

# SQL (Structured Query Language)

## Datenbanken und Tabellen erstellen

- RRZN-Handbuch "SQL, Grundlagen und Datenbank-Design"
- Alan Beaulieu: Einführung in SQL; O'Reilly-Verlag; ISBN-Nr. 978-3-89721-443-9
- Lynn Beighly: SQL von Kopf bis Fuß; O'Reilly-Verlag; ISBN-Nr. 978-3-89721-760-7
- Matthiesen & Unterstein: Relationale Datenbanken und Standard-SQL; Addison-Wesley; ISBN-Nr. 978-3-8273-2656-0
- Wolfgang D. Misgeld: SQL; Hanser; ISBN-Nr. 3-446-21636-7

- <http://www.torsten-horn.de/techdocs/sql.htm>
- <http://www.sql-und-xml.de/sql-tutorial/>
- <http://sqlzoo.net/de/>
- <http://sql.1keydata.com/de/>
- <http://www.w3schools.com/sql/default.asp>
- <http://reeg.junetz.de/DSP/node8.html>
- <http://dev.mysql.com/doc/>

- ... ist eine Sammlung von Daten zu einem bestimmten Thema oder einen bestimmten Zweck.
- ... ermöglicht eine strukturierte Ablage von Informationen.
- ... kann Arbeitsabläufe eines Unternehmens abbilden.
- ... ist ein Container,
  - ... in dem die Daten in Tabellenform abgelegt werden und
  - ... verschiedene Sichten auf die Daten erstellt werden können.
- Beispiele:
  - Adressverwaltung.
  - Sammlung aller Informationen zu CDs oder Büchern.
  - Ablage von Informationen für die Abwicklung von Bestellungen.
  - Abbildung eines Warenlagers.

- ... sind das am weitesten verbreitete Modell.
- Merkmale:
  - Die Daten werden in Tabellen abgelegt.
  - Jede Information, die in einer Tabelle abgelegt wird, besitzt den gleichen Aufbau.
  - Daten in verschiedenen Tabellen können eine Beziehung zueinander haben. Die Daten in den Tabellen können mit Hilfe eines Schlüsselwertes verknüpft werden.

- Informationen zu einer Buchbestellung sammeln:
  - In einer Tabelle werden alle Informationen zu einem bestimmten Buch gesammelt.
  - In einer anderen Tabelle werden alle Lagerinformationen zu einem Buch gesammelt.
  - In einer anderen Tabelle werden alle Informationen zu dem Besteller gesammelt.
  - In einer weiteren Tabelle werden Besteller und deren Bestellung abgelegt.
- Informationen zu Studenten:
  - In einer Tabelle werden Informationen zu Studenten eines Fachbereichs gesammelt. Die gesammelten Daten beschreiben den Studenten.
  - In einer Tabelle werden alle Fachbereiche gesammelt.
  - In einer weiteren Tabelle werden die Studenten den Fachbereichen zugeordnet.

<b>Adressbuch als Datenbank</b>	<b>Adressbuch aus Papier</b>
Eine Datenbank legt Adressen in Tabellen ab.	Ein Adressbuch enthält sehr viele Adressen.
Daten werden strukturiert abgelegt. Pro Zeile wird die Adresse einer bestimmten Person abgelegt. Die Informationen zu einer Person werden so aufgeteilt, dass die einzelnen Elemente in Spalten untereinander erfasst werden.	In einem Adressbuch werden Name, Straße, Wohnort und vielleicht die Telefonnummer jeweils einer Person erfasst und abgelegt.
In Abhängigkeit einer bestimmten Eigenschaft können die Daten in Tabellen sortiert werden.	Ein Adressbuch wird zum Beispiel nach dem Namen der Personen sortiert. Eine andere Sortierung ist anschließend nicht möglich.
Die Daten können mit Hilfe von Abfragen nach bestimmten Kriterien gefiltert und neu zusammengestellt werden.	Eine Filterung ist nur durch ein Neuschreiben der Daten möglich.

<b>... am Beispiel eines Karteikastens</b>	<b>entspricht in einer Datenbank</b>
In einem Karteikasten werden Informationen von Studenten in einem Fachgebiet gesammelt.	In einer Tabelle werden strukturiert Daten zu Studenten in einem Fachgebiet abgelegt.
Auf Karteikarten werden bestimmte Informationen zu einem Studenten gesammelt. Die Daten identifizieren den Studenten genau.	In einer Zeile einer Tabelle werden alle Daten zu einem bestimmten Studenten abgelegt. Jede Zeile wird als Datensatz bezeichnet. Der Datensatz definiert einen Studenten exakt.
Jede Karteikarte hat den gleichen Aufbau. Die Daten stehen immer an exakt der gleichen Stelle.	Die Informationen werden in Spalten abgelegt. In jeder Spalte steht immer exakt eine Information, deren Art überall gleich ist. Zum Beispiel in der Spalte Wohnort steht immer der Wohnort eines Studenten und nicht das Geburtsdatum.
In jedem Feld auf der Karteikarte sollte immer die gleiche Information stehen. Es findet aber keine Gültigkeitsprüfung statt. Eine Formatierung der Daten ist nicht vorgegeben.	Innerhalb einer Zelle stehen Informationen in Abhängigkeit der zu beschreibenden Person abgelegt. Der Datentyp ist abhängig von der Spalte. Die Gültigkeit der Daten kann überprüft werden.

- Strukturierte Abfragesprache für relationale Datenbanken.
- Datendefinition, -manipulation, -abfrage.
- ... definiert Kriterien, um nach Daten zu suchen.
- ... kann eine Menge von Datensätze automatisch aktualisieren oder löschen.
- ... besteht aus sehr wenigen Kommandos, sehr vielen Schlüsselwörter und einfachen Funktionen.
- In SQL sind keine Schleifen, bedingte Anweisungen oder die Nutzung von Variablen implementiert.
- Die aktuelle Norm ist momentan SQL:2003. Jede SQL-Datenbank setzt diese Norm unterschiedlich stark um. Teilweise werden eigene Befehle hinzugefügt.

- DDL (Data Definition Language)
  - Erstellen von Datenbanken, Tabellen und Indizes.
- DQL (Data Query Language)
  - Abfragen von Daten.
- DML (Data Manipulation Language)
  - Anlegen, ändern und löschen von Datensätzen.
- DCL (Data Controlling Language)
  - Anlegen von Benutzern und Vergabe von Zugriffsrechten.

