SQL_DQL_Cheatsheet_#1

SQL

Die Abkürzung SQL steht für den Begriff **S**tructured **Q**uery **L**anguage und bezeichnet eine Sprache für die Kommunikation mit relationalen Datenbanken. Mit SQL-Befehlen lassen sich Daten relativ einfach einfügen, verändern, löschen und abfragen.

Beispiel Daten

country

Code	Name	Population	LifeExpectancy
FRA	France	59225700	78.8
DEU	Germany	82164700	77.4
…	…	…	…

city

ID	Name	CountryCode	Population	District
2974	Paris	FRA	2125246	Île-de-France
3106	Erfurt	DEU	201267	Thüringen
…	…	…	…	…

Abfragen über eine Tabelle

Abfrage aller Spalten der Tabelle city:

```
SELECT *
FROM city;
```

Abfrage der Spalten ID und Name von der Tabelle city:

```
SELECT ID, Name
FROM city;
```

Abfrage aller Spalten der Tabelle city mit absteigender Sortierung nach der Einwohnerzahl:

```
SELECT *
FROM city
ORDER BY Population DESC;
```

- *Aufsteigende Sortierung ASC*
- *Absteigende Sortierung DESC*

Alias

Mit der AS – Klausel können den Spalten einer Abfrage alternative Spaltennamen zugeordnet werden. Hier LifeExpectancy → Lebenserwartung:

```
SELECT LifeExpectancy AS Lebenserwartung
```

Gefilterte Abfragen über eine Tabelle

Vergleichsoperatoren

Abfrage aller Länder, die mehr als 50 Millionen Einwohner haben:

```
SELECT Name
FROM country
WHERE Population > 50000000;
```

- *Operatoren: >, >=, =, <=, <, !=*

Abfrage aller Städtenamen außer Fürth:

```
SELECT Name
FROM city
WHERE Name != 'Fürth';
```

Textoperatoren

Abfrage aller Städtenamen, die mit 'P' beginnen oder 's' enden:

```
SELECT Name
FROM city
WHERE Name like 'P%'
   OR Name like '%s';
```

Abfrage aller Städtenamen die mit einem beliebigen Buchstaben beginnen, gefolgt von 'ublin' (Dublin in Irland oder Lublin in Polen):

```
SELECT Name
FROM city
WHERE Name like '_ublin';
```

Weitere Operatoren

Abfrage aller Städte zwischen 50.000 und 250.000 Einwohner:

```
SELECT Name
FROM city
WHERE Population BETWEEN 50000 AND 250000;
```

Abfrage aller Länder mit unbekannter Lebenserwartung (Eintrag fehlt):

```
SELECT Name
FROM country
WHERE LifeExpectancy IS NULL;
```

Abfrage aller Städtenamen in Deutschland und Frankreich:

```
SELECT Name
FROM city
WHERE CountryCode IN ('DEU', 'FRA');
```

Logische Verknüpfungen

Abfrage aller Länder mit mehr als 25 Millionen Einwohner und einer Lebenserwartung über 75 Jahre:

```
SELECT Name
FROM country
WHERE Population > 25000000 AND LifeExpectancy > 75;
```

- *Operatoren: AND, OR, NOT*
- *Eingeklammerte Verknüpfungen werden zuerst ausgewertet*
- *NOT geht vor AND geht vor OR*

Sortierte gefilterte Abfrage über eine Tabelle

Abfrage aller Städte mit mehr als 100.000 Einwohner in Deutschland und aufsteigende Sortierung nach der Einwohnerzahl:

```
SELECT *
FROM city
WHERE Population > 100000 AND CountryCode = 'DEU'
ORDER BY Population ASC;
```

Aggregatfunktionen

Anzahl aller Städte bestimmen und mit dem Alias Anzahl ausgeben:

```
SELECT COUNT(*) AS Anzahl
FROM city;
```

Die kleinste Einwohnerzahl bestimmen:

```
SELECT MIN(Population) AS 'Kleinste Einwohnerzahl'
FROM city;
```

Die größte Einwohnerzahl bestimmen:

```
SELECT MAX(Population) AS 'Größte Einwohnerzahl'
FROM city;
```

Die weltweite Gesamtbevölkerung bestimmen:

```
SELECT SUM(Population) AS Gesamtbevölkerung
FROM country;
```

