



Datenbanken und SQL

(Woche 2 - Tag 3)

Agenda

Data Manipulation Language (DML)

- Definition + Motivation
- INSERT INTO
 - alle Attribute
 - ausgewählte Attribute
 - Übergabewert CURDATE()
 - Wirkung des Constraints DEFAULT
- DELETE FROM
 - Kurze Vorbemerkung zur internen Verarbeitung von SQL-Anweisungen
 - ohne WHERE [vgl. TRUNCATE]
 - mit WHERE
- UPDATE [+ WHERE]

DML (Data Manipulation Language)

Definition + Motivation

- Wir haben uns zu Beginn ausschließlich mit der Implementierung von „leeren“ Datenbanken befasst, also mit der „**Definition**“ des Schemas, bzw. der **DDL**.
- Wir wollen uns nun anschauen, wie wir die Werte eines Datensatzes innerhalb einer Datenbank **einpflügen**, **aktualisieren** und/oder **löschen** können.
- Auf diese Weise beeinflussen wir den Inhalt (bzw. führen eine **Manipulation** dieser Inhalte durch). Daher gehören die kommenden Befehle zur Data „**Manipulation Language**“.
- Die **Motivation** dieser Lerninhalte ergibt sich dann aber offensichtlich von selbst, denn natürlich wollen wir nach knapp 1,5 Wochen unseres Unterrichts nicht bei „leeren“ Datenbanken stehen bleiben, sondern (endlich) auch Daten eintragen 😊.

INSERT INTO

INSERT INTO Tabellen-Name(Liste der Attribute) **VALUES** (einziger Datensatz);

Einpflügen des Kunden Peter Müller, Email-Adresse: abc@xyz (die ID soll automatisch vergeben werden):

```
INSERT INTO Kunden(Vorname, Nachname, Email)
VALUES ("Peter", "Müller", "abc@xyz");
```

Einpflügen der Abrechnung (ID 1) des Kunden (ID 3) vom 2. Januar 2017 (ID wird händisch gesetzt):

```
INSERT INTO Abrechnung(Abrechnung_ID, Kunde_ID, Datum)
VALUES (1, 3, "2017-01-02");
```

Bei MariaDB ist der Kalender-Datums-Typ standardmäßig vom Format "jjjj-mm-tt"

INSERT INTO Tabellen-Name(Liste der Attribute) VALUES (einziger Datensatz);

Die **Reihenfolge** der Attribute (die in den Klammern hinter dem Tabellennamen eingetragen werden) kann **frei gewählt** werden. Selbstverständlich müssen dann aber alle Daten des Datensatzes genau dieser zuvor gewählten Attribut-Reihenfolge entsprechen:

Einpfelegen des **Kunden Peter Müller**, Email-Adresse: **abc@xyz** (die **ID** soll **automatisch** vergeben werden):

```
INSERT INTO Kunden(Nachname, Email, Vorname)
VALUES ("Müller", "abc@xyz", "Peter", );
```

INSERT INTO Tabellen-Name(Liste der Attribute) VALUES (einziger Datensatz);

Falls man **alle Attribute** eintragen möchte, so kann man auf die Klammer hinter dem Tabellennamen verzichten, muss dann aber beim Datensatz genau jene Reihenfolge beachten, die bei CREATE TABLE festgelegt wurde:

Einpfelegen des **Kunden Peter Müller**, Email-Adresse: **abc@xyz** (die **ID** (=1) soll **händisch** vergeben werden):

```
INSERT INTO Kunden VALUES (1, "Peter", "Müller", "abc@xyz");
```

Einpfelegen des **Kunden Peter Müller**, Email-Adresse: **abc@xyz** (die **ID** soll **automatisch** vergeben werden):

```
INSERT INTO Kunden VALUES (DEFAULT, "Peter", "Müller", "abc@xyz");
```


INSERT INTO Tabellen-Name(Liste der Attribute) **VALUES** (Datensatz 1), (Datensatz 2), (Datensatz 3), ...;

Einpfelegen von (z.B.) den folgenden 5 **Produkten** (nur Produktname und Preis, AUTO_INCREMENT soll genutzt werden):

INSERT INTO Produkt(Produkt_Name, Euro_Preis)

VALUES

("Produkt A", 25.99),

("Produkt B", 2.22),

("Produkt C", 179.8),

("Produkt D", 0.95),

("Produkt E", 2);

Hinweise:

Falls ein Attribut mit dem Constraint **NOT NULL** versehen wurde, so **muss** es bei INSERT INTO **mit einem Wert** belegt werden.

Bei Zahlen vom Typ **FLOAT** (oder **DOUBLE**) müssen nicht alle (laut Typ festgelegten) Nachkommastellen explizit benannt werden:
Wenn (z.B.) **laut FLOAT(7,2)** maximal 2 Nachkommastellen vorgesehen sind, so werden die Werte „179.8“ und „2“ automatisch zu 179.80 und 2.00 ergänzt.

