Simulation einer mündlichen Abiturprüfung – Informatik

Thema: OOP, Algorithmen, Datenbanken, Formale Sprachen

May 31, 2025

Aufgabe 1: Analyse eines Algorithmus

Ein unbekannter Algorithmus zur Berechnung einer speziellen mathematischen Eigenschaft von Zahlen wird bereitgestellt. Der Name der Klasse lautet `Algorithmus`.

Gegebener Java-Code

```
public class Algorithmus {
 public static int berechne(int n) {
  if (n == 0) {
   return 1;
  }
  return n * berechne(n - 1);
 }

 public static void main(String[] args) {
  int n = 5;
  System.out.println("Ergebnis: " + berechne(n));
 }
}
```

Teilfragen zur Analyse des Codes

1. Beschreiben Sie die Funktionsweise des Algorithmus. Welche mathematische Funktion wird berechnet?
2. Wie oft wird die Funktion `berechne` für `n=5` aufgerufen?
3. Welche alternative Implementierung könnte vorteilhafter sein?
4. Bestimmen Sie die Zeitkomplexität des Algorithmus.
5. Wie kann die Laufzeit verbessert werden?

Aufgabe 2: Normalisierung einer Musikdatenbank

Ein Musikstreaming-Dienst speichert Informationen zu Liedern in einer relationalen Datenbank. Die ursprüngliche Tabellenstruktur ist wie folgt:

Table 1: Ursprüngliche nicht normalisierte Tabelle

SongID	Titel	Künstler	Album	Genre	Dauer (s)
1	Imagine	John Lennon	Imagine	Rock	183
2	Bohemian Rhapsody	Queen	A Night at the Opera	Rock	354
3	Shape of You	Ed Sheeran	Divide	Pop	233
4	Rolling in the Deep	Adele	21	Soul	228
5	Someone Like You	Adele	21	Soul	285

Teilaufgaben zur Normalisierung

1. Identifizieren Sie Redundanzen in der Tabelle.
2. Zerlegen Sie die Tabelle in mehrere normalisierte Tabellen bis zur 3. Normalform.
3. Welche Vorteile bringt die Normalisierung in diesem Fall?

Aufgabe 3: Analyse einer formalen Grammatik

Eine kontextfreie Grammatik G erzeugt gültige Datumsangaben im Format TT/MM/JJJJ:

- Terminale: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, /, \}$
- Nichtterminale: $\{S, T, M, J\}$
- Startsymbol: S
- Produktionsregeln:

$$
\begin{aligned}
S &\to T/M/J \\
T &\to 0D \mid 1D \mid 2D \mid 3D \\
D &\to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
M &\to 0N \mid 1N \\
N &\to 0 \mid 1 \mid 2 \\
J &\to DDDD \\
DDDD &\to DDDD \mid DD
\end{aligned}
$$

Teilfragen zur Grammatik

1. Leiten Sie das Datum `25/12/2024` mit der Grammatik ab.
2. Welche Einschränkungen hat diese Grammatik, um nur gültige Kalendertage zu erlauben?
3. Wie könnte die Grammatik angepasst werden, um Monate mit 30 oder 31 Tagen korrekt zu unterscheiden?

Lösung zu Aufgabe 2: Datenmodellierung und Normalisierung – Musikdatenbank

1. **Redundanzen und Anomalien:**

   Redundanzen:

   - **Künstlernamen**, **Albumtitel** und **Genres** erscheinen mehrfach in der Tabelle, wenn z. B. ein Künstler mehrere Songs hat oder ein Album mehrere Titel enthält.
   - Dadurch werden dieselben Informationen wiederholt gespeichert – z. B. derselbe Künstlername für jede seiner Songs.

   Anomalien:

   - **Einfügeanomalie:** Es ist nicht möglich, einen neuen Künstler oder ein neues Genre zu erfassen, ohne gleichzeitig mindestens einen Song einzutragen.
   - **Änderungsanomalie:** Eine Änderung am Namen eines Künstlers oder Genres müsste an mehreren Stellen durchgeführt werden, was zu Inkonsistenzen führen kann.
   - **Löschanomalie:** Wird ein Song gelöscht und war es der einzige eines Künstlers oder Genres, gehen auch die Informationen über diesen Künstler oder das Genre verloren.

2. **Schrittweise Normalisierung bis zur 3. Normalform (3NF):**

   1. Normalform (1NF):

   - Alle Attributwerte müssen **atomar** sein – d. h. jeder Wert ist unteilbar.
   - Die ursprüngliche Tabelle erfüllt dieses Kriterium bereits: Es gibt keine Listen, Mehrfachwerte oder zusammengesetzten Einträge.
   - **Ergebnis:** Tabelle ist formal in 1NF.

   2. Normalform (2NF):

   - Voraussetzung: Tabelle ist in 1NF.
   - Zusätzlich: Alle Nicht-Schlüsselattribute müssen **voll funktional abhängig** vom gesamten Primärschlüssel sein.
   - Da der Primärschlüssel vermutlich `SongID` ist (also kein zusammengesetzter Schlüssel), ist 2NF erfüllt.
   - Dennoch erkennt man **funktionale Abhängigkeiten**, die für eine bessere Struktur ausgelagert werden sollten:
     - Jeder Song gehört zu **genau einem Album**, daher ist der Albumname abhängig von einer `AlbumID`.

  - Jeder Song hat genau **einen Künstler**, daher ist der Künstlername abhängig von `KuenstlerID`.
  - Gleiches gilt für Genre.

Entstehende Tabellenstruktur (nach Zerlegung in 2NF):

- `Song(SongID, Titel, Dauer, AlbumID (FK), KuenstlerID (FK), GenreID (FK))`
- `Album(AlbumID, Albumname)`
- `Kuenstler(KuenstlerID, Kuenstlername)`
- `Genre(GenreID, Bezeichnung)`

3. Normalform (3NF):

- Voraussetzung: Tabelle ist in 2NF.
- Zusätzlich: Es dürfen keine **transitiven Abhängigkeiten** zwischen Nicht-Schlüsselattributen bestehen.
- Da in den Tabellen jedes Nichtschlüsselattribut direkt vom Primärschlüssel abhängt, ist die 3NF erreicht.

Endgültige Tabellenstruktur mit Schlüsseln:

- `Song(SongID, Titel, Dauer, AlbumID (FK), KuenstlerID (FK), GenreID (FK))`
- `Album(AlbumID, Albumname)`
- `Kuenstler(KuenstlerID, Kuenstlername)`
- `Genre(GenreID, Bezeichnung)`

3. **Vorteile der Normalisierung:**

- **Vermeidung von Redundanzen:** Künstler-, Album- und Genreinformationen werden nur einmal gespeichert.
- **Konsistente Datenpflege:** Änderungen (z. B. Korrektur eines Künstlernamens) müssen nur an einer Stelle erfolgen.
- **Keine Anomalien:** Einfüge-, Änderungs- und Löschanomalien werden vermieden.
- **Skalierbarkeit:** Die Datenbank kann effizient erweitert werden – z. B. mit neuen Genres, Künstlern oder Songs.
- **Bessere Wartbarkeit:** Die klare Trennung der Entitäten vereinfacht spätere Abfragen, Reports oder App-Anbindungen.