Published in Towards Data Science

Vijini Mallawaarachchi Follow

Sep 15, 2017 · 7 min read · ✨ · Listen

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
```

```
CREATE DATABASE university;

USE university;

SOURCE <path_of_DLL.sql_file>;

SOURCE <path_of_InsertStatements.sql_file>;
```

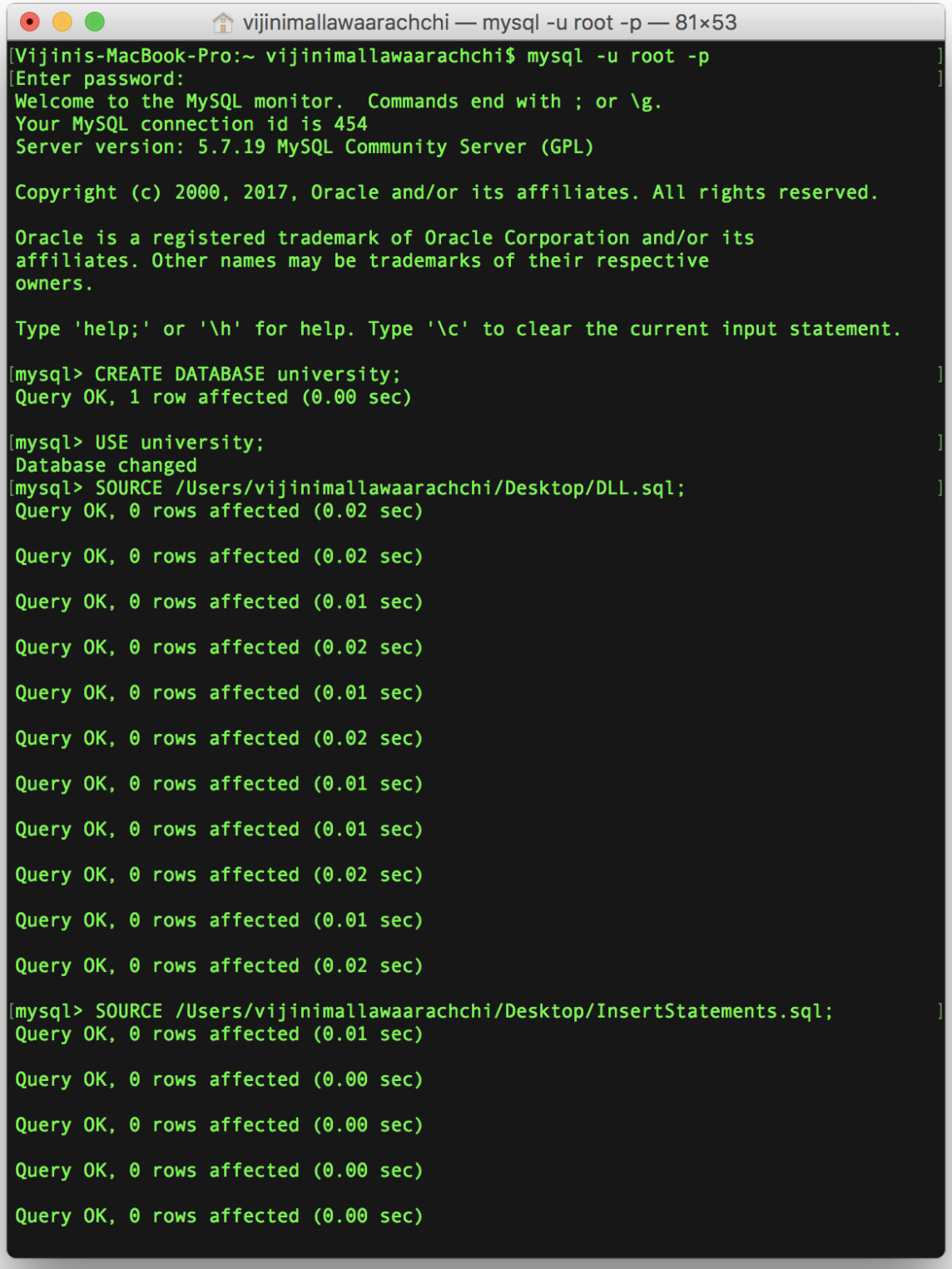


Figure 1: Executed queries to setup the database

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

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

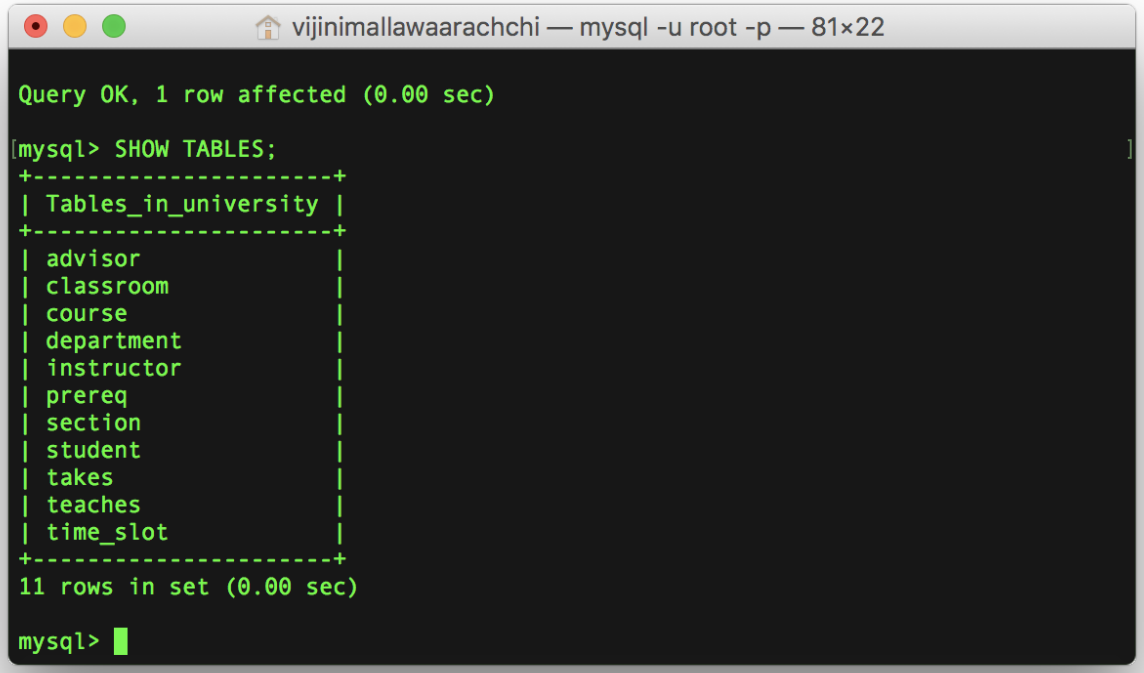


Figure 2: Tables in **university** database

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**

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>;
```


