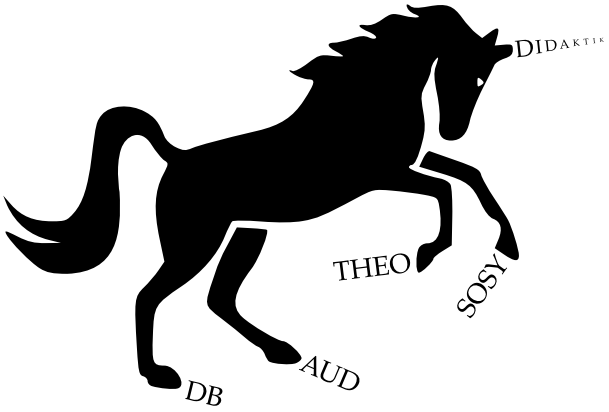
66116
Herbst
2019

Datenbanksysteme / Softwaretechnologie (vertieft)

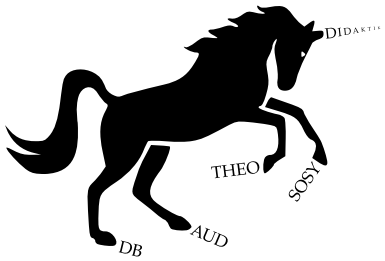
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 1

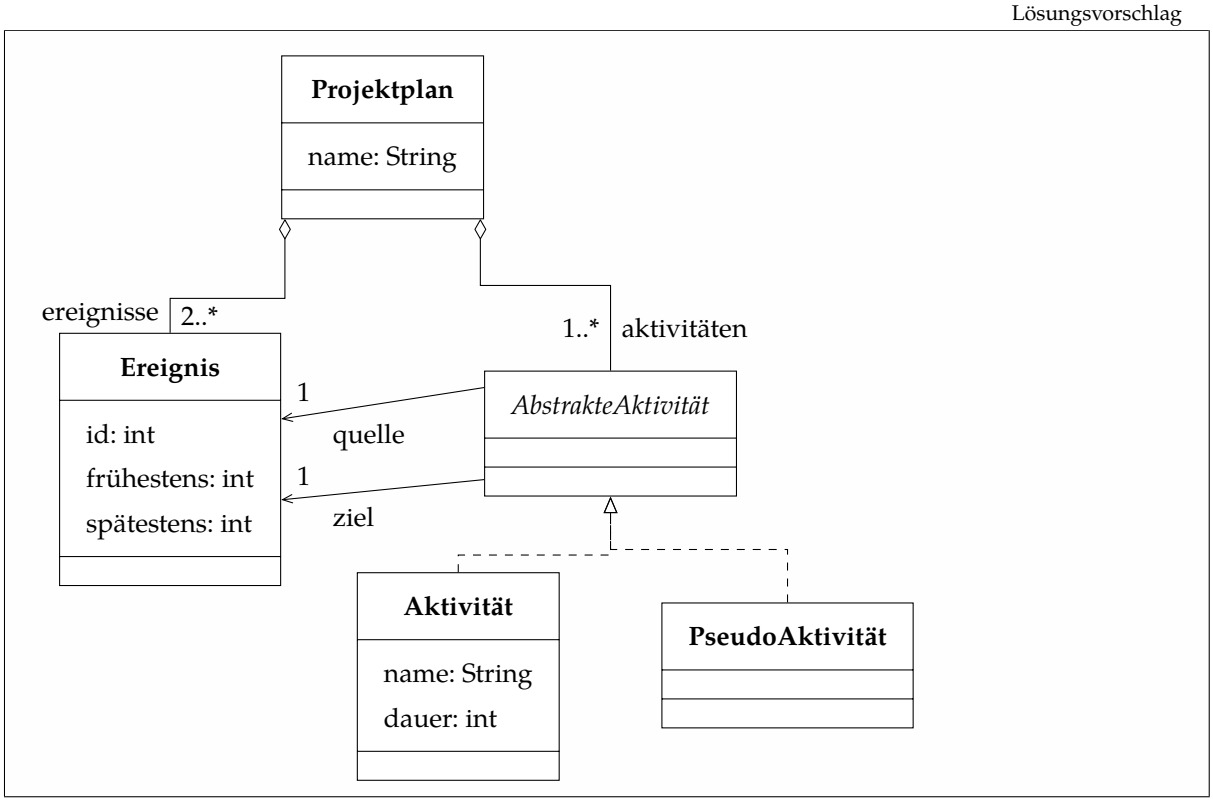
Teilaufgabe Nr. 1

Aufgabe 1 [Critical Path Method]

Ein CPM-Netzwerk („Critical Path Method“) ist ein benannter Projektplan, der aus Ereignissen und Aktivitäten besteht. Ein Ereignis wird durch eine ganze Zahl > 0 identifiziert. Jede Aktivität führt von einem Quellereignis zu einem Zielereignis. Eine reale Aktivität hat einen Namen und eine Dauer (eine ganze Zahl > 0). Eine Pseudoaktivität ist anonym. Ereignisse und Pseudoaktivitäten verbrauchen keine Zeit. Zu jedem Ereignis gibt es einen frühesten und einen spätesten Zeitpunkt (eine ganze Zahl > 0), deren Berechnung nicht Gegenstand der Aufgabe ist.

(a) Erstellen Sie ein UML-Klassendiagramm zur Modellierung von CPM-Netzwerken. Geben Sie für Attribute jeweils den Namen und den Typ an. Geben Sie für Assoziationen den Namen und für jedes Ende den Rollennamen und die Multiplizität an. Nutzen Sie ggf. abstrakte Klassen, Vererbung, Komposition oder Aggregation. Verzichten Sie auf Operationen und Sichtbarkeiten.

Lösungsvorschlag



(b) Erstellen Sie für das Klassendiagramm aus a) und das Beispiel aus der Aufgabenstellung ein Objektdiagramm. Geben Sie Rollennamen nur an, wenn es notwendig ist, um die Enden eines Links (Instanz einer Assoziation) zu unterscheiden.

Lösungsvorschlag

Diese Aufgabe hat noch keine Lösung. Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.

Aufgabe 3 [White-Box-Tests]

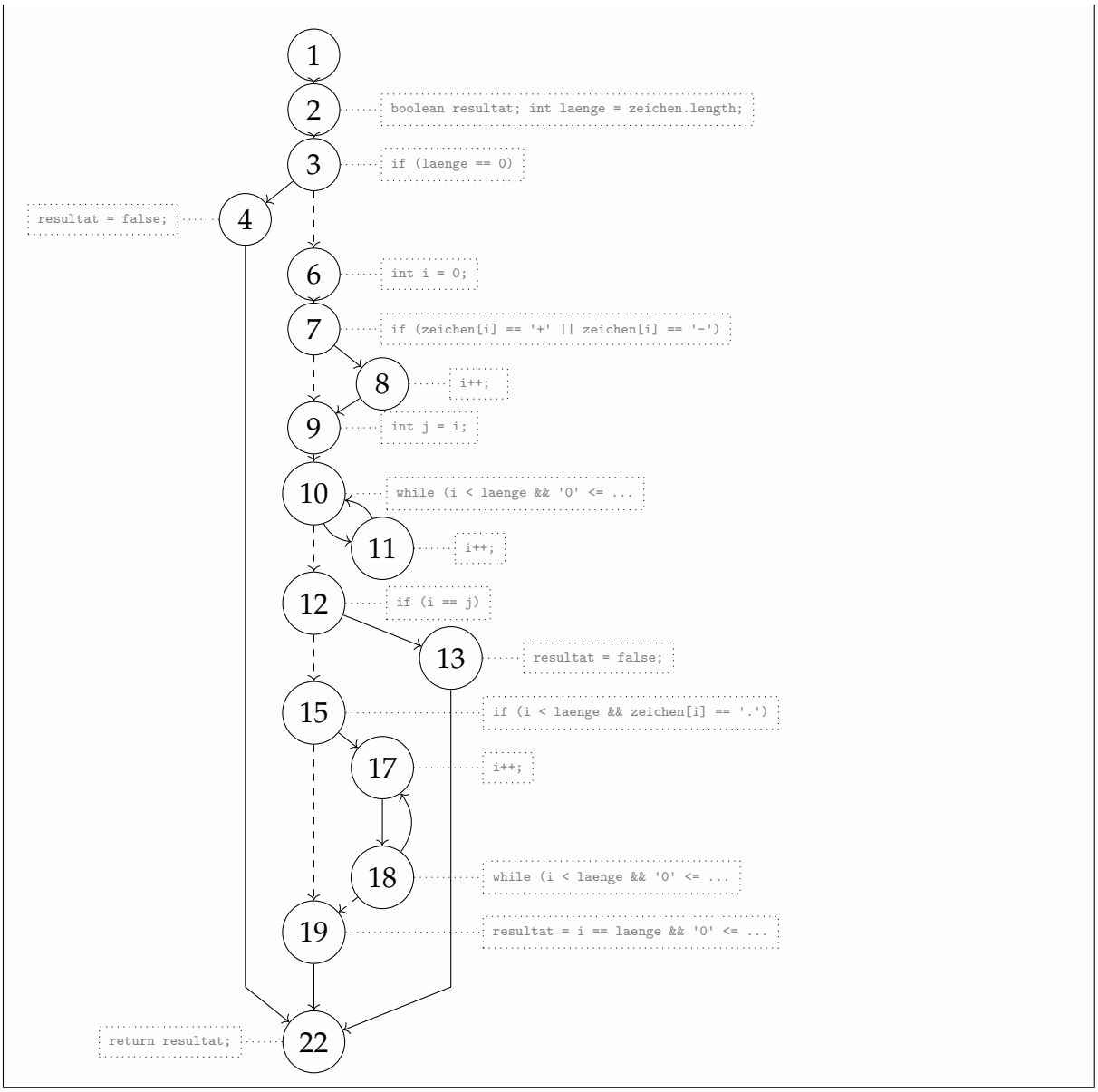
Eine Dezimalzahl hat ein optionales Vorzeichen, dem eine nichtleere Sequenz von Dezimalziffern folgt. Der anschließende gebrochene Anteil ist optional und besteht aus einem Dezimalpunkt, gefolgt von einer nichtleeren Sequenz von Dezimalziffern.

