FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Department Informatik
Lehrstuhl für Informatik 6
(Datenmanagement)

Konzeptionelle Modellierung
Sommersemester 2017

Übung 07/08: Structured Query Language

Allgemeine Hinweise zur Bearbeitung

Die nachfolgende Aufgabensammlung bieten Ihnen die Möglichkeit, die in der Vorlesung behandelten Sprachkonstrukte der Structured Query Language praktisch auszuprobieren.

Aufgrund der großen Anzahl der bereitgestellten Aufgaben können **nur ausgewählte Aufgaben in den Präsenzübungen** behandelt werden. Deren Aufgabenstellungen sind entsprechend markiert.

Wir erwarten von Ihnen jedoch, dass Sie **alle der nachfolgenden Aufgaben eigenständig bearbeiten**. Für alle Aufgaben, die nicht in den Präsenzübungen behandelt werden, stellen wir Ihnen wie gewohnt Lösungsvorschläge in StudOn bereit. Dabei handelt es sich ausdrücklich um Vorschläge, da alle Aufgaben auf verschiedene Arten gelöst werden können.

Technische Arbeitsumgebung

Die nachfolgenden Aufgaben beziehen sich auf das Datenbankmanagementsystem MySQL. Auf relevante Abweichungen vom SQL-Standard wird direkt in den Aufgabenstellungen gesondert hingewiesen.

Für die Installation unter Windows empfehlen wir Ihnen den **MySQL Installer**. Dieser enthält neben dem MySQL-Server auch die grafische MySQL-Workbench. In StudOn stellen wir Ihnen einen **Screencast** bereit, der die Installation und grundlegende Einrichtung demonstriert. MySQL ist ebenso für Linux verfügbar. Viele gängige Distributionen stellen dazu entsprechende Pakete bereit. Bitte konsultieren Sie die Dokumentation Ihrer verwendeten Distribution für weitere Details. Neben der Installation des MySQL-Servers empfehlen wir Ihnen, die MySQL Workbench als grafisches Werkzeug für den Umgang mit dem DBMS zu installieren. Eine Installation von MySQL ist auch unter OSX möglich. Bitte lesen Sie dazu die offizielle Installationsanleitung.

Wichtige Links:

- Downloadseite: `http://dev.mysql.com/downloads/mysql/`
- Installationsanleitung: `http://dev.mysql.com/doc/refman/5.7/en/installing.html`
- Dokumentation: `https://dev.mysql.com/doc/refman/5.7/en/`

Support

Nutzen Sie bitte das Forum der FSI Informatik, falls bei der Verwendung von MySQL Fehler auftreten. Geben Sie in Ihrem Posting unbedingt folgende Informationen an: Betriebssystem, verwendetes Client-Tool, ausgeführtes Skript/abgesetzte SQL-Befehle. Nur so kann Ihnen geholfen werden!
ABER: Versuchen Sie Ihre Probleme immer zuerst selbst mit Hilfe der Dokumentation von MySQL zu lösen, so lernen Sie am meisten!

Erläuterungen zur Beispieldatenbank

Die Beispieldatenbank in diesem Buch versucht, eine Versicherungsgesellschaft für Kfz-Versicherungen abzubilden. Das Beispiel ist eine starke Vereinfachung der Realität; zum Beispiel fehlen alle Teile der laufenden Abrechnung der Prämien und Schadensfälle. Die relevanten Begriffe sind für eine bessere Übersichtlichkeit fett gekennzeichnet. Folgender Sachverhalt wird in der Datenbank abgebildet: **Die Versicherungsgesellschaft *UnsereFirma* verwaltet mit dieser Datenbank ihre Kundendaten und die Schadensfälle. Wegen der Schadensfälle müssen auch Daten „fremder“ Kunden und Versicherungen gespeichert werden.**

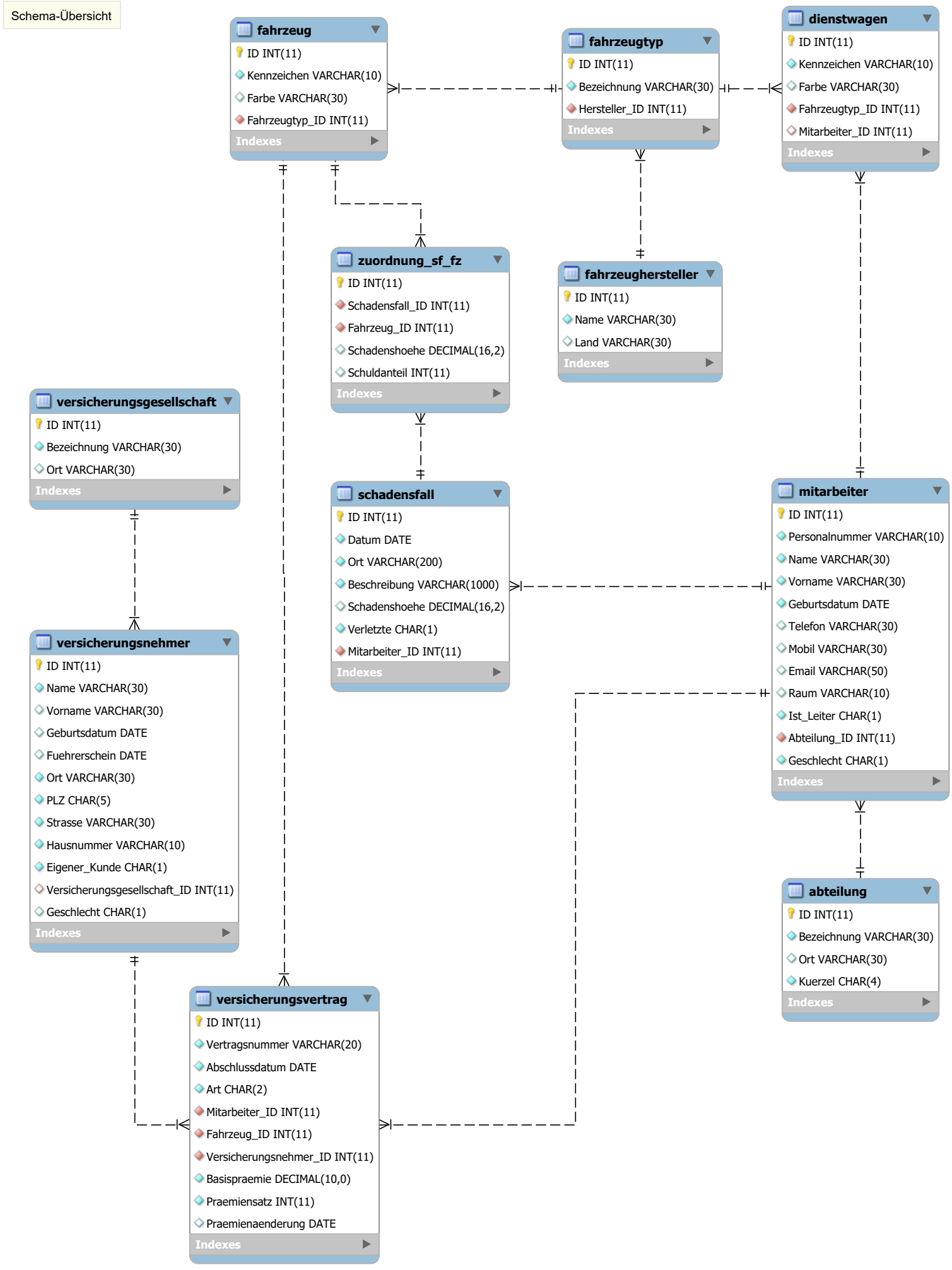
1. Jeder **Versicherungsvertrag** wird mit einem Versicherungsnehmer über genau ein Fahrzeug abgeschlossen. Der Versicherungsvertrag wird durch folgende Eigenschaften gekennzeichnet; zu jeder Eigenschaft gehört eine Spalte:
   1) Vertragsnummer
   2) Datum des Abschlusses, Art des Vertrages; dabei gibt es die Arten Haftpflicht (HP), Haftpflicht mit Teilkasko (TK), Vollkasko (VK).
   3) Verweis auf den Versicherungsnehmer
   4) Verweis auf das Fahrzeug
   5) Verweis auf den Mitarbeiter, der den Vertrag bearbeitet
   6) Basisprämie, Prämiensatz und das Datum der letzten Prämienänderung
2. Der **Versicherungsnehmer** ist gekennzeichnet durch diese Eigenschaften:
   1) Kundennummer
   2) Name und Anschrift: PLZ, Ort, Straße, Hausnummer
   3) bei natürlichen Personen zusätzlich durch Vorname, Geburtsdatum, Geschlecht, Datum des Führerscheins
   4) Kennzeichnung ob es sich um einen eigenen Kunden handelt, oder um den einer fremden Versicherungsgesellschaft (Flag "J/"N")
   5) Verweis auf eine „Fremdversicherung“, wenn ein (fremdes) Fahrzeug an einem Unfall beteiligt ist
3. Das **Fahrzeug** ist gekennzeichnet durch diese Eigenschaften:
   1) polizeiliches Kennzeichen
   2) Farbe
   3) Fahrzeugtyp und damit indirekt auch den Fahrzeughersteller

