# McDonald's Lieferdienst-Konzept

Projekt

*Einführung in Datenbanken*

Mihail Melhev & Leonard Berresheim

## Inhaltsverzeichnis



## Einleitung

Im Rahmen des Projektes im Kurs *"Einführung in Datenbank"* wurde ein Konzept für eine Datenbank zum betreiben eines theoretischen Lieferdienst der Firma McDonalds erstellt und in SQL umgesetzt.

## 1. Informelle Beschreibung

Wie folgt ist die informelle Problembeschreibung:

- Die Firma hat einen Speiseplan, der aus mehreren *items* besteht mit folgenden Informationen: *name*, *price*, *size*, *category*.
- die *category* des *items* bestimmt wann das Produkt gekauft werden kann (Breakfast: 5:00am - 10:00am/Lunch: 10:00am - 5:00am) .
- *items* können einzeln oder in einem *menu* bestellt werden.
- Ein *menu* besteht aus jeweils ein Getränk (*beverage*), eine Hauptspeise (*main*), einer Beilage (*side*) und ggf. eines Dessert (das Menu muss kein *dessert* beinhalten, alle anderen schon).
- Beim kauf eines *menu*, gibt es 10% Rabat auf den ursprünglischen Preis, der im *menu* bestellten Produkte.
- Die Firma hat eine Kundendatenbank, die folgende Information über Kunden enthält: Benutzternname (*userName*), Nachname (*lastName*), Vorname (*firstName*), Addresse (Straße (*street*), Hausnummer (*streetNr*) , PLZ, Stadt (*city*).
- Die Firma hat eine Bestelldatenbank, die folgende Informationen über abgeschlossene Bestellungen enthält: Benutzternname, Bestellzeitpunkt (*orderDate), Gesamtpreis (total).*

# 2. Konzeptioneller Datenbankentwurf

Die informelle Beschreibung wurde durch einen Konzeptionellen Datenbankentwurf in Abbildung 1 formalisiert.