- Die Auswahl sollte nicht vom Bekanntheitsgrad abhängen.
- Nutzen Sie lieber folgende Kriterien:
  - Welcher Funktionsumfang wird zur Lösung der Aufgabe benötigt?
  - Bedienbarkeit.
  - Wird die Datenbank nur in einem lokalen Netz oder im Web genutzt?
  - Liefert die Datenbank Schnittstellen zu der Programmiersprache, mit der ich ein Frontend etc. erstelle?
  - Kann die Datenbank die benötigte Datenmenge ohne Problem verarbeiten?
  - Welche Wünsche hat der Auftraggeber? Ist eine Datenbank vorhanden?

- MySQL: <http://www.mysql.de/>
  - ... wird für dynamische Webseiten im Internet eingesetzt.
  - ... wird häufig mit der Programmiersprache PHP eingesetzt.
  - ... ist am weitesten verbreitet.
- PostgreSQL: <http://www.postgresql.org>,  
<http://www.postgres.de/software.whtml#los>
  - ... unterstützt sehr stark Transaktionen.
  - ... hat die längste Entwicklungsgeschichte.
- Firebird: <http://www.firebirdsql.org/>
  - ... ist interessant, wenn Borland-Werkzeuge genutzt werden.
  - ... bietet viele Funktionen für Datenbankentwickler.
- ... und so weiter.

```
CREATE DATABASE datenbankName;
```

- Mit dieser SQL-Anweisung wird eine neue Datenbank angelegt.
- Der Name der Datenbank ist frei wählbar, sollte aber den Inhalt widerspiegeln.
- In MySQL wird für jede Datenbank ein Verzeichnis im Ordner data angelegt.

```
CREATE DATABASE myLager;
```

- ... können in einer Zeile oder in mehreren Zeilen eingegeben werden.
- ... werden mit einem Semikolon abgeschlossen.
- ... werden immer von links nach rechts gelesen.
- ...beginnen immer mit einem SQL-Befehl. Dem SQL-Befehl folgen Objekte, auf die der Befehl ausgeführt wird und Daten, die für die Ausführung benötigt werden.
- Beispiel
  - `CREATE DATABASE` ist ein SQL-Befehl
  - `datenbankName` ist das Objekt, auf dem der Befehl ausgeführt wird. Hier wird eine neue Datenbank mit diesen Namen erzeugt.

- ... können aus einer beliebigen Kombination von Buchstaben und Ziffern bestehen. Es dürfen alle alphanumerischen Zeichen genutzt werden.
- ... können zusätzlich den Unterstrich oder das Dollarzeichen enthalten.
- ... dürfen mit einem beliebigen alphanumerischen Zeichen, dem Unterstrich oder Dollarzeichen beginnen.
- ... dürfen nicht nur aus Zahlen bestehen.
- ... bestehen aus maximal 64 Zeichen.
- ... dürfen keine Leerzeichen enthalten
- ... sollten keine deutschen Umlaute enthalten.

- SQL-Befehle beachten nicht die Groß- und Kleinschreibung. Die Befehle werden häufig in Großbuchstaben geschrieben, um die Lesbarkeit der Anweisung zu erhöhen.
- Bei Datenbanknamen und Tabellennamen wird
  - ... unter Windows die Groß- und Kleinschreibung nicht beachtet.
  - ... unter UNIX / Linux die Groß- und Kleinschreibung beachtet.

```
CREATE DATABASE IF NOT EXISTS datenbankName;
```

- Falls die Datenbank nicht existiert, wird eine Datenbank mit diesen Namen angelegt.

CREATE DATABASE if not exist firma;

```
SHOW DATABASES;
```

- ... zeigt eine Liste der vorhandenen Datenbanken an.

```
SHOW DATABASES;
```

```
USE datenbankName;  
  
DROP DATABASE datenbankName;
```

- Mit Hilfe von `USE` wird die angegebene Datenbank für die Bearbeitung in MySQL ausgewählt.
  - Es kann immer nur eine Datenbank ausgewählt werden.
  - Alle weiteren Befehle beziehen sich auf diese Datenbank.
  - Andere SQL-Server nutzen den Befehl `CONNECT`.
- Mit Hilfe von `DROP` wird die Datenbank gelöscht.
  - Es wird keine Warnmeldung ausgegeben. Die Daten in der Datenbank sind verloren.
  - Mit Hilfe der Anweisung `DROP DATABASE IF EXISTS` wird die Datenbank gelöscht, falls vorhanden.

```
DROP firma;  
USE verein;  
SHOW DATABASES;
```

- Jede Tabelle sammelt nur Informationen zu einem bestimmten Thema.  
Beispiele:
  - Bestellungen werden in einer Tabelle gesammelt.
  - Kundeninformationen werden in einer Tabelle gesammelt.
  - Waren werden in einer Tabelle gesammelt.
- Tabellen können mit anderen Tabellen über Schlüsselwerte verknüpft werden. Beispiel:
  - Es sind die Tabellen „ware“ und „kunde“ vorhanden. Jede Ware hat eine eindeutige Artikelnummer sowie jeder Kunde eine eindeutige Kundennummer.
  - In der Tabelle „Lager“ wird nur die Artikelnummer einer Ware aufgelistet. Die Artikelnummer ist ein Verweis auf einen bestimmten Datensatz in der Tabelle „ware“, der die Details zu dem ausgewählten Artikel enthält.
  - In der Tabelle „Bestellung“ wird die Ware sowohl als auch der Kunde über den dazugehörigen Schlüsselwert eingebunden. Der Schlüsselwert „Artikelnummer“ sowie „Kundennummer“ identifizieren ein Objekt eindeutig und damit auch den dazugehörigen Datensatz.

• Feldnamen (Attribut)

IAmtNr	Vorname	Nachname	Straße	PLZ	Wohnort
123	Heinz	Peters	Friesenstr. 4	xxxxx	Hannover
456	Sonja	Müller	Eichenstr. 5	xxxxx	Hannover
753	Marlis	Schulze	Lerchenstr. 2	xxxxx	Mellendorf
789		Scholz		xxxxx	Hildesheim

• Datensatz (Tupel)

• Datenfeld (Attributwert)

- Eine Tabelle setzt sich immer aus Zeilen (Datensätze, Tupel) und Spalten (Datenfeldern, Attribute) zusammen.
- Eine Tabelle kann leer sein oder beliebig viele Datensätze enthalten.  
Beispiel:
  - MySQL: Die Größe der Tabelle wird durch die maximale Größe einer Datei in Abhängigkeit des Betriebssystems bestimmt.
  - ACCESS: Ca. 65.000 Datensätze pro Tabelle.
- Jede Tabelle hat einen eindeutigen Namen.

- ... beschreibt die Eigenschaft eines Objekts. Jedes Objekt hat die gleichen Eigenschaften (Datenfelder).
- ... besteht immer aus der gleichen Anzahl von Attributen.
- ... unterscheidet sich von allen anderen Datensätzen in mindestens einem Attributwert. Andernfalls ist das Objekt häufiger in der Tabelle abgelegt. Jedes Objekt ist einzigartig.
- ... hat einen Primärschlüssel zur Identifizierung des Objekts.
  - Der Schlüsselwert ist eindeutig.
  - Der Wert ändert sich nie.

