

[Open in app](#)[Get started](#)

Published in Towards Data Science



Vijini Mallawaarachchi

[Follow](#)

Sep 15, 2017 · 7 min read · · Listen

Save



# SQL Recap for Interviews

DIY with Examples and Sample Code

**SQL or Structured Query Language** is a language designed to manage data in a **Relational Database Management System (RDBMS)**. In this article, I will be walking you through the commonly used SQL commands which every programmer should know about. This article is perfect if you need to brush up your knowledge on SQL for an interview. All you have to do is try out the given examples and refresh what you have learnt quite a while ago in your database systems class. 😊

**NOTE:** Some database systems require a semicolon to be inserted at the end of each statement. The semicolon is the standard way to indicate the end of each SQL statement. I will be using **MySQL** for the examples, which requires a semicolon at the end of each statement.

## Setting up the sample database

A sample database will be used for the demonstrations of each command. You can find the two SQL scripts [DLL.sql](#) and [InsertStatements.sql](#) by clicking the links.

After saving the files to your local machine, open your terminal and log in to MySQL console using the following command. (I am assuming that you have installed MySQL already.)

```
mysql -u root -p
```



[Open in app](#)[Get started](#)

```
CREATE DATABASE university;  
USE university;  
SOURCE <path_of_DLL.sql_file>;  
SOURCE <path_of_InsertStatements.sql_file>;
```



[Open in app](#)[Get started](#)

```
vijinimallawaarachchi — mysql -u root -p — 81x53
[Vijinis-MacBook-Pro:~ vijinimallawaarachchi$ mysql -u root -p
[Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 454
Server version: 5.7.19 MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

[mysql> CREATE DATABASE university;
Query OK, 1 row affected (0.00 sec)

[mysql> USE university;
Database changed
[mysql> SOURCE /Users/vijinimallawaarachchi/Desktop/DLL.sql;
Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

[mysql> SOURCE /Users/vijinimallawaarachchi/Desktop/InsertStatements.sql;
Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.00 sec)
```

Figure 1: Executed queries to setup the database

Now let's start refreshing the SQL commands we have learned before



[Open in app](#)[Get started](#)

## 1. See currently available databases

```
SHOW DATABASES;
```

## 2. Create a new database

```
CREATE DATABASE <database_name>;
```

## 3. Select a database to use

```
USE <database_name>;
```

## 4. Import SQL commands from .sql file

```
SOURCE <path_of_.sql_file>;
```

## 5. Delete a database

```
DROP DATABASE <database_name>;
```

## Table Related Commands

## 6. See currently available tables in a database



[Open in app](#)[Get started](#)

```
vijinimallawaarachchi — mysql -u root -p — 81x22
Query OK, 1 row affected (0.00 sec)

[mysql> SHOW TABLES;
+-----+
| Tables_in_university |
+-----+
| advisor
| classroom
| course
| department
| instructor
| prereq
| section
| student
| takes
| teaches
| time_slot
+-----+
11 rows in set (0.00 sec)

mysql> ]
```

Figure 2: Tables in university database

## 7. Create a new table

```
CREATE TABLE <table_name1> (
    <col_name1> <col_type1>,
    <col_name2> <col_type2>,
    <col_name3> <col_type3>
    PRIMARY KEY (<col_name1>),
    FOREIGN KEY (<col_name2>) REFERENCES <table_name2>(<col_name2>)
);
```

### *Integrity Constraints in CREATE TABLE*

You may need to set constraints for certain columns of a table. Following constraints can be imposed while creating a table.

- NOT NULL



[Open in app](#)[Get started](#)

You can include more than one primary key which will create a **composite** or **concatenated primary key**.