Die folgende Java-Methode erkennt, ob eine Zeichenfolge eine Dezimalzahl ist:

```
public static boolean istDezimalzahl(char[] zeichen) { // 1
  boolean resultat; int laenge = zeichen.length; // 2
  if (laenge == 0) // 3
    resultat = false; // 4
  else { // 5
    int i = 0; // 6
    if (zeichen[i] == '+' || zeichen[i] == '-') // 7
      i++; // 8
    int j = i; // 9
    while (i < laenge && '0' <= zeichen[i] && zeichen[i] <= '9') // 10
      i++; // 11
    if (i == j) // 12
      resultat = false; // 13
    else { // 14
      if (i < laenge && zeichen[i] == '.') // 15
        do // 16
          i++; // 17
        while (i < laenge && '0' <= zeichen[i] && zeichen[i] <= '9'); // 18
      resultat = i == laenge && '0' <= zeichen[i - 1] && zeichen[i - 1] <= '9'; //
↪ 19
    } // 20
  } // 21
  return resultat; // 22
}
```

(a) Konstruieren Sie zu dieser Methode einen Kontrollflußgraphen. Markieren Sie dessen Knoten mit Zeilennummern des Quelltexts.

Lösungsvorschlag



(b) Geben Sie eine minimale Testmenge an, die das Kriterium der Knotenüberdeckung erfüllt. Geben Sie für jeden Testfall den durchlaufenen Pfad in der Notation $1 \to 2 \to \ldots$ an.

Lösungsvorschlag

- „“:
  (1) - (2) - (3) - (4) - (22)
- „1“:
  (1) - (2) - (3) - (6) - (7) - (9) - (10) - (11) - (10) - (12) - (15) - (19) - (22)
- „+1.0“:
  (1) - (2) - (3) - (6) - (7) - (8) - (9) - (10) - (11) - (10) - (12) - (15) - (17) - (18) - (19) - (22)
- „x“:
  (1) - (2) - (3) - (6) - (7) - (9) - (10) - (12) - (13) - (22)

(c) Verfahren Sie wie in b) für das Kriterium der Kantenüberdeckung.

Lösungsvorschlag

> Diese Aufgabe hat noch keine Lösung. Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.

(d) Wie stehen die Kriterien der Knoten- und Kantenüberdeckung zueinander in Beziehung? Begründen Sie Ihre Anwort.

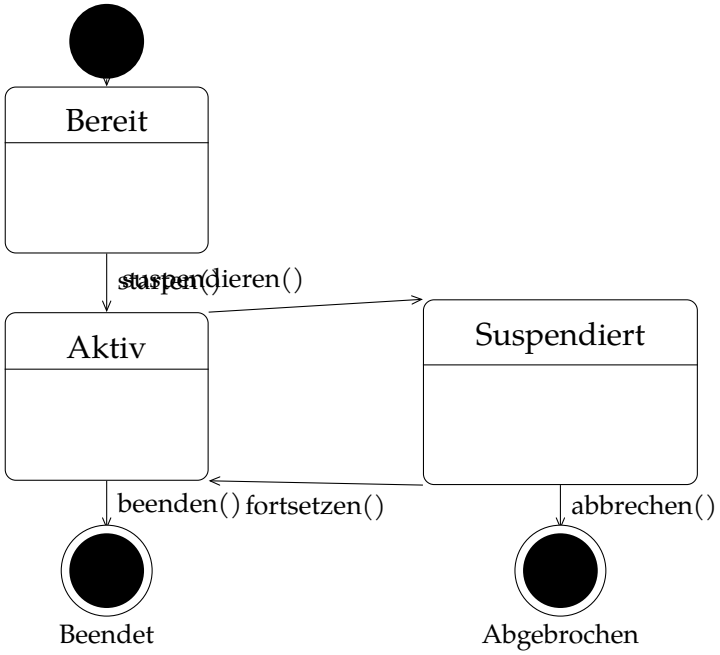
Hinweis: Eine Testmenge ist minimal, wenn es keine andere Testmenge mit einer kleineren Zahl von Testfällen gibt. Die Minimalität braucht nicht bewiesen zu werden.

Lösungsvorschlag

> Die Kantenüberdeckung fordert, dass jede *Kante* des Kontrollflussgraphen von mindestens einem Testfall durchlaufen werden muss. Um das Kriterium zu erfüllen, müssen die Testfälle so gewählt werden, dass jede Verzweigungsbedingung mindestens *einmal wahr* und mindestens *einmal falsch* wird. Da hierdurch alle Knoten ebenfalls mindestens einmal besucht werden müssen, ist die *Anweisungsüberdeckung* in der Zweigüberdeckung *vollständig enthalten*.

Aufgabe 4: (Entwurfsmuster) [Zustand-Entwurfsmuster bei Verwaltung von Prozessen]

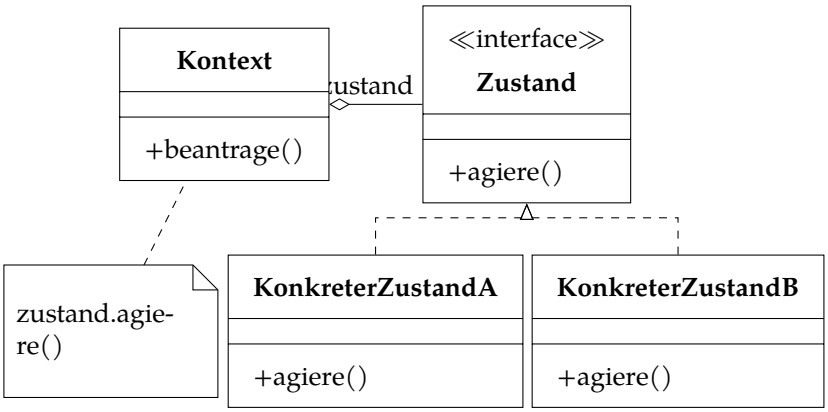
Zu den Aufgaben eines Betriebssystems zählt die Verwaltung von Prozessen. Jeder Prozess durchläuft verschiedene Zustände; Transitionen werden durch Operationsaufrufe ausgelöst. Folgendes Zustandsdiagramm beschreibt die Verwaltung von Prozessen:



Implementieren Sie dieses Zustandsdiagramm in einer Programmiersprache Ihrer Wahl mit Hilfe des Zustandsmusters; geben Sie die gewählte Sprache an. Die Methoden für die Transitionen sollen dabei die Funktionalität der Prozessverwaltung simulieren, indem der Methodenaufruf auf der Standardausgabe protokolliert wird. Falls Transitionen im aktuellen Zustand undefiniert sind, soll eine Fehlermeldung ausgegeben werden.

