

22.11.2022

Übungsblatt 6

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 01.01.2020 bis 31.01.2020 etwas gekauft haben.
- b) ☐ ★ Übersetzen Sie die SQL-Abfrage aus Aufgabe a) in die Relationale Algebra.
- c) ☐ ★★ Übersetzen Sie die gegebene SQL-Abfrage, welche eine korrelierte Subquery verwendet, in die Relationale Algebra.
- ```
1 SELECT StudentName
2 FROM Student
3 WHERE EXISTS (
4 SELECT 1
5 FROM attends
6 WHERE Student.StudentID = attends.StudentID
7 AND attends.grade = 2
8)
```
- d) ☐ ★★ Diskutieren Sie folgende Fragen mit Ihren Kolleginnen und Kollegen:
- Sind Outer Joins kommutativ?
  - Sind nicht-korrelierte Subqueries immer performanter als korrelierte Subqueries?

### Hausaufgabenteil (Zuhause zu lösen; Abgabe nötig)

Wir verwenden in diesem zweiten Übungsblatt zum Thema SQL dieselbe Beispieldatenbank (Pagila) wie bei Übungsblatt 5. Falls die Datenbank bei Ihnen nicht eingerichtet ist, erstellen Sie über Ihren SQL-Client eine neue Datenbank. Importieren Sie das Schema `pagila-schemaOLAP.sql` und die Daten `pagila-insert-dataOLAP.sql`.

Achten Sie bitte darauf, dass Ihre Lösungen auf PostgreSQL 13 lauffähig sein müssen. Ihre Lösungen werden automatisch von einem Skript auf Korrektheit überprüft.

## 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 (Query) und eine TXT-Datei (Resultat) mit den angegebenen Dateinamen ab.

- a) 1 Punkt Ermitteln Sie für jeden Film (Tabelle `film`), der schon mal ausgeliehen wurde, wie viel dieser über den Verleih (Tabelle `rental`) insgesamt eingespielt hat. Filtern Sie nach Filmen die über 210 eingespielt haben.

Reihenfolge und Bezeichnung der Ergebnisspalten:

- `title` (Name des Films)
- `total_payment` (Summe der Zahlungen)

Sortieren Sie das Resultat absteigend nach `total_payment`.

### Hinweis



Sie benötigen zusätzlich die Tabellen `inventory` und `payment`. Schauen Sie sich alle Tabellen genau an und versuchen Sie die Beziehungen zu verstehen.

### Abgabe



`exercise1/a.sql`

`exercise1/a_result.txt`

- b) 1 Punkt Ermitteln Sie wie oft jeder Schauspieler (Tabelle `actor`) in einem Film (Tabelle `film`) mitgespielt hat.

Reihenfolge und Bezeichnung der Ergebnisspalten:

- `first_name` (Vorname des Schauspielers)
- `last_name` (Nachname des Schauspielers)
- `movie_count` (Anzahl der Filme)

Sortieren Sie das Resultat absteigend nach `movie_count` und zusätzlich alphabetisch nach `last_name` und `first_name`.

### Abgabe



`exercise1/b.sql`

`exercise1/b_result.txt`

- c) 1 Punkt Ermitteln Sie für jede Kategorie (Tabelle `category`) die durchschnittliche Laufzeit der Filme (Tabelle `film`).


Reihenfolge und Bezeichnung der Ergebnisspalten:


- `category_name` (Name der Kategorie)
- `avg_film_length` (Durchschnittliche Laufzeit der Filme)

Sortieren Sie das Resultat absteigend nach `avg_film_length`.

### Abgabe



 exercise1/c.sql

 exercise1/c\_result.txt

## Aufgabe 2 (Subqueries)

[3 Punkte]


In dieser Aufgabe werden Sie einige Abfragen mithilfe von Subqueries schreiben. Geben Sie für jede Aufgabe eine SQL-Datei (Query) und eine TXT-Datei (Resultat) mit den 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`) *QUEEN LUKE* 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`. Reihenfolge und Bezeichnung der Ergebnisspalten:

- `first_name` (Vorname des Schauspielers)
- `last_name` (Nachname des Schauspielers)

### Abgabe



 exercise2/a.sql

 exercise2/a\_result.txt

- b) **1 Punkt** Ermitteln Sie für jeden Film (Tabelle `film`), wie viel dieser über den Verleih (Tabelle `rental`) insgesamt eingespielt hat. Auch jene Filme die nie ausgeliehen wurden, müssen im Ergebnis enthalten sein (hier muss `total_payment` explizit ein **NULL** Eintrag sein, also nicht 0). Die Lösung ist nicht dieselbe wie bei Aufgabe 1a, die eine ähnliche Aufgabenstellung hat.


Reihenfolge und Bezeichnung der Ergebnisspalten:


- `title` (Name des Films)
- `total_payment` (Summe der Zahlungen)

Sortieren Sie das Resultat **aufsteigend** nach `total_payment` und alphabetisch nach `title`.

