Tutorium 4
SQL
Big Data Engineering

Prof. Dr. Jens Dittrich

bigdata.uni-saarland.de

23. Mai/24. Mai 2022

Verbesserung Übungsblätter - Häufige Fehler

Aufgabe 1:

- DISTINCT vergessen bei (a) (insbesondere bei (a) 3).

Aufgabe 2:

- (a) 1: Das GROUP BY vergessen.
- (a) 3: AVG(Gehalt) bereits in Unteranfragen berechnet.
- (b) 2: Musik als Hauptfach der Lehrer*innen oder als Fach der Klausur.

Aufgabe 3:

- Oft nur ein Fehler je Anfrage identifiziert, obwohl mehrere existieren.
- Lösungsvorschläge vergessen.
- 1.: HAVING als falsch bezeichnet.
- 2.: WHERE in HAVING umgewandelt.

Aufgabe 4:

- 2.: MAX(purchases) nicht separat berechnet und verglichen, wodurch uneindeutige Attribute entstehen.
- 3.: Die Personen genommen, die zwischen 1900 und 1949 in Haushalten gelebt haben, anstatt eingezogen sind.
- 4.: Vergessen, dass die Einkäufe ebenfalls zwischen beiden Daten sein sollen.

SQL - Das heutige Modell

[Customers] : {[CustomerID:int, CustomerName:varchar,
ContactName:varchar, Address:varchar, Country:varchar]}

[Categories] : {[CategoryID:int, CategoryName:varchar, Description:varchar]}

[Employees] : {[EmployeeID:int, LastName:varchar, FirstName:varchar,
BirthDate:date]}

[Shippers] : {[ShipperID:int, ShipperName:varchar]}

[Suppliers] : {[SupplierID:int, SupplierName:varchar, ContactName:varchar,
Address:varchar, Country:varchar]}

[Products] : {[ProductID:int, ProductName:varchar,
SupplierID:(Suppliers→SupplierID),
CategoryID:(Categories→CategoryID), Price:double]}

[Orders] : {[OrderID:int, CustomerID:(Customers→CustomerID),
EmployeeID:(Employees→EmployeeID), OrderDate:date,
ShipperID:(Shippers→ShipperID)]}

[OrderDetails] : {[OrderDetailID:int, OrderID:(Orders→OrderID),
ProductID:(Products→ProductID), Quantity:int]}

SQL - w3schools Hands-On

Ihr könnt eure Anfragen auf w3schools direkt ausprobieren. Dort ist das vorherige Schema mit Beispieldaten bereits vorliegend.

ACHTUNG: w3schools verwendet auf der verlinkten Seite MySQL. Die verwendete SQL Syntax, sowie die Art und Weise in der Anfragen bearbeitet werden, kann bei Verwendung eines anderen Datenbankmanagementsystem von der dortigen abweichen. **In der Klausur gilt die in der Vorlesung vorgestellte Syntax**.

Wiederholung - Frage 1

Frage

Geben Sie die Ausführungsreihenfolge folgender SQL Anfrage an:

```
SELECT    a, b
FROM      Tabelle
WHERE     id = 42
GROUP BY  a, b
HAVING    SUM(c) = 42;
```

Wiederholung - Frage 1

Frage

Geben Sie die Ausführungsreihenfolge folgender SQL Anfrage an:

```
SELECT    a, b
FROM      Tabelle
WHERE     id = 42
GROUP BY  a, b
HAVING    SUM(c) = 42;
```

Lösung

1. `FROM Tabelle`
2. `WHERE id = 42`
3. `GROUP BY a, b`
4. `HAVING SUM(c) = 42`
5. `SELECT a, b`

Wiederholung - Frage 2

Frage

Was gibt die folgende SQL Anfrage aus?

```
SELECT Country
FROM   Suppliers
WHERE  Address LIKE 'an%';
```

(A): Das Land der Anbieter*innen, deren Adresse gleich dem String 'an%' ist.

(B): Das Land der Anbieter*innen, deren Adresse mit 'an' (case-insensitive) anfängt.

