

---

# INFI – Protokoll

## Datenbanken



"[Dieses Foto](#)" von Unbekannter Autor ist lizenziert gemäß [CC BY-SA-NC](#)

### Ausgeführt von:

Michelle Lechner

Wien, am 23.09.2022

## Inhaltsverzeichnis

1	Wiederholung / Einführung in SQL	3
1.1	SQL – Basics auf Khanacademy.org	3
1.1.1	Book List Challenge	3
1.1.2	Box Office	4
1.1.3	ToDo - List	4
1.1.4	Projekt: Design a store database	5
1.2	Komplexe Abfragen mit AND/ OR	6
1.2.1	Karaoke song selector	6
1.2.2	Playlist maker	6
1.2.3	The wordiest Author	7
1.2.4	Gradebook	7
	Abbildungsverzeichnis	8
	Tabellenverzeichnis	9
	Literaturverzeichnis	10
	Anhang	11

# 1 Wiederholung / Einführung in SQL

## 1.1 SQL – Basics auf Khanacademy.org

Difference between MySQL and SQLite:

S.NO.	MySQL	SQLite
1.	Developed by Oracle on May 1995.	Developed By D. Richard Hipp on August 2000.
2.	MySQL is developed in C and C++ languages.	SQLite is developed only in C language.
3.	MySQL requires a database server for its functioning. Hence, it follows client/server architecture.	SQLite does not require a server to run. Hence, it is serverless.
4.	It can handle multiple connections simultaneously.	It can handle only one connection at a time.
5.	It is highly scalable and can handle a large volume of data very efficiently.	It can handle only small set of data if the volume of data increased its performance degrades.
6.	It requires large space in the memory for its functioning (approx 600 Mb).	It requires only some KBs of space as it is very lightweight approx (250Kb-300Kb).
7.	MySQL supports multiple user environment.	SQLite does not support multiple user environment.
8.	It also supports XML format.	It does not supports XML format.

Abbildung 1 - <https://www.geeksforgeeks.org/difference-between-mysql-and-sqlite/>

### 1.1.1 Book List Challenge

The screenshot shows the Khan Academy interface for the 'Challenge: Book list database'. The main content area displays a SQL editor with the following code:

```
1 CREATE TABLE booklist
2 (id INTEGER PRIMARY KEY, name TEXT, rating INTEGER);
3 INSERT INTO booklist VALUES ("1", "Harry Potter", "5"), ("2",
4 "Saeculum", "7"), ("3", "Buch", "5");
```

To the right of the editor, the 'DATABASE SCHEMA' is shown:

booklist	3 rows
id (PK)	INTEGER
name	TEXT
rating	INTEGER

At the bottom right, a congratulatory message states: 'Congratulations! You earned 1050 points!' with a 'Spin-off' button.

Abbildung 2 - Book List Challenge

## 1.1.2 Box Office

The screenshot shows the Khan Academy interface for the 'Challenge: Box office hits database'. The left sidebar lists the course progress, with 'Challenge: Box office hits database' selected. The main content area displays the challenge title and instructions: 'Filter recent movies. Now, add a second query after the first, that retrieves only the movies that were released in the year 2000 or later, not before. Sort the results so that the earlier movies are listed first. You should have 2 SELECT statements after this step.'

The SQL code editor shows the following code:

```
1 CREATE TABLE movies (id INTEGER PRIMARY KEY, name TEXT, release_year INTEGER);
2 INSERT INTO movies VALUES (1, "Avatar", 2009);
3 INSERT INTO movies VALUES (2, "Titanic", 1997);
4 INSERT INTO movies VALUES (3, "Star Wars: Episode IV - A New Hope", 1977);
5 INSERT INTO movies VALUES (4, "Shrek 2", 2004);
6 INSERT INTO movies VALUES (5, "The Lion King", 1994);
7 INSERT INTO movies VALUES (6, "Disney's Up", 2009);
8
9 SELECT * FROM movies;
10 SELECT * FROM movies WHERE release_year >= 2000 ORDER BY release_year;
11
12
```

