Grundlagen SQL

Cologne Network Consulting CNC GmbH
Wilhelm-Schlombs-Allee 2, 50858 Köln
Tel.: (0221) 9833790
http://www.cncgmbh.eu
info@koeln-net.com

Inhaltsverzeichnis

1 Einleitung

Ein RDBMS[1] enthält eine Vielzahl von Objekten, die in der Form von Tabellen modelliert sind. Daten bzw. Informationen für eine Datenbank werden in diesen Tabellen strukturiert gespeichert. Tabellen werden eindeutig durch ihren Namen (Bezeichner) identifiziert und bestehen aus vordefinierten Spalten und Zeilen. Spalten werden durch den Spaltennamen, einen Datentyp für die Menge der zulässigen Werte, und in jeder Zeile durch einen einzigen Wert dargestellt. Zeilen können wie Datensätze verstanden werden, wobei die einzelnen Eintragungen geordnet sind, entsprechend der o.a. Spaltendefinition.
Beispiel: Stadt, Land, hoch und tief sind die Spalten. Die Zeilen enthalten die Werte für die Tabelle Wetter:

Wetter

Stadt	Land	hoch	tief
Phoenix	USA	38	30
Helsinki	Finnland	26	8
Rom	Italien	32	19
Frankfurt	Deutschland	19	14
London	Großbritannien	24	18

[1] RDBMS = **r**elationales **D**aten**b**ank**m**anagement**s**ystem

2 CREATE DATABASE: Erzeugen einer Datenbank

Die Anweisung **CREATE DATABASE** wird zum Erzeugen einer neuen Datenbank verwendet.

Das Format der **CREATE DATABASE** Anweisung ist wie folgt:

CREATE DATABASE *databasename*;

Beispiele:

```
CREATE DATABASE Kunden;

CREATE DATABASE Mitarbeiter;
```

3 CREATE TABLE: Das Erzeugen einer Tabelle

Die Anweisung **CREATE TABLE** wird zum Erzeugen einer neuen Tabelle verwendet.

Das Format einer einfachen **CREATE TABLE** Anweisung ist wie folgt:

CREATE TABLE *tablename*
(*column1* data type,
column2 data type,
column3 data type);

Man kann auch optional Bedingungen (*constraints*) verwenden:

CREATE TABLE *tablename*
(*column1* data type [*constraint*],
column2 data type [*constraint*],
column3 data type [*constraint*]);

[ ] = optional

Es können beliebig viele Spalten (=*columns*) angegeben werden.

Beispiel (für MySQL):

```
CREATE TABLE Kunden
(KNr integer auto_increment not null,
Vorname varchar(15),
Nachname varchar(20),
KAlter integer,
Adresse varchar(30),
PLZ varchar(10),
Stadt varchar(20),
Staat varchar(20),
primary key(KNr));
```

Die Namen der Tabelle und der Spalten müssen mit einem Buchstaben beginnen, gefolgt von einem Buchstaben, einer Zahl oder einem underscore ("_") bis zu einer Länge von 30 Zeichen. SQL-Schlüsselwörter (z.B. SELECT, CREATE, INSERT etc) sind nicht erlaubt.

3.1 Data type

Jeder Spalte einer Tabelle wird ein Data Type (Wertbereich, Menge der möglichen Werte) zugeordnet. Falls eine Spalte, z.B. "Nachname", Namenstexte (Strings) enthalten soll, dann ist "varchar" (variable-length character) anzugeben. Folgende Data Types sind möglich:

Data Type	Beschreibung
char(*size*)	Textfeld mit fester Größe *size*. Es werden genau *size* Bytes Speicherplatz reserviert ganz gleich, wie groß der eingegebene Text tatsächlich ist. Maximale Größe 255 Zeichen
varchar(size)	Textfeld mit variabler Größe. Es wird nur so-viel Speicherplatz reserviert, wie tatsächlich Text eingegeben wird. Es können jedoch nur Texte mit einer Länge von *size* Bytes ein-gegeben werden. Maximale Größe 255 Zeichen
text	Textfelder mit mehr als 255 Zeichen
memo	Textfelder mit mehr als 255 Zeichen
smallint	Integerzahlen (Wertebereich: -32768 - 32768)
integer	Integerzahlen (Wertebereich: -2147483648 - 2147483647)
float	Fliesskommazahlen, 4 Bit
double	Fliesskommazahlen, 8 Bit
date	Datum und Uhrzeit
enum	Aufzählungen (nur in MySQL)

3.2 Constraints

Wenn Tabellen erzeugt werden, sind Angaben für **Constraints** zu diesen Tabellen sinnvoll, um Anforderungen für die Wertbereiche der Spalten über die Datentyp-Angaben hinaus festzulegen. Ein Constraint ist eine Regel für eine Spalte bezogen auf die Gesamtsicht der Tabelle.
So ist eine UNIQUE Constraint geeignet, um sicherzustellen, dass keine zwei Sätze der gleichen Tabelle den gleichen Wert in derselben Spalte haben dürfen.
Andere Constraints sind NOT NULL (d.h. kein Wert in der Spalte darf leer gelassen werden), PRIMARY KEY und FOREIGN KEY. PRIMARY

KEY Constraint definiert eine eindeutige Identifikation jedes Satzes (Zeile) einer Tabelle. FOREIGN KEY sind solche Felder einer Tabelle, die in einer anderen Tabelle den PRIMARY KEY darstellen.
Auch die Vergabe von Default-Werte mittels des Schlüsselwortes DEFAULT gehört zu den Constraints.
Mittels des Constraints CHECK lassen sich Gültigkeitsregeln für einzelne Datenfelder definieren.