Die mittlere weltweite Lebenserwartung bestimmen:

```
SELECT AVG(LifeExpectancy) AS 'Mittlere Lebenserwartung'
```

SQL_DQL_Cheatsheet_#2

Reihenfolge Abarbeitung SQL – Interpreter

Klausel	Schritt	Erläuterung
SELECT	5	Auswahl aller genannter Spalten bzw. Gruppen, Aggregatfunktionen
FROM	1	Auswahl Tabelle
WHERE	2	Abfragebedingung von Spalten
GROUP BY	3	Gruppierung
HAVING	4	Abfragebedingung von Gruppen
ORDER BY	6	Sortiert Zeilen auf- oder absteigend nach Spalte(n)
LIMIT	7	Streicht Zeilen

Beispiel Daten

country

Code	Name	Continent	Population	GovernmentForm	Capital
FRA	France	Europe	59225700	Federal Republic	2974
DEU	Germany	Europe	82164700	Republic	3068
TUR	Turkey	Asia	66591000	Republic	3358
…	…	…	…		

city

ID	Name	CountryCode	Population	District
2974	Paris	FRA	2125246	Île-de-France
3106	Erfurt	DEU	201267	Thüringen
…	…	…	…	…

LIMIT – Klausel; Begrenzung Anzahl Zeilen

Abfrage der 5 größten Städte in Deutschland:

```
SELECT *
FROM city
WHERE CountryCode = 'DEU'
ORDER BY Population DESC
LIMIT 5;
```

GROUP BY – Klausel; Gruppieren

Abfrage der Gesamtbevölkerung von Städten in den jeweiligen Ländern:

```
SELECT CountryCode, SUM(Population) AS Stadtbewohner
FROM city
GROUP BY CountryCode;
```

Die GROUP BY – Klausel kann auch auf mehrere Spalten angewendet werden. Eine Verwendung von Aliasen außerhalb der SELECT Klausel sollte vermieden werden, da dies nicht alle SQL – DBMS gleich unterstützen (siehe Reihenfolge Schritte SQL -Interpreter). Bei gruppierten Daten gibt man mit der SELECT - Klausel nur die gruppierten Spalten und gegebenenfalls aggregierten Daten aus.

Gruppierung nach Kontinent und Regierungsform, Zählen der gleichen Einträge und Ausgabe der 3 Häufigsten.

```
SELECT Continent, GovernmentForm, count(GovernmentForm) AS Anzahl
FROM country
GROUP BY Continent, GovernmentForm
ORDER BY count(GovernmentForm) DESC
LIMIT 3;
```

```
+-----------+----------------+--------+
| Continent | GovernmentForm | Anzahl |
+-----------+----------------+--------+
| Africa    | Republic       |     46 |
| Asia      | Republic       |     26 |
| Europe    | Republic       |     25 |
+-----------+----------------+--------+
```

Having – Klausel; Abfragebedingung mit gruppierten Daten und ggf. Aggregatfunktion

Die HAVING – Klausel ermöglicht es, gruppierte Datensätze zu filtern. Man kann die HAVING – Klausel nur auf gruppierte Datensätze anwenden!
Die HAVING – Klausel wird erst angewendet, nachdem gegebenenfalls mit der der WHERE – Klausel Datensätze selektiert wurden.

Ausgabe der Länderkennzeichen und der Anzahl der Großstädte (Bevölkerung > 100.000) für Länder mit mehr als 200 Großstädte:

```
SELECT CountryCode, count(name) AS AnzahlGroßstädte
FROM city
WHERE Population > 100000
GROUP BY CountryCODE
HAVING count(name) > 200
ORDER BY count(name) DESC;
```

```
+-------------+------------------+
| CountryCode | AnzahlGroßstädte |
+-------------+------------------+
| CHN         |              341 |
| IND         |              310 |
| USA         |              245 |
| JPN         |              226 |
| BRA         |              216 |
+-------------+------------------+
```

Unterabfragen – Subqueries

Bei der Verwendung mehrerer Tabellen sollte man den Spaltennamen und den Tabellennamen mit einem Punkt verbinden, da es möglich ist, dass Spalten die gleiche Bezeichnung haben (Spalte Name in Tabelle country und city). Auch für Tabellennamen können Aliase verwendet werden.

