Proseminar Datenbanksysteme

Universität Innsbruck — Institut für Informatik

Frontull S., Ipek Y., Mayerl M., Vötter M., Zangerle E.

10.10.2019

Übungsblatt 7 – Lösungsvorschlag

Wir verwenden in diesem zweiten Übungsblatt zum Thema SQL dieselbe Beispieldatenbank wie auf Blatt 6 (Pagila). Für nähere Informationen zu dieser Datenbank und wie Sie sie einspielen können, sehen Sie sich das GitHub Repository[1] dazu an.
Achten Sie bitte darauf, dass Ihre Lösungen auf PostgreSQL 10.5 lauffähig sein müssen. Verwenden Sie für die Abgabe Ihrer Lösungen exakt die in den Aufgabenstellungen angegebenen Dateinamen. Geben Sie in OLAT die `*.sql`-Dateien *direkt* ab — nicht gepackt in ZIPs oder anderen Archivformaten. Ihre Lösungen werden automatisch von einem Skript auf Korrektheit überprüft, und das Skript kann Ihre Lösungen nicht finden, wenn sie nicht exakt wie angegeben benannt sind.

Diskussionsteil (im PS zu lösen; keine Abgabe nötig)

a) ☆ Gegeben sei folgende Relation:

`Verkauf(ID, KundeID, ArtikelID, Datum, Menge, Einzelpreis)`

Schreiben Sie eine SQL-Abfrage, welche ermittelt, wie viele unterschiedliche Kunden im Zeitraum von `01.01.2019` bis `31.01.2019` etwas gekauft haben.

Lösung ✔

```sql
SELECT   COUNT(DISTINCT KundeID) AS AnzahlKunden
FROM     Verkauf
WHERE    Datum BETWEEN '2019-01-01' AND '2019-01-31'
```

b) ☆ Übersetzen Sie die SQL-Abfrage aus Aufgabe a) in die Relationale Algebra.

Lösung ✔

$$\gamma_{;\text{CountDistinct}(\text{KundeID})\rightarrow\text{AnzahlKunden}}(\sigma_{\text{Datum}\geq'2019-01-01'\wedge\text{Datum}\leq'2019-01-31'}\text{Verkauf})$$

c) ☆☆ Übersetzen Sie die gegebene SQL-Abfrage, welche eine korrelierte Subquery verwendet, in die Relationale Algebra.

```sql
SELECT      StudentName
FROM        Student
WHERE       EXISTS (
```

[1] https://github.com/devrimgunduz/pagila

```
4       SELECT      1
5       FROM        attends
6       WHERE       Student.StudentID = attends.StudentID
7       AND         attends.grade = 2
8   )
```

Lösung ✔

$\pi_{\text{StudentName}}\text{Student} \ltimes_{\text{Student.StudentID}=\text{attends.StudentID}} \sigma_{\text{grade}=2}\text{attends}$

d) ★★ Diskutieren Sie folgende Fragen mit Ihren Kolleginnen und Kollegen:

- Sind Outer Joins kommutativ?

Lösung ✔

Nein, Outer Joins sind im Gegensatz zu Inner Joins nicht kommutativ. Beispielsweise ist $A ⟕ B$ offensichtlich nicht dasselbe wie $B ⟕ A$.

- Sind nicht-korrelierte Subqueries immer performanter als korrelierte Subqueries?

Lösung ✔

Nein, sind sie nicht. Eine korrelierte Subquery kann beispielsweise schneller sein, wenn sie aus einem Index bedient wird, der gezieltes Lesen relevanter Zeilen ermöglicht.

Hausaufgabenteil (Zuhause zu lösen; Abgabe nötig)

Halten Sie sich bei den folgenden Aufgaben strikt an die angegebenen Dateinamen und achten Sie darauf, dass Ihre Queries die geforderten Ausgabespalten in der richtigen Reihenfolge ausgeben. Der Name der ausgegebenen Spalten ist nicht wichtig, aber die Reihenfolge muss stimmen.

