Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 2 SoSe 2024
Fakultät für Angewandte Informatik
Institut für Informatik
Prof R. Lorenz, L. Petrak, J. Linne, V. Le Claire, P. Schalk 05.07.2024

Lösungsvorschlag zu Übungsblatt 10

Abgabe: 15.07.2024, 10:00 Uhr (im Digicampus via VIPS: .java-Dateien für Code, .uxf für UML, .pdf für alles andere)

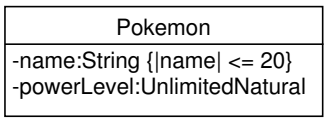
- Dieses Übungsblatt muss im Team abgegeben werden (Einzelabgaben sind nicht erlaubt!).
- Die **Zeitangaben** geben zur Orientierung an, wie viel Zeit für eine Aufgabe später in der Klausur vorgesehen wäre; gehen Sie davon aus, dass Sie zum jetzigen Zeitpunkt wesentlich länger brauchen und die angegebene Zeit erst nach ausreichender Übung erreichen.

* leichte Aufgabe / ** mittelschwere Aufgabe / *** schwere Aufgabe

Aufgabe 37 * *(Relationale Datenbanken, 32 Minuten)*

In den folgenden Teilaufgaben sollen Sie jeweils zu einem Klassendiagramm Tabellen einer relationalen Datenbank konstruieren, indem Sie jeweils eine SQL-`CREATE`-Anweisung angeben.
Die Datenbank soll genau die Informationen des Klassendiagramms enthalten. Sie sollen also keine Informationen weglassen oder hinzufügen. Vergessen Sie dabei nicht die Angabe von Primär- und Fremdschlüsseln und Einschränkungen.

a) *(*, 4 Minuten)*
Konstruieren Sie die Tabellen einer relationale Datenbank zu folgendem Klassendiagramm:



Lösung:

```
CREATE TABLE Pokemon (
        id INTEGER NOT NULL AUTO_INCREMENT,
        name VARCHAR(20) NOT NULL,
        powerLevel INTEGER NOT NULL,
        CHECK(powerLevel >= 0),
        PRIMARY KEY(id)
);
```

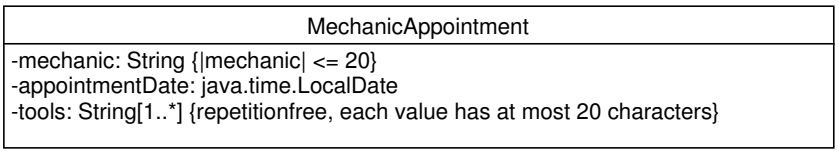
b) *(*, 6 Minuten)*
Konstruieren Sie die Tabellen einer relationale Datenbank zu folgendem Klassendiagramm:

Lösung:

```
CREATE TABLE Printer (
        MACAdress CHAR(12) NOT NULL,
        pricePerPage FLOAT NOT NULL,
        PRIMARY KEY (MACAdress)
);

CREATE TABLE PrintJob (
        id INTEGER NOT NULL AUTO_INCREMENT,
        documentName VARCHAR(50) NOT NULL,
        numberOfCopies INTEGER NOT NULL,
        printerAdress CHAR(12) NOT NULL,
        CHECK(numberOfCopies >= 0),
        PRIMARY KEY(id),
        FOREIGN KEY (printerAdress) REFERENCES Printer(MACAdress)
);
```

c) *(*, 9 Minuten)*
Konstruieren Sie die Tabellen einer relationale Datenbank zu folgendem Klassendiagramm:

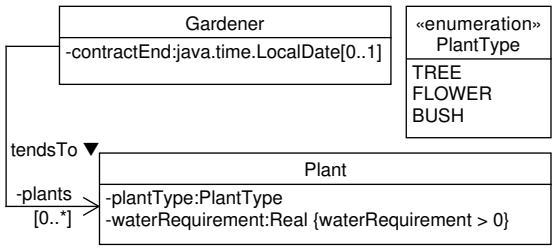


Lösung:

```
CREATE TABLE MechanicAppointment (
        id INTEGER NOT NULL AUTO_INCREMENT,
        mechanic VARCHAR(20) NOT NULL,
        appointmentDate DATE NOT NULL,
        PRIMARY KEY(id)
);

CREATE TABLE MechanicAppointmentTools (
        mechanicAppointmentId INTEGER NOT NULL,
        tool VARCHAR(20) NOT NULL,
        PRIMARY KEY (mechanicAppointmentId, tool),
        FOREIGN KEY (mechanicAppointmentId) REFERENCES mechanicAppointment(id)
);
```

d) *(*, 13 Minuten)*
Konstruieren Sie die Tabellen einer relationale Datenbank zu folgendem Klassendiagramm:



Lösung:

```
CREATE TABLE Gardener (
        id INTEGER NOT NULL AUTO_INCREMENT,
        contractEnd DATE,
        PRIMARY KEY(id)
);

CREATE TABLE Plant (
        id INTEGER NOT NULL AUTO_INCREMENT,
        plantType VARCHAR(13) NOT NULL,
        waterRequirement FLOAT NOT NULL,
        CHECK(plantType IN ('TREE', 'FLOWER', 'BUSH')),
        CHECK(waterRequirement > 0),
        PRIMARY KEY(id)
);

CREATE TABLE GardenerPlant (
        gardenerId INTEGER NOT NULL,
        plantId INTEGER NOT NULL,
        PRIMARY KEY(gardenerId, plantId),
        FOREIGN KEY (gardenerId) REFERENCES Gardener(id),
        FOREIGN KEY (plantId) REFERENCES Plant(id)
);
```

Diskussion: Alternativ kann PlantType auch als eigene Tabelle modelliert werden:

```
CREATE TABLE PlantType (
        plantType VARCHAR(13) NOT NULL,
        PRIMARY KEY(plantType)
);

CREATE TABLE Plant (
        id INTEGER NOT NULL,
        plantType VARCHAR(13) NOT NULL,
        age Integer NOT NULL,
        waterRequirement FLOAT NOT NULL,
        CHECK(age >= 0),
        CHECK(waterRequirement > 0),
        PRIMRAY KEY(id),
        FOREIGN KEY (plantType) REFERENCES PlantType(plantType)
);
```

Aufgabe 38 * *(SQL, 14 Minuten)*

In den folgenden Teilaufgaben ist jeweils eine relationale Datenbank durch SQL-`CREATE`-Anweisungen vorgegeben. Sie sollen jeweils eine natürlichsprachlich formulierte Aktion auf der Datenbank in SQL formulieren.

a) *(*, alte Klausuraufgabe, 4 Minuten)*
Betrachten Sie die folgende relationale Datenbank:

```
CREATE TABLE Point (
        id INTEGER NOT NULL,
        x INTEGER NOT NULL,
        y INTEGER NOT NULL,
        CHECK (x >= 0),
        CHECK (y >= 0),
        PRIMARY KEY(id)
);

CREATE TABLE Circle (
        id INTEGER NOT NULL,
        radius INTEGER NOT NULL,
        center INTEGER NOT NULL,
        CHECK (radius >= 0),
        FOREIGN KEY(center) REFERENCES Point(id),
        PRIMARY KEY(id)
);
```

Entwerfen Sie SQL-Anfragen, um folgende Aktionen auf dieser Datenbank auszuführen:

- Einen neuen Punkt mit der ID `5`, der `x`-Koordinate `0` und der `y`-Koordinate `0` anlegen.
- Für den Kreis mit der ID `7` den Mittelpunkt auf diesen Punkt und den Radius auf `10` ändern.

Lösung:

```
INSERT INTO Point VALUES (5, 0, 0);

UPDATE Circle
        SET center = 5, radius = 10
        WHERE id = 7;
```

b) *(*, 6 Minuten)*
Betrachten Sie die folgende relationale Datenbank:

```
CREATE TABLE Book (
        id INTEGER NOT NULL,
        PRIMARY KEY (id)
);

CREATE TABLE Customer (
        id INTEGER NOT NULL,
        PRIMARY KEY (id)
);

CREATE TABLE Loan (
        startDate DATE NOT NULL,
        endDate DATE NOT NULL,
        bookID INTEGER NOT NULL,
        customerID INTEGER NOT NULL,
        CHECK (startDate < endDate),
        FOREIGN KEY (bookID) REFERENCES Book(id),
        FOREIGN KEY (customerID) REFERENCES Customer(id),
        PRIMARY KEY (bookID, customerID)
);
```