### Example

Create the table instructor.

```
CREATE TABLE instructor (
    ID CHAR(5),
    name VARCHAR(20) NOT NULL,
    dept_name VARCHAR(20),
    salary NUMERIC(8,2),
    PRIMARY KEY (ID),
    FOREIGN KEY (dept_name) REFERENCES department(dept_name))
```

## 8. Describe columns of a table

You can view the columns of a table with details such as the type and key, using the command given below. Figure 3 shows a few examples.

```
DESCRIBE <table_name>;
```



[Open in app](#)[Get started](#)

```
vijinimallawaarachchi — mysql -u root -p — 81x53

[mysql]> DESCRIBE advisor;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| s_ID  | varchar(5) | NO   | PRI  | NULL    |       |
| i_ID  | varchar(5) | YES  | MUL  | NULL    |       |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

[mysql]> DESCRIBE classroom;
+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| building   | varchar(15) | NO   | PRI  | NULL    |       |
| room_number | varchar(7)  | NO   | PRI  | NULL    |       |
| capacity   | decimal(4,0) | YES  |      | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

[mysql]> DESCRIBE course;
+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| course_id  | varchar(8) | NO   | PRI  | NULL    |       |
| title      | varchar(50) | YES  |      | NULL    |       |
| dept_name  | varchar(20) | YES  | MUL  | NULL    |       |
| credits    | decimal(2,0) | YES  |      | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

[mysql]> DESCRIBE department;
+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| dept_name  | varchar(20) | NO   | PRI  | NULL    |       |
| building   | varchar(15) | YES  |      | NULL    |       |
| budget     | decimal(12,2) | YES  |      | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

[mysql]> DESCRIBE instructor;
+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| ID         | varchar(5) | NO   | PRI  | NULL    |       |
| name       | varchar(20) | NO   |      | NULL    |       |
| dept_name  | varchar(20) | YES  | MUL  | NULL    |       |
| salary     | decimal(8,2) | YES  |      | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

Figure 3: Details of table columns



[Open in app](#)[Get started](#)

If you are inserting values to all the columns of a table, then there is no need to specify the column names at the beginning.

```
INSERT INTO <table_name>
    VALUES (<value1>, <value2>, <value3>, ...);
```

## 10. Update a table

```
UPDATE <table_name>
    SET <col_name1> = <value1>, <col_name2> = <value2>, ...
    WHERE <condition>;
```

## 11. Delete all contents of a table

```
DELETE FROM <table_name>;
```

## 12. Delete a table

```
DROP TABLE <table_name>;
```

## Querying Related Commands

### 13. SELECT

The **SELECT** statement is used to select data from a particular table.



[Open in app](#)[Get started](#)

You can select all the contents of a table as follows. Refer Figure 4 as well.

```
SELECT * FROM <table_name>;
```

The screenshot shows a terminal window titled "vijinimallawaarachchi — mysql -u root -p — 81x36". It displays two SQL queries: one for the "department" table and one for the "course" table. Both queries use the asterisk (\*) wildcard to select all columns for every row.

```
[mysql]> SELECT * FROM department;
+-----+-----+-----+
| dept_name | building | budget |
+-----+-----+-----+
| Biology    | Watson   | 90000.00 |
| Comp. Sci. | Taylor   | 100000.00 |
| Elec. Eng. | Taylor   | 85000.00 |
| Finance    | Painter  | 120000.00 |
| History    | Painter  | 50000.00 |
| Music      | Packard  | 80000.00 |
| Physics    | Watson   | 70000.00 |
+-----+-----+-----+
7 rows in set (0.00 sec)

[mysql]> SELECT * FROM course;
+-----+-----+-----+-----+
| course_id | title           | dept_name | credits |
+-----+-----+-----+-----+
| BIO-101   | Intro. to Biology | Biology   | 4       |
| BIO-301   | Genetics         | Biology   | 4       |
| BIO-399   | Computational Biology | Biology | 3       |
| CS-101    | Intro. to Computer Science | Comp. Sci. | 4       |
| CS-190    | Game Design       | Comp. Sci. | 4       |
| CS-315    | Robotics          | Comp. Sci. | 3       |
| CS-319    | Image Processing  | Comp. Sci. | 3       |
| CS-347    | Database System Concepts | Comp. Sci. | 3       |
| EE-181    | Intro. to Digital Systems | Elec. Eng. | 3       |
| FIN-201   | Investment Banking | Finance   | 3       |
| HIS-351   | World History     | History   | 3       |
| MU-199    | Music Video Production | Music   | 3       |
| PHY-101   | Physical Principles | Physics   | 4       |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

