Kapitel 2

DQL (Data Query Language) und DML (Data Manipulation Language)

2.1 Einfügen von Datensätzen

```
INSERT INTO Tabellenname [(Spaltenname [,Spaltenname ...] ) ]
    VALUES ( <Ausdruck> [, Ausdruck ...] )
```

oder

```
INSERT INTO Tabellenname [(Spaltenname [,Spaltenname ...] ) ]
    <SELECT-Abfrage>
```

2.2 Ändern von Datensätzen

```
UPDATE Tabellenname
    SET <Spaltenname>=<Ausdruck> [,<Spaltenname>=<Ausdruck> ...]
    [WHERE <Bedingung>]
```

2.3 Löschen von Datensätzen

```
DELETE FROM Tabellenname
    [WHERE <Bedingung>]
```

2.4 Abfragen von Datensätzen

```
SELECT [DISTINCT] {    | Spaltenname [AS <Alias>] [, Spaltenname [AS <Alias>]...] }
FROM Tabellenname [<Alias>] [, Tabellenname [<Alias>] ...]
WHERE <Bedingung>
GROUP BY Spaltenname [, Spaltenname ...]
HAVING <Bedingung>
ORDER BY Spaltenname [ ASC | DESC ] [, Spaltenname ...]
```

WHERE-Klausel	Diese Klausel gibt an, welche Zeilen aus den benannten Tabellen in der FROM-Klausel ausgewählt werden. Sie kann benutzt werden, um Joins zwischen mehreren Tabellen einzurichten
GROUP BY-Klausel	Sie können nach Spalten, Aliasnamen oder Funktionen gruppieren. Das Ergebnis der Abfrage enthält eine Zeile für jede unterschiedliche Menge von Werten in den benannten Spalten, Aliasse oder Funktionen. Die sich ergebenden Zeilen werden oftmals auch Gruppen genannt. Für diese Gruppen können Aggregatfunktionen (COUNT, MAX, MIN, SUM, AVG) angewendet werden.
HAVING-Klausel	Diese Klausel wählt die Zeile auf der Grundlage der Gruppenwerte und nicht der einzelnen Zeilenwerte aus. Die HAVING-Klausel kann nur verwendet werden, wenn entweder die Anweisung eine GROUP BY-Klausel hat oder die Auswahlliste nur aus Aggregatfunktionen besteht. Sämtliche Spaltennamen, die in der HAVING-Klausel referenziert werden, müssen entweder in der GROUP BY-Klausel enthalten sein oder in der HAVING-Klausel als ein Parameter für eine Aggregatfunktion verwendet werden.
ORDER BY-Klausel	Diese Klausel sortiert die Ergebnisse einer Abfrage. Jedes Element in der ORDER BY-Liste kann als ASC für aufsteigende Sortierfolge (der Standardwert) oder DESC für absteigende Sortierfolge benannt werden.

2.4.1 Aggregatfunktionen

Eine Aggregatfunktion berechnet ein einzelnes Ergebnis aus mehreren Eingabezeilen. Zum Beispiel gibt es Aggregatfunktionen, die die Anzahl der Zeilen (`count`), die Summe der Eingabewerte (`sum`), den Durchschnitt (`avg`), die Minimal- (`min`) oder den Maximalwert (`max`) jeweils aus mehreren Zeilen berechnen. Bei `count` ist zu beachten, dass dabei alle Datensätze gezählt werden, bei denen die entsprechende Spalte nicht NULL ist (Ausnahme: `count(*)`)

2.4.2 Subqueries

Subqueries sind eine andere Möglichkeit zur Formulierung von Anfragen, die mehrere Relationen umfassen. Dabei werden mehrere SELECT-Anfragen ineinander geschachtelt. Meistens stehen Subqueries dabei in der WHERE-Zeile.

```
SELECT <attr-list>
FROM <table-list>
WHERE <attribute> <rel> <subquery>;
```

wobei `<subquery>` eine SELECT-Anfrage (Subquery) ist und

- falls `<subquery>` nur einen einzigen Wert liefert, ist `rel` eine der Vergleichsrelationen $=, <, >, <=, >=$,
- falls `<subquery>` mehrere Werte/Tupel liefert, ist `rel` entweder `IN` oder von der Form Φ ANY oder Φ ALL wobei Φ eine der o.g. Vergleichsrelationen ist.

