EXPERIMENT - 1

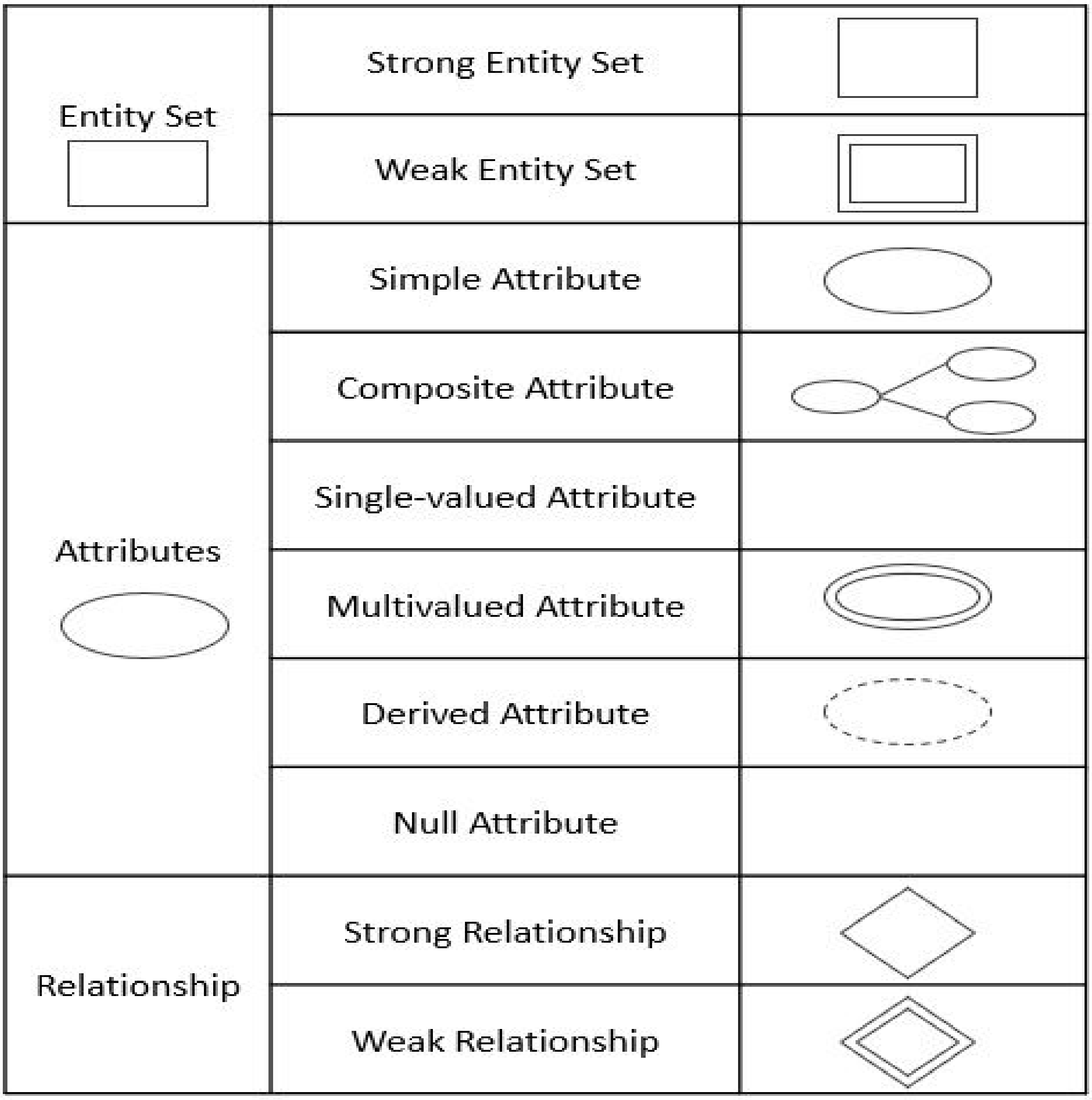
Objective: Creating Entity-Relationship Diagram using case tools.

Theory:

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

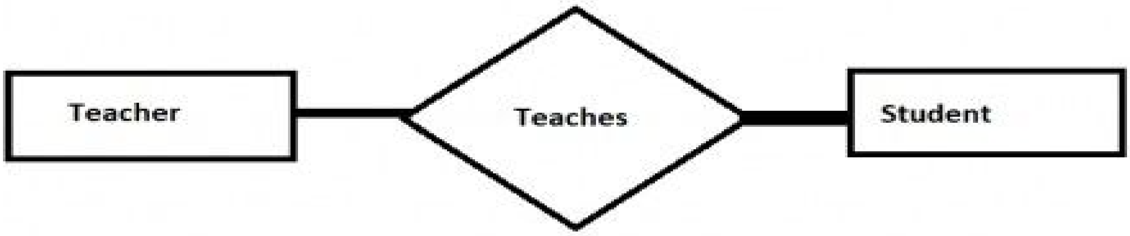
A Relationship describes relations between entities. Relationships are represented using diamonds or rhombus.

Symbols and Notations: As in the ER diagram we have to represent each component of Er model graphically so there must be some symbols or notation to represent each component. The table below displays some symbols to represent the components of the ER model.



Entity Set □	Strong Entity Set	□
	Weak Entity Set	⧈
Attributes ⬭	Simple Attribute	⬭
	Composite Attribute	⬭<⬭⬭
	Single-valued Attribute	
	Multivalued Attribute	◎
	Derived Attribute	⬯
	Null Attribute	
Relationship	Strong Relationship	◇
	Weak Relationship	◈

For Example,

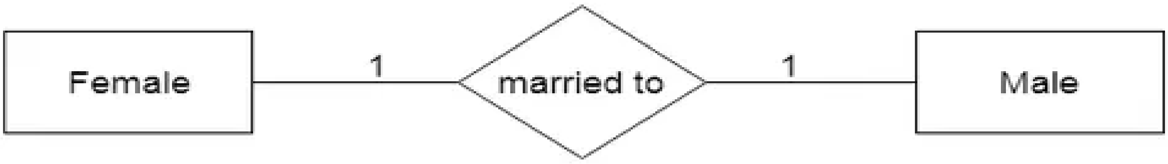


There are four types of relationships :

- One to one
- One to many
- Many to many
- Many to one

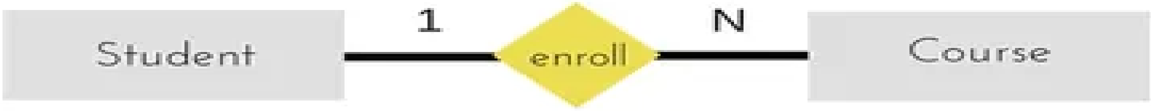
One to one Relationship: A one-to-one relationship is mostly used to split an entity in two to provide information concisely and make it more understandable. The figure below shows an example of a one-to-one relationship.

Example :



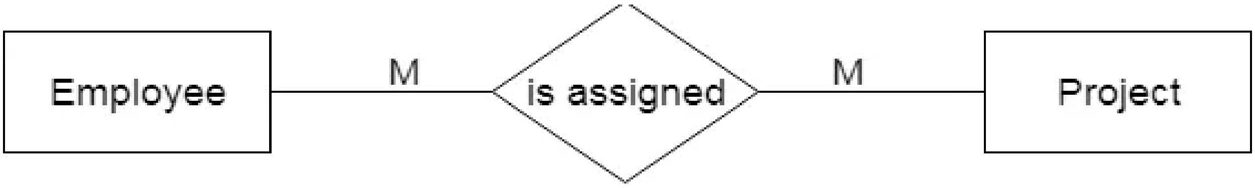
One to many Relationship: A one-to-many relationship refers to the relationship between two entities X and Y in which an instance of X may be linked to many instances of Y, but an instance of Y is linked to only one instance of X. The figure below shows an example of a one-to-many relationship.

Example: Students can enrol for only one course.



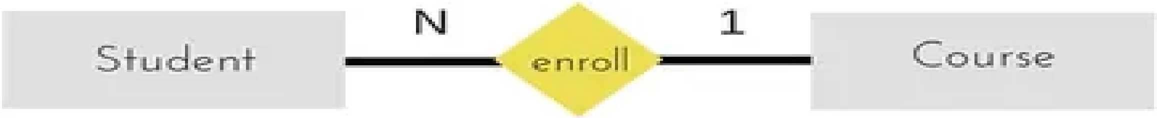
Many-to-Many cardinality: A many-to-many relationship refers to the relationship between two entities X and Y in which X may be linked to many instances of Y and vice versa. The figure below shows an example of a many-to-many relationship. Note that a many-to-many relationship is split into a pair of one-to-many relationships in a physical Entity Relationship Diagram.

Example: Employees can be assigned to many projects and projects can have many employees.



Many-to-one relationship: When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

Example: Student enrols for only one course, but a course can have many students.



EXPERIMENT - 2

Objective: Write SQL queries to implement DDL commands.

Theory:

It is used to communicate with databases. DDL is used to:

- Create an object
- Alter the structure of an object
- To drop the object created

The commands used are:

- Create
- Alter
- Drop
- Truncate
- Rename

The description of the following commands are as follows:

1. Command Name: **CREATE**
   COMMAND DESCRIPTION: CREATE command is used to create objects in the database.
2. Command Name: **DROP**
   COMMAND DESCRIPTION: DROP command is used to delete the object from the database.
3. Command Name: **TRUNCATE**
   COMMAND DESCRIPTION: TRUNCATE command is used to remove all the records from the table.
4. Command Name: **ALTER**
   COMMAND DESCRIPTION: ALTER command is used to alter the structure of the database.
5. Command Name: **RENAME**
   COMMAND DESCRIPTION: RENAME command is used to rename the objects.

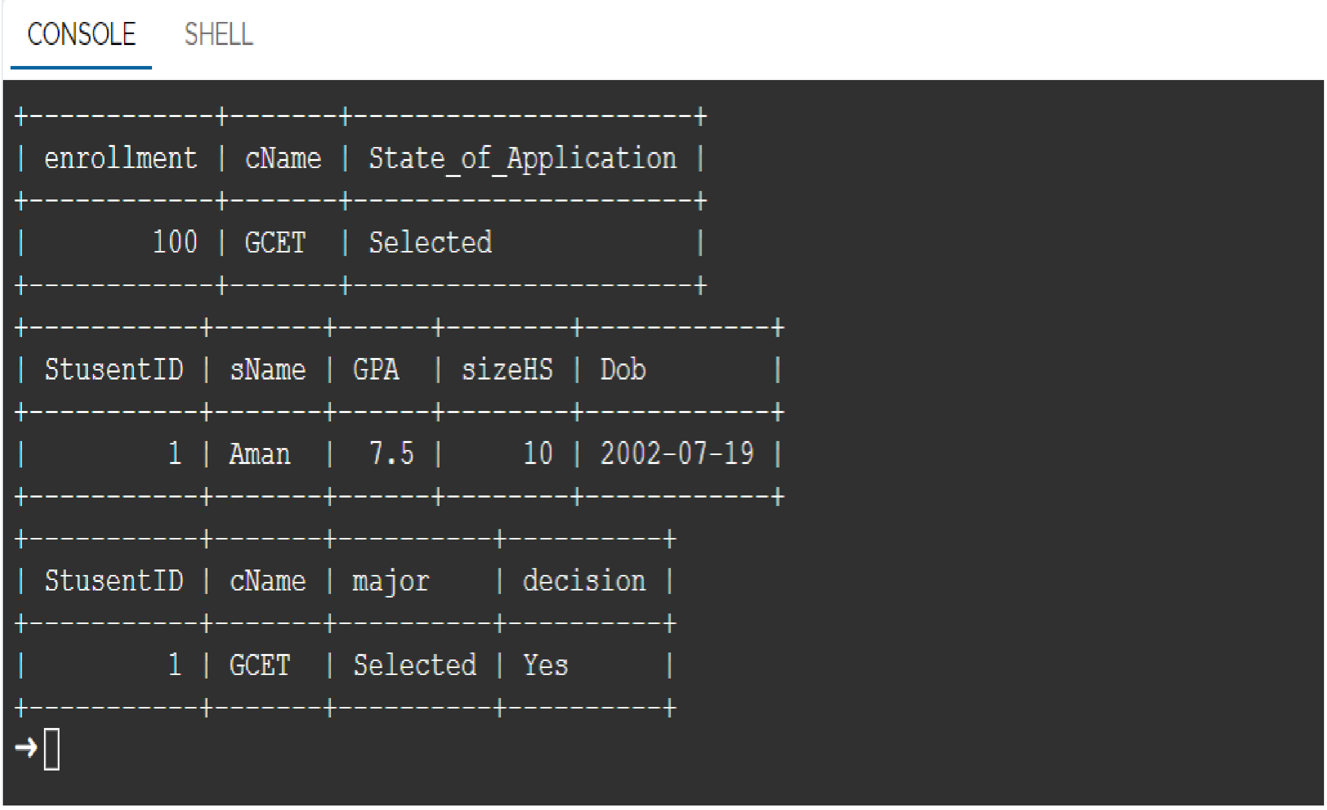
Exercise-1:

Problem Statement: To create a database named “Collegeapplication” with tables named “College”, “Student”, “Applicant” with some data entries.