Beispiele:

```
CREATE TABLE Kunden
(KNr integer NOT NULL,
Vorname varchar(15),
Nachname varchar(20),
KAlter integer,
Adresse varchar(30),
PLZ varchar(10),
Stadt varchar(20),
CONSTRAINT agecheck CHECK (KAlter >=16),
PRIMARY KEY(KNr));

CREATE TABLE Rechnungen
(RgNr integer NOT NULL,
KdNr integer,
RgDatum date,
RgBetrag double,
CONSTRAINT fk_Kunde FOREIGN KEY KdNr REFERENCES Kunden (KNr),
PRIMARY KEY(RgNr));
```

3.3 Übung zu CREATE TABLE

Legen Sie eine neue Tabelle Telefonbuch mit den Spalten pid, PersNr, LandCode, Vorwahl und TelefonNr.
Als Data Types verwenden Sie für pid und PersNr integer, für die übrigen Spalten varchar(10).
Das Feld pid ist der Primärschlüssel. Um diesen nicht immer selbst eingeben zu müssen, soll die Autoincrement-Funktion[2] genutzt werden.
Das Feld LandCode soll als Vorgabe bzw. Default mit dem Wert '0049' gefüllt werden.

Legen Sie eine weitere Tabelle Programmierer an mit den Felder PersNr und Position.
Das Feld PersNr ist vom Datentyp integer, darf keine NULL-Werte enthalten und stellt den Primärschlüssel dar, das Feld Position ist vom Datentyp varchar(50).

[2] Autoincrement ist nicht in allen Datenbanken verfügbar, Oracle z.B. kennt diese Konstrukt in der Form nicht!

3.4 Lösung

MySQL:

```
CREATE TABLE Telefonbuch
(pid integer auto_increment NOT NULL,
 PersNr integer,
 LandCode varchar(10) default '0049',
 Vorwahl varchar(10),
 TelefonNr varchar(10),
 primary key(pid));
```

Access:

```
CREATE TABLE Telefonbuch
(pid integer NOT NULL,
 PersNr integer,
 LandCode varchar(10),
 Vorwahl varchar(10),
 TelefonNr varchar(10),
 primary key(pid));


CREATE TABLE Programmierer
(PersNr integer NOT NULL,
 Position varchar(50),
 primary key(PersNr));
```

4 SHOW: Informationen über Tabellen und Datenbanken

Um Informationen über die in MySQL vorhandenen Datenbanken und deren Tabellen zu erlangen, bedient man sich des Befehls SHOW.

Format:

SHOW DATABASES|TABLES|COLUMNS
FROM *tablename* [FROM *databasename*]

[ ] = optional

Beispiele:

1) Alle Datenbanken des DBMS MySQL anzeigen

```
SHOW DATABASES;
```

2) Alle Tabellen der Datenbank Mitarbeiter anzeigen

```
SHOW TABLES FROM Mitarbeiter;
```

3) Alle Spalten der Tabelle MAStamm der Datenbank Mitarbeiter anzeigen[3]

```
SHOW COLUMNS FROM MAStamm [FROM Mitarbeiter];
```

[3] Anstelle des nachfolgenden SHOW-Befehls kann unter MySQL hier auch die Anweisung DESC MAStamm verwendet werden; dieser Befehl existiert auch in anderen DBMS wie Oracle.

5 SELECT: Abfragen über eine Tabelle

Die **SELECT**-Anweisung wird zur Abfrage einer Datenbank und zur Auswahl von Daten, die bestimmte Kriterien zu erfüllen haben, benutzt. Das Format der Anweisung ist wie folgt:

SELECT [DISTINCT] *column1* [, *column2*, ...]
FROM *tablename*
[GROUP BY *groupcolumn*]
[WHERE *condition*]
[ORDER BY *ordercolumn* ASC|DESC];

[ ] = optional

- Die Spaltennamen, die dem Schlüsselwort **SELECT** folgen, geben an, welche Spalten in den Ergebnissen berücksichtigt werden sollen. Man kann beliebig viele (existierende) Spaltennamen oder "*" für sämtliche Spalten (=columns) einer Tabelle verwenden.
- Der Tabellen-Name, der dem Schlüsselwort **FROM** folgt, spezifiziert die Tabelle, die für die Anfrage der erwünschten Ergebnisse durchsucht werden soll.
- Die **WHERE**-Klausel (optional) spezifiziert, welche Werte in der Tabelle bzw. welche Zeilen angezeigt bzw. ausgewählt werden sollen, wobei die Einzelheiten hinter dem Schlüsselwort **WHERE** folgen.

