Dokumentation Übung 1.1

TECHNISCHE UNIVERSITÄT DARMSTADT

11. Mai 2016

Informatik im Bauwesen II – Gruppe 1
Florian Saumweber, 2354534
Benjamin Krauß, 2388173

Anforderungen an die Software

Die Verwaltung von Kunden und Aufträgen in einfacher Form ist das Ziel der Software. Diese Applikation soll es dem Benutzer ermöglichen, über die Kommandozeile (Konsolenanwendung) Kunden- und Auftrags-Daten einzugeben und sich diese ebenfalls in der Kommandozeile anzeigen zu lassen.

Folgende Daten werden hierbei verwaltet:

- Adresse (ID, Straße, Hausnummer, PLZ, Ort, Land)
- Kunde (ID, Name, Vorname, Adresse)
- Auftrag (ID, Beschreibung, erledigt (ja/nein), auftraggebender Kunde)

Folgende Funktionen werden zur Verfügung gestellt:

- Eingabe eines neuen Kunden
- Eingabe eines neuen Auftrags
- Alle Kunden ausgeben
- Alle Aufträge ausgeben
- Einen Kunden mit allen zugehörigen Aufträgen ausgeben

Die Umsetzung soll mithilfe der Programmiersprache Java und der Datenbanksprache SQL im Datenbankmanagementsystem MySQL realisiert werden. Der Zugriff auf die Datenbank erfolgt über die JDBC Schnittstelle.

Entity-Relationship-Modell

Die Datenmodellierung innerhalb des Entity-Relationship-Modells (ERM) dient dazu die Anforderungen aufzugreifen und einheitlich zu visualisieren. Sie bildet die Grundlage des Datenbankentwurfes.
Die Entitäten in diesem Modell sind die *Adresse*, der *Kunde* und der *Auftrag*. Diese werden in *Abbildung 1* durch Attribute weiter verfeinert. Die Relationen *gehört zu* und *veranlasst* setzen die Entitäten zueinander in Beziehung. Die Kardinalitäten erweitern die Semantik der Relationen. In diesem Fall wird davon ausgegangen, dass ein Kunde nur eine Adresse hat. Allerdings ist es möglich, dass mehrere Kunden ihren Sitz an der gleichen Adresse, wie beispielsweise einem Büro-Komplex, haben.

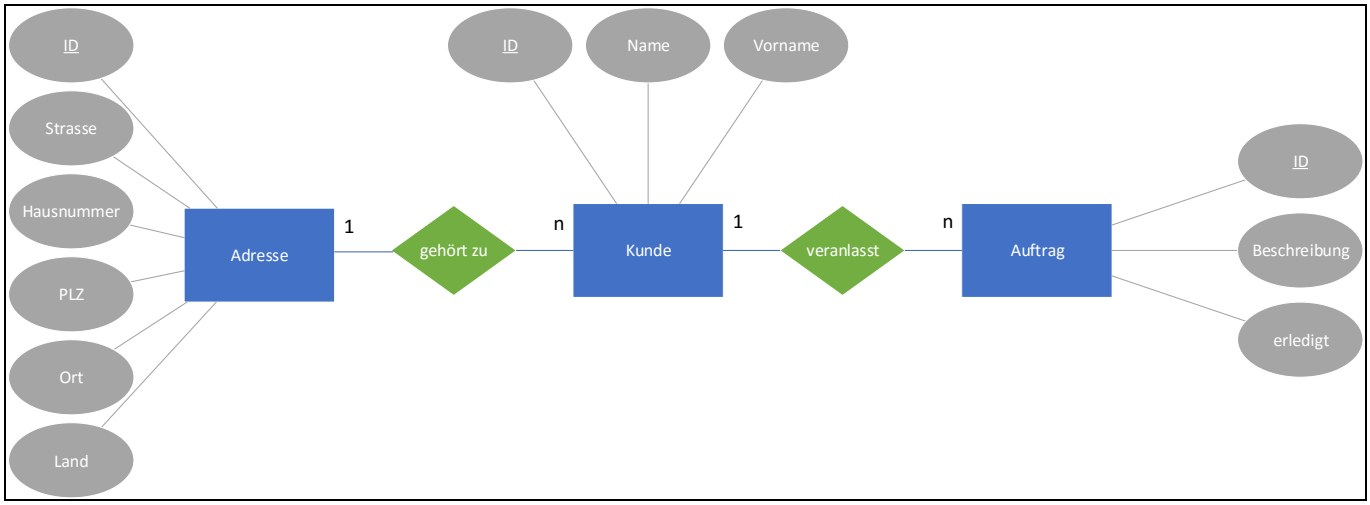

Abbildung 1: ERM der Übung 1.1

Relationaler Datenbankentwurf

Der relationale Datenbankentwurf leitet sich aus dem ERM in *Abbildung 1* ab. Eine Übersicht zeigt *Abbildung 2*, die das MySQL Model aus der Workbench wiederspiegelt.