(C): Das Land der Anbieter*innen, deren Adresse mit 'an' (case-sensitive) anfängt.

(D): Das Land der Anbieter*innen, deren Adresse mit 'an' (case-sensitive) endet.

Wiederholung - Frage 2

Frage

Was gibt die folgende SQL Anfrage aus?

```
SELECT  Country
FROM    Suppliers
WHERE   Address LIKE 'an%';
```

Lösung

Die richtige Antwort lautet (B):

Das Land der Anbieter*innen, deren Adresse mit 'an' (case-insensitive) anfängt.

ACHTUNG! Dies ist nur das default-Verhalten von SQLite. Man kann auch einstellen, dass Stringvergleiche case-sensitive sind. In anderen Datenbankmanagementsystemen kann das default-Verhalten ebenfalls abweichen.

Wiederholung - Frage 3

Frage

Was gibt die folgende SQL Anfrage aus?

```
SELECT    ProductName, Price
FROM      Products
WHERE     Price > (SELECT AVG(Price)
                   FROM Products)
GROUP BY ProductName, Price;
```

(A): Den Namen und Preis der Produkte, deren Preis über dem durchschnittlichen Preis aller anderen Produkte liegt. Die Ausgabe kann Duplikate enthalten.

(B): Den Namen und Preis der Produkte, deren Preis über dem durchschnittlichen Preis aller anderen Produkte liegt. Die Ausgabe enthält keine Duplikate, da das GROUP BY hier wie ein DISTINCT im SELECT agiert.

(C): Den Namen und Preis der Produkte, deren Preis über dem durchschnittlichen Preis aller Produkte liegt. Die Ausgabe kann Duplikate enthalten.

(D): Den Namen und Preis der Produkte, deren Preis über dem durchschnittlichen Preis aller Produkte liegt. Die Ausgabe enthält keine Duplikate, da das GROUP BY hier wie ein DISTINCT im SELECT agiert.

Wiederholung - Frage 3

Frage

Was gibt die folgende SQL Anfrage aus?

```
SELECT    ProductName, Price
FROM      Products
WHERE     Price > (SELECT AVG(Price)
                   FROM Products)
GROUP BY ProductName, Price;
```

Lösung

Die richtige Antwort lautet (D):

Den Namen und Preis der Produkte, deren Preis über dem durchschnittlichen Preis aller Produkte liegt. Die Ausgabe enthält keine Duplikate, da das GROUP BY hier wie ein DISTINCT im SELECT agiert.

Wiederholung - Frage 4

Frage

Was ist der Fehler in folgender SQL Anfrage?

```
SELECT    OrderDate, MAX(OrderDate)
FROM      Orders
GROUP BY CustomerID;
```

Wiederholung - Frage 4

Frage

Was ist der Fehler in folgender SQL Anfrage?

```
SELECT    OrderDate, MAX(OrderDate)
FROM      Orders
GROUP BY  CustomerID;
```

Lösung

Wir greifen im SELECT auf 'OrderDate' zu ohne darüber gruppiert zu haben. Das ist problematisch, denn der Wert eines Attributs, auf dem nicht gruppiert wurde, ist innerhalb einer Gruppe nicht notwendigerweise gleich. Laut dem SQL-Standard ist eine solche Anfrage nicht korrekt. Je nach System wird die Anfrage aber unterschiedlich behandelt.

Wiederholung - Frage 5

Frage

Was ist der Fehler in folgender SQL Anfrage?

```
SELECT    Price, 42 AS answer
FROM      Products
GROUP BY  Price
WHERE     ProductName = 'Fehler';
```

Wiederholung - Frage 5

Frage

Was ist der Fehler in folgender SQL Anfrage?

```
SELECT    Price, 42 AS answer
FROM      Products
GROUP BY  Price
WHERE     ProductName = 'Fehler';
```

Lösung

WHERE ist eine Bedingung auf Tupeln und nicht auf Gruppen, darf also nicht nach dem GROUP BY stehen.

