Probeklausur Datenbank-Programmierung

Datenbankschema (5m)

Unser Datenbankschema beschreibt einen Teil einer Wander-App, bei der Wanderer Abzeichen für das Erreichen bestimmter Orte erreichen können. Außerdem Organisieren sich die Wanderer in ihren Wandergruppen, Vereinen, und Dachverbänden hierarchisch. Unser Datenbankschema enthält dabei folgende Relationen:

```
wanderer(#id, name, aktiv_seit, org?->organisation, mail, password)
organisation(#id, name, parent?->organisation)
abzeichen(#wanderer->wanderer, #ort->orte, erstes_erreichen)
orte(#id, name, l, b, h)
```

Der **Wanderer** hat eine ID über die er sich identifiziert, und einen Namen. Außerdem wir abgespeichert, seit wann er in der App aktiv ist und in welcher Organisationsstruktur, wenn überhaupt, er sich befindet. Für das Login in der App gibt es Außerdem eine Mailadresse (Alternativschlüssel) und ein Passwort im Klartext. (Diese Tabelle ist in der Beispielinstanz **bereits vorhanden**)

Die **Organisation** identifiziert sich über eine ID und hat einen Namen. Möglicherweise ist sie einer anderen Organisation untergeordnet. (Diese Tabelle ist in der Beispielinstanz **bereits vorhanden**)

Die **Orte** identifizieren sich über eine ID, haben einen Namen, einen Längen- und Breitengrad, sowie eine Höhe über dem Meeresspiegel. (Diese Tabelle ist in der Beispielinstanz **noch nicht vorhanden**)

Die Relation der **Abzeichen** repräsentiert die Viele-zu-Viele-Beziehung zwischen Orten und Wanderern. Dort wird ein Eintrag hinzugefügt, wenn ein Wanderer einen Ort das erste Mal erreicht. Wann dieser Ort das erste Mal von dem Wanderer erreicht worden ist, wird dieses Datum zusätzlich abgespeichert. (Diese Tabelle ist in der Beispielinstanz **noch nicht vorhanden**. Die Tabelle `abzeichen_` mit Testdaten ist in der Beispielinstanz vorhanden.)

Tabelle Orte erzeugen (10m)

Schreiben Sie für die Tabelle `orte` ein `CREATE TABLE` Statement. Die Kurznotation der Relation ist `orte(#id, name, l, b, h)`. Wählen Sie passende Datentypen und Constraints.

- Die ID ist eine ganze Zahl, für die bei Bedarf automatisch neue Werte erzeugt werden.
- Der Breitengrad ist eine Zahle (Grad) zwischen `-90.0000` und `+90.0000` mit einer Genauigkeit von vier Stellen nach dem Komma. Der Längengrad ist eine Zahl (Grad) zwischen `-180.0000` und `+180.0000` mit einer Genauigkeit von vier Stellen nach dem Komma.
- Die Höhe über dem Meeresspiegel (Meter) ist eine 4-Stellige Ganzzahl, die nicht negativ ist.

- Der Name ist eine Zeichenkette von bis zu 30 Zeichen.
- Der Längen- und Breitengrad sind zusammen ein Alternativschlüssel.

Tabelle Abzeichen erzeugen (5m)

Schreiben Sie für die Tabelle `abzeichen` ein `CREATE TABLE` Statement. Die Kurznotation der Relation ist `abzeichen(#wanderer->wanderer, #ort->orte, erstes_erreichen)`.

- Die Spalten `wanderer` und `ort` sind zusammen der Primärschlüssel und referenzieren jeweils die Tabellen `wanderer` und `orte`.
- Wenn für die Spalte `erstes_erreichen` (Datum) nichts eingetragen wird, soll dort das aktuelle Datum eingesetzt werden.
- Wenn ein Wanderer gelöscht wird, sollten auch die entsprechenden Einträge in `abzeichen` automatisch mit gelöscht werden. Ein Ort soll dagegen nicht löschbar sein, wenn er noch in Einträgen in der Tabelle `abzeichen` verwendet wird.

CHECK-Constraints (5m)

Für die Spalte `erstes_erreichen` in der `abzeichen`-Tabelle sollte gelten, dass alle eingetragenen Daten in der Vergangenheit liegen. Ihr Kollege schlägt das `CHECK`-Constraint `CHECK (erstes_erreichen <= NOW())` vor. Bewerten Sie diesen Vorschlag kurz.

Funktionsvolatilität a (2.5m)

Mit der Funktion `trim ( text ) → text` können Sie eine Zeichenkette ohne führende und abschließende Leerzeichen zurückgeben. Ordnen Sie die Funktion einer Funktionsvolatilitätsstufe zu:

☐ Immutable
☐ Stable
☐ Volatile

Funktionsvolatilität b (2.5m)

Mit der Funktion `random ( ) → float` wird eine zufällige Zahl zwischen 0 und 1 zurückgegeben. Ordnen Sie die Funktion einer Funktionsvolatilitätsstufe zu:

☐ Immutable
☐ Stable
☐ Volatile

Trigger (20m)

Schreiben Sie eine Triggerfunktion und binden Sie diese als Zeilentrigger an die passende Tabelle an. Der Trigger soll folgendes Constraint erzwingen:

Wenn ein Abzeichen hinzugefügt oder dessen Datum geändert wird, darf das Datum des Erreichens des Abzeichens nicht vor dem Datum liegen, seit dem der Wanderer, der das Abzeichen erreicht hat, aktiv ist. Ansonsten soll das Datum, seit dem der Wanderer aktiv ist, eingesetzt werden.