vijinimallawaarachchi — mysql -u root -p — 81×53

```
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> 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

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.


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

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

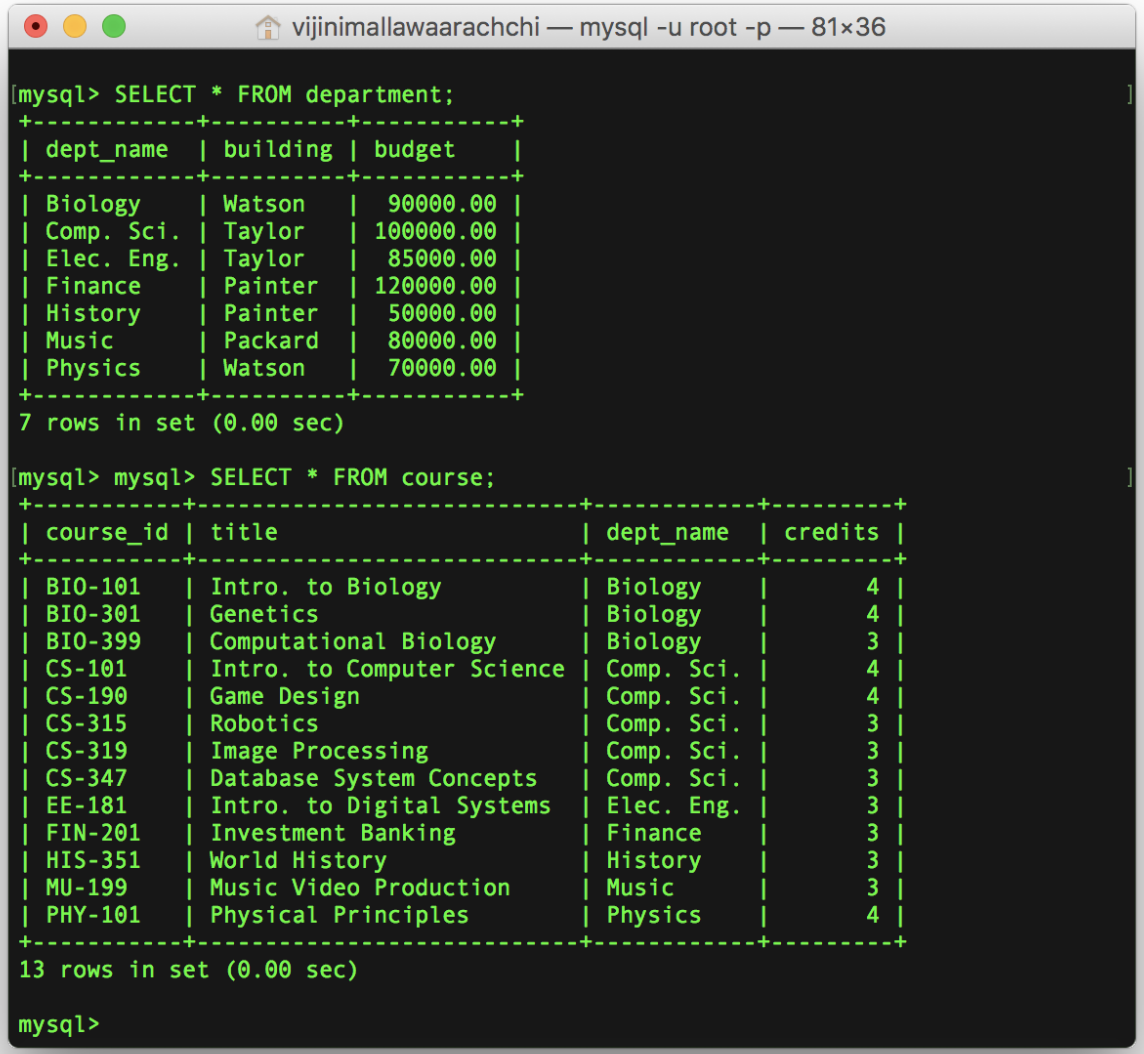