Übergabewert **CURDATE()**

Wenn sich ein Kunde (z.B. der Kunde mit ID 5) auf der Website von „Geld_her“ einloggt, so wird ihm automatisch eine Abrechnung zugeordnet.

- Die Abrechnung_ID wird durch **AUTO_INCREMENT** gesetzt.
- Der **Fremdschlüssel** ist die **ID** des sich anmeldenden **Kunden**.
- Das Kalender-Datum soll das jeweils **aktuelle Datum** sein.

Dies können wir durch folgenden Befehl erreichen:

```
INSERT INTO Abrechnung(Kunde_ID, Datum) VALUES (5, CURDATE());
```

Wirkung des Constraints **DEFAULT**

Um diesen Constraint erläutern zu können, ändern wir zunächst das Attribut „Euro_Preis“:

```
ALTER TABLE Produkt CHANGE Euro_Preis Euro_Preis FLOAT(7,2) DEFAULT 0.99;
```

Falls wir nun ein neues Produkt (Produkt-ID 77, Hersteller-ID 15, Produktname „ABC“) einpflegen (und also keinen Wert für das Attribut Euro_Preis eintragen), so lautet der entsprechende Befehl:

```
INSERT INTO Produkt(Produkt_ID, Hersteller_ID, Produkt_Name) VALUES (77, 15, "ABC");
```

Wenn wir anschließend nachschauen, was in der Tabelle Produkt „ankam“, so werden wir feststellen, dass der Euro_Preis **„Default-mäßig“** auf den Betrag 0.99 (€) gesetzt wurde.

DELETE FROM

Interne Verarbeitung (vieler) SQL-Anweisungen

- Wir werden beim kommenden Befehl (DELETE FROM) erstmalig eine typische Vorgehensweise eines **SQL-Parsers** (grob gesprochen: „eine Art von Compiler“) kennenlernen.
- Diese Vorgehensweise ist Ausdruck der Tatsache, dass es sich bei SQL um eine „**Deklarative Sprache**“ handelt. Wer diese Sprache nutzt, der formuliert lediglich „Ziele“, der Parser entscheidet „auf welche Weise“ dieses Ziel erreicht wird.
- Viele SQL-Statements teilen nun aber „Ziele“ mit, die bzgl. einer (explizit angesprochenen) Tabelle abgearbeitet werden sollen. Intern wird dann eine **Schleife** gestartet, die **jeden Datensatz** der angesprochenen **Tabelle** durchläuft.
- Was dies bedeutet schauen wir uns nun am **konkreten Beispiel** an:

DELETE FROM Tabellen-Name; [ohne „WHERE“]

DELETE (=„Löschen“) teilt das „Ziel“ mit.

FROM fragt nach der Tabelle, in der dieses Ziel umgesetzt werden soll.

Wenn wir nun also **alle** Datensätze (z.B.) der Tabelle Produkt löschen wollen, so notieren wir:

DELETE FROM Produkt;

Intern wird dann - wie erwähnt - eine Schleife gestartet, die **jeden** Datensatz der Tabelle Produkt durchläuft und (in diesem Fall) jeden dieser Datensätze **löscht**.

Hinweis:

Mit „**DELETE FROM** Tabellen_Name;“ werden alle Datensätze „physisch“ gelöscht (und können also nicht rekonstruiert werden).
Mit „**TRUNCATE TABLE** Tabellen_Name;“ wird lediglich die Verlinkung auf diese Daten gelöscht (und der Speicherplatz freigegeben).

TRUNCATE ist daher (insbesondere bei sehr großen Datenmengen) **performanter**, aber „Datenschutz-technisch“ **unsicherer**.

DELETE FROM Tabellen-Name; [mit „WHERE“]

Üblicherweise werden wir nicht alle Datensätze einer Tabelle löschen wollen, sondern nur solche, die eine bestimmte **Bedingung** erfüllen. (Alternative Formulierung: Nur solche Datensätze, „**wo**“ diese Bedingung erfüllt ist).

Solche Bedingungen werden wir mit **WHERE** einleiten (und das Konstrukt „WHERE+Bedingung“ als **WHERE-Klausel** bezeichnen).

Wir betrachten einige erste Beispiele:

Es sollen alle Produkte gelöscht werden, die teurer als 20 Euro sind:

DELETE FROM Produkt **WHERE** Euro_Preis > 20;

Es sollen alle Abrechnungen gelöscht werden, die am „10. März 2022“ eingereicht wurden:

DELETE FROM Abrechnung **WHERE** Datum = "2022-03-10"; # **Achtung:** Es wird mit = (und nicht mit ==) gearbeitet!