- ... besitzt einen Datentyp, der die Daten beschreibt.
  - In einem Feld, welches nur Ziffern aufnehmen kann, können keine Buchstaben genutzt werden.
  - Textfelder können jedes beliebige Zeichen aufnehmen.
  - Mit Ziffern-Feldern kann nur gerechnet werden.
- Der Wert eines Datenfeldes beschreibt immer eine bestimmte Eigenschaft.
- ... können nicht weiter zerlegt werden. Zum Beispiel das Element "30159 Hannover" ist nicht atomar. Es kann in die Bestandteile "30159" und "Hannover" zerlegt werden. Der Ort sowie die Postleitzahl sind nicht mehr aufteilbar.

Mit Hilfe des SQL-Befehls wird die Tabelle mitarbeiter erzeugt.

```
CREATE TABLE mitarbeiter  
(  
    mitarbeiter_nachname VARCHAR(50),  
    mitarbeiter_vorname VARCHAR(20),  
    mitarbeiter_geburtsdatum DATE  
);
```

Hier werden die Spalten definiert.

```
CREATE TABLE artikel(  
    artikelname VARCHAR(50),  
    artikelnummer VARCHAR(11),  
    kategorie VARCHAR(20));
```

```
CREATE TABLE lager(  
    artikelnummer VARCHAR(11),  
    verpackung VARCHAR(10)  
    preis DECIMAL, menge INTEGER);
```

Hier beginnt die Spaltenliste

Die verschiedenen Spalten-  
definitionen werden durch ein  
Kommata getrennt.

```
CREATE TABLE mitarbeiter  
(  
    mitarbeiter_nachname VARCHAR(50),  
    mitarbeiter_vorname VARCHAR(20),  
    mitarbeiter_geburtsdatum DATE  
);
```

... und hier endet sie.

Für jede Spalte wird ein Datentyp angegeben.

```
CREATE TABLE mitarbeiter
(  
    mitarbeiter_nachname VARCHAR(50),  
    mitarbeiter_vorname VARCHAR(20),  
    mitarbeiter_geburtsdatum DATE  
);
```

Jede Spalte hat einen eindeutigen Namen.

- ... sind die Baupläne für die Werte einer Spalte in einer Tabelle.
- ... geben über das Format eines Wertes Auskunft.
- ... legen Regeln fest, wie ein Wert interpretiert und verwendet werden kann.
- ... legen einen bestimmten Wertebereich fest.
- ... werden in
  - ... numerische Datentypen,
  - ... Datums- und Uhrzeit-Datentypen und
  - ... Text-Datentypen unterteilt.
- Die Auswahl des passenden Datentyps reduziert die Tabellengröße und beschleunigt Operationen.
- Die Bezeichnungen der Datentypen sind abhängig von der gewählten SQL-Implementierung.

Datentyp	Beschreibung	Bereich
SMALLINT	Ganzzahl	-32768 bis 32767 (0 bis 65535)
INT	Ganzzahl	-2147283648 bis 2147283647 (0 bis 4294967295)
FLOAT	Fließkommazahlen	einfache Genauigkeit
DOUBLE	Fließkommazahlen	doppelte Genauigkeit
DECIMAL (Länge, Nachkomma)	Festkommazahlen	

- Fließkommazahlen können Rundungsfehler erzeugen. Intern wird die Zahl 5.2 nur näherungsweise dargestellt.
- Für Währungsangaben wird der `DECIMAL` genutzt. Beispiel:
  - `DECIMAL (2, 2)` speichert Zahlen zwischen -99,99 und 99,99.
  - `DECIMAL (5, 3)` speichert Zahlen zwischen -99,999 und 99,999.
  - `DECIMAL` kann zum Beispiel in MySQL insgesamt 65 Stellen und 30 Nachkommastellen besitzen.

- ... können vom Datentyp BOOLEAN (SQL99-Standard) sein.
- ... haben die Werte TRUE(wahr, 1) oder FALSE(falsch, 0).
- ... wird nicht von allen Datenbankimplementierung unterstützt.

Datentyp	Beschreibung	Bereich
DATE	Datumswert	1.1.1000 bis 31.12.9999
TIME	Uhrzeit	-838:59:59 bis 8838:59:59
DATETIME	Datum und Uhrzeit	01.01.1000 00:00:00 bis 9999-12.31 23:59:59

- Die Formatierung der Datumsangaben ist abhängig von der gewählten SQL-Implementierung.
- Datumsangaben werden in der Form `jjjj-mm-tt`.
  - Zweistellige Jahreszahlen zwischen 00 und 69 werden von MySQL als 2000 bis 2069 interpretiert.
  - Zweistellige Jahreszahlen zwischen 70 und 99 werden von MySQL als 1970 bis 1999 interpretiert.
- Zeitangaben erfolgen in der Form `hh:mm:ss`.

Datentyp	Beschreibung
CHAR (Laenge)	Text mit bis zu 255 Zeichen. Text mit einer festen Länge.
VARCHAR (maxLaenge)	Text mit bis zu 255 Zeichen. Text mit einer variablen Länge.
BLOB	Große Textdateien, Grafiken etc.

```
DESC tabellenname;
```

Field	Type	Null	Key	Default	Extra
vorname	varchar(20)	YES		NULL	
nachname	varchar(50)	YES		NULL	
geburt	date	YES		NULL	

3 rows in set (0.04 sec)

- ... beschreibt die Struktur einer Tabelle.
- ... zeigt nicht die Daten in einer Tabelle an.

```
DESC lager;
```

```
ShOW TABLES;
```

- ... zeigt alle Tabellen und Sichten in der aktiven Datenbank an.

```
SHOW TABLES;
```

```
DROP TABLE tabellename;
```

- ... löscht eine Tabelle aus einer Datenbank.
- Daten in der Tabelle sowie die Tabellenstruktur werden gelöscht.

DROP artikel;

In welche Tabelle sollen die Daten eingefügt werden?

Liste von Spaltennamen, in der die Daten abgelegt werden. Die Listenelemente werden durch Kommata getrennt.

```
INSERT INTO tabellename
  (Spaltenname, Spaltenname, ...)
VALUES (Wert1, Wert2, ...);
```

Mit welchen Werten werden die Spalten belegt?

```
INSERT INTO lager(artikelnummer, verpackung, preis, menge)
VALUES();
```

```
INSERT INTO tabellenname  
    (Spaltenname, Spaltenname, ...)
```

- Die Anzahl der angegebenen Spalten entspricht der Anzahl der anzugebenden Werte für die Spalten.
- Die Klammern kann leer gelassen werden. Es muss dann für jede, in der Tabelle vorhandenen Spalte, ein Wert angegeben werden.