Figure 4: Select all from **department** and **course** tables

14. SELECT DISTINCT

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

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.

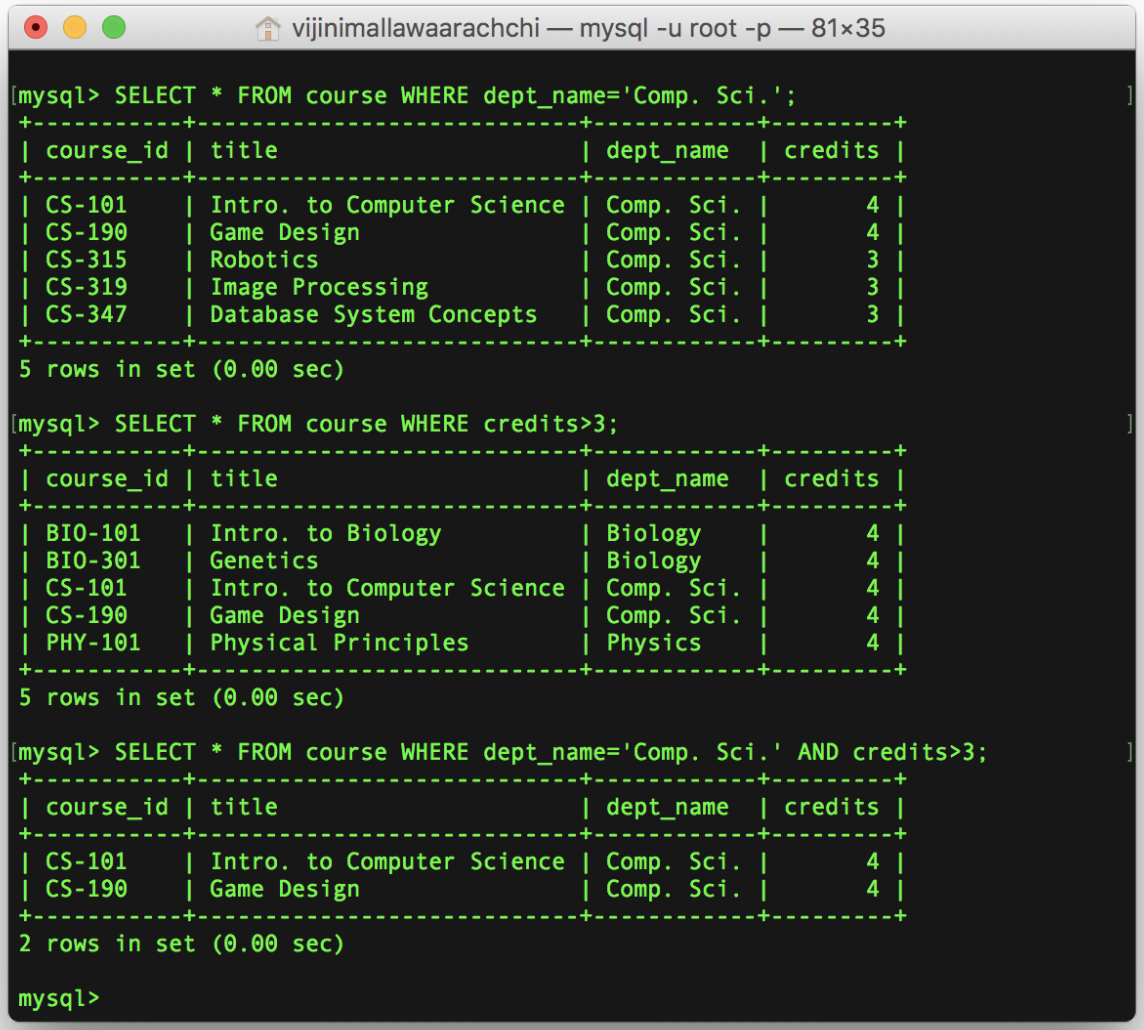


Figure 6: Usage of **WHERE** in **SELECT**

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>;
```