4. Ein **Mitarbeiter** ist gekennzeichnet durch diese Eigenschaften:
   1) Name, Vorname, Geburtsdatum
   2) Personalnummer
   3) Verweis auf Abteilung, Vermerk, ob es sich um den Leiter der Abteilung handelt (Flag "J/"N")
   4) Kontaktdaten wie Telefon, Mobiltelefon, Email, Raum
   5) Geschlecht
5. Die **Abteilung** ist gekennzeichnet durch diese Eigenschaften:
   1) Nummer
   2) Kurzbezeichnung, Bezeichnung
   3) Ort
6. Zusätzlich gibt es **Dienstwagen**. Dabei handelt es sich um eine Tabelle mit den gleichen Eigenschaften wie bei Fahrzeug und zusätzlich:
   1) Verweis auf den Mitarbeiter, zu dem ein Dienstwagen gehört
   2) denn es gibt auch Dienstwagen, die keinem Mitarbeiter persönlich zugeordnet sind
7. Ein **Schadensfall** ist gekennzeichnet durch diese Eigenschaften:
   1) Datum, Ort und Umstände des Unfalls
   2) Vermerk, ob es Verletzte gab, sowie Höhe des Gesamtschadens
   3) Verweis auf den Mitarbeiter, der den Schadensfall bearbeitet
8. An einem Schadensfall können mehrere Fahrzeuge unterschiedlicher Versicherungen beteiligt sein. (Unfälle mit Radfahrern und Fußgängern werden nicht betrachtet.) Deshalb gibt es eine weitere Tabelle **Zuordnung_SF_FZ**:
   1) Liste aller Schadensfälle und aller beteiligten Fahrzeuge
   2) anteilige Schadenshöhe eines Fahrzeugs an dem betreffenden Unfall
   3) Verweis auf den Mitarbeiter, der den Schadensfall bearbeitet
   4) Anteil am Verschulden

   Über die Verbindung Schadensfall → Fahrzeug → Versicherungsvertrag → Versicherungsnehmer → Versicherungsgesellschaft können alle beteiligten Gesellschaften festgestellt und in die Schadensabwicklung eingebunden werden.

Anschließend finden Sie zusätzlich eine grafische Darstellung aller Tabellen und Attribute. Dabei wird die Ihnen aus der Vorlesung bekannte Krähenfuß-Notation verwendet. Diese Darstellung können Sie sich mit Hilfe der MySQL-Workbench auch selbst generieren (`Database` → `Reverse Engineer`, Kürzel `Strg+R`).

Schema-Übersicht



1 Aufgaben für die siebte Übungswoche

Wir erwarten von Ihnen, dass Sie mindestens die nachfolgenden Aufgaben für die siebte Übungswoche eigenständig lösen.

1.1 Anlegen der Datenbank

Verwenden Sie die in StudOn bereitgestellte Datei **`Versicherung.sql`**, um Ihre Datenbank mit dem benötigten Schema und einigen Testdaten zu füllen. Sie können diese Datei auch verwenden, um die Datenbank bei unbeabsichtigten Änderungen wieder auf einen definierten Zustand zurückzusetzen.

Folgen Sie dem Screencast aus StudOn, um ihre Datenbank unter Verwendung der MySQL-Workbench zu initialisieren.

1.2 Allgemeine Fragen

1. Welche der folgenden Spaltenlisten aus der Beispieldatenbank sind richtig, welche nicht?

   1) `SELECT * FROM Mitarbeiter;`
   2) `SELECT ID, Name FROM Mitarbeiter;`
   3) `SELECT ID, Name FROM Mitarbeiter, Abteilung;`
   4) `SELECT ID, Name, Kuerzel FROM Mitarbeiter, Abteilung;`
   5) `SELECT ab.ID, Name FROM Mitarbeiter, Abteilung ab;`
   6) `SELECT ab.ID, Name, Krz Kuerzel FROM Mitarbeiter, Abteilung ab;`
   7) `SELECT ab.ID, Name, Kuerzel Krz FROM Mitarbeiter, Abteilung ab;`
   8) `SELECT ab.ID, mi.Name, ab.Kuerzel FROM Mitarbeiter mi, Abteilung ab;`

   Richtig sind 1, 2, 5, 7, 8.
   Falsch sind 3, 4 (ID ist mehrdeutig), 6 (Spaltenname und -Alias in falscher Reihenfolge).

2. Welche der folgenden Selektionsbedingungen für die Tabelle `Mitarbeiter` sind richtig, welche nicht? Welche korrekten Bedingungen liefern immer `FALSE` als Ergebnis? Einige der Selektionsbedingungen können Sie aufgrund Ihres Wissens aus der Vorlesung beurteilen, bei anderen können Sie mögliche Fehlermeldungen des DBMS auswerten.

   1) `WHERE Name NOT = 'Meyer';`
   2) `WHERE 1 = 2;`
   3) `WHERE NOT Name LIKE 'M%';`
      (Hinweis: Recherchieren Sie eigenständig die Semantik von `LIKE`)
   4) `WHERE ID BETWEEN 20 AND 10;`
      (Hinweis: Recherchieren Sie eigenständig die Semantik von `BETWEEN .. AND ..`)
   5) `WHERE Name LIKE 'L%' AND LIKE '_a%';`
   6) `WHERE ID IN (1, 3, 'A');`
   7) `WHERE ID IN (1, 3, '5');`

1) Falsch.

2) Richtig, immer FALSE: 1 ist immer ungleich 2.

3) Richtig, weil die gültige Teilbedingung "Name LIKE 'M%'"nur verneint wird. Ebenfalls richtig wäre es, das NOT hinter Name zu schreiben.

4) Richtig, immer FALSE: es gibt keine Zahl „größer/gleich 20“ und gleichzeitig „kleiner/-gleich 10“.

5) Falsch, weil Name in der zweiten Bedingung hinter AND fehlt.

6) Falsch, weil 'A' keine Zahl ist, aber zu ID bei IN eine Liste von Zahlen gehört.

7) Richtig, weil '5' automatisch als Zahl konvertiert wird.

3. [**Besprechung Ihrer Lösung in der Präsenzübung**] Nennen Sie eine gültige Reihenfolge der `INSERT`-Befehle, wenn ein Schadensfall mit drei beteiligten Fahrzeugen aufzunehmen ist. Dabei soll ein Fahrzeug zu einem „eigenen Kunden“ und zwei Fahrzeuge zu „Fremdkunden“ gehören; die eine „fremde“ Versicherungsgesellschaft soll schon gespeichert sein, die andere nicht. Mögliche Reihenfolge:

   1) die „neue“ Versicherungsgesellschaft speichern

   2) deren Kunden speichern

   3) dessen Fahrzeug speichern

   4) dessen Versicherungsvertrag speichern

   5) den Fremdkunden der schon registrierten Versicherungsgesellschaft speichern

   6) dessen Fahrzeug speichern

   7) dessen Versicherungsvertrag speichern

   8) den Schadensfall speichern

   9) den Schadensfall mit den Fahrzeugen verknüpfen:

      i. mit dem Fahrzeug des eigenen Kunden

      ii. mit dem Fahrzeug des einen Fremdkunden

      iii. mit dem Fahrzeug des anderen Fremdkunden

   Für eine gültige Reihenfolge sind alle Fremdschlüsselbeziehungen zwischen den Tabellen beachten! Falls benötigte Informationen wie der zuständige Mitarbeiter oder der Fahrzeugtyp fehlen sollten, müssen diese ebenfalls angelegt werden (`NOT NULL`-Constraints beachten!).

