

28.11.2023

## Übungsblatt 7 – Lösungsvorschlag

### Diskussionsteil (im PS zu lösen; keine Abgabe nötig)

- a) ☐ ★ Gegeben sei ein Relationenmodell mit Relationenschema Verkauf (ID, KundeID, ArtikelID, Datum, Menge, Einzelpreis). Schreiben Sie eine SQL-Abfrage, die ermittelt, wie viele unterschiedliche Kund\*innen im Jänner 2020 zumindest einen Artikel 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 der vorherigen Unteraufgabe in einen zu ihr äquivalenten Relationenalgebra-Ausdruck.

#### Lösung



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

- c) ☐ ★★ Übersetzen Sie folgende SQL-Abfrage, die auf einen korrelierte Subquery zurückgreift, in einen zu ihr äquivalenten Relationenalgebra-Ausdruck.

```
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)
```

#### Lösung



$\pi_{\text{StudentName}} \left( \text{Student} \times_{\text{Student.StudentID}=\text{attends.StudentID}} \sigma_{\text{grade}=2}(\text{attends}) \right)$

d) ★★ Diskutieren Sie die folgenden beiden Fragen mit Ihren Kolleg\*innen:

- a) Sind der LEFT OUTER JOIN und der RIGHT OUTER JOIN – unter Vernachlässigung der Reihenfolge der Attribute in der Ergebnistabelle – im Allgemeinen kommutativ, d. h. gilt beispielsweise  $A \bowtie B = B \bowtie A$ ?

**Lösung**



Nein, sowohl der LEFT OUTER JOIN als auch der RIGHT OUTER JOIN sind im Gegensatz zum Kreuzprodukt oder den Inner-Joins nicht kommutativ. Beispielsweise ergibt  $A \bowtie B$  offensichtlich nicht dasselbe wie  $B \bowtie A$ .

- b) 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)

Wie bereits im vorherigen Übungsblatt zum Thema SQL, wird dieselbe Beispieldatenbank (Pagila) benötigt. Falls die Datenbank bei Ihnen nicht eingerichtet ist, erstellen Sie über Ihren SQL-Client eine neue Datenbank. Importieren Sie das Schema `pagila-schema.sqlOLAT` und die Daten `pagila-insert-data.sqlOLAT`.

### Hinweis



Achten Sie bitte darauf, dass Ihre Lösungen auf PostgreSQL 15 lauffähig sein müssen. Ihre Lösungen werden automatisch auf Korrektheit überprüft.

## Aufgabe 1 (Gruppierung und Aggregation)

[3 Punkte]

Diese Aufgabe befasst sich mit Abfragen, die Gruppierungen und Aggregationen beinhalten.

### Hinweis



Geben Sie für jede Unteraufgabe eine SQL-Datei, die die Abfrage (Query) und eine TXT-Datei, die das Resultat beinhaltet, mit den angegebenen Dateinamen ab.

- a) 1 Punkt Ermitteln Sie für jeden Film (Tabelle `film`), der länger als 180 ist und bereits einmal ausgeliehen wurde, wie viel dieser über den Verleih (Tabelle `rental`) insgesamt eingespielt hat. Es sind nur Filme relevant, die weniger als 170 eingespielt haben.

Reihenfolge und Bezeichnung der Ergebnisspalten:

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

Sortieren Sie das Resultat **absteigend** nach `total_payment`.



### Hinweis



Sie benötigen zusätzlich die Tabellen `inventory` und `payment`. Machen Sie sich mit allen Tabellen vertraut und versuchen die Beziehungen zwischen den Tabellen zu verstehen.

### Abgabe



 `exercise1/a.sql`  
 `exercise1/a_result.txt`

### Lösung



#### Query

```
1  SELECT      title, length, 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  WHERE length > 180
10 GROUP BY   film.film_id, film.title
11 HAVING     SUM(payment.amount) < 170
12 ORDER BY  total_payment DESC

```

### Result

title	length	total_payment
-----+-----+		
GANGS PRIDE	185	112.73
MUSCLE BRIGHT	185	95.70
DARN FORRESTER	185	93.82
CHICAGO NORTH	185	89.84
MOONWALKER FOOL	184	88.87
...		
CONSPIRACY SPIRIT	184	16.95
WILD APOLLO	181	15.94
SMOOCHY CONTROL	184	14.88
RUNAWAY TENENBAUMS	181	12.92
YOUNG LANGUAGE	183	6.93
(35 rows)		

- b) 1 Punkt Ermitteln Sie wie oft jede\*r Schauspieler\*in (Tabelle actor) in einem Film (Tabelle film) mitgespielt hat.

Reihenfolge und Bezeichnung der Ergebnisspalten:

- first\_name (Vorname Schauspieler\*in)
- last\_name (Nachname Schauspieler\*in)
- movie\_count (Anzahl der Filme)

Sortieren Sie das Resultat **absteigend** nach movie\_count und zusätzlich **alphabetisch absteigend** nach last\_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 DESC

```

### Result

first_name	last_name	movie_count
GINA	DEGENERES	42
WALTER	TORN	41
MARY	KEITEL	40
MATTHEW	CARREY	39
SANDRA	KILMER	37
...		
ADAM	GRANT	18
JULIA	ZELLWEGER	16
JULIA	FAWCETT	15
JUDY	DEAN	15
EMILY	DEE	14

(200 rows)

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


Reihenfolge und Bezeichnung der Ergebnisspalten:


- category\_name (Name der Kategorie)
- avg\_rental\_duration (Durchschnittliche Mietdauer der Filme)

Sortieren Sie das Resultat **aufsteigend** nach avg\_rental\_duration.

### Abgabe



 exercise1/c.sql

 exercise1/c\_result.txt

### Lösung



#### Query