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

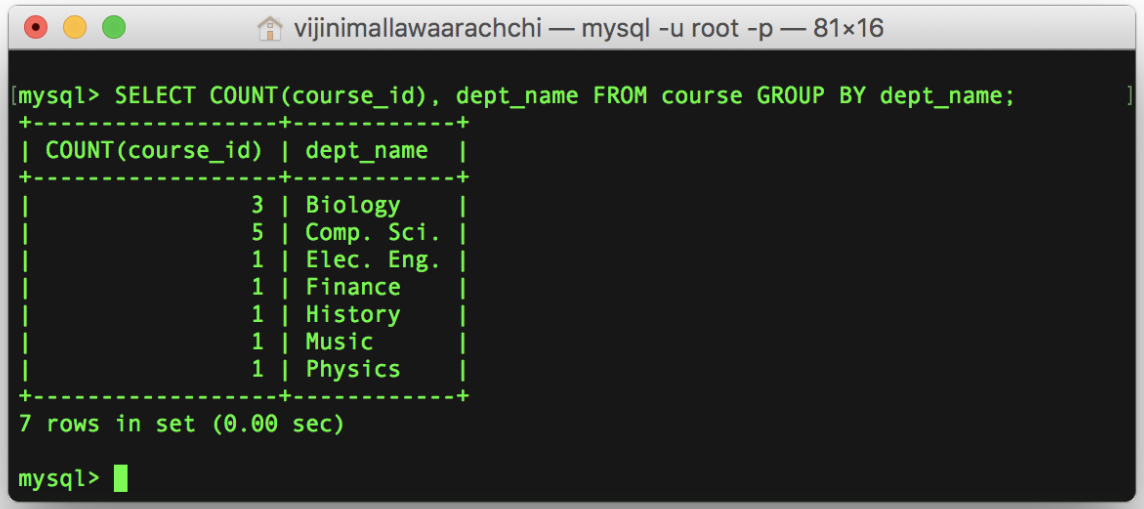


Figure 7: **Number of courses** for each **department**

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
```

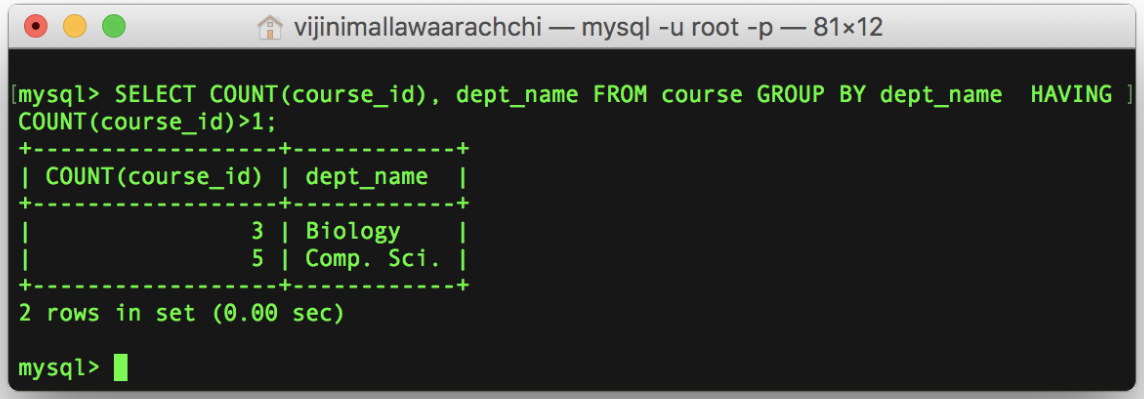


Figure 8: **Departments** offering more than one **course**

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;
```