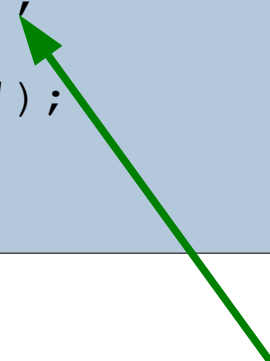
```
...  
VALUES (Wert1, Wert2, ...);
```

- Die Werte müssen in der gleichen Reihenfolge wie die Spaltennamen stehen.
  - Der erste Wert wird in der ersten angegebenen Spalte gespeichert. Die Reihenfolge der Spaltennamen ist egal.
  - Der Wert muss dem Datentyp der Spalte entsprechen!
- Text und Datums- sowie Zeitangaben werden durch die Apostrophs begrenzt. Beispiel: 'RRZN', '1973-02-01'.
- Falls Texte ein Apostroph enthalten, muss es mit einem Backslash maskiert werden. Beispiel: 'Otto\'s'.

```
INSERT INTO lager(artikelnummer,, menge, verpackung, preis)  
VALUES();
```

```
CREATE TABLE mitarbeiter(  
  nachname VARCHAR(50),  
  vorname VARCHAR(20),  
  geburtsdatum DATE  
);  
  
INSERT INTO mitarbeiter(nachname, vorname, geburtsdatum)  
  VALUES ('Müller', 'Sven', '1969-02-03');  
  
INSERT INTO mitarbeiter(nachname, geburtsdatum)  
  VALUES ('Meier', '1969-12-04');  
  
INSERT INTO mitarbeiter(vorname, nachname, geburtsdatum)  
  VALUES ('Heinz', 'Frosch', '1969-03-03');  
  
INSERT INTO mitarbeiter()  
  VALUES ('Schmidt', 'Ute', '1969-08-03');
```

```
CREATE TABLE mitarbeiter(  
  nachname VARCHAR(50),  
  vorname VARCHAR(20),  
  geburtsdatum DATE  
);  
  
INSERT INTO mitarbeiter(nachname, vorname, geburtsdatum)  
VALUES  
  ('Müller', 'Sven', '1969-02-03'),  
  ('Meier', 'Ute', '1968-03-03'),  
  ('Scholz', 'Heinz', '1967-10-03');
```



```
INSERT INTO lager()  
VALUES();
```

Die verschiedenen Datensätze  
werden durch Kommata getrennt.

```
SELECT * FROM tabellename;
```

```
mysql> SELECT * FROM mitarbeiter;
+-----+-----+-----+
| vorname | nachname | geburt |
+-----+-----+-----+
| NULL    | mueller  | 2001-02-20 |
| NULL    | meier    | 1967-02-01 |
| NULL    | meier    | 2067-02-01 |
| Heinz   | meier    | 2067-02-01 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

- Durch das Sternchen werden alle Datensätze aus einer Tabelle angezeigt.
- SELECT (Wähle aus, Selektierte) alle FROM (Wo?, von) tabelle.

```
SELECT * FROM kunde;
```

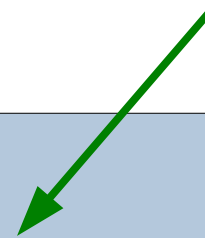
## ■ NULL

- ... ist ein undefinierter Wert.
- Das Attribut kann jeden beliebigen Wert besitzen. Der Wert muss bloß dem angegebenen Datentyp entsprechen.
- ... ist eine leere Kiste.

## ■ Leere Zeichenfolge

- ... wird durch zwei Apostrophs hintereinander erzeugt.
- ... ist eine Zeichenfolge der Länge Null.

Für diese Spalte muss ein Wert eingegeben werden. Der Wert der Spalte darf nicht undefiniert sein.

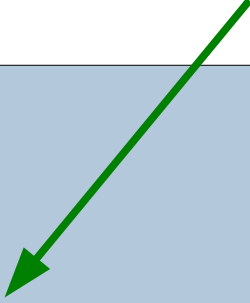


```
CREATE TABLE mitarbeiter
(
  mitarbeiter_nachname VARCHAR(50) NOT NULL,
  mitarbeiter_vorname VARCHAR(20),
  mitarbeiter_geburtsdatum DATE
);
```

```
CREATE TABLE artikel(
  artikelname VARCHAR(50) ;
  artikelnummer VARCHAR(11) NOT NULL,
  kategorie VARCHAR(20) NOT NULL);
```

Hier wird ein Standardwert von 1 definiert. Falls der Nutzer kein Wert eingibt, wird der Standardwert gesetzt. Der Standardwert muss dem Datentyp des Feldes entsprechen.

```
CREATE TABLE bestellung
(
  artikel VARCHAR(50) NOT NULL,
  menge INTEGER NOT NULL DEFAULT 1,
);
```



```
CREATE TABLES kunde(
kundenname VARCHAR(30) NOT IS NULL,
anrede VARCHAR(5) DEFAULT 'FRAU',
strasse VARCHAR(30),
plz VARCHAR(5) NOT NULL DEFAULT '37154',
ort VARCHAR(50),
angelegtam DATE);
INSERT INTO lager() VALUES ();
```

- ... beginnen immer mit SELECT.
- ... liefern eine bestimmte Anzahl von Datensätzen aus einer Datenmenge zurück.
- ... wählen Daten nach bestimmten Kriterien aus.
- ... haben als Grundlage immer eine Tabelle.
- ... liefern keine Datensätze zurück, wenn keine passende Datensätze in der genutzten Tabelle vorhanden sind.

```
SELECT [DISTINCT] datenfelder FROM tabelle
      [WHERE bedingung]
      [GROUP BY datenfelder [HAVING bedingung]]
      [ORDER BY datenfelder]
      [LIMIT [start, ] anzahl]
```

- Die einzelnen Befehle werden im weiteren Verlauf erläutert.
- Die eckige Klammern umschließen mögliche Optionen.

