

06.12.2022

Übungsblatt 8

Diskussionsteil (im PS zu lösen; keine Abgabe nötig)

- a) ☐ ★ Um Datenbanken sinnvoll zu nutzen, müssen wir oft von Applikationen aus auf die Datenbank zugreifen, um Daten entweder auszulesen oder in die Datenbank zu schreiben. Beantworten Sie dazu folgende Fragen:
- Was versteht man unter dem Cursor-Prinzip? Wofür wird es verwendet?
 - Was sind ODBC und JDBC?
- b) ☐ ★★ Zur Vorbereitung auf die Hausaufgaben wollen wir eine minimale Applikation schreiben, welche auf eine Datenbank zugreift und Daten daraus ausliest. Erstellen Sie dafür eine kleine Datenbank, legen Sie die folgende Tabelle darin an und fügen Sie dann die gegebenen Daten ein:

```
1  CREATE DATABASE discussion;
2  CREATE TABLE sales (
3      sales_date date NOT NULL PRIMARY KEY,
4      amount real NOT NULL
5  );
6  INSERT INTO sales
7  VALUES      ('2022-11-01', 1252.5),
8               ('2022-11-02', 1104.2),
9               ('2022-11-03', 904.9),
10              ('2022-11-04', 1321.4),
11              ('2022-11-05', 1222.9),
12              ('2022-11-06', 1052.1),
13              ('2022-11-07', 943.6);
```

Schreiben Sie anschließend eine kleine Applikation, welche auf die Datenbank zugreift, die Tabelle `sales` ausliest und das Ergebnis auf der Konsole ausgibt. Berechnen Sie in Ihrem Programm weiters die Summe aller Verkäufe und geben Sie diese nach der Tabelle aus.

Hinweis



Starten Sie, je nach Wahl Ihrer Programmiersprache, mit den folgenden Dokumentationsseiten:

- C/C++: <https://www.postgresql.org/docs/13/libpq.html>
- Java: <https://jdbc.postgresql.org/documentation/>
- Python: <https://www.psycopg.org/docs/>

Hausaufgabenteil (Zuhause zu lösen; Abgabe nötig)

Aufgabe 1 (Datenbankanbindung)

[6 Punkte]

In dieser Aufgabe werden Sie, ähnlich wie in der zweiten Diskussionsaufgabe, eine kleine Anwendung schreiben, welche mit einer PostgreSQL-Datenbank kommuniziert. Als Datenbank werden wir hier wie immer Pagila verwenden. Für die Wahl der Programmiersprache und Bibliothek, welche Sie für diese Aufgabe verwenden, stellen wir Ihnen folgende Optionen frei:

- C/C++ (mit libpq)
- Java (mit pgJDBC)
- Python (mit Psycopg)

Wir empfehlen, dass Sie sich aus diesen Sprachen jene aussuchen, mit der Sie am meisten Erfahrung haben.

Hinweis



Für die Lösung dieser Aufgabe werden Sie low-level Schnittstellen verwenden, um über einen Cursor auf die Datenbank zuzugreifen. In der Praxis verwendet man heute Werkzeuge wie Objekt-Relational-Mapper (ORM), um sauberer auf die Datenbank zuzugreifen. Sie werden in späteren Lehrveranstaltungen im Studium lernen, wie man diese verwendet.

Die Anwendung soll ein einfaches Konsolen-Interface zur Verfügung stellen, mit welchem einige Berichte aus der Pagila-Datenbank abgerufen werden können. Das Interface soll hierbei dem herkömmlichen Ablauf für eine Kommandozeile folgen: Warten Sie darauf, dass der Benutzer einen Befehl eingibt, führen Sie diesen dann aus und warten Sie anschließend auf den nächsten Befehl.