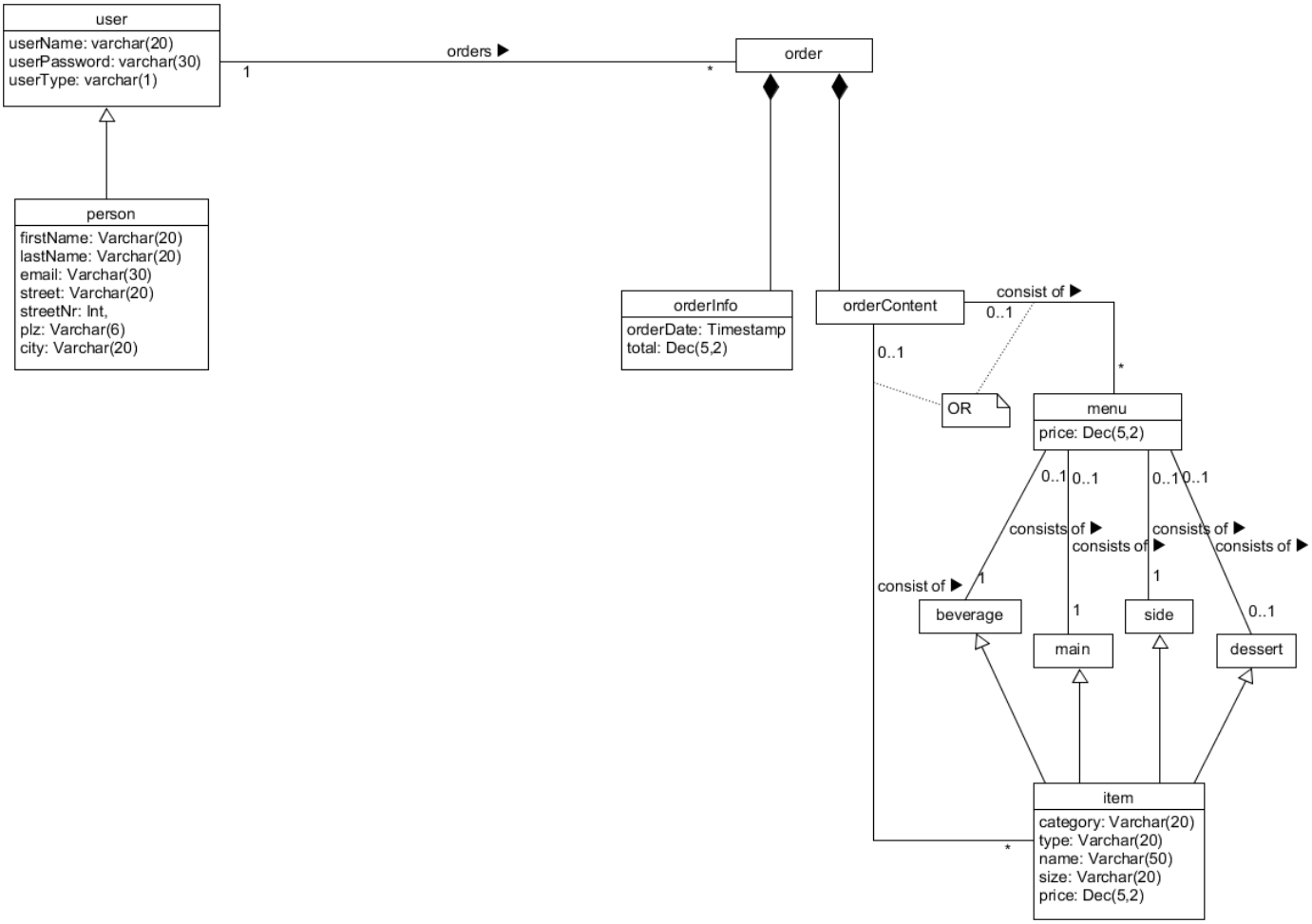


*Illustration 1: Konzeptioneller Datenbankentwurf*

Als relevante Entitäten haben wir u*ser*, o*rder*, *item* mit ihren aus dem Diagramm ersichtlichen Attributen (genaue Dokumentation der Bedeutung der Attribute erfolgt weiter unten) identifiziert. Die Assoziationen, Vererbungen, Compositionen formalisieren:

- Die Entität u*ser* erbt von der Entität Person.
- Jede Bestellung (*order*) wir von genau einem u*ser* getätigt.
- Die Entität Bestellung setzt sich zusammen aus:
  - orderInfo.
  - orderContent.
- Eine Bestellung (*order*) besteht aus beliebig vielen *items* und *menus*.
- Ein *menu* besteht aus genau einem *beverage*, *main*, *side* und einem oder keinem d*essert*.

# 3. Relationaler Datenbankentwurf

Der Konzeptionelle Datenbankentwurf wurde nun in einem Relationalen Datenbankentwurf umgesetzt. Der Umsetzung wegen wurde dieser jedoch nicht eins zu eins übertragen. Die Entität *menu* beispielsweise fällt weg, deren Funktionalität wird intern geregelt. Weitere Erklärungen bei der detaillierten Dokumentation der Tabellen.
In Abbildung 2 sehen wir den relationellen Datenbankentwurf.

