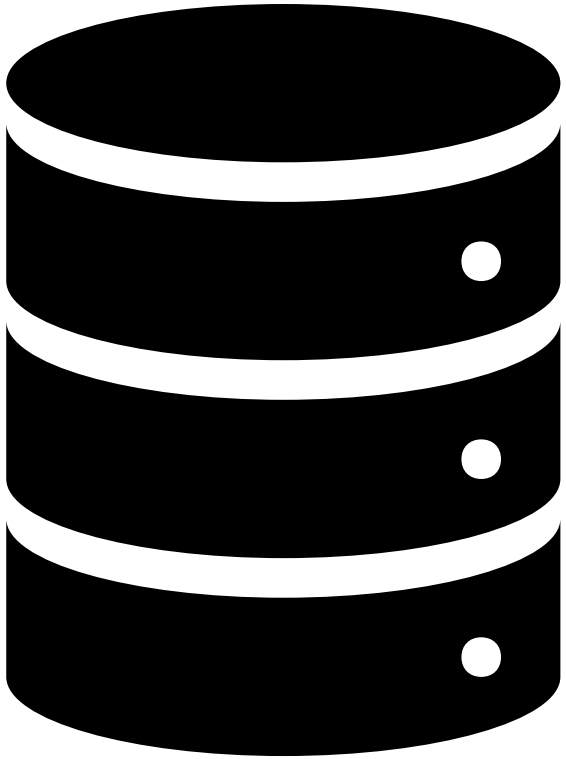


JDBC

März 2025

Plan für die Woche

Montag	Dienstag	Mittwoch	Donnerstag	Freitag
• Aggregate • JDBC • SQL-Injection	• Implementierungs-weisen • BestPractises • MVC-Architektur • ER-Modelle	• Graphische Oberflächen-programmierung • Projekt	• Projekt	• Projekt

Plan für heute

- manuelle Implementierung einer Datenbankverbindung
- Exkurs: Connection-Pooling
- Exkurs: Implementierung einer Datenbankverbindung mit ORM
  - Was ist ORM?
- ER-Modelle
- MVC-Architektur

SMART INDUSTRY CAMPUS

Manuelle Implementierung

Was heißt „manuelle“ Implementierung

SMART INDUSTRY CAMPUS

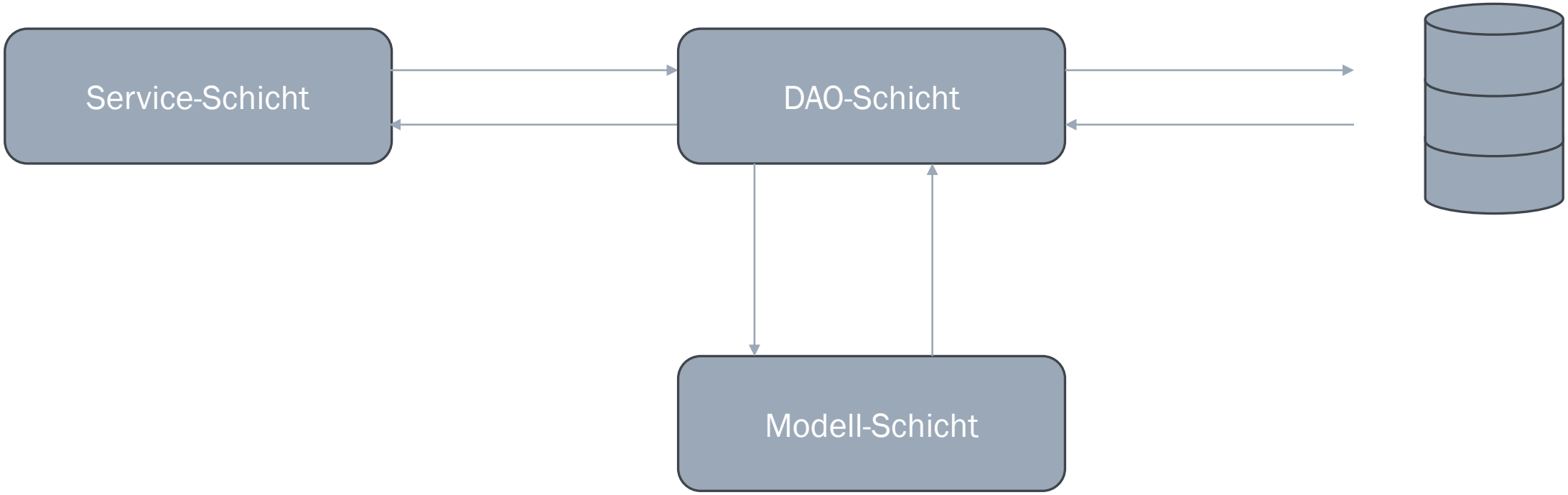
- moderne Technologien können Teile der Implementierung einer Datenbankverbindung abnehmen
- zum Erlernen der Konzepte kann es jedoch sinnvoll sein, vorerst alles manuell zu implementieren
- es gibt verschiedene Ansatzweisen den Code gut zu strukturieren
  - Ein Beispiel wäre das Schichtenmodell