z.B. Abfrage aller Städtenamen

```
SELECT Name
FROM city;
```

```
SELECT city.Name
FROM city;
```

```
SELECT cy.Name
FROM city AS cy;
```

Unterabfrage in WHERE – Klausel

Abfrage aller Städte die mindestens so viele Einwohner, wie Berlin haben:

```
SELECT Name, Population
FROM city
WHERE Population >= (
    SELECT Population
    FROM city
    WHERE Name = 'Berlin'
);
```

Unterabfrage in SELECT – Klausel

Abfrage aller europäischen Hauptstädte von Ländern mit mehr als 60 Millionen Einwohner (Ausgabe Land, Bevölkerung, Hauptstadt und Bevölkerung):

```
SELECT co.Name AS Land,
       co.Population AS Einwohner,
       (SELECT cy.Name
        FROM city AS cy
        WHERE cy.ID = co.Capital)
        AS Hauptstadt,
       (SELECT cy.Population
        FROM city AS cy
        WHERE cy.ID = co.Capital)
        AS Einwohner
FROM country AS co
WHERE co.Continent = 'Europe' AND co.Population > 60000000;
```

```
+--------------------+-----------+------------+-----------+
| Land               | Einwohner | Hauptstadt | Einwohner |
+--------------------+-----------+------------+-----------+
| Germany            |  82164700 | Berlin     |   3386667 |
| Russian Federation | 146934000 | Moscow     |   8389200 |
+--------------------+-----------+------------+-----------+
```

SQL_DDL_Cheatsheet_#4

DDL Data Definition Language

- Definition von Daten und Datenfeldern
- Erzeugung der Strukturen und Beziehungen zueinander

Wichtige SQL-Datentypen

Datentyp	Verwendung → Numerische Datentypen hier signed
INT	Ganze Zahlen von -2147483648 bis 2147483647
BIGINT	Ganze Zahlen von -9223372036854775808 bis 9223372036854775807
FLOAT	Fließkommazahlen von -3.402823466E+38 bis 3.402823466E+38
VARCHAR(n)	Zeichenkette mit variabler Größe, max n Zeichen, 1 – 255 Zeichen möglich
CHAR(n)	Zeichenkette mit genau n Zeichen
TEXT	Zeichenkette, max 65535 Zeichen
MEDIUMTEXT	Zeichenkette, max 16777215 Zeichen
DATE	YYYY-MM-DD
TIME	HH:MM:SS.ssssss
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYY-MM-DD HH:MM:SS; UTC → unabhängig von Zeitzone

Im folgenden Beispiel wird exemplarisch eine Datenbank zur Mitgliederverwaltung eines Vereins gezeigt

Datenbank anlegen und verwenden

```
CREATE DATABASE Verein;
USE Verein
```

Tabelle anlegen

Beim Erstellen einer Tabelle muss ihre Struktur vorgegeben werden. Diese wird bestimmt durch:

- Spaltennamen + Datentyp
- Primärschlüssel, weitere Bedingungen und Attribute

Beispiel:

```
CREATE TABLE Mitglieder(
     PK_ID_Verein INT PRIMARY KEY AUTO_INCREMENT,
     Name VARCHAR(20) NOT NULL,
     Nutzername VARCHAR(20) UNIQUE,
     Telefonnummer INT
);
```

```
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| PK_ID_Verein  | int(11)     | NO   | PRI | NULL    | auto_increment |
| Name          | varchar(20) | NO   |     | NULL    |                |
| Nutzername    | varchar(20) | YES  | UNI | NULL    |                |
| Telefonnummer | int(11)     | YES  |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
```

ID_Verein INT PRIMARY KEY AUTO_INCREMENT
Festlegung der Spalte ID_Verein als Primärschlüssel. Das DBMS verhindert doppelte Werte und leere Einträge.
Durch die **optionale** Verwendung von AUTO_INCREMENT erfolgt eine automatische Zuweisung der Werte.

Name VARCHAR(20) NOT NULL
Verhindert, dass die Spalte leer bleibt.

Nutzername VARCHAR(20) UNIQUE
Durch die Vorgabe mit UNIQUE wird die Einzigartigkeit von Einträgen sichergestellt, jedoch darf das Feld leer bleiben.