   Hinweis: Neben der oben angegebenen Reihenfolge sind daher auch noch weitere Varianten denkbar. Zur Überprüfung Ihrer eigenen Lösung können Sie eigenständig ein DDL-Skript mit sinnvollen Beispielwerten anlegen.

4. Welche der folgenden Aussagen sind wahr, welche sind falsch?

   1) Um alle Mitarbeiter mit Dienstwagen aufzulisten, benötigt man einen `LEFT OUTER JOIN`.

   2) `LEFT JOIN` ist nur eine Kurzschreibweise für `LEFT OUTER JOIN` und hat keine zusätzliche inhaltliche Bedeutung.

   3) Ein `LEFT JOIN` von zwei Tabellen enthält alle Zeilen, die nach Auswahlbedingung in der linken Tabelle enthalten sind.

4) Ein `RIGHT JOIN` von zwei Tabellen enthält nur noch diejenigen Zeilen, die nach der Verknüpfungsbedingung in der linken Tabelle enthalten sind.

5) Wenn wir bei einer `LEFT JOIN`-Abfrage mit zwei Tabellen die beiden Tabellen vertauschen und stattdessen einen `RIGHT JOIN` verwenden, erhalten wir dieselben Zeilen in der Ergebnismenge.

6) Wir erhalten dabei nicht nur dieselben Zeilen, sondern auch dieselbe Reihenfolge.

Die Aussagen 2, 3, 5 sind richtig, die Aussagen 1, 4, 6 sind falsch.

5. Welche der folgenden Aussagen sind wahr, welche falsch?

1) Die SQL-Schlüsselwörter `JOIN` und `INNER JOIN` sind äquivalent.

2) Eine Tabelle kann mit sich selbst verknüpft werden.

3) SELF JOIN ist nur ein inhaltlicher Begriff, aber kein SQL-Schlüsselwort.

4) Bei einem SELF JOIN sind nur `INNER JOIN`s erlaubt.

5) Eine bestimmte Tabelle darf in einem SELF JOIN nur zweimal verwendet werden.

6) Für einen SELF JOIN können Tabellen-Aliase benutzt werden, aber sie sind nicht überall erforderlich.

7) Ein `CROSS JOIN` ist eine Verknüpfung zweier Tabellen ohne Verknüpfungsbedingung.

8) Bei einem `CROSS JOIN` darf sich die `WHERE`-Klausel nicht auf die (rechte) Tabelle des `JOIN`s beziehen.

Die Aussagen 1, 2, 3, 7 sind wahr, die Aussagen 4, 5, 6, 8 sind falsch.

6. [**Besprechung Ihrer Lösung in der Präsenzübung**] Liefern folgende Queries (bei beliebigem Zustand der Datenbank) unterschiedliche Ergebnisse? Warum bzw. warum nicht?

```
   SELECT *
     FROM Versicherungsvertrag vv
LEFT JOIN Versicherungsnehmer vn
       ON (vv.Versicherungsnehmer_ID = vn.ID
           AND vn.ID > 5)

   SELECT *
     FROM Versicherungsvertrag vv
LEFT JOIN Versicherungsnehmer vn
       ON (vv.Versicherungsnehmer_ID = vn.ID)
    WHERE vn.ID > 5
```

Ja, die obere hat u.U. eine größere Ergebnisrelation, da Tupel, die wegen der zweiten Bedingung keinen Joinpartner finden, dank LEFT JOIN trotzdem in die Ergebnisrelation kommen. In der unteren Query werden diese aussortiert.

1.3 Fehlersuche

1. Was ist an dieser Anfrage falsch?

```
SELECT ID, Abschlussdatum, Art,
       vv.Kennzeichen, Farbe
  FROM Versicherungsvertrag vv, Fahrzeug
 WHERE vv.Fahrzeug_ID = Fahrzeug.ID
   AND Kennzeichen LIKE 'BO%';
```

   Die ID muss mit Tabellennamen oder Alias versehen sein, weil sie in beiden Tabellen enthalten ist. Die Spalte Kennzeichen gehört zur Tabelle Fahrzeug, also ist der Alias vv falsch.

2. **[Besprechung Ihrer Lösung in der Präsenzübung]** Was ist an dieser Anfrage falsch?

```
SELECT ID, Vorname + '␣' + Name AS Kunde, Ort,
       Name AS Sachbearbeiter, Telefon
  FROM Versicherungsvertrag, Versicherungsnehmer, Mitarbeiter
 WHERE ID = 27
   AND Versicherungsvertrag.Versicherungsnehmer_ID
       = Versicherungsnehmer.ID
   AND Versicherungsvertrag.Mitarbeiter_ID = Mitarbeiter.ID;
```

   Die ID muss sowohl in der Spaltenliste als auch in der WHERE-Klausel mit Tabellennamen oder Alias versehen sein, weil sie in allen Tabellen enthalten ist. Gleiches gilt für Name und Vorname, weil diese Angaben in mehreren Tabellen enthalten sind.

3. Zeigen Sie zu jedem Mitarbeiter die Daten seines Dienstwagens (Kennzeichen, Typ, Hersteller) an. Berichtigen Sie dazu den folgenden SELECT-Befehl.

```
SELECT ID, Name, Vorname,
       Kennzeichen, Bezeichnung, Name
  FROM Mitarbeiter mi, Dienstwagen dw,
       Fahrzeugtyp ft, Fahrzeughersteller fh
 WHERE ID = dw.Mitarbeiter_ID
   AND ID = dw.Fahrzeugtyp_ID
   AND ID = ft.Hersteller_ID;
```

```
SELECT mi.ID, mi.Name, mi.Vorname,
       dw.Kennzeichen, ft.Bezeichnung, fh.Name
  FROM Mitarbeiter mi, Dienstwagen dw,
       Fahrzeugtyp ft, Fahrzeughersteller fh
 WHERE mi.ID = dw.Mitarbeiter_ID
   AND ft.ID = dw.Fahrzeugtyp_ID
   AND fh.ID = ft.Hersteller_ID;
```

1.4 Anfrageformulierung

1. Wir benötigen zu allen Abteilungen die ID der Abteilung und die Anzahl der zugehörigen Mitarbeiter pro Abteilung. Sortieren Sie das Ergebnis absteigend nach der Anzahl der Mitarbeiter.

```
  SELECT Abteilung_ID, COUNT(*) as AnzahlMitarbeiter
    FROM Mitarbeiter
GROUP BY Abteilung_ID
ORDER BY AnzahlMitarbeiter DESC;
```

2. Erstellen Sie mit Hilfe eines Mengenquantors eine Liste der IDs aller Mitarbeiter, die denselben Vornamen wie mindestens ein Versicherungsnehmer haben.

```
SELECT ID
  FROM Mitarbeiter
 WHERE Vorname = ANY(SELECT Vorname FROM Versicherungsnehmer);
```

3. **[Besprechung Ihrer Lösung in der Präsenzübung]** Erstellen Sie unter Verwendung eines Existenzquantors eine aufsteigend sortierte, wiederholungsfreie Liste aller Geburtsdaten von Versicherungsnehmern. Berücksichtigen Sie dabei nur Versicherungsnehmer, die aus Orten stammen, in denen mindestens eine Abteilung ansässig ist.

```
  SELECT DISTINCT Geburtsdatum
    FROM Versicherungsnehmer
   WHERE EXISTS (
     SELECT Ort
     FROM Abteilung
     WHERE Abteilung.Ort = Versicherungsnehmer.Ort
   )
ORDER BY Geburtsdatum ASC;
```

   Hierbei handelt es sich um eine korrelierte Unteranfrage, da ein Attribut der äußeren Anfrage (Ort) in der inneren Anfrage verwendet wird.