Bedingungsangaben in der **WHERE**-Klausel:

=	Gleichheit
>	Größer als
<	Kleiner als
>=	Größer als oder gleich
<=	Kleiner als oder gleich
<>	Nicht gleich
LIKE	Gleichheit bei Textvergleich

- Der **LIKE**-Operator ist ziemlich mächtig und erlaubt die Angabe von Vergleichsmustern. Das Prozentzeichen "%" wird als Substitut (wild card) benutzt, um irgendwelche möglichen Zeichen zu repräsentieren, die vor oder hinter spezifisch angegebenen Zeichen

in einer Auswahl stehen sollen. Neben dem Zeichen %, das für kein oder beliebig viele Zeichen steht, existiert noch der Paltzhalter _, der genau ein anderes Zeichen ersetzt.[4]

- Strings müssen in **single quotes** ' stehen.[5]
- Das optionale Schlüsselwort **DISTINCT** bewirkt, dass keine doppelten Datensätze ausgegeben werden.

Beispiele:

Für die nachfolgenden Beispiele und Aufgaben wird die Tabelle *MAStamm* verwendet:

PersNr	Vorname	Nachname	PersAlter	Stadt	Position	Eintritt
99980	John	Jones	45	Payson	CEO	01.02.1989
99982	Mary	Jones	25	Payson	Programmierer	15.03.2001
88232	Eric	Edwards	32	San Diego	Programmierer	01.06.1997
88233	Mary Ann	Edwards	32	Phoenix	Programmierer II	15.12.1995
98002	Ginger	Howell	42	Cottonwood	Programmierer II	15.07.1991
92001	Sebastian	Smith	23	Los Angeles	Assistent	02.01.2002
22322	Gus	Gray	35	Bagdad	Programmierer	01.08.1993
32326	Mary Ann	May	52	Tucson	Programmierer	15.10.1976
32327	Erica	Williams	60	Show Low	Programmierer II	01.11.1970
32380	Leroy	Brown	22	Pinetop	Assistent	02.01.2002
32382	Elroy	Cleaver	22	Los Angeles	Assistent	15.06.2000

1) Die folgende SQL-Anweisung gibt alle Vornamen, Nachnamen und die Stadt aus der Tabelle *MAStamm* aus, bei denen der Vorname mit 'Er' beginnt.

```
SELECT Vorname, Nachname, Stadt
FROM MAStamm
WHERE Vorname LIKE 'Er%';
```

[4] In MS Access müssen anstelle der Zeichen _ und % die Platzhalter ? und * verwendet werden, dies entspricht jedoch nicht dem SQL-Standard und führt daher normalerweise zu falschen Ergebnissen.

[5] Manche DBMS wie MS Access oder MySQL erlauben auch den Gebrauch von doppelten Anführungszeichen “.

2) Um alle Nachnamen zu erhalten, die mit 's' enden, dient folgende Anweisung:

```
SELECT Vorname, Nachname
FROM MAStamm
WHERE Nachname LIKE '%s';
```

3) Die SQL-Anweisung sucht nur Datensätze aus, die im Feld Vorname exakt den Wert 'Eric' enthalten.

```
SELECT *
FROM MAStamm
WHERE Vorname = 'Eric';
```

4) Die SQL-Anweisung sucht nur Datensätze aus, die am 01.11.1970 in die Firma eingetreten sind. Datumswerte sind dabei in der amerikanischen Schreibweise anzugeben.

```
SELECT *
FROM MAStamm
WHERE Eintritt = '1970-11-01';
```

5) Die SQL-Anweisung alle Städte aus, in denen Mitarbeiter wohnen; jede Stadt wird nur einmal ausgegeben.

```
SELECT DISTINCT Stadt
FROM MAStamm;
```

6) Ausgabe der drei ältesten Mitarbeiter.

```
In MySQL:
SELECT PersAlter
FROM MAStamm
ORDER BY PersAlter DESC
LIMIT 0, 3;

In Access:
SELECT TOP 3 PersAlter
FROM MAStamm
ORDER BY PersAlter DESC;

In Oracle:
SELECT ROWNUM,PersAlter FROM
(SELECT PersAlter FROM MAStamm ORDER BY PersAlter DESC)
WHERE ROWNUM <= 3;
```

5.1 Aggregatfunktionen

Mittels Aggregat- bzw. Gruppierungsfunktionen lassen sich zu bestimmten Feldern einer Tabelle statistische Aussagen wie Summe, Durchschnitt oder maximaler bzw. minimaler Wert berechnen.