Exkurs: Zustand-(State)-Entwurfsmuster

UML-Klassendiagramm



Teilnehmer

Kontext (Context) definiert die clientseitige Schnittstelle und verwaltet die separaten Zustandsklassen.

State (Zustand) definiert eine einheitliche Schnittstelle aller Zustandsobjekte und implementiert gegebenenfalls ein Standardverhalten.

KontreterZustand (ConcreteState) implementiert das Verhalten, das mit dem Zustand des Kontextobjektes verbunden ist.

Implementierung in der Programmiersprache „Java“:

Methoden	Zustände	Klassennamen
	Bereit	ZustandBereit
starten(), fortsetzen()	Aktiv	ZustandAktiv
suspendieren()	Suspendiert	ZustandSuspendiert
beenden()	Beendet	ZustandBeendet
abbrechen()	Abgebrochen	ZustandAbgebrochen

```
/**
 * Entspricht der „Kontext"-Klasse in der Terminologie der „Gang of
 * Four".
 */
public class Prozess {
```

```
  private ProzessZustand aktuellerZustand;

  public Prozess() {
    aktuellerZustand = new ZustandBereit(this);
  }

  public void setzeZustand(ProzessZustand zustand) {
    aktuellerZustand = zustand;
  }

  public void starten() {
    aktuellerZustand.starten();
  }

  public void suspendieren() {
    aktuellerZustand.suspendieren();
  }

  public void fortsetzen() {
    aktuellerZustand.fortsetzen();
  }

  public void beenden() {
    aktuellerZustand.beenden();
  }

  public void abbrechen() {
    aktuellerZustand.abbrechen();
  }

  public static void main(String[] args) {
    Prozess prozess = new Prozess();
    prozess.starten();
    prozess.suspendieren();
    prozess.fortsetzen();
    prozess.beenden();
    prozess.starten();

    // Ausgabe:
    // Der Prozess ist im Zustand „bereit"
    // Der Prozess wird gestartet.
    // Der Prozess ist im Zustand „aktiv"
    // Der Prozess wird suspendiert.
    // Der Prozess ist im Zustand „suspendiert"
    // Der Prozess wird fortgesetzt.
    // Der Prozess ist im Zustand „aktiv"
    // Der Prozess wird beendet.
    // Der Prozess ist im Zustand „beendet"
    // Im Zustand „beendet" kann die Transition „starten" nicht ausführt werden!
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/Prozess.java`

```java
/**
 * Entspricht der „Zustand"-Klasse in der Terminologie der "Gang of
 * Four".
 */
abstract class ProzessZustand {

  Prozess prozess;

  String zustand;

  public ProzessZustand(String zustand, Prozess prozess) {
    this.zustand = zustand;
    this.prozess = prozess;
    System.out.println(String.format("Der Prozess ist im Zustand „%s"", zustand));
  }

  private void gibFehlermeldungAus(String transition) {
    System.err.println(
        String.format("Im Zustand „%s" kann die Transition „%s" nicht ausführt werden!",
            zustand, transition));
  }

  public void starten() {
    gibFehlermeldungAus("starten");
  }

  public void suspendieren() {
    gibFehlermeldungAus("suspendieren");
  }

  public void fortsetzen() {
    gibFehlermeldungAus("fortsetzen");
  }

  public void beenden() {
    gibFehlermeldungAus("beenden");
  }

  public void abbrechen() {
    gibFehlermeldungAus("abbrechen");
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ProzessZustand.java

```java
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der "Gang of
 * Four".
 */
public class ZustandAbgebrochen extends ProzessZustand {

  public ZustandAbgebrochen(Prozess prozess) {
    super("abgebrochen", prozess);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandAbgebrochen.java`

```java
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der „Gang of
 * Four".
 */
public class ZustandAktiv extends ProzessZustand {

  public ZustandAktiv(Prozess prozess) {
    super("aktiv", prozess);
  }

  public void suspendieren() {
    System.out.println("Der Prozess wird suspendiert.");
    prozess.setzeZustand(new ZustandSuspendiert(prozess));
  }

  public void beenden() {
    System.out.println("Der Prozess wird beendet.");
    prozess.setzeZustand(new ZustandBeendet(prozess));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandAktiv.java`

```java
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der „Gang of
 * Four".
 */
public class ZustandBeendet extends ProzessZustand {

  public ZustandBeendet(Prozess prozess) {
    super("beendet", prozess);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandBeendet.java`

```java
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der „Gang of
 * Four".
 */
public class ZustandBereit extends ProzessZustand {

  public ZustandBereit(Prozess prozess) {
    super("bereit", prozess);
  }

  public void starten() {
    System.out.println("Der Prozess wird gestartet.");
    prozess.setzeZustand(new ZustandAktiv(prozess));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandBereit.java`

```java
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der „Gang of
 * Four".
 */
public class ZustandSuspendiert extends ProzessZustand {

  public ZustandSuspendiert(Prozess prozess) {
    super("suspendiert", prozess);
  }

  public void fortsetzen() {
    System.out.println("Der Prozess wird fortgesetzt.");
    prozess.setzeZustand(new ZustandAktiv(prozess));
  }

  public void abbrechen() {
    System.out.println("Der Prozess wird abgebrochen.");
    prozess.setzeZustand(new ZustandAbgebrochen(prozess));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandSuspendiert.java`

Teilaufgabe Nr. 2

Aufgabe 1 [Wissensfragen]

Antworten Sie kurz und prägnant.

(a) Nennen Sie einen Vorteil und einen Nachteil der Schichtenarchitektur.

Lösungsvorschlag

Vorteil

Physische Datenunabhängigkeit:

Die interne Ebene ist von der konzeptionellen und externen Ebene getrennt.

Physische Änderungen, z. B. des Speichermediums oder des Datenbankprodukts, wirken sich nicht auf die konzeptionelle oder externe Ebene aus.

Logische Datenunabhängigkeit:

Die konzeptionelle und die externe Ebene sind getrennt. Dies bedeutet, dass Änderungen an der Datenbankstruktur (konzeptionelle Ebene) keine Auswirkungen auf die externe Ebene, also die Masken-Layouts, Listen und Schnittstellen haben.

a

Nachteil

Overhead durch zur Trennung der Ebenen benötigten Schnittstellen

[a] https://de.wikipedia.org/wiki/ANSI-SPARC-Architektur

(b) Wie ermöglicht es ein Datenbankensystem, verschiedene Sichten darzustellen?

Lösungsvorschlag

Die Sichten greifen auf die zwei darunterliegenden Abstraktionsebenen eines Datenbanksystems zu, nämlich auf die logische Ebene und die logische Ebene greift auf die physische Ebene zu.

(c) Was beschreibt das Konzept der Transitiven Hülle? Erklären Sie dies kurz und nennen Sie ein Beispiel für (1) die Transitive Hülle eines Attributes bei funktionalen Abhängigkeiten und (2) die Transitive Hülle einer SQL-Anfrage.

Lösungsvorschlag

Die transitive Hülle einer Relation R mit zwei Attributen A und B gleichen Typs ist definitert als

Sie enthält damit alle Tupel (a, b), für die ein Pfad beliebiger Länge k in R existiert.

Berechnung rekursiver Anfragen (z. B. transitive Hülle) über rekursiv definierte Sichten (Tabellen)[a]

[a] https://dbs.uni-leipzig.de/file/dbs2-ss16-kap4.pdf

(d) Nennen Sie zwei Indexstrukturen und beschreiben Sie jeweils ihren Vorteil.

Lösungsvorschlag

In Hauptspeicher-Datenbanksystemen werden oft Hashtabellen verwendet, um effizierte Punkt-Abfragen (exact Match) zu unterstützen.

Wenn auch Bereichs-Abfragen (range queries) vorkommen, werden zumeister balancierte Suchbäume - AVL- oder rot/schwarz-Bäume - verwendet.

a

[a] https://de.wikipedia.org/wiki/Indexstruktur

Aufgabe 2: (ER-Modellierung) [Schule Hogwarts aus Harry Potter]

Entwerfen Sie ein ER-Diagramm für eine Schule aus einer imaginären Film-Reihe. Geben Sie alle Attribute an und unterstreichen Sie Schlüsselattribute. Für die Angabe der Kardinalitäten von Beziehungen soll die Min-Max-Notation verwendet werden. Führen Sie wenn nötig einen Surrogatschlüssel ein.

An der Schule werden Schüler ausgebildet. Sie haben einen Namen, ein Geschlecht und ein Alter. Da die Schule klein ist, ist der Name eindeutig. Jeder Schüler ist Teil seines Jahrgangs, bestimmt durch Jahr und Anzahl an Schüler (Jahrgänge ohne Schüler sind erlaubt), und besucht mit diesem Kurse. Dabei wird jeder Kurs von min. einem Jahrgang besucht und jeder Jahrgang hat zwischen 2 und 5 Kurse. Kurse haben einen Veranstaltungsort und einen Namen.

Außerdem wird jeder Schüler einem von vier Häusern zugeordnet. Diese Häuser sind Gryffindor, Slytherin, Hufflepuff und Ravenclaw. Jedes Haus hat eine Anzahl an Mitgliedern.

Um die Organisation an der Schule zu erleichtern, gibt es pro Haus einen Vertrauensschiiler und pro Jahrgang einen Jahrgangssprecher. AuBerdem können Schüler Quidditch spielen. Dabei können sie die Rollen Sucher, Treiber, Jäger und Hüter spielen. Jedes Haus der Schule hat eine Mannschaft. Diese besteht aus genau einem Sucher, einem Hüter, drei Jäger und zwei Treiber. Jedes Jahr gibt es an der Schule eine Trophäe zu gewinnen. Diese ist abhängig von der Mannschaft und weiterhin durch das Jahr identifiziert.

Aufgabe 3 [Game of Thrones]

Formulieren Sie folgende Anfragen in SQL gegen die angegebene Datenbank aus einer imaginären Serie.