Entwerfen Sie SQL-Anfragen, um folgende Aktionen auf dieser Datenbank auszuführen:

- Den Kunden mit der ID `123` aus der Datenbank löschen
- Eine wiederholungsfreie Liste der IDs aller Bücher, die am 20.12.2017 zurückgegeben wurden, ausgeben

Lösung:

```
DELETE FROM Loan
WHERE customerId = 123;

DELETE FROM Customer
WHERE id = 123;

SELECT DISTINCT bookID
FROM Loan
WHERE endDate = '2017-12-20';
```

c) *(*, 4 Minuten)*
Betrachten Sie die folgende relationale Datenbank:

```
CREATE TABLE Patient (
        id INTEGER NOT NULL,
        name VARCHAR(20) NOT NULL,
        PRIMARY KEY (id)
);

CREATE TABLE Doctor (
        id INTEGER NOT NULL,
        name VARCHAR(20) NOT NULL,
        PRIMARY KEY (id)
);

CREATE TABLE Treatment (
        diagnosis VARCHAR(50) NOT NULL,
        patientID INTEGER NOT NULL,
        doctorID INTEGER NOT NULL,
        FOREIGN KEY (patientID) REFERENCES Patient(id),
        FOREIGN KEY (doctorID) REFERENCES Doctor(id),
        PRIMARY KEY (patientID, doctorID)
);
```

Entwerfen Sie eine SQL-Anfrage, um folgende Aktionen auf dieser Datenbank auszuführen:

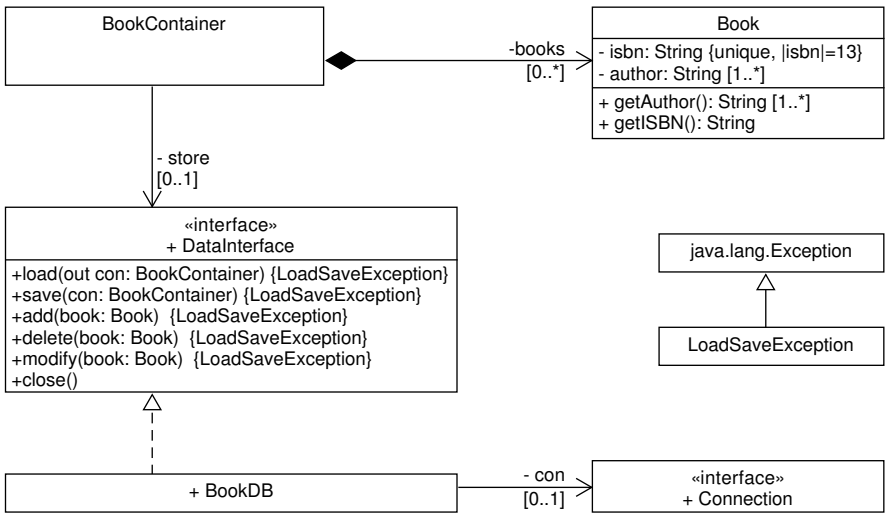
- Eine wiederholungsfreie Liste der Namen der Ärzte und Patienten zu allen Behandlungen wegen Grippe ausgeben

Lösung:

```
SELECT DISTINCT Doctor.name, Patient.name
FROM Treatment, Doctor, Patient
WHERE doctorID = Doctor.id
AND patientID = Patient.id
AND diagonsis = 'Grippe';
```

Aufgabe 39 *** *(Datenbankzugriff implementieren, 20 Minuten)*

Betrachten Sie das folgende UML-Klassendiagramm:



und die folgende Datenbank:

```
CREATE TABLE Book (
      isbn CHAR(13) NOT NULL,
      PRIMARY KEY(isbn)
);

CREATE TABLE BookAuthor (
      author VARCHAR(30) NOT NULL,
      isbn CHAR(13) NOT NULL,
      PRIMARY KEY(isbn,author),
      FOREIGN KEY(isbn) REFERENCES Book(isbn)
);
```

Implementieren Sie in der Klasse `BookDB` die Methode `add`, um ein Buch in der Datenbank zu speichern. Implementieren Sie die Methode `delete`, um ein Buch aus der Datenbank zu löschen. Achten Sie dabei darauf, Ressourcen auf jeden Fall freizugeben, sobald diese nicht mehr benötigt werden.