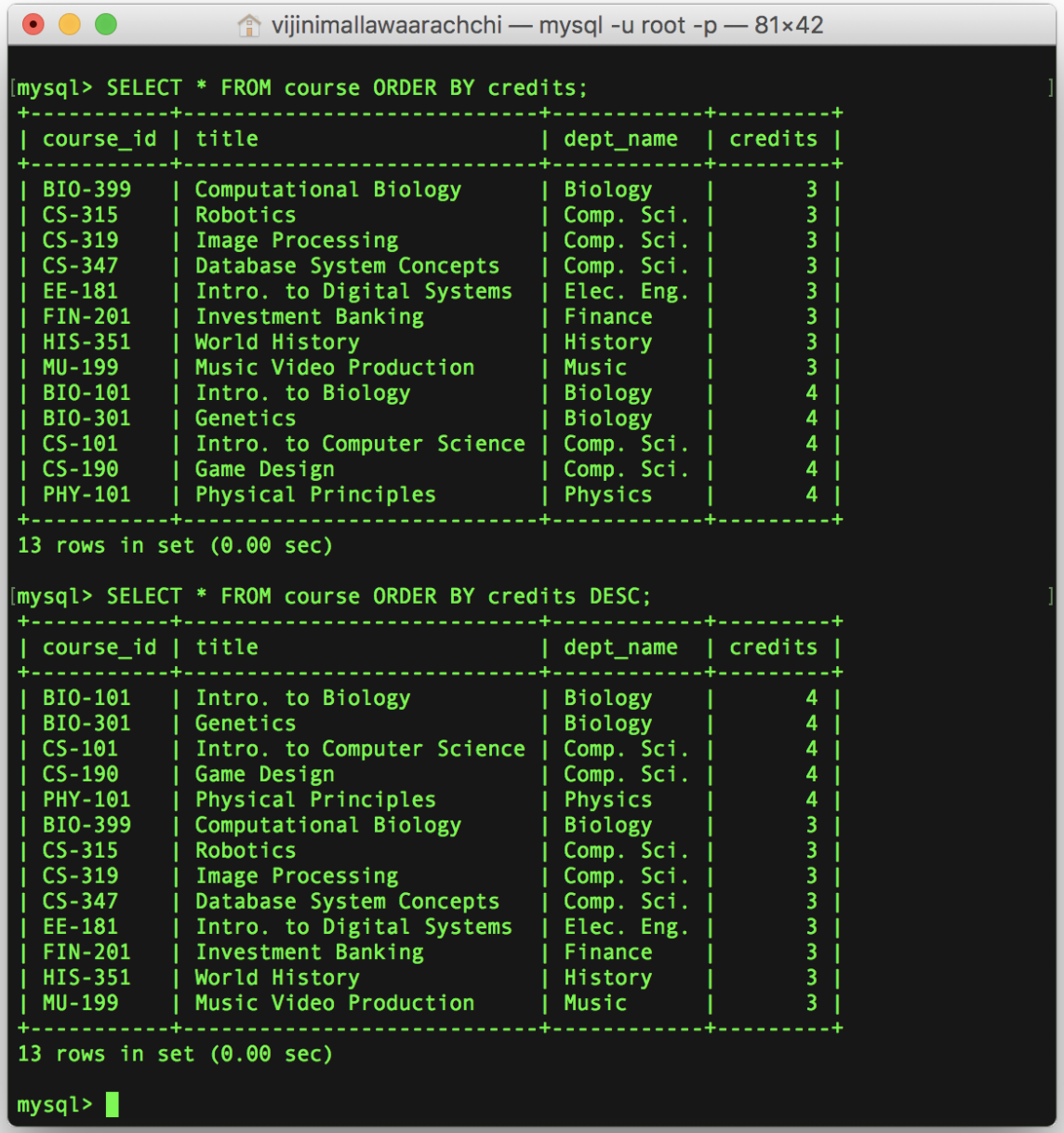


Figure 9: **Courses** sorted according to **credits**

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>, …
```

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

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

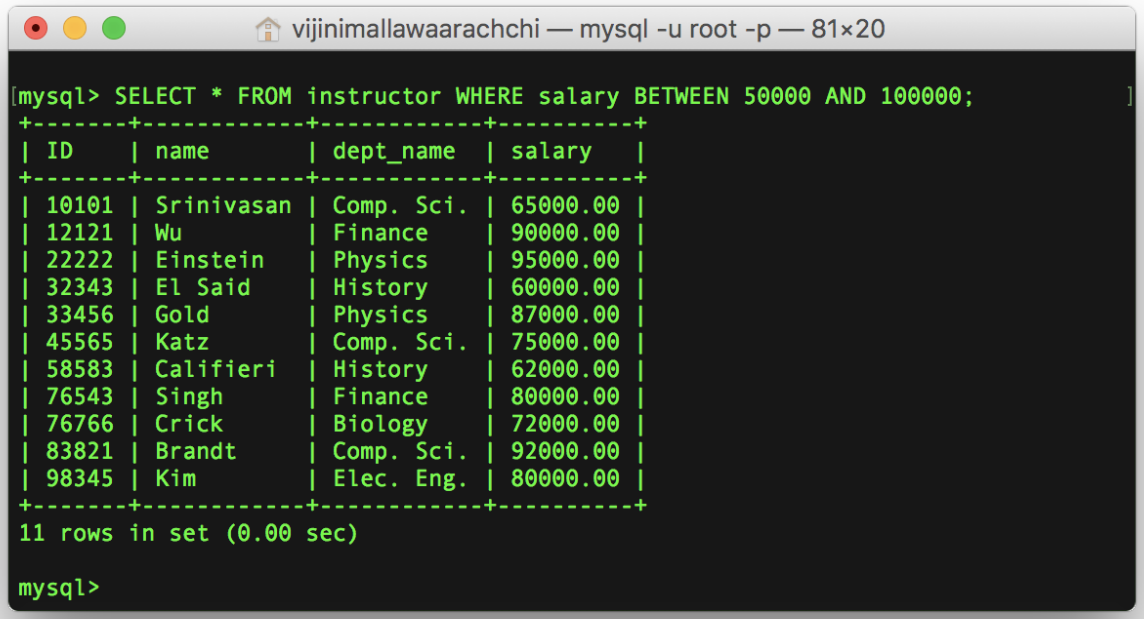


Figure 10: **Instructors** who get a salary in between 50 000 and 100 000

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>
```