```
Figur : {[ Id, Name, Schwertkunst, Lebendig, Titel ]}

gehört_zu : {[ Id, Familie, FK (Id) references Figur(Id), FK (Familie) references Familie(Id)

]}

Familie : {[ Id, Name, Reichtum, Anführer ]}

Drache : {[ Name, Lebendig ]}

besitzt : {[ Id, Name, FK (Id) references Figur(Id), FK (Name) references Drache(Name) ]}

Festung : {[ Name, Ort, Ruine ]}

besetzt : {[ Familie, Festung, FK (Familie) references Familie(Id), FK (Festung) references Festung(Nam

]}

lebt : {[ Id, Name, FK (Id) references Figur(Id), FK (Name) references Festung(Name) ]}
```

```sql
CREATE TABLE Figur (
  Id integer PRIMARY KEY,
  Name VARCHAR(20),
  Schwertkunst integer,
  Lebendig boolean,
  Titel VARCHAR(50)
);

CREATE TABLE Familie (
  Id integer PRIMARY KEY,
  Name VARCHAR(20),
  Reichtum numeric(11,2),
  Anführer VARCHAR(20)
);

CREATE TABLE gehört_zu (
  Id integer REFERENCES Figur(id),
  Familie integer REFERENCES Familie(id)
```

```
);

CREATE TABLE Drache (
  Name VARCHAR(20) PRIMARY KEY,
  Lebendig boolean
);

CREATE TABLE besitzt (
  Id integer REFERENCES Figur(Id),
  Name VARCHAR(20) REFERENCES Drache(Name)
);

CREATE TABLE Festung (
  Name VARCHAR(20) PRIMARY KEY,
  Ort VARCHAR(30),
  Ruine boolean
);

CREATE TABLE besetzt (
  Familie integer REFERENCES Familie(Id),
  Festung VARCHAR(20) REFERENCES Festung(Name)
);

CREATE TABLE lebt (
  Id integer REFERENCES Figur(Id),
  Name VARCHAR(20) REFERENCES Festung(Name)
);

INSERT INTO Figur VALUES
  (1, 'Eddard Stark', 5, FALSE, 'Lord von Winterfell'),
  (2, 'Rodd Stark', 4, FALSE, 'Lord von Winterfell'),
  (3, 'Tywin Lennister', 5, FALSE, 'Lord von Casterlystein'),
  (4, 'Cersei Lennister', 2, TRUE, 'Lady von Casterlystein'),
  (5, 'Brandon Stark', 0, TRUE, 'König der Andalen'),
  (6, 'Jon Schnee', 5, TRUE, 'König-jenseits-der-Mauer');

INSERT INTO Familie VALUES
  (1, 'Haus Stark', 76873.12, 'Eddard Stark'),
  (2, 'Haus Lennister', 82345.43, 'Tywin Lennister');

INSERT INTO gehört_zu VALUES
  (1, 1),
  (2, 1),
  (3, 2),
  (4, 2),
  (5, 1),
  (6, 1);

INSERT INTO Festung VALUES
  ('Roter Bergfried', 'Westeros', FALSE),
  ('Casterlystein', 'Westeros', FALSE),
  ('Winterfell', 'Westeros', FALSE);

INSERT INTO besetzt VALUES
  (1, 'Winterfell'),
```

```
  (2, 'Roter Bergfried'),
  (2, 'Casterlystein');

INSERT INTO lebt VALUES
  (1, 'Winterfell'),
  (2, 'Winterfell'),
  (3, 'Casterlystein'),
  (4, 'Roter Bergfried'),
  (5, 'Winterfell'),
  (6, 'Winterfell');
```

(a) Geben Sie für alle Figuren an, wie oft alle vorhandenen Titel vorkommen.

Lösungsvorschlag

```
SELECT count(*) AS Anzahl, f.Titel
FROM Figur f
GROUP BY f.Titel;
```

```
 anzahl |          titel
--------+--------------------------
      1 | König der Andalen
      2 | Lord von Winterfell
      1 | Lord von Casterlystein
      1 | König-jenseits-der-Mauer
      1 | Lady von Casterlystein
(5 rows)
```

(b) Welche Figuren (Name ist gesucht) kommen aus „Kings Landing“?

Lösungsvorschlag

```
SELECT
  f.Name
FROM
  Figur f,
  Festung b,
  lebt l
WHERE
  b.Name = l.Name AND
  f.Id = l.Id AND
  b.Ort = 'Königsmund';
```

(c) Geben Sie für jede Familie (Name) die Anzahl der zugehörigen Charaktere und Festungen an.

Lösungsvorschlag

```
SELECT
  f.Name,
  (SELECT COUNT(*) FROM Figur fi, gehört_zu ge WHERE fi.id = ge.id AND ge.Familie
↪  = f.id) AS Anzahl_Charaktere,
  (SELECT COUNT(*) FROM Festung fe, besetzt be WHERE fe.Name = be.Festung AND
↪  be.Familie = f.id) AS Anzahl_Festungen
FROM
  Familie f;
```

(d) Gesucht sind die besten fünf Schwertkämpfer aus Festungen aus dem Ort „Westeros“. Es soll der Name, die Schwertkunst und die Platzierung ausgegeben werden. Die Ausgabe soll nach der Platzierung sortiert erfolgen.

Lösungsvorschlag

```
-- Problem: Es gibt 3 mal 3. Platz und nicht 3 mal 1. Platz
SELECT f1.Name, f1.Schwertkunst, COUNT(*) FROM Figur f1, Figur f2
WHERE f1.Schwertkunst <= f2.Schwertkunst
GROUP BY f1.Name, f1.Schwertkunst
ORDER BY COUNT(*)
LIMIT 5;
```

(e) Schreiben Sie eine Anfrage, die alle Figuren löscht, die tot sind. Das Attribut *Lebendig* kann dabei die Optionen „ja“ und „nein“ annehmen.

Lösungsvorschlag

```
-- Nur zu Testzwecken auflisten:
SELECT * FROM Figur;
SELECT * FROM gehört_zu;

-- PostgreSQL unterstützt kein DELETE JOIN
-- DELETE f, g, b, l FROM Figur AS f
--  JOIN gehört_zu AS g ON f.id = g.id
--  JOIN besitzt AS b ON f.id = b.id
--  JOIN lebt AS l ON f.id = l.id
-- WHERE f.Lebendig = FALSE;