mysql>
```

Figure 4: Select all from **department** and **course** tables

## 14. SELECT DISTINCT

A column of a table can often contain duplicate values. **SELECT DISTINCT** allows you to retrieve the distinct values.



[Open in app](#)[Get started](#)

```
vijinimallawaarachchi — mysql -u root -p — 81x16
[mysql]> SELECT DISTINCT dept_name FROM course;
+-----+
| dept_name |
+-----+
| Biology   |
| Comp. Sci. |
| Elec. Eng. |
| Finance   |
| History   |
| Music     |
| Physics   |
+-----+
7 rows in set (0.00 sec)

mysql> ]
```

Figure 5: **SELECT DISTINCT** example

## 15. WHERE

You can use **WHERE** keyword in a **SELECT** statement in order to include conditions for your data.

```
SELECT <col_name1>, <col_name2>, ...
FROM <table_name>
WHERE <condition>;
```

You can include conditions consisting of,

- Comparison of **text**
- Comparison of **numbers**
- Logical operations including **AND**, **OR** and **NOT**.

### Example

Go through the following example statements. Note how conditions have been included with the **WHERE** clause.





Open in app

Get started

```
vijinimallawaarachchi — mysql -u root -p — 81x35

[mysql]> SELECT * FROM course WHERE dept_name='Comp. Sci.';
+-----+-----+-----+
| course_id | title | dept_name | credits |
+-----+-----+-----+
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |
| CS-319 | Image Processing | Comp. Sci. | 3 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 |
+-----+-----+-----+
5 rows in set (0.00 sec)

[mysql]> SELECT * FROM course WHERE credits>3;
+-----+-----+-----+
| course_id | title | dept_name | credits |
+-----+-----+-----+
| BIO-101 | Intro. to Biology | Biology | 4 |
| BIO-301 | Genetics | Biology | 4 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| PHY-101 | Physical Principles | Physics | 4 |
+-----+-----+-----+
5 rows in set (0.00 sec)

[mysql]> SELECT * FROM course WHERE dept_name='Comp. Sci.' AND credits>3;
+-----+-----+-----+
| course_id | title | dept_name | credits |
+-----+-----+-----+
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Figure 6: Usage of WHERE in SELECT

## 16. GROUP BY

The **GROUP BY** statement is often used with **aggregate functions** such as **COUNT**, **MAX**, **MIN**, **SUM** and **AVG** to group the result-set.

```
SELECT <col_name1>, <col_name2>, ...
FROM <table_name>
GROUP BY <col_namex>;
```



[Open in app](#)[Get started](#)

```
SELECT COUNT(course_id), dept_name
FROM course
GROUP BY dept_name;
```

```
vijinimallawaarachchi — mysql -u root -p — 81x16
[mysql]> SELECT COUNT(course_id), dept_name FROM course GROUP BY dept_name;
+-----+-----+
| COUNT(course_id) | dept_name |
+-----+-----+
| 3 | Biology |
| 5 | Comp. Sci. |
| 1 | Elec. Eng. |
| 1 | Finance |
| 1 | History |
| 1 | Music |
| 1 | Physics |
+-----+-----+
7 rows in set (0.00 sec)

mysql> ]
```

Figure 7: Number of courses for each department

## 17. HAVING

The **HAVING** clause was introduced to SQL because the **WHERE** keyword could not be used to compare values of **aggregate functions**.

```
SELECT <col_name1>, <col_name2>, ...
  FROM <table_name>
  GROUP BY <column_namex>
 HAVING <condition>
```

### Example

List the number of courses for each department which offers more than one course.

```
SELECT COUNT(course_id), dept_name
```