Beziehungsintegrität PRIMARY KEY → FOREIGN KEY

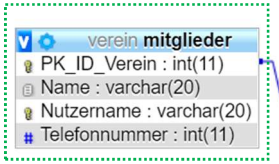
Mögliche Bedingungen beim Löschen (ON DELETE) bzw. Ändern (ON UPDATE) eines Datensatzes in der „Elterntabelle“:

Bedingung	Bedeutung
RESTRICT	Inhalte des Primärschlüssels in der Elterntabelle können nicht gelöscht / geändert werden
CASCADE	Werden Datensätze in der Elterntabelle gelöscht / geändert, dann auch in allen Kindertabellen

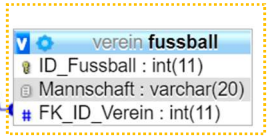
Beispiel Beziehung PRIMARY KEY → FOREIGN KEY:

```
CREATE TABLE Fussball (
     ID_Fussball INT PRIMARY KEY AUTO_INCREMENT,
     Mannschaft VARCHAR(20) NOT NULL,
     FK_ID_Verein INT,
     CONSTRAINT FK_Mitglieder_Fussball
         FOREIGN KEY (FK_ID_Verein) REFERENCES Mitglieder (PK_ID_Verein)
         ON DELETE CASCADE
         ON UPDATE RESTRICT
);
```

Elterntabelle:



Kindertabelle:



Zusammengesetzte Primärschlüssel

```
CREATE TABLE Bankverbindung(
     Bankleitzahl CHAR(8) NOT NULL,
     Kontonummer CHAR(10) NOT NULL,
     PRIMARY KEY (Bankleitzahl, Kontonummer)
);
```

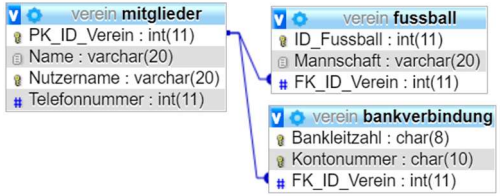
Änderungen an einer Tabelle vornehmen

Hinzufügen einer Spalte für den Fremdschlüssel:

```
ALTER TABLE Bankverbindung
ADD FK_ID_Verein INT;
```

Definition von FK_ID_Verein als Fremdschlüssel

```
ALTER TABLE Bankverbindung
ADD Constraint FK_Mitglieder_Bankverbindung
    FOREIGN KEY (FK_ID_Verein) REFERENCES Mitglieder (PK_ID_Verein)
    ON DELETE CASCADE
    ON UPDATE RESTRICT
;
```



Änderung des Datentyps einer Spalte:

```
ALTER TABLE Tabellenname
MODIFY Spaltenname Neuer Datentyp;
```

Primärschlüssel nachträglich definieren:

```
ALTER TABLE Tabellenname
ADD PRIMARY KEY (Spaltenname);
```

Tabelle umbenennen löschen

Tabelle umbenennen:

```
RENAME TABLE Mitglieder TO Mitglieder_Alt;
```

Tabelle löschen:

```
DROP TABLE Mitglieder_Alt;
```

SQL_DML_DCL_Cheatsheet_#5

DML Data Manipulation Language

Die DML wird verwendet, um in einer Datenbank Daten:

- Einzufügen → INSERT
- zu Ändern → UPDATE
- zu Löschen → DELETE

Beispiel Tabelle Mitglieder:

Field	Type	Null	Key	Default	Extra
PK_ID_Verein	int(11)	NO	PRI	NULL	auto_increment
Nachname	varchar(20)	NO		NULL	
Nutzername	varchar(20)	YES	UNI	NULL	
Telefonnummer	varchar(14)	YES		NULL	
Mitgliedsbeitrag	decimal(6,2)	YES		NULL	

Datensätze einfügen INSERT

Das Einfügen von Datensätzen erfolgt mithilfe der Anweisung INSERT. Geben Sie stets die Spalten an. Auto_Increment Spalten sollten nicht ausgefüllt werden.