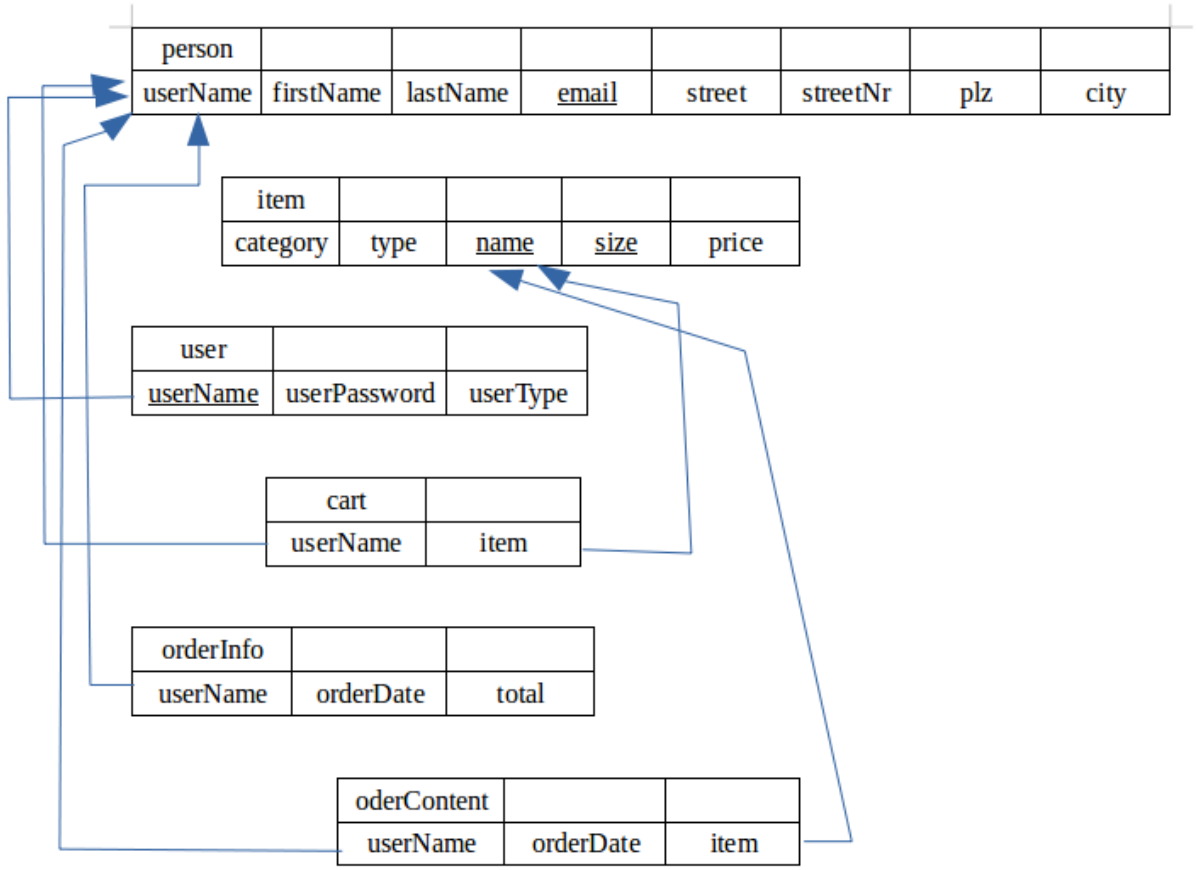


*Abbildung 2: Relationaler Datenbankentwurf*

## Detaillierte Dokumentation der Tabellen

| Tabelle / Spalte | | Beschreibung |
|---|---|---|
| item | | beinhaltet alle Items die zum Kauf zur verfügung stehen |
| | category | die Item-Category (Breakfast, Lunch), bestimmt zu welcher Zeit das Item gekauft werden kann |
| | type | der Item-Type, ist für Menus relevant. (Main + Side + Beverage (+ Dessert) ergibt ein Menu |
| | name | der Item-Name. Ergibt zusammen mit size den PRIMARY KEY |
| | size | die größe des Items. Ergibt zusammen mit name den PRIMARY KEY |
| | price | der Preis des Items. In Decimalform (dec(5,2)) |
| | | |

