Staatsexamen 66116 / 2020 / Frühjahr

Thema 2 / Teilaufgabe 2 / Aufgabe 2

(*Mode-Kollektionen*)

Stichwörter: SQL, SQL mit Übungsdatenbank

Gegeben sei der folgende Ausschnitt eines Schemas für die Verwaltung von Kollektionen:

Die Tabelle *Promi* beschreibt Promis über ihren eindeutigen Namen, ihr Alter und ihren Wohnort. Kollektion enthält Informationen über *Kollektionen*, nämlich deren eindeutigen Namen, das Jahr und die Saison. Die Tabelle *promotet* verwaltet über Referenzen, welcher Promi welche Kollektion promotet. *Kleidungsstück* speichert die IDs von Kleidungsstücken zusammen mit dem Hauptbestandteil und einer Referenz auf die zugehörige Kollektion. Die Tabelle *hat_getragen* verwaltet über Referenzen, welcher Promi welches Kleidungsstück an welchem Datum getragen hat.

Beachten Sie bei der Formulierung der SQL-Anweisungen, dass die Ergebnisrelationen keine Duplikate enthalten dürfen. Sie dürfen geeignete Views definieren.

```sql
CREATE TABLE Promi (
  Name VARCHAR(255) PRIMARY KEY,
  Alter INTEGER,
  Wohnort VARCHAR(255)
);

CREATE TABLE Kollektion (
  Name VARCHAR(255) PRIMARY KEY,
  Jahr INTEGER,
  Saison VARCHAR(255)
);

CREATE TABLE Kleidungsstueck (
  ID INTEGER PRIMARY KEY,
  Hauptbestandteil VARCHAR(255),
  gehoert_zu VARCHAR(255) REFERENCES Kollektion(Name)
);

CREATE TABLE hat_getragen (
  PromiName VARCHAR(255) REFERENCES Promi(Name),
  KleidungsstueckID INTEGER REFERENCES Kleidungsstueck(ID),
  Datum DATE,
  PRIMARY KEY(PromiName, KleidungsstueckID, Datum)
);

CREATE TABLE promotet (
  PromiName VARCHAR(255),
  KollektionName VARCHAR(255),
  PRIMARY KEY(PromiName, KollektionName)
);

INSERT INTO Promi
  (Name, Alter, Wohnort)
VALUES
  ('Till Schweiger', 52, 'Dortmund'),
  ('Lena Meyer-Landrut', 30, 'Hannover');

INSERT INTO Kollektion VALUES
  ('Gerry Weber', 2020, 'Sommer'),
  ('H & M', 2020, 'Sommer');

INSERT INTO promotet
  (PromiName, KollektionName)
VALUES
  ('Till Schweiger', 'Gerry Weber'),
  ('Lena Meyer-Landrut', 'H & M');

INSERT INTO Kleidungsstueck
  (ID, Hauptbestandteil, gehoert_zu)
VALUES
```

```
  (1, 'Hose', 'Gerry Weber');

INSERT INTO hat_getragen
  (PromiName, KleidungsstueckID, Datum)
VALUES
  ('Till Schweiger', 1, '2021-08-03');
```

(a) Schreiben Sie SQL-Anweisungen, welche die Tabelle `hat_getragen` inklusive aller benötigten Fremdschlüsselconstraints anlegt. Erläutern Sie kurz, warum die Spalte `Datum` Teil des Primärschlüssels ist.

```
CREATE TABLE IF NOT EXISTS hat_getragen (
  PromiName VARCHAR(255) REFERENCES Promi(Name),
  KleidungsstueckID INTEGER REFERENCES Kleidungsstueck(ID),
  Datum DATE,
  PRIMARY KEY(PromiName, KleidungsstueckID, Datum)
);
```

Das Datenbanksystem achtet selbst darauf, dass Felder des Primärschlüssels nicht NULL sind. Deshalb muss man NOT NULL bei keinem der drei Felder angeben.

(b) Schreiben Sie eine SQL-Anweisung, welche die Namen der Promis ausgibt, die eine Sommer-Kollektion promoten (Saison ist „Sommer“).

```
SELECT p.PromiName
FROM promotet p, Kollektion k
WHERE
  k.Saison = 'Sommer' AND
  p.KollektionName = k.Name;
```

```
     prominame
--------------------
 Till Schweiger
 Lena Meyer-Landrut
(2 rows)
```

(c) Schreiben Sie eine SQL-Anweisung, die die Namen aller Promis und der Kollektionen bestimmt, welche der Promi zwar promotet, aber daraus noch kein Kleidungsstück getragen hat.

Lösung mit NOT IN:

```
SELECT * FROM promotet p
WHERE p.KollektionName NOT IN (
  SELECT k.Name FROM Kollektion k
  INNER JOIN Kleidungsstueck s ON s.gehoert_zu = K.Name
  INNER JOIN hat_getragen t ON t.KleidungsstueckID = s.ID
  WHERE t.PromiName = p.PromiName
);
```

Mit EXCEPT:

```
(
  SELECT p.PromiName, p.KollektionName FROM promotet p
)
EXCEPT
(
  SELECT p.PromiName, p.KollektionName FROM promotet p
  INNER JOIN Kollektion k ON p.KollektionName = k.Name
  INNER JOIN Kleidungsstueck s ON s.gehoert_zu = k.Name
  INNER JOIN hat_getragen t ON t.KleidungsstueckID = s.ID
  WHERE t.PromiName = p.PromiName
);
```

(d) Bestimmen Sie für die folgenden SQL-Anweisungen die minimale und maximale Anzahl an Tupeln im Ergebnis. Beziehen Sie sich dabei auf die Größe der einzelnen Tabellen.