Aufgabe 1 (Gruppierung und Aggregation) [3 Punkte]

In dieser Aufgabe werden Sie einige Abfragen mit Gruppierungen und Aggregationen schreiben. Geben Sie für jede Aufgabe eine SQL-Datei mit dem angegebenen Dateinamen ab.

a) *1 Punkt* Ermitteln Sie für jeden Kunden (Tabelle `customer`), der schon einmal etwas bezahlt hat, die Summe all seiner Zahlungen (Tabelle `payment`). Sortieren Sie das Resultat absteigend nach dieser Summe. Dazu sollen vier Spalten ausgegeben werden (in dieser Reihenfolge):

- Kunden-ID (Quellspalte: `customer_id`)
- Vorname (Quellspalte: `first_name`)
- Nachname (Quellspalte: `last_name`)
- Summe der Zahlungen (Quellspalte: `amount`)

Abgabe

`exercise1_a.sql`

b) *1 Punkt* Ermitteln Sie für jeden Film (Tabelle `film`), der mindestens einmal entliehen worden ist, wie oft er entliehen wurde und zusätzlich wie viele *unterschiedliche* Kunden ihn entliehen haben. Die Information, wer wann welchen Film entliehen hat, finden Sie in der Tabelle `rental`. Sie brauchen zusätzlich auch noch die Tabelle `inventory`, um Inventarplätze auf Filme zu mappen.

Geben Sie in Ihrer Lösunge folgende Spalten aus (in dieser Reihenfolge):

- Film-ID (Quellspalte: `film_id`)
- Titel (Quellspalte: `title`)
- Anzahl Entleihungen
- Anzahl unterschiedlicher Kunden, die den Film entliehen haben

Abgabe

`exercise1_b.sql`

c) *1 Punkt* Ermitteln Sie für alle Rating-Kategorien (Spalte `rating`), wie viel die Entleihgebühr (Spalte `rental_rate`) für einen Film dieser Kategorie im Durchschnitt ist. Es sollen dabei nur jene Kategorien ausgegeben werden, deren durchschnittliche Entleihgebühr mindestens `3.0` ist. Dazu sollen zwei Spalten ausgegeben werden (in dieser Reihenfolge):

- Rating (Quellspalte: `rating`)
- Durchschnittliche Entleihgebühr

Abgabe

`exercise1_c.sql`

Aufgabe 2 (Subqueries) [3 Punkte]

In dieser Aufgabe werden Sie einige Abfragen mithilfe von Subqueries schreiben. Geben Sie für jede Aufgabe eine SQL-Datei mit dem angegebenen Dateinamen ab.

a) *1 Punkt* Ermitteln Sie unter Verwendung einer Subquery, welche Schauspieler (Tabelle `actor`) im Film (Tabelle `film`) mit dem Titel (Spalte `title`) *SUNSET RACER* mitgespielt haben. Verwenden Sie dafür eine Subquery — etwa mittels eines `IN`-Operators in der `WHERE`-Klausel. Die Information, welcher Schauspieler in welchem Film mitgespielt hat, finden Sie in der Tabelle `film_actor`. Es sollen zwei Spalten ausgegeben werden (in dieser Reihenfolge):

- Vorname (Quellspalte: `first_name`)
- Nachname (Quellspalte: `last_name`)

Abgabe

`exercise2_a.sql`

b) *1 Punkt* Ermitteln Sie mithilfe einer korrelierten Subquery für jeden Kunden (Tabelle `customer`) wie viel er oder sie insgesamt bereits bezahlt hat (Tabelle `payment`). Beachten Sie, dass Sie diese Aufgabe mit einer korrelierten Subquery lösen müssen — die Lösung ist also nicht dieselbe wie bei Aufgabe 1a, die eine sehr ähnliche Aufgabenstellung hat. Es sollen vier Spalten ausgegeben werden (in dieser Reihenfolge):

- Kunden-ID (Quellspalte: `customer_id`)
- Vorname (Quellspalte: `first_name`)
- Nachname (Quellspalte: `last_name`)
- Summe der Zahlungen (Quellspalte: `amount`)

