Lösungen

Kapitel 3

Die Lösungen bestehen pro Aufgabe aus dem SQL-Statement, der Größe der Ergebnismenge sowie je nach Komplexität der Aufgabe einer Erklärung. Beachten Sie, dass es sich hier nur um Musterlösungen handelt. SQL ist vielfältig, wodurch mehrere Lösungswege möglich sind.

1. Geben Sie die Attribute »Kommentar« und »Bewertung« der Relation »Kommentar« aus.

```
SELECT Kommentar, Bewertung
FROM Kommentar
```

   Größe der Ergebnismenge: 371

2. Geben Sie alle Attribute der Relation »Gutschein« aus.

```
SELECT *
FROM Gutschein
```

   Größe der Ergebnismenge: 15

3. Geben Sie die ersten sieben Einträge der Relation »Bestellung« aus. Die Einträge dürfen sich dabei nicht doppeln.

```
SELECT DISTINCT *
FROM Bestellung
LIMIT 7
```

   Größe der Ergebnismenge: 7

4. Geben Sie alle verschiedenen »Kurs_IDs« aus, die in der Relation »Bestellung« zu finden sind. Die Ausgabe soll aus den dritt-, viert- und fünfthöchsten IDs bestehen.

```
SELECT DISTINCT Kurs_ID
FROM Bestellung
ORDER BY Kurs_ID DESC
LIMIT 3 OFFSET 2
```

Alternativ kann in MySQL auch »LIMIT 2, 3« geschrieben werden.

Größe der Ergebnismenge: 3

5. Geben Sie zwei beliebige Attribute der Relation »Aktion« aus. Das erste Attribut soll dabei als »A« und das zweite als »B« bezeichnet werden.

```
SELECT Anfangsdatum AS A, Enddatum AS B
FROM Aktion
```

Größe der Ergebnismenge: 20

Kapitel 4

Die Lösungen bestehen pro Aufgabe aus dem SQL-Statement, der Größe der Ergebnismenge sowie je nach Komplexität der Aufgabe einer Erklärung. Beachten Sie, dass es sich hier nur um Musterlösungen handelt. SQL ist vielfältig, wodurch mehrere Lösungswege möglich sind.

1. Geben Sie alle Nutzer an, die in Deutschland wohnen und deren Heimatstadt nicht Berlin ist. Begrenzen Sie die Ausgabe auf 5 Einträge.

```
SELECT *
FROM Nutzer
WHERE Land = 'Deutschland' AND Stadt != 'Berlin'
LIMIT 5
```

Statt `Stadt != 'Berlin'` würde auch `NOT Stadt = 'Berlin'` passen.

Größe der Ergebnismenge: 5

2. Geben Sie alle Nutzer an, die nicht in Deutschland, aber auch nicht in Portugal leben. Außerdem sollte der Vorname der Nutzer mit einem Buchstaben größer »F« anfangen. Geben Sie nur den Vornamen, Nachnamen und den Status an.

```
SELECT Vorname, Nachname, Status
FROM Nutzer
WHERE Land != 'Deutschland' AND Land != 'Portugal' AND Vorname
>= 'G'
```


Die Bedingung `Vorname > 'F'` funktionieren nicht, da hiermit auch Vornamen wie »Felix« eingeschlossen werden. Da der Anfangsbuchstabe jedoch größer »F« sein muss, wird hier `Vorname >= 'G'` verwendet.

Größe der Ergebnismenge: 47

3. Geben Sie alle Aktionen an, die im Jahr 2022 stattfinden oder die nur für die Kategorie »Programmierung« gelten.

```
SELECT *
FROM Aktion
WHERE     (Anfangsdatum >= '2022-01-01'
          AND Enddatum <= '2022-12-31')
          OR Kategorie = 'Programmierung'
```

   Größe der Ergebnismenge: 6

4. Geben Sie die Namen und IDs aller Kurse an, die nicht in der Kategorie »Lifestyle« oder »Wirtschaft« liegen.

```
SELECT Kurs_ID, Kurs_Name
FROM Kurs
WHERE NOT Kategorie IN ('Lifestyle', 'Wirtschaft')
```

