

22.11.2022

Übungsblatt 6 – Lösungsvorschlag

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.

Lösung



```
1  SELECT  COUNT(DISTINCT KundeID) AS AnzahlKunden
2  FROM    Verkauf
3  WHERE   Datum BETWEEN '2020-01-01' AND '2020-01-31'
```

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

Lösung



$\gamma_{\text{CountDistinct}(\text{KundeID}) \rightarrow \text{AnzahlKunden}}(\sigma_{\text{Datum} \geq '2020-01-01' \wedge \text{Datum} \leq '2020-01-31'} \text{Verkauf})$

- 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  )
```

Lösung

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

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 $A \bowtie B$ offensichtlich nicht dasselbe wie $B \bowtie A$.

- 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)

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-schema.sql`^{OLAT} und die Daten `pagila-insert-data.sql`^{OLAT}.

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

Lösung



Query

```
1  SELECT      film.title, SUM(payment.amount) AS total_payment
2  FROM        film
3  INNER JOIN  inventory
4  ON          film.film_id = inventory.film_id
5  INNER JOIN  rental
6  ON          rental.inventory_id = inventory.inventory_id
7  INNER JOIN  payment
8  ON          payment.rental_id = rental.rental_id
9  GROUP BY   film.film_id, film.title
10 HAVING      SUM(payment.amount) > 210
11 ORDER BY   total_payment DESC
```

Result

title	total_payment
TELEGRAPH VOYAGE	231.73
WIFE TURN	223.69
ZORRO ARK	214.69

(3 rows)

- 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

Lösung



Query

```
1  SELECT      actor.first_name,
2              actor.last_name,
3              COUNT(actor.actor_id) AS movie_count
4  FROM        actor
5  INNER JOIN  film_actor
6  ON          actor.actor_id = film_actor.actor_id
7  INNER JOIN  film
8  ON          film.film_id = film_actor.film_id
9  GROUP BY   actor.actor_id, actor.first_name, actor.last_name
10 ORDER BY   movie_count DESC, last_name, first_name
```

Result

first_name	last_name	movie_count
GINA	DEGENERES	42
WALTER	TORN	41
MARY	KEITEL	40
MATTHEW	CARREY	39
SANDRA	KILMER	37
...		
JUDY	DEAN	15
JULIA	FAWCETT	15
EMILY	DEE	14

(200 rows)

- 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

Lösung



Query

```
1  SELECT      category.name AS category_name,
2              AVG(film.length) AS avg_film_length
3  FROM        film
4  INNER JOIN  film_category
5  ON          film.film_id = film_category.film_id
6  INNER JOIN  category
7  ON          film_category.category_id = category.category_id
8  GROUP BY   category.category_id, category.name
9  ORDER BY   avg_film_length DESC
```

Result

category_name	avg_film_length
Sports	128.2027027027027027
Games	127.8360655737704918
Foreign	121.6986301369863014
Drama	120.8387096774193548
Comedy	115.8275862068965517
...	
Children	109.8000000000000000
Documentary	108.7500000000000000
Sci-Fi	108.1967213114754098

(16 rows)

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

Lösung



Query

```
1  SELECT      actor.first_name,
2              actor.last_name
3  FROM        actor
4  WHERE       actor_id IN (
5      SELECT      actor_id
6      FROM        film_actor
7      INNER JOIN  film
8      ON         film_actor.film_id = film.film_id
9      WHERE       film.title = 'QUEEN LUKE'
10 )
```

Result

first_name	last_name
EWAN	GOODING
SPENCER	PECK
MARY	TANDY
RIVER	DEAN
JAYNE	SILVERSTONE
RIP	WINSLET

(6 rows)

- 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

Lösung



Query

```
1  SELECT      film.title, total_payment
2  FROM        film
3  LEFT JOIN
```

```

4      (
5          SELECT      film.film_id AS fid,
6                      SUM(payment.amount) AS total_payment
7          FROM        film
8          INNER JOIN   inventory
9          ON           film.film_id = inventory.film_id
10         INNER JOIN   rental
11         ON           rental.inventory_id = inventory.inventory_id
12         INNER JOIN   payment
13         ON           payment.rental_id = rental.rental_id
14         GROUP BY     film.film_id
15     ) v
16 ON      film.film_id = fid
17 ORDER BY total_payment ASC, film.title

```

Result

title	total_payment
OKLAHOMA JUMANJI	5.94
TEXAS WATCH	5.94
FREEDOM CLEOPATRA	5.95
DUFFEL APOCALYPSE	6.93
YOUNG LANGUAGE	6.93
...	
VOLUME HOUSE	NULL
WAKE JAWS	NULL
WALLS ARTIST	NULL
(1000 rows)	

- c) **1 Punkt** Ermitteln Sie für jeden Mitarbeiter (Tabelle staff), wie viel die Einnahmen pro Kunde durchschnittlich betrugen. 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

Lösung



Query

