# Inhaltsverzeichnis



# 1 Vorbemerkungen

Bisher haben wir Datenbanken nur über einzelne Tabellen kennen gelernt. Stehen mehrere Tabellen in gewissen Beziehungen zur Beschreibung einer bestimmten Situation, so spricht man von einer **relationalen Datenbank** (zum relationalen Datenbankmodell folgen später mehr Informationen).
Damit Datenbanken geeignet benutzt werden können, muss dafür eine betriebssystem- und plattformunabhängige Möglichkeit zur Abfrage, Erstellung und Manipulation vorhanden sein, dies insbesondere im Hinblick auf die Erfordernisse des Internets, wo *Datenbankserver* die vielfältigsten Anfragen zu Informationen aus Datenbanken verarbeiten müssen.
Als Quasi-Standard hat sich als Abfragesprache **SQL** (Structured Query Language) mittlerweile durchgesetzt. Die Ursprünge gehen auf ein IBM-Forschungsprojekt zurück. 1986 wurde

SQL als Standard von der amerikanischen Normbehörde (ANSI) übernommen, 1987 folgte die internationale Normierung (ISO). SQL ist die Sprache, mit der die meisten relationalen Datenbanken erstellt, manipuliert und abgefragt werden.
SQL lässt sich innerhalb der Programmiersprachen als Sprache der 4. Generation bezeichnen (4GL), denn sie ist im Gegensatz zu prozeduralen Sprachen der 3. Generation, die durch Datenstrukturen und Steuerstrukturen gekennzeichnet sind, nicht-prozedural und mengenorientiert. Nicht-prozedural heißt hier, dass der Fragesteller eine Frage stellt, aber keinen Algorithmus zur Lösung vorgeben muss. Es ist nicht nötig, sich aus Programmierersicht Gedanken über das Öffnen und Schließen einer Datei und das schrittweise Durchgehen geeigneter Datensätze zu machen.
Der folgende Text bezieht sich in bestimmten Aspekten auf spezielle Eigenschaften der Implementierung **MySQL**. Bei der Arbeit mit MySQL ist zu beachten, dass Texte in Hochkommata eingegeben werden. Sicher geht man, wenn man alle Feldinhalte in Hochkommata setzt. Die Feldnamen sollten mit Accent grave eingeschlossen werden. Beispiel:

```
SELECT `Name`, `Telephon` FROM `Mitarbeiter` WHERE `Name` = 'Maier';
```

# 2 Auswahl-Abfragen mit SELECT

Mit Auswahl-Abfragen sind insbesondere die Operationen *Projektion* und *Selektion* auf einer Tabelle von Interesse (Die Abfrage über mehrere Tabellen hinweg folgt später).
Die vereinfachte Syntax des `SELECT`-Befehls lautet wie folgt:

```
SELECT [DISTINCT | ALL] select_expression,... FROM tables ...
[WHERE where_definition]
[GROUP BY feld,...]
[ORDER BY feld [ASC | DESC] ,...
[LIMIT [offset,] rows]  ;
```

Am einfachsten versteht man das an Beispielen. Im Folgenden sei die Tabelle *Mitarbeiter* die Grundlage der Beispiele.

| MitarbNr | VorgesetztenNr | AbtNr | Name | GebDatum | Telephon |
|---|---|---|---|---|---|
| 1 | NULL | 3 | Christoph Reeg | 13.05.1979 | NULL |
| 2 | 1 | 1 | junetz.de | 05.03.1998 | 069-764758 |
| 3 | 1 | 1 | Uli | NULL | NULL |
| 4 | 3 | 1 | JCP | NULL | 069-764758 |
| 5 | 1 | 2 | Maier | NULL | 06196-671797 |
| 6 | 5 | 2 | Meier | NULL | 069-97640232 |

## 2.1 Selektion

1. Die kürzestmögliche `SELECT`-Anweisung ist `SELECT * FROM Mitarbeiter;`

   Es sollen alle Mitarbeiter ausgegeben werden, d.h. es wird dann einfach die Tabelle wie oben dargestellt geliefert.