The query results are displayed in two tables:

id	name	release_year
1	Avatar	2009
2	Titanic	1997
3	Star Wars: Episode IV - A New Hope	1977
4	Shrek 2	2004
5	The Lion King	1994
6	Disney's Up	2009

The second table shows the results of the second query, filtered by release year:

id	name	release_year
4	Shrek 2	2004
1	Avatar	2009
6	Disney's Up	2009

A congratulatory message states: 'Congratulations! You earned 1500 points!'. The challenge progress is shown as 'Step 2/2' with a 'Spin-off' button.

Abbildung 3 - Box Office

## 1.1.3 ToDo - List

The screenshot shows the Khan Academy interface for the 'Challenge: TODO list database stats'. The left sidebar lists the course progress, with 'Challenge: TODO list database stats' selected. The main content area displays the challenge title and instructions: 'Step 2. Select the SUM of minutes it will take to do all of the items on your TODO list. You should only have one SELECT statement.'

The SQL code editor shows the following code:

```
1 CREATE TABLE todo_list (id INTEGER PRIMARY KEY, item TEXT, minutes INTEGER);
2 INSERT INTO todo_list VALUES (1, "Wash the dishes", 15);
3 INSERT INTO todo_list VALUES (2, "vacuuming", 20);
4 INSERT INTO todo_list VALUES (3, "Learn some stuff on KA", 30);
5
6 INSERT INTO todo_list VALUES (4, "print homework", 5);
7
8 SELECT SUM (minutes) FROM todo_list;
9
```

The query results are displayed in a table:

SUM (minutes)
70

A congratulatory message states: 'Congratulations! You earned 2100 points!'. The challenge progress is shown as 'Step 2/2' with a 'Spin-off' button.

Abbildung 4 - ToDo-List

## 1.1.4 Projekt: Design a store database

**Spin-off of "Project: Design a store database"**

Create your own store! Your store should sell one type of things, like clothing or bikes, whatever you want your store to specialize in. You should have a table for all the items in your store, and at least 5 columns for the kind of data you think you'd need to store. You should sell at least 15 items, and use select statements to order your items by price and show at least one statistic about the items.

```
1 CREATE TABLE Store_Items (id INTEGER PRIMARY KEY, name TEXT,
2   lagerplatz TEXT, gewicht_kg NUMERIC, preisEUR NUMERIC);
3 INSERT INTO Store_Items VALUES
4   ("1", "Kugelschreiber", "R4", 0.2, 1.99),
5   ("2", "Block", "R9", 0.5, 1.50),
6   ("3", "FarbstifteSet", "R2", 0.8, 1.39),
7   ("4", "Papierliniert", "R5", 0.9, 0.99),
8   ("5", "Papierkarliert", "R5", 0.9, 1.50),
9   ("6", "PapierBlank", "R5", 0.9, 19.99),
10  ("7", "PapierDibunt", "R5", 0.9, 14.59),
11  ("8", "Bleistiftkatl", "R3", 0.1, 12.59),
12  ("9", "Kreide", "R9", 0.8, 1.29),
13  ("10", "Klarsichtfolien", "R7", 0.5, 1.19),
14  ("11", "OrdnerA4", "R7", 1.0, 1.49),
15  ("12", "Schnellhefter", "R8", 0.8, 14.28),
16  ("13", "FeinlinierSet", "R2", 0.9, 13.58),
17  ("14", "Radiergummi", "R1", 0.5, 1.79),
18  ("15", "Fühlfeder", "R4", 0.8, 3.29);
19 SELECT * FROM Store_Items
20 ORDER BY preisEUR DESC;
21
22 SELECT AVG(preisEUR) FROM Store_Items;
23 SELECT MIN(preisEUR) FROM Store_Items;
24
```

**DATABASE SCHEMA**

Store_Items	15 rows
id (PK)	INTEGER
name	TEXT
lagerplatz	TEXT
gewicht_kg	NUMERIC
preisEUR	NUMERIC