[Open in app](#)[Get started](#)

```
vijinimallawaarachchi — mysql -u root -p — 81x12
[mysql]> SELECT COUNT(course_id), dept_name FROM course GROUP BY dept_name HAVING COUNT(course_id)>1;
+-----+-----+
| COUNT(course_id) | dept_name |
+-----+-----+
|            3 | Biology   |
|            5 | Comp. Sci. |
+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Figure 8: Departments offering more than one course

## 18. ORDER BY

**ORDER BY** is used to sort a result set in ascending or descending order. **ORDER BY** will sort in ascending order if you do not specify ASC or DESC.

```
SELECT <col_name1>, <col_name2>, ...
FROM <table_name>
ORDER BY <col_name1>, <col_name2>, ... ASC|DESC;
```

### Example

List the courses in ascending and descending order of number of credits.

```
SELECT * FROM course ORDER BY credits;
SELECT * FROM course ORDER BY credits DESC;
```





Open in app

Get started

```
vijinimallawaarachchi — mysql -u root -p — 81x42

[mysql]> SELECT * FROM course ORDER BY credits;
+-----+-----+-----+-----+
| course_id | title | dept_name | credits |
+-----+-----+-----+-----+
| BIO-399 | Computational Biology | Biology | 3 |
| CS-315 | Robotics | Comp. Sci. | 3 |
| CS-319 | Image Processing | Comp. Sci. | 3 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 |
| FIN-201 | Investment Banking | Finance | 3 |
| HIS-351 | World History | History | 3 |
| MU-199 | Music Video Production | Music | 3 |
| BIO-101 | Intro. to Biology | Biology | 4 |
| BIO-301 | Genetics | Biology | 4 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| PHY-101 | Physical Principles | Physics | 4 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

[mysql]> SELECT * FROM course ORDER BY credits DESC;
+-----+-----+-----+-----+
| course_id | title | dept_name | credits |
+-----+-----+-----+-----+
| BIO-101 | Intro. to Biology | Biology | 4 |
| BIO-301 | Genetics | Biology | 4 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| PHY-101 | Physical Principles | Physics | 4 |
| BIO-399 | Computational Biology | Biology | 3 |
| CS-315 | Robotics | Comp. Sci. | 3 |
| CS-319 | Image Processing | Comp. Sci. | 3 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 |
| FIN-201 | Investment Banking | Finance | 3 |
| HIS-351 | World History | History | 3 |
| MU-199 | Music Video Production | Music | 3 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

mysql>
```

Figure 9: Courses sorted according to credits

## 19. BETWEEN

BETWEEN clause is used to select data within a given range. The values can be numbers, text or even dates.

`SELECT <col_name1>, <col_name2>, ...`



[Open in app](#)[Get started](#)

List the instructors who get a salary in between 50 000 and 100 000.

```
SELECT * FROM instructor
WHERE salary BETWEEN 50000 AND 100000;
```

| ID    | name       | dept_name  | salary   |
|-------|------------|------------|----------|
| 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 12121 | Wu         | Finance    | 90000.00 |
| 22222 | Einstein   | Physics    | 95000.00 |
| 32343 | El Said    | History    | 60000.00 |
| 33456 | Gold       | Physics    | 87000.00 |
| 45565 | Katz       | Comp. Sci. | 75000.00 |
| 58583 | Califieri  | History    | 62000.00 |
| 76543 | Singh      | Finance    | 80000.00 |
| 76766 | Crick      | Biology    | 72000.00 |
| 83821 | Brandt     | Comp. Sci. | 92000.00 |
| 98345 | Kim        | Elec. Eng. | 80000.00 |

11 rows in set (0.00 sec)

mysql>

Figure 10: Instructors who get a salary in between 50 000 and 100 000

## 20. LIKE

The **LIKE** operator is used in a **WHERE** clause to search for a **specified pattern in text**.

There are two wildcard operators used with LIKE.

- % (Zero, one, or multiple characters)
- \_ (A single character)

```
SELECT <col_name1>, <col_name2>, ...
FROM <table_name>
```





Open in app

Get started

List the courses where the course name contains “to” and the courses where the course\_id starts with “CS-”.