4. Suchen Sie alle Versicherungsnehmer, die folgenden Bedingungen entsprechen: Der erste Buchstabe des Namens ist nicht bekannt, der zweite ist ein 'r'. Der Vorname enthält ein 'a'. Die Postleitzahl gehört zum Bereich Essen (PLZ 45...).

```
SELECT *
  FROM Versicherungsnehmer
 WHERE Name LIKE '_r%'
   AND Vorname LIKE '%a%'
   AND PLZ LIKE '45___';
```

5. Wir benötigen eine Liste aller Orte, in denen mehr als ein Versicherungsnehmer ansässig ist. Neben dem Städtenamen ist dann die Anzahl dortigen Versicherungsnehmer interessant. Die Sortierung erfolgt nach der Anzahl absteigend.

```
  SELECT Ort, COUNT(*) AS AnzahlKunden
    FROM Versicherungsnehmer
GROUP BY Ort
  HAVING COUNT(*) > 1;
ORDER BY AnzahlKunden DESC;
```

6. Formulieren Sie eine Abfrage zur Tabelle `Versicherungsvertrag`, die nur die wichtigsten Informationen (Vertragsnummer, Abschlussdatum, Art, sowie die Fremdschlüssel auf andere Tabellen) ausgibt. Wie viele Ergebnistupel erhalten Sie?

```
SELECT Vertragsnummer, Abschlussdatum, Art,
       Versicherungsnehmer_ID, Fahrzeug_ID, Mitarbeiter_ID
  FROM Versicherungsvertrag;
```

Es werden 25 Zeilen angezeigt.

7. Erweitern Sie die vorherige Anfrage, sodass anstelle der ID des Versicherungsnehmers dessen Name und Vorname angezeigt werden. Verzichten Sie auf eine `WHERE`-Klausel. Wie viele Ergebnistupel erhalten Sie?

```
SELECT vv.Vertragsnummer, vv.Abschlussdatum, vv.Art,
       vn.Name, vn.Vorname,
       Fahrzeug_ID,
       Mitarbeiter_ID
  FROM Versicherungsvertrag vv, Versicherungsnehmer vn;
```

Es werden 575 Zeilen angezeigt.

8. Erweitern Sie die Anfrage 7, sodass anstelle der ID des Fahrzeugs das Kennzeichen und anstelle der ID des Mitarbeiters dessen Name und Vorname angezeigt werden. Verzichten Sie auf eine `WHERE`-Klausel. Wie viele Ergebnistupel erhalten Sie?

```
SELECT vv.Vertragsnummer, vv.Abschlussdatum, vv.Art,
       vn.Name, vn.Vorname,
       fz.Kennzeichen,
       mi.Name, mi.Vorname
  FROM Versicherungsvertrag vv, Versicherungsnehmer vn,
       Fahrzeug fz, Mitarbeiter mi;
```

Es werden $>$ 400000 Zeilen angezeigt.

9. Erweitern Sie die Anfrage 7, sodass Name und Vorname des Versicherungsnehmers genau zum jeweils passenden Vertrag ausgegeben werden. Wie viele Ergebnistupel erhalten Sie?

```
SELECT vv.Vertragsnummer, vv.Abschlussdatum, vv.Art,
       vn.Name, vn.Vorname,
       Fahrzeug_ID,
       Mitarbeiter_ID
  FROM Versicherungsvertrag vv, Versicherungsnehmer vn
 WHERE vn.ID = vv.Versicherungsnehmer_ID;
```

Es werden 25 Zeilen angezeigt.

10. Erweitern Sie die Anfrage 8, sodass Name und Vorname des Mitarbeiters sowie das Fahrzeug-Kennzeichen genau zum jeweils passenden Vertrag ausgegeben werden. Wie viele Einträge zeigt die Ergebnismenge an?

```
SELECT vv.Vertragsnummer, vv.Abschlussdatum, vv.Art,
       vn.Name, vn.Vorname,
       fz.Kennzeichen,
       mi.Name, mi.Vorname
  FROM Versicherungsvertrag vv, Versicherungsnehmer vn,
       Fahrzeug fz, Mitarbeiter mi
 WHERE vn.ID = vv.Versicherungsnehmer_ID
   AND fz.ID = vv.Fahrzeug_ID
   AND mi.ID = vv.Mitarbeiter_ID;
```

Es werden 25 Zeilen angezeigt.

11. [**Besprechung Ihrer Lösung in der Präsenzübung**] Erweitern Sie die vorherige Anfrage, sodass die ausgewählten Zeilen den folgenden Bedingungen entsprechen: Es geht ausschließlich um eigene Kunden (Eigener_kunde = 'J'). Vollkasko-Verträge sollen immer angezeigt werden, ebenso Fahrzeuge aus dem Kreis Recklinghausen 'RE'. Teilkasko-Verträge sollen angezeigt werden, wenn sie nach '1990-12-31' abgeschlossen wurden. Haftpflicht-Verträge sollen angezeigt werden, wenn sie nach '1985-12-31' abgeschlossen wurden. Wie viele Einträge zeigt die Ergebnismenge an?

```
SELECT vv.Vertragsnummer, vv.Abschlussdatum, vv.Art,
       vn.Name, vn.Vorname,
       fz.Kennzeichen,
       mi.Name, mi.Vorname
  FROM Versicherungsvertrag vv, Versicherungsnehmer vn,
       Fahrzeug fz, Mitarbeiter mi
 WHERE vn.ID = vv.Versicherungsnehmer_ID
   AND fz.ID = vv.Fahrzeug_ID
   AND mi.ID = vv.Mitarbeiter_ID
   AND vn.Eigener_kunde = 'J'
   AND ( ( vv.Art = 'HP' AND vv.Abschlussdatum > '1985-12-31' )
       OR ( vv.Art = 'TK' AND vv.Abschlussdatum > '1990-12-31' )
       OR ( vv.Art = 'VK' )
       OR ( fz.Kennzeichen LIKE 'RE-%' ) );
```

Es werden 18 Zeilen angezeigt.

12. Zeigen Sie alle Mitarbeiter der Abteilungen „Vertrieb“ (Kürzel 'Vert') und „Ausbildung“ (Kürzel 'Ausb') an.
Bestimmen Sie dazu zunächst die IDs der gesuchten Abteilungen in einer Unteranfrage und benutzen Sie das Ergebnis für die eigentliche Anfrage.

```
SELECT *
  FROM Mitarbeiter
 WHERE Abteilung_ID IN (
       SELECT id FROM Abteilung
       WHERE Kuerzel in ('Vert', 'Ausb')
);
```

13. Zu jedem Eintrag der Tabelle `Versicherungsvertrag` sollen Vertragsnummer, Basisprämie und Prämiensatz angegeben sowie die aktuelle Prämie (Basisprämie * Prämiensatz / 100) berechnet werden.

```
SELECT Vertragsnummer,
       Basispraemie,
       Praemiensatz,
       Basispraemie * Praemiensatz / 100 as Aktuell
  FROM Versicherungsvertrag;
```

14. Geben Sie die Gesamtzahl der Versicherungsverträge sowie den Gesamtbetrag aller aktuellen Prämien an.

```
SELECT COUNT(*) as Gesamtzahl,
       SUM(Basispraemie * Praemiensatz / 100) as Praemiensumme
  FROM Versicherungsvertrag;
```

15. [**Besprechung Ihrer Lösung in der Präsenzübung**] Suchen Sie zu jedem Mitarbeiter den Namen und Vornamen des Leiters der zugehörigen Abteilung (Ist_Leiter = 'J'). Die Abteilungsleiter in unserer einfachen Firmenhierarchie haben keinen Vorgesetzten; sie sollen in der Liste deshalb nicht aufgeführt werden.

```
SELECT mi1.Abteilung_ID as Abt, mi1.Name, mi1.Vorname,
       mi2.Name as LtrName, mi2.Vorname as LtrVorn
  FROM Mitarbeiter mi1
       JOIN Abteilung ab on mi1.Abteilung_ID = ab.ID
       JOIN Mitarbeiter mi2 on mi2.Abteilung_ID = ab.ID
 WHERE mi2.Ist_Leiter = 'J'
   AND mi1.Ist_Leiter = 'N'
```

16. [**Besprechung Ihrer Lösung in der Präsenzübung**] Suchen Sie Einträge in der Tabelle `Versicherungsnehmer`, bei denen Name, Vorname, PLZ, Strasse übereinstimmen. Jeweils zwei dieser Adressen sollen mit ihrer ID und den übereinstimmenden Angaben aufgeführt werden.
Hinweis: Benutzen Sie einen JOIN, der sich nicht auf übereinstimmende IDs bezieht.

```
SELECT a.Name, a.Vorname, a.PLZ, a.Strasse, a.ID, b.ID
  FROM Versicherungsnehmer a
  JOIN Versicherungsnehmer b
    ON a.Name = b.Name
   AND a.Vorname = b.Vorname
   AND a.PLZ = b.PLZ
   AND a.Strasse = b.Strasse
 WHERE a.ID < b.ID;
```

Als Alternative zu `<` sind auch `>`, `WHERE NOT a.ID <= b.ID` und `WHERE NOT a.ID >= b.ID` möglich.
Die Varianten `<>`, `!=` sind streng genommen nicht korrekt, da sie jedes Duplikat zweifach auflisten.