   Größe der Ergebnismenge: 14

5. Geben Sie die Namen aller Aktionen an, die im Jahr 2022 starten und einen Rabatt von 45 % oder höher anbieten.

```
SELECT Aktionsname
FROM Aktion
WHERE     Rabatt >= 45 AND Anfangsdatum
          BETWEEN '2022-01-01' AND '2022-12-31'
```

   Größe der Ergebnismenge: 2

6. Geben Sie alle Gutscheine an, deren Name kein kleines »l« oder großes »L« beinhaltet.

   Lösung für PostgreSQL:

```
SELECT *
FROM Gutschein
WHERE Gutscheinname NOT ILIKE '%L%'
```

   Lösung für MySQL:

```
SELECT *
FROM Gutschein
WHERE Gutscheinname NOT LIKE '%L%'
```

In MySQL ist die Groß- und Kleinschreibung bei Zeichenketten egal, weshalb hier der Befehl `WHERE Gutscheinname NOT LIKE '%L%'` gesucht ist.

Größe der Ergebnismenge: 7

7. Geben Sie die Vor- und Nachnamen aller Nutzer an, die in einer Stadt wohnen, deren Name ein großes »Y« beinhaltet oder deren Vorname ein kleines »x« beinhaltet. Das kleine »x« darf allerdings nicht am Ende des Vornamens stehen.

```
SELECT Vorname, Nachname
FROM Nutzer
WHERE Stadt LIKE '%Y%' OR Vorname LIKE '%x%_'
```

Da in MySQL auch bei Zeichenketten kein Wert auf Groß- und Kleinschreibung gelegt wird, ist die Aufgabe nur teilweise zu erfüllen. Nach einer Stadt mit »y« kann zwar gesucht werden, ob das »y« jedoch groß oder klein ist, kann nicht beeinflusst werden.

Größe der Ergebnismenge: 4

8. Geben Sie alle Nutzer an, die in einer Stadt wohnen, deren Name Umlaute (»ä«, »ö« oder »ü«) beinhaltet oder die an einem ungeraden Tag geboren sind (z.B. 01. November 2002).

Lösung für PostgreSQL:

```
SELECT *
FROM Nutzer
WHERE     Stadt ~ '^.*[ÄÖÜäöü].*$' OR CAST(Geburtstag AS Text)
          ~ '^[0-9]{4}-[0-9]{2}-[0123][13579]$'
```

Lösung für MySQL:

```
SELECT *
FROM Nutzer
WHERE     Stadt RLIKE '^.*[ÄÖÜ].*$' OR
          Geburtstag RLIKE '^[0-9]{4}-[0-9]{2}-[0123][13579]$'
```

Größe der Ergebnismenge: 50

Kapitel 5

Die Lösungen bestehen pro Aufgabe aus dem SQL-Statement, der Größe der Ergebnismenge sowie je nach Komplexität der Aufgabe einer Erklärung. Beachten Sie, dass es sich hier nur um Musterlösungen handelt. SQL ist vielfältig, wodurch mehrere Lösungswege möglich sind.

1. Geben Sie die verschiedenen Länder an, aus denen die Nutzer stammen. Zählen Sie außerdem, wie viele verschiedene Städte pro Land bewohnt sind und sortieren Sie nach der Städteanzahl absteigend.

```
SELECT Land, COUNT(DISTINCT Stadt)
FROM Nutzer
GROUP BY Land
ORDER BY COUNT(DISTINCT Stadt) DESC
```

   Größe der Ergebnismenge: 18

2. Geben Sie an, wie viele Nutzer pro Stadt existieren. Dabei sollen nur Städte berücksichtigt werden, deren Namen keine Umlaute (ä, ö, ü) enthalten. Begrenzen Sie die Ausgabe auf 5 Einträge und sortieren Sie sie absteigend nach der Studentenanzahl.

```
SELECT Stadt, COUNT(*)
FROM Nutzer
WHERE NOT (Stadt ILIKE '%ä%' OR Stadt ILIKE '%ö%' OR
          Stadt ILIKE '%ü%')
GROUP BY Stadt
ORDER BY COUNT(*) DESC
LIMIT 5
```

   Diese Aufgabe kann über zahlreiche Wege gelöst werden, wie beispielsweise über reguläre Ausdrücke.