SQl Program:

```
create Database Collegeapplication;
use Collegeapplication;
CREATE TABLE College (
    enrollment int,
    cName varchar(255),
    State_of_Application varchar(20)
);
CREATE TABLE Student (
    Student_ID int,
    sName varchar(20),
    GPA float,
    sizeHS int,
    Dob date
);
CREATE TABLE Applicant (
     Student_ID int,
     cName varchar(20),
     major varchar(20),
     decision varchar(10)
);
INSERT INTO College
VALUES (100,"GCET","Selected");
Insert Into Student
VALUES (1,"Aman",7.5,10,"2002-07-19");
INSERT INTO Applicant
VALUES (1,"GCET","Selected","Yes");
Select * from College;
Select * from Student;
Select * from Applicant;
```

Output:



```
+------------+-------+----------------------+
| enrollment | cName | State_of_Application |
+------------+-------+----------------------+
|        100 | GCET  | Selected             |
+------------+-------+----------------------+
+-----------+-------+------+--------+------------+
| StusentID | sName | GPA  | sizeHS | Dob        |
+-----------+-------+------+--------+------------+
|         1 | Aman  |  7.5 |     10 | 2002-07-19 |
+-----------+-------+------+--------+------------+
+-----------+-------+----------+----------+
| StusentID | cName | major    | decision |
+-----------+-------+----------+----------+
|         1 | GCET  | Selected | Yes      |
+-----------+-------+----------+----------+
➜
```

Exercise-2:

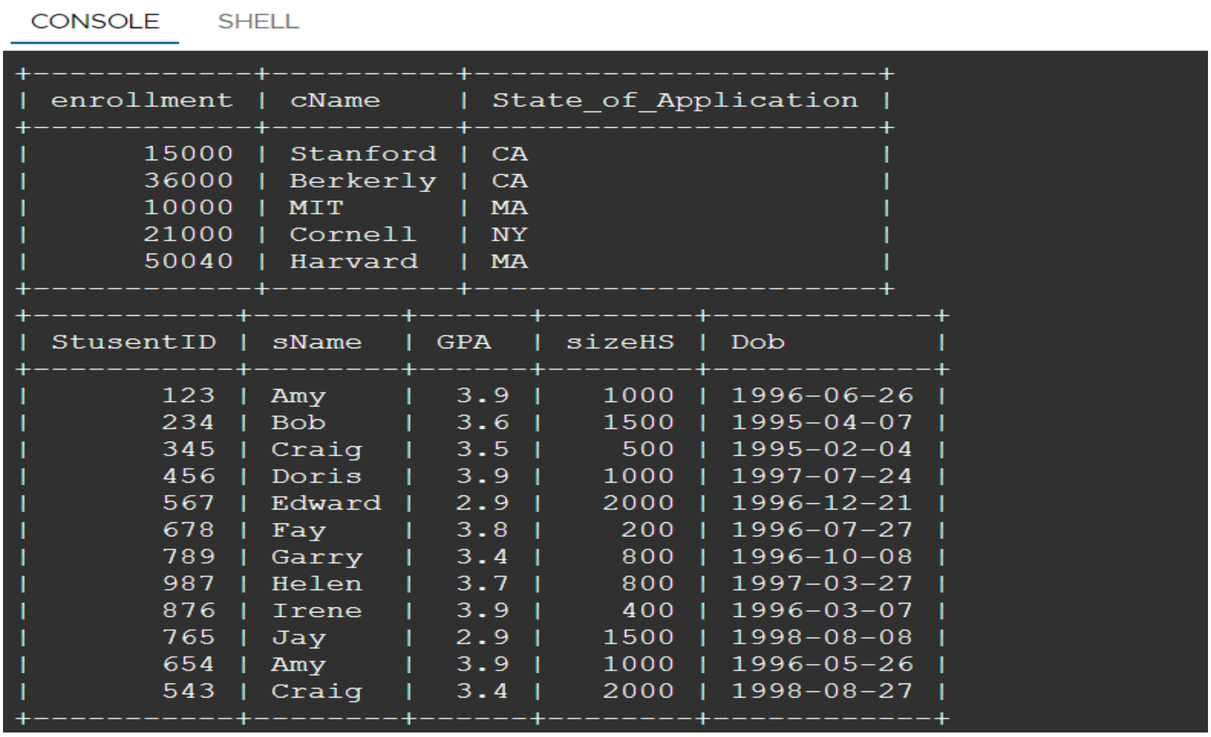
Problem Statement: Create the “Student”, “College” and “Apply” tables in the “CollegeApplication” database and insert the provided data in them.

SQl Program:

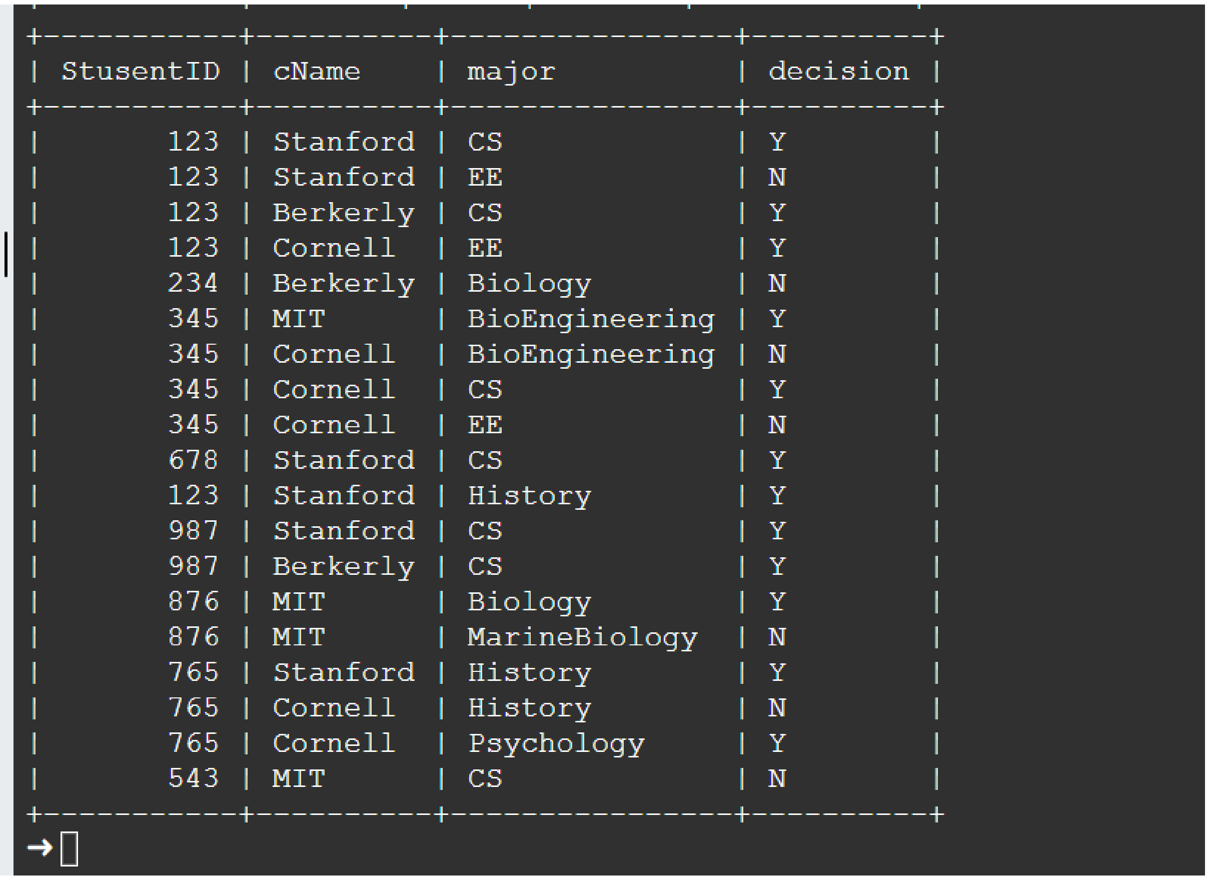
```
create Database Collegeapplication;
use Collegeapplication;
CREATE TABLE College (
    enrollment int,
    cName varchar(255),
    State_of_Application varchar(20)
);
CREATE TABLE Student (
    Student_ID int,
    sName varchar(20),
    GPA float,
    sizeHS int,
    Dob date
);
CREATE TABLE Applicant (
     Student_ID int,
     cName varchar(20),
     major varchar(20),
     decision varchar(10)
);
INSERT INTO College
VALUES (15000,"Stanford","CA"),
(36000,"Berkerly","CA"),
(10000,"MIT","MA"),
(21000,"Cornell","NY"),
(50040,"Harvard","MA")
;
Insert Into Student
VALUES (123,"Amy",3.9,1000,"1996-06-26"),
(234,"Bob",3.6,1500,"1995-04-07"),
(345,"Craig",3.5,500,"1995-02-04"),
(456,"Doris",3.9,1000,"1997-07-24"),
(567,"Edward",2.9,2000,"1996-12-21"),
(678,"Fay",3.8,200,"1996-07-27"),
(789,"Garry",3.4,800,"1996-10-08"),
(987,"Helen",3.7,800,"1997-03-27"),
(876,"Irene",3.9,400,"1996-03-07"),
(765,"Jay",2.9,1500,"1998-08-08"),
(654,"Amy",3.9,1000,"1996-05-26"),
(543,"Craig",3.4,2000,"1998-08-27")
```

```
;
INSERT INTO Applicant
VALUES (123,"Stanford","CS","Y"),
(123,"Stanford","EE","N"),
(123,"Berkerly","CS","Y"),
(123,"Cornell","EE","Y"),
(234,"Berkerly","Biology","N"),
(345,"MIT","BioEngineering","Y"),
(345,"Cornell","BioEngineering","N"),
(345,"Cornell","CS","Y"),
(345,"Cornell","EE","N"),
(678,"Stanford","CS","Y"),
(123,"Stanford","History","Y"),
(987,"Stanford","CS","Y"),
(987,"Berkerly","CS","Y"),
(876,"MIT","Biology","Y"),
(876,"MIT","MarineBiology","N"),
(765,"Stanford","History","Y"),
(765,"Cornell","History","N"),
(765,"Cornell","Psychology","Y"),
(543,"MIT","CS","N")
;
Select * from College;
Select * from Student;
Select * from Applicant;
```

Output:



```
+------------+----------+----------------------+
| enrollment | cName    | State_of_Application |
+------------+----------+----------------------+
|      15000 | Stanford | CA                   |
|      36000 | Berkerly | CA                   |
|      10000 | MIT      | MA                   |
|      21000 | Cornell  | NY                   |
|      50040 | Harvard  | MA                   |
+------------+----------+----------------------+
+-----------+--------+------+--------+------------+
| StusentID | sName  | GPA  | sizeHS | Dob        |
+-----------+--------+------+--------+------------+
|       123 | Amy    |  3.9 |   1000 | 1996-06-26 |
|       234 | Bob    |  3.6 |   1500 | 1995-04-07 |
|       345 | Craig  |  3.5 |    500 | 1995-02-04 |
|       456 | Doris  |  3.9 |   1000 | 1997-07-24 |
|       567 | Edward |  2.9 |   2000 | 1996-12-21 |
|       678 | Fay    |  3.8 |    200 | 1996-07-27 |
|       789 | Garry  |  3.4 |    800 | 1996-10-08 |
|       987 | Helen  |  3.7 |    800 | 1997-03-27 |
|       876 | Irene  |  3.9 |    400 | 1996-03-07 |
|       765 | Jay    |  2.9 |   1500 | 1998-08-08 |
|       654 | Amy    |  3.9 |   1000 | 1996-05-26 |
|       543 | Craig  |  3.4 |   2000 | 1998-08-27 |
+-----------+--------+------+--------+------------+
```



```
+-----------+----------+----------------+----------+
| StusentID | cName    | major          | decision |
+-----------+----------+----------------+----------+
|       123 | Stanford | CS             | Y        |
|       123 | Stanford | EE             | N        |
|       123 | Berkerly | CS             | Y        |
|       123 | Cornell  | EE             | Y        |
|       234 | Berkerly | Biology        | N        |
|       345 | MIT      | BioEngineering | Y        |
|       345 | Cornell  | BioEngineering | N        |
|       345 | Cornell  | CS             | Y        |
|       345 | Cornell  | EE             | N        |
|       678 | Stanford | CS             | Y        |
|       123 | Stanford | History        | Y        |
|       987 | Stanford | CS             | Y        |
|       987 | Berkerly | CS             | Y        |
|       876 | MIT      | Biology        | Y        |
|       876 | MIT      | MarineBiology  | N        |
|       765 | Stanford | History        | Y        |
|       765 | Cornell  | History        | N        |
|       765 | Cornell  | Psychology     | Y        |
|       543 | MIT      | CS             | N        |
+-----------+----------+----------------+----------+
→ 
```

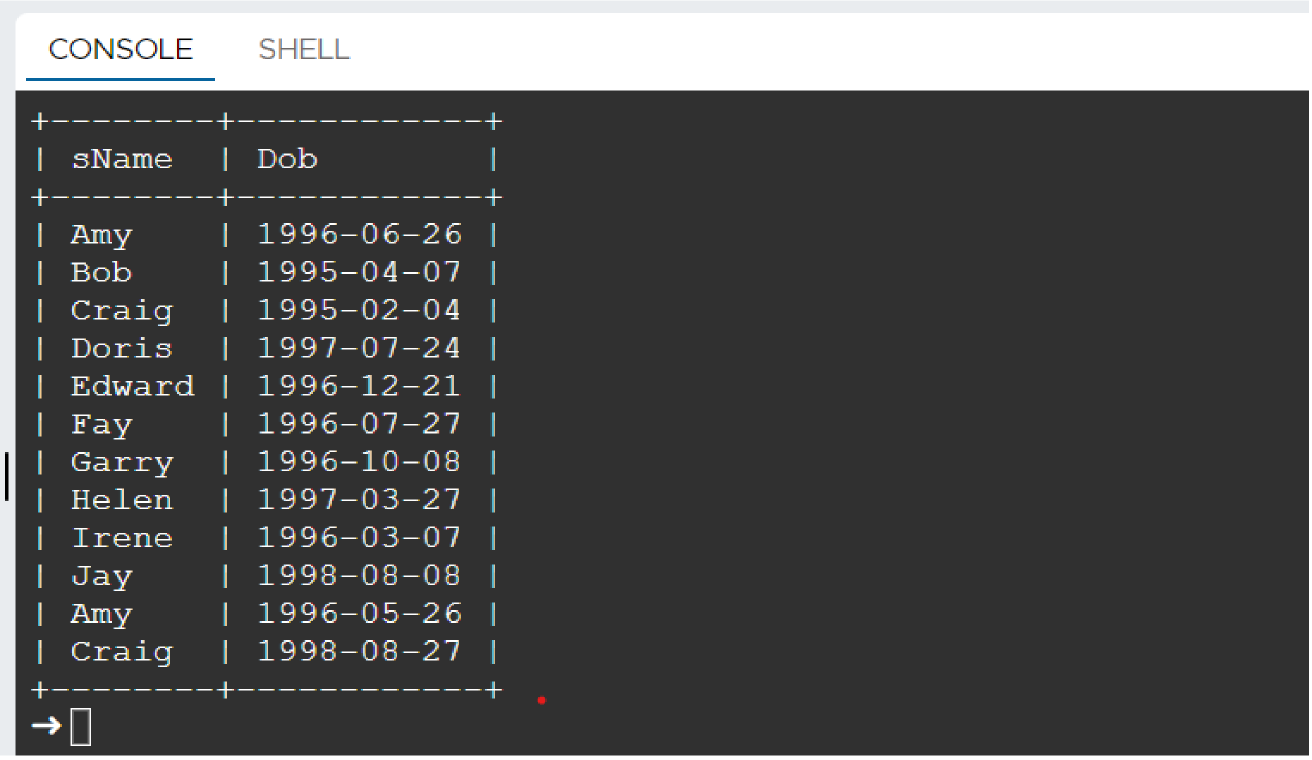
Exercise-3:

Problem Statement-1: List the Student name and dob from the student table.