```
1  SELECT staff.first_name,  
2         staff.last_name,  
3         (  
4         SELECT MAX(avg_payment_from_customer)  
5         FROM  
6         (  
7         SELECT      AVG(payment.amount)  
8         AS avg_payment_from_customer  
9         FROM        payment  
10        WHERE       payment.staff_id = staff.staff_id  
11        GROUP BY    payment.customer_id  
12        ) AS v  
13        ) AS highest_avg_payment_from_customer  
14 FROM staff
```

Result

first_name	last_name	highest_avg_payment_from_customer
Mike	Hillyer	6.1150000000000000
Jon	Stephens	6.6053846153846154

(2 rows)

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

Lösung



Query

```

1  WITH t AS
2  (
3  SELECT      film.film_id AS film_id,
4              category.category_id AS category_id,
5              film.title AS film_title,
6              payment.amount AS payment_amount,
7              category.name AS category_name
8  FROM        category
9  INNER JOIN  film_category
10 ON          film_category.category_id = category.category_id
11 INNER JOIN  film
12 ON          film.film_id = film_category.film_id
13 INNER JOIN  inventory
14 ON          film.film_id = inventory.film_id

```

```

15     INNER JOIN rental
16     ON         rental.inventory_id = inventory.inventory_id
17     INNER JOIN payment
18     ON         payment.rental_id = rental.rental_id
19 )
20 SELECT *
21 FROM
22 (
23     SELECT      t.film_title AS title,
24                t.category_name AS category_name,
25                SUM(t.payment_amount) AS total_earnings,
26                AVG(t.payment_amount) AS average_payment
27     FROM        t
28     GROUP BY    t.film_id, t.film_title, t.category_name
29
30     UNION ALL
31
32     SELECT      'Category Pricings' AS title,
33                t.category_name AS category_name,
34                SUM(t.payment_amount) AS total_earnings,
35                AVG(t.payment_amount) AS average_payment
36     FROM        t
37     GROUP BY    t.category_id, category_name
38 ) v
39 ORDER BY total_earnings DESC, title, category_name

```

Result

title	category_name	total_earnings	average_payment
Category Pricings	Sports	5314.21	4.5073876166242578
Category Pricings	Sci-Fi	4756.98	4.3205994550408719
Category Pricings	Animation	4656.30	3.9933962264150943
Category Pricings	Drama	4587.39	4.3277264150943396
Category Pricings	Comedy	4383.58	4.6584272051009564
...			
FREEDOM CLEOPATRA	Comedy	5.95	1.19000000000000000000
OKLAHOMA JUMANJI	New	5.94	0.99000000000000000000
TEXAS WATCH	Horror	5.94	0.99000000000000000000

(974 rows)

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^a an. Weiters stellt Ihnen PostgreSQL^b Funktionen zum extrahieren des Datums zur Verfügung.

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

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

Abgabe



4.sql

4_result.txt

Lösung



Query

```
1  SELECT      category.name AS category_name,
2              COUNT(*) FILTER (WHERE EXTRACT(ISODOW FROM rental.rental_date) = 1) AS mon,
3              COUNT(*) FILTER (WHERE EXTRACT(ISODOW FROM rental.rental_date) = 2) AS tue,
4              COUNT(*) FILTER (WHERE EXTRACT(ISODOW FROM rental.rental_date) = 3) AS wed,
```

```

5          COUNT(*) FILTER (WHERE EXTRACT(ISODOW FROM rental.rental_date) = 4) AS thu,
6          COUNT(*) FILTER (WHERE EXTRACT(ISODOW FROM rental.rental_date) = 5) AS fri,
7          COUNT(*) FILTER (WHERE EXTRACT(ISODOW FROM rental.rental_date) = 6) AS sat,
8          COUNT(*) FILTER (WHERE EXTRACT(ISODOW FROM rental.rental_date) = 7) AS sun,
9          COUNT(*) AS total
10 FROM      rental
11 INNER JOIN inventory
12 ON        rental.inventory_id = inventory.inventory_id
13 INNER JOIN film_category
14 ON        inventory.film_id = film_category.film_id
15 INNER JOIN category
16 ON        film_category.category_id = category.category_id
17 WHERE     EXTRACT(MONTH FROM rental.rental_date) = 6 AND
18          EXTRACT(YEAR FROM rental.rental_date) = 2005
19 GROUP BY  category.name
20 ORDER BY  category.name

```

Result

category_name	mon	tue	wed	thu	fri	sat	sun	total
Action	22	24	27	23	13	27	24	160
Animation	25	22	34	16	28	25	24	174
Children	17	14	14	14	25	23	23	130
Classics	28	18	18	17	21	16	18	136
Comedy	28	20	11	21	16	17	22	135
...								
Sci-Fi	29	23	26	17	24	22	21	162
Sports	19	15	23	26	23	26	27	159
Travel	21	8	28	21	17	10	19	124

(16 rows)

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.