

Programmentwicklung 2

Übungsblatt 1

Aufgabe 1

1.1

a)

```
SELECT * FROM todos WHERE id = 1;

CREATE TABLE IF NOT EXISTS todos(
    id INTEGER PRIMARY KEY,
    title TEXT,
    description TEXT
);

INSERT INTO todos
(id, title, description)
VALUES
    (id 1, title 'Dekorieren', description 'Es ist nun endlich so weit! Mit dem 01. November wird es Zeit, zügig die Weihnachtsdekorationen auszupacken.'),
    (id 2, title 'New todo', description NULL),
    (id 3, title 'Weitere Todos für die TodoAPI! eintragen', description NULL),
    (id 4, title 'Backen', description 'Bald sollte ich Weihnachtsplätzchen backen.'),
    (id 13, title 'Die Attribute eines Todo-Objekts für die TodoAPI definieren', description NULL),
    (id 42, title 'Die Geschäftslogik fuer die TodoAPI entwerfen', description '');
```

b)

```
INSERT INTO todos
(id, title, description)
VALUES
    (1, 'Dekorieren', 'Es ist nun endlich so weit! Mit dem 01. November wird es Zeit, zügig die Weihnachtsdekorationen auszupacken.'),
```

c)

```
SELECT description from todos;

SELECT description
FROM todos
WHERE description LIKE '%Weihnacht%';
```

1.2a)

Lösungswort: EntwickLUnGPrOgrAMMII

```

1 package org.example;
2
3 import com.j256.ormlite.field.DatabaseField;
4 import com.j256.ormlite.table.DatabaseTable;
5
6 @DatabaseTable(tableName = "letters")
7 public class Letter {
8     @DatabaseField(id = true)
9     private int id;
10
11    @DatabaseField
12    private String letter;
13
14    public Letter() {}
15
16    public int getId(){ return id;}
17    public String getLetter(){ return letter;}
18
19
20 }
21

```

```

1 package org.example;
2
3 > import ...
4
5
6 public class Main {
7     public static <ConnectionSource> void main(String[] args) throws Exception {
8         String url = "jdbc:mariadb://bilbao.informatik.uni-stuttgart.de/pe2-db-a1";
9         String user = "pe2-nutzer";
10        String password = "pe2-db-a1";
11
12        int[] arrayIndexes = {
13            20, 44, 50, 13, 17, 33, 41,
14            68, 77, 44, 29, 72, 48, 71,
15            37, 48, 11, 69, 5, 65, 65
16        };
17        try (ConnectionSource connectionSource = new JdbcConnectionSource(url, user, password)) {
18            Dao<Letter, Integer> letterDao = DaoManager.createDao(connectionSource, Letter.class);
19
20            StringBuilder word = new StringBuilder();
21
22            for (int id : arrayIndexes) {
23                Letter letter = letterDao.queryForId(id);
24                if (letter != null) {
25                    word.append(letter.getLetter());
26                } else {
27                    word.append("-");
28                }
29            }
30            System.out.println(word);
31        } catch (Exception e) {
32            e.printStackTrace();
33        }
34    }
35 }
36
37
38
39
40

```

(Problem mit den Imports gehabt deswegen rote Tokens)

b)

```

package org.example;

import com.j256.ormlite.dao.Dao;
import com.j256.ormlite.dao.DaoManager;
import com.j256.ormlite.jdbc.JdbcConnectionSource;
import com.j256.ormlite.support.ConnectionSource;

import java.util.List;

▷ public class Main {
▷   public static void main(String[] args) {
      String url = "jdbc:mariadb://bilbao.informatik.uni-stuttgart.de/pe2-db-a1";
      String username = "pe2-nutzer";
      String password = " esJLtfM6ksCT4mCyOS";

      try(ConnectionSource connectionSource = new JdbcConnectionSource(url, username, password)){
          Dao<Letter, Integer> letterDao = DaoManager.createDao(connectionSource, Letter.class);

          String[] letters = {"V", "b", "t"};

          for(String letter : letters){
              List<Letter> results = letterDao.queryForEq( <: "letter", letter.toLowerCase());

              System.out.println("IDs fuer " + letter + ": ");

              if(results.isEmpty()){
                  System.out.println("Keinen Eintrag gefunden");
              }else{
                  for(Letter entry : results){
                      System.out.println(entry.getId());
                  }
              }
              System.out.println();
          }

      } catch (Exception e) {
          e.printStackTrace();
      }
    }
}