Beispiele:

1) „Gesamtalter“ aller Mitarbeiter:

```
SELECT SUM(PersAlter)
FROM MAStamm;
```

2) Durchschnittsalter aller Mitarbeiter:

```
SELECT AVG(PersAlter)
FROM MAStamm;
```

3) Anzahl der verschiedenen Städte, aus denen die Mitarbeiter stammen sortiert nach den Städten:

```
SELECT Stadt, COUNT(Stadt)
FROM MAStamm
GROUP BY Stadt
ORDER BY Stadt;
```

5.2 Berechnete Felder in SELECT-Abfragen

Manchmal ist es sinnvoll, bestimmte Aussagen bzw. Werte auszugeben, die sich aus den Informationen/Feldern einer Tabelle berechnen lassen, ohne diese auch in der Datenbank abzuspeichern. Hierzu gibt es, abhängig vom jeweils verwendeten DBMS, unterschiedliche Funktionen.

Beispiele:

1) Ausgabe von Vorname und Nachname der Mitarbeiter in der Form Nachname, Vorname mit der Spaltenüberschrift VName:

```
In MySQL:
SELECT CONCAT(Nachname, ', ', Vorname) AS VName
FROM MAStamm;

In Oracle:
SELECT Nachname || ', ' || Vorname AS VName
FROM MAStamm;

In Access:
SELECT Nachname + ', ' + Vorname AS VName
FROM MAStamm;
```

2) Ausgabe der Mitarbeiter mit ihrem Alter in 5 Jahren:

```
SELECT Vorname, Nachname, PersAlter + 5 as AlterNeu
FROM MAStamm;
```

3) Ausgabe der Mitarbeiter mit der Dauer der Firmenzugehörigkeit in Jahren:

```
SELECT PersNr, Eintritt,
       YEAR(NOW())-YEAR(Eintritt) AS Dauer
FROM MAStamm;
```

5.3 Vorübung zu SELECT

Geben Sie folgende SELECT-Anweisungen ein. Vor der Ausführung der Anweisung, schreiben Sie sich die erwarteten Ergebnisse auf und vergleichen nach der Ausführung der Anweisung die Ergebnisse.

```
SELECT Vorname, Nachname, Stadt FROM MAStamm;
```

```
SELECT Nachname, Stadt, PersAlter FROM MAStamm
  WHERE PersAlter > 30;
```

```
SELECT Vorname, Nachname, Stadt FROM MAStamm
  WHERE Vorname LIKE 'J%';
```

```
SELECT * FROM MAStamm;
```

```
SELECT Vorname, Nachname FROM MAStamm
  WHERE Nachname LIKE '%s';
```

```
SELECT Vorname, Nachname, PersAlter FROM MAStamm
  WHERE Nachname LIKE '%illia%';
```

```
SELECT * FROM MAStamm WHERE Vorname = 'Eric';
```

5.4 Übungen zu SELECT

Erstellen Sie folgende SELECT-Anweisungen:

1) Auswahl der Spalten Vorname und PersAlter für alle Mitarbeiter in der Tabelle MAStamm.

2) Auswahl der Spalten Vorname, Nachname und Stadt für alle, die nicht aus Payson kommen.

3) Auswahl aller Spalten für diejenigen, die über 40 Jahre alt sind.

4) Auswahl der Spalten Vorname und Nachname für alle Mitarbeiter, deren Nachname mit "ay" endet.

5) Auswahl aller Spalten für diejenigen, deren Vorname gleich "Mary" ist.

6) Auswahl aller Spalten für diejenigen, deren Vorname "Mary" enthält.

7) Anzahl der Beschäftigten pro Wohnort (Stadt).

5.5 Antworten

1) Auswahl der Spalten Vorname und PersAlter für alle Mitarbeiter in der Tabelle MAStamm.

```
SELECT Vorname, PersAlter FROM MAStamm;
```

2) Auswahl der Spalten Vorname, Nachname und Stadt für alle, die nicht aus Payson kommen.[6]

```
SELECT Vorname, Nachname, Stadt FROM MAStamm
   WHERE Stadt NOT LIKE 'Payson';
```

3) Auswahl aller Spalten für diejenigen, die über 40 Jahre alt sind.

```
SELECT * FROM MAStamm
  WHERE PersAlter > 40;
```

4) Auswahl der Spalten Vorname und Nachname für alle Mitarbeiter, deren Nachname mit "ay" endet.

```
SELECT Vorname, Nachname FROM MAStamm
  WHERE Nachname LIKE '%ay';
```

5) Auswahl aller Spalten für diejenigen, deren Vorname gleich "Mary" ist.[7]

```
SELECT * FROM MAStamm
  WHERE Vorname LIKE 'Mary';
```

6) Auswahl aller Spalten für diejenigen, deren Vorname "Mary" enthält.

```
SELECT * FROM MAStamm
  WHERE Vorname LIKE '%Mary%';
```

7) Anzahl der Beschäftigten pro Wohnort (Stadt).[8]

```
SELECT Stadt, COUNT(PersNr)
  FROM MAStamm
  GROUP BY Stadt;
```

[6] Die WHERE-Bedingung kann hier auch wie folgt lauten:

```
WHERE Stadt <> 'Payson'
```

[7] Die WHERE-Bedingung kann hier auch wie folgt lauten:

```
WHERE Vorname = 'Mary'
```

[8] Anstelle des Feldes PersNr kann in der COUNT-Funktion jedes andere Feld sowie das Zeichen * verwendet werden.