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-___';
```

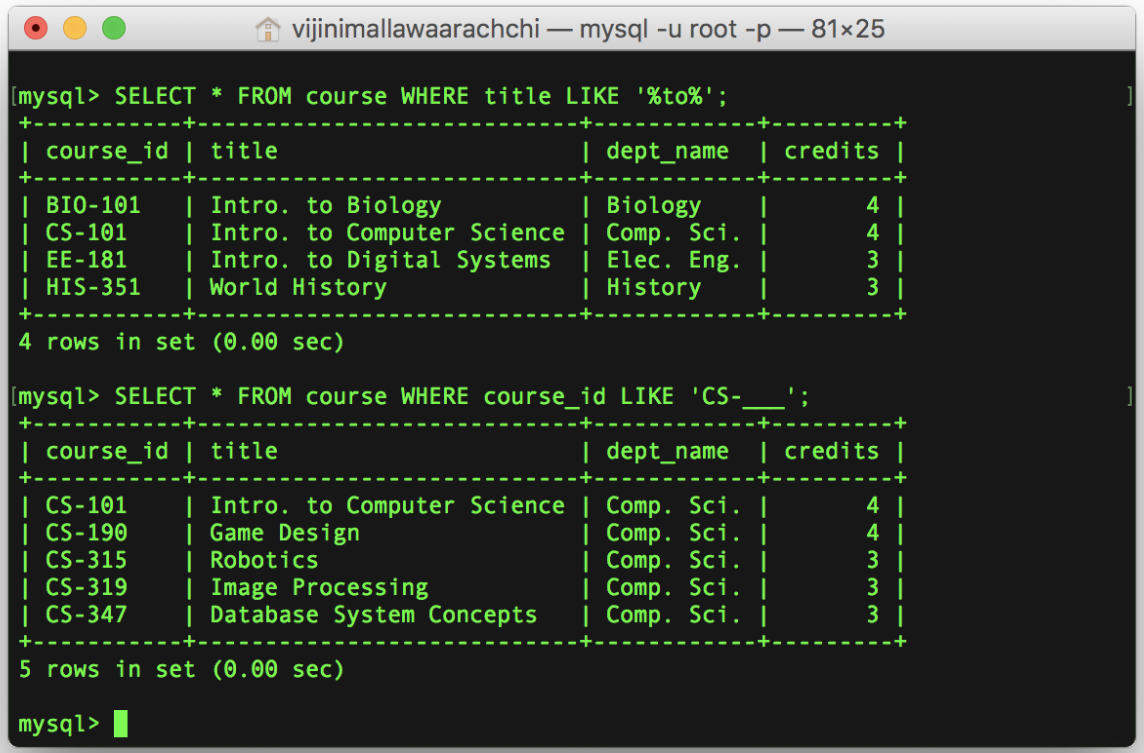



Figure 11: Use of **wildcard operators** with **LIKE**

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>, …);
```

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

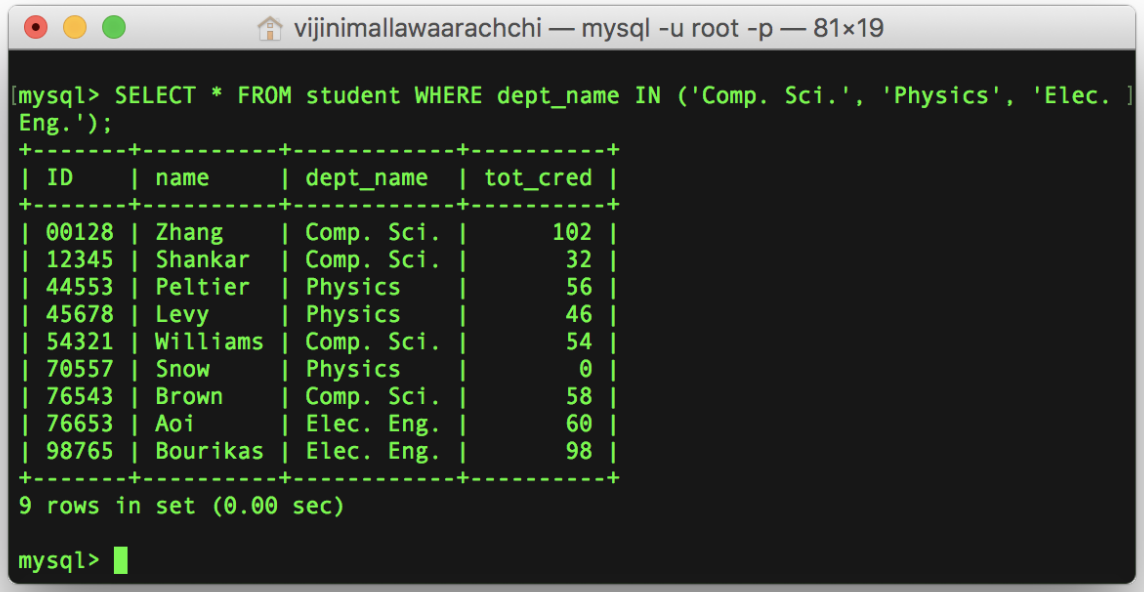


Figure 12: **Students** in the **departments** of Comp. Sci., Physics, and Elec. Eng

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**.

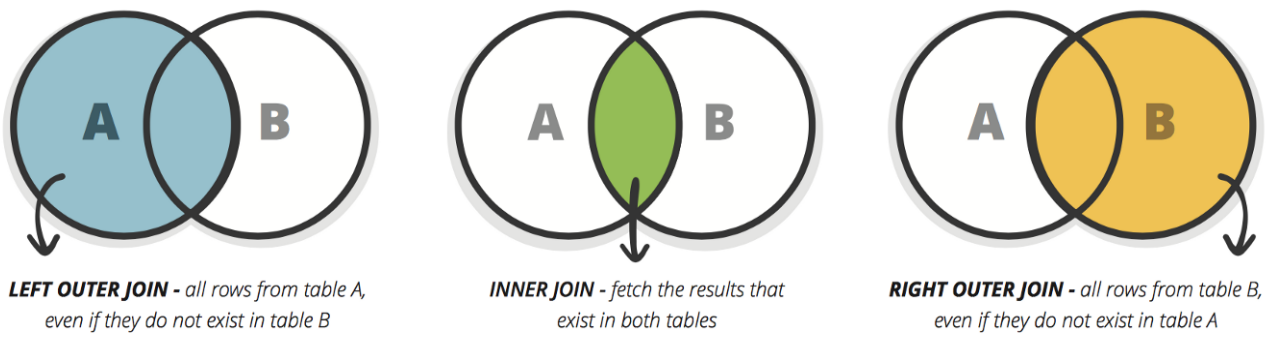


Figure 13: Types of joins (Source: http://files.zeroturnaround.com/pdf/zt_sql_cheat_sheet.pdf)

```
    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;
```