Es sollen alle Kunden gelöscht werden, die mit Nachnamen „Müller“ heißen:

DELETE FROM Kunde **WHERE** Nachname = "Müller"; # **Texte** wie üblich in „**Anführungszeichen**“

Es sollen alle Hersteller gelöscht werden, deren ID kleiner oder gleich 3 ist:

DELETE FROM Hersteller **WHERE** Hersteller_ID <= 3;

Es sollen alle Speditionen gelöscht werden, deren Name alphabetisch hinter „ABC Spedition“ liegen:

DELETE FROM Spedition **WHERE** Spedition_Name > "ABC Spedition";

UPDATE

UPDATE Tabellen-Name SET Attribut [WHERE+Bedingung] ;

UPDATE (=„Aktualisieren“) teilt das „Ziel“ mit.


Tabellen-Name teilt mit, in welcher Tabelle dieses Ziel umgesetzt werden soll.

SET kündigt die Attribute an, die es in dieser Tabelle zu aktualisieren gilt.

Beispiele:

- Die Preise aller Produkte sollen auf 0.99 (€) gesetzt werden.
UPDATE Produkt **SET** Euro_Preis = 0.99;
- Alle Produkte mit einer ID>2 sollen den Preis 100 (€) erhalten.
UPDATE Produkt **SET** Euro_Preis = 100 **WHERE** Produkt_ID > 2;
- Die Abrechnung mit ID 3 muss korrigiert werden: sie stammt vom Kunden mit ID 5 und fand am 3. August 2019 statt.
UPDATE Abrechnung **SET** Kunde_ID = 5, Datum = "2019-08-03" **WHERE** Abrechnung_ID = 3;
- Der Kunde mit der ID 2 hat geheiratet und trägt nun den Nachnamen „Blume“.
UPDATE Kunde **SET** Nachname = "Blume" **WHERE** Kunde_ID = 2;

Gemeinsame Übung („Live-Coding“) -> A_02_03_01



Aufgabe_02_03_01

Nutzen Sie bitte die Musterlösung **ML_02_01_01** mit der Sie die leere Datenbank „Geld_her“ implementieren können.


Bearbeiten Sie bitte die heutigen Lerninhalte an Hand der folgenden **Aufgabenstellungen und Zusatzfragen**:

- a) Pflegen Sie bitte in der Tabelle „Kunde“ die **erste** Kundin namens **Martha Mustermann** mit der Email-Adresse „**a@b**“ ein.
- b) Versuchen Sie bitte anschließend eine Abrechnung (ID=1, Kunde_ID=2, Datum: 02.02.2022) einzupflegen. Warum kommt es zu einer Fehlermeldung?
- c) Wiederholen Sie bitte Aufgabe (b) mit **deaktivierter Fremdschlüsselüberprüfung**.
- d) (1) **Aktivieren** Sie bitte für alle übrigen Aufgaben die **Fremdschlüsselüberprüfung**.
(2) Korrigieren Sie bitte die Eingabe aus (c), indem Sie die Kunde_ID auf 1 ändern.
- e) (1) Warum können wir nicht mit der Hilfstabelle fortfahren?
(2) Pflegen Sie bitte die Spedition (ID: 1, Name: „Stolper AG“) ein.
- f) Pflegen Sie bitte den Hersteller (ID:1, Spedition_ID: 1, Name: „Wucher AG“) ein.
- g) Pflegen Sie bitte das Produkt (ID:1, Hersteller_ID: 1, Name: „Murks I“, Preis: 50 €) ein.
- h) Pflegen Sie bitte das Produkt (ID:2, Hersteller_ID: 1, Name: „Murks II“, Preis: 50 €) ein.
- i) Löschen Sie bitte das Produkt mit der ID = 1.
- j) Ändern Sie bitte beim Produkt(ID:2) den Namen und Preis in „Bombig“ und 100 €.
- k) Tragen Sie bitte in der Hilfstabelle den Einkauf von Produkt 2 auf Abrechnung 1 ein.

WBS TRAINING AG
Lorenzweg 5
D-12099 Berlin
Amtsgericht Berlin HRB 68531
Sitz der Gesellschaft: Berlin

Vorstand:
Heinrich Kronbichler,
Joachim Giese
Aufsichtsrat (Vorsitz): Dr. Daniel Stadler
USt-IdNr.: DE 209 768 248

GLS Gemeinschaftsbank eG
IBAN: DE18 4306 0967 1146 1814 00
BIC: GENODEM33GLS



GLS ist ein eingetragenes
Markenzeichen der GLS Gemeinschaftsbank eG
Stammregistergericht: Amtsgericht Berlin-Charlottenburg, HRB 151146, USt-IdNr. DE 209 768 248

Vielen Dank für Ihre Aufmerksamkeit!