6 JOIN: SELECT-Abfragen über mehrere Tabellen

Zur Erläuterung der nachfolgenden Aussagen wird neben der bereits bekannten Tabelle *MAStamm* noch die Tabelle *Telefonbuch* benötigt.

pid	PersNr	LandCode	Vorwahl	TelefonNr
1	92001	0049	979	1234
2	98002	0033	898	9876
3	32382	0049	221	7654

Die folgende SQL-Anweisung gibt alle Vornamen, Nachnamen und die Stadt aus der Tabelle *MAStamm* mit den dazugehörigen Werten aus der Tabelle *Telefonbuch*.

```
SELECT Vorname, Nachname, Stadt, LandCode, Vorwahl, TelefonNr
FROM MAStamm, Telefonbuch
WHERE MAStamm.PersNr = Telefonbuch.PersNr
ORDER BY MAStamm.PersNr;
```

Anstelle dieses Befehls, bei dem zwei Tabellen über die WHERE-Bedingung verknüpft werden, sollte sinnvollerweise ein anderes SQL-Statement, die JOIN-Anweisung, verwendet werden. Das Format der JOIN-Anweisung ist dabei wie folgt:

SELECT [DISTINCT] *column1* [, *column2*, ...]
FROM *tablename1* [INNER] JOIN *tablename2*
ON *tablename1.columnX* = *tablename2.columnY*
[WHERE *condition*];

[ ] = optional

Mit Hilfe eines JOINs erzielt man durch die folgende Anweisung dasselbe Resultat wie oben

```
SELECT Vorname, Nachname, Stadt, LandCode, Vorwahl, TelefonNr
FROM MAStamm INNER JOIN Telefonbuch
ON MAStamm.PersNr = Telefonbuch.PersNr
ORDER BY MAStamm.PersNr;
```

Durch diese Anweisung erhält man nur solche Datensätze, auf die die mit dem Schlüsselwort ON formulierte Bedingung zutrifft, d.h. man erhält nur die Mitarbeiter aus der Tabelle *MAStamm*, für die auch ein Eintrag in der Tabelle *Telefonbuch* existiert.

Um nun aber alle Datensätze aus *MAStamm* und bei den Datensätzen, zu denen auch ein Eintrag in der Tabelle *Telefonbuch* existiert,

zusätzlich die Telefonnummer auszugeben, muss die JOIN-Anweisung wie folgt leicht verändert werden:

```
SELECT Vorname, Nachname, Stadt, LandCode, Vorwahl, TelefonNr
FROM MAStamm LEFT OUTER JOIN Telefonbuch
ON MAStamm.PersNr = Telefonbuch.PersNr
ORDER BY MAStamm.PersNr;
```

Neben LEFT OUTER JOIN existieren auch noch die Anweisungen RIGHT OUTER JOIN und FULL OUTER JOIN[9].

[9] FULL OUTER JOIN ist in MySQL derzeit nicht implementiert.

7 ALTER: Modifizieren einer Tabelle

Die Anweisung **ALTER TABLE** ermöglicht es, eine bestehende Tabelle in ihrer Struktur zu verändern. Es lassen sich sowohl Spalten hinzufügen, löschen als auch in ihrer Definition zu ändern.

Das Format der **ALTER TABLE**-Anweisung ist wie folgt:

ALTER TABLE *tablename*

AKTION column data_type [*constraints*]

[ ] = optional

AKTION kann je nach verwendetem DBMS einer der folgenden Befehle ADD, DROP, RENAME, CHANGE, MODIFY sein.

Beispiele:

1) Es wird eine Spalte *Staat* zur Tabelle *MAStamm* hinzugefügt. Der Data Type der neuen Spalte ist varchar(50), als Defaultwert wird 'Arizona' vorgegeben.

```
ALTER TABLE MAStamm
ADD Staat varchar(50) DEFAULT 'Arizona';
```

2) Die Spalte *Gehalt* wird als Data Type integer zur Tabelle *MAStamm* hinzugefügt.

```
ALTER TABLE MAStamm
ADD Gehalt integer;
```

3) Das Feld *Anrede* wird zur Tabelle *MAStamm* hinzugefügt

```
ALTER TABLE MAStamm
ADD Anrede char(4);
```

4) Die Tabelle *Telefonbuch* wird *Telefonliste* umbenannt.[10]

```
ALTER TABLE Telefonbuch
RENAME Telefonliste;
```

5) Die Länge der Spalte *Vorwahl* in der Tabelle *Telefonliste* wird von 10 auf 15 geändert.

```
In MySQL:
ALTER TABLE Telefonliste
```

[10] In Oracle ist ein Umbenennen einer Tabelle so nicht möglich. Hier muss eine neue Tabelle mit dem gewünschten Namen angelegt und die Datensätze aus der alten in die neue Tabelle kopiert werden; anschließend wird die alte Tabelle dann noch gelöscht.

```
CHANGE Vorwahl Vorwahl varchar(15);

In Oracle:
ALTER TABLE Telefonliste
MODIFY (Vorwahl varchar2(15));
```