```
SELECT * FROM course WHERE title LIKE '%to%';
SELECT * FROM course WHERE course_id LIKE 'CS-__';
```

```
vijinimallawaarachchi — mysql -u root -p — 81x25
[mysql]> SELECT * FROM course WHERE title LIKE '%to%';
+-----+-----+-----+
| course_id | title | dept_name | credits |
+-----+-----+-----+
| BIO-101 | Intro. to Biology | Biology | 4 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 |
| HIS-351 | World History | History | 3 |
+-----+-----+-----+
4 rows in set (0.00 sec)

[mysql]> SELECT * FROM course WHERE course_id LIKE 'CS-__';
+-----+-----+-----+
| course_id | title | dept_name | credits |
+-----+-----+-----+
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |
| CS-319 | Image Processing | Comp. Sci. | 3 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

Figure 11: Use of wildcard operators with LIKE

## 21. IN

Using IN clause, you can allow multiple values within a WHERE clause.

```
SELECT <col_name1>, <col_name2>, ...
FROM <table_name>
WHERE <col_namen> IN (<value1>, <value2>, ...);
```



[Open in app](#)[Get started](#)

```
SELECT * FROM student
WHERE dept_name IN ('Comp. Sci.', 'Physics', 'Elec. Eng.');
```

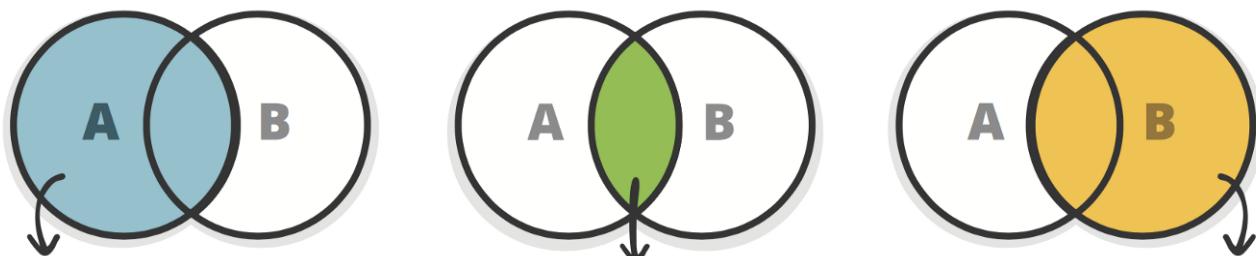
```
vijinimallawaarachchi — mysql -u root -p — 81x19
mysql> SELECT * FROM student WHERE dept_name IN ('Comp. Sci.', 'Physics', 'Elec. Eng.');
+----+-----+-----+-----+
| ID | name | dept_name | tot_cred |
+----+-----+-----+-----+
| 00128 | Zhang | Comp. Sci. | 102 |
| 12345 | Shankar | Comp. Sci. | 32 |
| 44553 | Peltier | Physics | 56 |
| 45678 | Levy | Physics | 46 |
| 54321 | Williams | Comp. Sci. | 54 |
| 70557 | Snow | Physics | 0 |
| 76543 | Brown | Comp. Sci. | 58 |
| 76653 | Aoi | Elec. Eng. | 60 |
| 98765 | Bourikas | Elec. Eng. | 98 |
+----+-----+-----+-----+
9 rows in set (0.00 sec)

mysql> █
```

Figure 12: Students in the departments of Comp. Sci., Physics, and Elec. Eng

## 22. JOIN

JOIN is used to combine values of two or more tables based on common attributes within them. Figure 13 given below depicts the different types of SQL joins. Note the difference between **LEFT OUTER JOIN** and **RIGHT OUTER JOIN**.