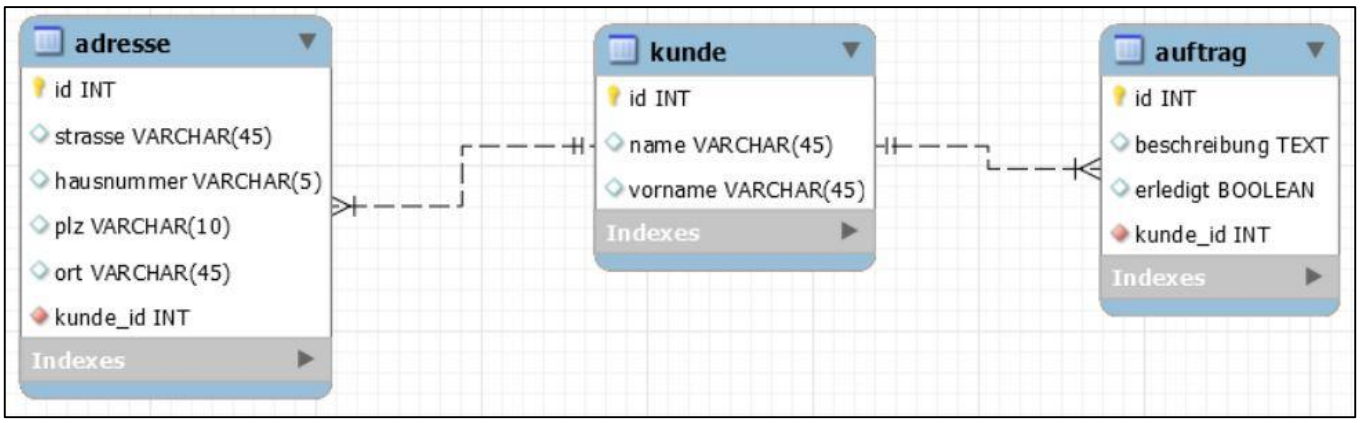

Abbildung 2: MySQL Model der Workbench

Die entsprechenden Entitäten mit den zugehörigen Attributen wurden in Beziehung gesetzt und spezifiziert.
Aus dem Model ergeben sich die konkreten Befehle und Anweisungslisten zur Erstellung der Datenbank. Im Folgenden ist der SQL-Code zur Generierung der Datenbank zu sehen. Die Daten des Beispieldatensatzes finden sich in Zeile 76, 86 und 96.

```
1 -- MySQL Workbench Forward Engineering
2
3 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
4 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
5 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
6
7 -- -----------------------------------------------------
8 -- Schema iib2_gruppe1
9 -- -----------------------------------------------------
10 DROP SCHEMA IF EXISTS `iib2_gruppe1` ;
11
12 -- -----------------------------------------------------
13 -- Schema iib2_gruppe1
14 -- -----------------------------------------------------
15 CREATE SCHEMA IF NOT EXISTS `iib2_gruppe1` DEFAULT CHARACTER SET utf8 ;
16 USE `iib2_gruppe1` ;
17
18 -- -----------------------------------------------------
19 -- Table `iib2_gruppe1`.`kunde`
20 -- -----------------------------------------------------
21 CREATE TABLE IF NOT EXISTS `iib2_gruppe1`.`kunde` (
22   `id` INT NOT NULL AUTO_INCREMENT,
23   `name` VARCHAR(45) NULL,
```

```
  `vorname` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `iib2_gruppe1`.`auftrag`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `iib2_gruppe1`.`auftrag` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `beschreibung` TEXT NULL,
  `erledigt` TINYINT(1) NULL,
  `kunde_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Auftrag_Kunde1_idx` (`kunde_id` ASC),
  CONSTRAINT `fk_Auftrag_Kunde1`
    FOREIGN KEY (`kunde_id`)
    REFERENCES `iib2_gruppe1`.`kunde` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `iib2_gruppe1`.`adresse`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `iib2_gruppe1`.`adresse` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `strasse` VARCHAR(45) NULL,
  `hausnummer` VARCHAR(5) NULL,
  `plz` VARCHAR(10) NULL,
  `ort` VARCHAR(45) NULL,
  `kunde_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Adresse_Kunde1_idx` (`kunde_id` ASC),
  CONSTRAINT `fk_Adresse_Kunde1`
    FOREIGN KEY (`kunde_id`)
    REFERENCES `iib2_gruppe1`.`kunde` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

-- -----------------------------------------------------
-- Data for table `iib2_gruppe1`.`kunde`
-- -----------------------------------------------------
START TRANSACTION;
USE `iib2_gruppe1`;
INSERT INTO `iib2_gruppe1`.`kunde` (`id`, `name`, `vorname`) VALUES (DEFAULT, 'Duck', 'Donald');

COMMIT;


-- -----------------------------------------------------
-- Data for table `iib2_gruppe1`.`auftrag`
-- -----------------------------------------------------
START TRANSACTION;
USE `iib2_gruppe1`;
```

```
 86 INSERT INTO `iib2_gruppe1`.`auftrag` (`id`, `beschreibung`, `erledigt`, `kunde_id`) VALUES (DEFAULT, '2x
Blumenstrauß', false, 1);
 87
 88 COMMIT;
 89
 90
 91 -- -----------------------------------------------------
 92 -- Data for table `iib2_gruppe1`.`adresse`
 93 -- -----------------------------------------------------
 94 START TRANSACTION;
 95 USE `iib2_gruppe1`;
 96 INSERT INTO `iib2_gruppe1`.`adresse` (`id`, `strasse`, `hausnummer`, `plz`, `ort`, `kunde_id`) VALUES (DEFAULT,
'Blumenstraße', '13', '10101', 'Entenhausen', 1);
 97
 98 COMMIT;
 99
100
```