2. `SELECT * FROM Mitarbeiter WHERE Name='Maier';` liefert

| MitarbNr | VorgesetztenNr | AbtNr | Name | GebDatum | Telephon |
|---|---|---|---|---|---|
| 5 | 1 | 2 | Maier | NULL | 06196-671797 |

3. `SELECT * FROM Mitarbeiter`
   `WHERE (VorgesetztenNr=1) AND (AbtNr=1);`

   liefert

| MitarbNr | VorgesetztenNr | AbtNr | Name | GebDatum | Telephon |
|---|---|---|---|---|---|
| 2 | 1 | 1 | junetz.de | 05.03.1998 | 069-764758 |
| 3 | 1 | 1 | Uli | NULL | NULL |

Mehr zu den Bedingungen für `WHERE` folgt unten im Abschnitt `where_definition`.

## 2.2 Projektion

1. `SELECT Telephon FROM Mitarbeiter;` oder

   `SELECT ALL Telephon FROM Mitarbeiter;` liefert

| Telephon |
|---|
| NULL |
| 069-764758 |
| NULL |
| 069-764758 |
| 06196-671797 |
| 069-97640232 |

2. `SELECT DISTINCT Telephon FROM Mitarbeiter;` liefert

| Telephon |
|---|
| NULL |
| 069-764758 |
| 06196-671797 |
| 069-97640232 |

   Mehrfach vorkommende Einträge werden nur einmal angezeigt.

3. `SELECT AbtNr, Name, Telephon FROM Mitarbeiter;`

   liefert

| AbtNr | Name | Telephon |
|---|---|---|
| 3 | Christoph Reeg | NULL |
| 1 | junetz.de | 069-764758 |
| 1 | Uli | NULL |
| 1 | JCP | NULL |
| 2 | Maier | 06196-671797 |
| 2 | Meier | NULL |

## 2.3 Sortierung

Mit `ORDER BY` wird festgelegt, nach welcher Spalte bzw. welchen Spalten sortiert werden soll. Mit `ASC` wird aufsteigend, mit `DESC` absteigend sortiert. Wenn nichts angegeben wird, wird aufsteigend sortiert.

1. `SELECT * FROM Mitarbeiter ORDER BY Name;` oder

   `SELECT * FROM Mitarbeiter ORDER BY Name ASC;` liefert

| MitarbNr | VorgesetztenNr | AbtNr | Name | GebDatum | Telephon |
|---|---|---|---|---|---|
| 1 | NULL | 3 | Christoph Reeg | 13.05.1979 | NULL |
| 4 | 3 | 1 | JCP | NULL | 069-764758 |
| 2 | 1 | 1 | junetz.de | 05.03.1998 | 069-764758 |
| 5 | 1 | 2 | Maier | NULL | 06196-671797 |
| 6 | 5 | 2 | Meier | NULL | 069-97640232 |
| 3 | 1 | 1 | Uli | NULL | NULL |

2. `SELECT * FROM Mitarbeiter ORDER BY Name DESC;` liefert

| MitarbNr | VorgesetztenNr | AbtNr | Name | GebDatum | Telephon |
|---|---|---|---|---|---|
| 3 | 1 | 1 | Uli | NULL | NULL |
| 6 | 5 | 2 | Meier | NULL | 069-97640232 |
| 5 | 1 | 2 | Maier | NULL | 06196-671797 |
| 2 | 1 | 1 | junetz.de | 05.03.1998 | 069-764758 |
| 4 | 3 | 1 | JCP | NULL | 069-764758 |
| 1 | NULL | 3 | Christoph Reeg | 13.05.1979 | NULL |

3. `SELECT * FROM Mitarbeiter ORDER BY GebDatum,Name;` liefert

| MitarbNr | VorgesetztenNr | AbtNr | Name | GebDatum | Telephon |
|---|---|---|---|---|---|
| 4 | 3 | 1 | JCP | NULL | 069-764758 |
| 5 | 1 | 2 | Maier | NULL | 06196-671797 |
| 6 | 5 | 2 | Meier | NULL | 069-97640232 |
| 3 | 1 | 1 | Uli | NULL | NULL |
| 1 | NULL | 3 | Christoph Reeg | 13.05.1979 | NULL |
| 2 | 1 | 1 | junetz.de | 05.03.1998 | 069-764758 |

   Da die ersten vier Mitarbeiter kein Geburtsdatum haben, wird ein zweites Sortierkriterium für *Name* eingegeben.

## 2.4 Funktionen

1. `SELECT count(*) FROM Mitarbeiter;` liefert

| count(*) |
|---|
| 6 |

2. Mit `GROUP BY` können in Verbindung mit *Gruppenfunktionen* Gruppen gebildet werden.
   `SELECT count(*), AbtNr FROM Mitarbeiter GROUP BY AbtNr;` liefert

| count(*) | AbtNr |
|---|---|
| 3 | 1 |
| 2 | 2 |
| 1 | 3 |

Näheres zu Funktionen folgt in einem eigenen Abschnitt.