**QUERY RESULTS**

id	name	lagerplatz	gewicht_kg	preisEUR
6	PapierBlank	R5	0.9	19.99
7	PapierDibunt	R5	0.9	14.5
12	Schnellhefter	R8	0.8	14.28

Buttons: Request help, Spin-off, Save

Abbildung 5 - Projekt: Design a store database - Screensh.1

**Spin-off of "Project: Design a store database"**

Create your own store! Your store should sell one type of things, like clothing or bikes, whatever you want your store to specialize in. You should have a table for all the items in your store, and at least 5 columns for the kind of data you think you'd need to store. You should sell at least 15 items, and use select statements to order your items by price and show at least one statistic about the items.

```
1 CREATE TABLE Store_Items (id INTEGER PRIMARY KEY, name TEXT,
2   lagerplatz TEXT, gewicht_kg NUMERIC, preisEUR NUMERIC);
3 INSERT INTO Store_Items VALUES
4   ("1", "Kugelschreiber", "R4", 0.2, 1.99),
5   ("2", "Block", "R9", 0.5, 1.50),
6   ("3", "FarbstifteSet", "R2", 0.8, 1.39),
7   ("4", "Papierliniert", "R5", 0.9, 0.99),
8   ("5", "Papierkarliert", "R5", 0.9, 1.50),
9   ("6", "PapierBlank", "R5", 0.9, 19.99),
10  ("7", "PapierDibunt", "R5", 0.9, 14.59),
11  ("8", "Bleistiftkatl", "R3", 0.1, 12.59),
12  ("9", "Kreide", "R9", 0.8, 1.29),
13  ("10", "Klarsichtfolien", "R7", 0.5, 1.19),
14  ("11", "OrdnerA4", "R7", 1.0, 1.49),
15  ("12", "Schnellhefter", "R8", 0.8, 14.28),
16  ("13", "FeinlinierSet", "R2", 0.9, 13.58),
17  ("14", "Radiergummi", "R1", 0.5, 1.79),
18  ("15", "Fühlfeder", "R4", 0.8, 3.29);
19 SELECT * FROM Store_Items
20 ORDER BY preisEUR DESC;
21
22 SELECT AVG(preisEUR) FROM Store_Items;
23 SELECT MIN(preisEUR) FROM Store_Items;
24
```

2	Block	R9	0.5	1.5
5	Papierkarliert	R5	0.9	1.5
11	OrdnerA4	R7	1	1.49
3	FarbstifteSet	R2	0.8	1.39
9	Kreide	R9	0.8	1.29
10	Klarsichtfolien	R7	0.5	1.19
4	Papierliniert	R5	0.9	0.99

**AVG(preisEUR)**  
6.09066666666667

**MIN(preisEUR)**  
0.99

Buttons: Request help, Spin-off, Save

Abbildung 6 - Projekt: Design a store database - Screensh.2

## 1.2 Komplexe Abfragen mit AND/ OR

### 1.2.1 Karaoke song selector

Computing > Computer programming > Intro to SQL: Querying and managing data > More advanced SQL queries

Challenge: Karaoke song selector

Step 3

People get picky at the end of the night. Add another SELECT that uses AND to show the titles of songs that are 'epic', and released after 1990, and less than 4 minutes long. Note that the duration column is measured in seconds.

Hint What's this?

