1 INSERT Statement

Das `INSERT` Statement haben wir bereits kennengelernt. Hier gab es zwei verschiedene Versionen. Einmal die Option, mehrere Daten auf einmal zu schreiben:

```
INSERT INTO Kunde (ID, Vorname, Nachname, Geschlecht, eMail,
Telefon, Geburtsdatum, KundeSeit, LetzterLogin) VALUES
(1, 'Lio', 'Kraft', 'M', 'lio.kraft@fake_arcor.de',
 '0170/787460', '1999-08-27', '2020-03-04', '2025-06-02 05:15:20'),
(2, 'Toni', 'Karger', 'M', 'toni.karger@fake_mail.yahoo.com',
 0174/84527', '1966-08-23', '2020-03-09', '2024-12-28 04:11:55');
```

Die andere Option wäre, die Werte mittels SET direkt zu setzen:

```
INSERT INTO Kunde SET
ID = 3,
Vorname = 'Lenia',
Nachname = 'Blum',
Geschlecht = 'W',
eMail = 'lenia.blum@fake_postmail.de',
Telefon = '0174/6260351',
Geburtsdatum = '1963-06-16',
KundeSeit = '2020-03-13',
LetzterLogin = '2025-06-14 03:19:45';
```

2 DELETE Statement

Das Löschstatement heißt in SQL `DELETE`. Hiermit löschen wir Daten. Ganze Tabellen werden mit `DROP` gelöscht!

```
DELETE FROM Kunde WHERE ID = 3;
```

Achtung: wenn das „WHERE“ nicht folgt, werden **alle Datensätze gelöscht**!

3 REPLACE Statement

Durch die Definition eines Primärschlüssels in der Kundentabelle, haben wir jedem Datensatz eine eigene, eindeutige ID gegeben. Versuchen wir eine bereits existierende ID nochmal zu schreiben, erhalten wir eine Fehlermeldung. Möchten wir einen Datensatz austauschen, so müssten wir eigentlich zuerst den Datensatz mit der entsprechenden ID löschen:

```
DELETE FROM Kunde WHERE ID = 3;
```

um ihn anschließend mit einem weiteren `INSERT` Statement neu, mit anderen Werten (bspw. Blume anstatt Blum) aber mit gleicher ID neu zu schreiben. Das `REPLACE` Statement macht dies jedoch in einem Schritt:

```
REPLACE INTO Kunde SET
ID = 3,
Vorname = 'Lenia',
Nachname = 'Blume',
Geschlecht = 'W',
eMail = 'lenia.blum@fake_postmail.de',
Telefon = '0174/6260351',
Geburtsdatum = '1963-06-16',
KundeSeit = '2020-03-13',
LetzterLogin = '2025-06-14 03:19:45';
```

4 UPDATE Statement

Wollen wir nur einzelne Attribute ändern, so können wir dies mit einem `UPDATE` machen. Hier müssen wir aber exakt angeben, welchen Datensatz wir ändern wollen. Vergessen wir dies, werden alle Datensätze geändert!

```
UPDATE Kunde SET Nachname = 'Blum' WHERE ID = 3;
```

5 LOAD DATA INFILE

Vor allem größere Datenmengen sollten mit einem alternativen Verfahren geladen werden – dem `LOAD DATA INFILE`. Dieser Befehl ist MySQL spezifisch – bei anderen Datenbanken finden sich andere Lösungen (wie SQL*Loader bei Oracle oder dem DB2 LOAD Befehl).
Grundgedanke dieser Befehle ist es, aus einem Textfile (auch „Flatfile“ genannt) die Daten strukturiert in die Datenbank zu laden. Hierzu muss mit Hilfe des Befehls (oder wie beim SQL*Loader auch mit Hilfe eines Kontrollfiles) dem System mitgeteilt werden, wie die Daten zu laden sind. Sehen wir uns im Fall von MySQL mal ein einfaches File an, welches in unsere Kundentabelle geladen werden kann:

```
5|Maxim|Maier|M|maxim.maier@fake_mail.aol.com|0174/086677|1952-6-6|2020-3-17|2023-9-13 21:26:4
6|Piet|Ney|M|piet.ney@fake_combomail.de|0176/745334|1996-3-6|2020-3-18|2024-3-7 12:6:1
```

Wir erzeugen also ein einfaches Textfile „Kunde.txt“ und geben die genannten Daten ein. Nun versuchen wir das File in unsere Tabelle zu laden. Dazu kopieren wir es in den Upload-Ordner von MySQL:

```
C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\
```

Nun führen wir im SQL-Client den folgenden Befehl aus. Achtung – der Backslash ist in vielen Systemen – und so auch in MySQL – ein Escapezeichen, weshalb wir es doppelt eintragen müssen (oder im Fall von MySQL alternativ auch durch einen Forwardslash „/“ ersetzt werden kann:

```
LOAD DATA INFILE
'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\Kunde.txt'
INTO TABLE KeyTestTab
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\r\n';
```

Der Befehl lädt also die Daten aus dem File direkt in die Tabelle. Hierbei muss aber folgendes sichergestellt sein:

- Die Reihenfolge der Datenspalten muss mit der Reihenfolge der Tabellenspalten übereinstimmen
- Das Textfile darf keine Überschriften haben
- Das Textfile muss durch Pipe-Character „|“ separiert sein
- Das Textfile benötigt den Windows Zeilentrenner „\r\n“
- In dem Textfile dürfen die Felder nicht mit Anführungsstrichen eingeschlossen sein

Wird bspw. kein „`FIELDS TERMINATED BY`” angegeben, so erwartet MySQL die Felder Tabulator-Separiert. Ohne die „`LINES TERMINATED`“ Option erwartet MySQL den UNIX Zeilentrenner „\n“.

Hinweis: Bei der Nutzung des Befehls `LOAD DATA INFILE` ist der häufigste Fehler eine falsche Reihenfolge der Parameter. Im Internet steht eine sehr ausführliche Beschreibung dieses Befehls, inklusive notwendiger Reihenfolge!

6 Aufgabenstellung

In Teams finden Sie nun die Textfiles für unser Datenmodell. Erstellen sie nun ein SQL-Skript, welches eine Datenbank „onlineshop“ erstellt, alle Tabellen erzeugt und anschließend die entsprechenden Textfiles einlädt. Die Textfiles sind Tabulator-Separiert, weshalb sie auf die Angabe der „`FIELD TERMINATED BY`“ Direktive verzichten können.