```
SELECT * FROM tabellename;
```

```
mysql> SELECT * FROM mitarbeiter;
+-----+-----+-----+
| vorname | nachname | geburt |
+-----+-----+-----+
| NULL    | mueller  | 2001-02-20 |
| NULL    | meier    | 1967-02-01 |
| NULL    | meier    | 2067-02-01 |
| Heinz   | meier    | 2067-02-01 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

- Durch das Sternchen werden alle Datensätze aus einer Tabelle angezeigt.
- `SELECT` (Wähle aus, Selektierte) alle `FROM` (Wo?, von) tabelle.

```
SELECT * FROM lager;
```

```
SELECT datenfeld1, datenfeld2, ... FROM tabellename;
```

```
mysql> SELECT nachname, geburt FROM mitarbeiter;
+-----+-----+
| nachname | geburt |
+-----+-----+
| mueller  | 2001-02-20 |
| meier    | 1967-02-01 |
| meier    | 2067-02-01 |
| meier    | 2067-02-01 |
+-----+-----+
4 rows in set (0.01 sec)
```

- Das Sternchen kann durch Spaltennamen in der Tabelle ersetzt werden.
- Es werden nur die Daten in den angegebenen Spalten angezeigt.
- Die Spaltennamen
  - ... werden durch ein Komma getrennt.
  - ... müssen in der angegebenen Tabelle vorkommen. Andernfalls wird ein Fehler angezeigt.
- Die Reihenfolge der Spaltennamen spielt keine Rolle.

```
SELECT artikelname, kategorie FROM artikel;
```

```
SELECT * FROM tabellenname ORDER BY datenfeld;
```

```
mysql> SELECT nachname, geburt FROM mitarbeiter ORDER BY geburt ASC;
+-----+-----+
| nachname | geburt |
+-----+-----+
| meier    | 1967-02-01 |
| mueller  | 2001-02-20 |
| meier    | 2067-02-01 |
| meier    | 2067-02-01 |
+-----+-----+
4 rows in set (0.00 sec)
```

- Hier werden die Daten sortiert.
- ... ORDER BY datenfeld ASC.
  - ASC entspricht einer aufsteigenden Sortierung (A bis Z und 0 bis 9).
  - ... ist die Standardsortierung.
- ... ORDER BY datenfeld DESC.
  - DESC entspricht einer absteigenden Sortierung (Z bis A und 9 bis 0).
- Es kann mit Hilfe von ORDER BY feld1, feld2 mehrere Felder sortiert werden. Die Daten werden zuerst nach feld1 sortiert und anschließend nach feld2.

```
SELECT artikelname, kategorie FROM artikel ORDER BY kategorie.;
```

```
SELECT * FROM tabellenname WHERE bedingung;
```

```
mysql> SELECT nachname, geburt FROM mitarbeiter
-> WHERE geburt = '1967-02-01';
+-----+-----+
| nachname | geburt |
+-----+-----+
| meier    | 1967-02-01 |
+-----+-----+
1 row in set (0.01 sec)
```

- Es werden alle Datensätze angezeigt, die der Bedingung entsprechen.
- Die Bedingung oder das Kriterium für die Auswahl beginnt mit `WHERE`.
- Eine Bedingung kann sich folgendermaßen aufbauen:  
feld Vergleichsoperator Wert
- Mehrere Bedingungen können miteinander verknüpft werden.

- ... sind Ausdrücke, die einen boolschen Wert zurückliefern.
- Ein boolscher Wert ist true(wahr) oder false(falsch, 0)
- ... vergleichen mit Hilfe von bestimmten Operatoren zwei Werte.
- ... sind zum Beispiel:
  - Wenn die Bestellmenge eine gewisse Höchstmenge überschreitet...
  - Wenn der Kontostand dem Dispo entspricht...
  - Wenn die Strecke A doppelt so lang ist wie Strecke B...
  - Wenn die Warenmenge eine Mindestmenge unterschreitet...

Operator	Rechenart	Beispiel
=	ist gleich	(7 = 3 ) => False
<	ist kleiner als	(7 < 3 ) => False
<=	ist kleiner gleich als	(7 <= 3 ) => False
>	ist größer als	(7 > 3 ) => True
>=	ist größer gleich als	(7 >= 3 ) => True
<>	ist ungleich	(7<> 3 ) =>True

```
SELECT artikel, preis, menge FROM lager
WHERE artikel = 'Banane';
```

```
SELECT artikel, preis, menge FROM lager
WHERE preis >= 2.53;
```

```
SELECT artikel, preis, menge FROM lager
WHERE menge < 10;
```

```
SELECT artikelnummer, verpackung FROM lager WHERE menge>= 5;
SELECT artikelname FROM artikel WHERE kategorie = 'Obst';
```

- ... oder relationale Operatoren.
- ... verknüpfen zwei oder mehr Bedingungen miteinander.
- Folgende Möglichkeiten sind vorhanden:
  - AND (Und, Konjunktion) ist nur wahr, wenn alle Bedingungen wahr sind.
  - OR (Oder, Disjunktion) ist wahr, sobald eine der Bedingungen wahr ist.
  - NOT (Negation) invertiert den booleschen Wert der Bedingung.

```
SELECT artikel, preis, menge FROM lager
WHERE artikel = 'Banane' AND preis >= 1.23;
```

```
SELECT artikel, preis, menge FROM lager
WHERE (preis = 1.99) OR (preis = 2.89);
```

```
SELECT artikel, preis, menge FROM lager
WHERE (menge > 5) AND (menge < 10);
```