DELETE FROM gehört_zu WHERE id IN (SELECT id FROM Figur WHERE Lebendig = FALSE);
DELETE FROM besitzt WHERE id IN (SELECT id FROM Figur WHERE Lebendig = FALSE);
DELETE FROM lebt WHERE id IN (SELECT id FROM Figur WHERE Lebendig = FALSE);
DELETE FROM Figur WHERE Lebendig = FALSE;

-- Nur zu Testzwecken auflisten:
SELECT * FROM Figur;
SELECT * FROM gehört_zu;
```

(f) Löschen Sie die Spalten „Lebendig“ aus der Datenbank.

Lösungsvorschlag

```
ALTER TABLE Figur DROP COLUMN Lebendig;
ALTER TABLE Drache DROP COLUMN Lebendig;
```

(g) Erstellen Sie eine weitere Tabelle mit dem Namen *Waffen*, welche genau diese auflistet. Eine Waffe ist genau einer Figur zugeordnet, hat einen eindeutigen Namen und eine Stärke zwischen 0 und 5. Wählen Sie sinnvolle Typen für die Attribute.

Lösungsvorschlag

```
CREATE TABLE Waffen (
  Name VARCHAR(20) PRIMARY KEY,
  Figur integer NOT NULL REFERENCES Figur(Id),
  Stärke integer NOT NULL CHECK(Stärke BETWEEN 0 AND 5)
);
```

```sql
-- Sollte funktionieren:
INSERT INTO Waffen VALUES
  ('Axt', 1, 5);

-- Sollte Fehler ausgeben:
-- INSERT INTO Waffen VALUES
--  ('Schleuder', 1, 6);
```

Thema Nr. 2

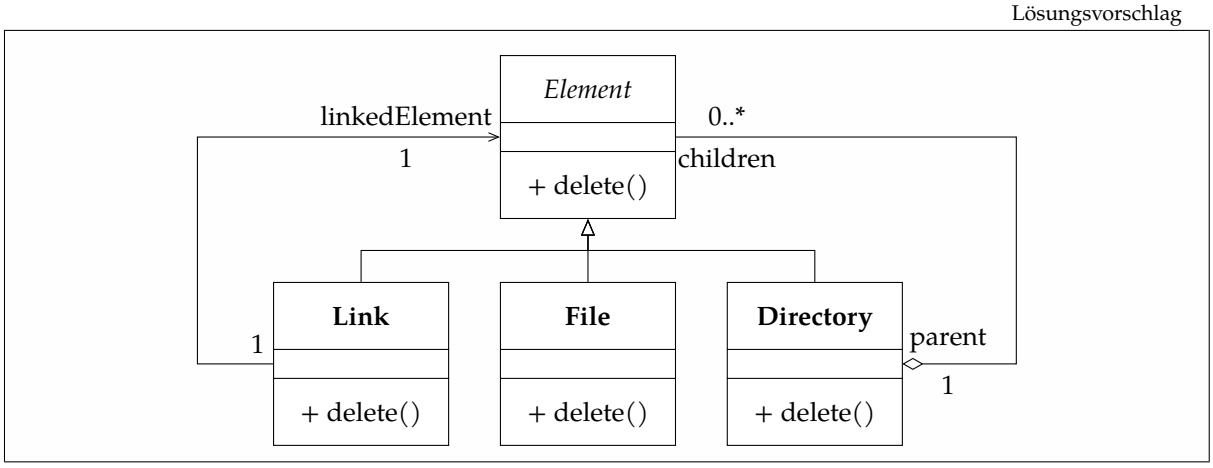
Teilaufgabe Nr. 1

Aufgabe 1 [Dateisystem: Implementierung durch Kompositum]

Hierarchische Dateisysteme bestehen aus den FileSystemElements Ordner, Dateien und Verweise. Ein Ordner kann seinerseits Ordner, Dateien und Verweise beinhalten; jedem Ordner ist bekannt, welche Elemente (children) er enthält. Mit Ausnahme des Root-Ordners auf der obersten Hierarchieebene ist jeder Ordner, jede Datei und jeder Verweis Element eines Elternordners. Jedem Element ist bekannt, was sein Elternordner ist (parent). Ein Verweis verweist auf einen Verweis, eine Datei oder einen Ordner (link). Wenn ein Ordner gelöscht wird, werden alle seine Bestandteile ggf. rekursiv ebenfalls gelöscht. Sie dürfen die Lösungen für Aufgabenteil b) und c) in einem gemeinsamen Code kombinieren.

(a) Modellieren Sie diesen Sachverhalt mit einem UML-Klassendiagramm. Benennen Sie die Rollen von Assoziationen und geben Sie alle Kardinalitäten an. Ihre Lösung soll mindestens eine sinnvolle Spezialisierungsbeziehung enthalten.

Lösungsvorschlag



(b) Implementieren Sie das Klassendiagramm als Java- oder C++-Programm. Jedes Element des Dateisystems soll mindestens über ein Attribut `name` verfügen. Übergeben Sie den Elternordner jedes Elements als Parameter an den Konstruktor; der Elternordner des Root-Ordners kann dabei als `null` implementiert werden. Dokumentieren Sie Ihren Code.

Lösungsvorschlag

a

```
public abstract class Element {

  protected String name;

  protected Element parent;

  protected Element(String name, Element parent) {
    this.name = name;
```

```
    this.parent = parent;
  }

  public String getName() {
    return name;
  }

  public abstract void delete();

  public abstract boolean isDirectory();

  public abstract void addChild(Element child);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Element.java`

```
import java.util.ArrayList;
import java.util.List;

public class Directory extends Element {
  private List<Element> children;

  public Directory(String name, Element parent) {
    super(name, parent);
    children = new ArrayList<Element>();
    if (parent != null)
      parent.addChild(this);
  }

  public void delete() {
    System.out.println("The directory "" + name +
↪   "" was deleted and it's children were also deleted.");
    for (int i = 0; i < children.size(); i++) {
      Element child = children.get(i);
      child.delete();
    }
  }

  public void addChild(Element child) {
    children.add(child);
  }

  public boolean isDirectory() {
    return true;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Directory.java`

```
public class File extends Element {

  public File(String name, Element parent) {
    super(name, parent);
    parent.addChild(this);
```

```
  }

  public void delete() {
    System.out.println("The File "" + name + "" was deleted.");
  }

  public boolean isDirectory() {
    return false;
  }

  /**
   * Eine Datei kann keine Kinder haben. Deshalb eine Methode mit leerem
   * Methodenrumpf.
   */
  public void addChild(Element child) {
  }