- 1 Punkt** Implementieren Sie ein einfaches Konsolen-Interface.
- 1 Punkt** Implementieren Sie die Verbindungsverwaltung für den Zugriff auf die Datenbank. Achten Sie hier besonders auf korrekte Fehlerbehandlung. Der Benutzer der Anwendung soll sinnvolle Fehlermeldungen erhalten, wenn beim Verbindungsaufbau etwas schiefgeht.
- 2 Punkte** Stellen Sie dem Benutzer einen Befehl `rental_report` zur Verfügung, sich einen Bericht über die erfolgten Entleihungen in einem frei wählbaren Zeitraum anzeigen zu lassen. Verwenden Sie hierfür die Abfrage aus Aufgabe 4 von Übungsblatt 6 - der Bericht soll also

aufgeschlüsselt nach Kategorie spaltenweise die Anzahl der entliehenen Filme pro Wochentag anzeigen.

Der Datumsfilter für den Bericht muss vom Benutzer frei wählbar sein. Ein Beispiel für die Verwendung dieses Befehls könnte etwa wie folgt aussehen:

```
(Pagila)> rental_report
```

```
From: 2022-05-01
```

```
To: 2022-05-31
```

category_name	mon	tue	wed	thu	fri	sat	sun	total
Action	9	13	10	13	11	20	11	87
Animation	9	13	13	10	10	8	11	74
Children	10	10	11	8	9	9	14	71
Classics	6	10	9	5	11	9	12	62
Comedy	12	8	8	10	10	13	11	72
Documentary	8	13	9	12	18	18	8	86
Drama	17	12	6	13	10	18	9	85
Family	12	10	7	13	11	19	13	85
Foreign	8	9	10	12	11	15	6	71
Games	12	10	6	11	12	11	7	69
Horror	5	12	5	10	6	9	6	53
Music	9	6	8	7	11	8	12	61
New	8	11	2	11	9	12	7	60
Sci-Fi	12	10	15	13	11	8	15	84
Sports	10	13	5	16	11	12	9	76
Travel	8	12	12	11	7	5	5	60

Achten Sie auch hier wieder auf eine sinnvolle Fehlerbehandlung.

Hinweis



Da hier User-Input in ein SQL-Statement eingebettet werden soll, müssen wir darauf achten, keine Möglichkeit für einen Angriff via SQL-Injection offenzulassen. Recherchieren Sie zum Thema *prepared statements* und finden Sie eine Lösung, wie diese Schwachstelle vermieden werden kann.

- d) 2 Punkte In der vorherigen Unteraufgabe haben wir direkt eine komplexere Abfrage in unserem Programmcode verwendet. Eine Alternative dazu ist es, in der Datenbank eine tabellewertige Funktion zu erstellen und diese aufzurufen, um das Resultat zu erhalten. Sehen Sie sich hierzu die PostgreSQL-Dokumentation für benutzerdefinierte Funktionen¹ an, speziell Funktionen mit RETURNS TABLE.

Legen Sie dann in Ihrer Pagila-Datenbank eine Funktion an, welche denselben Bericht wie in Unteraufgabe c) generiert (die Funktion muss also auch einen Datumsbereich als Parameter entgegennehmen!). Implementieren Sie anschließend in Ihrer Applikation einen Befehl `rental_report_function`, der anstatt die Abfrage direkt abzusetzen, Ihre Funktion verwendet - die von Ihrer Applikation abgesetzte Abfrage soll hier also die Form `SELECT * FROM report_function(from, to);` haben.