Aufgabe 1

Frage

Übersetzen Sie folgenden Ausdruck der relationalen Algebra in eine SQL Anfrage:

$R_1 := \rho_{\text{ShipperID'} \leftarrow \text{ShipperID}}\ \text{Orders}$

$R_2 := R_1 \bowtie_{\text{ShipperID'} = \text{ShipperID}}\ \text{Shippers}$

$\pi_{\text{ShipperName}}\ (\sigma_{\text{OrderDate}>11.05.2007\ \wedge\ \text{OrderID} < 10}\ R_2)$

Aufgabe 1

Lösung

```
SELECT DISTINCT ShipperName
FROM   Orders AS O
       JOIN Shippers AS S
            ON O.ShipperID = S.ShipperID
WHERE  OrderID < 10
       AND OrderDate > '2007-05-11';
```

Zu beachten ist hier, dass die Umbenennung des Attributes 'ShipperID' in SQL nicht nötig ist, da die Attribute durch einen Tabellenprefix eindeutig sind!

Aufgabe 2

Frage

Beschreiben Sie in natürlicher Sprache was folgende SQL Anfrage ausgibt.

```
SELECT    P.ProductID, AVG(Quantity)
FROM      OrderDetails AS O, Products AS P
WHERE     O.ProductID = P.ProductID
GROUP BY P.ProductID;
```

Aufgabe 2

Lösung

Die Anfrage gibt für jedes Produkt dessen ProductID sowie die durchschnittliche Anzahl an pro Bestellung bestellten Exemplaren dieses Produktes an.

Aufgabe 3.1

Frage

Geben Sie eine SQL Anfrage an, die je Bestellung deren ID sowie die maximale Quantität von einem Buch, das 'Harry Potter' im Titel trägt ausgibt, das in dieser bestellt wurde.

Aufgabe 3.1

Lösung

```
SELECT    OD.OrderID, MAX(OD.Quantity)
FROM      OrderDetails AS OD
          JOIN Products AS P
               ON P.ProductID = OD.ProductID
          JOIN Categories AS C
               ON P.CategoryID = C.CategoryID
WHERE     ProductName LIKE '%Harry Potter%'
          AND CategoryName = 'Book'
GROUP BY  OD.OrderID;
```

Aufgabe 3.2

Frage

Geben Sie eine SQL Anfrage an, die für jede*n Angestelle*n, die/der nach dem 17.03.1983 geboren wurde, deren Vor- und Nachnamen ausgibt, sowie die Anzahl des von dieses/dieser Angestellten bearbeiteten Bestellungen als 'Anzahl'.

Aufgabe 3.2

Lösung

```
SELECT    FirstName, LastName,
          COUNT(*) AS Anzahl
FROM      Employees AS E
          JOIN Orders AS O
               ON O.EmployeeID = E.EmployeeID
WHERE     BirthDate > '1983-03-17'
GROUP BY E.EmployeeID;
```

Aufgabe 3.3

Frage

Geben Sie eine SQL Anfrage an, die für jedes Bestelldatum die insgesamt an diesem Tag bestellten Produktquantitäten ausgibt, wobei lediglich Tage betrachtet werden, an denen mindestens 5 (nicht zwangsweise verschiedene) Produkte bestellt wurden. Sortieren Sie Ihre Ausgabe absteigend nach dem Datum.

Aufgabe 3.3

Lösung

```
SELECT    OrderDate, SUM(Quantitiy)
FROM      OrderDetails AS OD
          JOIN Orders AS O
               ON OD.OrderID = O.OrderID
GROUP BY  OrderDate
HAVING    COUNT(*) >= 5
ORDER BY  OrderDate DESC;
```

Aufgabe 3.4

Frage

Geben Sie eine SQL Anfrage an, die für jeden/jede Anbieter*in deren/dessen Namen, der/die nicht aus Frankreich stammt, die Anzahl an Produkten angibt, die von diesem/dieser angeboten werden. Betrachten Sie hierbei lediglich Produkte, deren Kategoriename auf 'n' endet, geben Sie lediglich die ersten 10 Elemente aus und sortieren Sie Ihre Ausgabe aufsteigend nach dem Namen der Anbieter*innen.