```

```

1 package org.example;
2
3 import com.j256.ormlite.field.DatabaseField;
4 import com.j256.ormlite.table.DatabaseTable;
5
6 @DatabaseTable(tableName = "letters") 4 usages
7 public class Letter {
8     @DatabaseField(id = true) 1 usage
9     private int id;
10
11     @DatabaseField 1 usage
12     private String letter;
13
14     public Letter(){}
15
16     public int getId(){return id;} 1 usage
17     public String getLetter(){return letter;} no usages
18 }
19

```

```
↑ 2025-10-27 21:17:34,885 [DEBUG] DaoManager created dao for class class org.example.Letter with reflection
↓ 2025-10-27 21:17:34,894 [DEBUG] StatementBuilder built statement SELECT * FROM `letters` WHERE `letter` = 'v'
→ 2025-10-27 21:17:35,239 [DEBUG] BaseJdbcConnectionSource opened connection to jdbc:mariadb://bilbao.informatik.uni-stuttgart.de/pe2-db-a1
→ 2025-10-27 21:17:35,257 [DEBUG] BaseMappedStatement prepared statement 'SELECT * FROM `letters` WHERE `letter` = 'v'' with 0 args
→ 2025-10-27 21:17:35,283 [DEBUG] SelectIterator starting iterator @1151593579 for 'SELECT * FROM `letters` WHERE `letter` = 'v''
→ 2025-10-27 21:17:35,286 [DEBUG] SelectIterator closed iterator @1151593579 after 2 rows
→ 2025-10-27 21:17:35,287 [DEBUG] StatementExecutor query of 'SELECT * FROM `letters` WHERE `letter` = 'v'' with 0 args returned 2 results
IDs fuer V:
52
78

2025-10-27 21:17:35,288 [DEBUG] StatementBuilder built statement SELECT * FROM `letters` WHERE `letter` = 'b'
2025-10-27 21:17:35,288 [DEBUG] BaseMappedStatement prepared statement 'SELECT * FROM `letters` WHERE `letter` = 'b'' with 0 args
2025-10-27 21:17:35,307 [DEBUG] SelectIterator starting iterator @323326911 for 'SELECT * FROM `letters` WHERE `letter` = 'b''
2025-10-27 21:17:35,307 [DEBUG] SelectIterator closed iterator @323326911 after 3 rows
2025-10-27 21:17:35,307 [DEBUG] StatementExecutor query of 'SELECT * FROM `letters` WHERE `letter` = 'b'' with 0 args returned 3 results
IDs fuer b:
9
32
58

2025-10-27 21:17:35,308 [DEBUG] StatementBuilder built statement SELECT * FROM `letters` WHERE `letter` = 't'
2025-10-27 21:17:35,308 [DEBUG] BaseMappedStatement prepared statement 'SELECT * FROM `letters` WHERE `letter` = 't'' with 0 args
2025-10-27 21:17:35,327 [DEBUG] SelectIterator starting iterator @1270144618 for 'SELECT * FROM `letters` WHERE `letter` = 't''
2025-10-27 21:17:35,327 [DEBUG] SelectIterator closed iterator @1270144618 after 2 rows
2025-10-27 21:17:35,328 [DEBUG] StatementExecutor query of 'SELECT * FROM `letters` WHERE `letter` = 't'' with 0 args returned 2 results
IDs fuer t:
50
76

2025-10-27 21:17:35,337 [DEBUG] BaseJdbcConnectionSource closed connection #334203599
```

IDs für V:

52, 78

IDs für b:

9, 32, 58

IDs für t:

50, 76

c)

```

1 package org.example;
2
3 > import ...
4
5
6 public class Main {
7     public static void main(String[] args) {
8         String url = "jdbc:mysql://siliba.informatik.uni-stuttgart.de/pe2-db-s1";
9         String username = "pe2-nutzer";
10        String password = "esJLtfmeksCT4mCyDS";
11
12        try(DataSource connectionSource = new JdbcConnectionSource(url, username, password)){
13
14            Dao<Letter, Integer> letterDao = DaoManager.createDao(connectionSource, Letter.class);
15
16            double sum = 0;
17            List<Letter> allLetters = letterDao.queryForAll();
18
19            for(Letter letter : allLetters){
20                sum += letter.getId();
21            }
22
23            double average = sum / allLetters.size();
24
25            System.out.println("Durchschnittswert: " + average);
26            System.out.println("Summe: " + sum);
27
28        } catch (Exception e) {
29            e.printStackTrace();
30        }
31    }
32
33 }

```

The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains Java code that connects to a MySQL database and calculates the average ID of all letters. The terminal window shows the execution output, including the average value (50,81707317073171) and the sum (4167). The code is annotated with line numbers 1 through 33.