Verwenden Sie für die Lösung folgende Notation: – Promi – beschreibt die Größe der Tabelle Promi.

(i)

```sql
SELECT k.Name
FROM Kollektion k, Kleidungsstueck kl
WHERE k.Name = kl.gehoert_zu and k.Jahr = 2018
GROUP BY k.Name
HAVING COUNT(kl.Hauptbestandteil) > 10;
```

> – Kollektion – beschreibt die Anzahl der Tupel in der Tabelle Kollektion. Die minimale Anzahl an Tupeln im Ergebnis ist 0, da es sein kann, dass keine Kollektion den Anforderungen `k.Jahr = 2018` oder `HAVING COUNT(...)` genügt. Die maximale Anzahl an Tupeln im Ergebnis ist – Kollektion –, da nur Namen aus Kollektion ausgewählt werden.

(ii)

```sql
SELECT DISTINCT k.Jahr
FROM Kollektion k
WHERE k.Name IN (
  SELECT pr.KollektionName
  FROM Promi p, promotet pr
  WHERE p.Alter < 30 AND pr.PromiName = p.Name
);
```

> Die minimale Anzahl an Tupeln im Ergebnis ist 0, da es sein kann, dass keine Kollektion promotet wird. Die maximale Anzahl ist das Minimum von – Kollektion – und 30, da die Promis, die Kollektionen beworben haben, die älter als 30 Jahre sind, selbst mindestens 30 Jahre alt sein müssen.
>
> Zugrundeliegende Annahmen: Kollektionen werden nur im Erscheinungsjahr von Promis beworben; Neugeborene, die Kollektionen bewerben, werden ggf. Promis.

(e) Beschreiben Sie den Effekt der folgenden SQL-Anfrage in natürlicher Sprache

```sql
SELECT pr.KollektionName
FROM promotet pr, Promi p
WHERE pr.PromiName = p.Name
GROUP BY pr.KollektionName
HAVING COUNT (*) IN (
  SELECT MAX(anzahl)
  FROM (
    SELECT k.Name, COUNT(*) AS anzahl
    FROM Kollektion k, promotet pr
    WHERE k.Name = pr.KollektionName
    GROUP BY k.Name
  ) as tmp
);
```

> Die Abfrage gibt die Namen aller Kollektionen aus, bei denen die Anzahl der bewerbenden Promis am größten ist.

(f) Formulieren Sie folgende SQL-Anfrage in relationaler Algebra. Die Lösung kann in Baum- oder in Term-Schreibweise angegeben werden, wobei eine Schreibweise genügt.

```sql
SELECT p.Wohnort
FROM Promi p, promotet pr, Kollektion k
WHERE
  p.Name = pr.PromiName AND
  k.Name pr.KollektionName AND
  k.Jahr 2018;
```

(i) Konvertieren Sie zunächst die gegebene SQL-Anfrage in die zugehörige Anfrage in relationaler Algebra nach Standard-Algorithmus.

$$\pi_{\text{Promi.Wohnort}}(\sigma_{\text{Promi.Name=promotet.PromiName} \wedge \text{Kollektion.Name=promotet.KollektionName} \wedge \text{Kollektion.Jahr=2018}} (\text{Promi} \times \text{promotet} \times \text{Kollektion}))$$

(ii) Führen Sie anschließend eine relationale Optimierung durch. Beschreiben und begründen Sie dabei kurz jeden durchgeführten Schritt.

1. Schritt: Um die kartesischen Produkte in Joins zu verwandeln, muss die Selektion in drei Selektionen zerlegt werden.

$$\pi_{\text{Promi.Wohnort}}(\sigma_{\text{Promi.Name=promotet.PromiName}}(\sigma_{\text{Kollektion.Name=promotet.KollektionName}}(\sigma_{\text{Kollektion.Jahr=2018}}(\text{Promi} \times (\text{promotet} \times \text{Kollektion})))))$$

2. Schritt: Man kann die Selektion nach dem Jahr ausführen, bevor die kartesischen Produkte ausgeführt werden. Dadurch werden von vornherein nur die Kollektionen betrachtet, die in dem entsprechenden Jahr aufgetreten sind.

$$\pi_{\text{Promi.Wohnort}}(\sigma Promi.Name = promotet.PromiName(\sigma Kollektion.Name = promotet.KollektionName(Pr$$

3. Schritt: Die zweitinnerste Selektion kann direkt nach dem innersten kartesischen Produkt ausgeführt werden; dadurch wird das Gesamtprodukt nicht so groß. Man benötigt also weniger Rechenzeit und Speicher.

$$\pi_{\text{Promi.Wohnort}}(\sigma Promi.Name = promotet.PromiName(Promi \times \sigma Kollektion.Name = promotet.Kollektion1$$

4. Schritt: Beide Selektionen in Verbindung mit dem jeweiligen kartesischen Produkt können in einen Join verwandelt werden. So wird nicht zuerst das gesamte Produkt berechnet und danach die passenden Eintraege ausgewaehlt, sondern gleich nur die zusammengehörigen Eintraege kombiniert. Auch dies spart Rechenzeit und Speicher.

$$\pi_{\text{Promi.Wohnort}}(Promi \bowtie Promi.Name = promotet.PromiName(promotet \bowtie Kollektion.Name = promotet.K$$