Bei den Vergleichsrelationen muss die Subquery ein einspaltiges Ergebnis liefern, bei `IN` sind im SQL-Standard und in Oracle seit Version 8 auch mehrere Spalten erlaubt.

Man unterscheidet unkorrelierte Subqueries und korrelierte Subqueries: Eine Subquery ist unkorreliert, wenn sie unabhängig von den Werten des in der umgebenden Anfrage verarbeiteten Tupels ist. Solche Subqueries dienen dazu, eine Hilfsrelation oder ein Zwischenergebnis zu bestimmen, das für die übergeordnete Anfrage benötigt wird. In diesem Fall wird die Subquery vor der umgebenden Anfrage einmal ausgewertet, und das Ergebnis wird bei der Auswertung der WHERE-Klausel der äußeren Anfrage verwendet.

Eine Subquery ist korreliert, wenn in sie von Attributwerten des gerade von der umgebenden Anfrage verarbeiteten Tupels abhängig ist. In diesem Fall wird die Subquery für jedes Tupel der umgebenden

Anfrage einmal ausgewertet.

Beispiel für eine korrelierte Abfrage:

Es existieren 2 Relationen:

```
City(Name, Country, Population, ....)
Country(Name, Population, Code, ....)
```

Es sollen alle Städte bestimmt werden, in denen mehr als ein Viertel der Bevölkerung des jeweiligen Landes wohnt:

```
SELECT Name, Country
FROM City
WHERE Population * 4 >
(SELECT Population
FROM Country
WHERE Code = City.Country);
```

Das Schlüsselwort `EXISTS` bzw. `NOT EXISTS` bildet den Existenzquantor nach. Subqueries mit `EXISTS` sind i.a. korreliert um eine Beziehung zu den Werten der äußeren Anfrage herzustellen.

```
SELECT <attr-list>
FROM <table-list>
WHERE [NOT] EXISTS
(<select-clause>);
```

Beispiel:

Gesucht seien diejenigen Länder, für die Städte mit mehr als 1 Mio. Einwohnern in der Datenbasis abgespeichert sind.

```
SELECT Name
FROM Country
WHERE EXISTS
(SELECT *
FROM City
WHERE Population > 1000000
AND City.Country = Country.Code);
```

Subqueries in der FROM-Zeile

Zusätzlich zu den bisher gezeigten Anfragen, wo die Subqueries immer in der WHERE-Klausel verwendet wurden, sind [in einigen Implementierungen; im SQL-Standard ist es nicht vorgesehen] auch Subqueries in der FROM-Zeile erlaubt.

```
SELECT <attr-list>
FROM <table/subquery-list>
WHERE <condition>;
```

Beispiel:

Gesucht ist die Zahl der Menschen, die nicht in den gespeicherten Städten leben.

```
SELECT Population-Urban_Residents
FROM
```

```
(SELECT SUM(Population) AS Population
FROM Country),
(SELECT SUM(Population) AS Urban_Residents
FROM City);
```

2.4.3 UNION, INTERSECT

Mit dem `UNION` Befehl kann man die Result Sets von zwei oder mehr `SELECT` kombinieren. Doppelte Werte werden dabei allerdings ignoriert. Bei `UNION` muss man darauf achten, dass die selektierten Spalten beider Tabellen vom gleichen Typ sind.

Die Ergebnismenge bei `INTERSECT` enthält die Schnittmenge der Teilmengen, d.h. die Datensätze müssen in beiden Abfragen enthalten sein. Oft ist dieser Operator auch durch `EXISTS` oder `IN` realisierbar.

2.5 Views

- Was ist eine View?
- Wozu dient eine View?
- Geben Sie die Syntax an, mit der eine View angelegt werden kann.
- Kann man in eine View etwas einfügen (ja/nein/evt. - welche Voraussetzungen müssen gelten)?
- Kann man eine View über eine View legen?
- Wie könnte man physisch gespeicherte Views in MariaDB erzeugen?