Schichtenmodell

- Modell-Schicht:
  - Repräsentiert eine Entität (z.B. Patient, Book, Author,...) und enthält Attribute sowie grundlegende Validierungen
- Service-Schicht:
  - enthält die Logik und koordinieren die Abläufe zwischen verschiedenen Komponenten
  - Was soll mit den Daten passieren? (Berechnungen, Prüfung, Regeln, ...)
  - Stellt Methoden bereit, die von einem Controller als API genutzt werden können
- DAO (Data Access Object)-Schicht:
  - Ist verantwortlich für den direkten Zugriff auf die Datenbank
  - verbirgt die Datenbank-Logik

Schichtenmodell

– es benötigt eine logische Struktur, um die Anwendung skalierbar, wartbar und sicher zu gestalten



SMART INDUSTRY CAMPUS

DEMO-Schichtenmodell

SMART INDUSTRY CAMPUS

Best Practices

- Verwende ein **Singleton** für die Datenbankverbindung
  - Da das wiederholte Öffnen und Schließen von Verbindungen die Performance beeinflussen kann, sollte es nur eine Instanz der Verbindung geben
  - Eine einzelne Instanz einer Klasse, auf die von mehreren Seiten zugegriffen wird, bezeichnet man als einen Singleton
- Verwende bei großen Anwendungen mit vielen Abfragen ein **Connection-Pooling**
  - HikariCP beispielsweise verbessert die Performance
- Nutze **PreparedStatements**
- Logge alle Fehler und Ereignisse
  - Java hat eine integrierte Klasse: java.util.logging

Beispiel Singleton

Einziger Zugriff über:
DBCreator.getConnection()

SMART INDUSTRY CAMPUS

```
public class DBCreator {
  private static Connection connection;

  private DBCreator() {}

  public static synchronized Connection getConnection() {
      if (connection == null) {
          try {
              String url = "jdbc:mysql://localhost:3306/hospital";
              String user = "root";
              String password = "";
              connection = DriverManager.getConnection(url, user, password);
              System.out.println("Verbindung hergestellt.");
          } catch (SQLException e) {
              e.printStackTrace();
              throw new RuntimeException("Fehler bei der Datenbankverbindung!");
          }
      }
      return connection;
  }
}
```

SMART INDUSTRY CAMPUS

Exkurs: HikariCP

SMART INDUSTRY CAMPUS

Was ist HikariCP?

- ist ein leistungsstarker JDBC Connection Pool für Java-Anwendungen
  - Ist vor allem für Anwendung mit sehr vielen Anfragen gedacht
- sorgt dafür, dass Datenbankverbindungen effizient verwaltet, wiederverwendet und optimiert werden
- im Code ersetzt es die Verwendung des DriverManagers

Wie funktioniert Connection Pooling?

SMART INDUSTRY CAMPUS

- es erstellt eine feste Mindestanzahl an Datenbankverbindungen und verwaltet diese
- ungenutzte Verbindungen werden nicht sofort geschlossen, sondern bleiben verfügbar
- defekte Verbindungen werden erkannt und ersetzt
- die Verbindungsanzahl ist dynamisch
  - Falls besonders viele Verbindungen benötigt werden, werden neue Verbindungen geöffnet

SMART INDUSTRY CAMPUS

Wie integriert man es in Java?