Bedingung				
a	b	Not a	a AND b	a OR b
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

```
SELECT artikelnummer FROM lager
WHERE Not ( menge =4));
WHERE menge <> 4);
SELECT artikelnummer FROM lager
WHERE (preis > 2) AND (menge = 5)
WHERE (preis > 2) OR (menge = 5)
```

```
SELECT * FROM tabellenname  
WHERE datenfeld BETWEEN untergrenze AND obergrenze;
```

```
SELECT * FROM tabellenname  
WHERE NOT datenfeld BETWEEN untergr AND obergr;
```

```
+-----+-----+  
| standort | groesse |  
+-----+-----+  
| Hamburg |    2000 |  
| Hannover |    1200 |  
| Sarstedt |     150 |  
| Northeim |     200 |  
+-----+-----+  
4 rows in set (0.01 sec)  
  
mysql> SELECT * FROM lager WHERE groesse BETWEEN 1000 AND 2000;  
+-----+-----+  
| standort | groesse |  
+-----+-----+  
| Hamburg |    2000 |  
| Hannover |    1200 |  
+-----+-----+  
2 rows in set (0.00 sec)  
  
mysql> SELECT * FROM lager WHERE groesse >= 1000 AND groesse <= 2000;  
+-----+-----+  
| standort | groesse |  
+-----+-----+  
| Hamburg |    2000 |  
| Hannover |    1200 |  
+-----+-----+  
2 rows in set (0.00 sec)
```

```
SELECT artikelnummer  
FROM lager  
WHERE menge BETWEEN 5 AND 10;
```

```
SELECT * FROM tabellename  
WHERE datenfeld IN (element1, element2, ...);
```

```
SELECT * FROM tabellename  
WHERE datenfeld NOT IN (element1, element2, ...);
```

```
+-----+-----+  
| standort | groesse |  
+-----+-----+  
| Hamburg |    2000 |  
| Hannover |    1200 |  
| Sarstedt |     150 |  
| Northeim |     200 |  
+-----+-----+  
4 rows in set (0.00 sec)  
  
mysql> SELECT * FROM lager WHERE standort IN ('Hamburg', 'Northeim');  
+-----+-----+  
| standort | groesse |  
+-----+-----+  
| Hamburg |    2000 |  
| Northeim |     200 |  
+-----+-----+  
2 rows in set (0.00 sec)  
  
mysql> SELECT * FROM lager WHERE standort = 'Hamburg' OR standort = 'Northeim';  
+-----+-----+  
| standort | groesse |  
+-----+-----+  
| Hamburg |    2000 |  
| Northeim |     200 |  
+-----+-----+  
2 rows in set (0.00 sec)
```

```
SELECT artikelnummer  
FROM lager  
WHERE  
menge IN('1 kg', '5 kg', '10 kg');
```

```
SELECT * FROM tabellenname WHERE datenfeld IS NULL;
```

```
SELECT * FROM tabellenname WHERE NOT(datenfeld IS NULL);
```

```
+-----+-----+-----+
| vorname | nachname | geburt |
+-----+-----+-----+
| NULL    | mueller  | 2001-02-20 |
| NULL    | meier    | 1967-02-01 |
| NULL    | meier    | 2067-02-01 |
| Heinz   | meier    | 2067-02-01 |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT vorname, nachname FROM mitarbeiter WHERE vorname IS NULL;
+-----+-----+
| vorname | nachname |
+-----+-----+
| NULL    | mueller  |
| NULL    | meier    |
| NULL    | meier    |
+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT vorname, nachname FROM mitarbeiter WHERE NOT(vorname IS NULL);
+-----+-----+
| vorname | nachname |
+-----+-----+
| Heinz   | meier    |
+-----+-----+
1 row in set (0.00 sec)
```

```
SELECT kundenname FROM kunde
WHERE anrede IS NULL;
```

```
SELECT * FROM tabellenname WHERE datenfeld = '';  
SELECT * FROM tabellenname WHERE NOT(datenfeld = '');
```

```
+-----+-----+-----+  
| vorname | nachname | geburt |  
+-----+-----+-----+  
| NULL    | mueller  | 2001-02-20 |  
| NULL    | meier    | 1967-02-01 |  
| NULL    | meier    | 2067-02-01 |  
| Heinz   | meier    | 2067-02-01 |  
|         | Scholz   | 1968-03-02 |  
+-----+-----+-----+  
5 rows in set (0.00 sec)  
  
mysql> SELECT vorname, nachname FROM mitarbeiter WHERE (vorname = '');  
+-----+-----+  
| vorname | nachname |  
+-----+-----+  
|         | Scholz   |  
+-----+-----+  
1 row in set (0.00 sec)  
  
mysql> SELECT vorname, nachname FROM mitarbeiter WHERE NOT(vorname = '');  
+-----+-----+  
| vorname | nachname |  
+-----+-----+  
| Heinz   | meier    |  
+-----+-----+  
1 row in set (0.00 sec)
```

```
SELECT kundenname FROM kunde  
WHERE anrede = '';
```

- ... entspricht dem Gleichheitszeichen.
- ... kann nur für Textwerte genutzt werden.
- ... wird für Mustervergleiche genutzt werden.
- Im Suchmuster können folgende Platzhalter an beliebiger Position genutzt werden:
  - Das Prozentzeichen steht für kein, ein oder mehrere Zeichen.
  - Der Unterstrich steht für exakt ein beliebiges Zeichen.

```
SELECT nachname, eintritt FROM mitarbeiter
WHERE nachname LIKE 'M%';
```

```
SELECT nachname, eintritt FROM mitarbeiter
WHERE nachname LIKE 'Me_er';
```

```
SELECT artikel, preis, menge FROM lieferbar
WHERE artikel LIKE 'Banane_';
```

```
SELECT artikelnr, artikel FROM lieferbar
WHERE artikelnr LIKE '456-*-78-_';
```

```
SELECT standort, plz, groesse menge FROM lager
WHERE plz LIKE '38____';
```

```
SELECT * FROM artikel
WHERE
artikelnummer LIKE '400-_0-*';
```

```
SELECT * FROM kunde
WHERE kundenname LIKE 'M_ier';
WHERE plz='37*';
```

```
DELETE FROM tabellenname WHERE Bedingung;
```

- In Abhängigkeit einer Bedingung werden Datensätze aus einer Tabelle gelöscht.
- Falls keine Bedingung angegeben wird, werden alle Datensätze einer Tabelle gelöscht.
- Beispiele:
  - `DELETE FROM kunde WHERE plz LIKE '38____';`
  - `DELETE FROM buch WHERE autor = 'Kehlmann';`
  - `DELETE FROM lager WHERE menge BETWEEN 1 AND 5;`
- Es werden Datensätze gelöscht und nicht einzelne Werte aus Datenfeldern!

```
DELETE FROM lager  
WHERE artikelnummer = '300-1*';
```

- ... werden in der Form [ausdruck Operator ausdruck] gebildet.
- ... bestehen aus Operanden, die einen Wert symbolisieren und Operatoren, die die Berechnungsart angeben.
- ... geben immer einen Wert zurück.
  - Eine Berechnung von numerischen Wert gibt einen numerischen Wert zurück.
  - Eine Bedingung zur Abfrage von bestimmten Werten gibt einen boolschen Wert zurück.
- ... können in SELECT- und WHERE-Anweisungen genutzt werden.

- Als Operand kann
  - ... die Bezeichnungen einer Spalte (Feldnamen, Attribut) einer Tabelle oder
  - ... ein konstanter Wert genutzt werden.
- Operatoren
  - ... verknüpfen Werten.
  - ... vergleichen Werte.
  - ... berechnen Werte.
- Beispiele:
  - `artikel.preis * artikel.menge`
  - `preis + mehrwertsteuer`
  - `lager.menge > 0`
  - `BETWEEN '2008.01.01' AND '2008.06.31'`
  - `(lagermenge > 0) AND (kategorie = 'CDPlayer')`

Operator	Beschreibung	Beispiel
+	Addition zweier Werte	$3 + 4 = 7$
-	Subtraktion zweier Werte	$4 - 3 = 1$
*	Multiplikation zweier Werte	$3 * 4 = 12$
/	Division zweier Werte	$4.8 / 2.3 = 2,086$
%	Division mit Rest von Ganzzahlen	$4 \% 3 = 1$

```
SELECT preis, menge, (preis * menge)
FROM lager;
```

- In der SELECT-Anweisung kann statt dem Namen eines Datenfeldes auch ein Ausdruck angegeben werden.
- Wegen der besseren Lesbarkeit wird der Ausdruck geklammert.
- Beachten Sie die Reihenfolge der Operatoren!