SQl Program:

```
SELECT sName,Dob FROM Student;
```

Output:



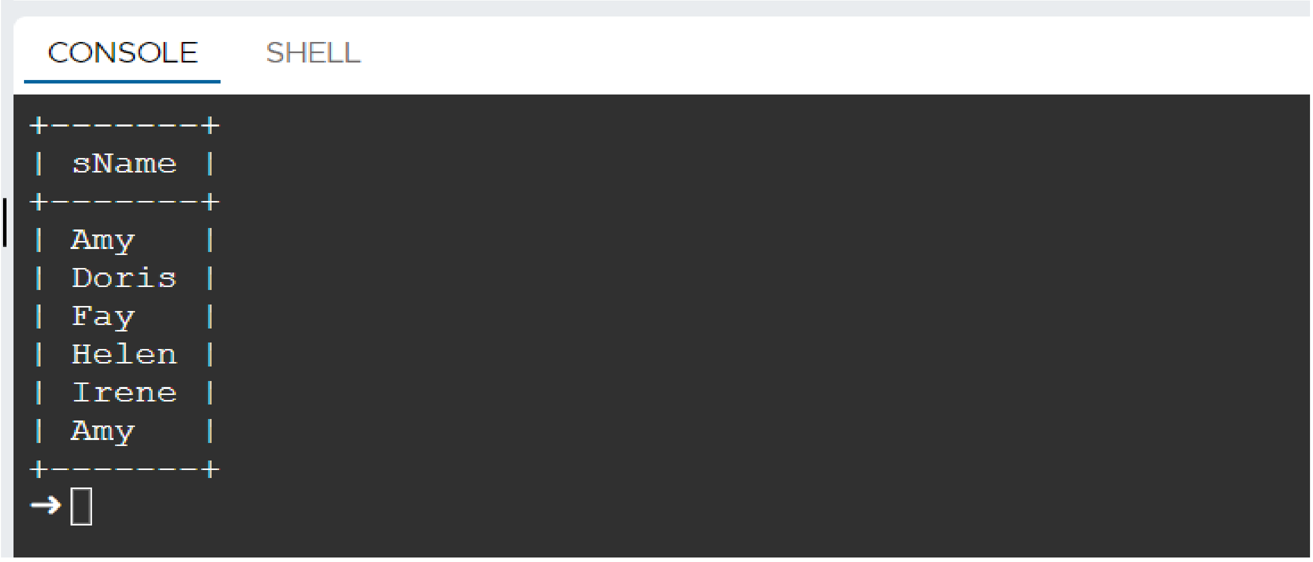
```
+---------+------------+
| sName   | Dob        |
+---------+------------+
| Amy     | 1996-06-26 |
| Bob     | 1995-04-07 |
| Craig   | 1995-02-04 |
| Doris   | 1997-07-24 |
| Edward  | 1996-12-21 |
| Fay     | 1996-07-27 |
| Garry   | 1996-10-08 |
| Helen   | 1997-03-27 |
| Irene   | 1996-03-07 |
| Jay     | 1998-08-08 |
| Amy     | 1996-05-26 |
| Craig   | 1998-08-27 |
+---------+------------+
→
```

Problem Statement-2: List the names of students scoring more than 3.7 GPA.

SQl Program:

```
SELECT sName FROM Student where GPA > 3.7;
```

Output:



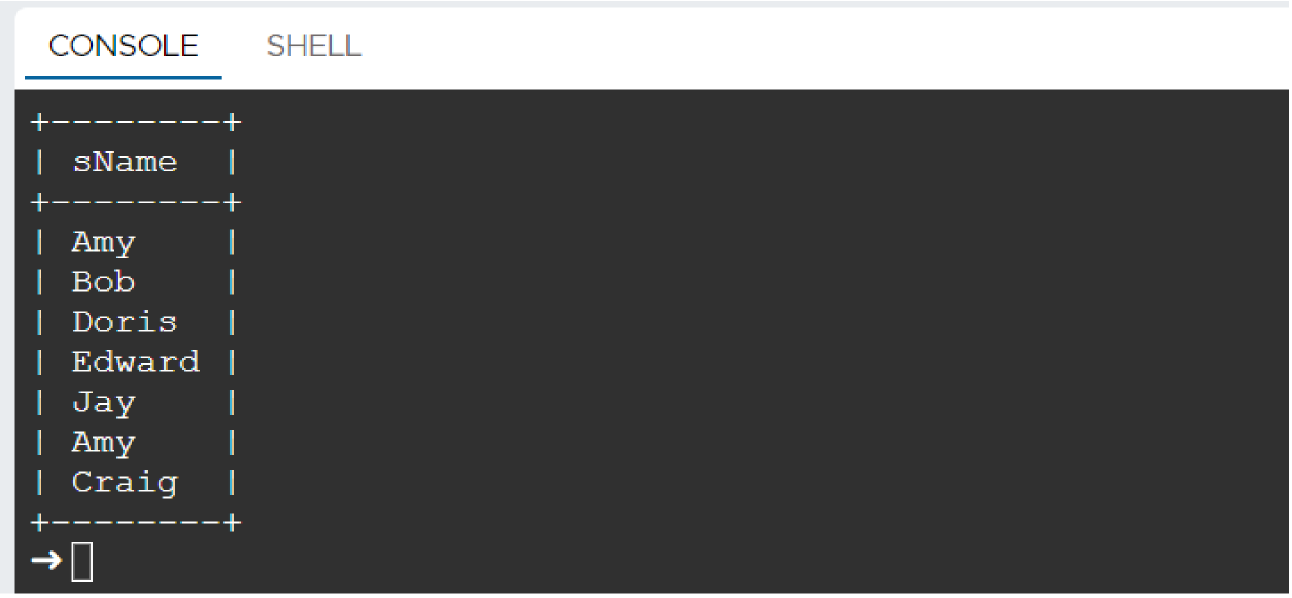
```
+--------+
| sName  |
+--------+
| Amy    |
| Doris  |
| Fay    |
| Helen  |
| Irene  |
| Amy    |
+--------+
→
```

Problem Statement-3: List the names of students whose high school size is at least 1000 and born after 1996.

SQl Program:

```
SELECT sName FROM Student where sizeHS>=1000 and Dob >
1996-12-31;
```

Output:



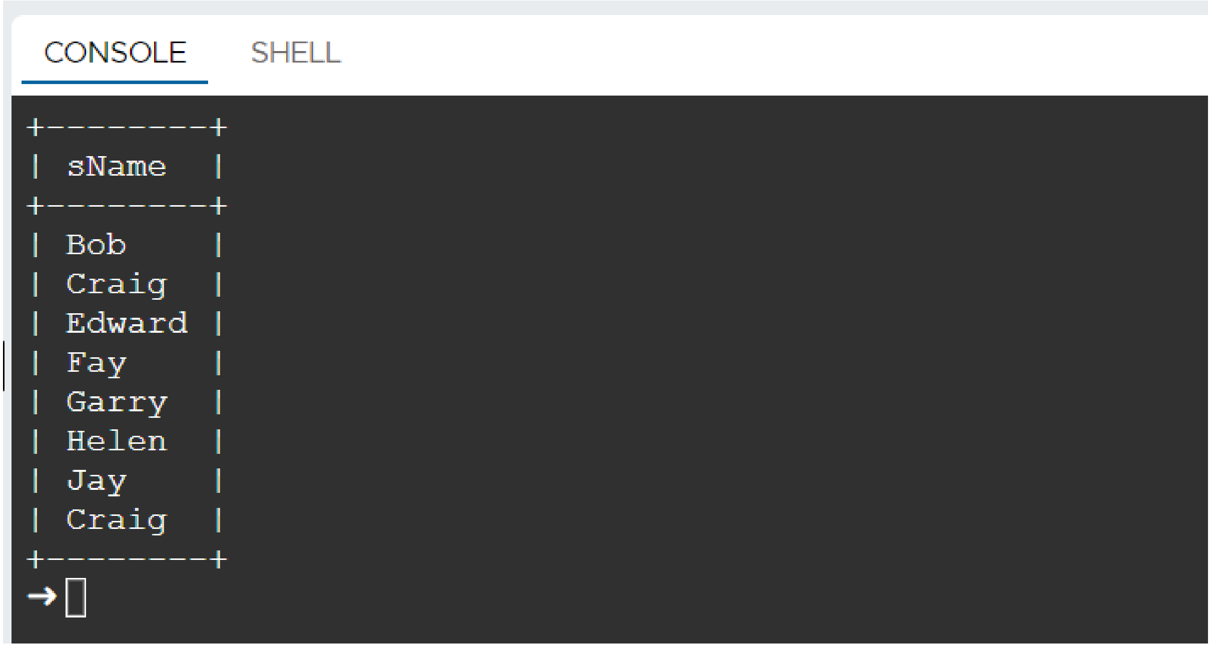
```
+---------+
| sName   |
+---------+
| Amy     |
| Bob     |
| Doris   |
| Edward  |
| Jay     |
| Amy     |
| Craig   |
+---------+
→
```

Problem Statement-4: List the names of students who are scoring GPA in between 2.9 and 3.9.

SQl Program:

```
SELECT sName FROM Student where GPA>2.9 and GPA<3.9;
```

Output:



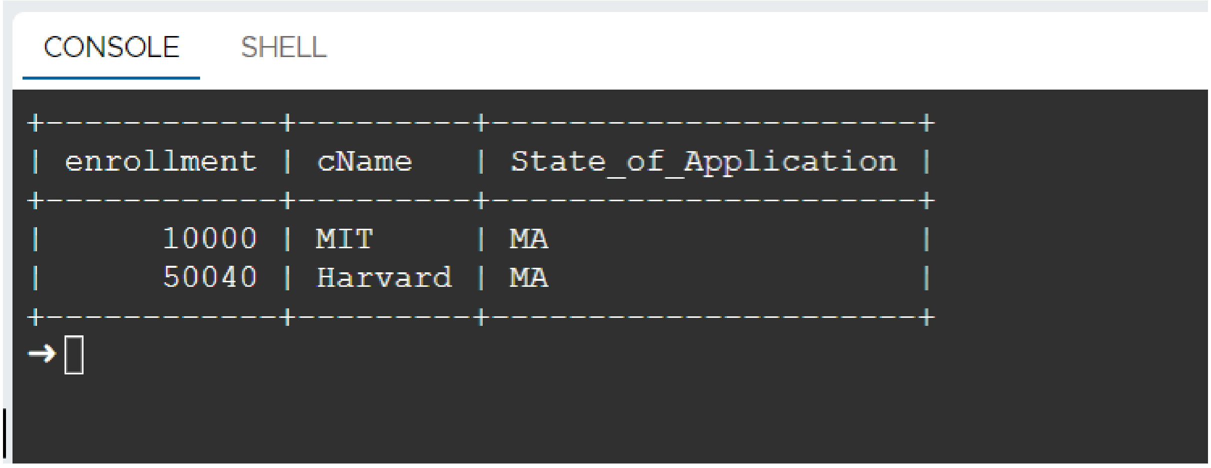
```
+---------+
| sName   |
+---------+
| Bob     |
| Craig   |
| Edward  |
| Fay     |
| Garry   |
| Helen   |
| Jay     |
| Craig   |
+---------+
→
```

Problem Statement-5: List all the details of colleges who are situated in MA.

SQl Program:

```
SELECT * from College where
State_of_Application="MA";
```

Output:



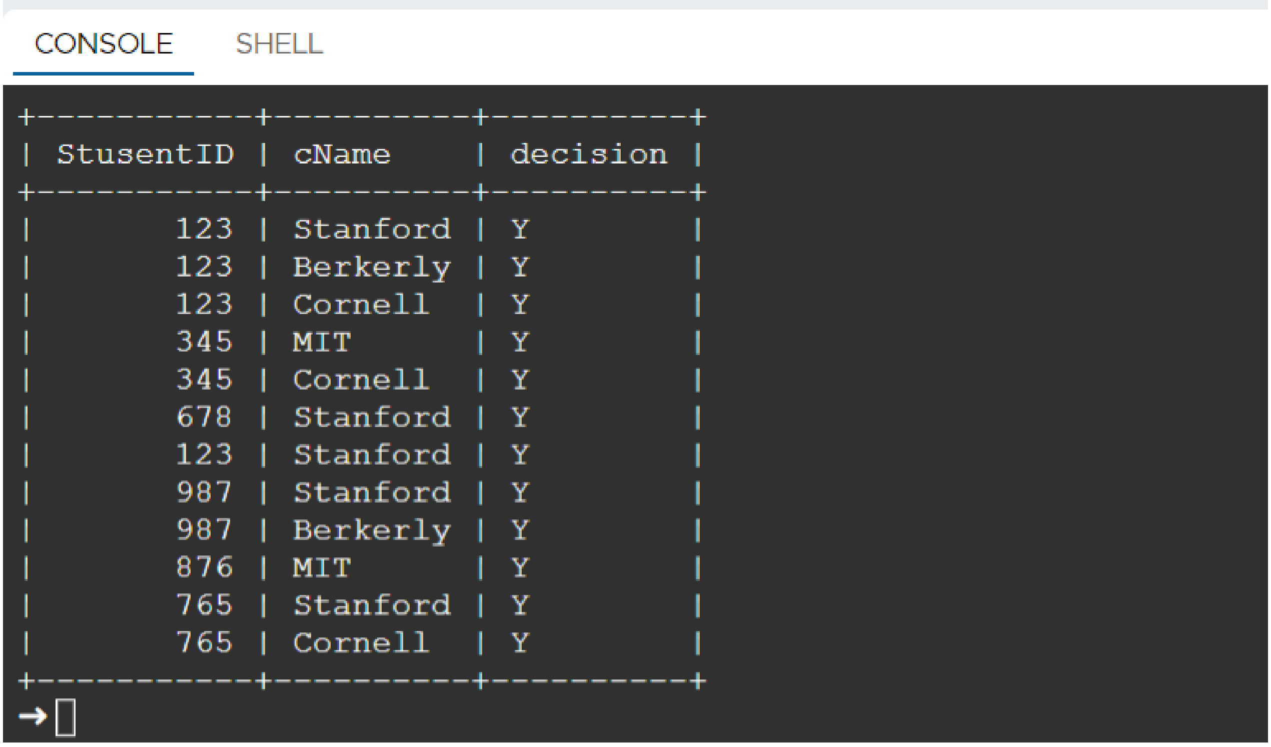
```
+------------+---------+----------------------+
| enrollment | cName   | State_of_Application |
+------------+---------+----------------------+
|      10000 | MIT     | MA                   |
|      50040 | Harvard | MA                   |
+------------+---------+----------------------+
→
```

Problem Statement-6: List the sID, cName, decision of the applications that are accepted.

