY,
    name varchar(24) NOT NULL,
    bild BLOB,
    subkategorie_id INTEGER DEFAULT NULL,
    -- TODO check whether delete cascade on subcategory deletes main category
    FOREIGN KEY (subkategorie_id) REFERENCES kategorie(id) ON DELETE CASCADE
);
```

Die Kategorien haben eine rekursive Beziehung zu sich selbst, da sie Subkategorien enthalten können. Somit haben wir hier einen Fremdschlüssel der auf den PK einer anderen Kategorie zeigt. Eine Kategorie MUSS aber nicht zwingend eine Subkategorie haben daher ist dieses Attribut NULLABLE und die DEFAULT Value ist ebenfalls NULL. Der Name eine Kategorie ist nicht NULLABLE, weil wir Tiere die man keiner existierenden Kategorie zuordnen kann anders handhaben als durch eine unbekannte Kategorie.

```
CREATE TABLE einkaufswagen (
    id INTEGER NOT NULL PRIMARY KEY
--     Anzahl Deprecated weil belangloses Attribut, da man in hypothetischem Frontend sowieso ein Array von Objekten haben würde um die Liste von Produkten im Warenkorb anzuzeigen.
--     Somit könnte man für die Anzahl einfach Array.length oder ArrayList.size() oder Javascript Equivalent verwenden
--     anzahl INTEGER
);
```

Beim Einkaufswagen haben wir die ID als PRIMARY KEY festgelegt und dementsprechend mit NOT NULL versehen.

```
CREATE TABLE nutzer (
    id INTEGER NOT NULL PRIMARY KEY,
    vorname varchar(64) NOT NULL,
    nachname varchar(64) NOT NULL,
    anrede varchar(10) CHECK (anrede IN ('Herr', 'Frau')) ENABLE,
    mail varchar(80) NOT NULL UNIQUE,
    iban varchar(22) NOT NULL UNIQUE
);
```

Bei der Nutzertabelle haben wir die ID als PRIMARY KEY festgelegt und dementsprechend mit NOT NULL versehen. Der Vorname und der Nachname sind auch auf NOT NULL versehen, weil jeder Kunde einen Vor- und einen Nachnamen besitzen sollte. Gür die Lieferung ist das wichtig, da es auf die Bestellung Übernommen wird. Für die Anrede wurde CHECK genutzt um die Bedingung für "Herr" oder "Frau" zu überprüfen. Die Mail und die Iban ist

mit NOT NULL versehen, da man diese Werte benötigt um die Einzahlung zu bekommen und die Bestätigung abzuschicken. UNIQUE bewirkt, dass die Werte in der entsprechenden INDEX-Spalte eindeutig sind damit man die Daten und die Rechnungen auf die richtige Email bekommt und die für die Bestätigung für den Eingang des Geldes braucht man eine eindeutige IBAN. Sind sie nicht eindeutig fliegt eine Exception dass der UNIQUE Constraint verletzt wurde.

```
CREATE TABLE nutzer_einkaufswagen (
    nutzer_id INTEGER NOT NULL PRIMARY KEY,
    einkaufswagen_id INTEGER NOT NULL UNIQUE,

    FOREIGN KEY (nutzer_id) REFERENCES nutzer(id) ON DELETE CASCADE,
    FOREIGN KEY (einkaufswagen_id) REFERENCES einkaufswagen(id) ON DELETE CASCADE
);
```

Die nutzer_einkaufswagen Tabelle ist die Junction-Tabelle von Nutzer und Einkaufswagen, die die 1:1 Relation zwischen Nutzer und Einkaufswagen modelliert. Hier haben wir zwei FKs welche auf den Nutzer bzw. den Einkaufswagen referenzieren. Die nutzer_id ist hierbei der gewählte Primary Key, weil der Nutzer sozusagen, aussagenlogisch, der "Parent" der Relation ist, prinzipiell wäre es aber irrelevant welchen der beiden man nimmt, es darf aber nur einer sein. Der andere FK muss UNIQUE sein, so erzielen wir, dass in keiner Spalte ein Duplikat vorkommen darf. Damit ist eine 1:1 Relation erfolgreich hergestellt.

```
CREATE TABLE produkt (
    id INTEGER NOT NULL PRIMARY KEY,
    name varchar(64) NOT NULL,
    preis NUMBER(6, 2) NOT NULL,
    skin varchar(24),
    geschlecht varchar(10) CHECK (geschlecht IN ('Männlich', 'Weiblich')) ENABLE,
    age INTEGER,
    kategorie_id INTEGER DEFAULT NULL,
    unternehmen_id INTEGER NOT NULL,


    FOREIGN KEY (unternehmen_id) REFERENCES unternehmen(id) ON DELETE CASCADE,
    FOREIGN KEY (kategorie_id) REFERENCES kategorie(id) ON DELETE SET NULL
);
```

Bei der Produkttabelle ist ID der PRIMARY KEY. NAME und PREIS haben die Intregitätsbedinungen NOT NULL, weil ein Produkt nicht namenlos sein sollte und einen Preis haben muss, hier gibt es nichts umsonst. Des Weiteren haben wir bei Geschlecht eine CHECK Klausel eingebaut um sicherzugehen, dass nur 'Männlich' und 'Weiblich' benutzt wird. Mit ENABLE geben wir an, dass die Einschränkung auf die Daten in der Tabelle angewendet werden sollen. Kategorie_id hat die Integritätsbedinnung DEFAULT NULL, weil es auch NULL sein kann, da ein Produkt bei der Erstellung zunächst nicht zwingend einer Kategorie zugeordnet ist. Z.B. weil dafür noch keine Kategorie existiert und sich die Frage stellt, ob dafür eine neue Kategorie angelegt werden sollte oder nicht. Unternehmen_id ist mit NOT NULL versehen, weil man damit sieht von welchem Unternehmen das Produkt stammt.