- muss eine Dependency (Bibliothek) für HikariCP heruntergeladen werden
- z.B. Maven

```
<!-- https://mvnrepository.com/artifact/com.zaxxer/HikariCP -->
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>6.2.1</version>
</dependency>
```

Wie integriert man es in Java?

– zusätzlich muss man eine Klasse erstellen, auf die dann zugegriffen wird

```java
public class DatabaseConnection {

    private static HikariDataSource dataSource;
    static {
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl("jdbc:mysql://localhost:3306/hospital");
        config.setUsername("root");
        config.setPassword("");
        config.setMaximumPoolSize(10); // Max. 10 Verbindungen im Pool
        config.setMinimumIdle(2); // Mind. 2 Verbindungen immer aktiv
        config.setIdleTimeout(30000); // Inaktive Verbindungen nach 30s schließen
        config.setMaxLifetime(1800000); // Max. Lebensdauer einer Verbindung: 30min

        dataSource = new HikariDataSource(config);
    }

    public static Connection getConnection() throws SQLException {
        return dataSource.getConnection();
    }
}
```

SMART INDUSTRY CAMPUS

Implementierung mit ORM

SMART INDUSTRY CAMPUS

Was ist ORM?

- steht für Object-Relational Mapping
- ist ein Konzept, um Daten aus einer relationalen Datenbank als Objekte in einer objektorientieren Programmiersprache zu behandeln
- ORM abstrahiert die SQL-Datenbankabfragen, indem es die Datenbanktabellen mit Objekten in der Programmiersprache verbindet
- zur Verwendung von ORM werden meistens Frameworks verwendet:
  - Für Java beispielsweise Hibernate
  - Für C# beispielsweise Entity Framework

SMART INDUSTRY CAMPUS

Vorteile/Nachteile

- Vorteile:
  - Vereinfachte Entwicklung:
    - da einfache SQL-Anfragen nicht mehr manuell geschrieben werden müssen
    - Das Übersetzen (=Mapping) von Datenbankzeilen in Java-Objekte passiert automatisch (Auslesen von einzelnen Zeilen im ResultSet nicht mehr nötig)
  - Schützt vor SQL-Injections
- Nachteile:
  - Komplexität bei großen Datenmodellen

SMART INDUSTRY CAMPUS

Hibernate

- ORM-Framework
- vereinfacht die Interaktion zwischen Java-Objekten und einer relationalen Datenbank
- übernimmt das Mapping zwischen Java-Klassen und Datenbanktabellen

Wie integrieren?

- eine Dependency muss heruntergeladen werden
- z.B. Maven

```
<!--
https://mvnrepository.com/artifact/org.hibernate.orm/hiber
nate-core -->
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.6.11.Final</version>
</dependency>
```

SMART INDUSTRY CAMPUS

Wie integrieren?

– Configuration XML-Datei erstellen

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
      <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/Library</property>
      <property name="hibernate.connection.username">root</property>
      <property name="hibernate.connection.password">password</property>
      <property name="hibernate.hbm2ddl.auto">validate</property>
      <property name="show_sql">false</property>

      <mapping class="de.sic.examples.hibernate.model.Book"/>
      <mapping class="de.sic.examples.hibernate.model.Author"/>
      <mapping class="de.sic.examples.hibernate.model.Member"/>
      <mapping class="de.sic.examples.hibernate.model.BorrowedBook"/>
   </session-factory>
</hibernate-configuration>
```

Wie integrieren?

- HibernateUtils Klasse erstellen

```
package de.sic.examples.hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
   private static final SessionFactory sessionFactory = buildSessionFactory();

   private static SessionFactory buildSessionFactory() {
      try {
         return new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
      } catch (Throwable ex) {
         System.err.println("Initial SessionFactory creation failed." + ex);
         throw new ExceptionInInitializerError(ex);
      }
   }

   public static SessionFactory getSessionFactory() {
      return sessionFactory;
   }