SQl Program:

```
SELECT StudentID,cName,decision from Applicant where
decision="Y";
```

Output:



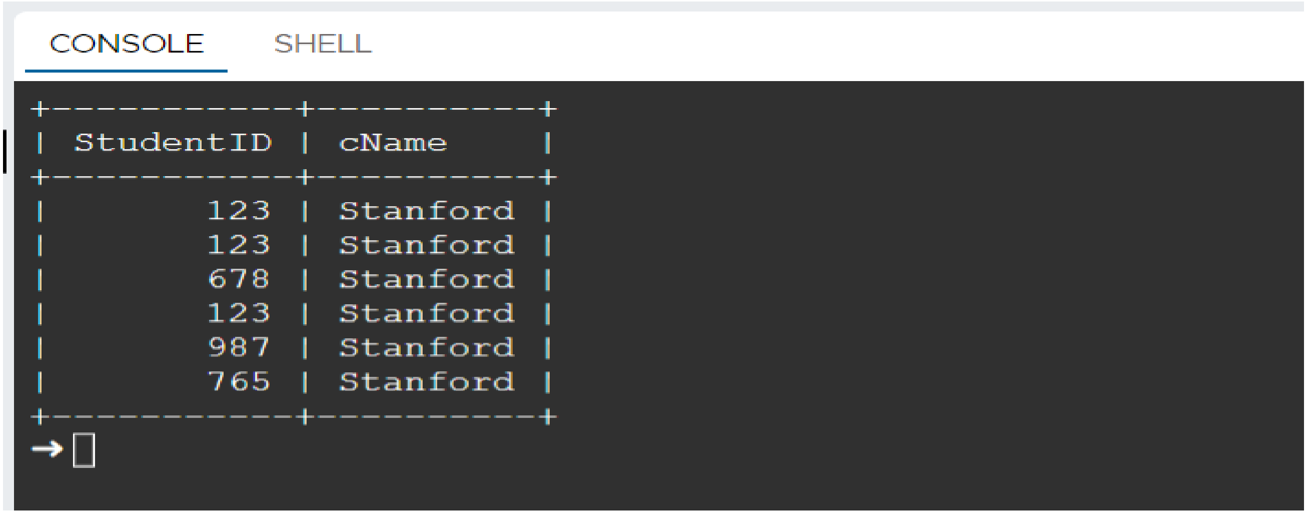
```
+-----------+----------+----------+
| StusentID | cName    | decision |
+-----------+----------+----------+
|       123 | Stanford | Y        |
|       123 | Berkerly | Y        |
|       123 | Cornell  | Y        |
|       345 | MIT      | Y        |
|       345 | Cornell  | Y        |
|       678 | Stanford | Y        |
|       123 | Stanford | Y        |
|       987 | Stanford | Y        |
|       987 | Berkerly | Y        |
|       876 | MIT      | Y        |
|       765 | Stanford | Y        |
|       765 | Cornell  | Y        |
+-----------+----------+----------+
→
```

<u>Problem Statement-7</u>: List the sID,cName of applications which are filled at Stanford.

<u>SQl Program</u>:

```
Select StudentID,cName from Applicant where
cName="Stanford";
```

<u>Output</u>:



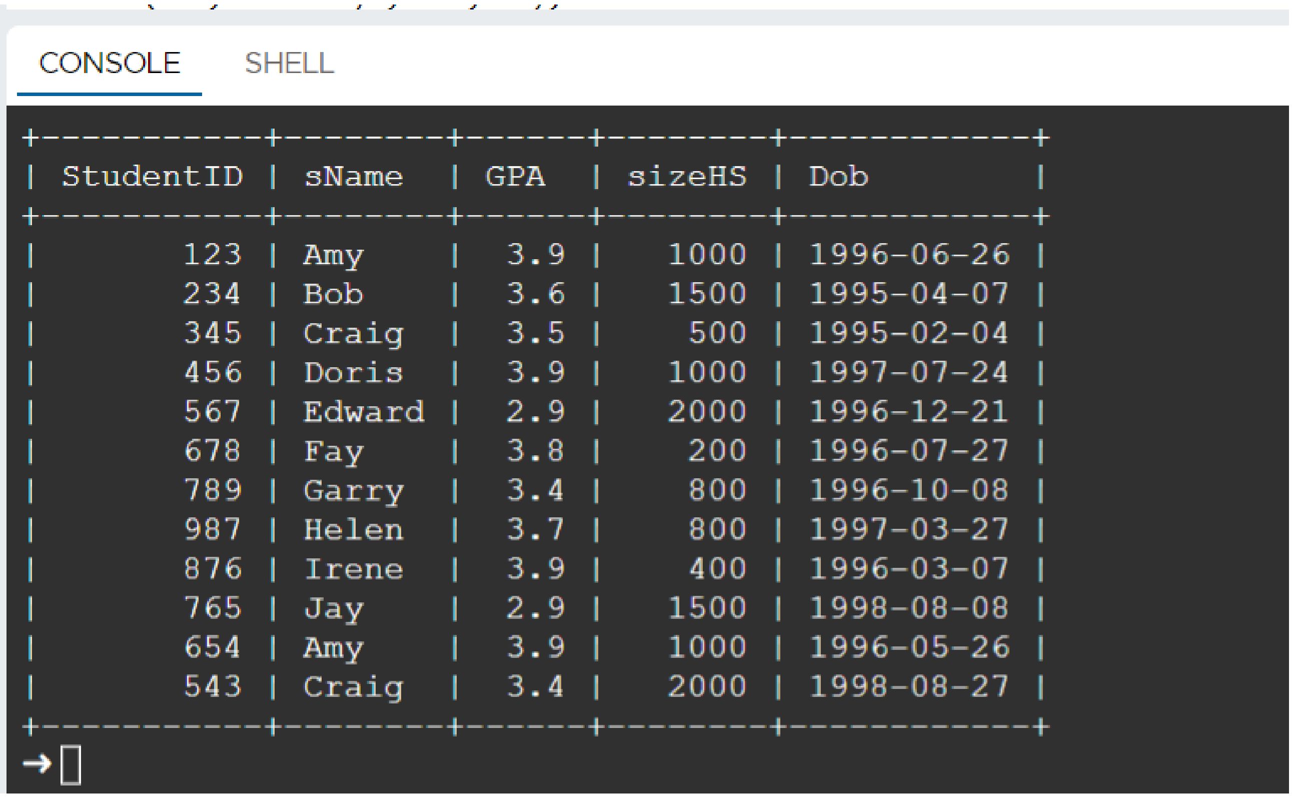
```
+-----------+----------+
| StudentID | cName    |
+-----------+----------+
|       123 | Stanford |
|       123 | Stanford |
|       678 | Stanford |
|       123 | Stanford |
|       987 | Stanford |
|       765 | Stanford |
+-----------+----------+
→
```

<u>Problem Statement-8</u>: Display the details of all students.

<u>SQl Program</u>:

```
Select * from Student;
```

<u>Output</u>:



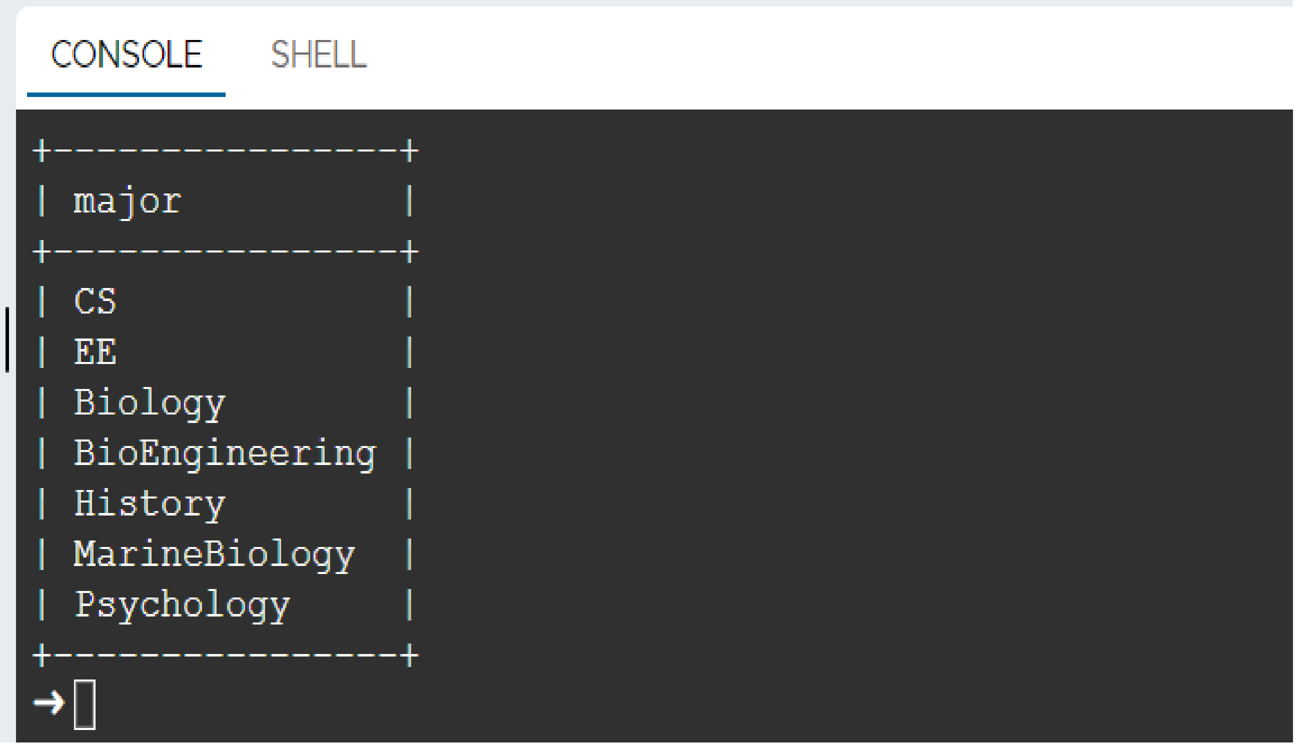
```
+-----------+--------+------+--------+------------+
| StudentID | sName  | GPA  | sizeHS | Dob        |
+-----------+--------+------+--------+------------+
|       123 | Amy    |  3.9 |   1000 | 1996-06-26 |
|       234 | Bob    |  3.6 |   1500 | 1995-04-07 |
|       345 | Craig  |  3.5 |    500 | 1995-02-04 |
|       456 | Doris  |  3.9 |   1000 | 1997-07-24 |
|       567 | Edward |  2.9 |   2000 | 1996-12-21 |
|       678 | Fay    |  3.8 |    200 | 1996-07-27 |
|       789 | Garry  |  3.4 |    800 | 1996-10-08 |
|       987 | Helen  |  3.7 |    800 | 1997-03-27 |
|       876 | Irene  |  3.9 |    400 | 1996-03-07 |
|       765 | Jay    |  2.9 |   1500 | 1998-08-08 |
|       654 | Amy    |  3.9 |   1000 | 1996-05-26 |
|       543 | Craig  |  3.4 |   2000 | 1998-08-27 |
+-----------+--------+------+--------+------------+
→
```

Problem Statement-9: Display unique majors.

SQl Program:

```
Select distinct major from Applicant;
```

Output:



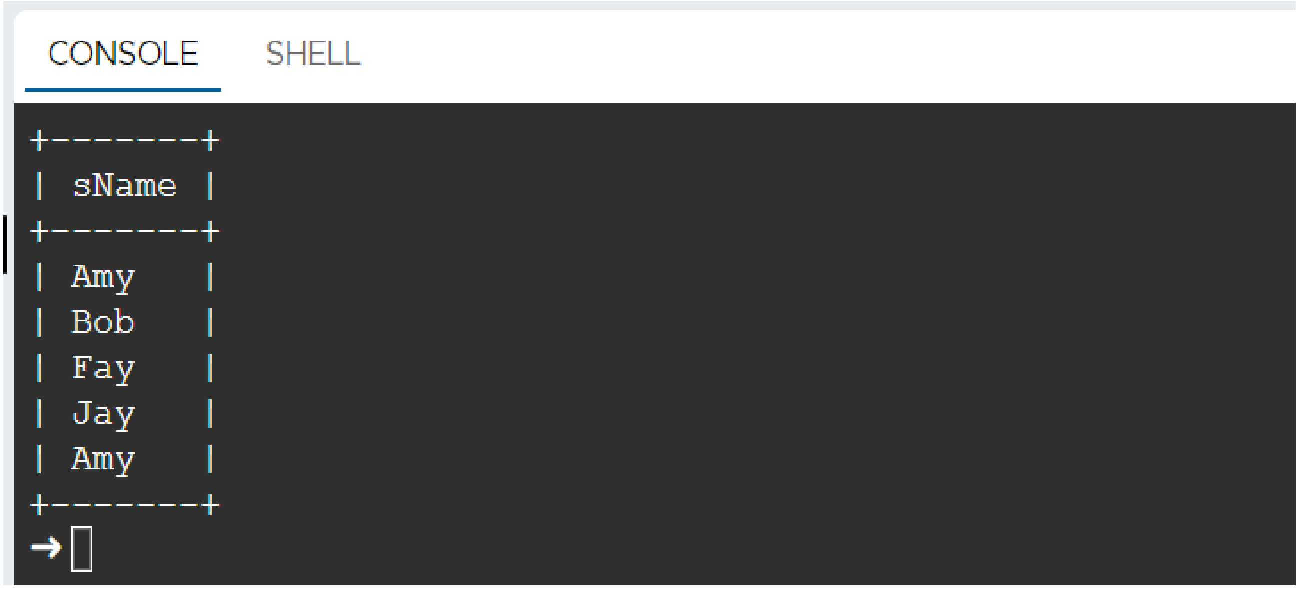
```
+----------------+
| major          |
+----------------+
| CS             |
| EE             |
| Biology        |
| BioEngineering |
| History        |
| MarineBiology  |
| Psychology     |
+----------------+
➜
```

Problem Statement-10: List the student names that have three characters in their Names.