### Abgabe



 exercise2/b.sql

 exercise2/b\_result.txt

- c) **1 Punkt** Ermitteln Sie für jeden Mitarbeiter (Tabelle `staff`), wie viel die Einnahmen pro Kunde durchschnittlich betragen. Nehmen Sie dann den höchsten Wert und geben Sie diesen als `highest_avg_payment_from_customer` an. Beachten Sie, dass Sie diese Aufgabe mit einer korrelierten Subquery lösen müssen.

Reihenfolge und Bezeichnung der Ergebnisspalten:


- `first_name` (Vorname des Mitarbeiters)
- `last_name` (Nachname des Mitarbeiters)


- `highest_avg_payment_from_customer` (durchschnittliche Zahlung des jeweiligen Kunden mit dem höchsten Wert)

#### Hinweis

Am Ende sollte für jeden Mitarbeiter genau eine Zeile im Ergebnis enthalten sein.

#### Abgabe

 `exercise2/c.sql`

 `exercise2/c_result.txt`

## Aufgabe 3 (Mengenoperationen)

[2 Punkte]

In dieser Aufgabe werden Sie eine Abfrage mithilfe des Mengenoperators `UNION ALL` schreiben. Geben Sie dafür eine SQL-Datei (Query) und eine TXT-Datei (Resultat) mit den angegebenen Dateinamen ab.

Für die erste Menge müssen Sie (wie in Aufgabe 1a) für jeden Film, die Summe der Zahlungen ermitteln. Geben Sie zusätzlich an, wie viel beim Verleih im Durchschnitt für den jeweiligen Film gezahlt wurde.

Für die zweite Menge müssen Sie das gleiche Prinzip auf Kategorien anwenden, um herauszufinden wie viel jede einzelne Kategorie insgesamt eingespielt hat und wie viel durchschnittlich gezahlt worden ist. Fügen Sie bei den Einträgen der Kategorie die Spalte `title` mit dem Inhalt *Category Pricings* ein.

Vereinigen Sie diese zwei Mengen anschließend mittels dem `UNION ALL`-Operator, beispielsweise in einer Abfrage der Form:

```
1 SELECT /* snip - calculate sum and avg for each film */
2 UNION ALL
3 SELECT /* snip - calculate sum and avg for each category */
```

Reihenfolge und Bezeichnung der Ergebnisspalten:

- `title` (Filmtitel bzw. bei Kategorien *Category Pricings*)
- `category_name` (Name der Kategorie)
- `total_earnings` (Summe der Zahlungen)
- `average_payment` (Durchschnittliche Zahlung)


Achten Sie darauf, dass die Ergebnisse **absteigend** nach `total_earnings`, alphabetisch nach `title` und `category_name` sortiert sein sollen.


Die Ausgabe sollte also etwa wie folgt aussehen (Beispiel):

| title             | category_name | total_earnings | average_payment |
|-------------------|---------------|----------------|-----------------|
| Category Pricings | Sports        | 5959.61        | 8.76            |
| Category Pricings | Sci-Fi        | 5189.42        | 7.63            |
| ...               | ...           | ...            | ...             |
| VIDEOTAPE ARSENIC | Games         | 131.27         | 6.56            |
| DOGMA FAMILY      | Animation     | 116.83         | 5.84            |
| ...               | ...           | ...            | ...             |

## Abgabe



 3.sql

 3\_result.txt

## Aufgabe 4 (Report Entleihungen)

[2 Punkte]

In dieser Aufgabe sollen sie mittels SQL einen kleinen Bericht erstellen. Geben Sie dafür eine SQL-Datei (Query) und eine TXT-Datei (Resultat) mit den angegebenen Dateinamen ab. Stellen Sie sich folgendes Szenario vor: Ihr Chef möchte, um Werbemaßnahmen gezielter zu steuern, wissen, welche Kategorie von Filmen im Juni 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 folgt aus (Beispiel):

| category_name | mon | tue | wed | thu | fri | sat | sun | total |
|---------------|-----|-----|-----|-----|-----|-----|-----|-------|
| Action        | 15  | 27  | 53  | 61  | 55  | 89  | 73  | 373   |

Reihenfolge und Bezeichnung der Ergebnisspalten:

- category\_name (Name der Kategorie)
- mon (Montag)
- tue (Dienstag)
- wed (Mittwoch)
- thu (Donnerstag)
- fri (Freitag)
- sat (Samstag)
- sun (Sonntag)
- total (Summe der Entleihungen für den Zeitraum)

## Hinweis




Sehen Sie sich die FILTER-Klausel für Aggregatfunktionen<sup>a</sup> an. Weiters stellt Ihnen PostgreSQL<sup>b</sup> Funktionen zum extrahieren des Datums zur Verfügung.


<sup>a</sup><https://www.postgresql.org/docs/13/sql-expressions.html#SYNTAX-AGGREGATES>

<sup>b</sup><https://www.postgresql.org/docs/13/functions-datetime.html#FUNCTIONS-DATETIME-EXTRACT>

## Abgabe



 4.sql

 4\_result.txt

**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.