   Größe der Ergebnismenge: 5

3. Geben Sie den maximalen Aktionsrabatt an, der in jeder Kurs-Kategorie angeboten wird. Kurs-Kategorien, die Aktionen mit einer Aktions-ID unter 6 beinhalten, sollen nicht berücksichtigt werden. Sortieren Sie die Ergebnismenge aufsteigend nach dem durchschnittlichen Rabatt.

```
SELECT Kategorie, MAX(Rabatt)
FROM Aktion
GROUP BY Kategorie
HAVING MIN(Aktions_ID) > 5
ORDER BY AVG(Rabatt) ASC
```

Größe der Ergebnismenge: 5

4. Gruppieren Sie die Kurse nach der Bewertung. Hierbei sollten höchstens 5 Gruppen entstehen. Kurse der Kategorie »Musik« sollen verworfen werden. Geben Sie die Bewertung als »BEW« und den maximalen Preis als »MAX_Preis« aller Gruppen an, die eine summierte Teilnehmerzahl von über 150 haben. Sortieren Sie die Ausgabe aufsteigend nach dem maximalen Preis.

```
SELECT    ROUND(Bewertung, 0) AS BEW,
          MAX(Preis) AS MAX_Preis
FROM Kurs
WHERE Kategorie != 'Musik'
GROUP BY ROUND(Bewertung, 0)
HAVING SUM(Teilnehmerzahl) > 150
ORDER BY MAX(Preis) ASC
```

Größe der Ergebnismenge: 2

Kapitel 6

Die Lösungen bestehen pro Aufgabe aus dem SQL-Statement, der Größe der Ergebnismenge sowie je nach Komplexität der Aufgabe einer Erklärung. Beachten Sie, dass es sich hier nur um Musterlösungen handelt. SQL ist vielfältig, wodurch mehrere Lösungswege möglich sind.

1. Geben Sie die Anzahl aller Studenten pro Land an, die ihre bestellten Kurse noch nicht bezahlt haben.

```
SELECT Land, COUNT(DISTINCT Nutzer.Nutzer_ID)
FROM Nutzer
JOIN Bestellung
ON Nutzer.Nutzer_ID = Bestellung.Nutzer_ID
WHERE Bezahl_Status = False
GROUP BY Land
```

   Da ein Nutzer theoretisch mehr als einen Kurs bestellen und damit nicht bezahlen kann, ist es wichtig, mit `DISTINCT` diese doppelten Nutzer auszusortieren. Nur so kann garantiert werden, dass pro Land die verschiedenen Nutzer gezählt werden.

   Größe der Ergebnismenge: 18

2. Geben Sie den minimalen und maximalen Aktionsrabatt für jede Kurs-Kategorie an, für die es 5 oder mehr Kurse gibt.

```
SELECT Kurs.Kategorie, MIN(Rabatt), MAX(Rabatt)
FROM Kurs
JOIN Aktion
ON  Kurs.Kategorie = Aktion.Kategorie OR
                     Aktion.Kategorie = 'Alle'
GROUP BY Kurs.Kategorie
HAVING COUNT(DISTINCT Kurs_ID) >= 5
```

   Größe der Ergebnismenge: 3

3. Geben Sie den Vor- und Nachnamen aller Dozenten an, deren Kurs nicht durch einen Gutschein rabattiert werden kann. (**Tipp:** `WHERE x IS NULL` prüft, ob x NULL ist.)

```
SELECT DISTINCT(Vorname, Nachname)
FROM Nutzer
JOIN Kurs
ON Nutzer_ID = Dozent_ID
LEFT JOIN Gutschein
ON Kurs.Kurs_ID = Gutschein.Kurs_ID
WHERE Gutschein_ID IS NULL
```

Größe der Ergebnismenge: 6

Kapitel 7

Die Lösungen bestehen pro Aufgabe aus dem SQL-Statement, der Größe der Ergebnismenge sowie je nach Komplexität der Aufgabe einer Erklärung. Beachten Sie, dass es sich hier nur um Musterlösungen handelt. SQL ist vielfältig, wodurch mehrere Lösungswege möglich sind.