SQl Program:

```
Select sName from Student where LENGTH(sName)=3;
```

Output:



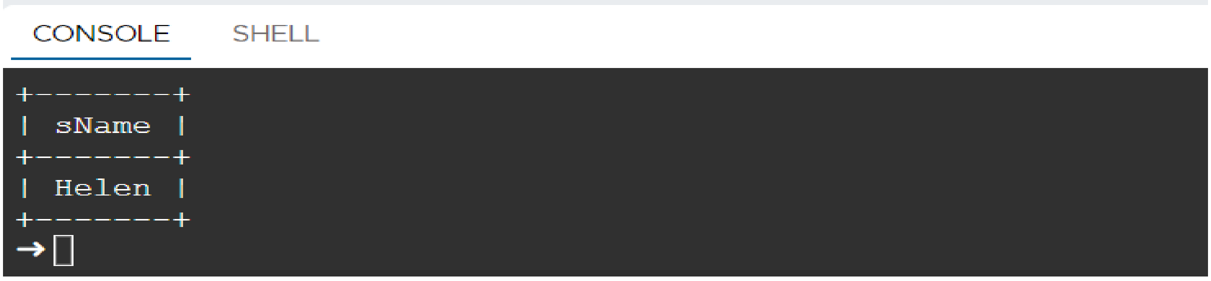
```
+-------+
| sName |
+-------+
| Amy   |
| Bob   |
| Fay   |
| Jay   |
| Amy   |
+-------+
➜
```

Problem Statement-11: List the student names that are starting with ‘H’ and with five characters.

SQl Program:

```
select sName from Student where sName LIKE 'H%' and
LENGTH(sName)=5;
```

Output:



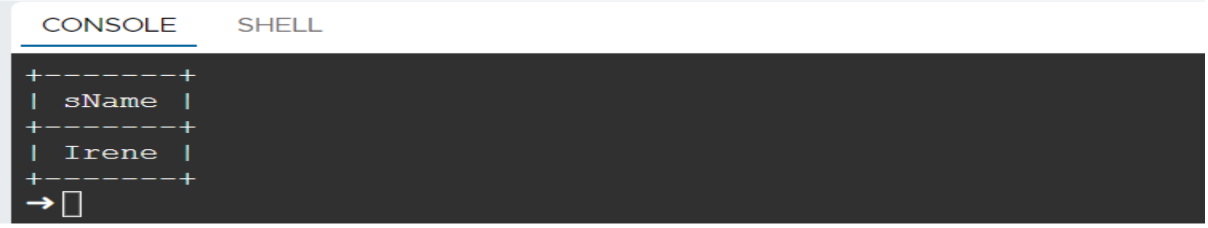
```
+--------+
| sName  |
+--------+
| Helen  |
+--------+
→▯
```

Problem Statement-12: List the student names that are having third character and fifth character as “e”.

SQl Program:

```
select sName from Student where sName LIKE '__e_e%';
```

Output:



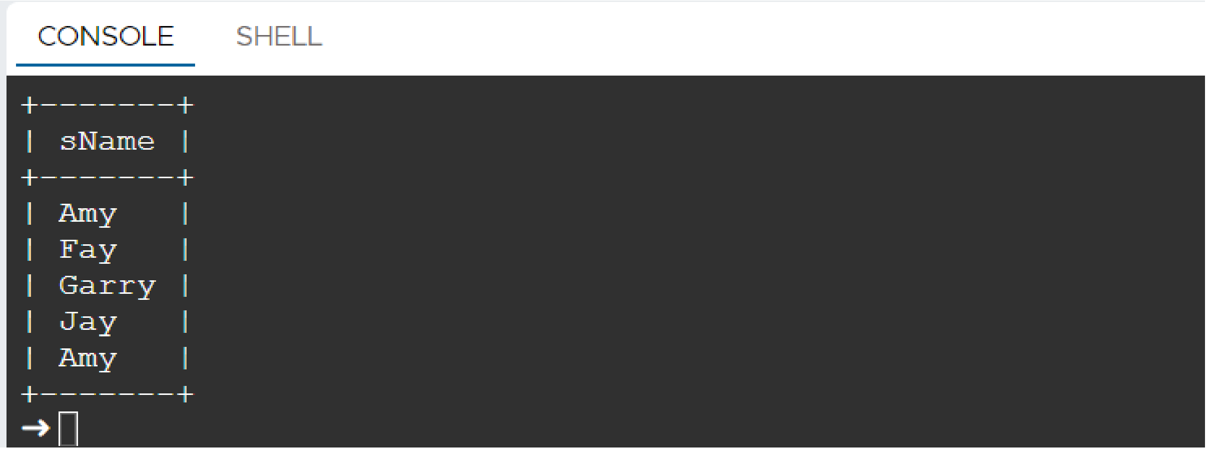
```
+--------+
| sName  |
+--------+
| Irene  |
+--------+
→▯
```

Problem Statement-13: List the student names that are ending with character “y”

SQl Program:

```
select sName from Student where sName LIKE '%y';
```

Output:



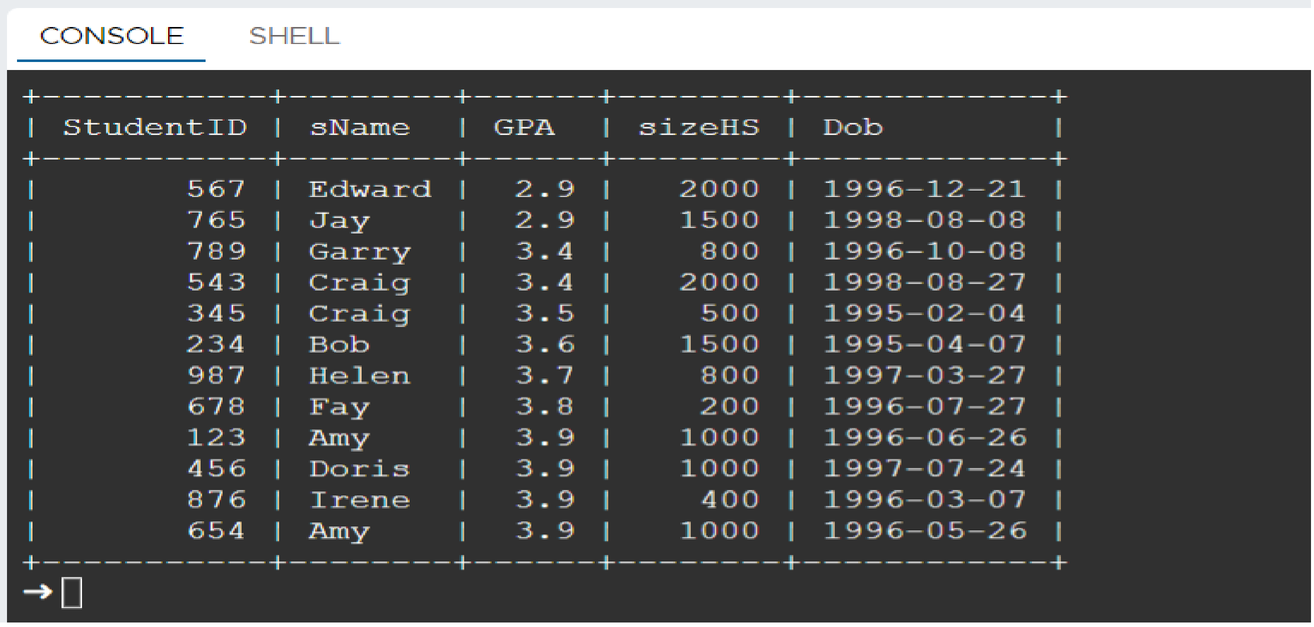
```
+--------+
| sName  |
+--------+
| Amy    |
| Fay    |
| Garry  |
| Jay    |
| Amy    |
+--------+
→▯
```

Problem Statement-14: List the students in the order of their GPA.

SQl Program:

```
select * from Student order by GPA ASC;
```

Output:



```
+-----------+--------+------+--------+------------+
| StudentID | sName  | GPA  | sizeHS | Dob        |
+-----------+--------+------+--------+------------+
|       567 | Edward |  2.9 |   2000 | 1996-12-21 |
|       765 | Jay    |  2.9 |   1500 | 1998-08-08 |
|       789 | Garry  |  3.4 |    800 | 1996-10-08 |
|       543 | Craig  |  3.4 |   2000 | 1998-08-27 |
|       345 | Craig  |  3.5 |    500 | 1995-02-04 |
|       234 | Bob    |  3.6 |   1500 | 1995-04-07 |
|       987 | Helen  |  3.7 |    800 | 1997-03-27 |
|       678 | Fay    |  3.8 |    200 | 1996-07-27 |
|       123 | Amy    |  3.9 |   1000 | 1996-06-26 |
|       456 | Doris  |  3.9 |   1000 | 1997-07-24 |
|       876 | Irene  |  3.9 |    400 | 1996-03-07 |
|       654 | Amy    |  3.9 |   1000 | 1996-05-26 |
+-----------+--------+------+--------+------------+
→
```

EXPERIMENT - 3

Objective: Understand the Data Manipulation Language(DML) Commands.

Theory:

Data manipulation is

- The retrieval of information stored in the database.
- The insertion of new information into the database.

The deletion of information from the database. The modification of information stored by the appropriate data model. There are basically two types:

(i) Procedural DML:

Require a user to specify what data is needed and how to get that data.

(ii) Non Procedural DML:

Require a user to specify what data are needed without specifying how to get those data

Updating the content of a table:

In creation situations we may wish to change a value in the table without changing all values in the tuple. For this purpose the update statement can be used.

Update table name Set columnname expression, columnname expression... Where columnname expression;

Deletion Operation: We can delete whole tuples (rows) we can delete values on only particular attributes.

Deletion of all rows

Syntax: Delete from tablename;

Deletion of specified number of rows

Syntax: Delete from table name where search_condition;

Computation in expression lists used to select data:

- Addition
- Subtraction
- Multiplication
- Exponentiation
- Division
- Enclosed Operation

Renaming columns used with Expression Lists:

The default output column names can be renamed by the user if required

Syntax:

Select columnname result_columnname result_columnname, columnname from tablename;

Logical Operators:
The logical operators that can be used in SQL sentenced are:
AND: all of must be included
OR: any of may be included
NOT: none of could be included

Range Searching: Between operations are used for range searching.

Pattern Searching: The most commonly used operation on string is pattern matching using the operation we describe patterns by using two special characters.

- Percent (%): The % character matches any substring we consider the f

Pattern Searching: The most commonly used operation on string is pattern matching using the operation 'like' we describe patterns by using two special characters.

- Percent (%): The % character matches any substring we consider the following examples.
    - Perry % matches any string beginning with perry
    - % idge % matches any string containing idge as a substring.
    - --- matches any string with exactly three characters.
    - % matches any string of at least three characters.

Ordering tuples in a particular order:

- The order by clause is used to sort the table data according to one or more columns of the table.
- The table rows are ordered in ascending order of the column values by default. The keyword used for the same is 'asc'. For sorting the table data according to col_name in descending order, keyword 'desc' is used.

Example: select col_namel, colname2,... from tablename where search condition orderby col_namel asc/desc, col_name2 asc/desc....

EXPERIMENT - 4

Objective: To implement the restrictions on the structure of the table.

Theory:

Data constraints: Besides the column name, column length and column data type. there are other parameters ie, other data constraints that can be passed by the DBA at check creation time. The constraints can either be placed at column level or at the table level.

1. Column Level Constraints:

If the constraints are defined along with the column definition, it is called a column level constraint.

2. Table Level Constraints:

If the data constraint attached to a specified column in a table references the contents of another column in the table then the user will have to use table level constraints.

List of most used Constraint:

- Not Null
- Default
- Unique
- Check
- Primary Key
- Foreign Key
    - On delete Cascade
    - On delete set Null

Null Value Concepts:-

While creating tables if a row lacks a data value for a particular column that value is said to be null. Columns of any data type may contain null values unless the column was defined as not null when the table was created.

Syntax:

Create table tablename
(cloumn_name data type(size) not null…..)

Note: Not Null constraint cannot be defined at the table level.

Primary Key:

Primary key is one or more columns is a table used to uniquely identify each row in the table. Primary key values must not be null and must be unique across the column. A multicolumn primary key is called composite primary key.

Syntax:

Create table tablename (column_name datatype (size) primary key.....)

Composite Primary key as a table constraint:
Create table tablename (column_name datatype (size), columnn_ame data type(size)...
Primary key (columnname,columnname));

Unique key concept:
A unique key is similar to a primary key except that the purpose of a unique key is to ensure that information in the column for each record is unique as with telephone or devices licence numbers. A table may have many unique keys.
Syntax: Unique as a column constraint
Create table table name (column_name datatype (size) unique);

Unique as table constraint:
Create table tablename (column_name datatype(size).column_name datatype(size)...
unique (columnname));

Default value concept:
At the time of column creation, a default value can be assigned to it. When the user is loading a record with values and leaves this column empty, the DBA will automatically load this column with the default value specified. The data type of the default value should match the data type of column.
Syntax:
Create table tablename (column_name datatype (size) default value,....);
Note: The default value constraint cannot be specified at table level.

Foreign Key Concept:
Foreign key represents the relationship between tables. A foreign key is a column whose values are derived from the primary key of the same attribute of some other table. A foreign key must have corresponding primary key value in the primary key table to have meaning.
Foreign key as a column constraint
Syntax: Create table table name (columun_name datatype(size) references another-table name);

Foreign key as a table constraint:
Syntax:
Create table name (column_name datatype(size).....
Constraint constraint_name check(expression);

Check Integrity Constraints:
Use the check constraints when you need to enforce integrity rules that can be evaluated based on a logical expression.
Following are a few examples of appropriate check constraints:

- A check constraints on the column 'name' of the Employee table so that the name is entered in upper case.

- A check constraint on the column Emp no of the Employee table so that no Emp_no value starts with 'e'.