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/File.java

```
public class Link extends Element {

  private Element linkedElement;

  public Link(String name, Element parent, Element linkedElement) {
    super(name, parent);
    this.linkedElement = linkedElement;
    parent.addChild(this);
  }

  public void delete() {
    System.out.println("The Symbolic Link "" + name + "" was deleted.");
    linkedElement.delete();
    System.out.println("The linked element "" + name + "" was deleted too.");
  }

  public void addChild(Element child) {
    if (linkedElement.isDirectory())
      linkedElement.addChild(child);
  }

  public boolean isDirectory() {
    return linkedElement.isDirectory();
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Link.java

[a] TU Darmstadt: Dr. Michael Eichberg - Case Study Using the Composite and Proxy Design Patterns

(c) Ordnen Sie eine Methode `delete`, die Dateien, Ordner und Verweise rekursiv löscht, einer oder mehreren geeigneten Klassen zu und implementieren Sie sie. Zeigen Sie die Löschung jedes Elements durch eine Textausgabe von `name` an. `toString()` müssen

Sie dabei nicht implementieren. Gehen Sie davon aus, dass Verweis- und Ordnerstrukturen azyklisch sind und dass jedes Element des Dateisystems höchstens einmal existiert. Wenn ein Verweis gelöscht wird, wird sowohl der Verweis als auch das verwiesene Element bzw. transitiv die Kette der verwiesenen Elemente gelöscht. Bedenken Sie, dass die Löschung eines Elements immer auch Konsequenzen für den dieses Element beinhaltenden Ordner hat. Es gibt keinen Punktabzug, wenn Sie die Löschung des Root-Ordners nicht zulassen.

Lösungsvorschlag

Siehe Antwort zu Aufgabe b)

(d) Was kann im Fall von `delete` passieren, wenn die Linkstruktur zyklisch ist oder die Ordnerstruktur zyklisch ist? Kann es zu diesen Problemen auch dann führen, wenn weder die Linkstruktur zyklisch ist, noch die Ordnerstruktur zyklisch ist? Wie kann man im Programm das Problem lösen, falls man Zyklizitäten zulassen möchte?

Lösungsvorschlag

Falls die Link- oder Ordnerstruktur zyklisch ist, kann es aufgrund der Rekursion zu einer Endlosschleife kommen. Diese Problem tritt bei azyklischen Strukturen nicht auf, weil der rekursive Löschvorgang beim letzten Element abgebrochen wird (Abbruchbedingung).

Das Problem kann zum Beispiel durch ein neues Attribut `gelöscht` in der Klasse `Link` oder `Directory` gelöst werden. Dieses Attribut wird auf `true` gesetzt, bevor es in die Rekursion einsteigt. Rufen sich die Klassen wegen der Zyklizität selbst wieder auf, kommt es durch entsprechende IF-Bedingungen zum Abbruch.

Außerdem ist ein Zähler denkbar, der sich bei jeder Rekursion hochsetzt und ab einem gewissen Grenzwert zum Abbruch führt.

(e) Was ist ein Design Pattern? Nennen Sie drei Beispiele und erläutern Sie sie kurz. Welches Design Pattern bietet sich für die Behandlung von hierarchischen Teil-Ganzes-Beziehungen an, wie sie im Beispiel des Dateisystems vorliegen?

Lösungsvorschlag

Design Pattern sind wiederkehrende, geprüfte, bewährte Lösungsschablonen für typische Probleme.

Drei Beispiele

Einzelstück (Singleton) Stellt sicher, dass nur *genau eine Instanz einer Klasse* erzeugt wird.

Beobachter (Observer) Das Observer-Muster ermöglicht einem oder mehreren Objekten, automatisch auf die *Zustandsänderung* eines bestimmten Objekts *zu reagieren*, um den eigenen Zustand anzupassen.

Stellvertreter (Proxy) Ein Proxy stellt einen Platzhalter für eine andere Komponente (Objekt) dar und kontrolliert den Zugang zum echten Objekt.

Für hierarchischen Teil-Ganzes-Beziehungen eigent sich das Kompositum (Composite). Es ermöglicht die Gleichbehandlung von Einzelelementen und Elementgruppierungen in einer verschachtelten Struktur (z. B. Baum), sodass aus Sicht des Clients keine explizite Unterscheidung notwendig ist.

Aufgabe 3 [Softwarearchitektur und Agilität]

Die Komponentenarchitektur eines Softwaresystems beschreibt die unterschiedlichen Softwarebausteine, deren Schnittstellen und die Abhängigkeiten von Softwarekomponenten wie beispielsweise Klassen. Wir unterscheiden zwischen der Soll- und der Ist- Architektur eines Systems.

(a) Nennen und definieren Sie drei Qualitätsattribute von Software, die durch die Architektur beeinflusst werden und charakterisieren Sie jeweils eine „schlechte“ Architektur, die diese Qualitätsattribute negativ beeinflusst.

Lösungsvorschlag

Skalierbarkeit Definition: Ob die Software auch für große Zugriffslasten konzipiert wurde. Charakterisierung einer schlechten Architektur: Eine Anwendung läuft nur auf einem Server.

Modifizierbarkeit Definition: Ob die Software leicht verändert, erweitert werden kann. Charakterisierung einer schlechten Architektur: Monolithische Architektur, dass kein Laden von Modulen zulässt.

Verfügbarkeit Definition: Ob die Software ausfallsicher ist, im Laufenden Betrieb gewartet werden kann. Charakterisierung einer schlechten Architektur: Ein-Server-Architektur, die mehrmal neugestartet werden muss, um ein Update einzuspielen.

(b) Erläutern Sie, was Information Hiding ist. Wie hängt Information Hiding mit Softwarearchitektur zusammen”? Wie wird Information Hiding auf Klassenebene in Java implementiert? Gibt es Situationen, in denen Information Hiding auch negative Effekte haben kann?

Lösungsvorschlag

Das Prinzip der Trennung von Zuständigkeiten (engl. separation of concerns) sorgt dafür, dass jede Komponente einer Architektur nur für eine einzige Aufgabe zuständig ist. Das Innenleben von Komponenten wird durch Schnittstellen verkapselt, was auf das Prinzip des Verbergens von Informationen (engl. information hiding) zurückgeht.

Java: Durch die Sichbarkeits-Schlüsselwörter: private, protected

Zusätzlicher Overhead durch Schnittestellen API zwischen den Modulen.

(c) Erklären Sie, was Refactoring ist.

Verbesserungen des Code durch bessere Lesbarkeit, Wartbarkeit, Performanz, Sicherheit. Keine neuen Funktionen werden programmiert.

(d) Skizzieren Sie die Kernideen von Scrum inkl. der wesentlichen Prozessschritte, Artefakte und Rollen. Beschreiben Sie dann die Rolle von Ist- und Soll-Architektur in agilen Entwicklungskontexten wie Scrum.

Teilaufgabe Nr. 2

Aufgabe 1 [Sportverein]

Erstellen Sie ein möglichst einfaches ER-Schema, das alle gegebenen Informationen enthält. Attribute von Entitäten und Beziehungen sind anzugeben, Schlüsselattribute durch Unterstreichen zu kennzeichnen. Verwenden Sie für die Angabe der Kardinalitäten von Beziehungen die Min-MaxNotation. Führen Sie Surrogatschlüssel nur dann ein, wenn es nötig ist und modellieren Sie nur die im Text vorkommenden Elemente.

Ein örtlicher Sportverein möchte seine Vereinsangelegenheiten mittels einer Datenbank verwalten. Der Verein besteht aus verschiedenen Abteilungen, welche eine eindeutige Nummer und einen aussagekräftigen Namen besitzen. Für jede Abteilung soll zudem automatisch die Anzahl der Mitglieder gespeichert werden, wobei ein Mitglied zu mehreren Abteilungen gehören kann. Die Mitglieder des Vereins können keine, eine oder mehrere Rollen (auch Ämter genannt) einnehmen. So gibt es die Ämter: 1. Vorstand, 2. Vorstand, Kassier, Jugendleiter, Trainer sowie einen Abteilungsleiter für jede Abteilung. Es ist dabei auch möglich, dass ein Abteilungsleiter mehrere Abteilungen leitet oder ein Mitglied mehrere Aufgaben übernimmt, mit der Einschränkung, dass die Vorstandsposten und Kassier nicht von der gleichen Person ausgeübt werden dürfen. Zu jedem Trainer wird eine Liste von Lizenzen gespeichert. Jeder Trainer ist zudem in mindestens einer Abteilung eine bestimmte Anzahl von Stunden tätig. Zu allen Mitgliedern werden Mitgliedsnummer, Name (bestehend aus Vor- und Nachname), Geburtsdatum, E-Mail, Eintrittsdatum, Adresse (bestehend aus PLZ, Ort, Straße, Hausnummer), IBAN und die Vereinszugehörigkeit in Jahren gespeichert.

Im Verein fallen Finanztransaktionen an. Zu jeder Transaktion wird ein Zeitstempel, der Betrag und eine eindeutige Transaktionsnummer gespeichert. Die Mitglieder leisten Zahlungen an den Verein. Umgekehrt erstattet der Verein auch bestimmte Kosten. Im Verein existieren drei verschiedene Mitgliedsbeiträge. So gibt es einen Kinder-und-Jugendlichen-Tarif, einen Erwachsenentarif und einen Familientarif.

Mitgliedern entstehen des Öfteren Fahrtkosten. Für jede Fahrtkostenabrechnung werden das Datum, die gefahrenen Kilometer und Start und Ziel, der Zweck sowie das Mitglied gespeichert, welches den Antrag gestellt hat. Zu jeder Fahrtkostenabrechnung existiert genau eine Erstattung.

Durch die Teilnahme an verschiedenen Wettbewerben besteht die Notwendigkeit die von einem Team (also mehreren Mitgliedern zusammen) oder Mitgliedern erzielten sportlichen Erfolge, d.h. Platzierungen, zu verwalten. Jeder Wettkampf besitzt eine eindeutige ID, ein Datum und eine Kurzbeschreibung.

Das Vereinsleben besteht aus zahlreichen Terminen, die durch Datum und Uhrzeit innerhalb einer

Abteilung eindeutig identifiziert werden können. Zu jedem Termin wird zusätzlich eine Kurzbeschreibung gespeichert.

Aufgabe 5 [Relationen „Professor“ und „Vorlesung“]

Gegeben seien die beiden Relationen Professor und Vorlesung mit folgendem Umfang: Relation Professor: 5 Spalten, 164 Datensätze Relation Vorlesung: 10 Spalten, 333 Datensätze.

Optimieren Sie auf geeignete Weise folgende SQL-Anweisung möglichst gut und berechnen Sie wie stark sich die Datenmenge durch jede Optimierung reduziert (nehmen Sie für Ihre Berechnung an, dass Herr Mustermann genau zwei Vorlesungen hält). Geben Sie jeweils den Operatorbaum vor und nach Ihren jeweiligen Optimierungen an.

SELECT Titel FROM Professor, Vorlesung WHERE Name = Mustermann' AND PersNr = gelesenVon;

Aufgabe 6 [Vermischte Datenbank-Fragen]

Begründen oder erläutern Sie Ihre Antworten.

(a) Erklären Sie kurz den Unterschied zwischen einem Natural-Join und einem Equi-Join.

Lösungsvorschlag

Ein Natural Join ist eine Kombination von zwei Tabellen, in denen Spalten gleichen Namens existieren. Die Werte in diesen Spalten werden sodann auf Übereinstimmungen geprüft (analog Equi-Join). Einige Datenbanksysteme erkennen das Schlüsselwort NATURAL und eliminieren entsprechend automatisch doppelte Spalten.

Während beim Kreuzprodukt keinerlei Anforderungen an die Kombination der Datensätze gestellt werden, führt der Equi-Join eine solche ein: Die Gleichheit von zwei Spalten.

a

[a] wiki.selfhtml.org

(b) Erläutern Sie kurz was man unter einem Theta-Join versteht.

Lösungsvorschlag

Ein Theta-Join ist eine Verbindung von Relationen bezüglich beliebiger Attribute und mit einem Selektionsprädikat. [a]

[a] `https://www.datenbank-grundlagen.de/theta-join.html`

(c) Was versteht man unter Unionkompatibilität? Nennen Sie drei SQL-Operatoren welche Unionkompatibilität voraussetzen.

Lösungsvorschlag

Bestimmte Operationen der relationalen Algebra wie Vereinigung, Schnitt und Differenz verlangen Unionkompatibilität. Unionkompatibilität ist eine Eigenschaft des Schemas einer Relation. Zwei Relationen R und S sind genau dann union-kompatibel, wenn folgende Bedingungen erfüllt sind:

(i) Die Relationen R und S besitzen dieselbe Stelligkeit n, ðsie haben die selbe Anzahl von Spalten.

(ii) Für alle Spalten der Relationen gilt, dass die Domäne der i-ten Spalte der Relation R mit dem Typ der i-ten Spalte der Relation S übereinstimmt $(0 < i < n)$.

Die Namen der Attribute spielen dabei keine Rolle. [a]

SQL-Operatoren mit Unionkompatibilität

- UNION
- INTERSECT
- EXCEPT

[a] `https://studylibde.com/doc/1441274/übungstool-für-relationale-algebra`

(d) Erläutern Sie Backward und Forward Recovery und grenzen Sie diese voneinander ab.

(e) Erklären Sie das Zwei-Phasen-Freigabe-Protokoll.

(f) Erläutern Sie Partial Undo / Redo und Global Undo / Redo und deren Bedeutung für die Umsetzung des ACID-Prinzips. Geben Sie zu jeder dieser Konzepte an, ob System-, Programm- oder Gerätefehler damit korrigiert werden können.

(g) Erklären Sie das WAL-Prinzip (Write ahead logging)!

Lösungsvorschlag

Das sogenannte write ahead logging (WAL) ist ein Verfahren der Datenbanktechnologie, das zur Gewährleistung der Atomarität und Dauerhaftigkeit von Transaktionen beiträgt. Es besagt, dass Modifikationen vor dem eigentlichen Schreiben (dem Einbringen in die Datenbank) protokolliert werden müssen.

Durch das WAL-Prinzip wird ein sogenanntes „update-in-place“ ermöglicht, ðdie alte Version eines Datensatzes wird durch die neue Version an gleicher Stelle überschrieben. Das hat vor allem den Vorteil, dass Indexstrukturen bei Änderungsoperationen nicht mit aktualisiert werden müssen, weil die geänderten Datensätze immer noch an der gleichen Stelle zu finden sind. Die vorherige Protokollierung einer Änderung ist erforderlich, um im Fehlerfall die Wiederholbarkeit der Änderung sicherstellen zu können.

(h) Erklären Sie den Begriff „Datenbankindex“ und nennen Sie zwei häufige Arten.

Lösungsvorschlag

Ein Datenbankindex ist eine von der Datenstruktur getrennte Indexstruktur in einer Datenbank, die die Suche und das Sortieren nach bestimmten Feldern beschleunigt.

Gruppierte Indizes (Clustered Index)

Bei der Verwendung eines gruppierten Index werden die Datensätze entsprechend der Sortierreihenfolge ihres Index-Schlüssels gespeichert. Wird für eine Tabelle beispielsweise eine Primärschlüssel-Spalte „NR“ angelegt, so stellt diese den Index-Schlüssel dar. Pro Tabelle kann nur ein gruppierter Index erstellt werden. Dieser kann jedoch aus mehreren Spalten zusammengesetzt sein.

Nicht-gruppierte Indizes (Nonclustered Index)

Besitzt eine Tabelle einen gruppierten Index, so können weitere nicht-gruppierte Indizes angelegt werden. Dabei zeigen die Einträge des Index auf den Speicherbereich des gesamten Datensatzes. Die Verwendung eines nicht-gruppierten Index bietet sich an, wenn regelmäßig nach bestimmten Werten in einer Spalte gesucht wird z.B. dem Namen eines Kunden. [a]

[a] https://www.datenbanken-verstehen.de/datenmodellierung/datenbank-index

Aufgabe 7 [Formel-1-Rennen]

Gegeben sind folgende Relationen aus einem Verwaltungssystem für die jährlichen Formel-1-Rennen:

```
Strecke(Strecken_ID, Streckenname, Land, Länge)

Fahrer(Fahrer_ID, Fahrername, Nation, Rennstall)

Rennen(Strecken_ID[Strecke], Jahr, Wetter)

Rennteilnahme(Fahrer_ID[Fahrer], Strecken_ID[Rennen], Jahr[Rennen], Rundenbestzeit, Gesamtzeit,

disqualifiziert)
FK (Strecken_ID, Jahr) referenziert Rennen (Strecken_ID, Jahr)
```

```
CREATE TABLE Fahrer (
  Fahrer_ID INTEGER PRIMARY KEY,
  Fahrername VARCHAR(100) NOT NULL,
  Nation VARCHAR(100) NOT NULL,
  Rennstall VARCHAR(100) NOT NULL
);

CREATE TABLE Strecke (
  Strecken_ID INTEGER PRIMARY KEY,
  Streckenname VARCHAR(100) NOT NULL,
  Land VARCHAR(100) NOT NULL,
  Länge NUMERIC(5,3) NOT NULL
);

CREATE TABLE Rennen (
  Strecken_ID INTEGER REFERENCES Strecke(Strecken_ID),
  Jahr INTEGER NOT NULL,
  Wetter VARCHAR(10) NOT NULL,
  PRIMARY KEY (Strecken_ID, Jahr)
```

```
);

CREATE TABLE Rennteilnahme (
  Fahrer_ID INTEGER REFERENCES Fahrer(Fahrer_ID),
  Strecken_ID INTEGER REFERENCES Strecke(Strecken_ID),
  Jahr INTEGER NOT NULL,
  Rundenbestzeit NUMERIC(5,3) NOT NULL,
  Gesamtzeit NUMERIC(5,3) NOT NULL,
  disqualifiziert BOOLEAN NOT NULL,
  PRIMARY KEY (Fahrer_ID, Strecken_ID, Jahr)
);