```

1  SELECT      category.name AS category_name,
2              AVG(rental_duration) AS avg_rental_duration
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_rental_duration ASC

```

### Result

category_name	avg_rental_duration
Sports	4.7162162162162162
New	4.7460317460317460
Documentary	4.7647058823529412
Horror	4.8571428571428571
Sci-Fi	4.8852459016393443
...	
Foreign	5.1095890410958904
Family	5.1739130434782609
Music	5.2352941176470588
Travel	5.3508771929824561

(16 rows)

## Aufgabe 2 (Subqueries)

[3 Punkte]

Diese Aufgabe befasst sich mit Abfragen, die Subqueries beinhalten.

### Hinweis



Geben Sie für jede Unteraufgabe eine SQL-Datei, die die Abfrage (Query) und eine TXT-Datei, die das Resultat beinhaltet, 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) *LUKE MUMMY* mitgespielt haben. Verwenden Sie dafür eine Subquery — etwa mittels eines IN-Operators in der WHERE-Klausel. Die Information, welche\*r Schauspieler\*in in welchem Film mitgespielt hat, finden Sie in der Tabelle film\_actor.

Reihenfolge und Bezeichnung der Ergebnisspalten:

- first\_name (Vorname Schauspieler\*in)
- last\_name (Nachname Schauspieler\*in)

Sortieren Sie das Resultat **alphabetisch aufsteigend** nach last\_name.

### 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 = 'LUKE MUMMY'
10 )
11 ORDER BY last_name ASC
```

### Result

first_name	last_name
CHRISTIAN	AKROYD
JULIA	FAWCETT
ANGELA	HUDSON
MARY	KEITEL
CUBA	OLIVIER
BURT	POSEY
RITA	REYNOLDS
JOHN	SUVARI
BELA	WALKEN

(9 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).

### Hinweis



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 aufsteigend** 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 ASC
```

### 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 jede\*n Mitarbeiter\*in (Tabelle staff), wie viel die Einnahmen pro Kund\*in durchschnittlich betrugen. Entnehmen Sie anschließend den höchsten Wert und geben Sie diesen als highest\_avg\_payment\_from\_customer an.



### Hinweis



Diese Aufgabe muss mit einer korrelierten Subquery gelöst werden.

Reihenfolge und Bezeichnung der Ergebnisspalten:

- first\_name (Vorname Mitarbeiter\*in)
- last\_name (Nachname Mitarbeiter\*in)
- highest\_avg\_payment\_from\_customer (Höchstwert der durchschnittlichen Zahlungen der Kund\*innen pro Mitarbeiter\*in)

Sortieren Sie das Resultat **aufsteigend** nach highest\_avg\_payment\_from\_customer und **alphabetisch aufsteigend** nach last\_name.

### 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]

Bei dieser Aufgabe muss eine Abfrage mithilfe des Mengenoperators UNION ALL geschrieben werden.

### Hinweis



Geben Sie für diese Aufgabe eine SQL-Datei, die die Abfrage (Query) und eine TXT-Datei, die das Resultat beinhaltet, mit den angegebenen Dateinamen ab.

Ermitteln Sie das Ergebnis der folgenden Abfragen und bilden anschließend die Vereinigung, mittels dem UNION ALL-Operator, der beiden Abfragen:

- Für die erste Abfrage müssen Sie (ähnlich 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 Abfrage 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.

Beispielsweise eine Abfrage in folgender Form ist gefragt:

```
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 absteigend** 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

## 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 DESC, category_name DESC
```

### 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
...			
YOUNG LANGUAGE	Documentary	6.93	0.99000000000000000000
DUFFEL APOCALYPSE	Documentary	6.93	0.99000000000000000000
FREEDOM CLEOPATRA	Comedy	5.95	1.19000000000000000000
TEXAS WATCH	Horror	5.94	0.99000000000000000000
OKLAHOMA JUMANJI	New	5.94	0.99000000000000000000

(974 rows)

## Aufgabe 4 (Report Entleihungen)

[2 Punkte]

In dieser Aufgabe soll ein kleiner Bericht mittels SQL erstellt werden.

### Hinweis



Geben Sie für diese Aufgabe eine SQL-Datei, die die Abfrage (Query) und eine TXT-Datei, die das Resultat beinhaltet, 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 August 2005 an welchem Wochentag wie oft entliehen wurden. Die Ausgabe sollte nach Kategorie **alphabetisch** sortiert sein. Neben der Kategorie sollen Spalten für alle Wochentage und eine Gesamtspalte ausgegeben werden.

Ein Ergebnis für die Abfrage sieht beispielsweise wie folgt aus:

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

## 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) = 8 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	84	92	57	22	43	40	46	384
Animation	91	97	37	40	42	52	49	408
Children	72	72	31	38	41	36	42	332
Classics	82	79	30	44	36	32	45	348

Comedy		76		84		39		43		36		29		35		342
...																
Music		56		60		31		35		30		34		31		277
New		78		77		25		35		51		40		40		346
Sci-Fi		86		91		36		41		45		39		47		385
Sports		101		87		49		50		51		49		45		432
Travel		56		66		36		37		35		29		39		298
(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 16:00 ab.