SELECT ... WHERE ... AND ...;

```
9 INSERT INTO songs (title, artist, mood, duration, released)
10 VALUES ("Bohemian Rhapsody", "Queen", "epic", 60, 1975);
11 INSERT INTO songs (title, artist, mood, duration, released)
12 VALUES ("Let it go", "Idina Menzel", "epic", 227, 2013);
13 INSERT INTO songs (title, artist, mood, duration, released)
14 VALUES ("I will survive", "Gloria Gaynor", "epic", 198, 1978);
15 INSERT INTO songs (title, artist, mood, duration, released)
16 VALUES ("Hust and Shout", "The Beatles", "happy", 152, 1963);
17 INSERT INTO songs (title, artist, mood, duration, released)
18 VALUES ("La Bamba", "Ritchie Valens", "happy", 166, 1958);
19 INSERT INTO songs (title, artist, mood, duration, released)
20 VALUES ("I will always love you", "Whitney Houston", "epic",
21 273, 1992);
22 INSERT INTO songs (title, artist, mood, duration, released)
23 VALUES ("Sweet Caroline", "Neil Diamond", "happy", 201, 1969);
24 INSERT INTO songs (title, artist, mood, duration, released)
25 VALUES ("Call me maybe", "Carly Rae Jepsen", "happy", 193, 2011
26 );
27 SELECT title FROM songs
28 WHERE mood IN ("epic") OR released >
29 1990;
30 SELECT title FROM songs WHERE mood IN ("epic") AND released >
31 1990 AND duration < 240;
```

Undo Start over

Step 3/3 Spin-off

12:43 16.09.2022

Abbildung 7 - Karaoke song selector

### 1.2.2 Playlist maker

Computing > Computer programming > Intro to SQL: Querying and managing data > More advanced SQL queries

Challenge: Playlist maker

Step 3

To finish creating the 'Pop' playlist, add another query that will select the title of all the songs from the 'Pop' artists. It should use IN on a nested subquery that's based on your previous query.

Hint What's this?

SELECT title FROM ...;

```
40 VALUES ("Rihanna", "Stay");
41 INSERT INTO songs (artist, title)
42 VALUES ("Celine Dion", "My heart will go on");
43 INSERT INTO songs (artist, title)
44 VALUES ("Celine Dion", "A new day has come");
45 INSERT INTO songs (artist, title)
46 VALUES ("Shania Twain", "Party for two");
47 INSERT INTO songs (artist, title)
48 VALUES ("Gloria Estefan", "Conga");
49 INSERT INTO songs (artist, title)
50 VALUES ("Led Zeppelin", "Stairway to heaven");
51 INSERT INTO songs (artist, title)
52 VALUES ("ABBA", "Mamma mia");
53 INSERT INTO songs (artist, title)
54 VALUES ("Queen", "Bicycle Race");
55 INSERT INTO songs (artist, title)
56 VALUES ("Queen", "Bohemian Rhapsody");
57 INSERT INTO songs (artist, title)
58 VALUES ("Guns N' Roses", "Don't cry");
59
60 SELECT title FROM songs WHERE artist = "Queen";
61 SELECT name FROM artists WHERE genre = "Pop";
62 SELECT title FROM songs WHERE artist IN (SELECT name FROM artists
63 WHERE genre = "Pop");
```

Undo Start over

Step 3/3 Spin-off

12:50 16.09.2022

Abbildung 8 - Playlist maker



## 1.2.3 The wordiest Author

Computing > Computer programming > Intro to SQL: Querying and managing data > More advanced SQL queries

### Challenge: The wordiest author

Step 2

Now select all the authors that write more than an average of 150,000 words per book. Your results table should include the 'author' and average words as an 'avg\_words' column.

Hint: What's this? [Report a problem](#)

SELECT ... FROM ... GROUP BY ... HAVING ...;

197651);

```
21 INSERT INTO books (author, title, words)
22 VALUES ("Stephenie Meyer", "Twilight", 118501);
23 INSERT INTO books (author, title, words)
24 VALUES ("Stephenie Meyer", "New Moon", 132807);
25 INSERT INTO books (author, title, words)
26 VALUES ("Stephenie Meyer", "Eclipse", 147930);
27 INSERT INTO books (author, title, words)
28 VALUES ("Stephenie Meyer", "Breaking Dawn", 192196);
29
30 INSERT INTO books (author, title, words)
31 VALUES ("J.R.R. Tolkien", "The Hobbit", 95022);
32 INSERT INTO books (author, title, words)
33 VALUES ("J.R.R. Tolkien", "Fellowship of the Ring", 177227);
34 INSERT INTO books (author, title, words)
35 VALUES ("J.R.R. Tolkien", "Two Towers", 143436);
36 INSERT INTO books (author, title, words)
37 VALUES ("J.R.R. Tolkien", "Return of the King", 134462);
38
39 SELECT author, SUM(words) AS total_words FROM books GROUP BY author
40 HAVING total_words > 1000000;
41
42 SELECT author, AVG(words) AS avg_words FROM books GROUP BY author
43 HAVING avg_words > 150000;
```