1. Geben Sie alle Kurse an, die keine 5-Sterne-Bewertungen erhalten haben.

```
SELECT *
FROM Kurs
WHERE Kurs_ID NOT IN     (SELECT Kurs_ID
                          FROM Kommentar
                          WHERE Bewertung = 5)
```

   Größe der Ergebnismenge: 2

2. Geben Sie alle Nutzer aus, die den Kurs mit der ID 10 kommentiert haben. Die Nutzer sollen dabei im folgenden Format ausgegeben werden: »Vorname + die ersten drei Buchstaben des Nachnamens in Großschreibung + Punkt«.

   Beispiel: Für den Nutzer Max Mustermann soll eine Spalte mit »Max MUS.« ausgegeben werden.

```
SELECT    CONCAT(Vorname, ' ',
          UPPER(LEFT(Nachname, 3)), '.')
FROM Nutzer
WHERE Nutzer_ID IN  (SELECT Nutzer_ID
                     FROM Kommentar
                     WHERE Kurs_ID = 10)
```

   Größe der Ergebnismenge: 21

3. Geben Sie alle Kursnamen, die aufgerundeten Preise und die abgerundeten Bewertungen von allen Kursen an, die mit Gutscheinen rabattiert werden, deren Namen länger als 15 Zeichen sind.

```
SELECT Kurs_Name, CEIL(Preis), FLOOR(Bewertung)
FROM Kurs
WHERE Kurs_ID IN
          (SELECT Kurs_ID
          FROM Gutschein
          WHERE LENGTH(Gutscheinname) > 15)
```

Größe der Ergebnismenge: 7

4. Finden Sie heraus, ob die Teilnehmerzahl der Kurse mit den getätigten Bestellungen übereinstimmt. Gibt es Kurse, die seltener oder öfter bestellt wurden, als in der Teilnehmerzahl angegeben ist?

```
SELECT    K.Kurs_ID,
          K.Teilnehmerzahl = B.Anzahl AS Gleichheit
FROM Kurs AS K
JOIN
          (SELECT Kurs_ID, COUNT(*) AS Anzahl
          FROM Bestellung
          GROUP BY Kurs_ID) AS B
ON K.Kurs_ID = B.Kurs_ID
```

Größe der Ergebnismenge: 25

Erklärung: Neben der Teilnehmerzahl eines Kurses, für die eine eigene Spalte existiert, müssen die Bestellungen eines Kurses erst durch den GROUP BY-Befehl zusammengefasst werden. Durch `SELECT Kurs_ID, COUNT(*) AS Anzahl` kann somit jeder Kurs-ID ihre Anzahl an Bestellungen zugeordnet werden. Um diese Anzahl ins Verhältnis mit der angegebenen Teilnehmerzahl zu bringen, wird mit einem JOIN die Kurs-ID der Kurs-Relation mit der der Unterabfrage verbunden. Im SELECT-Befehl wird anschließend überprüft, ob Teilnehmerzahl und Anzahl der Bestellungen übereinstimmen.

Kapitel 8

Die Lösungen bestehen pro Aufgabe aus dem SQL-Statement, der Größe der Ergebnismenge sowie je nach Komplexität der Aufgabe einer Erklärung. Beachten Sie, dass es sich hier nur um Musterlösungen handelt. SQL ist vielfältig, wodurch mehrere Lösungswege möglich sind.

1. Geben Sie alle einzigartigen Nutzer-IDs von Nutzern an, die selbst Kurse veröffentlichen oder mindestens einen Kommentar verfasst haben, der genau 15 Zeichen lang ist.

```
SELECT Nutzer_ID
FROM Nutzer
WHERE Status = 'Dozent' OR Status = 'Student und Dozent'
UNION

SELECT Nutzer_ID
FROM Kommentar
WHERE LENGTH(Kommentar) = 15
```

   Größe der Ergebnismenge: 14

2. Geben Sie alle Nutzer-IDs von Nutzern an, die aus Deutschland kommen und keinen der bestellten Kurse bewertet bzw. kommentiert haben.