6) Der Typ der Spalte *Gehalt* in der Tabelle *MAStamm* wird von integer auf double geändert.

```
ALTER TABLE MAStamm
CHANGE Gehalt Gehalt double;
```

7) Es soll eine Beziehung zwischen den Feldern *PersNr* der beiden Tabellen *Telefonliste* und *MAStamm* angelegt werden.

```
ALTER TABLE Telefonliste
ADD CONSTRAINT FOREIGN KEY (PersNr)
REFERENCES MAStamm(PersNr);
```

8) Das Feld *Anrede* wird aus der Tabelle *MAStamm* gelöscht

```
In MySQL:
ALTER TABLE MAStamm
DROP Anrede;

In Oracle:
ALTER TABLE MAStamm
DROP COLUMN Anrede;
```

8 INSERT: Einfügen von Datensätzen

INSERT wird zum Einfügen eines Datensatzes (d.h. einer Zeile) in eine Tabelle benutzt.

Format:

INSERT INTO *tablename*
(*Vorname_column*, ..., *Nachname_column*)
VALUES
(*Vorname_value*, ..., *Nachname_value*);

[ ] = optional

Beispiel:

```
INSERT INTO MAStamm
 (PersNr, Vorname, Nachname, PersAlter, Position, Stadt)
VALUES
 (22, 'Luke', 'Duke', 45, 'Assistent', 'Tucson');

INSERT INTO MAStamm
VALUES
 (4711, 'Schmitz', 'Jupp', 55, 'Köln', 'Hausmeister', '1977-11-
11', 'NRW', 1111.11);
```

- Anmerkung: Strings und Datumswerte sind zwischen single quotes: 'string' einzuschließen, Zahlen dagegen nicht, da sie sonst als string verstanden würden.

Es ist ebenfalls möglich, Datensätze aus einer bestehenden Tabelle ganz oder teilweise in eine andere Tabelle einzufügen.

Beispiel:

```
INSERT INTO Programmierer
 (PersNr, Position)
SELECT PersNr, Position
 FROM MAStamm WHERE Position LIKE 'Prog%';
```

8.1 Übungen zu INSERT

1. Sie wollen Daten in die Tabelle MAStamm einfügen:

PersNr	Vorname	Nachname	Eintritt	PersAlter	Position
11111	Jonie	Weber	15.03.2003	28	Assistent
11122	Potsy	Weber	01.02.2000	32	Programmierer
11133	Dirk	Smith	03.07.1998	45	Programmierer II

2. Dann prüfen Sie mit SELECT:
   - 2.1. Alle Spalten für jeden Mitarbeiter in der Tabelle.
   - 2.2. Alle Spalten für jeden Mitarbeiter, die nach dem 31.12.1999 in die Firma eingetreten sind.
   - 2.3. Vorname und Nachname für alle Mitarbeiter jünger als 30 Jahre.
   - 2.4. Vorname, Nachname, und Gehalt für alle mit "Programmierer" in ihrer Position.
   - 2.5. Alle Spalten für alle, deren Nachname den Teilstring "ebe" enthält.
   - 2.6. Vorname für alle, deren Vorname "Potsy" lautet.
   - 2.7. Alle Spalten für alle, die 45 Jahre oder älter sind.
   - 2.8. Alle Spalten für diejenigen, deren Nachname mit "ith" endet.

8.2 Antworten

1.

```
INSERT into MAStamm
 (PersNr, Vorname, Nachname, Eintritt, PersAlter, Position)
VALUES
 (11111, 'Jonie', 'Weber', '2003-03-15', 28, 'Assistent');

INSERT into MAStamm
(PersNr, Vorname, Nachname, Eintritt, PersAlter, Position)
VALUES
 (11122, 'Potsy', 'Weber', '2000-02-01', 32, 'Programmierer');

INSERT into MAStamm
(PersNr, Vorname, Nachname, Eintritt, PersAlter, Position)
VALUES
 (11133, 'Dirk', 'Smith', '1998-07-03', 45, 'Programmierer II');
```

2.

2.1. Alle Spalten für jeden Mitarbeiter in der Tabelle.

```
SELECT * FROM MAStamm;
```

2.2. Alle Spalten für jeden , die nach dem 31.12.1999 in die Firma eingetreten sind.

```
SELECT * FROM MAStamm
WHERE Eintritt > '1999-12-31';
```

2.3. Vorname und Nachname für alle Mitarbeiter jünger als 30 Jahre.

```
SELECT Vorname, Nachname FROM MAStamm
WHERE PersAlter < 30;
```

2.4. Vorname, Nachname, und Gehalt für alle mit "Programmierer" in ihrer Position.

```
SELECT Vorname, Nachname, Gehalt FROM MAStamm
WHERE Position like '%Programmierer%';
```