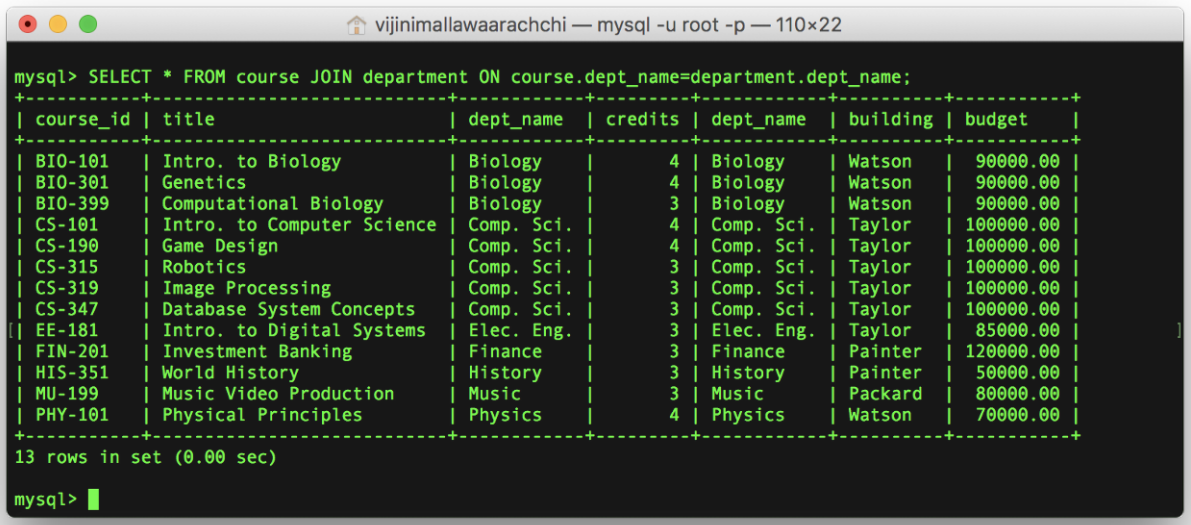


Figure 14: All the **courses** with **department** details

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;
```

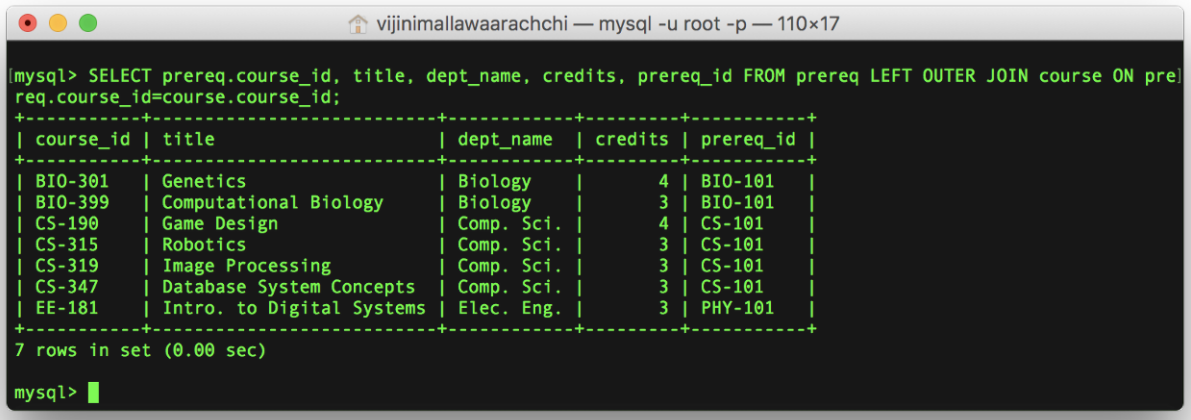
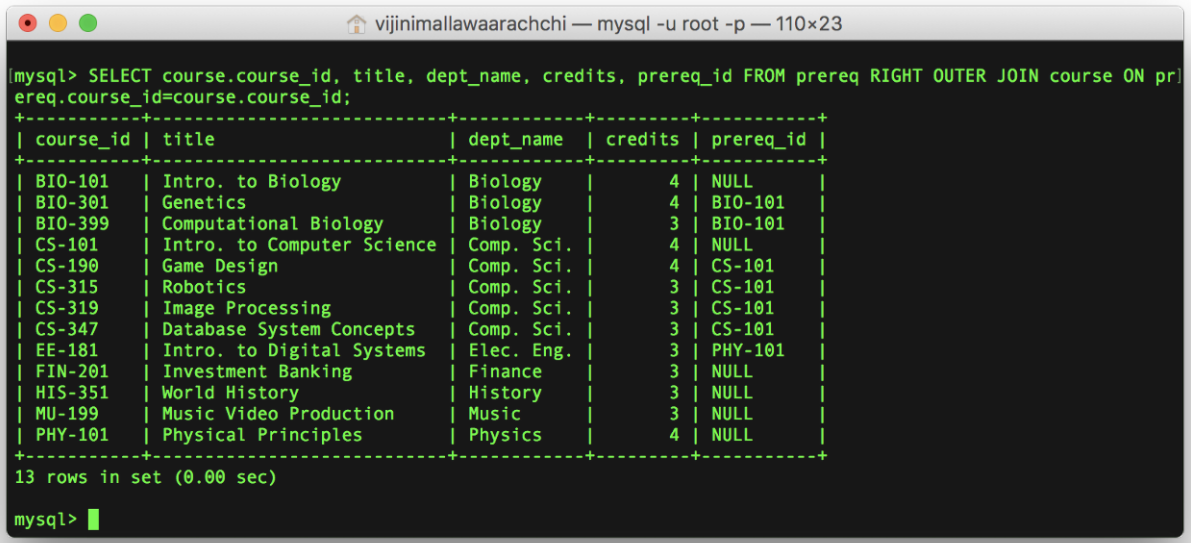


Figure 15: All the **prerequisite** courses with their **course** details

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;
```