   PostgreSQL:

```
SELECT Nutzer_ID
FROM Nutzer
WHERE Land = 'Deutschland'
EXCEPT
SELECT Nutzer_ID
FROM Kommentar
```

   MySQL:

```
SELECT Nutzer_ID
FROM Nutzer
WHERE Land = 'Deutschland'
      AND Nutzer_ID NOT IN (SELECT Nutzer_ID
FROM Kommentar)
```

   Größe der Ergebnismenge: 4

3. Geben Sie die IDs aller Nutzer-Kurs-Paare an, bei denen der Nutzer den Kurs bereits bezahlt und dem Kurs eine 2-Sterne-Bewertung gegeben hat.

   PostgreSQL:

```
SELECT Nutzer_ID, Kurs_ID
FROM Bestellung
WHERE Bezahl_Status = True
INTERSECT
SELECT Nutzer_ID, Kurs_ID
FROM Kommentar
WHERE Bewertung = 2
```

   MySQL:

```
SELECT B.Nutzer_ID, B.Kurs_ID
FROM Bestellung AS B
JOIN Kommentar AS K
ON B.Nutzer_ID = K.Nutzer_ID AND B.Kurs_ID = K.Kurs_ID
WHERE B.Bezahl_Status = True AND K.Bewertung = 2
```

   Größe der Ergebnismenge: 29

4. Geben Sie alle Kurskategorien an, bei denen die Kursbewertung kleiner als die durchschnittliche Kursbewertung ist und die durch keine Aktionen mit einem Rabatt über 70 Prozent rabattiert werden können.

   PostgreSQL:

```
SELECT DISTINCT(Kategorie)
FROM Kurs
WHERE Bewertung < (SELECT AVG(Bewertung) FROM Kurs)
EXCEPT
SELECT Kategorie
FROM Aktion
WHERE Rabatt > 70
```

MySQL:

```
SELECT DISTINCT(Kategorie)
FROM Kurs
WHERE Bewertung < (SELECT AVG(Bewertung) FROM Kurs)
AND Kategorie NOT IN
(SELECT Kategorie
FROM Aktion
WHERE Rabatt > 70)
```

Größe der Ergebnismenge: 4

Kapitel 9

Die Lösungen bestehen pro Aufgabe aus dem SQL-Statement sowie je nach Komplexität der Aufgabe einer Erklärung. Beachten Sie, dass es sich hier nur um Musterlösungen handelt. SQL ist vielfältig, wodurch mehrere Lösungswege möglich sind.

1. Erstellen Sie die Nutzer-Relation, die aus folgenden Attributen besteht: Nutzer_ID, Vorname, Nachname, Land, Stadt, Geburtstag und Status.

```
CREATE TABLE Nutzer(
     Nutzer_ID INT,
     Vorname VARCHAR(20),
     Nachname VARCHAR(20),
     Land VARCHAR(30),
     Stadt VARCHAR(30),
     Geburtstag DATE,
     Status VARCHAR(20))
```

   Bei der Wahl der Datentypen gibt es mehrere Möglichkeiten, die Nutzer-Relation zu definieren. Beispielsweise können die Grenzen beim VARCHAR-Datentyp anders gewählt werden.

2. Fügen Sie in die neue Relation folgende Nutzer ein:
   - den Studenten Franz Scholler aus Deutschland, Berlin, zu dem kein Geburtsdatum bekannt ist
   - die Studentin Emma Zimmermann aus Deutschland, Köln, die am 05.09.1999 geboren ist
   - die Dozentin Mia Hernandez aus Spanien, Málaga, die am 05.04.1992 geboren ist

```
INSERT INTO Nutzer
VALUES    (1, 'Franz', 'Scholler', 'Deutschland',
          'Berlin', NULL, 'Student'),

          (2, 'Emma', 'Zimmermann', 'Deutschland',
          'Köln', '1999-09-05', 'Studentin'),

          (3, 'Mia', 'Hernandez', 'Spanien',
          'Málaga', '1992-04-05', 'Dozentin');
```

In diesem SQL-Statement werden alle Werte über einen INSERT INTO-Befehl eingetragen. Durch »NULL« ist dies möglich, auch wenn das Geburtsdatum des Studenten Franz Scholler unbekannt ist.