| | | |
|---|---|---|
| user | | beinhalt Login-Informationen aller User die im System Registriert sind |
| | userName | der Benutzername. Ergibt den PRIMARY KEY |
| | userPassword | das Passwort. |
| | userType | der User-Type. Admin oder User, bestimmt auf welchen inhalt der der angemeldete User zugriff hat. NOT NULL |
| | | |
| person | | enthält weiter Informationen zu den registrierten Usern |
| | userName | FOREIGN KEY auf user_userName |
| | firstName | der Vorname des Users |
| | lastName | der Nachname des Users |
| | email | die Email. PRIMARY KEY |
| | street | die Straße |
| | streetNr | die Straßennummer |
| | plz | die Postleitzahl |
| | city | die Stadt |
| | | |
| cart | | aktueller Inhalt des Warenkorbs. Wird bei Beendigung des Einkaufs gelöscht bzw. starten einer neuen Bestellung |
| | userName | FOREIGN KEY auf user_userName |
| | item | die sich im Warenkorb befindenden Items. FOREIGN KEY auf items_name |
| | | |
| oderInfo | | beinhaltet alle abgeschlossenen Bestellungen |
| | userName | FOREIGN KEY auf user_userName |
| | orderDate | der Besellzeitpunkt. Wird über einen Timestamp festgelegt |
| | total | der Endpreis. Summe aller Einzelpreise bzw. redurzierter Preis bei Menus. SELECT auf SUM(cart_price) |
| | | |
| orderContents | | beinhaltet den Inhalt der Bestellung |
| | userName | der bestellende User. FOREIGN KEY auf orderInfo_userName |
| | orderDate | der Zeitpunkt der Bestellung. Wird durch ein Timestamp festgelegt |
| | item | die bestellten Items. FOREIGN KEY auf items_name |

## Bemerkungen:

- Wenn ein *user* seine Profilinformationen ändert, sollen diese Informationen bei einer bereits getätigten Bestellung jedoch beibehalten werden, also nicht geupdatet werden. Aus diesem Grund ist ein Foreign Key auf die *user* Datenbank nicht möglich, es ensteht also Redundanz (Foreign Key auf *userName* bleibt bestehen, dieser kann vom *user* nicht geändert werden).

Die Tabellebeschreibung für *orderInfo* sieht also wie folgt aus:

| oderInfo | | beinhaltet alle abgeschlossenen Bestellungen |
|---|---|---|
| | userName | FOREIGN KEY auf user_userName |
| | orderDate | der Besellzeitpunkt. Wird über einen Timestamp festgelegt |
| | total | der Endpreis. Summe aller Einzelpreise bzw. redurzierter Preis bei Menus |
| | firstName | der Vorname |
| | lastName | der Nachname |
| | email | FOREIGN KEY auf user_email |
| | street | die Straße |
| | streetNr | die Straßennummer |
| | plz | die Postleitzahl |
| | city | die Stadt |

- Die Tabelle *Cart* wird bei jeder neuen Bestellung wieder gelöscht, dadurch entsteht auch hier ein Problem mit dem Foreign Key und Redundanz ist unvermeidbar.
  Die Tabellenbeschreibung für *cart* sieht also wie folgt aus:

| cart | | aktueller Inhalt des Warenkorbs. Wird bei Beendigung des Einkaufs gelöscht bzw. starten einer neuen Bestellung |
|---|---|---|
| | userName | FOREIGN KEY auf user_userName |
| | type | der Item-Typ |
| | name | der Item-Name |
| | size | die Item-Größe |
| | price | der Item-Preis |

- In der Tabelle person wurde *streetNr* der Typ int zugewiesen, da *streetNr* jedoch nicht als Zahl benutzt wird (es wird nicht Inkrementiert, Addiert usw.), wäre es sinnvoller streetNr ebenfalls als varchar zu speichern.