1.5 Datenmodifikation

1. Fügen Sie sich selbst als Mitarbeiter mit der ID 29 in den vorhandenen Datenbestand ein.

```
INSERT INTO Mitarbeiter
(ID, Personalnummer, Name, Vorname, Geburtsdatum
, Telefon, Mobil, Email, Raum, Ist_Leiter
, Abteilung_ID, Geschlecht)
VALUES ('29', '1337', 'Mustermann', 'Max', '1994-10-02'
, '091311337', '01701337', 'max@muster.org'
, '100', 'J', '12', 'M');
```

2. Entfernen Sie Ihren soeben angelegten Eintrag (und nur diesen) wieder aus der Datenbank.

```
DELETE FROM Mitarbeiter
WHERE ID = 29;
```

3. Überlegen Sie sich, was passiert, wenn Sie die folgende Anfrage ausführen:

```
DELETE FROM Mitarbeiter;
```

Nichts (bzw. Fehlermeldung), da andere Tabellen noch auf die Tabelle `Mitarbeiter` verweisen. Falls keine Verweise vorhanden wären: Die Tabelle **Mitarbeiter** wird vollständig geleert, aber jedoch nicht gelöscht!

4. [**Besprechung Ihrer Lösung in der Präsenzübung**] Von der Deutschen Post AG wird eine Tabelle `plz_aenderung`(ID, PLZalt, Ortalt, PLZneu, Ortneu) geliefert:

```
CREATE TABLE plz_aenderung (
  ID INT NOT NULL AUTO_INCREMENT,
  PLZalt INT NOT NULL,
  Ortalt VARCHAR(64) NOT NULL,
  PLZNeu INT NOT NULL,
  Ortneu VARCHAR(64) NOT NULL,
  PRIMARY KEY (ID));

INSERT INTO plz_aenderung(ID, PLZalt, Ortalt, PLZNeu, Ortneu)
           VALUES ('1', '45658', 'Recklinghausen', '45659'
                  , 'Recklinghausen');
INSERT INTO plz_aenderung(ID, PLZalt, Ortalt, PLZNeu, Ortneu)
           VALUES ('2', '45721', 'Hamm-Bossendorf', '45721'
                  , 'Haltern-Hamm');
INSERT INTO plz_aenderung(ID, PLZalt, Ortalt, PLZNeu, Ortneu)
           VALUES ('3', '45772', 'Marl', '45770', 'Marl');
INSERT INTO plz_aenderung(ID, PLZalt, Ortalt, PLZNeu, Ortneu)
           VALUES ('4', '45701', 'Herten', '45699', 'Herten');
```

Ändern Sie die Tabelle `Versicherungsnehmer` so, dass bei Adressen, bei denen `PLZ/Ort` mit `PLZalt/Ortalt` übereinstimmen, diese Angaben durch `PLZneu/Ortneu` geändert werden. Beschränken Sie sich auf die Änderung mit der `ID` 3.

Zusatzaufgabe zur Vertiefung (über den Stoff der Lehrveranstaltung hinausgehend): Überlegen Sie sich darüber hinaus, welches zusätzliche Sprachkonstrukt Sie bräuchten, um alle Änderungen auf einmal durchzuführen.

```
UPDATE Versicherungsnehmer
   SET PLZ = ( SELECT PLZneu FROM PLZ_Aenderung WHERE ID = 3 ),
       Ort = ( SELECT Ortneu FROM PLZ_Aenderung WHERE ID = 3 )
 WHERE PLZ = ( SELECT PLZalt FROM PLZ_Aenderung WHERE ID = 3 )
   AND Ort = ( SELECT Ortalt FROM PLZ_Aenderung WHERE ID = 3 );
```

Ohne das Selektionsprädikat `WHERE ID = 3` liefern die Unteranfragen je nach dem konkreten Inhalt der Tabelle `plz_aenderung` mehr als ein Ergebnis - die Wertzuweiung/der Vergleich mittels `=` scheitert daher.

Lösungsmöglichkeit bei MySQL (siehe `http://dev.mysql.com/doc/refman/5.7/en/update.html`):

```
UPDATE versicherungsnehmer vn, plz_aenderung c
   SET vn.PLZ = c.PLZneu, vn.Ort = c.Ortneu
 WHERE vn.PLZ = c.PLZalt
   AND vn.Ort = c.Ortalt;
```

Die Multiple-Table-Syntax von MySQL ist jedoch nicht standardisiert, daher ist obige Query auf anderen Systemen unter Umständen nicht ausführbar.

Laut SQL-Standard sind aber korrelierte Unteranfragen in der SET-Klausuel zulässig. Für unsere Beispielextension aus der Tabelle `PLZ_Aenderung` auch mit dem nachfolgenden

Statement durchgeführt werden.
Hinweis: Sobald die Beispielextension sich ändert (z.B. mehrfache Änderungen eines Eintrags vorkommen oder PLZ und Ort für einen Eintrag gleichzeitig geändert werden), funktioniert das nachfolgende Statement unter MySQL nicht mehr korrekt!

```
UPDATE Versicherungsnehmer
   SET PLZ = ( SELECT PLZneu
                FROM PLZ_Aenderung
               WHERE Versicherungsnehmer.PLZ = plz_aenderung.PLZalt),
       Ort = ( SELECT Ortneu
                FROM PLZ_Aenderung
               WHERE Versicherungsnehmer.Ort = plz_aenderung.Ortalt )
 WHERE EXISTS ( SELECT *
                FROM PLZ_Aenderung
               WHERE Versicherungsnehmer.Ort = plz_aenderung.Ortalt
                 AND versicherungsnehmer.PLZ = plz_aenderung.PLZalt
             );
```

Die Klausel `WHERE EXISTS` stellt dabei sicher, dass keine `NOT NULL`-Constraints verletzt werden.

Um beliebige Änderungen auf einmal durchzuführen ist es erforderlich, über alle Tupel zu iterieren. SQL stellt dazu auch prozedurale Sprachkonstrukte zur Verfügung. Diese fortgeschrittenen Sprachkonstrukte sind nicht Teil unserer Lehrveranstaltung, aber in der Praxis weit verbreitet. Einen ersten Einstiegspunkt in diese Thematik finden Sie bei Interesse hier: https://de.wikibooks.org/wiki/Einführung_in_SQL:_Programmierung. Aufgrund systemspezifischer Implementierungsdetails sollten Sie bei prozeduraler Programmierung in SQL auch immer die Dokumentation des von Ihnen verwendeten DBMS konsultieren.

Lesson learned: Viele Aspekte von SQL sind sehr spezifisch für das von Ihnen in der Praxis jeweils eingesetzte DBMS. In der Lehrveranstaltung *Konzeptionelle Modellierung* können Sie jedoch davon ausgehen, dass wir uns immer auf ein standardkonformes DBMS beziehen.

2 Aufgaben für die achte Übungswoche

Wir erwarten von Ihnen, dass Sie die nachfolgenden Aufgaben spätestens für die achte Übungswoche eigenständig lösen. Selbstverständlich können Sie die Aufgaben auch schon vorher bearbeiten.