Kapitel 10

Die Lösungen bestehen pro Aufgabe aus dem SQL-Statement sowie je nach Komplexität der Aufgabe einer Erklärung. Beachten Sie, dass es sich hier nur um Musterlösungen handelt. SQL ist vielfältig, wodurch mehrere Lösungswege möglich sind.

Erstellen Sie die Gutschein-Relation, die aus folgenden Attributen besteht: Gutschein_ID, Kurs_ID, Gutscheinname, Rabatt, Ablaufdatum und Code. Beachten Sie dabei folgende Kriterien:

1. Die Gutschein_ID ist der Primärschlüssel der Relation und soll automatisch hochgezählt werden.
2. Die Kurs_ID ist ein Fremdschlüssel und verweist auf die Kurs-Relation (Hinweis: Sollten Sie die Kurs-Relation noch nicht erstellt haben, muss dies vor diesem Schritt geschehen).
3. Der Gutscheinname muss eindeutig sein.
4. Der Rabatt darf weder NULL noch außerhalb der Grenzen von 0 bis 100 liegen.
5. Das Ablaufdatum darf nicht NULL sein und nicht vor 2024 liegen.
6. Der Code soll, wenn er nicht angegeben wird, automatisch auf `_GIFTCODE_` gesetzt werden.

```
CREATE TABLE Gutschein (
     Gutschein_ID SERIAL PRIMARY KEY,
     Kurs_ID INT,
     Gutscheinname VARCHAR(25) UNIQUE,
     Rabatt INT
          NOT NULL
          CHECK (Rabatt BETWEEN 0 AND 100),
     Ablaufdatum DATE
          NOT NULL
          CHECK(Ablaufdatum >= '2024-01-01'),
     Code VARCHAR(10) DEFAULT '_GIFTCODE_',
     FOREIGN KEY (Kurs_ID) REFERENCES Kurs(Kurs_ID))
```

Gegebenenfalls müssen Sie die Kurs-Relation vorher erstellen, falls dies noch nicht geschehen ist. Welche Einschränkungen Sie dieser Relation zuweisen, ist dabei irrelevant. Lediglich der Primärschlüssel ist von Bedeutung:

```
CREATE TABLE Kurs(
     Kurs_ID INT PRIMARY KEY,
     Kurs_Name VARCHAR(50),
     Beschreibung VARCHAR(100),
     Kategorie VARCHAR(20),
     Sprache VARCHAR(20),
     Dozent_ID INT,
     Preis DECIMAL(5,2),
     Bewertung DECIMAL(3,2),
     Teilnehmerzahl INT)
```

Kapitel 11

1. Erstellen Sie eine Relation namens Unternehmen, die aus den Attributen »Unternehmen_ID«, »Name«, »Adresse« und »Kontakt« besteht. Die ID soll den Primärschlüssel darstellen und der Name mit der NOT NULL-Einschränkung versehen werden.

```
CREATE TABLE Unternehmen (
     Unternehmen_ID INT PRIMARY KEY,
     Name VARCHAR(100) NOT NULL,
     Adresse VARCHAR(200),
     Kontakt VARCHAR(100))
```

2. Erstellen Sie eine weitere Relation namens »Prüfung«, die als Primärschlüssel das Attribut »Prüfung_ID«, als Fremdschlüssel »Unternehmen_ID« und als weitere Attribute eine Punktzahl, Fragen und Antworten enthält.

```
CREATE TABLE Prüfung (
     Prüfung_ID INT PRIMARY KEY,
     Unternehmen_ID INT,
     Punktzahl INT,
     Fragen TEXT,
     Antworten TEXT,
     FOREIGN KEY (Unternehmen_ID)
     REFERENCES Unternehmen(Unternehmen_ID))
```

3. Fügen Sie das Unternehmen »SQL-Lösungen GmbH« mit der Adresse »Mommsenstraße 13, 01069 Dresden«, der ID 1 und dem Kontakt »Benedikt Kremser« zur Unternehmen-Relation hinzu.

```
INSERT INTO Unternehmen
VALUES    (1, 'SQL-Lösungen GmbH',
          'Mommsenstraße 13, 01069 Dresden',
          'Benedikt Kremser')
```