Lösung:

```
    public void add(Book book) throws LoadSaveException {
        if (con == null)
            return;
        try (PreparedStatement queryBook = con.prepareStatement("INSERT INTO Book VALUES (?)")) {
            queryBook.setString(1, book.getISBN());
            queryBook.executeUpdate();
            try (PreparedStatement queryBookAuthor = con.prepareStatement("INSERT INTO BookAuthor
            ↪  VALUES (?,?)")) {
                queryBookAuthor.setString(2, book.getISBN());
                for (String Author : book.getAuthor()) {
                    queryBookAuthor.setString(1, Author);
                    queryBookAuthor.executeUpdate();
                }
            }
        } catch (SQLException e) {
            throw new LoadSaveException("Insert failed: ", e);
        }
    }
```

```
public void delete(Book book) throws LoadSaveException {
    if (con == null)
        return;
    try (PreparedStatement queryBookAuthor = con.prepareStatement("DELETE FROM bookAuthor WHERE
    ↪  isbn = ?")) {
        queryBookAuthor.setString(1, book.getISBN());
        queryBookAuthor.executeUpdate();
        try (PreparedStatement queryBook = con.prepareStatement("DELETE FROM book WHERE isbn =
        ↪  ?")) {
            queryBook.setString(1, book.getISBN());
            queryBook.executeUpdate();
        }
    } catch (SQLException e) {
        throw new LoadSaveException("Delete failed: ", e);
    }
}
```

Allgemeine Hinweise zur Verbindung zur Lehrstuhl-SQL-Datenbank:
Um sich mit der SQL-Datenbank des Lehrstuhls verbinden zu können, müssen Sie einen SQL-Connector in ihrer Entwicklungsumgebung (z.B. Eclipse oder IntelliJ) einbinden. Den SQL-Connector finden Sie unter https://dev.mysql.com/downloads/connector/j/[1]. zur Angabe im Digicampus zur Verfügung gestellt.

Für Eclipse folgen sie diesen Anweisungen:

1. In Projekt Rechtsklick → Properties
2. in Suche “Buildpath“ eingeben und auf “Java Build Path“ klicken
3. auf Reiter Libraries klicken
4. auf Classpath unten klicken
5. auf “Add external JARs“ klicken
6. in der Ordnerstruktur zum mysql-connector navigieren
7. Apply and Close klicken

Für IntelliJ folgen Sie diesen Anweisungen:

1. File → Project Structure
2. auf Reiter Libraries klicken
3. Mit +-Symbol Java Project Library hinzufügen
4. in der Ordnerstruktur zum mysql-connector navigieren
5. Bestätigen und Änderungen anwenden

[1] Wählen Sie für Windows „Platform Independent“ als Operating System

Aufgabe 40 ** *(Datenbankzugriff implementieren, 27 Minuten)*

Für diese Aufgabe haben wir eine Datenbank `theDatabase` aufgesetzt, die unter anderem die Tabellen `Customer` und `Ticket` und `CustomerTicket` enthält.

Zugriffsinformationen:

- URL:
  `jdbc:mysql://educos-srv01.informatik.uni-augsburg.de:3306/theDatabase?useSSL=false`
- Benutzername: `student`
- Passwort: `inFormatik2`

Beachten Sie, dass nur über das Uni-Netz auf die Datenbank zugegriffen werden kann.[2]
In dieser Aufgabe ist folgende relationale Datenbank durch drei SQL-`CREATE`-Anweisungen vorgegeben:

```
CREATE TABLE Customer (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(45) NOT NULL,
  birthday date,
  PRIMARY KEY (id)
);

CREATE TABLE Ticket (
  id int(11) NOT NULL AUTO_INCREMENT,
  name varchar(45) NOT NULL,
  price float NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE CustomerTicket (
  id int NOT NULL AUTO_INCREMENT,
  customerId int NOT NULL,
  ticketId int NOT NULL,
  boughtOn date NOT NULL,
  amount int NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (customerId) REFERENCES Customer(id),
  FOREIGN KEY (ticketId) REFERENCES Ticket(id)
);
```

a) *(**, 6 Minuten)*
Implementieren Sie eine Methode

```
public static void printCustomers(Connection connection)
```

nach folgenden Vorgaben:

- Nehmen Sie an, dass der Parameter `connection` eine Verbindung zur obigen Datenbank repräsentiert.
- Die Methode soll zeilenweise den Namen und den Geburtstag derjenigen Kunden ausgeben, die einen Geburtstag angegeben haben.