Vorzeichen	+ -
Mathematische Op.	* / % + -
Vergleichsop.	= > < >= <= <>
Logische Op.	NOT AND BETWEEN IN LIKE OR
Zuweisungsop.	=

```
SELECT preis, menge, (preis * menge) AS gesamt  
FROM lager;
```

- Mit Hilfe des Schlüsselwortes `AS` wird ein Alias-Name für eine Spalte vergeben.
- Eine Spalte, die den Wert eines Ausdrucks enthält, bekommt standardmäßig als Namen den Ausdruck.
- Durch den Alias-Namen wird der temporären Spalte ein selbsterklärender Namen zugewiesen.

```
SELECT preis, menge  
FROM lager  
WHERE (preis * menge) > 10;
```

- Die Bedingung selber ist ein Ausdruck.
- Der zu überprüfende Wert wird durch einen mathematischen Ausdruck berechnet.

```
UPDATE tabelle
SET
    spalte = wert,
    spalte = wert,
    ...
WHERE bedingung;
```

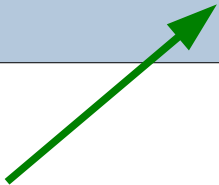
- UPDATE (ändere Datensätze) tabelle.
- SET (setze) die Spalten vor dem Gleichheitszeichen auf den Wert nach dem Gleichheitszeichen. Die zu verändernden Spalten werden mit einem Komma getrennt.
- Am Ende der Anweisung kann eine Bedingung für die Änderung angegeben werden. Falls keine Bedingung angegeben wird, werden alle Daten der angegebenen Spalte geändert.

```
UPDATE
SET preis = preis * 1,01
WHERE preis < 3;
```

```
UPDATE artikel
SET artikelnr = '3-2456-6' WHERE artikelname = 'Apfel';

UPDATE lager
SET packet = '2 kg', preis = 2.4 WHERE artikel = 'Apfel';

UPDATE verkauf
SET preis = preis * 0.2 WHERE preis < 2;
```



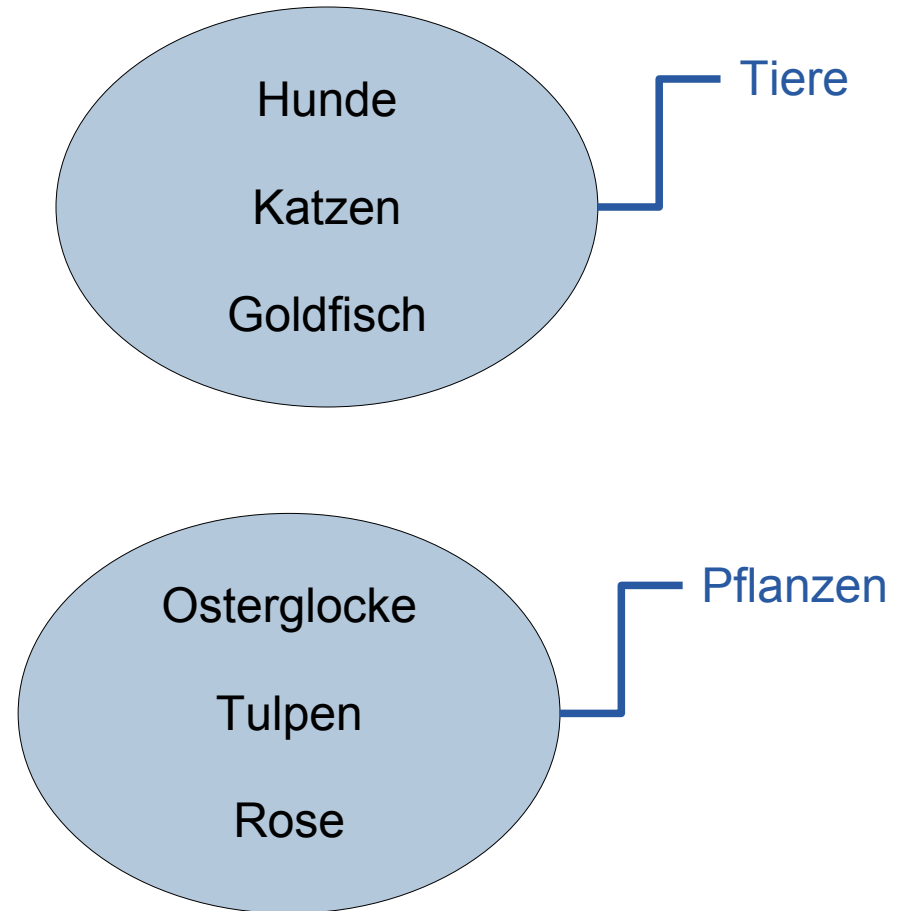
Es können alle  
mathematischen Ausdrücke  
genutzt werden.

```
CREATE TABLE mitarbeiter_08(  
  nachname VARCHAR(50),  
  vorname VARCHAR(20),  
  eintritt VARCHAR(4)  
);  
  
INSERT INTO mitarbeiter(nachname, vorname, eintritt)  
VALUES  
  ('Müller', 'Sven', '2007'),  
  ('Meier', 'Ute', '2008'),  
  ('Scholz', 'Heinz', '2008');  
  
INSERT INTO mitarbeiter_08(nachname, vorname, eintritt)  
SELECT nachname, vorname, eintritt  
FROM mitarbeiter  
WHERE eintritt = '2008';
```

kunde where plzLIKE '38\*' -> newTable

- Daten werden nach bestimmten Kriterien zusammengefasst.
- Kategorisierung von Objekten.
- Daten, die einen gleichen Wert besitzen, werden zusammengefasst.
- Datengruppierungen beantworten Fragen in der Form:
  - Welche Projekte bearbeitet Mitarbeiter A?
  - Welche Mitarbeiter arbeiten in einer Abteilung?
  - Welche Gemüseorten sind vorrätig?
  - Welche Studenten legen Prüfung X ab?
  - Welcher Autor hat welche Bücher geschrieben.