2.6 Aufgabe

Entwickeln Sie die nötigen SQL-Anweisungen, um folgende Aufgaben durchzuführen:

- Fügen Sie zur Relation `Kunde` einige Datensätze ein.(Es soll dabei einen Kunden mit einer `ID=1` geben und auch einige, die einen negativen Kontostand haben)
- Erstellen Sie eine neue Relation `Kunde_Mahnung` mit den gleichen Attributen wie die Relation `Kunde`. Fügen Sie anschließend jene Tupel der Relation `Kunde` in die Relation `Kunde_Mahnung` ein, die einen negativen Kontostand besitzen.
- Der Weinhändler möchte jene Kunden, die ihm kein Geld schulden (`Kontostand`$\geq$`0`), belohnen und schreibt ihnen einen Betrag von 10 Euro auf ihrem Kontostand gut.
- Der Kunde mit der `KundenID=1` hat auf die Mahnung des Weinhändlers bezahlt, hat jedoch versichert, dass er nie wieder Kunde bei ihm sein wird. Löschen Sie diesen Kunden aus der Relation `Kunde`.
- Löschen Sie alle Tupel der Relation `Kunde_Mahnung` und danach die gesamte Relation.

2.7 Aufgabe

In einem Weinkeller werden viele Weinflaschen (`WEIN`) gelagert. Jede Weinflasche wurde von einem Winzer (`WINZER`) befüllt und kann anhand der Datenbank leicht in den Regalreihen (`KELLER`) des Kellers gefunden werden. Es kann dabei angenommen werden, dass immer ausreichend Platz in den Regalen des Kellers vorhanden ist. Wird eine Flasche aus dem Keller entfernt, wird ein entsprechender Eintrag (`PROTOKOLL`) vermerkt. Wenn der Eigentümer des Kellers eine Flasche selbst trinkt, wird im Protokoll als Verwendung '`Eigenbedarf`' eingetragen.

2.7.1 Tabellen und deren Inhalt

```
SQL> SELECT * FROM winzer;
+-----+--------------------------+--------------------+------+-----------------+----------+
| wnr | name                     | strasse            | plz  | ort             | telefon  |
+-----+--------------------------+--------------------+------+-----------------+----------+
|   1 | Lackner Tinnacher        | Steinbach 8        | 4567 | Gamlitz         | 1234567  |
|   2 | Weingut Prager           | Weissenkirchen 48  | 3610 | Weissenkirchen  | 1234567  |
|   3 | Weingut Emmerich Knoll   | Unterloiben 10     | 3601 | Unterloiben     | 1234456  |
|   4 | Weingut F.X.Pichler      | Unterloiben 27     | 3601 | Unterloiben     | 11122233 |
|   5 | Weingut Spatlese         | Weintalstrasse 23  | 1136 | Wien            |          |
|   6 | Freie Weingarten Wachau  | Kremstalstrasse 23 | 3600 | Krems           | 2304002  |
|   7 | Stiftskellerei           |                    |      |                 |          |
|   8 | Weingut Biegler          | Wienerstrasse 88   | 2352 | Gumpoldskirche  | 54564565 |
+-----+--------------------------+--------------------+------+-----------------+----------+

SQL> SELECT * FROM wein;
+----+--------------------+--------------+----------+-------+--------+-----+----------+
| nr | bezeichnung        | sorte        | jahrgang | preis | anzahl | wnr | position |
+----+--------------------+--------------+----------+-------+--------+-----+----------+
|  1 | Riesling Kellerberg | Riesling     |     1999 | 28.00 |     24 |   4 |        1 |
|  2 | Loibenberg         | Gr. Veltliner |     2000 | 19.00 |     36 |   4 |        2 |
|  3 | Ried Kreutles      | Gr. Veltliner |     2000 | 19.00 |     15 |   3 |        4 |
|  4 | Riesling Smaragd   | Riesling     |     2000 | 21.00 |     30 |   2 |        5 |
|  5 | Grauburgunder      | Grauburgunder |     2003 | 16.00 |     72 |   1 |        6 |
|  6 | Morillon           | Chardonnay   |     2003 |  9.00 |     55 |   1 |        7 |
|  7 | Riesling Federspiel | Riesling     |     2003 |  9.90 |     80 |   6 |        3 |
|  8 | Chardonnay         | Chardonnay   |     2003 |  9.00 |     16 |   8 |        8 |
+----+--------------------+--------------+----------+-------+--------+-----+----------+

SQL> SELECT * FROM keller;
+-----+-------+-------+------+
| knr | reihe | regal | fach |
+-----+-------+-------+------+
|   1 |     1 |     1 |    1 |
|   2 |     1 |     1 |    2 |
|   3 |     1 |     1 |    3 |
|   4 |     1 |     2 |    1 |
|   5 |     1 |     2 |    2 |
|   6 |     2 |     1 |    1 |
|   7 |     2 |     1 |    2 |
|   8 |     2 |     2 |    1 |
+-----+-------+-------+------+

SQL> SELECT * FROM protokoll;
+-----+-----+------------+--------------------------+---------+
| pnr | nr  | pDatum     | verwendung               | anzahl  |
+-----+-----+------------+--------------------------+---------+
|   1 |   1 | 2003-01-12 | Geschenk an Herrn Berger |      12 |
|   2 |   3 | 2003-07-10 | Eigenbedarf              |       2 |
|   3 |   1 | 2003-07-23 | Eigenbedarf              |       4 |
|   4 |   6 | 2003-08-14 | Geschenk (Frau Kunz)     |       6 |
|   5 |   1 | 2003-08-27 | Glasbruch                |       1 |
|   6 |   4 | 2003-11-03 | Korkgeruch               |       1 |
|   7 |   6 | 2003-11-03 | Eigenbedarf              |       3 |
+-----+-----+------------+--------------------------+---------+
```