2.5. Alle Spalten für alle, deren Nachname den Teilstring "ebe" enthält.

```
SELECT * FROM MAStamm
WHERE Nachname like '%ebe%';
```

2.6. Vorname für alle, deren Vorname "Potsy" lautet.

```
SELECT Vorname FROM MAStamm
WHERE Vorname = 'Potsy';
```

2.7. Alle Spalten für alle, die 45 Jahre oder älter sind.

```
SELECT * FROM MAStamm
WHERE PersAlter >= 45;
```

2.8. Alle Spalten für diejenigen, deren Nachname mit "ith" endet.

```
SELECT * FROM MAStamm
WHERE Nachname like '%ith';
```

9 UPDATE: Ändern von Datensätzen

Die **UPDATE**-Anweisung ändert oder aktualisiert Zeilen, die die angegebenen Kriterien erfüllen. Die Kriterien müssen sorgfältig in der WHERE-Klausel definiert sein.

Format:

UPDATE *tablename*
SET *columnname* = *newvalue* [, *nextcolumn* = *newvalue2,* ...]
WHERE *columnname* OPERATOR *newvalue*
[AND|OR *columnname* OPERATOR *newvalue*];

[ ] = optional

Beispiele:

```
UPDATE Telefonliste
  SET LandCode = ’623’
  WHERE Vorwahl = ’979’;

UPDATE Telefonliste
  SET PersNr = 88232,
      Vorwahl = ’555’,
      TelefonNr = ’9292’
  WHERE PersNr =77777;

UPDATE MAStamm
  SET PersAlter = PersAlter + 1
  WHERE Vorname LIKE 'Erica'
   AND Nachname LIKE 'Williams';

UPDATE MAStamm
  SET Gehalt = Gehalt * 1.1
  WHERE PersNr = 4711;

UPDATE MAStamm
  SET Gehalt = 25000
  WHERE Position LIKE 'Assistent';

UPDATE MAStamm
  SET Gehalt = 25000
  WHERE Position LIKE 'Assistent';

UPDATE MAStamm
  SET Gehalt = 35000
  WHERE Position LIKE 'Programmierer%';
```

9.1 Übungen zu UPDATE

(nach jedem UPDATE sollte der Effekt sofort mittels SELECT geprüft werden)

1) Jonie Weber hat Bob Williams geheiratet. Sie heißt nun mit Nachnamen Weber-Williams.

2) Dirk Smith hat heute Geburtstag, das Alter ist anzupassen.

3) Alle Assistenten sollen jetzt "Administrativer Assistent" genannt werden.

4) Alle mit dem Titel "CEO" verdienen ab sofort 67000,00 EUR.

5) Jeder mit einem Gehalt unter 30000 soll 3500 mehr verdienen.

6) Jeder mit einem Gehalt über 33500 soll 4500 mehr verdienen.

7) Alle mit dem Titel "Programmierer II" sollen "Programmierer III" werden.

8) Alle mit dem Titel "Programmierer" werden "Programmierer II".

9.2 Antworten

1) Jonie Weber hat Bob Williams geheiratet. Sie möchte ihren Nachnamen mit Weber-Williams eintragen lassen.

```
UPDATE MAStamm
SET Nachname = 'Weber-Williams'
WHERE Vorname = 'Jonie' and Nachname = 'Weber';
```

2) Dirk Smith hat heute Geburtstag, das Alter ist anzupassen.

```
UPDATE MAStamm
SET PersAlter = PersAlter + 1
WHERE Vorname = 'Dirk' and Nachname = 'Smith';
```

3) Alle Assistenten sollen jetzt "Administrativer Assistent" genannt werden.

```
UPDATE MAStamm
SET Position = 'Administrativer Assistent'
WHERE Position = 'Assistent';
```

4) Alle mit dem Titel "CEO" verdienen ab sofort 67000,00 EUR.

```
UPDATE MAStamm
SET Gehalt = 67000
WHERE Position = 'CEO';
```

5) Jeder mit einem Gehalt unter 30000 soll 3500 mehr verdienen.

```
UPDATE MAStamm
SET Gehalt = Gehalt + 3500
WHERE Gehalt < 30000;
```

6) Jeder mit einem Gehalt über 33500 soll 4500 mehr verdienen.

```
UPDATE MAStamm
SET Gehalt = Gehalt + 4500
WHERE Gehalt > 33500;
```

7) Alle mit dem Titel "Programmierer II" sollen "Programmierer III" werden.

```
UPDATE MAStamm
SET Position = 'Programmierer III'
WHERE Position = 'Programmierer II';
```

8) Alle mit dem Titel "Programmierer" werden "Programmierer II".

```
UPDATE MAStamm
SET Position = 'Programmierer II'
WHERE Position = 'Programmierer';
```

10 DELETE: Löschen von Datensätzen

Format:

DELETE FROM *tablename*
WHERE *columnname* OPERATOR *newvalue*
[AND|OR *columnname* OPERATOR *newvalue*];

[ ] = optional

Beispiele:

```
DELETE FROM Telefonliste;

DELETE FROM MAStamm
  WHERE Nachname = 'May';

DELETE FROM MAStamm
  WHERE Vorname = 'Mike' or Vorname = 'Eric';
```

- Um ganze Zeilen zu löschen, wird DELETE FROM mit dem Tabellen-Namen und der genauen WHERE-Klausel gewählt.