# 4. Implementierung der Datenbank in SQL

- Die Tabelle *item* wurde wie folgt implementiert:

```
create table items
(
     category varchar(20),
     type varchar(20),
     name varchar(50),
     size varchar(20),
     price dec(5,2),
     primary key(name, size)
);
```

  Da *items* mit gleichen Namen in verschiedenen Größen vorhanden sind, wurde hier als PRIMARY KEY die Kombination (*name*,*size*) gewählt, die in jedem Fall Eindeutig ist.
  Bei *name* sind inerhalb des Programms (QT) Probleme aufgetaucht, bei Namen die sehr lang sind, dehalbt wurde von der anfänglichen varchar(30) auf varchar(50) erhöht. Dies hat das Problem jedoch nicht behoben.

- Die Tabelle *user* wurde wie folgt implementiert:

```
create table user
 (
     userName varchar(20) not null unique primary key,
     userPassword varchar(30) not null,
     userType varchar(1) not null
);
```

  *userName* ist PRIMARY KEY. *userPassword* und *userType* dürfen nicht NULL sein.
  *userType* wird bei einer neuen Registrierung automatisch auf "u" für user gesetzt. Erstellen von admins ("a") nur im Code möglich.

- Die Tabelle *person* wurde wie folgt implementiert:

```
create table person
 (
     userName varchar(20),
     firstName varchar(20),
     lastName varchar(20),
     email varchar(30) unique primary key,
     street varchar(20),
     streetNr int,
     plz varchar(6),
     city varchar(20),
     FOREIGN KEY (userName) REFERENCES user(userName) ON
     UPDATE CASCADE ON DELETE CASCADE
);
```

  *userName* ist eine REFERENCE auf *user(userName)* und wird ON UPDATE CASCADE und ON DELETE CASCADE.

- die Tabelle *cart* wurde wir folg implementiert:

```
create table cart(
     userName varchar(20),
```

```
        type varchar(20),
        name varchar(50),
        size varchar(20),
        price dec(5,2),
        FOREIGN KEY (userName) REFERENCES user(userName) ON
        UPDATE CASCADE ON DELETE CASCADE
    );
```

*userName* ist wieder eine REFERENCE auf *user(userName)* und wird ON UPDATE CASCADE und ON DELETE CASCADE. Hier ensteht Redundanz aus den oben bereits genannten Gründen: Da die Tabelle bei jeder neuen Bestellung wieder gelöscht wird führen FOREIGN KEYS zu internen Problemen, die durch Redundanz behoben worden sind (quick-and-dirty-Methode).

- Die Tabelle *oderInfo* wurde wie folgt implementiert:

```
create table orderInfo (
        userName varchar(20),
        orderDate datetime,
        total dec(5,2),
        firstName varchar(20),
        lastName varchar(20),
        email varchar(30),
        street varchar(20),
        streetNr int,
        plz varchar(6),
        city varchar(20),
        FOREIGN KEY (userName) REFERENCES user(userName) ON
        UPDATE CASCADE ON DELETE CASCADE
    );
```

Wie bereits oben genannt, kann der Kunde seine Profilinformationen ändern, dies soll aber keinen Einfluss auf die Informationen zu bereits getätigten Bestellung haben. Also werden *user*-Informationen nocheinmal in der Tabelle oderInfo gespeichert. Dies hat sich als am einfachsten und effektivsten erwiesen.

- Die Tabelle *orderContents* wurde wie folgt implementiert:

```
create table orderContents(
        userName varchar(20),
        orderDate datetime,
        type varchar(20),
        name varchar(50),
        size varchar(20),
        price dec(5,2),
        FOREIGN KEY (userName) REFERENCES orderInfo(userName) ON
UPDATE          CASCADE ON DELETE CASCADE
    );
```

Redundanz hätte in diesem Fall vermieden werden können (TODO).

## Nicht Implementiert:

Aus Zeitgründen wurde die Menu-Funktionalität nicht fertig implementiert. Kunden müssen also vorerst auf Rabate verzichten.

# 5. Umsetzung der GUI

Die GUI wurde mit Hilfe von QT implementiert. Abbildung 3 zeigt den Zustandsautomaten der GUI.