2.1 NULL-Semantik

1. Der Wert `NULL` steht für einen nicht-definierten Wert. Vergleiche mit `NULL`-Werten liefern daher immer `FALSE` als Ergebnis. Überzeugen Sie sich davon, indem Sie die folgenden beiden Anfragen ausführen:

   ```
   SELECT *
   FROM Versicherungsnehmer
   WHERE Geburtsdatum = NULL;

   SELECT *
   FROM Versicherungsnehmer
   WHERE Geburtsdatum <> NULL;
   ```

2. Verwenden Sie aufgrund Ihrer Erfahrungen aus der vorherigen Aufgabe für den korrekten Vergleich mit `NULL`-Werten die Operatoren `IS NULL` oder `IS NOT NULL`. Wieviele Ergebnistupel erhalten Sie jeweils, wenn Sie die beiden Anfragen aus der vorherigen Aufgabe entsprechend modifizieren?

   Die erste Anfrage liefert ein Ergebnistupel, die zweite 22 Ergebnistupel.

3. [**Besprechung Ihrer Lösung in der Präsenzübung**] Führen Sie die beiden folgenden SQL-Anfragen aus und notieren Sie sich die Anzahl der Ergebnistupel. Wie können Sie sich die Differenz erklären?

   ```
   SELECT * FROM Versicherungsnehmer
   WHERE (Geburtsdatum = '1950-02-01')
         OR NOT (Geburtsdatum = '1950-02-01');

   SELECT * FROM Versicherungsnehmer;
   ```

   Bei einem Tupel ist statt einem gültigen Geburtsdatum `NULL` gespeichert. Dieses Tupel wird daher beim obigen Vergleich nicht erfasst.

   Das betroffene Tupel kann zum Beispiel wie folgt bestimmt werden:

   ```
   SELECT * FROM Versicherungsnehmer WHERE Geburtsdatum IS NULL;
   ```

   Allgemeine Lösung mit Nachbildung des von MySQL nicht unterstützten `EXCEPT`-Operators:

   ```
   SELECT *
   FROM Versicherungsnehmer
   WHERE ID NOT IN (

       SELECT ID
       FROM Versicherungsnehmer
       WHERE Geburtsdatum = '1950-02-01'

       UNION
   ```

```
    SELECT ID
    FROM Versicherungsnehmer
    WHERE NOT Geburtsdatum = '1950-02-01'
);
```

4. [**Besprechung Ihrer Lösung in der Präsenzübung**] Wie müssen Sie die erste Anfrage aus der vorherigen Aufgabe erweitern, damit Sie vollständig äquivalent zur zweiten Anfrage ist?

```
SELECT * FROM Versicherungsnehmer
 WHERE (Geburtsdatum = '1950-02-01')
    OR NOT (Geburtsdatum = '1950-02-01')
    OR Geburtsdatum IS NULL;
```

5. Der Umgang mit NULL-Werten in SQL ist auch in der Fachwelt nicht unumstritten. Recherchieren Sie bei Interesse die Hintergründe dieser Diskussion. Einen guten Startpunkt liefern die beiden Publikationen von Rubinson (http://www.u.arizona.edu/~rubinson/scrawl/Rubinson.2007.Nulls_Three-Valued_Logic_and_Ambiguity_in_SQL.pdf) und Grant (http://www09.sigmod.org/sigmod/record/issues/0809/p23.grant.pdf).

2.2 Anfrageformulierung

1. Zeigen Sie alle Dienstwagen an, die keinem Mitarbeiter zugeordnet sind.

```
SELECT *
  FROM Dienstwagen
 WHERE Mitarbeiter_ID IS NULL;
```

2. [**Besprechung Ihrer Lösung in der Präsenzübung**] Gesucht werden Kombinationen von Mitarbeitern (Vorname und Name) und ihren Dienstwagen (ID, Kennzeichen und Fahrzeugtyp). Es geht um die Abteilungen 1 bis 5; auch nicht-persönliche Dienstwagen sollen aufgeführt werden.

```
    SELECT mi.Name, mi.Vorname,
           dw.ID AS DIW, dw.Kennzeichen, dw.Fahrzeugtyp_ID AS Typ
      FROM Mitarbeiter mi
RIGHT JOIN Dienstwagen dw
        ON dw.Mitarbeiter_ID = mi.ID
     WHERE mi.Abteilung_ID <= 5
        OR mi.ID IS NULL;
```

3. Gesucht werden die Bezeichnungen aller registrierten Versicherungsgesellschaften und (soweit vorhanden) deren jeweils zugehörige Kunden mit Name und Vorname.

```
   SELECT Bezeichnung, Name, Vorname
     FROM Versicherungsgesellschaft vg
LEFT JOIN Versicherungsnehmer vn
       ON vg.ID = vn.Versicherungsgesellschaft_ID;
```

4. [**Besprechung Ihrer Lösung in der Präsenzübung**] Erstellen Sie eine Anfrage für die Tabellen `Versicherungsvertrag, Fahrzeug, Zuordnung_SF-FZ`, in der folgende Bedingungen berücksichtigt werden:

   Es sollen Vertragsnummer, Abschlussdatum, Kennzeichen sowie ggf. anteilige Schadenshöhe angezeigt werden. Alle Fahrzeuge mit einem Schaden sollen angezeigt werden. Fahrzeuge

ohne Schaden sollen nur angezeigt werden, wenn der Vertrag vor 1990 abgeschlossen wurde. Das Ergebnis soll nach Schadenshöhe und Kennzeichen sortiert werden.
Hinweise: Benutzen Sie `UNION` zur Verknüpfung der Verträge mit und ohne Schaden und denken Sie daran, dass Sie auch Konstanten (z.B. natürliche Zahlen) als Spalte in der Projektion angeben können.

```
SELECT * FROM (

    SELECT Vertragsnummer, Abschlussdatum, Kennzeichen, Schadenshoehe
      FROM Versicherungsvertrag vv
INNER JOIN Fahrzeug fz on fz.ID = vv.Fahrzeug_ID
INNER JOIN Zuordnung_SF_FZ zu on fz.ID = zu.Fahrzeug_ID

  UNION

    SELECT Vertragsnummer, Abschlussdatum, Kennzeichen, 0
      FROM Versicherungsvertrag vv
INNER JOIN Fahrzeug fz on fz.ID = vv.Fahrzeug_ID
LEFT JOIN Zuordnung_SF_FZ zu on fz.ID = zu.Fahrzeug_ID
     WHERE Abschlussdatum < '1990-01-01'
       AND zu.ID IS NULL
) Schaeden

ORDER BY Schadenshoehe, Kennzeichen;
```

5. Gesucht werden die Bezeichnungen aller registrierten Versicherungsgesellschaften sowie alle Kunden mit Name, Vorname, soweit der Name mit 'S' beginnt.
Hinweis: Bedenken Sie, dass MySQL nativ keinen `FULL OUTER JOIN` unterstützt.

```
  (SELECT Bezeichnung, Name, Vorname
     FROM versicherungsgesellschaft vg
LEFT JOIN versicherungsnehmer vn
       ON vn.Versicherungsgesellschaft_ID = vg.ID
      AND vn.Name like "S%")

  UNION

    (SELECT Bezeichnung, Name, Vorname
       FROM versicherungsgesellschaft vg
RIGHT JOIN versicherungsnehmer vn
         ON vg.ID = vn.Versicherungsgesellschaft_ID
      WHERE vg.ID IS NULL
        AND vn.Name LIKE "S%")
```

6. Gesucht werden die Kennzeichen aller Dienstwagen, die Bezeichnungen deren Fahrzeugtypen sowie die Namen deren Hersteller. Alle Fahrzeugtypen sollen in der Ergebnisrelation auftauchen.

```
    SELECT Kennzeichen, Bezeichnung, Name
      FROM Dienstwagen dw
RIGHT JOIN Fahrzeugtyp ft ON ft.ID = dw.Fahrzeugtyp_ID
INNER JOIN Fahrzeughersteller fh ON fh.ID = ft.Hersteller_ID;
```

