```
6,Neenu J,70,EC
7,Bob A,30,EC
8,Anu M,90,AE
9,Sruthi,89,AE
10,Andrew,89,AE
```

Execute the scripts in the following sequence:

```
$ ./create_db.sh
Created DB
Created table students

$ ./write_to_db.sh studentdat.csv
Wrote data into DB

$ ./read_db.sh
Department : CS
rank  name  mark
1  Navin M  98
2  Nawaz O  80
3  Kavya N  70

Department : EC
rank  name  mark
1  Hari S  80
2  Neenu J 70
3  Alex M  50
4  Bob A   30

Department : AE
rank  name  mark
1  Anu M    90
2  Sruthi   89
3  Andrew   89
```

How it works...

We will now see the explanation of the preceding scripts one by one. The first script `create_db.sh` is used to create a database called `students` and a table named `students` inside it. We need the MySQL username and password to access or modify data in the DBMS. The variables USER and PASS are used to store the username and password. The `mysql` command is used for MySQL manipulations. The `mysql` command can specify the username by using `-u` and the password by using `-pPASSWORD`. The other command argument for the `mysql` command is the database name. If a database name is specified as an argument to the `mysql` command, it will use that for database operations, else we have to explicitly specify in the SQL query about which database is to be used with the use `database_name` query. The `mysql` command accepts the queries to be executed through standard input (`stdin`). The convenient way of supplying multiple lines through `stdin` is by using the `<<EOF` method. The text that appears in between `<<EOF` and `EOF` is passed to `mysql` as standard input. In the CREATE DATABASE query, we have redirected `stderr` to `/dev/null` in order to prevent displaying an error message. Also, in the table creation query, we have redirected `stderr` to `/dev/null` to ignore any errors that occur. Then, we check the exit status for the `mysql` command by using the exit status variable `$?` to know if a table or database already exists. If the database or table already exists, a message is displayed to notify that, else we will create them.

The next script `write_to_db.sh` accepts a filename of the student data CSV file. We read each line of the CSV file by using the `while` loop. So in each iteration, a line with comma separated values will be received. We then need to formulate the values in the line to an SQL query. For that, the easiest way to store data items in the comma-separated line is by using an array. We know that an array assignment is in the form `array=(val1 val2 val3)`. Here, the space character is the **Internal Field Separator** (**IFS**). We have a line with comma separated values, hence by changing the IFS to a comma, we can easily assign values to the array (`IFS=,`). The data items in the comma separated line are `id`, `name`, `mark`, and `department`. `id` and `mark` are integer values, whereas `name` and `dept` are strings (strings must be quoted). Also, the name can contain space characters. Space can conflict with the Internal Field Separator. Hence, we should replace the space in the name with a character (#) and replace it later after formulating the query. In order to quote the strings, the values in the array are reassigned prefix and suffix with `\"`. The `tr` is used to substitute space in the name to #. Finally, the query is formed by replacing the space character with a comma and replacing # with a space, and this query is executed.

The third script `read_db.sh` is used to find out the department and print the rank list of students for each department. The first query is used to find distinct names of departments. We use a `while` loop to iterate through each department and run the query to display student details in the order of highest marks. `SET @i=0` is an SQL construct used to set the variable `i=0`. On each row it is incremented and is displayed as the rank of the student.