Abschlussprüfung Sommer 2022
Lösungshinweise

Fachinformatiker/Fachinformatikerin Anwendungsentwicklung (AO 2020) 1201

2 Entwicklung und Umsetzung von Algorithmen

Teil 2 der Abschlussprüfung

Allgemeine Korrekturhinweise

Die Lösungs- und Bewertungshinweise zu den einzelnen Handlungsschritten sind als Korrekturhilfen zu verstehen und erheben nicht in jedem Fall Anspruch auf Vollständigkeit und Ausschließlichkeit. Neben hier beispielhaft angeführten Lösungsmöglichkeiten sind auch andere sach- und fachgerechte Lösungsalternativen bzw. Darstellungsformen mit der vorgesehenen Punktzahl zu bewerten. Der Bewertungsspielraum des Korrektors (z. B. hinsichtlich der Berücksichtigung regionaler oder branchenspezifischer Gegebenheiten) bleibt unberührt.

Zu beachten ist die unterschiedliche Dimension der Aufgabenstellung (nennen – erklären – beschreiben – erläutern usw.).

Für die Bewertung gilt folgender Punkte-Noten-Schlüssel:

Note 1	=		100 – 92 Punkte	Note 2	=	unter	92 – 81 Punkte
Note 3	=	unter	81 – 67 Punkte	Note 4	=	unter	67 – 50 Punkte
Note 5	=	unter	50 – 30 Punkte	Note 6	=	unter	30 – 0 Punkte

1. Aufgabe (30 Punkte)

Lösungsbeispiel Pseudocode

```
Jahresstatistik statistik(verbrauch: int [][], limit: int) {
    int minMonat = Integer.MAX_VALUE;
    int maxMonat = Integer.MIN_VALUE;
    List<Monatsverbrauch> greaterLimit = new List<Monatsverbrauch>();

    for (int vn = 0; vn < verbrauch.length; vn++) {
        for (int m = 1; m <= 12; m++) {
                int v = verbrauch[vn][m+1] - verbrauch[vn][m];
                if (v > limit) {
                        greaterLimit.add(new Monatsverbrauch( verbrauch[vn][0], m , v));
                }
                if( v < minMonat) minMonat =v;
                else if (v>maxMonat) maxMonat =v;
        }
     }
     return new Jahresstatistik( minMonat, maxMonat, greaterLimit);
}
```

Alternative korrekte Lösungen sind ebenfalls als richtig zu bewerten.

2. Aufgabe (20 Punkte)

aa) 3 Punkte

Bei einem Unit-Test werden einzelne Teile oder Einheiten einer Anwendung, die sogenannten Units, auf ihre Funktionalität hin überprüft.

ab) 4 Punkte

F.I.R.S.T-Prinzipien für Unit-Tests

- **Fast**: Die Testausführung soll möglichst schnell sein, damit sehr oft getestet werden kann.
- **Independent**: Unit-Tests sollen unabhängig voneinander sein, damit sie in beliebiger Reihenfolge, parallel oder einzeln ausführt werden können.
- **Repeatable**: Führt man einen Unit-Test mehrfach aus, muss er immer das gleiche Ergebnis liefern.
- **Self-Validating**: Ein Unit-Test soll entweder fehlschlagen oder gut gehen. Diese Entscheidung muss der Test treffen. Es dürfen keine weiteren manuellen Prüfungen nötig sein.
- **Timely**: Man soll Unit-Tests vor der Entwicklung des Produktivcodes schreiben.
- u. a.

ba) 5 Punkte

Jeweils 1 Punkt pro richtigem Testergebnis.

Test	Ergebnis des Tests
1.	OK
2.	Fehler
3.	Fehler
4.	OK
5.	OK

bb) 4 Punkte

```
int berechneDifferenz (int zaehlerstandAlt, int zaehlerstandNeu) {
    int differenz = -1;
    if (zaehlerstandAlt >= 0 && zaehlerstandNeu >= 0) {
        if (zaehlerstandAlt <= zaehlerstandNeu) {
            differenz = zaehlerstandNeu - zaehlerstandAlt;
        }
    }
    return differenz;
}
```

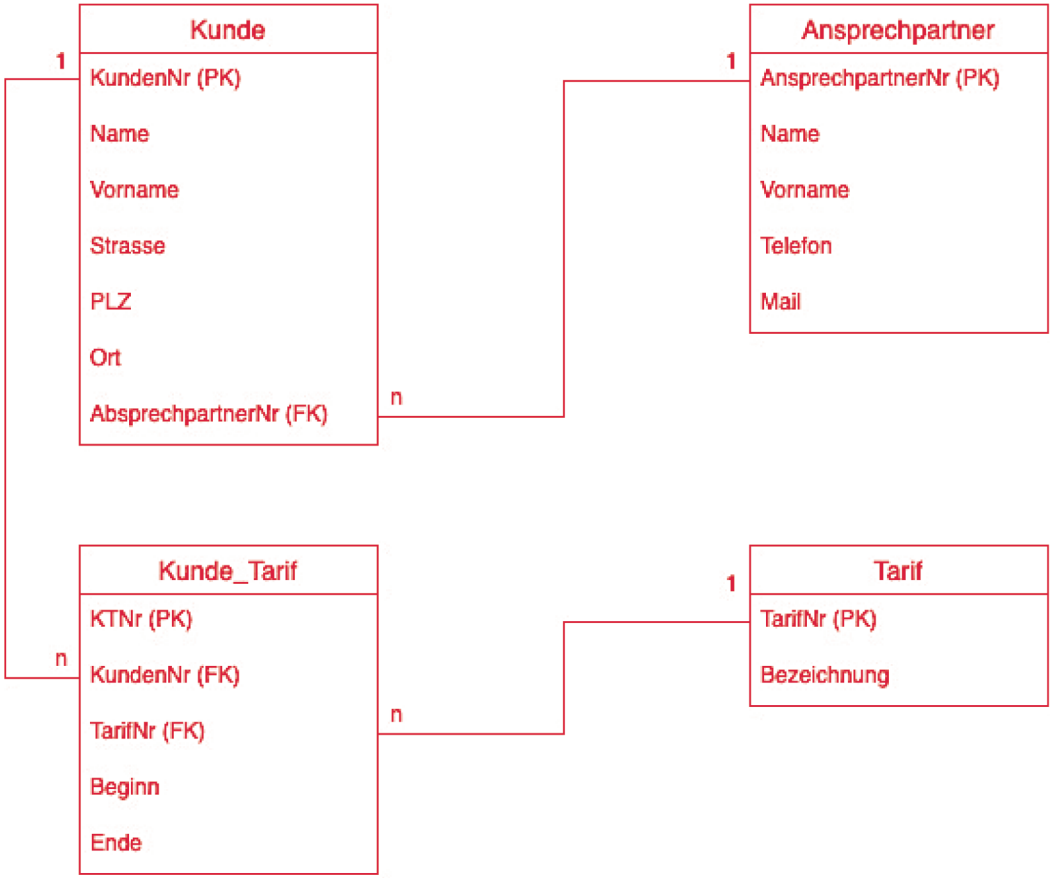
Auch andere Lösungen sind möglich.