7. [**Besprechung Ihrer Lösung in der Präsenzübung**] Gesucht werden Angaben zu den Mitarbeitern und den Dienstwagen. Beim Mitarbeiter sollen Name und Vorname angegeben werden, bei den Dienstwagen Kennzeichen, Fahrzeugtyp-Bezeichnung und Name des Herstellers. Alle Fahrzeugtypen sollen in der Ergbenisrelation auftauchen.

```
    SELECT mi.Name, mi.Vorname,
           dw.Kennzeichen, ft.Bezeichnung, fh.Name as HST
      FROM Dienstwagen dw
INNER JOIN Mitarbeiter mi ON mi.id = dw.Mitarbeiter_id
RIGHT JOIN Fahrzeugtyp ft ON ft.Id = dw.Fahrzeugtyp_id
INNER JOIN Fahrzeughersteller fh ON fh.Id = ft.Hersteller_id;
```

8. Ergänzen Sie die vorherige Anfrage insofern, dass nur Mitarbeiter der Abteilungen 1 bis 5 angezeigt werden. Die Zeilen ohne Mitarbeiter sollen unverändert ausgegeben werden.

```
    SELECT mi.Name, mi.Vorname,
           dw.Kennzeichen, ft.Bezeichnung, fh.Name as HST
      FROM Dienstwagen dw
INNER JOIN Mitarbeiter mi ON mi.id = dw.Mitarbeiter_id
RIGHT JOIN Fahrzeugtyp ft ON ft.Id = dw.Fahrzeugtyp_id
INNER JOIN Fahrzeughersteller fh ON fh.Id = ft.Hersteller_id
     WHERE (mi.Abteilung_id <= 5)
        OR (mi.id IS NULL);
```

9. Bestimmen Sie, wie viele Fahrzeugtypen pro Hersteller registriert sind, und nennen Sie Namen und Land der Hersteller.
   Hinweis: Erstellen Sie zunächst eine Anfrage für Anzahl plus Hersteller-ID und verknüpfen Sie das Ergebnis mit der Tabelle Hersteller.

```
   SELECT Name, Land, Anzahl
     FROM (
            SELECT ft.Hersteller_ID as ID
                   , COUNT(ft.Hersteller_ID) as Anzahl
              FROM Fahrzeugtyp ft
          GROUP BY ft.Hersteller_ID
         ) temp
     JOIN Fahrzeughersteller fh
       ON fh.ID = temp.ID;
```

10. [**Besprechung Ihrer Lösung in der Präsenzübung**] Finden Sie heraus, wer die ältesten zehn Versicherungsnehmer sind.

```
  SELECT COUNT(*) AS Rang, v1.ID, v1.Geburtsdatum
    FROM Versicherungsnehmer v1 , Versicherungsnehmer v2
   WHERE v2.Geburtsdatum <= v1.Geburtsdatum
GROUP BY v1.ID, v1.Geburtsdatum
  HAVING COUNT(*) <= 10
ORDER BY Rang ASC;
```

    Funktionsweise der Topliste:

    1) Auto-JOIN liefert Kreuzprodukt

    2) >=-Operator erzeugt Dreieckstruktur:

    3) Platz 1 hat nur genau einen Datensatz, der größer-gleich ihm ist: er selber.

4) Platz 2 hat genau zwei Datensätze, die größer-gleich ihm sind: er selber und Platz 1. usw.

5) GROUP BY fasst die „zweidimensionalen Daten“ wieder zu einer Liste zusammen. Wichtig: Alle Attribute aus der Projektion müssen im GROUP BY auftauchen, um eine korrekte Ausführung der Anfrage sicherzustellen!

6) Mit COUNT(*) können nach der Gruppierung die Anzahl der Datensätze abgelesen werden, die größer-gleich sind: das entspricht genau dem Rang.

7) HAVING COUNT(*) <= x limitiert die Topliste auf die besten x Einträge

11. Welche Verträge (ID, Vertragsnummer, Abschlussdatum, Art) hat der Mitarbeiter Christian Braun abgeschlossen? Ignorieren Sie die Möglichkeit, dass es mehrere Mitarbeiter dieses Namens geben könnte.

```
SELECT ID, Vertragsnummer, Abschlussdatum, Art
  FROM Versicherungsvertrag
 WHERE Mitarbeiter_ID
       IN ( SELECT ID
              FROM Mitarbeiter
             WHERE Name = 'Braun'
               AND Vorname = 'Christian' );
```

12. Zeigen Sie alle Verträge (ID, Vertragsnummer, Abschlussdatum, Art), die zum Kunden „Heckel Obsthandel GmbH“ gehören. Ignorieren Sie die Möglichkeit, dass der Kunde mehrfach gespeichert sein könnte.

```
SELECT ID, Vertragsnummer, Abschlussdatum, Art
  FROM Versicherungsvertrag
 WHERE Versicherungsnehmer_ID
       = ( SELECT ID FROM Versicherungsnehmer
             WHERE Name ='Heckel␣Obsthandel␣GmbH' );
```

13. Ändern Sie die die vorherige Anfrage derart ab, sodass auch mehrere Kunden mit diesem Namen als Ergebnis denkbar sind.

```
SELECT ID, Vertragsnummer, Abschlussdatum, Art
  FROM Versicherungsvertrag
 WHERE Versicherungsnehmer_ID
       IN ( SELECT ID FROM Versicherungsnehmer
             WHERE Name ='Heckel␣Obsthandel␣GmbH' );
```

14. Zeigen Sie alle Fahrzeuge, die an einem Schadensfall beteiligt waren.

```
SELECT ID, Kennzeichen, Fahrzeugtyp_ID as TypID
  FROM Fahrzeug fz
 WHERE ID IN ( SELECT Fahrzeug_ID
                 FROM Zuordnung_sf_fz
             );
```

15. **[Besprechung Ihrer Lösung in der Präsenzübung]** Zeigen Sie alle Fahrzeugtypen (mit ID, Bezeichnung und Name des Herstellers), die an einem Schadensfall beteiligt waren.

```
   SELECT DISTINCT ft.ID as TypID, ft.Bezeichnung as Typ
                 , fh.Name as Hersteller
     FROM Fahrzeugtyp ft
INNER JOIN Fahrzeughersteller fh ON fh.ID = ft.Hersteller_ID
INNER JOIN Fahrzeug fz ON ft.ID = fz.Fahrzeugtyp_ID
    WHERE fz.ID IN ( SELECT Fahrzeug_ID
                       FROM Zuordnung_sf_fz
                   );
```

16. Bestimmen Sie alle Fahrzeuge des Herstellers Volkswagen mit Angabe des jeweiligen Typs.

```
SELECT fz.ID, fz.Kennzeichen, VWTypen.ID AS Typ, VWTypen.Bezeichnung
  FROM Fahrzeug fz
      ,( SELECT ID, Bezeichnung
          FROM Fahrzeugtyp
         WHERE Hersteller_ID =
               ( SELECT ID
                   FROM Fahrzeughersteller
                  WHERE Name = 'Volkswagen' )
       ) VWTypen
 WHERE fz.Fahrzeugtyp_ID = VWTypen.ID;
```

17. [**Besprechung Ihrer Lösung in der Präsenzübung**] Zeigen Sie zu jedem Mitarbeiter der Abteilung „Vertrieb“ die ID des ersten Vertrags an, den er abgeschlossen hat. Der Mitarbeiter soll mit ID und Vorname sowie Name angezeigt werden.

```
SELECT vv.ID, mi.ID, mi.Name, mi.Vorname
FROM (SELECT MIN(Abschlussdatum) AS mindate, Mitarbeiter_ID
    FROM versicherungsvertrag
    GROUP BY Mitarbeiter_ID) ab
JOIN Versicherungsvertrag vv ON vv.Abschlussdatum = ab.mindate
AND ab.Mitarbeiter_ID = vv.Mitarbeiter_ID
RIGHT JOIN Mitarbeiter mi ON mi.ID = vv.Mitarbeiter_ID
WHERE mi.Abteilung_ID = (SELECT ID FROM Abteilung WHERE Bezeichnung = 'Vertrieb')
GROUP BY mi.ID, mi.Name, mi.Vorname;
```

2.3 Schemamodifikation

1. Bei der Suche nach Dienstwagen sollen mit der View `Dienstwagen_Anzeige` immer auch angezeigt werden:

   a) Name und Vorname des Mitarbeiters

   b) ID und Bezeichnung seiner Abteilung

   c) der Fahrzeugtyp (nur als ID)