INSERT INTO Fahrer VALUES
  (1, 'Kimi Räikkönen', 'Finnland', 'Alfa Romeo'),
  (2, 'Rubens Barrichello', 'Brasilien', 'Brawn'),
  (3, 'Fernando Alonso', 'Spanien', 'Ferrari'),
  (4, 'Michael Schumacher', 'Deutschland', 'Ferrari'),
  (5, 'Jenson Button', 'Vereinigtes Königreich Großbritannien', 'McLaren'),
  (6, 'Felipe Massa', 'Brasilien', 'Ferrari'),
  (7, 'Lewis Hamilton', 'Vereinigtes Königreich Großbritannien', 'Williams'),
  (8, 'Riccardo Patrese', 'Italien', 'Williams'),
  (9, 'Sebastian Vettel', 'Deutschland', 'Ferrari'),
  (10, 'Jarno Trulli', 'Italien', 'Toyota');

INSERT INTO Strecke VALUES
  (1, 'Autodromo Nazionale Monza', 'Italien', 5.793),
  (2, 'Circuit de Monaco', 'Monaco', 3.340),
  (3, 'Silverstone Circuit', 'Vereinigtes Königreich', 5.891),
  (4, 'Circuit de Spa-Francorchamps', 'Belgien', 7.004),
  (5, 'Circuit Gilles-Villeneuve', 'Kanada', 4.361),
  (6, 'Nürburgring', 'Deutschland', 5.148),
  (7, 'Hockenheimring', 'Deutschland', 4.574),
  (8, 'Interlagos', 'Brasilien', 4.309),
  (9, 'Hungaroring', 'Ungarn', 4.381),
  (10, 'Red Bull Ring', 'Österreich', 5.942),
  (11, 'Abu Dhabi', 'Abu Dhabi', 5.554);

INSERT INTO Rennen VALUES
  (11, 2011, 'sonnig'),
  (10, 2006, 'sonnig'),
  (9, 2007, 'regnerisch'),
  (8, 2008, 'regnerisch'),
  (7, 2009, 'sonnig'),
  (6, 2010, 'regnerisch'),
  (5, 2011, 'sonnig'),
  (4, 2012, 'sonnig'),
  (3, 2013, 'sonnig'),
  (2, 2014, 'regnerisch'),
  (1, 2015, 'regnerisch');

INSERT INTO Rennteilnahme VALUES
  (1, 11, 2011, 2.001, 90.001, FALSE),
  (2, 11, 2011, 2.002, 90.002, FALSE),
  (3, 11, 2011, 2.003, 90.003, FALSE),
  (4, 11, 2011, 2.004, 89.999, FALSE),
```

```
(5, 11, 2011, 2.005, 90.005, FALSE),
(6, 11, 2011, 2.005, 99.009, FALSE),
(4, 10, 2006, 2.782, 90.005, TRUE),
(3, 10, 2006, 2.298, 90.005, TRUE),
(3, 9, 2009, 2.253, 90.005, TRUE),
(2, 10, 2006, 2.005, 90.005, TRUE),
(2, 9, 2009, 3.298, 90.342, TRUE),
(2, 8, 2008, 4.782, 78.005, TRUE);
```

Der Einfachheit halber wird angenommen, dass Fahrer den Rennstall nicht wechseln können. Das Attribut „disqualifiziert“ kann die Ausprägungen „ja“ und „nein“ haben. Formulieren Sie folgende Abfragen in SQL. Vermeiden Sie nach Möglichkeit übermäßige Nutzung von Joins und Views.

(a) Geben Sie für jeden Fahrer seine ID sowie die Anzahl seiner Disqualifikationen in den Jahren 2005 bis 2017 aus. Ordnen Sie die Ausgabe absteigend nach der Anzahl der Disqualifikationen.

Lösungsvorschlag

```
SELECT Fahrer_ID, COUNT(disqualifiziert) as anzahl_disqualifikationen FROM
↪  Rennteilnahme
WHERE disqualifiziert = TRUE
GROUP BY Fahrer_ID, disqualifiziert
ORDER BY anzahl_disqualifikationen DESC;
```

(b) Gesucht sind alle Länder, aus denen noch nie ein Fahrer disqualifiziert wurde.

Lösungsvorschlag

```
SELECT Nation FROM Fahrer GROUP BY Nation
EXCEPT
SELECT f.Nation FROM Fahrer f, Rennteilnahme t
WHERE f.Fahrer_ID = t.Fahrer_ID AND t.disqualifiziert = TRUE
GROUP BY f.Nation;
```

(c) Gesucht sind die ersten fünf Plätze des Rennens von 2011 in „Abu Dhabi“ (Streckenname). Die Ausgabe soll nach der Platzierung absteigend erfolgen. Geben Sie Fahrer_ID, Fahrername, Nation und Rennstall mit aus.

Lösungsvorschlag

Mit LIMIT

```
SELECT f.Fahrer_ID, f.Fahrername, f.Nation, f.Rennstall
FROM Fahrer f, Rennteilnahme t, Strecke s
WHERE
  f.Fahrer_ID = t.Fahrer_ID AND
  s.Strecken_ID = t.Strecken_ID AND
  s.Streckenname = 'Abu Dhabi' AND
  t.Jahr = 2011
ORDER BY t.Gesamtzeit ASC LIMIT 5;
```

Als Top-N-Query:

```sql
CREATE VIEW Rennen_Abu_Dhabi AS
  SELECT f.Fahrer_ID, f.Fahrername, f.Nation, f.Rennstall, t.Gesamtzeit
  FROM Fahrer f, Rennteilnahme t, Strecke s
  WHERE
    f.Fahrer_ID = t.Fahrer_ID AND
    s.Strecken_ID = t.Strecken_ID AND
    s.Streckenname = 'Abu Dhabi' AND
    t.Jahr = 2011
  ORDER BY t.Gesamtzeit ASC;

SELECT a.Fahrer_ID, a.Fahrername, a.Nation, a.Rennstall
FROM Rennen_Abu_Dhabi a, Rennen_Abu_Dhabi b
WHERE
  a.Gesamtzeit >= b.Gesamtzeit
GROUP BY a.Fahrer_ID, a.Fahrername, a.Nation, a.Rennstall, a.Gesamtzeit
HAVING COUNT(*) <= 5
ORDER BY a.Gesamtzeit;
```

(d) Führen Sie eine neue Spalte `Gehalt` in die Tabelle `Fahrer` ein. Da sich die Prämien für die Fahrer nach einem Rennstallwechsel ändern, soll ein Trigger geschrieben werden, mit dem das Gehalt des betreffenden Fahrers um 10% angehoben wird.

Lösungsvorschlag

Lösung für PostgreSQL:

```sql
ALTER TABLE Fahrer ADD Gehalt numeric(12,2);

UPDATE Fahrer SET Gehalt = 1000000 WHERE Fahrer_ID = 1;
UPDATE Fahrer SET Gehalt = 2000000 WHERE Fahrer_ID = 2;
UPDATE Fahrer SET Gehalt = 3000000 WHERE Fahrer_ID = 3;
UPDATE Fahrer SET Gehalt = 4000000 WHERE Fahrer_ID = 4;
UPDATE Fahrer SET Gehalt = 5000000 WHERE Fahrer_ID = 5;
UPDATE Fahrer SET Gehalt = 6000000 WHERE Fahrer_ID = 6;
UPDATE Fahrer SET Gehalt = 7000000 WHERE Fahrer_ID = 7;
UPDATE Fahrer SET Gehalt = 8000000 WHERE Fahrer_ID = 8;
UPDATE Fahrer SET Gehalt = 9000000 WHERE Fahrer_ID = 9;
UPDATE Fahrer SET Gehalt = 10000000 WHERE Fahrer_ID = 10;

CREATE FUNCTION trigger_function()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
AS $$
  BEGIN
    UPDATE Fahrer
      SET gehalt = gehalt * 1.1
      WHERE Fahrer_ID = NEW.Fahrer_ID;
    RETURN NEW;
  END;
$$;

CREATE TRIGGER mehr_gehalt
  AFTER UPDATE OF Rennstall ON Fahrer
  FOR EACH ROW EXECUTE PROCEDURE trigger_function();
```

```sql
-- Test:
SELECT * FROM FAHRER WHERE Fahrer_ID = 1;
UPDATE Fahrer SET Rennstall = 'Red Bull' WHERE Fahrer_ID = 1;
SELECT * FROM FAHRER WHERE Fahrer_ID = 1;
```