Konzept und Umsetzung

Konzeptionell wird der Code in zwei Teilaufgaben strukturiert:

- die auf die Datenbank bezogenen Logik
- die Visualisierung und Übersetzung der Anweisungen für den Nutzer

Daraus resultierend ergeben sich zwei Arten von Klassen mit unterschiedlichen Aufgaben.

Die erste Klasse regelt die Verbindung zur Datenbank und übergibt beispielsweise die Parameter für eine derartige Benutzeranmeldung. Des Weiteren werden dort Methoden hinterlegt, die SQL Befehle ausführen können. Somit verwaltet die Klasse auch die Einfüge- und Auslese-Operationen.
In der zweiten Klasse werden die Eingaben des Benutzers verarbeitet, die GUI-Eingaben an die entsprechenden Methoden für die Datenbank weitergegeben und die Ergebnisse von Anfragen visuell ausgegeben.

Konkret werden in der Klasse *DbConnection* die Parameter zur Verbindung, wie beispielsweise die Portnummer, gespeichert. Sie regelt außerdem die allgemeinen Verbindungsmethoden, wie connect() oder close(). Überdies werden die SQL Befehle INSERT und SELECT in entsprechender Anpassung an die vorkommenden Entitäten Auftrag, Kunde und Adresse in Methoden, wie iKunde() oder sKunde() repräsentiert.
Die *Iib2_ue1_1_gruppe1*-Klasse spiegelt die oben erwähnte zweite Klassenart wieder. Die Startanzeige oder auch die Verarbeitung der Benutzereingabe mittels einer Fallunterscheidung werden hier aufgegriffen. Die Werte werden z. B. an die in DbConnection ausgelagerten Methoden zur Datenbankabfrage weitergegeben oder dem Nutzer anhand einer Listenform präsentiert.

Erläuterungen zur graphischen Benutzeroberfläche

Mit Rechtsklick auf die Datei *Iib2_ue1_1_gruppe1.java* innerhalb des NetBeans-Projektes kann der Befehl *Run File* ausgeführt werden. Die Applikation startet.
Alternativ kann die Anwendung auch über die Konsole gestartet werden. Hierzu wird über den Befehl *cd C:\PfadAngeben* das Verzeichnis mit der Datei ausgewählt. Die anschließende Anweisung *java -jar FileName.jar* startet die Applikation.

Nach dem Start wird eine Zusammenfassung der möglichen Befehle dargestellt, wie sie in *Abbildung 3* zu sehen ist.