   Stellen Sie sicher, dass auch nicht-persönliche Dienstwagen immer angezeigt werden, und kontrollieren Sie das Ergebnis durch eine Anfrage ähnlich diesem Muster:

```
SELECT * FROM Dienstwagen_Anzeige
WHERE ( Abt_ID BETWEEN 5 AND 8 ) OR ( Mi_Name IS NULL );
```

```
CREATE VIEW Dienstwagen_Anzeige
     ( Kennzeichen, TypId
       , Mi_Name, Mi_Vorname
       , Ab_ID, Ab_Name )
    AS SELECT dw.Kennzeichen, dw.Fahrzeugtyp_ID,
              mi.Name, mi.Vorname,
              mi.Abteilung_ID,
              ab.Bezeichnung
         FROM Dienstwagen dw
     LEFT JOIN Mitarbeiter mi
            ON mi.ID = dw.Mitarbeiter_ID
     LEFT JOIN Abteilung ab
            ON ab.ID = mi.Abteilung_ID;
```

2. Erstellen Sie eine Sicht `Vertrag_Anzeige`, bei der zu jedem Vertrag angezeigt werden:

   a) ID, Vertragsnummer, Abschlussdatum, Art

   b) Name, Vorname des Mitarbeiters

   c) Name, Vorname des Versicherungsnehmers

   d) Kennzeichen des Fahrzeugs

```
CREATE VIEW Vertrag_Anzeige
   ( ID, Vertragsnummer, Abschlussdatum, Art,
     Mi_Name, Mi_Vorname,
     Vn_Name, Vn_Vorname,
     Kennzeichen )
  AS SELECT vv.ID, vv.Vertragsnummer, vv.Abschlussdatum, vv.Art,
            mi.Name, mi.Vorname,
            vn.Name, vn.Vorname,
            fz.Kennzeichen
       FROM Versicherungsvertrag vv
       JOIN Mitarbeiter mi
         ON mi.ID = vv.Mitarbeiter_ID
       JOIN Versicherungsnehmer vn
         ON vn.ID = vv.Versicherungsnehmer_ID
       JOIN Fahrzeug fz
         ON fz.ID = vv.Fahrzeug_ID;
```

3. **[Besprechung Ihrer Lösung in der Präsenzübung]** Erstellen Sie eine Sicht `Schaden_Anzeige`, bei der zu jedem an einem Schadensfall beteiligten Fahrzeug angezeigt werden:

   a) ID, Datum, Gesamthöhe eines Schadensfalls

   b) Kennzeichen und Typ des beteiligten Fahrzeugs

   c) Anteiliger Schaden

   d) ID des Versicherungsvertrags

```
CREATE VIEW Schaden_Anzeige
   ( ID, Datum, Gesamtschaden,
     Kennzeichen, Typ,
     Schadensanteil,
     VV_ID )
```

```
AS SELECT sf.ID, sf.Datum, sf.Schadenshoehe,
          fz.Kennzeichen, fz.Fahrzeugtyp_ID,
          zu.Schadenshoehe,
          vv.ID
       FROM Zuordnung_SF_FZ zu
       JOIN Schadensfall sf
         ON sf.ID = zu.Schadensfall_ID
       JOIN Fahrzeug fz
         ON fz.ID = zu.Fahrzeug_ID
       JOIN Versicherungsvertrag vv
         ON fz.ID = vv.Fahrzeug_ID;
```

4. Erstellen Sie die Definition für eine Tabelle mit internationalen Postleitzahlen: laufende Nummer, Land, Code, Ortsname. Legen Sie für jede Spalte möglichst viele Einzelheiten fest. Seien Sie kreativ und verwenden Sie die Sprachkonstrukte, die Sie bereits in der Vorlesung kennengelernt haben.

```
CREATE TABLE PLZ_Codes
     ( ID INTEGER PRIMARY KEY,
       Land CHAR (2) NOT NULL DEFAULT 'DE',
       Code VARCHAR(10) NOT NULL, -- auch CHAR(10) ist denkbar
       Ort VARCHAR(30) NOT NULL,
       CONSTRAINT PLZ_UK UNIQUE (Land, Code)
     );
```

5. [**Besprechung Ihrer Lösung in der Präsenzübung**] Ändern Sie die Tabelle `Versicherungsnehmer` so, dass die Spalte Eigener_Kunde nur die Werte 'J' und 'N' annehmen darf.

```
   ALTER TABLE Versicherungsnehmer
ADD CONSTRAINT Versicherungsnehmer_Eigener_Kunde
        CHECK ( Eigener_Kunde = 'J'
                OR Eigener_Kunde = 'N' );
```

   Anmerkung: MySQL nimmt Ihren gültigen CHECK-Constraint zwar entgegen, in der aktuellen Implementierung wird die Einhaltung jedoch nicht durch das DBMS überprüft.

6. Erstellen Sie nun die nötigen SQL-Anweisungen, um alle Tabellen und Views in Ihrer Datenbank wieder löschen zu können.

```
DROP TABLE PLZ_Codes;
DROP VIEW Schaden_Anzeige;
DROP VIEW Vertrag_Anzeige;
DROP VIEW Dienstwagen_Anzeige;
DROP TABLE PLZ_Aenderung;
DROP TABLE Abteilung;
DROP TABLE Fahrzeughersteller;
DROP TABLE Fahrzeugtyp;
DROP TABLE Fahrzeug;
DROP TABLE Mitarbeiter;
DROP TABLE Dienstwagen;
DROP TABLE Schadensfall;
DROP TABLE Versicherungsgesellschaft;
DROP TABLE Versicherungsnehmer;
DROP TABLE Versicherungsvertrag;
DROP TABLE Zuordnung_SF_FZ;
```

Beim Löschen muss die richtige Reihenfolge eingehalten worden, falls die Überprüfung der referentiellen Integrität durch das DBMS nicht temporär deaktiviert wird.

Zusätzliche Materialien zum Lernen und Einüben von SQL

1. Kontrollfragen in den Vorlesungsunterlagen
2. Online:
   a) http://www.sql-island.de (spannendes Text-Adventure zum Lernen von SQL)
   b) https://en.wikibooks.org/wiki/SQL_Exercises (viele leichtere Aufgaben, aber auch komplexe Queries)
   c) https://lagunita.stanford.edu/courses/DB/SQL/SelfPaced/ (Online-Kurs der Stanford University)
3. Literatur:
   a) Joe Celko: **SQL for Smarties** (ausführliche Sprachreferenz)
   b) Bill Karwin: **SQL Antipatterns: Avoiding the Pitfalls of Database Programming** (Diskussion von Antipatterns im Alltagsgebrauch von SQL)
   c) C. J. Date: **SQL and Relational Theory** (Formulierung von formal präzisem SQL-Code)
   d) Alfons Kemper: **Übungsbuch Datenbanksysteme** (eigenes Kapitel mit SQL-Aufgaben, passt zum Lehrbuch und enthält Lösungsvorschläge)
   e) Lynn Beighley: **Head First SQL** (humorvolle Einführung in SQL mit Aufgaben)

Anmerkungen zur Lizenz

Zahlreiche Aufgaben auf dem vorliegenden Übungsblatt sowie die zur Verfügung gestellte Beispieldatenbank basieren auf dem Wikibook **Einführung in SQL**. Hauptautor dieses Werkes ist Jürgen Thomas. Das vollständige Buch kann unter https://de.wikibooks.org/wiki/Einführung_in_SQL abgerufen werden und bietet einen guten Startpunkt, falls Sie sich weiterführend mit SQL beschäftigen wollen. Bitte beachten Sie auch die weiteren Nutzungsbedingungen unter https://wikimediafoundation.org/wiki/Terms_of_Use/de.

Die Aufgaben wurden hinsichtlich ihrer Eignung im Kontext der Lehrveranstaltung *Konzeptionelle Modellierung* ausgewählt, entsprechend modifiziert und ergänzt. Das zugrundeliegende Wikibook steht unter der Lizenz **Creative Commons: Namensnennung - Weitergabe unter gleichen Bedingungen** (https://creativecommons.org/licenses/by-sa/3.0/deed.de). Das vorliegende Übungsblatt (inkl. aller neu hinzugefügten Aufgaben) steht dementsprechend unter derselben Lizenz.