Abgabe

`exercise2_b.sql`

c) *1 Punkt* Ermitteln Sie mithilfe von Subqueries alle Kunden (Tabelle `customer`), welche insgesamt bereits mehr bezahlt haben als der Durchschnitt aller Kunden (Tabelle `payment`). Es sollen drei Spalten ausgegeben werden:

- Kunden-ID (Quellspalte: `customer_id`)
- Vorname (Quellspalte: `first_name`)
- Nachname (Quellspalte: `last_name`)

Abgabe

`exercise2_c.sql`

Aufgabe 3 (Mengenoperationen) [2 Punkte]

In dieser Aufgabe werden Sie eine Abfragen mithilfe des Mengenoperators `UNION ALL` schreiben. Geben Sie dafür eine SQL-Datei mit dem angegebenen Dateinamen ab.
Ermitteln Sie, wie in Aufgabe 1a, für jeden Kunden, der schon einmal etwas bezahlt hat, die Summe seiner Zahlungen. Geben Sie hier aber zusätzlich noch eine Zeile aus, in der die Gesamtsumme *aller* Zahlungen steht. Dabei soll in der ID-Spalten *0*, in der Vornamen-Spalte *Summe* und in der Nachnamen-Spalte gar nicht (also ein leerer String) stehen. Insgesamt sollten diese vier Spalten ausgegeben werden:

- Kunden-ID
- Vorname
- Nachname
- Summe der Zahlungen

Eine Beispielausgabe sollte also etwa wie folgt aussehen:
Verwenden Sie hierfür einen `UNION ALL`-Operator, beispielsweise in einer Abfrage der Form

```
1       SELECT /* snip - calculate sums for individual customers */
2
3       UNION ALL
4
5       SELECT /* snip - calculate overall total row*/
```

customer_id	first_name	last_name	amount
248	CAROLINE	BOWMAN	50.85
281	LEONA	OBRIEN	50.86
...	...	...	...
0	Summe		67416.51

Achten Sie darauf, dass die Ergenisse nach wie vor **aufsteigend nach dem Betrag sortiert** sein sollen.

Abgabe

`exercise3.sql`

Aufgabe 4 (Report Entleihungen) [2 Punkte]

In dieser Aufgabe sollen sie mittels SQL einen kleinen Bericht erstellen. Stellen Sie sich folgendes Szenario vor: Ihr Chef möchte, um Werbemaßnahmen gezielter zu steuern, wissen, welche Kategorie von Filmen im Jahr 2005 an welchem Wochentag wie oft entliehen wurden. Die Ausgabe sollte alphabetisch sortiert nach Kategorie sein. Neben der Kategorie sollen Spalten für alle Wochentage und eine Gesamtspalte ausgegeben werden. Ein Ergebnis für die Abfrage sieht also wie in Tabelle 1 aus.

Category	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total
Action	144	167	169	134	138	170	173	1095
Animation	158	179	173	150	163	166	156	1145
Children	122	132	123	116	155	133	158	939
Classics	135	146	110	120	149	138	132	930
Comedy	53	143	115	135	138	127	121	932
Documentary	139	146	151	155	138	142	173	1044
Drama	148	125	151	152	153	174	150	1053
Family	154	160	151	139	134	178	167	1083
Foreign	159	142	134	143	167	154	123	1022
Games	132	138	132	152	125	135	141	955
Horror	131	121	117	126	98	126	115	834
Music	97	116	130	117	129	117	113	819
New	133	144	113	127	142	130	138	927
Sci-Fi	167	154	154	148	152	142	176	1093
Sports	159	153	177	162	174	168	171	1164
Travel	116	115	131	124	117	111	113	827

Tabelle 1: Beispielausgabe für Aufgabe 4.

Achten Sie darauf, dass Sie die Ergebnisspalten genau so nennen und sie in genau dieser Reihenfolge ausgeben.

Hinweis

Sehen Sie sich die FILTER-Klausel für Aggregatfunktionen an.

Abgabe

exercise4.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.