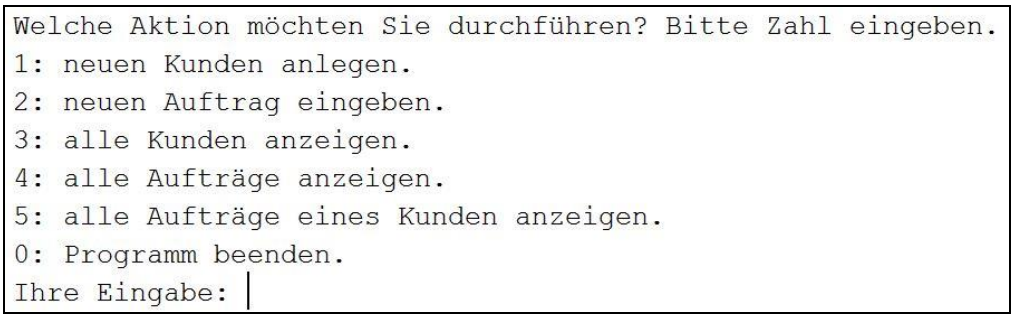

Abbildung 3: GUI beim Start

Nach der Eingabe und Bestätigung mittels Enter, erhält der Nutzer weitere Informationen im Output. Exemplarisch sei an dieser Stelle die Aktion *3: alle Kunden anzeigen* aufgeführt. Der bisher angezeigte Text wird um weitere Zeilen erweitert. Die gewünschten Informationen werden in Listenform präsentiert (siehe *Abbildung 4*). Der Beispieldatensatz enthält in diesem Fall den Kunden *Duck Donald*.

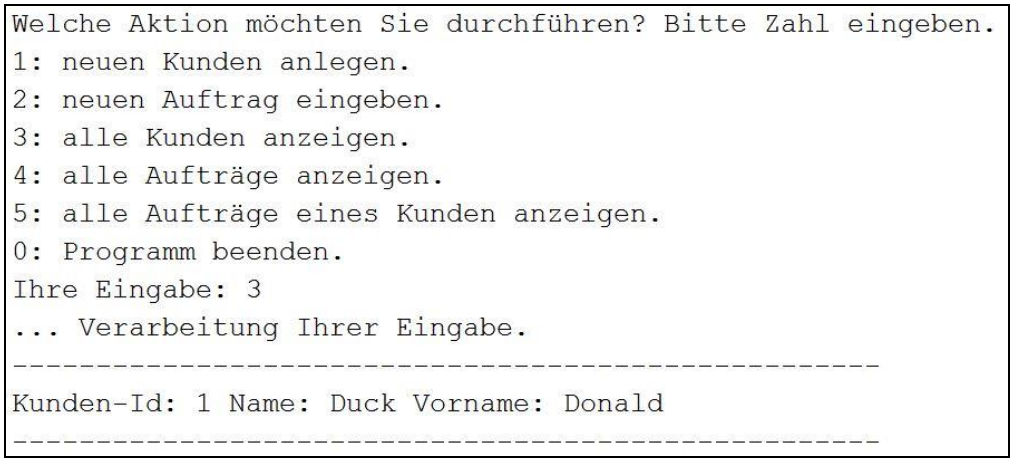

Abbildung 4: GUI mit Beispieleingabe

Bei der Eingabe von Befehlen oder Daten kann es zu Fehlern kommen. Der Nutzer erhält eine Rückmeldung über erkannte Fehler, falls z. B. als Name eines Kunden *0 0* eingegeben wird (siehe *Abbildung 5*).

```
Ihre Eingabe: 1
... Verarbeitung Ihrer Eingabe.
-----------------------------------------------------
Nachname des Kunden: 0
Vorname des Kunden: 0
Name ist leer bzw. zu kurz.
Vorname ist leer bzw. zu kurz.
Es trat ein Fehler, versuchen Sie es noch einmal.
-----------------------------------------------------
```

Abbildung 5: GUI mit Fehlermeldung

Im Allgemeinen sind IDs ein wesentlicher Bestandteil von Datenbanken. Der Nutzer kommt mit ihnen jedoch nicht in Berührung. Da der Fokus dieser Übung auf dem Kennenlernen der Technologien aus dem Anforderungskatalog liegt, wird beim Befehl *alle Aufträge eines Kunden anzeigen* der Einfachheit halber auf IDs zurückgegriffen (siehe *Abbildung 6*).

```
Ihre Eingabe: 5
... Verarbeitung Ihrer Eingabe.
-----------------------------------------------------
Kunden-Id des Auftraggebers. Wollen Sie zuvor die Kundenliste sehen (j/ n)? j
Kunden-Id: 1 Name: Duck Vorname: Donald
Kunden-Id des Auftraggebers: 1
Auftrag-Id: 1 Beschreibung: 2x Blumenstrauß erledigt: 0 Auftraggeber: 1
-----------------------------------------------------
```

Abbildung 6: Verwendung der ID