   public static void shutdown() {
      getSessionFactory().close();
   }
}
```

SMART INDUSTRY CAMPUS

DEMO Hibernate

SMART INDUSTRY CAMPUS

ER-Modelle

Grundlagen

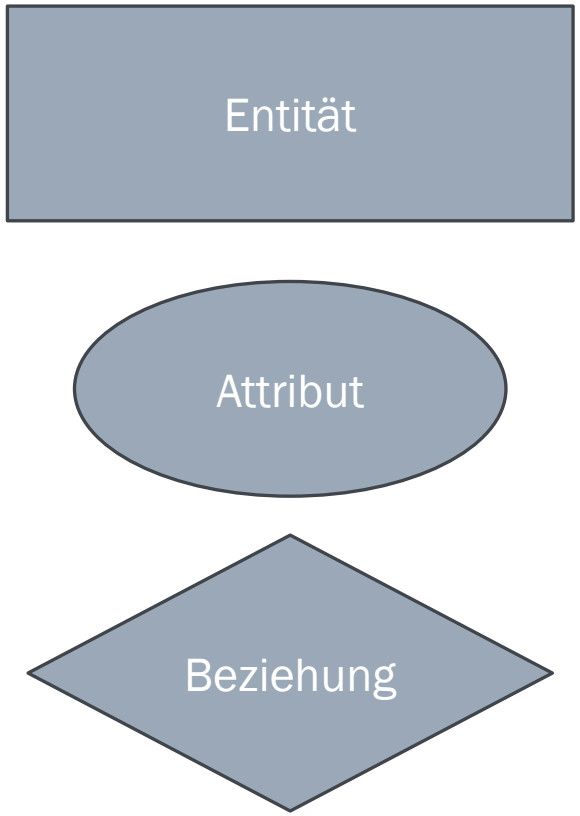
- ER-Modell steht für Entity-Relationship-Modell
  - Es sollen die Beziehungen zwischen den Entitäten dargestellt werden
- gibt verschiedene Möglichkeiten ein ER-Diagramm zu gestalten
- es gibt einfach zu verstehende Diagramme, aber auch komplexere

Beziehungen

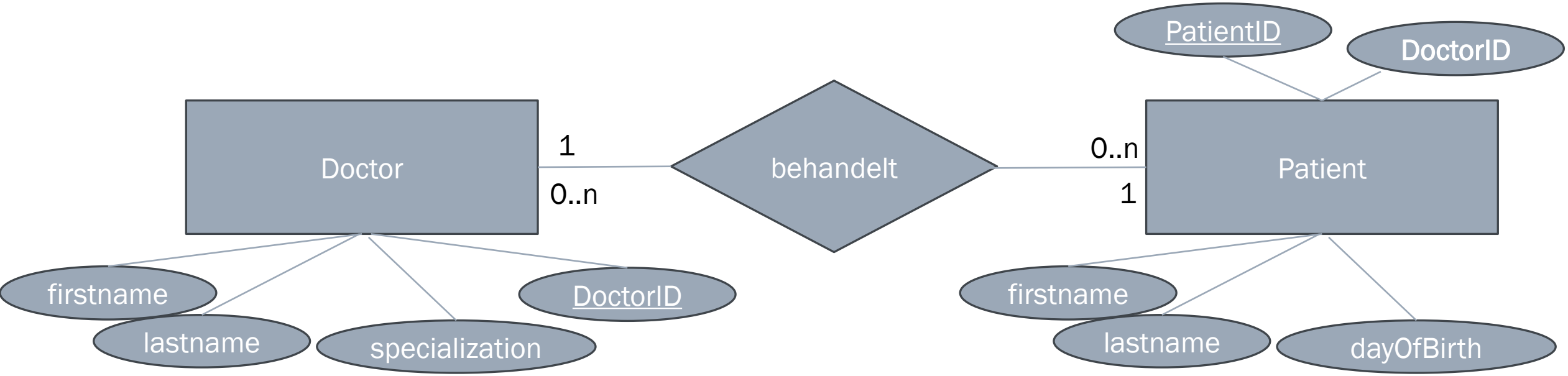
- 1:1 – Beziehung:
  - Beispiel: Jeder Benutzer hat genau ein Benutzerprofil
- 1: n – Beziehung:
  - Beispiel: Ein Autor kann mehrere Bücher geschrieben haben
- m:n – Beziehung:
  - Beispiel: Musikfans können viele Musiker mögen. Ein Musiker kann viele Fans haben
- gibt noch viele weitere

https://www.softwareideas.net/erd-relation-arrows

Einfaches ER-Diagramm

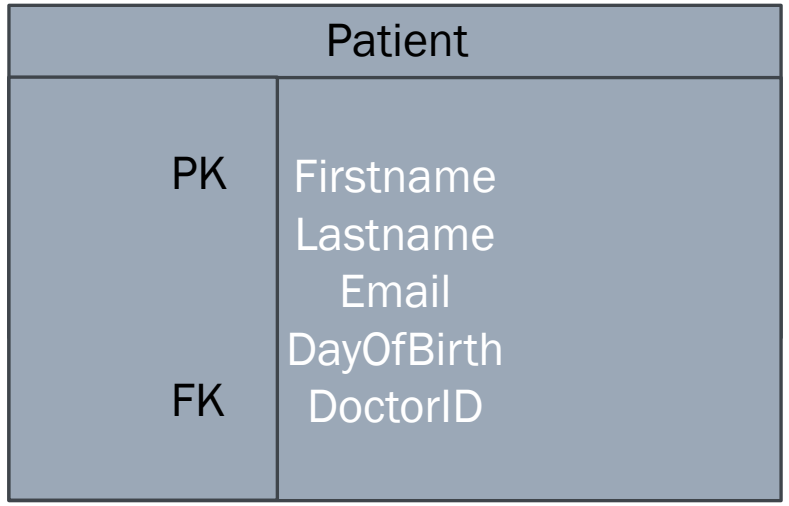