4. Fügen Sie die erste Prüfung mit der ID 100 des Unternehmens »SQL-Lösungen GmbH« hinzu und belassen Sie alle Attribute bis auf den Primär- und Fremdschlüssel auf NULL.

```
INSERT INTO Prüfung
VALUES (100, 1, NULL, NULL, NULL)
```

5. Ändern Sie den Namen der Prüfung-Relation zu »Aufnahmeprüfung« und passen Sie außerdem den Namen des Attributs »Kontakt« zu »Ansprechpartner« an.

```
ALTER TABLE Prüfung
RENAME TO Aufnahmeprüfung;

ALTER TABLE Unternehmen
RENAME COLUMN Kontakt TO Ansprechpartner;
```

6. Erweitern Sie die Aufnahmeprüfung-Relation um das Attribut »Zeitlimit« und passen Sie die Fremdschlüsselbeziehung so an, dass bei einem Löschvorgang eines Unternehmens auch alle abhängigen Prüfungen gelöscht werden.

```
ALTER TABLE Aufnahmeprüfung
ADD COLUMN Zeitlimit INT
```

Um die Fremdschlüsselbeziehung anzupassen, ist deren Name erforderlich:

```
SELECT CONSTRAINT_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'aufnahmeprüfung'
```

Mit dem passenden Namen kann nun die Einschränkung gelöscht und angepasst wieder eingefügt werden:

```
ALTER TABLE Aufnahmeprüfung
DROP CONSTRAINT prüfung_unternehmen_id_fkey;

ALTER TABLE Aufnahmeprüfung
ADD CONSTRAINT prüfung_unternehmen_id_fkey
FOREIGN KEY (Unternehmen_ID)
REFERENCES Unternehmen(Unternehmen_ID)
ON DELETE CASCADE;
```

Der Name des Fremdschlüssels kann bei Ihnen variieren. Beispielsweise kann dieser in MySQL auch als »aufnahmeprüfung_ibfk_1« bezeichnet werden.

7. Erweitern Sie die Unternehmen-Relation um den Standardwert »nicht angegeben«, der automatisch eingetragen werden soll, wenn die Adresse des Unternehmens nicht angegeben wurde.

   PostgreSQL:

```
ALTER TABLE Unternehmen
ALTER COLUMN Adresse SET DEFAULT 'nicht angegeben'
```

   MySQL:

```
ALTER TABLE Unternehmen
MODIFY COLUMN Adresse VARCHAR(200)
DEFAULT 'nicht angegeben'
```

8. Passen Sie die Prüfung mit der ID 100 so an, dass die Punktzahl auf 0 gesetzt wird und das Zeitlimit auf 60 Minuten.

```
UPDATE Aufnahmeprüfung
SET Punktzahl = 0, Zeitlimit = 60
WHERE Prüfung_ID = 100
```

9. Löschen Sie das Unternehmen mit der ID 1.

```
DELETE FROM Unternehmen
WHERE Unternehmen_ID = 1
```

10. Löschen Sie beide in dieser Übung erstellten Relationen.

```
DROP TABLE Aufnahmeprüfung;

DROP TABLE Unternehmen;
```

Kapitel 12

1. Erstellen Sie drei Benutzer mit den Namen »Aline«, »Sven« und »Lea«.

   PostgreSQL:

```
CREATE USER Aline WITH PASSWORD 'password';
CREATE USER Sven WITH PASSWORD 'password';
CREATE USER Lea WITH PASSWORD 'password';
```

   MySQL:

```
CREATE USER Aline IDENTIFIED BY 'password';
CREATE USER Sven IDENTIFIED BY 'password';
CREATE USER Lea IDENTIFIED BY 'password';
```

2. Erstellen Sie eine Rolle »Marketing«, die Lese- und Schreibzugriff auf alle Aktionen und Gutscheine der Onlinekurs-Plattform erhält.

```
CREATE ROLE Marketing;
```