Bei gegebenenfalls auftretenden Ausnahmen soll deren Nachricht auf der Kommandozeile ausgegeben werden.

[2] Von Zuhause aus mittels VPN,
siehe https://www.uni-augsburg.de/de/organisation/einrichtungen/rz/it-services/uaux/wlan/vpn/

Lösung:

```
public static void printCustomers(Connection connection) {
    try (PreparedStatement s = connection.prepareStatement("SELECT name, birthday FROM
    ↪  Customer WHERE birthday IS NOT NULL")) {
        ResultSet r = s.executeQuery();
        while (r.next()) {
            System.out.println(r.getString("name") + ": " + r.getString("birthday"));
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

b) *(**, 7 Minuten)*
Implementieren Sie eine Methode

```
public static int getTicketAmountLastMonth(Connection connection)
```

nach folgenden Vorgaben:

- Nehmen Sie an, dass der Parameter `connection` eine Verbindung zur obigen Datenbank repräsentiert.
- Die Methode soll zurückgeben, wie viele Tickets in den letzten 30 Tagen vor dem Aufrufdatum gekauft wurden.

Bei gegebenenfalls auftretenden Ausnahmen soll deren Nachricht auf der Kommandozeile ausgegeben werden.

Lösung:

```
public static int getTicketAmountLastMonth(Connection connection) {
    try (PreparedStatement s = connection.prepareStatement("SELECT amount FROM " +
            "CustomerTicket WHERE boughtOn > ?")) {
        s.setString(1, LocalDate.now().minusMonths(1).toString());
        ResultSet r = s.executeQuery();
        int sum = 0;
        while (r.next()) {
            sum += r.getInt("amount");
        }
        return sum;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    return -1;
}
```

c) *(**, 8 Minuten)*
Implementieren Sie eine Methode

```
public static double getTicketSumBoughtBy(Connection connection, String customer)
```

nach folgenden Vorgaben:

- Nehmen Sie an, dass der Parameter `connection` eine Verbindung zur obigen Datenbank repräsentiert.
- Die Methode soll zurückgeben, wie viel Geld der Kunde mit dem übergebenen Namen insgesamt ausgegeben hat.

Bei gegebenenfalls auftretenden Ausnahmen soll deren Nachricht auf der Kommandozeile ausgegeben und der Wert -1 zurückgegeben werden.

Lösung:

```
public static double getTicketSumBoughtBy(Connection connection, String customer) {
    try (PreparedStatement s = connection.prepareStatement("SELECT CustomerTicket.amount,
    ↪  Ticket.price FROM Customer, CustomerTicket, Ticket WHERE Customer.name = ? AND
    ↪  Customer.id = CustomerTicket.customerId AND Ticket.id =
    ↪  CustomerTicket.ticketId")) {
        s.setString(1, customer);
        ResultSet r = s.executeQuery();
        double sum = 0;
        while (r.next()) {
            sum += r.getInt("amount") * r.getFloat("price");
        }
        return sum;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    return -1;
}
```

d) *(**, 6 Minuten)*
Schreiben Sie eine Main-Methode, die mit Hilfe eines `Connection`-Objekts eine Verbindung zur Datenbank aufbaut und mit der Sie Ihre Lösungen aus den vorherigen Teilaufgaben testen, indem Sie

- alle Kunden ausgeben, die auch einen Geburtstag angegeben haben,
- ausgeben, wie viele Tickets im letzten Monat gekauft wurden und
- ausgeben, wie viel Geld Charlie Brown für Tickets ausgegeben hat.

Lösung:

```
public static void main(String[] args) {
    String url =
    ↪  "jdbc:mysql://educos-srv01.informatik.uni-augsburg.de:3306/theDatabase?useSSL=false";
    String user = "student";
    String password = "inFormatik2";
    try (Connection connection = DriverManager.getConnection(url, user, password)) {
        printCustomers(connection);
        System.out.println("Tickets bought last month: " +
        ↪  getTicketAmountLastMonth(connection));
        String name = "Charlie Brown";
        System.out.println(name + " spent : " + getTicketSumBoughtBy(connection, name));
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```