Tiere: Hunde  
Pflanzen: Osterglocke  
Tiere: Katzen  
Pflanzen: Tulpen  
Pflanzen: Rose  
Tiere: Goldfisch



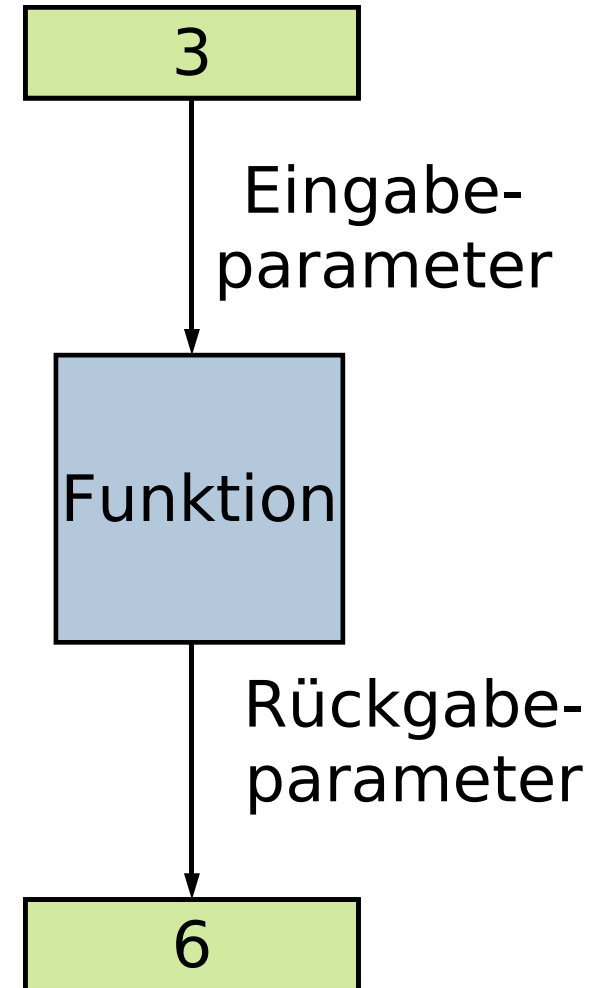
Die Daten werden nach einem bestimmten Feld zusammengefasst.

```
CREATE TABLE welt(  
  kategorie VARCHAR(15),  
  objektname VARCHAR(50),  
  kontinent VARCHAR(20)  
);  
  
SELECT * FROM welt  
  GROUP BY kategorie;  
  
SELECT * FROM welt  
  GROUP BY kategorie  
  HAVING kontinent = 'Europa';
```

Hier werden nur Datengruppen angezeigt, die ein bestimmtes Kriterium erfüllen. HAVING entspricht WHERE und kann nur für Datengruppen genutzt werden.

- ... sind vordefinierte Programme, die eine bestimmte Aufgabe lösen.
- ... haben eine bestimmte Anzahl von Eingabeparametern.
- ... geben einen Wert zurück.
- ... verändern nicht den gespeicherten Wert in einem Datenfeld.
- ... können in einer SELECT- sowie in einer WHERE-Anweisung genutzt werden.
- In dem Standard SQL99 sind einige Funktionen definiert.
  - Jeder Datenbankersteller ergänzt diese Funktionen um eigene.
  - Teilweise nutzen Datenbankersteller andere Eingabeparamater oder andere Bezeichnung als der Standard vorzieht.
- Aber viele SQL-Implementierungen bieten weitere Funktionen an.

- In einer Funktion wird eine bestimmte Aufgabe gelöst. Der Nutzer der Funktion muss nicht wissen, wie die Aufgabe gelöst wird. Dem Nutzer genügt es zu wissen, wie die Funktion aufgerufen wird. Die Funktion ist eine Blackbox.
- Der Funktion können Werte übergeben werden. Dem Nutzer ist nur die Anzahl und der Typ der zu übergebenden Parameter bekannt. Einige Funktionen besitzen keine Übergabeparameter.
- Die Lösung der Aufgabe kann an den Aufrufer zurückgegeben werden. Nicht jede Funktion besitzt einen Rückgabeparameter.



- ... werden häufig auf eine Datengruppe angewendet.
- ... können auf die gesamte Tabelle angewendet werden.
- ... ermitteln einen Wert in Bezug auf eine bestimmte Spalte.
- ... fassen eine Sammlung von Werten zu einem Wert zusammen.
- NULL-Werte werden nicht berücksichtigt.
- ... können nur in SELECT- oder HAVING-Anweisungen genutzt werden.

- Folgende Funktionen sind häufig definiert:

Funktion	Beschreibung
<code>COUNT ()</code>	Anzahl der Nicht-Null-Felder einer Spalte
<code>COUNT (DISTINCT)</code>	Anzahl der unterschiedlichen Nicht-Null-Felder
<code>Count (*)</code>	Anzahl der aller Datensätze
<code>AVG ()</code>	Durchschnittswert aller Nicht-Null-Felder
<code>MIN ()</code>	Kleinster Wert eines Datenfeldes
<code>MAX ()</code>	Größter Wert eines Datenfeldes
<code>SUM ()</code>	Summe der Nicht-Null-Felder einer Spalte

- Standardmäßig sind nur die Aggregatfunktionen `AVG`, `MIN`, `MAX` und `SUM` definiert.
- Für die Aggregatfunktionen, bis auf `COUNT()`, werden numerische Werte verarbeitet.

Hier wird durch die GROUP BY die Anzahl der Objekte in einer bestimmten Gruppe gezählt.

```
CREATE TABLE welt(  
  kategorie VARCHAR(15),  
  objektname VARCHAR(50),  
  kontinent VARCHAR(20)  
);  
  
SELECT kategorie, COUNT(objektname) FROM welt  
  GROUP BY kategorie;  
  
SELECT kategorie, COUNT(objektname) FROM welt  
  GROUP BY kategorie  
  HAVING kontinent = 'Europa';
```

```
CREATE TABLE welt(  
  kategorie VARCHAR(15),  
  objektname VARCHAR(50),  
  kontinent VARCHAR(20)  
);  
  
SELECT COUNT(objektname) FROM welt;
```

- Eine Kombination von Datenfeldern und Aggregatfunktionen ist nicht erlaubt. Ausnahme: Die Datenfelder werden zu Gruppen zusammengefasst.
- Die Aggregatfunktion bezieht sich auf alle Datensätze in der angegebenen Tabelle.

- ... können
  - Datenfelder einer Tabelle,
  - konstante Werte oder
  - Ergebnisse von Berechnungen übergeben bekommen.
- ... liefern ein Ergebnis zurück.
- ... gibt es für folgende Bereiche:
  - Datums- und Zeitberechnungen.
  - Trigometrische Funktionen.
  - Stringfunktionen.
  - etc.
- Deren Implementierung ist abhängig vom gewählten Datenbankprogramm.

```
SELECT messungID, messwert, sin(messwert) FROM messung;
```

```
SELECT bestellungID, kunde FROM bestellung  
WHERE length(bestellungID) > 5;
```

```
SELECT kunde, ort FROM kunde  
WHERE substring(ort, 1, 5) = '38112';
```

```
INSERT INTO bestellung (kunde, artikel, menge, datum)  
VALUES ('Meier', 'Banane', 5, CURRENT_DATE),  
      ('Schmidt', 'Äpfel', 3, CURRENT_DATE);
```