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

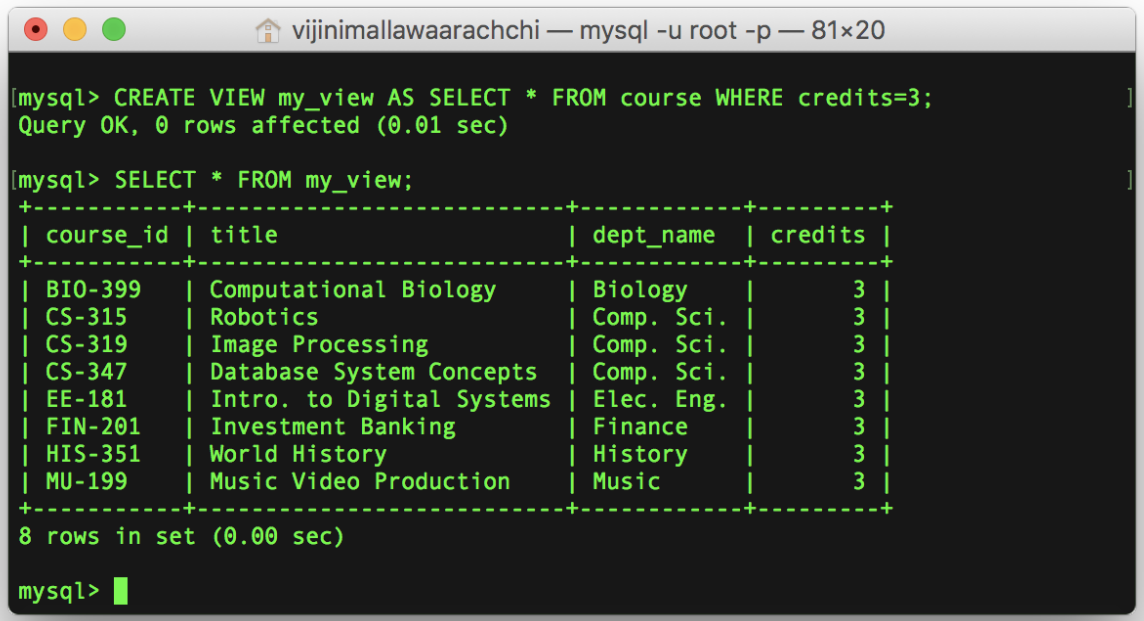
```
CREATE VIEW <view_name> AS
    SELECT <col_name1>, <col_name2>, …
    FROM <table_name>
    WHERE <condition>;
```

DROP VIEW

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

Example

Create a view consisting of courses with 3 credits.



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
    );
```

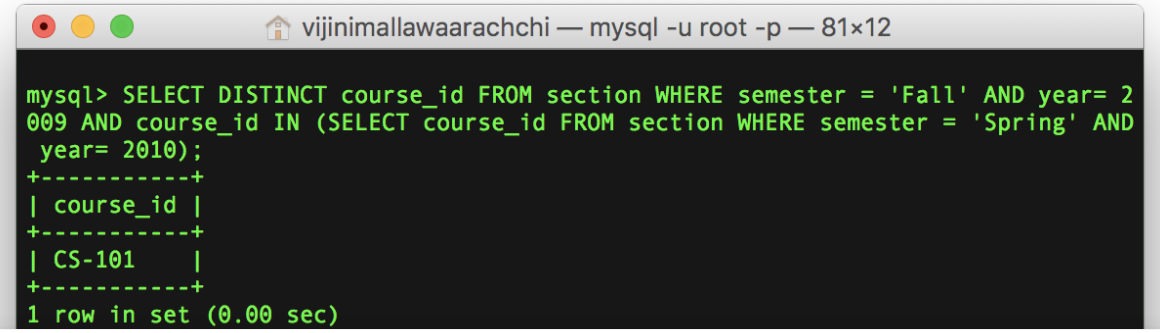


Figure 18: **Courses** offered in **Fall 2009** and in **Spring 2010**

Hope you found this article useful.

Thanks for reading everyone! 😃

Good luck for your interviews!