Syntax:
Create table tablename (column_name datatype(size),. CONSTRAINT constraint_name check(expression));

Modifying the Structure of Tables:
Alter table command is used to change the structure of a table. Using the alter table clause you cannot perform the following tasks:
(i) change the name of table
(ii) decrease the size of a column if table data exists and occupies a larger size.
The following tasks you can perform through alter table command:
(i) Adding new columns:
Syntax:
ALTER TABLE tablename ADD (newcolumn_namenew_datatype (size));

(ii) Modifying existing table:
Syntax:
ALTER TABLE table_name MODIFY (new columnname new data type (size));

(iii)Deleting a column
Syntax:
ALTER TABLE tablename DROP COLUMN columnname;

Removing/Deleting Tables:
Following command is used for removing or deleting a table.
Syntax:
DROP TABLE tablename;

Defining Integrity constraints in the ALTER TABLE command:
You can also define integrity constraints using the constraint clause in the ALTER TABLE command. The following examples show the definitions of several integrity constraints:

(1) Add PRIMARY KEY
Syntax: ALTER TABLE tablename ADD PRIMARY KEY(columnname);

(2) Add FOREIGN KEY
Syntax: ALTER TABLE table_name ADD CONSTRAINT constrain_tname FOREIGN KEY (columnname) REFERENCES table_name;

(3) Add CHECK CONSTRAINT
Syntax: CONSTRAINT ALTER TABLE table_name ADD CONSTRAINT constraint_name Check(expression);

Dropping integrity constraints in the ALTER TABLE command:

You can drop an integrity constraint if the rule that it enforces is no longer true or if the constraint is no longer needed. Drop the constraint using the ALTER TABLE command with the DROP clause.

The following examples illustrate the dropping of integrity constraints:

(1) DROP PRIMARY KEY

Syntax: ALTER TABLE table_name DROP PRIMARY KEY;

(2) DROP FOREIGN KEY

Syntax: ALTER TABLE table_name DROP FOREIGN KEY;

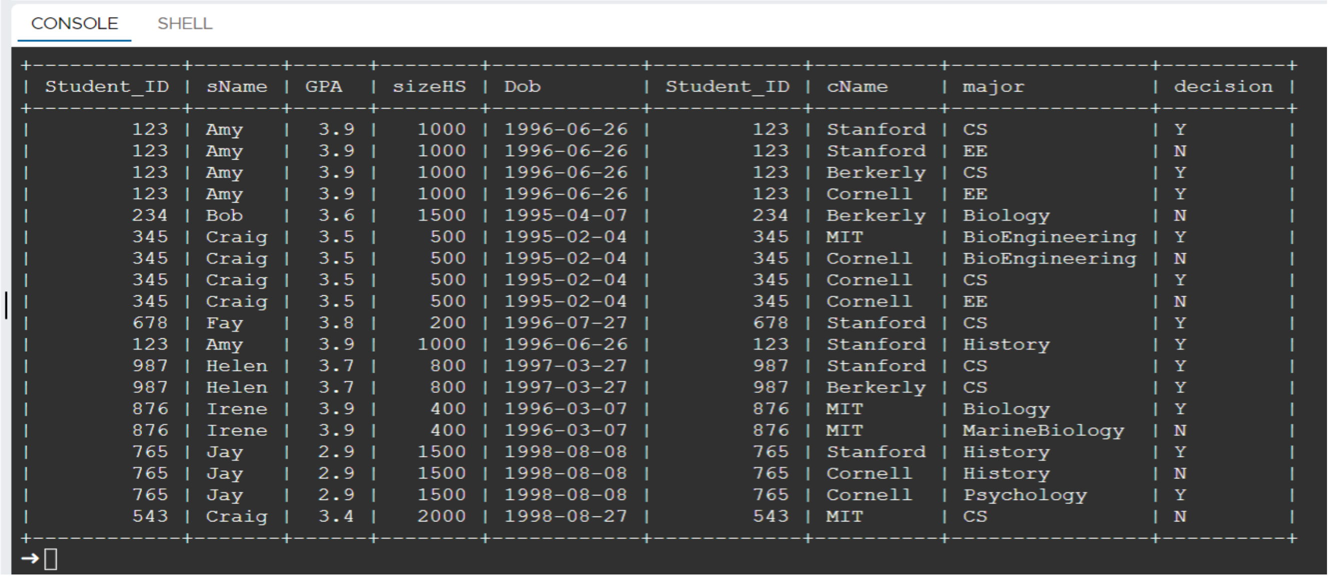
Exercise-4:

Problem Statement-1: Produce a combined table in which each student is combined with every other application.

SQl Program:

```
Select * from Student Inner Join Applicant on
Student.Student_ID = Applicant.Student_ID;
```

Output:



```
+------------+-------+------+--------+------------+------------+----------+----------------+----------+
| Student_ID | sName | GPA  | sizeHS | Dob        | Student_ID | cName    | major          | decision |
+------------+-------+------+--------+------------+------------+----------+----------------+----------+
|        123 | Amy   |  3.9 |   1000 | 1996-06-26 |        123 | Stanford | CS             | Y        |
|        123 | Amy   |  3.9 |   1000 | 1996-06-26 |        123 | Stanford | EE             | N        |
|        123 | Amy   |  3.9 |   1000 | 1996-06-26 |        123 | Berkerly | CS             | Y        |
|        123 | Amy   |  3.9 |   1000 | 1996-06-26 |        123 | Cornell  | EE             | Y        |
|        234 | Bob   |  3.6 |   1500 | 1995-04-07 |        234 | Berkerly | Biology        | N        |
|        345 | Craig |  3.5 |    500 | 1995-02-04 |        345 | MIT      | BioEngineering | Y        |
|        345 | Craig |  3.5 |    500 | 1995-02-04 |        345 | Cornell  | BioEngineering | N        |
|        345 | Craig |  3.5 |    500 | 1995-02-04 |        345 | Cornell  | CS             | Y        |
|        345 | Craig |  3.5 |    500 | 1995-02-04 |        345 | Cornell  | EE             | N        |
|        678 | Fay   |  3.8 |    200 | 1996-07-27 |        678 | Stanford | CS             | Y        |
|        123 | Amy   |  3.9 |   1000 | 1996-06-26 |        123 | Stanford | History        | Y        |
|        987 | Helen |  3.7 |    800 | 1997-03-27 |        987 | Stanford | CS             | Y        |
|        987 | Helen |  3.7 |    800 | 1997-03-27 |        987 | Berkerly | CS             | Y        |
|        876 | Irene |  3.9 |    400 | 1996-03-07 |        876 | MIT      | Biology        | Y        |
|        876 | Irene |  3.9 |    400 | 1996-03-07 |        876 | MIT      | MarineBiology  | N        |
|        765 | Jay   |  2.9 |   1500 | 1998-08-08 |        765 | Stanford | History        | Y        |
|        765 | Jay   |  2.9 |   1500 | 1998-08-08 |        765 | Cornell  | History        | N        |
|        765 | Jay   |  2.9 |   1500 | 1998-08-08 |        765 | Cornell  | Psychology     | Y        |
|        543 | Craig |  3.4 |   2000 | 1998-08-27 |        543 | MIT      | CS             | N        |
+------------+-------+------+--------+------------+------------+----------+----------------+----------+
→
```

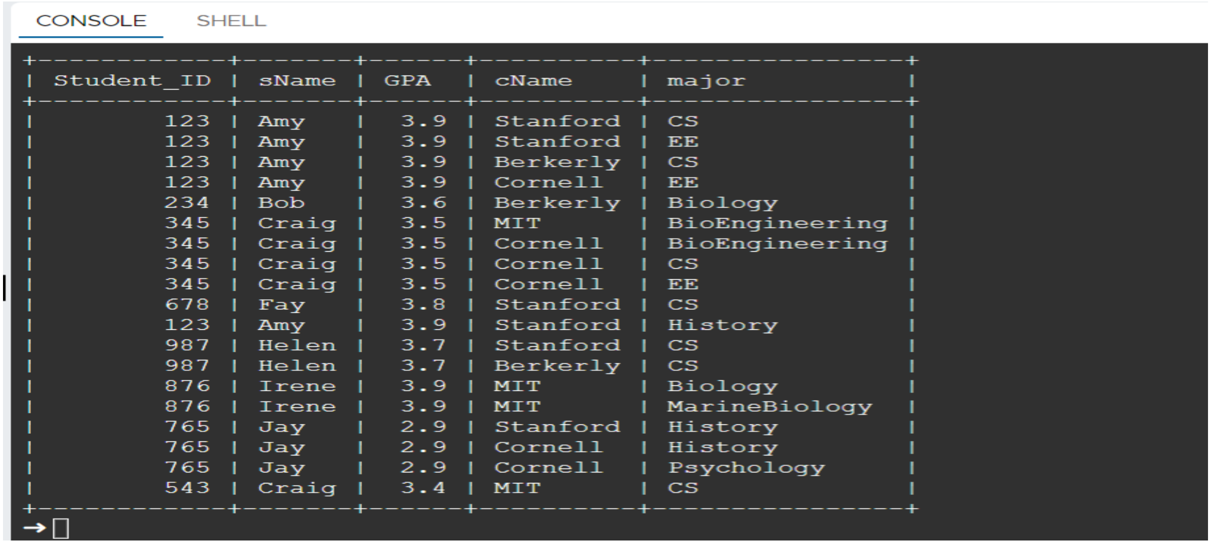
Exercise-4:

Problem Statement-2: Give Student ID, name, GPA and name of college and major each student applied to.

SQl Program:

```
Select    Student.Student_ID,Student.sName,Student.GPA,
Applicant.cName,Applicant.major from Student Inner Join
Applicant on Student.Student_ID = Applicant.Student_ID;
```

Output:



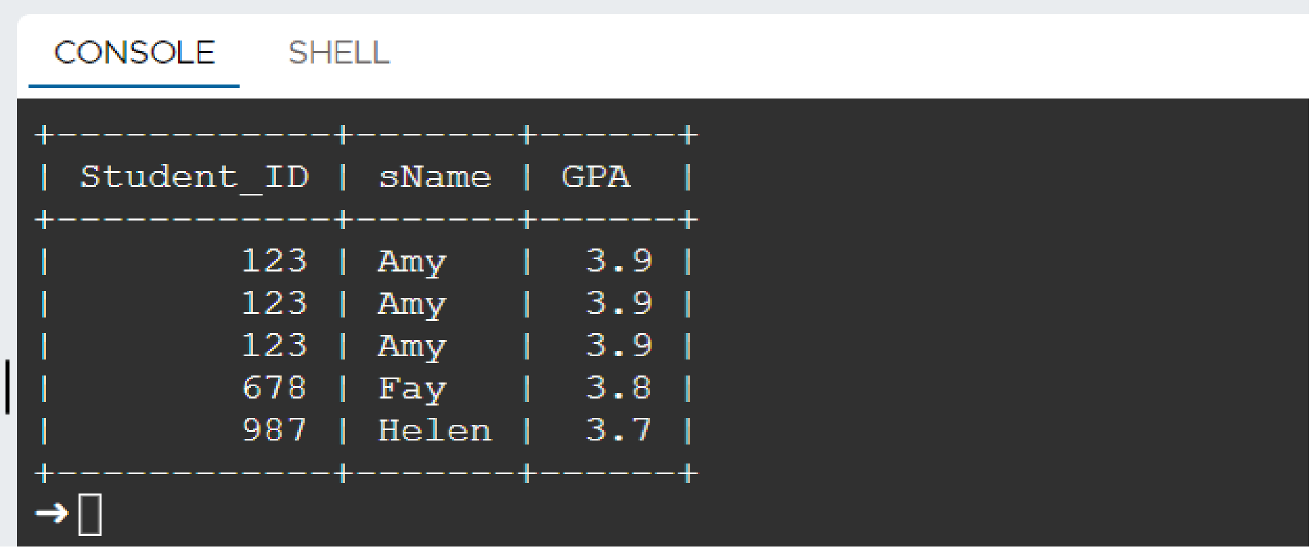
```
+------------+-------+------+----------+----------------+
| Student_ID | sName | GPA  | cName    | major          |
+------------+-------+------+----------+----------------+
|        123 | Amy   |  3.9 | Stanford | CS             |
|        123 | Amy   |  3.9 | Stanford | EE             |
|        123 | Amy   |  3.9 | Berkerly | CS             |
|        123 | Amy   |  3.9 | Cornell  | EE             |
|        234 | Bob   |  3.6 | Berkerly | Biology        |
|        345 | Craig |  3.5 | MIT      | BioEngineering |
|        345 | Craig |  3.5 | Cornell  | BioEngineering |
|        345 | Craig |  3.5 | Cornell  | CS             |
|        345 | Craig |  3.5 | Cornell  | EE             |
|        678 | Fay   |  3.8 | Stanford | CS             |
|        123 | Amy   |  3.9 | Stanford | History        |
|        987 | Helen |  3.7 | Stanford | CS             |
|        987 | Helen |  3.7 | Berkerly | CS             |
|        876 | Irene |  3.9 | MIT      | Biology        |
|        876 | Irene |  3.9 | MIT      | MarineBiology  |
|        765 | Jay   |  2.9 | Stanford | History        |
|        765 | Jay   |  2.9 | Cornell  | History        |
|        765 | Jay   |  2.9 | Cornell  | Psychology     |
|        543 | Craig |  3.4 | MIT      | CS             |
+------------+-------+------+----------+----------------+
→
```

Problem Statement-3: Find IDs, name, GPA of students and name of college with GPA > 3.7 applying to Stanford.

SQl Program:

```
Select Student.Student_ID,Student.sName,Student.GPA
from Student Inner Join Applicant on
Student.Student_ID = Applicant.Student_ID where
Student.GPA>3.7 and Applicant.cName="Stanford";
```

Output:



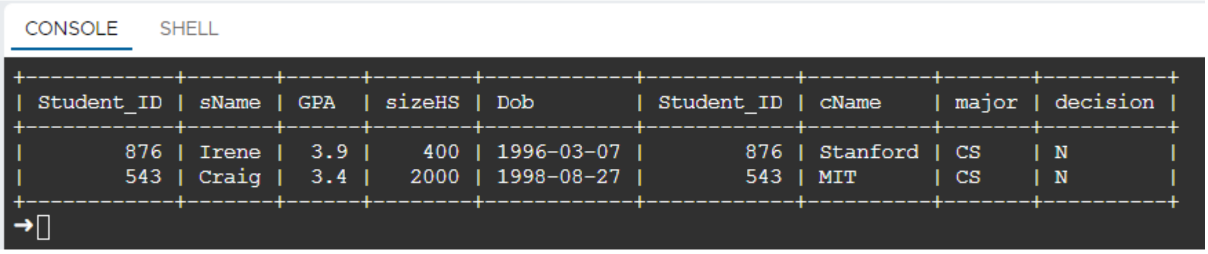
```
+------------+-------+-------+
| Student_ID | sName | GPA   |
+------------+-------+-------+
|        123 | Amy   |   3.9 |
|        123 | Amy   |   3.9 |
|        123 | Amy   |   3.9 |
|        678 | Fay   |   3.8 |
|        987 | Helen |   3.7 |
+------------+-------+-------+
➔
```

Problem Statement-4: Find details of Students who apply to CS major and their applications are rejected.