Syntax:

```
INSERT INTO Tabellenname
(Spalte1, Spalte2, Spalte3, …, SpalteX)
VALUES
('Wert1', 'Wert2', 'Wert3', 'WertX');
```

Beispiel:

```
INSERT INTO Mitglieder
(Nachname, Nutzername, Telefonnummer, Mitgliedsbeitrag)
VALUES
('Müller', 'Joe', '0911996633','47.98');
```

Mehrere Datensätze einfügen

Syntax:

```
INSERT INTO Tabellenname
(Spalte1, Spalte2, Spalte3, …, SpalteX)
VALUES
('Wert1', 'Wert2', 'Wert3', 'WertX');
```

Beispiel:

```
INSERT INTO Mitglieder
(Nachname, Nutzername, Telefonnummer, Mitgliedsbeitrag)
VALUES
('Huahi', 'Julia', '09111236633','47.98'),
('Prim', 'Theresa', '0911993453','34.98'),
('Koch', 'Marvin', '09111325253','28.28');
```

Datensätze aus einer bestehenden Tabelle einfügen

Beim Einfügen aus einer anderen Tabelle müssen die Spalten (Datentyp und Anzahl) identisch sein.

Syntax:

```
INSERT INTO Tabellenname1
(Spalte1, Spalte2, Spalte3, …, SpalteX)
SELECT
(Spalte1, Spalte2, Spalte3, …, SpalteX)
FROM Tabellename2
WHERE Bedingung;
```

Datensätze löschen DELETE

Das Löschen von Datensätzen erfolgt mithilfe der Anweisung DELETE. Verwenden Sie stets den Primärschlüssel, um einen Datensatz eindeutig zu identifizieren.

Syntax:

```
DELETE FROM Tabellenname
WHERE SpalteX = 'Wert';
```

Beispiel:

```
DELETE FROM Mitglieder
WHERE PK_ID_Verein = 5;
```

Datensätze ändern UPDATE

Das Ändern von Datensätzen erfolgt mithilfe der Anweisung UPDATE. Um nicht alle Datensätze einer Spalte zu verändern, sollte mithilfe der WHERE – Klausel eine Bedingung eingegeben werden. Verwenden Sie stets den Primärschlüssel, um einen Datensatz eindeutig zu identifizieren.

Syntax:

```
UPDATE Tabellenname
SET FeldX = 'NeuerWert'
WHERE Bedingung;
```

Beispiel:

```
UPDATE Mitglieder
SET Telefonnummer = '095171144'
WHERE PK_ID_Verein = 22;
```

Beispiel Änderung aller Einträge einer Spalte:

Erhöhung des Mitgliedsbeitrags aller Mitglieder um 2 Prozent.

```
UPDATE Mitglieder
SET Mitgliedsbeitrag = Mitgliedsbeitrag * 1.02;
```

DCL Data Control Language

Die DCL wird verwendet, um in einem Datenbankmanagementsystem Benutzer und Berechtigungen zu verwalten:

- Benutzer anlegen → CREATE
- Rechte Vergeben → GRANT
- Rechte Entziehen → REVOKE
- Benutzer löschen → DROP

Die folgenden Beispiele zeigen nur einen kleinen Auszug aus der DCL. Die komplette Dokumentation finden Sie hier: https://mariadb.com/kb/en/account-management-sql-commands/

Benutzer anlegen CREATE

```
CREATE USER 'Benutzername' IDENTIFIED BY 'Passwort';
```

Rechte vergeben GRANT

Zugriff auf Tabelle Mitglieder in der Datenbank Verein:

```
GRANT ALL ON Verein.Mitglieder TO 'Benutzername';
```

Zugriff auf die komplette Datenbank Verein:

```
GRANT ALL ON Verein.* TO 'Benutzername';
```

Zugriff auf alle Datenbanken:

```
GRANT ALL ON *.* TO 'Benutzername';
```

Nutzer darf nur neue Werte in der Datenbank Verein eingeben:

```
GRANT INSERT ON Verein.* TO 'Benutzername';
```

Nutzer darf nur Werte aus der Datenbank Verein auslesen:

```
GRANT SELECT ON Verein.* TO 'Benutzername';
```

Rechte entziehen REVOKE

Nutzer Rechte von der Datenbank Verein entziehen:

```
REVOKE DROP ON Verein.* FROM 'Benutzername';
```

Benutzer löschen DROP

```
DROP USER 'Benutzername';
```