## 2.5 LIMIT

Mit `LIMIT [offset,] rows` kann angegeben werden, wie viele Zeilen maximal von der `SELECT`-Anweisung zurück geliefert werden sollen. Mit `offset` kann angegeben werden, ab welcher Zeile angefangen werden soll. Man denke dabei an die Ergebnisdarstellung einer Internet-Suchmaschinenabfrage.
`SELECT * FROM Tabelle LIMIT 5,10;` liefert die Zeilen 6 bis 15 der Tabelle.
`SELECT * FROM Tabelle LIMIT 5;` liefert die ersten fünf Zeilen der Tabelle.
`SELECT * FROM Tabelle LIMIT 0,5;` liefert ebenso die ersten fünf Zeilen.

## 2.6 where_definition

### 2.6.1 Vergleichsoperatoren

Teilbedingungen können mit `AND`, `OR` und `NOT` zu Bedingungen zusammen gesetzt werden. An Vergleichsoperatoren sind in SQL folgende verfügbar:

| **Operator** | **Bedeutung** |
|---|---|
| `=` | gleich |
| `<>` oder `!=` | ungleich |
| `>` | größer |
| `<` | kleiner |
| `>=` | größer gleich |
| `<=` | kleiner gleich |
| `IS` | gleich (für Vergleich mit `NULL`) |

```
SELECT Name, AbtNr FROM Mitarbeiter
WHERE (AbtNr=1) OR (VorgesetztenNr IS NULL);
```

liefert

| Name | AbtNr | VorgesetztenNr |
|---|---|---|
| Christoph Reeg | 3 | NULL |
| junetz.de | 1 | 1 |
| Uli | 1 | 1 |
| JCP | 1 | 3 |

Achtung: NULL ist problematisch bei String-Einträgen, weil Leerstrings möglich sind.

### 2.6.2 LIKE

```
SELECT Name, Telephon FROM Mitarbeiter
WHERE (Telephon LIKE '%96-%');
```

liefert

| Name | Telephon |
|---|---|
| Maier | 06196-671797 |

```
SELECT Name, Telephon FROM Mitarbeiter
WHERE (Name LIKE 'M_ier');
```

liefert

| Name | Telephon |
|---|---|
| Maier | 06196-671797 |
| Meier | 069-97640232 |

`LIKE` hat seine Entsprechung im `WIE` in *Base*.

| Zeichen in SQL | Zeichen in *Base* | Bedeutung |
|---|---|---|
| % | * | steht für eine bel. Anzahl (auch 0) von Zeichen |
| _ | ? | steht für genau ein bel. Zeichen |

### 2.6.3 BETWEEN

`BETWEEN` wählt alle Werte aus, die zwischen dem unteren und oberen Wert (einschließlich) liegen.
`SELECT * FROM Mitarbeiter WHERE (AbtNr BETWEEN 2 AND 3);` liefert

| MitarbNr | VorgesetztenNr | AbtNr | Name | GebDatum | Telephon |
|---|---|---|---|---|---|
| 1 | NULL | 3 | Christoph Reeg | 13.05.1979 | NULL |
| 5 | 1 | 2 | Maier | NULL | 06196-671797 |
| 6 | 5 | 2 | Meier | NULL | 069-97640232 |

In *Base* lautet die Entsprechung `Zwischen...und...`.

### 2.6.4 IN

Wie in *Base* benutzt man `IN`.

```
SELECT * FROM Mitarbeiter
WHERE Telephon IN ('06196-671797','069-97640232');
```

liefert

| MitarbNr | VorgesetztenNr | AbtNr | Name | GebDatum | Telephon |
|---|---|---|---|---|---|
| 5 | 1 | 2 | Maier | NULL | 06196-671797 |
| 6 | 5 | 2 | Meier | NULL | 069-97640232 |

# 3 Verändern der Tabellenstruktur

## 3.1 Erzeugen einer Tabelle

Mit `CREATE` wird eine Tabelle in einer bestehenden Datenbank erzeugt:

```
CREATE TABLE tabellenname(
feldname feldtyp [feldoptionen]
[,feldname feldtyp [feldoptionen]]
[,PRIMARY KEY (feldname)] );
```