bc) 4 Punkte
Pro vollständig richtigem Beispiel 2 Punkte
Beispiele:

Nr.	zaehlerstandAlt	zaehlerstandNeu	differenz
1	-200	230	-1
2	-60	0	-1
3	-96	-35	-1

3. Aufgabe (20 Punkte)

Vorschlag für die Vergabe der Punkte:
- je 1 Punkt pro Tabelle: 4 Punkte
- je 1 Punkt für jede richtige Beziehung: 3 Punkte
- je 1 Punkt für jeden richtigen Primär- und Fremdschlüssel: 7 Punkte
- je 0,5 Punkte für jedes richtige Attribut: 6 Punkte



Alternative Lösungen sind möglich.

4. Aufgabe (30 Punkte)

aa) 2 Punkte

Es handelt sich dabei übersetzt um eine gespeicherte Prozedur. Sie ist eine Anweisung in einem Datenbankmanagementsystem, mit der ganze Abläufe von Anweisungen vom Datenbank-Client aufgerufen werden können. Stellt einen eigenständigen Befehl dar, der eine Abfolge weiterer gespeicherter Befehle ausführen kann.

ab) 2 Punkte

Ein Trigger ist eine spezielle Art einer Stored Procedure, die automatisch ausgeführt wird, wenn ein Ereignis auf dem Datenbankserver auftritt.

ac) 2 Punkte

Indizes werden vergeben, um die Suche nach Datensätzen zu erleichtern und zu beschleunigen. Die Indizierung von Spalten bietet vor allem bei sehr großen Datenmengen enorme Geschwindigkeitsvorteile.

ba) 3 Punkte

```
CREATE INDEX Idx_Datum ON Zaehlerstand (ZSt_Datum)
```

bb) 5 Punkte

```
DELETE  FROM Haushalte
WHERE   ( SELECT Count(Z_ID) FROM Zaehler WHERE Z_HHID = HH_ID ) = 0
```

Alternativ-Vorschlag:

```
DELETE  FROM Haushalte
WHERE  HH_ID NOT IN ( SELECT Z_HHID FROM Zaehler)
```

bc) 6 Punkte

```
SELECT HH.HH_Nachname AS Nachname, Count(Z.Z_ID) AS AnzZaehler
FROM  Haushalte AS HH  LEFT JOIN Zaehler AS Z ON HH.HH_ID = Z.Z_HHID
GROUP BY HH.HH_Nachname
ORDER BY Count(Z.Z_ID) DESC;
```

bd) 10 Punkte

```
SELECT HH.HH_Nachname AS Nachname, Z.Z_Nummer AS Zaehlernummer,
 ( ( SELECT ZSt.ZSt_Stand FROM ZaehlerStand AS ZSt
     WHERE Z.Z_ID = ZSt.ZSt_ZID AND YEAR(ZSt.ZSt_Datum) = 2022)  -
   ( SELECT ZSt.ZSt_Stand FROM ZaehlerStand AS ZSt
     WHERE Z.Z_ID = ZSt.ZSt_ZID AND YEAR(ZSt.ZSt_Datum) = 2021)
 ) AS Verbrauch
FROM Haushalte AS HH  INNER JOIN Zaehler AS Z ON HH.HH_ID = Z.Z_HHID
ORDER BY HH_Nachname;
```

Alternativ-Vorschlag:

```
SELECT HH_Nachname AS Nachname, Z_Nummer AS Zaehlernummer,
        (Stand2022- Stand2021) AS Verbrauch
FROM Haushalte INNER JOIN Zaehler ON HH_ID = Z_HHID
        INNER JOIN
        (       SELECT ZSt_ZID , ZSt_Stand AS Stand2022
                FROM ZaehlerStand
                WHERE YEAR(ZSt_Datum) = 2022
        )AS qry22 ON Z_ID = qry22.ZSt_ZID
        INNER JOIN
        (       SELECT ZSt_ZID , ZSt_Stand AS Stand2021
                FROM ZaehlerStand
                WHERE YEAR(ZSt_Datum) = 2021
        )AS qry21 ON Z_ID = qry21.ZSt_ZID
ORDER BY HH_Nachname;
```