SQl Program:

```
Select * from Student Inner Join Applicant on
Student.Student_ID = Applicant.Student_ID where
Applicant.major="CS" and Applicant.decision="N";
```

Output:



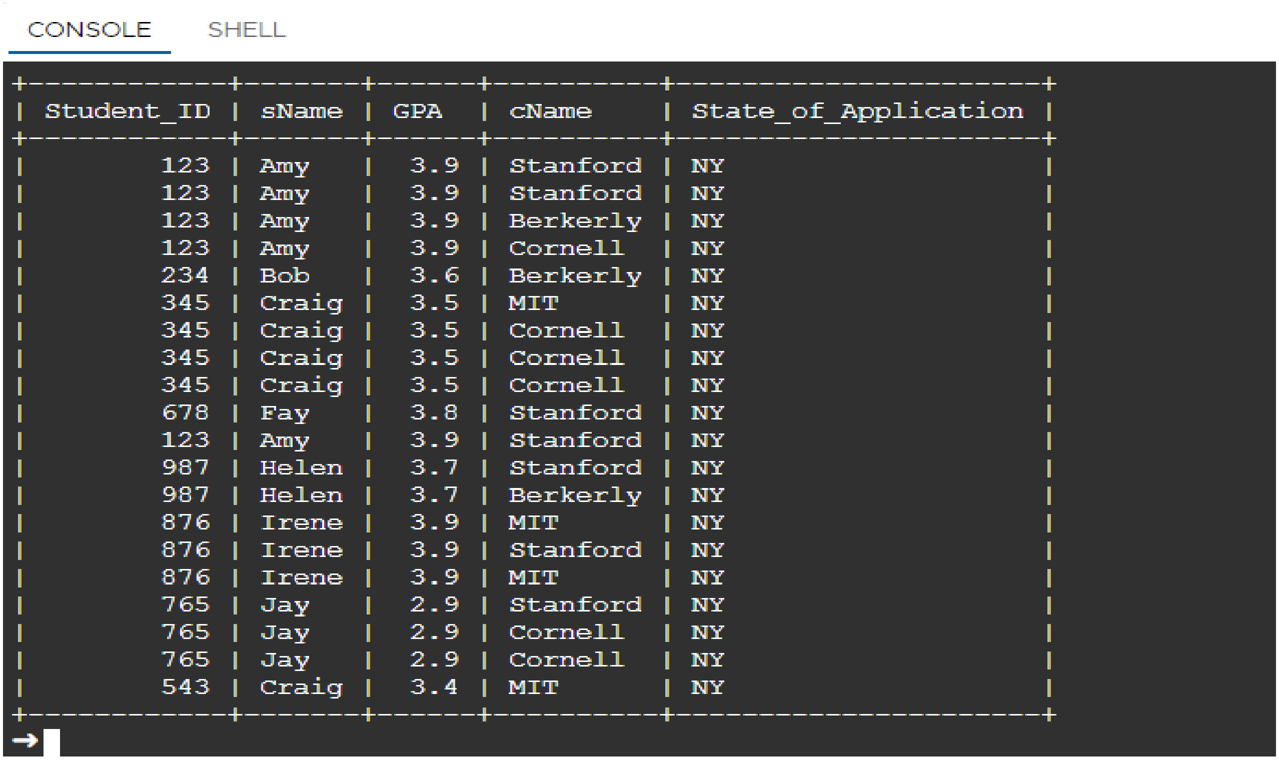
```
+------------+-------+-------+--------+------------+------------+----------+-------+----------+
| Student_ID | sName | GPA   | sizeHS | Dob        | Student_ID | cName    | major | decision |
+------------+-------+-------+--------+------------+------------+----------+-------+----------+
|        876 | Irene |   3.9 |    400 | 1996-03-07 |        876 | Stanford | CS    | N        |
|        543 | Craig |   3.4 |   2000 | 1998-08-27 |        543 | MIT      | CS    | N        |
+------------+-------+-------+--------+------------+------------+----------+-------+----------+
➔
```

Problem Statement-5: Find details of student and application who applied to colleges at New York.

SQl Program:

Select Student.Student_ID,Student.sName,Student.GPA,Applicant.cName, College.State_of_Application from Student Inner Join Applicant Inner Join College on Student.Student_ID = Applicant.Student_ID where College.State_of_Application="NY";

Output:



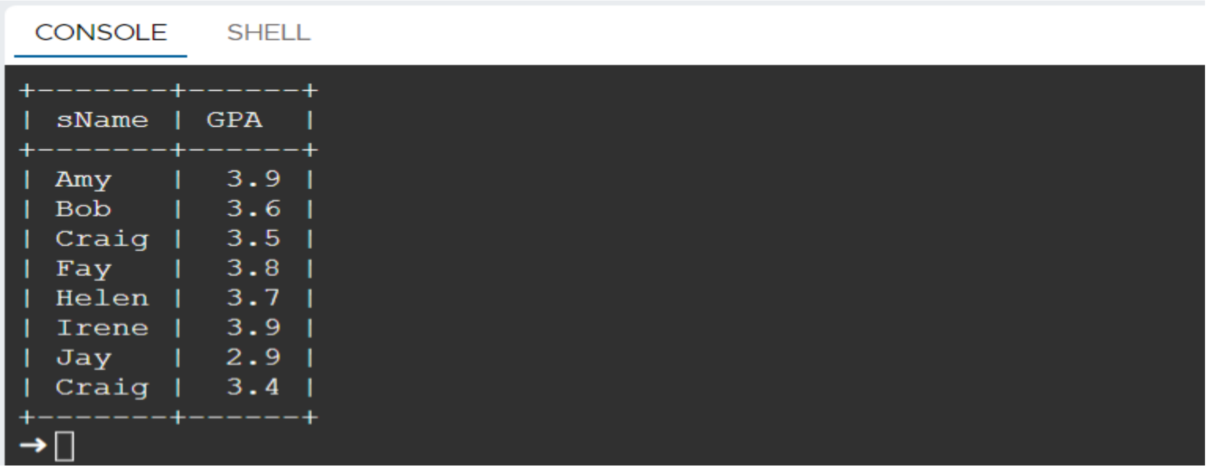
```
+------------+-------+------+-----------+----------------------+
| Student_ID | sName | GPA  | cName     | State_of_Application |
+------------+-------+------+-----------+----------------------+
|        123 | Amy   |  3.9 | Stanford  | NY                   |
|        123 | Amy   |  3.9 | Stanford  | NY                   |
|        123 | Amy   |  3.9 | Berkerly  | NY                   |
|        123 | Amy   |  3.9 | Cornell   | NY                   |
|        234 | Bob   |  3.6 | Berkerly  | NY                   |
|        345 | Craig |  3.5 | MIT       | NY                   |
|        345 | Craig |  3.5 | Cornell   | NY                   |
|        345 | Craig |  3.5 | Cornell   | NY                   |
|        345 | Craig |  3.5 | Cornell   | NY                   |
|        678 | Fay   |  3.8 | Stanford  | NY                   |
|        123 | Amy   |  3.9 | Stanford  | NY                   |
|        987 | Helen |  3.7 | Stanford  | NY                   |
|        987 | Helen |  3.7 | Berkerly  | NY                   |
|        876 | Irene |  3.9 | MIT       | NY                   |
|        876 | Irene |  3.9 | Stanford  | NY                   |
|        876 | Irene |  3.9 | MIT       | NY                   |
|        765 | Jay   |  2.9 | Stanford  | NY                   |
|        765 | Jay   |  2.9 | Cornell   | NY                   |
|        765 | Jay   |  2.9 | Cornell   | NY                   |
|        543 | Craig |  3.4 | MIT       | NY                   |
+------------+-------+------+-----------+----------------------+
→
```

Problem Statement-6: Find name and GPA of applicants who apply to any college whose enrollment is not more than 25000.

SQl Program:

Select distinct Student.sName,Student.GPA from Student Inner Join Applicant Inner Join College on Student.Student_ID = Applicant.Student_ID where College.enrollment<25000;

Output:



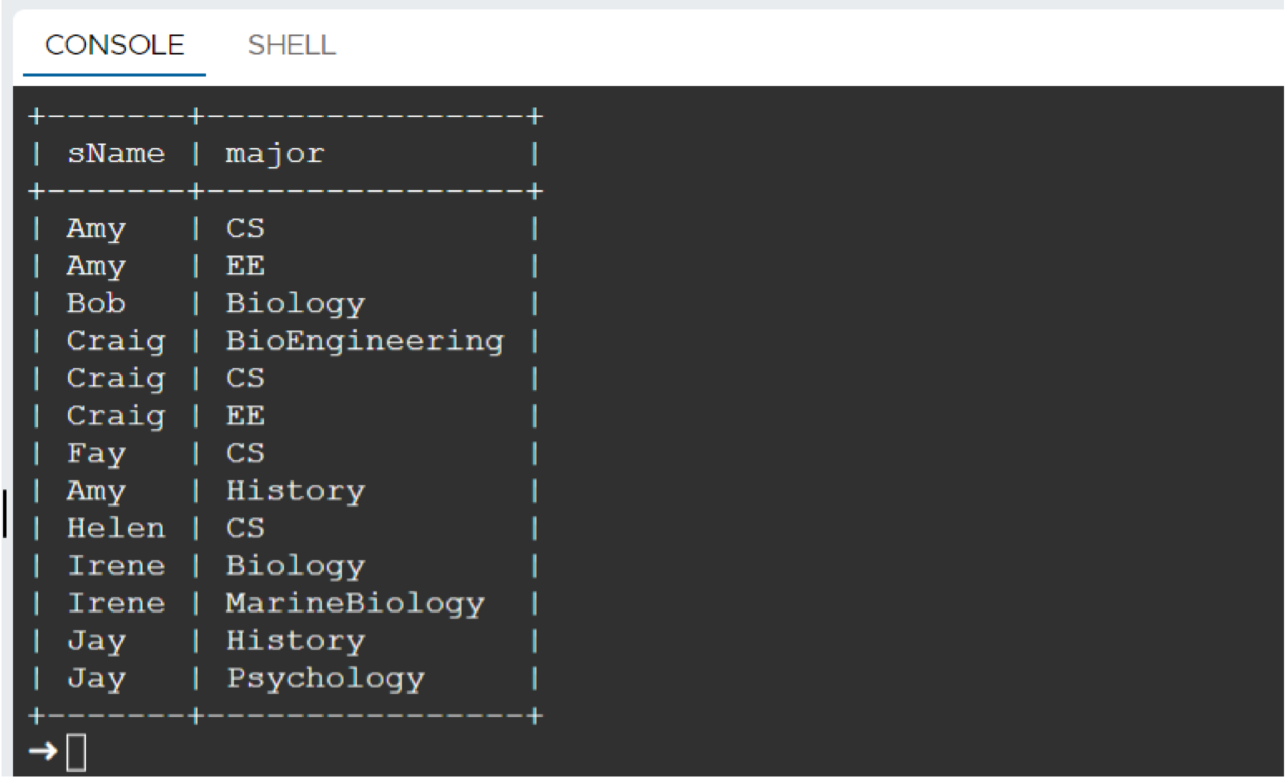
```
+-------+------+
| sName | GPA  |
+-------+------+
| Amy   |  3.9 |
| Bob   |  3.6 |
| Craig |  3.5 |
| Fay   |  3.8 |
| Helen |  3.7 |
| Irene |  3.9 |
| Jay   |  2.9 |
| Craig |  3.4 |
+-------+------+
→
```

Problem Statement-7: Find Student and major he/she applied to.

SQl Program:

Select distinct Student.sName,Applicant.major from Student Inner Join Applicant on Student.Student_ID = Applicant.Student_ID;

Output:



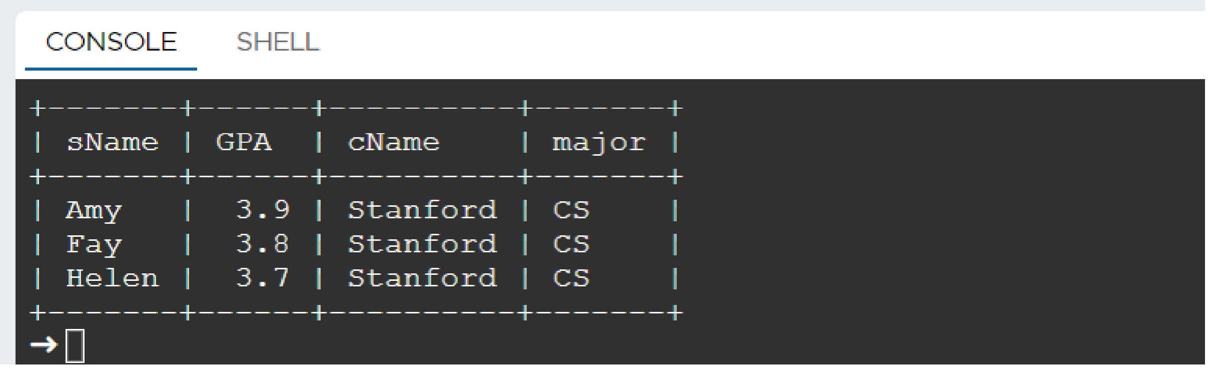
```
+-------+----------------+
| sName | major          |
+-------+----------------+
| Amy   | CS             |
| Amy   | EE             |
| Bob   | Biology        |
| Craig | BioEngineering |
| Craig | CS             |
| Craig | EE             |
| Fay   | CS             |
| Amy   | History        |
| Helen | CS             |
| Irene | Biology        |
| Irene | MarineBiology  |
| Jay   | History        |
| Jay   | Psychology     |
+-------+----------------+
→
```

Problem Statement-8: Find details of students who came from high school having size less than 20000 and applied to CS at Stanford.