Als Feldtypen stehen u.a. folgende zur Verfügung:

| *Typ* | Verwendung | Erläuterung |
|---|---|---|
| `CHAR` | `CHAR(1)` | Einzelner Buchstabe |
| | `CHAR(20)` | Text mit der Länge 20. Fehlende Buchstaben werden mit Leerzeichen aufgefüllt. Die maximale Länge ist 255. |
| `VARCHAR` | `VARCHAR(20)` | Text mit der Länge 20. Fehlende Buchstaben werden nicht mit Leerzeichen aufgefüllt. Die tatsächliche Länge wird gespeichert. Die maximale Länge ist auch 255. |
| `TEXT` | `TEXT` | Text mit einer maximalen Länge von 65.536 Zeichen. |
| `TINYINT` | `TINYINT` | 8-Bit-Ganzzahl (integer), Bereich: $-2^7 \ldots 2^7 - 1$ |
| `INT` | `INT` | 32-Bit-Ganzzahl (integer), Bereich: $-2^{31} \ldots 2^{31} - 1$ |
| `FLOAT` | `FLOAT` | Fließkommazahl mit einfacher Genauigkeit, Bereich: $-3{,}403 \cdot 10^{38} \ldots 3{,}403 \cdot 10^{38}$ |
| `DOUBLE` | `DOUBLE` | Fließkommazahl mit doppelter Genauigkeit, Bereich: $-1{,}798 \cdot 10^{308} \ldots 1{,}798 \cdot 10^{38}$ |
| `DATE` | `DATE` | Datumsformat im Format JJJJ-MM-TT. |
| `TIME` | `TIME` | Zeitformat im Format HH:MM:SS. Der Wertebereich ist erstaunlicherweise von -838:59:59 bis 838:59:59. |

Als Optionen sind möglich:

| | |
|---|---|
| NOT NULL | Das Feld darf nicht leer bleiben. |
| DEFAULT wert | Wird beim Speichern kein Wert angegeben, so wird diese Vorgabe genommen. |
| PRIMARY KEY | Primärschlüssel (kann es nur einmal geben). |
| AUTO_INCREMENT | praktisch für Primärschlüsselfelder, da MySQL den Wert automatisch erhöht. |

## 3.2 Ändern einer Tabelle

`DROP TABLE tabellenname;` löscht eine Tabelle komplett.

`ALTER TABLE Tabellenname DROP feldname;` löscht eine Spalte der Tabelle.

`ALTER TABLE Tabellenname ADD feldname feldtyp feldoptionen;` erweitert die Tabelle um eine Spalte.

`ALTER TABLE Tabellenname RENAME NeuerTabellenname;` ändert den Tabellennamen.

`ALTER TABLE Tabellenname CHANGE alterfeldname neuerfeldname feldtyp;` ändert den Feld- bzw. Spaltennamen. Der Typ muss mit angegeben werden.

`ALTER TABLE Tabellenname CHANGE feldname feldname neuerfeldtyp;` ändert nur den Feldtyp.

# 4 Verändern der Tabelleninhalte

## 4.1 Einfügen von Datensätzen

`INSERT` dient dem Einfügen von Datensätzen in eine Tabelle:

```
INSERT INTO tabellenname
[ (feldname1, feldname2, feldname3, ...)  ]
VALUES
(wert1, wert2, wert3, ...);
```

Wenn man die Feldnamen weglässt, dann müssen alle Werte in der richtigen Reihenfolge angegeben werden. Will man nur einzelne Werte eingeben, dann müssen die zugehörigen Feldnamen angegeben werden.

## 4.2 Löschen von Datensätzen

```
DELETE FROM tabellenname
WHERE feldname Vergleichsoperator wert;
```

löscht einen Datensatz, der eine bestimmte Bedingung erfüllt. Beispiel :

```
DELETE FROM Telephonliste WHERE Name = 'Maier';
```

## 4.3 Verändern von Datensätzen

```
UPDATE tabellenname
SET feldname = WERT
WHERE feldname Vergleichsoperator wert;
```

aktualisert einen vorhandenen Datensatz, der eine bestimmte Bedingung erfüllt. Beispiel :

```
UPDATE Telephonliste SET telephon = '0180-0815' WHERE Name = 'Maier';
```

# 5 SQL-Funktionen

## 5.1 Funktionen für aggregierte Daten

Die Aggregationsfunktionen beziehen sich auf selektierte Daten, d.h. auf eine Menge von Werten. Insbesondere in Verbindung mit `GROUP` ergeben sich interessante Anwendungen.

