Begriffe

- SQL … Structured Query Language
  - DDL … Data Definition Language
  - DML … Data Manipulation Language
  - DQL … Data Query Language

SQL

Die **Structured Query Language** ist eine **Abfragesprache**.

Sie legt fest, **welche Daten** zu verarbeiten sind.
Im Gegensatz zu prozeduralen Sprachen wird aber **nicht** festgelegt, **wie** diese Daten zu verarbeiten sind.

- SQL wurde vom **ANSI**-Komitee 1986 genormt (SQL-86).
- 1992 wurde die Norm erweitert (**SQL-92**), es entstand der **SQL-2-Standard**, bestehend aus ca. 200 Schlüssel-Wörtern. Dieser gilt derzeit als „**Stand der Technik**“.
- 1999 (SQL-99 bzw. SQL-3-Standard) und 2003 sowie 2008 wurde die Norm nochmals erweitert. Sie enthält nun u. a. auch objektrelationale Erweiterungen.

Groß- / Kleinschreibung

SQL ist **nicht case-sensitive**, d.h. bei Schlüsselwörtern, Tabellennamen und Spaltennamen ist es egal, ob die Groß- oder die Kleinschreibung verwendet wird.

Vorschlag: Schlüsselwörter in Großbuchstaben, benutzerdefinierte Namen (Datenbankname, Tabellennamen, Spaltennamen) in Kleinbuchstaben:

```
CREATE DATABASE dbname;

DROP TABLE tabname;

SELECT * FROM tabname;
```

Strings, Anführungszeichen

Strings werden in SQL – im Gegensatz zu vielen anderen Programmiersprachen – in **einfachen Anführungszeichen** eingeschlossen. Anführungszeichen in Strings entwerten!

```
INSERT INTO tabname VALUES (1, 'abc');

SELECT * FROM tabname WHERE xyz = 'abc';

INSERT INTO genre VALUES ('Rock\'n\'Roll';
```

Achtung:
Im Gegensatz zu anderen DBMS lässt MySQL einfache und doppelte Anführungszeichen für Strings zu. Da aber doppelte Anführungszeichen in SQL eine spezielle Bedeutung haben, sollten diese für Strings auch in MySQL nicht verwendet werden!

Zeilenvorschübe, Terminatoren

Eine SQL-Anweisung muss **immer** mit einem **Strichpunkt** beendet werden. Dazwischen enthaltene **Zeilenvorschübe** und zusätzliche **Leerzeichen** sind für die Ausführung der SQL-Anweisung **unerheblich**, werden aber zur besseren Lesbarkeit dringend empfohlen. **Kommas** als Trennzeichen.

```
-- DROP TABLE IF EXISTS tabname;
CREATE TABLE tabname (
                spaltenname1      datentyp,
                spaltenname2      datentyp,
                ...               ...,
                spaltenname       datentyp
);
```

Anweisungen, Klauseln

SQL-**Anweisungen** werden nach dem ersten verwendeten **Schlüsselwort** (CREATE, INSERT, SELECT, ...) benannt und bestehen aus mehreren **Klauseln**, die ebenfalls wieder nach dem betreffenden Schlüsselwort benannt werden.
Die **Reihenfolge** der Klauseln ist **einzuhalten**.

Beispiel: die SELECT-Anweisung besteht aus:

```
  SELECT - Klausel
[ FROM - Klausel     ]
[ WHERE - Klausel    ]
[ GROUP BY - Klausel ]
[ HAVING - Klausel   ]
[ ORDER BY - Klausel ];
```

DQL

Die **Data Query Language** beschreibt, wie man Daten mit Hilfe der **SELECT-Anweisung** aus der Datenbank abfragt.

Die Syntax der SELECT-Anweisung ist besonders umfangreich:

```
  SELECT - Klausel
[ FROM - Klausel     ]
[ WHERE - Klausel    ]
[ GROUP BY – Klausel ]
[ HAVING – Klausel   ]
[ ORDER BY – Klausel ];
```

SELECT-Klausel

Die SELECT-Klausel definiert, **welche Spalten** ermittelt werden sollen. Dies können **entweder bestehende Spalten** aus den abgefragten Tabellen sein **oder neue Spalten** sein, die nur für die Abfrage gebildet werden.

Beispiele:

```
SELECT * FROM interpret;
SELECT iname, homepage FROM interpret;
```

Mit „*“ werden alle Spalten der angegebenen Tabelle gelesen. Falls nur bestimmte Spalten gelesen werden sollen, müssen diese explizit angegeben werden.

FROM-Klausel

Die FROM-Klausel definiert, **welche Tabellen** für die Abfrage verwendet werden sollen.

Beispiel:

```
SELECT iname, stitel
  FROM interpret, song;
```

Achtung: diese Abfrage liefert das **kartesische Produkt**.
Jede Zeile der Tabelle `interpret` wird mit jeder Zeile der Tabelle `song` kombiniert und als Ergebniszeile angezeigt.
Für die meisten Anforderungen ist dieses Ergebnis sinnlos.
Siehe auch „Abfrage über mehrere Tabellen“ bzw. „JOIN“.

WHERE-Klausel

Die WHERE-Klausel **schränkt die Treffermenge** ein**.**

Beispiel:

```
SELECT stitel
  FROM song
 WHERE sdauer > 10;
```

- Zulässige Vergleichsoperatoren:
  `>, <, >=, <=, =, <>, !=`
- Vergleiche mit NULL:
  ... `IS NULL`
  ... `IS NOT NULL`

LIKE

Mit der Angabe von „LIKE“ als Vergleichsoperator können **teilqualifizierte Suchabfragen** implementiert werden.

```
SELECT iname
  FROM interpret
 WHERE iname LIKE 'A%';
```

- Mögliche Angaben für das Suchmuster:
  `%` … beliebige Zeichenfolge (ein / kein / mehrere Zeichen)
  `_` … ein beliebiges Zeichen
- Diese Angaben können auch kombiniert werden:
  z.B. `WHERE iname LIKE '_a%'`
  liefert alle Treffer mit 'a' an der zweiten Stelle im Namen

Ergebnisspalten (um)benennen

Durch die Angabe **AS " …"** können Ergebnisspalten (um)benannt werden:

Beispiele:

```
SELECT stitel AS "Songtitel" FROM song;
SELECT sum(sdauer) AS "Ges.Dauer" FROM song;
SELECT count(*) AS "Anzahl" FROM interpret;
```

Achtung: Strings müssen in SQL durch einfache Anführungszeichen (z.B. `WHERE name = 'abc'`) umschlossen werden. Hingegen zum Umbenennen von Spalten doppelte gerade Anführungszeichen ("abc") vorgesehen, niemals aber typografische Anführungszeichen („bla“).

ORDER-BY-Klausel

Mit Hilfe der ORDER-BY-Klausel können die in einer SELECT-Anweisung anzuzeigenden Datensätze **sortiert** werden:

```
SELECT * FROM song ORDER BY dauer ASC;
SELECT * FROM song ORDER BY dauer DESC;

SELECT sum(dauer) AS "summe"
  FROM song GROUP BY iid
  ORDER BY summe;
```

- `ASC` … sortiert aufsteigend (Default)
- `DESC` … sortiert absteigend
- Mit „AS“ wird eine (temporäre) Spalte benannt, damit sie in einer nachfolgenden Klausel verwendet werden kann.