View (15m)

Erstellen Sie eine `VIEW` `basisorganisationen`, welche alle diejenigen Organisationen (`id`, `name`) enthält, die unter sich keine weiteren Organisationen gegliedert haben.

```
SELECT * FROM basisorganisationen;
```

id	name
3	Ortsverband Wandern im Burgenland e.V.
4	Wandern mit Baby
5	Wandergruppe “Des Müllers Lust”
6	Frohe Wanderer 1894 e.V.

UPDATE (5m)

Wegen eines Programmierfehlers in der Android-Version der Wander-App schleichen sich in die `wanderer`-Relation immer wieder Nutzer ein, deren Namen mit Leerzeichen anfangen und/oder enden. Geben Sie ein UPDATE-Statement an, welches die überflüssigen Leerzeichen entfernt.

Datensicherheit (10m)

Erklären Sie kurz, wie Passwörter in der Datenbank gespeichert werden sollten und warum Passwörter nicht im Klartext gespeichert werden sollen.

Sperren (10m)

Gegeben sei folgender Inhalt der `wanderer`-Relation:

Zeile	id	name	aktiv_seit	org	mail	password
1	2	Kathrin Friedman	2018-10-10	2	kf@mail.com	geetu
2	4	Thomas Seiler	2018-10-23	NULL	ts@webmail.org	kohph
3	7	Thomas Weber	2018-11-01	7	tw@mailweb.to	okohg
4	9	Uwe Kastner	2018-12-07	2	uw@mail.com	paica
5	12	Kathrin Mehler	2019-02-18	3	km@org3.de	ohwoo

In einer Transaktion T wird der Befehl `START TRANSACTION; SELECT name FROM wanderer WHERE id > 8 OR name LIKE 'Kathrin %' FOR UPDATE;`

ausgeführt. Markieren Sie unter den folgenden Anfragen diejenigen Anfragen, die bei paralleler Ausführung auf die Beendigung der Transaktion T warten müssen, also *blockieren* (mehrere möglich).

- ☐ `UPDATE wanderer SET aktiv_seit = NOW() WHERE org = 2`
- ☐ `INSERT INTO wanderer VALUES (14, 'Kathrin Meyer', NOW(), NULL, 'km@freemail.org', 'geico')`
- ☐ `SELECT mail FROM wanderer WHERE password LIKE '%a%' FOR UPDATE`
- ☐ `SELECT mail, password FROM wanderer WHERE id = 12`
- ☐ `UPDATE wanderer SET org = NULL WHERE name LIKE 'Thomas %'`
- ☐ `LOCK TABLE wanderer`

Mehrbenutzerbetrieb (5m)

Geben Sie ein Beispiel für einen Deadlock. Schreiben Sie auf, welche SQL Befehle man in welcher Reihenfolge in zwei parallelen Transaktionen eingeben muss, um einen Deadlock zu provozieren. Nutzen Sie dafür das Wander-App Schema.

JDBC (10m)

Gegeben sei folgendes Programm, welches die Wander-App-Datenbank nutzt. Das Programm erhält eine Mailadresse und ein Passwort als Kommandozeilenparameter und soll die Daten und IDs aller erhaltenen Abzeichen in chronologischer Reihenfolge ausgeben. Nennen Sie drei Probleme, die dieses Programm aufweist, und erläutern Sie kurz, wie diese zu beheben sind.

```java
import java.sql.*;
import java.io.*;

public class WaApp {

public static void main(String[] args) {
  if (args.length != 2) {
    System.out.println("Bitte Mailadresse und Passwort eingeben");
    System.exit(1);
  }
  try (Connection conn = DriverManager.
    getConnection("jdbc:postgresql://srv/user",
      "user", "password")) {
    Statement stm = conn.createStatement();
    ResultSet result_login = stm.executeQuery(
        "SELECT id FROM wanderer " +
        "WHERE mail = '" + args[0] + "' " +
        "AND password = '" + args[1] + "'");
    int user_id = result_login.getInt(0);
    ResultSet result_badges = stm.executeQuery(
        "SELECT ort, erstes_erreichen FROM abzeichen " +
        "WHERE wanderer = " + user_id + " ORDER BY erstes_erreichen");
    while (result_badges.next()) {
      System.out.println(
        result_badges.getInt(0) + "@" + result_badges.getDate(1)
      );
    }
  } catch (Exception e) {
    System.out.println(e);
    System.exit(2);
  }
}

}
```

Die Verbindungsdaten zum Server (URL, Nutzername, Passwort) sind nur Platzhalter und zählen nicht als Fehler. Auch die Imports sind in Ordnung.

Rekursives SELECT (15m)

Geben Sie für jede Organisation (Name) die Zahl der Wanderer aus, die Mitglied dieser Organisation oder einer ihrer Unterorganisationen sind. Ein Mitglied zählt also mehrfach. Sowohl in der eigenen Organisation, als auch in allen darüberliegenden Organisationen. Für die Wanderer ohne Organisation soll es einen Eintrag “nicht organisiert” geben. Hinweis: Ordnen Sie erst alle Mitglieder ihren Organisationen zu und ordnen Sie dann rekursiv die Mitglieder einer Unterorganisation der jeweiligen Elternorganisation zu. Summieren Sie anschließend wie gewohnt auf.

mitgliederzahl	organisation
12	nicht organisiert
17	Wandern mit Baby
24	Frohe Wanderer 1894 e.V.
35	Ortsverband Wandern im Saalekreis e.V.
16	Ortsverband Wandern im Burgenland e.V.
13	Wandergruppe “Des Müllers Lust”""
54	Regionalverband Wandern in Mitteldeutschland e.V.