Beispiel


PatientID
DoctorID
Doctor
1
0..n
behandelt
0..n
1
Patient
firstname
lastname
specialization
DoctorID
firstname
lastname
dayOfBirth

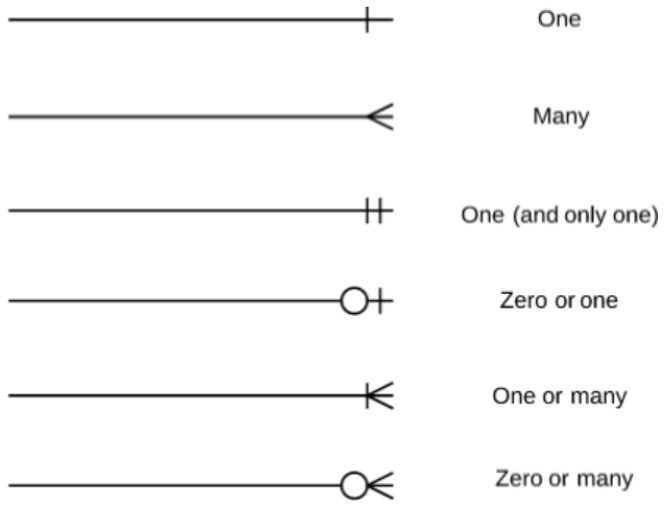
Komplexes ER-Diagramm

- Entität wird direkt mit Attributen und Schlüsseln versehen



Beziehungen

- 1:1 – Beziehung:
  - Beispiel: Jeder Benutzer hat genau ein Benutzerprofil
- 1: n – Beziehung:
  - Beispiel: Ein Autor kann mehrere Bücher geschrieben haben
- m:n – Beziehung:
  - Beispiel: Musikfans können viele Musiker mögen. Ein Musiker kann viele Fans haben

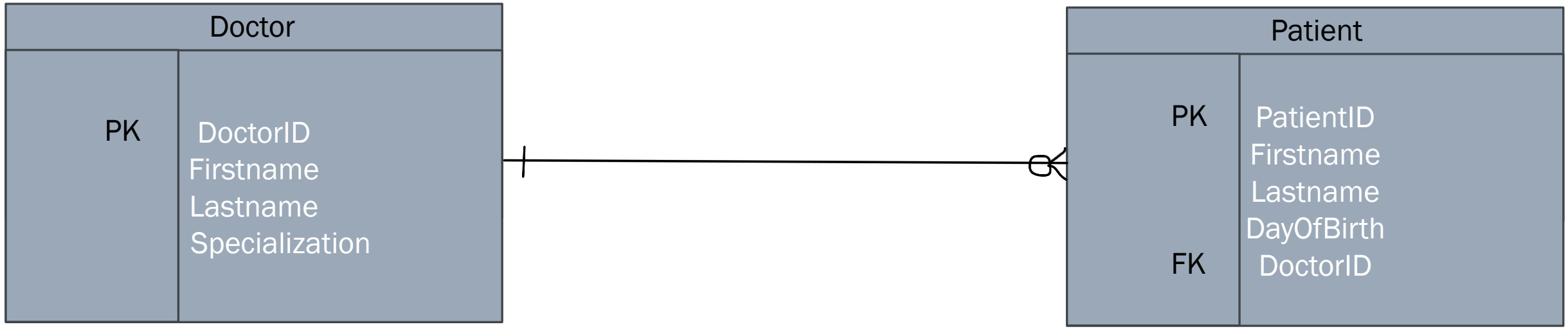


https://www.softwareideas.net/erd-relation-arrows

https://www.lucidchart.com/pages/de/symbole-er-diagramm

Komplexeres ER-Diagramm

- Entität wird direkt mit Attributen und Schlüsseln versehen



https://app.diagrams.net/

SMART INDUSTRY CAMPUS

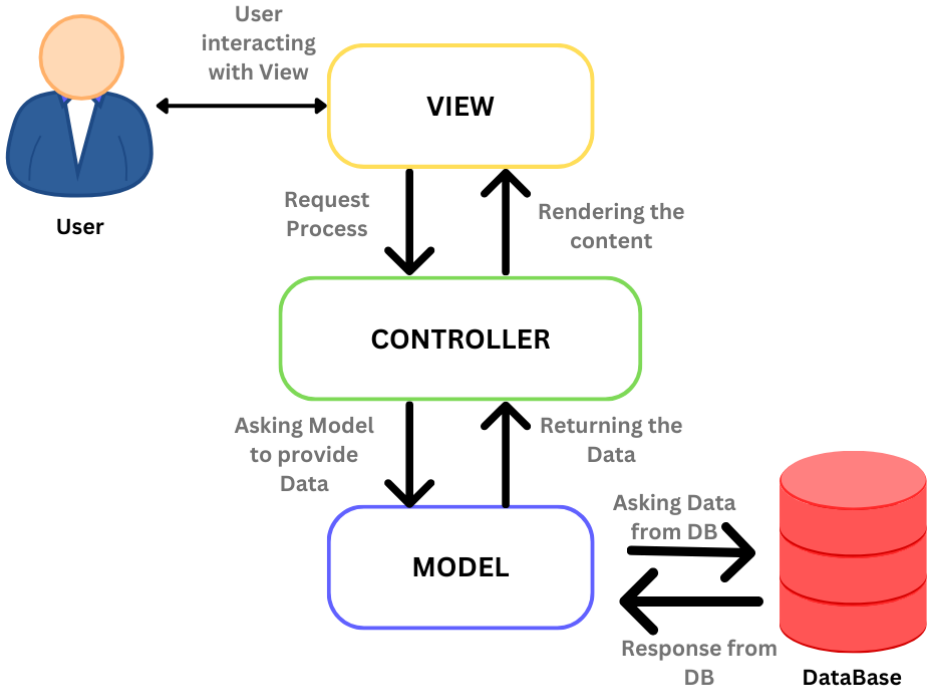
MVC-Pattern

SMART INDUSTRY CAMPUS

Einleitung

- MVC-Architektur steht für Model-View-Controller
- ist ein Entwurfsmuster in der Softwareentwicklung, das die Strukturierung von Anwendungen erleichtert
- es trennt folgende Hauptkomponenten:
  - Model (Daten)
  - View (Benutzeroberfläche)
  - Controller (Steuerung und Verarbeitung der Benutzereingaben)
- es gibt verschiedene Abwandlungen

Einleitung



https://medium.com/@sadikarahmantanisha/the-mvc-architecture-97d47e071eb2

Modell

- repräsentiert eine Datenstruktur der Anwendung
- enthält die Logik und Interaktion mit der Datenbank

View

- präsentiert dem Nutzer die Daten
- erzeugt keine Daten, sondern zeigt sie nur an

Controller

- Verarbeitet Benutzereingaben und steuert die Anwendung
- leitet Anfragen an das Model weiter und aktualisiert die View
- Beispiel:
  - Nutzer klickt auf „Buch ausleihen“, der Controller ruft die Methode im Model auf und aktualisiert die View

SMART INDUSTRY CAMPUS

DEMO MVC-Pattern