2.7.2 Abfragen

1. Geben Sie für die Sorte 'Riesling' die Namen aller Winzer sowie die Flaschenanzahl aus. Sortieren Sie dabei nach der Flaschenanzahl absteigend.

name	anzahl
Freie Weingarten Wachau	80
Weingut Prager	30
Weingut F.X. Pichler	24

2. Ermitteln Sie für jeden Winzer den durchschnittlichen Flaschenpreis und die Gesamtanzahl der Flaschen im Keller. Berücksichtigen Sie dabei nur Winzer, von denen bekannt ist, aus welchem Ort sie kommen. Sortieren Sie die Liste nach dem Preis absteigend.

name	durchschnittspreis	gesamtanzahl
Weingut F.X. Pichler	23.50	60
Weingut Prager	21.00	30
Weingut Emmerich Knoll	19.00	15
Lackner Tinnacher	12.50	127
Freie Weingarten Wachau	9.90	80
Weingut Biegler	9.00	16

3. Geben Sie eine Liste aller Weinbezeichnungen sowie den Namen des erzeugenden Winzers aus, von denen im Jahr 2003 keine Flasche getrunken worden ist (Verwendung in Tabelle Protokoll = 'Eigenbedarf'). Sie brauchen dabei nur Winzer berücksichtigen, von denen mindestens eine Flasche im Keller vorhanden ist.

nr	bezeichnung	name
2	Loibenberg	Weingut F.X. Pichler
4	Riesling Smaragd	Weingut Prager
5	Grauburgunder	Lackner Tinnacher
7	Riesling Federspiel	Freie Weingarten Wachau
8	Chardonnay	Weingut Biegler

4. Suchen Sie die Winzer, von denen der Kellereigentümer die meisten Flaschen getrunken (Verwendung in Tabelle Protokoll = 'Eigenbedarf') hat. Geben Sie jeweils den Namen des Winzers sowie die Gesamtkosten des von diesem Winzer konsumierten Weines aus.

name	anzahl	kosten
Weingut F.X. Pichler	4	112.00

5. Geben Sie für jeden Winzer aus,
   wie viele günstige (Preis $\leq$ 10 Euro, Preisklasse niedrig),
   wie viele im Mittelfeld (10 Euro - 20 Euro, Preisklasse mittel) und
   wie viele teure (Preis $>$ 20 Euro, Preisklasse gehoben) Weinflaschen im Keller liegen.

name	anzahl	preisklasse
Freie Weingarten Wachau	80	niedrig
Lackner Tinnacher	55	niedrig
Lackner Tinnacher	72	mittel
Weingut Biegler	16	niedrig
Weingut Emmerich Knoll	15	mittel
Weingut F.X. Pichler	24	gehoben
Weingut F.X. Pichler	36	mittel
Weingut Prager	30	gehoben

6. Erstellen Sie eine Liste, die angibt, wie viele Flaschen jedes in der Datenbank gespeicherten Winzers sich im Keller befinden. Sortieren Sie dabei nach der Flaschenanzahl absteigend.

name	anzahl
Lackner Tinnacher	127
Freie Weingarten Wachau	80
Weingut F.X. Pichler	60
Weingut Prager	30
Weingut Biegler	16
Weingut Emmerich Knoll	15
Weingut Spatlese	
Stiftskellerei	