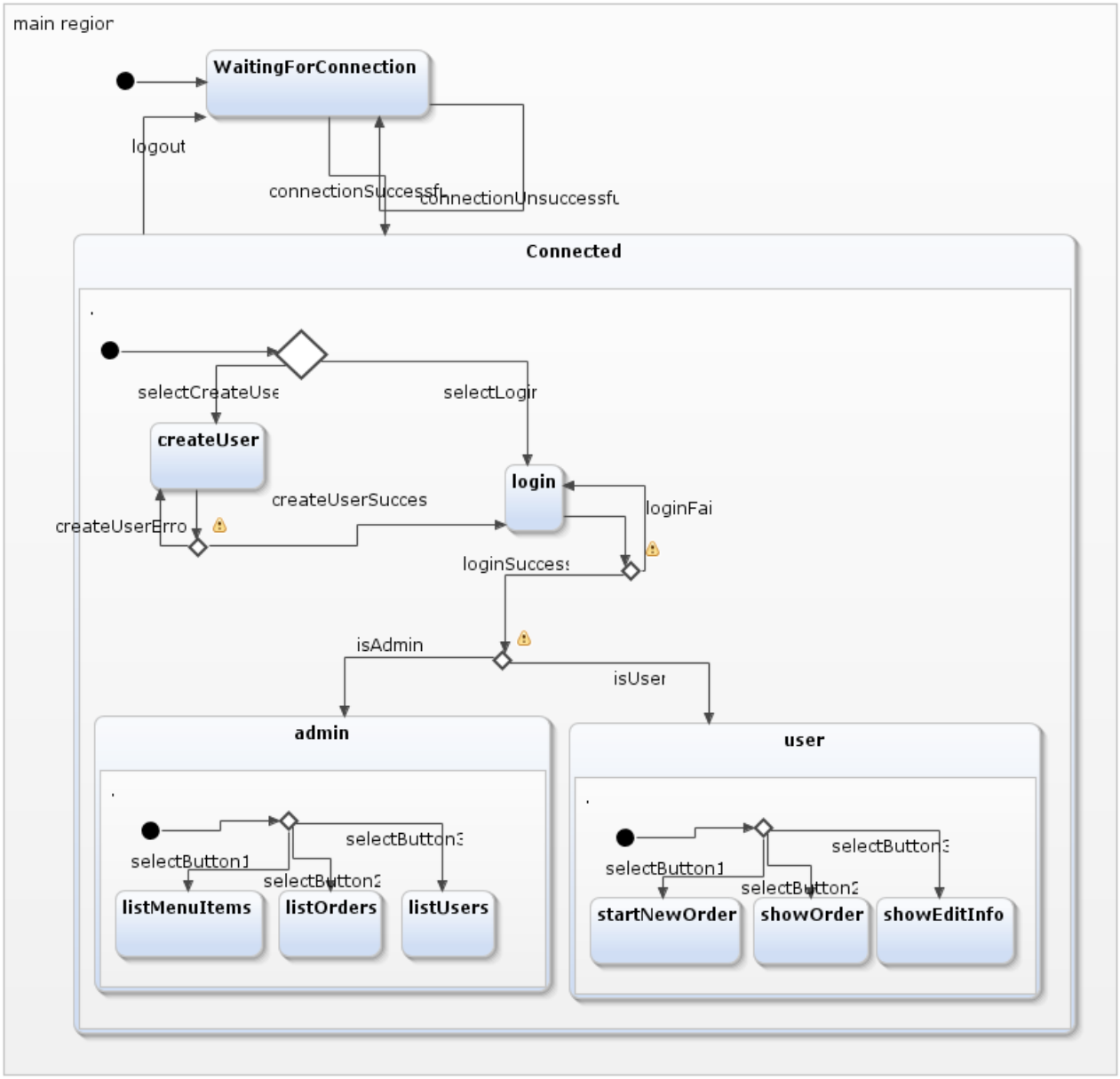


*Abbildung 3: Zustandsautomat*