Aufgabe 3.4

Lösung

```
SELECT    SupplierName, COUNT(*)
FROM      Suppliers AS S
          JOIN Products AS P
               ON P.SupplierID = S.SupplierID
          JOIN Categories AS C
               ON C.CategoryID = P.CategoryID
WHERE     S.Country <> 'France'
          AND C.CategoryName LIKE '%n'
GROUP BY S.SupplierID
ORDER BY SupplierName
LIMIT 10;
```

Aufgabe 3.5

Frage

Geben Sie eine SQL Anfrage an, die für jede*n Anbieter*in, das Land als 'Land', die Namen der Anbieter*innen als 'Anbieter*in', die Anzahl der von ihnen gelieferten (nicht zwangsweise verschiedenen) Produkte als 'AnzahlGeliefert', den Durchschnittspreis dieser Produkte als 'Durchschnittspreis' und die Summe der Produktpreise als 'Preis' ausgibt. Insgesamt sollen nur Produkte berücksichtigt werden, welche zwischen dem 19.05.1997 und dem 19.05.1998 (exklusive) bestellt wurden und zur Kategorie 'Beverages' gehören. Außerdem sollen alle Anbieter*innen ausgeschlossen werden, die nicht mindestens 20 (nicht zwangsweise verschiedene) Produkte geliefert oder Produktquantitäten im Gesamtwert von weniger als 200 Euro geliefert haben.

Aufgabe 3.5

Lösung

```
SELECT    Suppliers.Country AS Land,
          Suppliers.SupplierName AS Anbieter*in,
          COUNT(Products.ProductID) AS AnzahlGeliefert,
          AVG(Products.Price) AS Durchschnittspreis,
          SUM(Products.Price) AS Preis
FROM      Categories, Products, OrderDetails, Orders,
          Suppliers
WHERE     '1997-05-19' < OrderDate
          AND OrderDate < '1998-05-19'
          AND Categories.CategoryName = 'Beverages'
          AND Categories.CategoryID = Products.CategoryID
          AND OrderDetails.OrderID = Orders.OrderID
          AND OrderDetails.ProductID = Products.ProductID
          AND Suppliers.SupplierID = Products.SupplierID
GROUP BY  Suppliers.SupplierID
HAVING    COUNT(*) >= 20 AND
          SUM(Quantity * Products.Price) >= 200;
```

Appendix 1 - wichtige SQL Schlüsselwörter

Schlüsselwörter

- `SELECT`: Projektion; gibt an welche Spalten aus dem Ergebnis entnommen werden sollen
- `FROM`: gibt an welche Tabellen angesprochen werden sollen
- `WHERE`: Selektion; filtert Tupel nach einer Bedingung
- `GROUP BY`: Gruppierung; alle Tupel, die den gleichen Wert in diesen Attributen haben, landen in derselben Gruppe
- `HAVING`: Selektion; filtert durch GROUP BY entstandene Gruppen nach einer Bedingung

Appendix 2 - wichtige SQL Schlüsselwörter

Schlüsselwörter

- `JOIN`: kann im FROM für explizite Joins benutzt werden, z.B. FROM A JOIN B ON A.id = B.id
- `AS`: Umbenennung einer Tabelle
- `ORDER BY`: sortiert die Ausgabe nach den gegebenen Attributen
- `ASC/DESC`: aufsteigend/absteigend
- `LIMIT`: limitiert die Anzahl der Tupel die ausgegeben werden, z.B. LIMIT 3 → höchstens 3 Tupel werden ausgegeben
- `DISTINCT`: verhindert Duplikate in der Ausgabe
- `BETWEEN`: nützlich in Bedingungen, z.B. id BETWEEN x AND y ist äquivalent zu x <= id AND id <= y
- `NOT/AND/OR`: logische Operatoren für Bedingungen
- `LIKE`: für Stringvergleiche; % ist eine Wildcard für beliebig viele Zeichen, _ für genau ein Zeichen, z.B. name LIKE '%d' → name muss mit 'd' aufhören.