| | |
|---|---|
| `AVG(feldname)` | average; Durchschnittswert einer Spalte berechnen. |
| `COUNT(feldname)` | Zählt Zeilen, deren Wert nicht NULL ist. |
| `FIRST(feldname)` | Erster Wert einer Spalte. |
| `LAST(feldname)` | Letzter Wert einer Spalte. |
| `MAX(feldname)` | Größter Wert einer Spalte. |
| `MIN(feldname)` | Kleinster Wert einer Spalte. |
| `SUM(feldname)` | Summe über alle Spaltenwerte. |

## 5.2 Funktionen für skalare Daten

Im Gegensatz zu den Aggregationsfunktionen beziehen sich diese Funktionen nur auf einen Wert.

| | |
|---|---|
| `LCASE(str)` | Wandelt String in Kleinbuchstaben um. |
| `UCASE(str)` | Wandelt String in Großbuchstaben um. |
| `LEFT(str, n)` | Liefert die ersten $n$ Buchstaben des Strings. |
| `LEN(str)` | Liefert die Länge des Strings. |
| `DAYOFWEEK(date)` | Gibt den Wochentag des Datums an (Sonntag = 1). |
| `DAYOFMONTH(date)` | Gibt den Tag des Monats an. |
| `DAYOFYEAR(date)` | Gibt den Tag im Jahr zurück. |
| `MONTH(date)` | Gibt den Monat des Datums zurück. |
| `YEAR(date)` | Gibt das Jahr des Datums zurück. |
| `NOW()` | Aktuelles Datum. |

# 6 Subqueries

Wir betrachten die folgenden beiden Tabellen:

*jungen*

| ID | Name | Gewicht |
|---|---|---|
| 1 | Klaus | 57 |
| 2 | Kevin | 67 |
| 3 | Kevin-Alexander | 57 |
| 4 | Klaus-Peter | 61 |
| 5 | Kilian | 55 |
| 6 | Kelvin | 68 |
| 7 | Keno | 72 |
| 8 | Konrad | 54 |

*maedchen*

| ID | Name | Gewicht |
|---|---|---|
| 1 | Jenny | 53 |
| 2 | Jennifer | 54 |
| 3 | Jessica | 65 |
| 4 | Jolanda | 67 |
| 5 | Jasmin | 59 |
| 6 | Josephine | 60 |

und dazu die folgenden Fragen:

1. **Welche Mädchen wiegen weniger als der Durchschnitt der Mädchen?**

   Um das Durchschnittsgewicht zu erhalten, muss mit einer gesonderten SELECT-Anweisung gearbeitet werden.

   ```
   SELECT Name, Gewicht FROM maedchen
          WHERE Gewicht < (SELECT AVG(Gewicht) FROM maedchen);
   ```

   liefert dann

   | | |
   |---|---|
   | Jenny | 53 |
   | Jennifer | 54 |
   | Jasmin | 59 |

2. **Welche Mädchen sind gleich schwer wie ein Junge?**

   (a) Lösung 1:

   ```
   SELECT maedchen.Name, maedchen.Gewicht FROM junge, maedchen
          WHERE junge.Gewicht = maedchen.Gewicht
   ```

   liefert

   | | |
   |---|---|
   | Jolanda | 67 |
   | Jennifer | 54 |

   (b) Lösung 2 (mit Subquery):

   ```
   SELECT Name, Gewicht FROM maedchen
          WHERE Gewicht IN (SELECT Gewicht FROM Junge);
   ```

   liefert

   | | |
   |---|---|
   | Jennifer | 54 |
   | Jolanda | 67 |

3. **Welche Mädchen sind nicht gleich schwer wie ein Junge?**

```
SELECT Name, Gewicht FROM maedchen
       WHERE Gewicht NOT IN (SELECT Gewicht FROM Junge);
```

| | |
|---|---|
| Jenny | 53 |
| Jessica | 65 |
| Jasmin | 59 |
| Josephine | 60 |

4. **Welche Jungen wiegen mehr als der Durchschnitt aller Mädchen?**

```
SELECT Name, Gewicht FROM junge
       WHERE Gewicht > (SELECT AVG(Gewicht) FROM maedchen);
```

liefert

| | |
|---|---|
| Kevin | 67 |
| Klaus-Peter | 61 |
| Kelvin | 68 |
| Keno | 72 |