**LEFT OUTER JOIN** - all rows from table A, even if they do not exist in table B

**INNER JOIN** - fetch the results that exist in both tables

**RIGHT OUTER JOIN** - all rows from table B, even if they do not exist in table A

Figure 13: Types of joins (Source: [http://files.zereturnaround.com/pdf/zt\\_sql\\_cheat\\_sheet.pdf](http://files.zereturnaround.com/pdf/zt_sql_cheat_sheet.pdf))





Open in app

Get started

```
JOIN <table_name2>
ON <table_name1.col_namex> = <table2.col_namex>;
```

## Example 1

Select all the courses with relevant department details.

```
SELECT * FROM course
JOIN department
ON course.dept_name=department.dept_name;
```

```
vijinimallawaarachchi — mysql -u root -p — 110x22
mysql> SELECT * FROM course JOIN department ON course.dept_name=department.dept_name;
+-----+-----+-----+-----+-----+-----+
| course_id | title | dept_name | credits | dept_name | building | budget |
+-----+-----+-----+-----+-----+-----+
| BIO-101 | Intro. to Biology | Biology | 4 | Biology | Watson | 90000.00 |
| BIO-301 | Genetics | Biology | 4 | Biology | Watson | 90000.00 |
| BIO-399 | Computational Biology | Biology | 3 | Biology | Watson | 90000.00 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 | Comp. Sci. | Taylor | 100000.00 |
| CS-190 | Game Design | Comp. Sci. | 4 | Comp. Sci. | Taylor | 100000.00 |
| CS-315 | Robotics | Comp. Sci. | 3 | Comp. Sci. | Taylor | 100000.00 |
| CS-319 | Image Processing | Comp. Sci. | 3 | Comp. Sci. | Taylor | 100000.00 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 | Comp. Sci. | Taylor | 100000.00 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 | Elec. Eng. | Taylor | 85000.00 |
| FIN-201 | Investment Banking | Finance | 3 | Finance | Painter | 120000.00 |
| HIS-351 | World History | History | 3 | History | Painter | 50000.00 |
| MU-199 | Music Video Production | Music | 3 | Music | Packard | 80000.00 |
| PHY-101 | Physical Principles | Physics | 4 | Physics | Watson | 70000.00 |
+-----+-----+-----+-----+-----+-----+
13 rows in set (0.00 sec)

mysql> ■
```

Figure 14: All the courses with department details

## Example 2

Select all the prerequisite courses with their course details.

```
SELECT prereq.course_id, title, dept_name, credits, prereq_id
FROM prereq
LEFT OUTER JOIN course
ON prereq.course_id=course.course_id;
```



[Open in app](#)[Get started](#)

```
vijinimallawaarachchi — mysql -u root -p — 110x17
mysql> SELECT prereq.course_id, title, dept_name, credits, prereq_id FROM prereq LEFT OUTER JOIN course ON prereq.course_id=course.course_id;
+-----+-----+-----+-----+
| course_id | title | dept_name | credits | prereq_id |
+-----+-----+-----+-----+
| BIO-301 | Genetics | Biology | 4 | BIO-101 |
| BIO-399 | Computational Biology | Biology | 3 | BIO-101 |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-101 |
| CS-315 | Robotics | Comp. Sci. | 3 | CS-101 |
| CS-319 | Image Processing | Comp. Sci. | 3 | CS-101 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 | CS-101 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 | PHY-101 |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> 
```