¹<https://www.postgresql.org/docs/current/sql-createfunction.html>

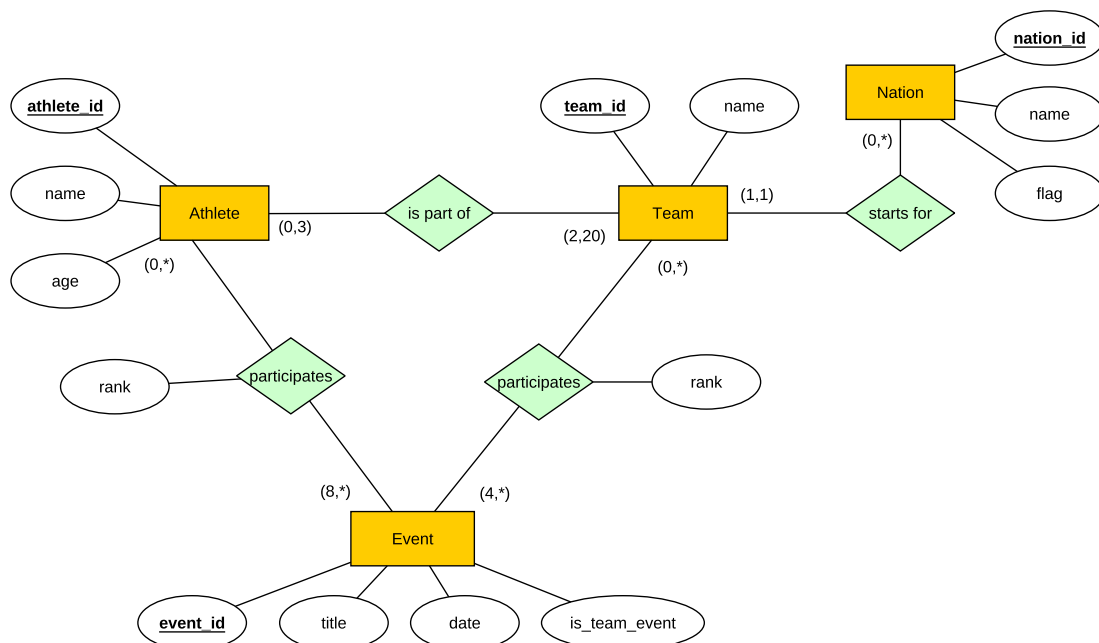
Abgabe

- 📄 exercise1.{c,java,py}: Ihren Programmcode. Wenn Sie Ihr Programm über mehrere Source-Dateien verteilt haben, geben Sie alle davon mit ab.
- 📄 1d.sql: Der SQL-Code für die tabellenwertige Funktion aus Aufgabe d.

Aufgabe 2 (Trigger)**[4 Punkte]**


Trigger erlauben es uns, in einer relationalen Datenbank ereignisabhängig Logik auszuführen. Dies kann zum Beispiel dafür verwendet werden, Änderungslogs zu schreiben oder bestimmte komplexere Einschränkungen zu überprüfen, welche mit den direkt von der Datenbank zur Verfügung gestellten Constraints (FOREIGN KEY, UNIQUE, CHECK etc.) nicht abgebildet werden können.

- a) **1 Punkt** Legen Sie für folgendes ER-Diagramm (aus Übungsblatt 2) eine Datenbank in PostgreSQL an. Achten Sie darauf, auch Fremdschlüsselbeziehungen etc. mit anzulegen und notwendige Mapping-Tabellen für n:m-Beziehungen zu erstellen.


**Abgabe**

📄 2a.sql

- b) **2 Punkte** Das oben gegebene ER-Diagramm definiert einige Einschränkungen, die nicht einfach direkt in PostgreSQL abgebildet werden können. Ein Beispiel dafür ist die Einschränkung, dass ein Team aus maximal 20 Athleten bestehen kann. Um diese Einschränkung zu erzwingen, können wir einen TRIGGER verwenden. Legen Sie also einen TRIGGER an, der verhindert, dass für ein Team mehr als 20 Einträge in die von Ihnen in Aufgabe a) angelegte Mapping-Tabelle zwischen Athleten und Teams eingefügt werden können. Geben Sie für den Fall, dass die Bedingung verletzt wurde, eine sinnvolle Fehlermeldung aus.

Abgabe 2b.sql

- c) 1 Punkt Testen Sie nun Ihren Trigger. Legen Sie mindestens ein Team und mindestens 21 Athleten in Ihrer Datenbank an und versuchen Sie dann, alle 21 Athleten dem selben Team zuzuweisen. Was passiert?

Abgabe 2c.sql

Wichtig: Laden Sie bitte Ihre Lösung in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 23:59 (Mitternacht) ab.