Summe = 4167

Durchschnittswert = 50,81707317073171

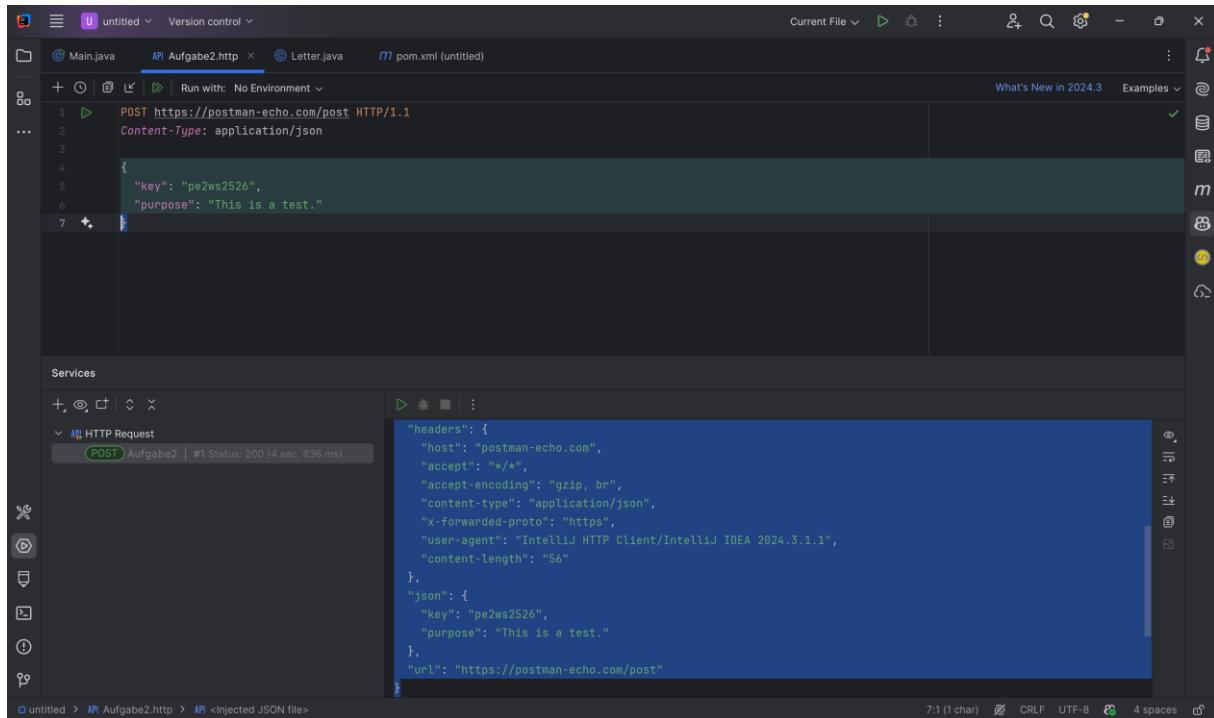
Aufgabe 2

a)

GET <https://api.chucknorris.io/jokes/random?category=sport>

```
{
  "categories": [
    "sport"
  ],
  "created_at": "2020-01-05 13:42:19.576875",
  "icon_url": "https://api.chucknorris.io/img/avatar/chuck-norris.png",
  "id": "2o6183z1rmkus1imghxsug",
  "updated_at": "2020-01-05 13:42:19.576875",
  "url": "https://api.chucknorris.io/jokes/2o6183z1rmkus1imghxsug",
  "value": "Chuck Norris won super bowls VII and VIII singlehandedly before unexpectedly retiring to pursue a career in ass-kicking."
}
```

b)



The screenshot shows the IntelliJ IDEA IDE with the following details:

- Project Structure:** Main.java, Aufgabe2.http, Letter.java, pom.xml (untitled)
- Current File:** Aufgabe2.http
- Content:**

```
POST https://postman-echo.com/post HTTP/1.1
Content-Type: application/json

{
    "key": "pe2ws2526",
    "purpose": "This is a test."
}
```
- HTTP Request Panel:** Shows a successful POST request to Aufgabe2 with status 200.
- Services:** Headers and JSON body are displayed.
- Bottom Status Bar:** 7:1 (1 char), CRLF, UTF-8, 4 spaces.

c)

Create

HTTP Methode: POST

Beschreibung: Diese Methode erstellt eine neue Ressource, also in diesem Fall ein neuen DVD Eintrag in der Datenbank.