Figure 15: All the **prerequisite** courses with their **course** details

### Example 3

Select all the courses with their course details, regardless of whether or not they have prerequisites.

```
SELECT course.course_id, title, dept_name, credits, prereq_id
FROM prereq
RIGHT OUTER JOIN course
ON prereq.course_id=course.course_id;
```

```
vijinimallawaarachchi — mysql -u root -p — 110x23
mysql> SELECT course.course_id, title, dept_name, credits, prereq_id FROM prereq RIGHT OUTER JOIN course ON prereq.course_id=course.course_id;
+-----+-----+-----+-----+
| course_id | title | dept_name | credits | prereq_id |
+-----+-----+-----+-----+
| BIO-101 | Intro. to Biology | Biology | 4 | NULL |
| BIO-301 | Genetics | Biology | 4 | BIO-101 |
| BIO-399 | Computational Biology | Biology | 3 | BIO-101 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 | NULL |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-101 |
| CS-315 | Robotics | Comp. Sci. | 3 | CS-101 |
| CS-319 | Image Processing | Comp. Sci. | 3 | CS-101 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 | CS-101 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 | PHY-101 |
| FIN-201 | Investment Banking | Finance | 3 | NULL |
| HIS-351 | World History | History | 3 | NULL |
| MU-199 | Music Video Production | Music | 3 | NULL |
| PHY-101 | Physical Principles | Physics | 4 | NULL |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

mysql> 
```



[Open in app](#)[Get started](#)

## 23. Views

Views are virtual SQL tables created using a result set of a statement. It contains rows and columns and is very similar to a general SQL table. A view always shows up-to-date data within the database.

### CREATE VIEW

```
CREATE VIEW <view_name> AS
  SELECT <col_name1>, <col_name2>, ...
  FROM <table_name>
  WHERE <condition>;
```

### DROP VIEW

```
DROP VIEW <view_name>;
```

### Example

Create a view consisting of courses with 3 credits.

```
vijinimallawaarachchi — mysql -u root -p — 81x20
[mysql]> CREATE VIEW my_view AS SELECT * FROM course WHERE credits=3;
Query OK, 0 rows affected (0.01 sec)

[mysql]> SELECT * FROM my_view;
+-----+-----+-----+
| course_id | title           | dept_name | credits |
+-----+-----+-----+
| BIO-399   | Computational Biology | Biology    | 3        |
| CS-315    | Robotics          | Comp. Sci. | 3        |
| CS-319    | Image Processing  | Comp. Sci. | 3        |
| CS-347    | Database System Concepts | Comp. Sci. | 3        |
| EE-181    | Intro. to Digital Systems | Elec. Eng. | 3        |
| FIN-201   | Investment Banking | Finance    | 3        |
| HIS-351   | World History     | History    | 3        |
| MU-199    | Music Video Production | Music     | 3        |
+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> ■
```



[Open in app](#)[Get started](#)

## 24. Aggregate Functions

These functions are used to obtain a cumulative result relevant to the data being considered. Following are the commonly used aggregate functions.

- **COUNT(col\_name)** — Returns the number of rows
- **SUM(col\_name)** — Returns the sum of the values in a given column
- **AVG (col\_name)** — Returns the average of the values of a given column
- **MIN(col\_name)** — Returns the smallest value of a given column
- **MAX(col\_name)** — Returns the largest value of a given column

## 25. Nested Subqueries

Nested subqueries are SQL queries which include a **SELECT-FROM-WHERE** expression that is nested within another query.

### Example

Find courses offered in Fall 2009 and in Spring 2010.

```
SELECT DISTINCT course_id
  FROM section
 WHERE semester = 'Fall' AND year= 2009 AND course_id IN (
    SELECT course_id
      FROM section
     WHERE semester = 'Spring' AND year= 2010
);
```

```
vijinimallawaarachchi — mysql -u root -p — 81x12
mysql> SELECT DISTINCT course_id FROM section WHERE semester = 'Fall' AND year= 2009 AND course_id IN (SELECT course_id FROM section WHERE semester = 'Spring' AND year= 2010);
+-----+
| course_id |
+-----+
| CS-101    |
+-----+
1 row in set (0.00 sec)
```



[Open in app](#)[Get started](#)

Figure 18: Courses offered in Fall 2009 and in Spring 2010

Hope you found this article useful.

Thanks for reading everyone! 😊

Good luck for your interviews!

---

## Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

[Get this newsletter](#)

[About](#)   [Help](#)   [Terms](#)   [Privacy](#)

[Get the Medium app](#)



[Open in app](#)[Get started](#)