   PostgreSQL:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON TABLE public.aktion
TO Marketing;

GRANT SELECT, INSERT, UPDATE, DELETE
ON TABLE public.gutschein
TO Marketing;
```

   MySQL:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON EigeneDatenbank.aktion
TO Marketing;

GRANT SELECT, INSERT, UPDATE, DELETE
ON EigeneDatenbank.gutschein
TO Marketing;
```

3. Ordnen Sie die drei Benutzer aus der ersten Aufgabe dieser Rolle zu.

```
GRANT Marketing TO Aline;
GRANT Marketing TO Sven;
GRANT Marketing TO Lea;
```

Kapitel 13

PostgreSQL

```
CREATE OR REPLACE FUNCTION Update_Bewertung()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
     UPDATE Kurs
     SET Bewertung =      (SELECT AVG(Bewertung)
                          FROM Kommentar
                          WHERE Kurs_ID = NEW.Kurs_ID
                          GROUP BY Kurs_ID)
     WHERE Kurs_ID = NEW.Kurs_ID;
     RETURN NEW;
END;
$$

CREATE TRIGGER Bewertung_Trigger
AFTER INSERT ON Kommentar
FOR EACH ROW
EXECUTE PROCEDURE Update_Bewertung()
```

Obwohl Trigger-Funktionen sehr komplex erscheinen, ist für die meisten Anwendungsfälle, einschließlich dieser Übung, nur der Inhalt des BEGIN-END-Blocks entscheidend. Die restlichen Teile können Sie deshalb aus den vorherigen Abschnitten übernehmen.

Um die Bewertung zu aktualisieren, wird der UPDATE-Befehl auf der Kurs-Relation angewendet. Um im SET-Teil die neue Bewertung zu erhalten, wird in einer Unterabfrage die neue Durchschnittsbewertung des Kurses berechnet. Hierfür werden alle Kommentare nach der Kurs-ID gruppiert, wobei nur die Kurs-ID relevant ist, bei der soeben ein neuer Kommentar hinzugefügt wurde (NEW). Anschließend wird in der WHERE-Klausel des UPDATE-Befehls die Kurs-ID auf `NEW.Kurs_ID` gesetzt, um nur den zu aktualisierenden Kurs anzupassen, und schließlich wird mit RETURN NEW die geforderte RETURN-Klausel angegeben (diese ist hier nicht weiter relevant).

In der folgenden Trigger-Definition gibt es die üblichen drei Teile: Event, Timing und Reaktion. Nur wenn in die Kommentar-Relation ein neues Tu-

pel eingefügt wird, soll anschließend die definierte Trigger-Funktion aktiviert werden.

MySQL

```
DELIMITER //
CREATE TRIGGER Bewertung_Trigger
AFTER INSERT ON Kommentar
FOR EACH ROW
BEGIN
     UPDATE Kurs
     SET Bewertung =     (SELECT AVG(Bewertung)
                          FROM Kommentar
                          WHERE Kurs_ID = NEW.Kurs_ID
                          GROUP BY Kurs_ID)
     WHERE Kurs_ID = NEW.Kurs_ID;
END //
DELIMITER ;
```

Wie in den vorherigen Abschnitten gilt es, den Trigger mit den drei Komponenten – Event, Timing und Reaktion – zu definieren. Dabei ist das übliche Gerüst mit dem DELIMITER-Befehl zu wählen.

Erst nach dem Einfügen eines neuen Tupels in die Kommentar-Relation soll die Reaktion, also das Aktualisieren der entsprechenden Kurs-Bewertung, erfolgen. Diese Reaktion wird wie gewohnt in den BEGIN-END-Block eingebettet.

Mit einem UPDATE-Befehl wird hier die Kurs-Bewertung angepasst, wobei nur der betroffene Kurs aktualisiert werden soll (`WHERE Kurs_ID = NEW.Kurs_ID`). Um die neue Durchschnittsbewertung herauszufinden, werden in einer Unterabfrage alle Kommentare nach der Kurs-ID gruppiert. Für die Gruppe des zu aktualisierenden Kurses wird anschließend mit `AVG(Bewertung)` die Durchschnittsbewertung berechnet.