DATABASE SCHEMA

books	15 rows
id (PK)	INTEGER
author	TEXT
title	TEXT
words	INTEGER

QUERY RESULTS

author	total_words
J.K. Rowling	1086594

author

J.K. Rowling

Congratulations! You earned 2100 points!

Step 2/2 Spin-off

Abbildung 9 - The wordiest author

## 1.2.4 Gradebook

Computing > Computer programming > Intro to SQL: Querying and managing data > More advanced SQL queries

### Challenge: Gradebook

Step 2

Now, this step is a little tricky. The goal is a table that shows how many students have earned which letter grade. You can output the letter grade by using CASE with the number\_grade column, outputting 'A' for grades > 90, 'B' for grades > 80, 'C' for grades > 70, and 'F' otherwise. Then you can use COUNT with GROUP BY to show the number of students with each of those grades.

Hint: What's this? [Report a problem](#)

SELECT ... CASE ... FROM ... GROUP BY ...;

```
8 INSERT INTO student_grades (name, number_grade, fraction_completed)
9 VALUES ("Winston", 90, 0.885);
10 INSERT INTO student_grades (name, number_grade, fraction_completed)
11 VALUES ("Winnefer", 95, 0.901);
12 INSERT INTO student_grades (name, number_grade, fraction_completed)
13 VALUES ("Winsteen", 85, 0.906);
14 INSERT INTO student_grades (name, number_grade, fraction_completed)
15 VALUES ("Wincifer", 66, 0.7854);
16 INSERT INTO student_grades (name, number_grade, fraction_completed)
17 VALUES ("Winsten", 76, 0.5013);
18 INSERT INTO student_grades (name, number_grade, fraction_completed)
19 VALUES ("Winstonia", 82, 0.9043);
20
21 SELECT name, number_grade, ROUND(fraction_completed * 100)
22 AS percent_completed FROM student_grades;
23
24 SELECT COUNT(*),
25 CASE
26 WHEN number_grade > 90 THEN "A"
27 WHEN number_grade > 80 THEN "B"
28 WHEN number_grade > 70 THEN "C"
29 ELSE "F"
30 END AS "grade"
31 FROM student_grades
32 GROUP BY grade;
```

name number\_grade percent\_completed

Winston	90	81
Winnefer	95	90
Winsteen	85	91
Wincifer	66	71
Winsten	76	50
Winstonia	82	90

COUNT(\*)

grade
A
B
C
F

Step 2/2 Spin-off

Abbildung 10 - Gradebook

# Abbildungsverzeichnis

Abbildung 1 - <a href="https://www.geeksforgeeks.org/difference-between-mysql-and-sqlite/">https://www.geeksforgeeks.org/difference-between-mysql-and-sqlite/</a>	3
Abbildung 2 - Book List Challenge	3
Abbildung 3 - Box Office	4
Abbildung 4 - ToDo-List	4
Abbildung 5 - Projekt: Design a store database - Screensh.1	5
Abbildung 6 - Projekt: Design a store database - Screensh.2	5
Abbildung 7 - Karaoke song selector	6
Abbildung 8 - Playlist maker	6
Abbildung 9 - The wordiest author	7
Abbildung 10 - Gradebook	7



## Tabellenverzeichnis

*Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.*

## Literaturverzeichnis

Im aktuellen Dokument sind keine Quellen vorhanden.

## Anhang