Pfad: /dvds

Read 1/2 (getAll)

HTTP Methode:

GET / <collection-name>

Beschreibung: Diese Methode liefert eine Repräsentation der DVDs. Dabei kann ein Format bzw. Filter festgelegt werden durch einen Query Parameter wie z.B. Titel, Kategorie und Alterseinschraenkung.

GET /dvds?category={category}&title={String}&restricted={boolean}

Read 2/2 (getById)

HTTP Methode:

GET /<collection-name>/{\$id}

Beschreibung: Diese Methode ruft eine bestimmte Ressource einer Sammlung ab, also hier eine spezifische DVD anhand ihrer ID.

GET/dvds/{\$id}

Update

HTTP Methode:

PUT /dvds/{id}

Beschreibung: Überschreibt eine DVD bzw. die ganzen Informationen.

Delete

HTTP Methode:

Beschreibung: Diese Methode entfernt eine DVD aus der Datenbank, entsprechend ihrer ID.

Aufgabe 3

a)

Der Statuscode steht dafür, dass eine Methode nicht erlaubt ist. In diesem Fall bedeutet es, dass die Methode PUT nicht auf cats angewendet werden kann. Für das Aufrufen der aller Katzen muss stattdessen eine andere Methode und zwar die GET Methode verwendet werden.

GET /api/v1/cats

b)

Der Statuscode 400 (Bad Request) deutet darauf hin, dass der Server die Anfrage nicht verarbeiten konnte, wobei der Fehler beim Client liegt, da hier die Anfrage falsch formuliert ist. In diesem Fall wurde der Request Body nicht im JSON Format geschrieben. Der Request Body müsste also folgendermaßen aussehen:

```
{  
  "name": "Garfield"  
  "ageInYears" : 40  
  "picUrl" : "https://upload.wikimedia.org/wikipedia/en/e/eb/Garfield_ver6.jpg"  
}
```

Aufgabe 4

Startseite

The screenshot shows the initial landing page of a Vue.js application. At the top, there is a dark header bar with navigation links: Home, About, Cats, and Create Cat. To the left of the header is a vertical sidebar with icons for Home, About, Cats, and Create Cat, each with a plus sign indicating they are collapsible. The main content area has a dark background. It displays the text "Welcome to Your Vue.js App" and "You need to navigate to [cats/](#)". Below this, there is a form with a text input placeholder "Type your name:" and a "Submit" button.

Cats Seite

The screenshot shows the 'Cats' page of the application. The header bar is identical to the start page, with the 'Cats' link being the active one. The main content area is titled "Cats". It displays two entries: "Octocat" (Age: 42 years) and "Grumpy Cat" (Age: 10 years). Each entry includes a circular profile picture, the cat's name, its age, and two buttons: "Edit" and "Delete". The "Edit" button for Octocat is highlighted with a green border, while the "Delete" button for Grumpy Cat is highlighted with a red border.

Cats Seite, nachdem eine neue Katze angelegt wurde.

The screenshot shows the 'Cats' section of a web application. There are three cards displayed:

- Octocat**: Age: 42 years. The image is a cartoon cat with a black body and white paws, standing on a blue base. Buttons for 'Edit' and 'Delete' are below the image.
- Grumpy Cat**: Age: 10 years. The image is a photograph of the famous grumpy cat. Buttons for 'Edit' and 'Delete' are below the image.
- Funny Cat**: Age: 17 years. The image is a close-up of a white cat's face. Buttons for 'Edit' and 'Delete' are below the image.

Cats Seite, nachdem eine Katze bearbeitet wurde.

The screenshot shows the 'Cats' section of a web application. There are three cards displayed:

- Octocat**: Age: 42 years. The image is a cartoon cat with a black body and white paws, standing on a blue base. Buttons for 'Edit' and 'Delete' are below the image.
- Grumpy Cat (bearbeitet)**: Age: 19 years. The image is a photograph of the famous grumpy cat. Buttons for 'Edit' and 'Delete' are below the image.
- Funny Cat**: Age: 17 years. The image is a close-up of a white cat's face. Buttons for 'Edit' and 'Delete' are below the image.

Cats Seite, nachdem eine Katze gelöscht wurde.

Screenshot of a web application interface titled "Cats". The top navigation bar includes links for Home, About, Cats, and Create Cat. On the left, there's a sidebar with icons for Home, About, Cats, Create Cat, and a plus sign. The main content area is titled "Cats" and displays two cards:

- Grumpy Cat (bearbeitet)**
Age: 19 years

[Edit](#) [Delete](#)
- Funny Cat**
Age: 17 years

[Edit](#) [Delete](#)