```
CREATE TABLE einkaufswagen_produkt (
    einkaufswagen_id INTEGER NOT NULL,
    produkt_id INTEGER NOT NULL,

    PRIMARY KEY (einkaufswagen_id, produkt_id),
    FOREIGN KEY (einkaufswagen_id) REFERENCES einkaufswagen(id) ON DELETE CASCADE,
    FOREIGN KEY (produkt_id) REFERENCES produkt(id) ON DELETE CASCADE
);
```

Hier haben wir wieder ein Junction Table der die Relation zwischen Einkaufswagen und Produkt herstellt. Dort haben wir zwei FKs die jeweils auf Einkaufswagen und Produkt referenzieren und zusammen den PK bilden. Der Composite Key verhindert, dass Duplikate der Kombination in der Tabelle vorkommen können. Es kann also nicht 10x derselbe Hund im Einkaufswagen liegen, warum wurde bereits in einer vorherigen Aufgabe erläutert. Die FKs haben auch wieder ON DELETE CASCADE damit die Relation gelöscht wird sollte ein Produkt oder ein Einkaufswagen gelöscht werden.

```
CREATE TABLE kategorie_business (
    kategorie_id INTEGER NOT NULL,
    business_id INTEGER NOT NULL,
    PRIMARY KEY (kategorie_id, business_id),
    FOREIGN KEY (kategorie_id) REFERENCES kategorie(id) ON DELETE CASCADE,
    FOREIGN KEY (business_id) REFERENCES unternehmen(id) ON DELETE CASCADE
);
```

Kategorie_Business stellt die Relation zwischen Kategorien und Unternehmen her. Jedes Unternehmen kann mehrere Kategorien bewirtschaften und jede Kategorie kann von mehreren Unternehmen bewirtschaftet werden. Wir haben zwei FKs die zusammen einen PK bilden, sodass Duplikat-Einträge verhindert werden. Ebenfalls haben diese den Constraint ON DELETE CASCADE, damit wenn eine Kategorie oder ein Business gelöscht wird auch die Relation entfernt wird.

```sql
-- TODO bestellung löschen check on delete cascade --
CREATE TABLE bestellung (
    id INTEGER NOT NULL PRIMARY KEY,
    unternehmen_id INTEGER NOT NULL,
    nutzer_id INTEGER NOT NULL,
    bestellstatus varchar(10) NOT NULL,
    bestelldatum DATE NOT NULL,
    lieferaddresse varchar(24) NOT NULL,
    FOREIGN KEY (nutzer_id) REFERENCES nutzer(id) ON DELETE CASCADE,
    FOREIGN KEY (unternehmen_id) REFERENCES unternehmen(id) ON DELETE CASCADE   ShadowD40, 51 minutes ag
);
```

Bei einer Bestellung ist es so gelöst dass diese immer genau einem Unternehmen und einem Nutzer zugeodnet ist. Deshalb zwei FKs die auf Nutzer und Unternehmen referenzieren. D.h. wenn ein Nutzer 2x Hund (1x Hund X 1x Hund Y) und 3x Löwe (Löwe X, Y und Z) kauft und Hund kommt von Unternehmen X und Löwe von Unternehmen Y gehen insgesamt 2 Bestellungen raus. Die FKs sind nicht NULLABLE weil eine Bestellung IMMER einem Unternehmen und einem Nutzer zugeordnet sein muss, weil sie jemand bezahlen und auch bearbeiten muss. Bestellstatus ist nicht NULLABLE, weil eine Bestellung logischerweise immer einen Zustand hat wie alles andere im Universum ebenfalls. Bestelldatum ist nicht NULLABLE, weil eine Bestellung immer zu einem Zeitpunkt aufgegeben wird. Die Lieferaddresse ist nicht NULLABLE, weil die Bestellung sonst nicht ankommt. Die FKs haben ON DELETE CASCADE, weil eine Bestellung gelöscht werden sollte, wenn der Nutzer oder das Unternehmen gelöscht wird. Sonst weiß man ja nicht wer die Bestellung aufgegben hat an wen und wohin sie geliefert werden soll.

```sql
CREATE TABLE bestellung_produkt (
    bestellung_id INTEGER NOT NULL,
    produkt_id INTEGER NOT NULL,

    PRIMARY KEY (bestellung_id, produkt_id),
    FOREIGN KEY (bestellung_id) REFERENCES bestellung(id) ON DELETE CASCADE,
    FOREIGN KEY (produkt_id) REFERENCES produkt(id) ON DELETE CASCADE
);
```

Die Tabelle bestellung_produkt stellt die Relation zwischen Bestellung und Produkt her. Hier haben wir zwei FKs die wieder einen gemeinsamen PK bilden, sodass keine Duplikate erlaubt sind. Die FKs referenzieren auf Bestellung und Produkt. Die FKs haben den Constraint ON DELETE CASCADE, weil die Relation gelöscht werden soll, wenn ein Produkt oder auch eine Bestellung gelöscht wird.