- Wird keine WHERE-Klausel genutzt, so löscht DELETE den ganzen Tabelleninhalt, jedoch nicht die Tabellendefinition.

10.1 Übungen zu DELETE

1) Jonie Weber-Williams hat gekündigt.

2) Alle Mitarbeiter mit einem Gehalt über 70000 werden gefeuert.

Antworten

1) Jonie Weber-Williams hat gekündigt.

```
DELETE FROM MAStamm
WHERE Nachname = 'Weber-Williams';
```

2) Alle Mitarbeiter mit einem Gehalt über 70000 werden gefeuert.

```
DELETE FROM MAStamm
WHERE Gehalt > 70000;
```

11 DROP: Löschen von Tabellen und Datenbanken

Format:

DROP TABLE *tablename* | DATABASE *databasename*

Beispiele:

```
DROP TABLE Programmierer;

DROP DATABASE Kunden;
```

DROP TABLE hat eine andere Wirkung als **DELETE TABLE** ohne WHERE-Klausel, da hierbei auch die Tabellen-Definition gelöscht wird.

Übung:

1) Löschen Sie die Tabelle *Telefonliste*.

12 CHECK/OPTIMIZE: Tabellen überprüfen und komprimieren

Mittels CHECK kann eine Tabelle auf Inkonsistenzen oder Fehler hin überprüft werden.
Format:

CHECK TABLE *tablename* [Option]

Option kann die Werte QUICK, FAST, CHANGED und EXTENDED enthalten.
Enthält eine Tabelle Fehler, so können diese mit dem MySQL-Befehl *myisamcheck* repariert werden.

Beispiel:

```
CHECK TABLE MAStamm EXTENDED;
```

Mittels DELETE gelöschte Datensätze werden nur als gelöscht markiert, jedoch nicht physikalisch gelöscht. Um dies zu erreichen wird der Befehl OPTIMIZE genutzt. Hierdurch wird dann auch eine Komprimierung der Tabelle erreicht, da nun der durch die als gelöscht markierten Datensätze belegte Speicherplatz wieder freigegeben wird.
Format:

OPTIMIZE TABLE *tablename*

Beispiel:

```
OPTIMIZE TABLE MAStamm;
```

13 Anhang

13.1 MySQL - Befehle des Textclients

Die folgenden Befehle können an der Kommandozeile des textbasierten Clients von MySQL verwendet werden; jede Befehlszeile ist dabei wie die SQL-Statements mit einem ; abzuschließen.

help	(\h)	Zeigt die Hilfe an
?	(\?)	Synonym für `help'.
clear	(\c)	Löscht einen Befehl
connect	(\r)	(erneutes) Verbinden mit dem Datenbank-Server
ego	(\G)	SQL-Befehl ausführen, die Ausgabe erfolgt vertikal
exit	(\q)	mysql-Client beenden
go	(\g)	SQL-Befehl ausführen (entspricht ;)
notee	(\t)	Beendet die Protokollierung
quit	(\q)	mysql-Client beenden
source	(\.)	Ausführen von SQL-Befehlen aus einer Textdatei
status	(\s)	Zeigt diverse Informationen über den MySQL-Server an.
tee	(\T)	Protokollieren aller Ein- und Ausgaben in einer Textdatei.
use	(\u)	Anbinden einer Datenbank

14 Links und Literatur

14.1 Links

- www.mysql.com
  Neben Binärdateien und Quellcode gibt es auf der offiziellen Website von MySQL auch Dokumentationen in verschiedenen Sprachen und Dateiformaten.
- www.dynamic-webpages.de
  Alles über dynamische Websites, darunter auch viele Beispiele mit MySQL, PHP und Perl.
- www.muniqsoft.de
  Tipps & Tricks zu Oracle

14.2 Literatur

- *MySQL & mSQL*
  R.J. Yarger, G. Reese, T. King, O'Reilly-Verlag
  Grundlage und Referenz für Anwender und Administratoren von MySQL und mSQL. Mit großem Teil zur Datenbank-Programmierung
- *MySQL – Einführung, Programmierung, Referenz*
  Michael Kofler, Addison-Wesley
  In diesem Buch findet der Leser alles über MySQL von der Installation über die Administration hin zur Programmierung, Sehr umfassende Darstellung.
- *SQL – Einstieg und Anwendung*
  W.D. Misgeld, Hanser-Verlag
  Umfangreiches SQL-Handbuch mit Erweiterungen zu IBM DB2, Informix und Oracle.
- *SQL – Grundlagen und Datenbankdesign*
  P. Teich, U. Böttcher, Herdt-Verlag
  Schulungsunterlage aus dem Herdt-Verlag, bietet einen guten Einblick in die Thematik