SQl Program:

Select Student.sName,Student.GPA,Applicant.cName,Applicant.major from Student Inner Join Applicant on Student.Student_ID = Applicant.Student_ID where Student.sizeHS<20000 and Applicant.cName="Stanford" and Applicant.major="CS";

Output:



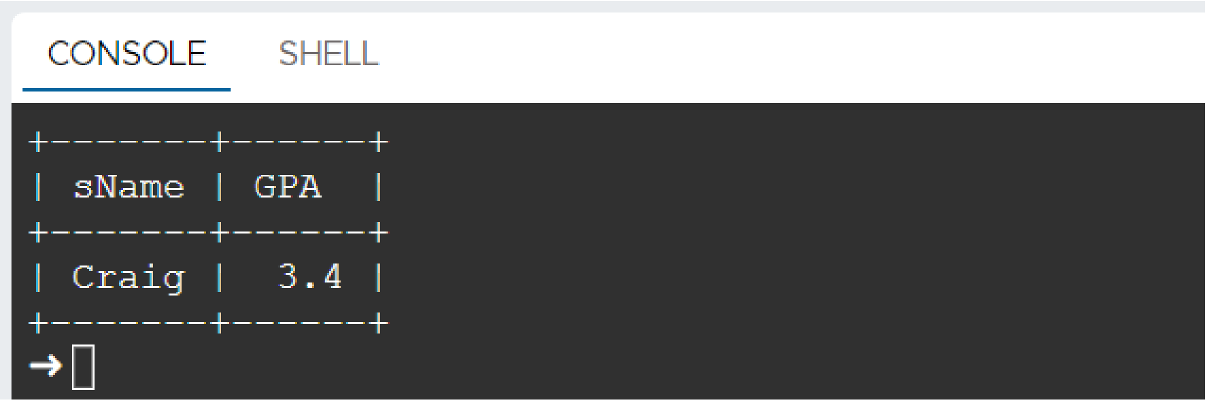
```
+-------+------+----------+-------+
| sName | GPA  | cName    | major |
+-------+------+----------+-------+
| Amy   |  3.9 | Stanford | CS    |
| Fay   |  3.8 | Stanford | CS    |
| Helen |  3.7 | Stanford | CS    |
+-------+------+----------+-------+
→
```

<u>Problem Statement-9</u>: Names and GPAs of Students with HS>1000 who applied to CS and were rejected.

<u>SQl Program</u>:

Select Student.sName,Student.GPA from Student Inner Join Applicant on Student.Student_ID = Applicant.Student_ID where Student.sizeHS>1000 and Applicant.decision="N" and Applicant.major="CS";

<u>Output</u>:



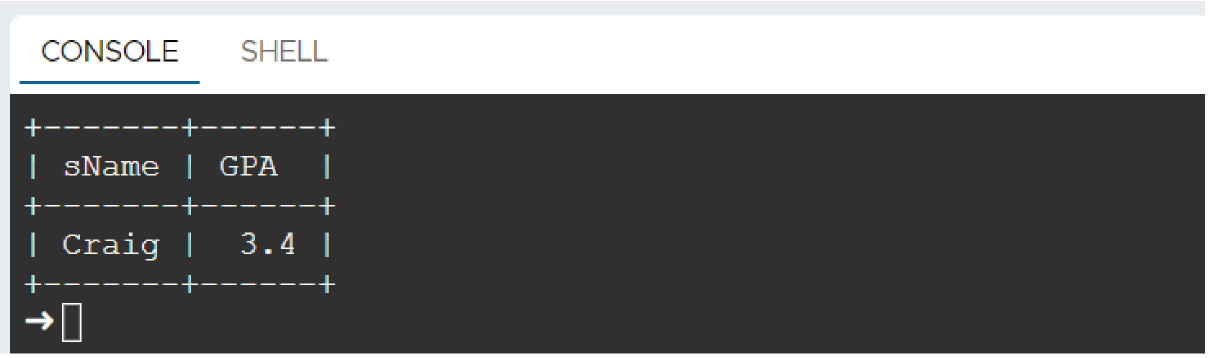
```
+-------+------+
| sName | GPA  |
+-------+------+
| Craig |  3.4 |
+-------+------+
→
```

<u>Problem Statement-10</u>: Names and GPAs of Students with HS>1000 who applied to CS at college with enrollment>20000 and were rejected.

<u>SQl Program</u>:

Select distinct Student.sName,Student.GPA from Student Inner Join Applicant Inner Join College on Student.Student_ID = Applicant.Student_ID where Student.sizeHS>1000 and Applicant.decision="N" and Applicant.major="CS" and College.enrollment>20000;

<u>Output</u>:



```
+-------+------+
| sName | GPA  |
+-------+------+
| Craig |  3.4 |
+-------+------+
→
```

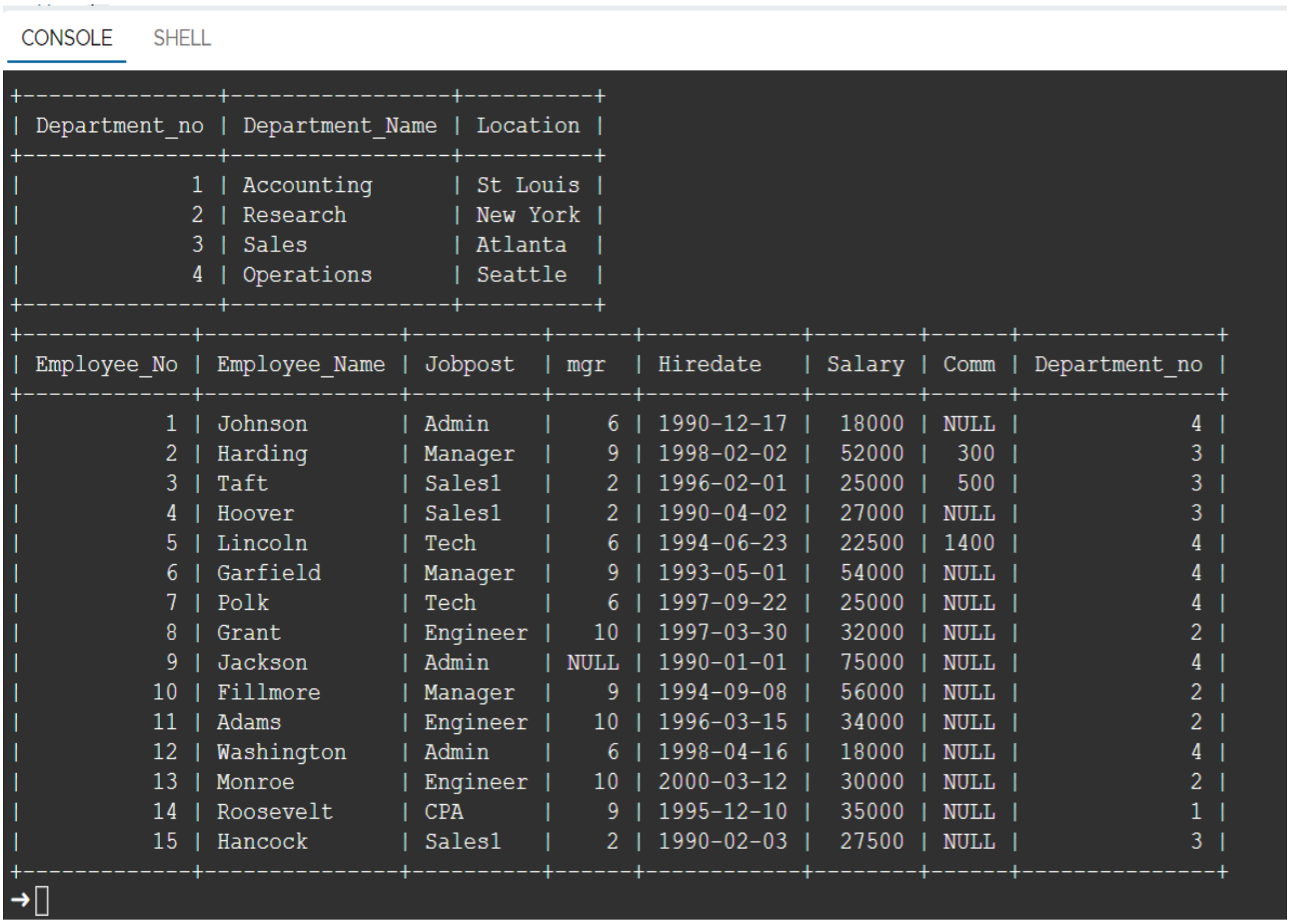
Exercise-5:

Problem Statement-1: Create the “Department”, “Employee” tables in the “Company” database and insert the provided data in them.

SQl Program:

```
create Database Company;
use Company;
CREATE TABLE Department (
    Department_no int,
    Department_Name varchar(25),
    Location varchar(20)
);
CREATE TABLE Employee (
    Employee_No int,
    Employee_Name varchar(20),
    Jobpost varchar(20),
    mgr int,
    Hiredate date,
    Salary int,
    Comm int,
    Department_no int
);
INSERT INTO Department
VALUES (1,"Accounting","St Louis"),
(2,"Research","New York"),
(3,"Sales","Atlanta"),
(4,"Operations","Seattle")
;
Insert Into Employee
VALUES (1,"Johnson","Admin",6,"1990-12-17",18000,null,4),
(2,"Harding","Manager",9,"1998-02-02",52000,300,3),
(3,"Taft","Sales1",2,"1996-02-01",25000,500,3),
(4,"Hoover","Sales1",2,"1990-04-02",27000,null,3),
(5,"Lincoln","Tech",6,"1994-06-23",22500,1400,4),
(6,"Garfield","Manager",9,"1993-05-01",54000,null,4),
(7,"Polk","Tech",6,"1997-09-22",25000,null,4),
(8,"Grant","Engineer",10,"1997-03-30",32000,null,2),
(9,"Jackson","Admin",null,"1990-01-01",75000,null,4),
(10,"Fillmore","Manager",9,"1994-09-08",56000,null,2),
(11,"Adams","Engineer",10,"1996-03-15",34000,null,2),
(12,"Washington","Admin",6,"1998-04-16",18000,null,4),
(13,"Monroe","Engineer",10,"2000-03-12",30000,null,2),
(14,"Roosevelt","CPA",9,"1995-12-10",35000,null,1),
(15,"Hancock","Sales1",2,"1990-02-03",27500,null,3)
;
Select * from Department;
Select * from Employee;
```

Output:



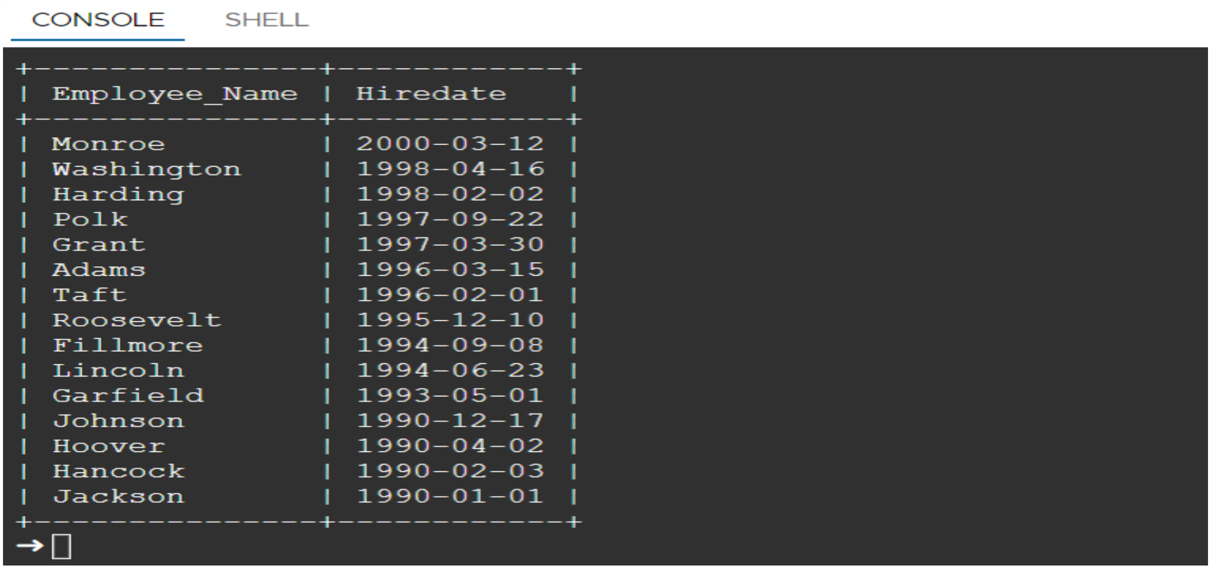
```
+---------------+-----------------+----------+
| Department_no | Department_Name | Location |
+---------------+-----------------+----------+
|             1 | Accounting      | St Louis |
|             2 | Research        | New York |
|             3 | Sales           | Atlanta  |
|             4 | Operations      | Seattle  |
+---------------+-----------------+----------+
+-------------+---------------+----------+------+------------+--------+------+---------------+
| Employee_No | Employee_Name | Jobpost  | mgr  | Hiredate   | Salary | Comm | Department_no |
+-------------+---------------+----------+------+------------+--------+------+---------------+
|           1 | Johnson       | Admin    |    6 | 1990-12-17 |  18000 | NULL |             4 |
|           2 | Harding       | Manager  |    9 | 1998-02-02 |  52000 |  300 |             3 |
|           3 | Taft          | Sales1   |    2 | 1996-02-01 |  25000 |  500 |             3 |
|           4 | Hoover        | Sales1   |    2 | 1990-04-02 |  27000 | NULL |             3 |
|           5 | Lincoln       | Tech     |    6 | 1994-06-23 |  22500 | 1400 |             4 |
|           6 | Garfield      | Manager  |    9 | 1993-05-01 |  54000 | NULL |             4 |
|           7 | Polk          | Tech     |    6 | 1997-09-22 |  25000 | NULL |             4 |
|           8 | Grant         | Engineer |   10 | 1997-03-30 |  32000 | NULL |             2 |
|           9 | Jackson       | Admin    | NULL | 1990-01-01 |  75000 | NULL |             4 |
|          10 | Fillmore      | Manager  |    9 | 1994-09-08 |  56000 | NULL |             2 |
|          11 | Adams         | Engineer |   10 | 1996-03-15 |  34000 | NULL |             2 |
|          12 | Washington    | Admin    |    6 | 1998-04-16 |  18000 | NULL |             4 |
|          13 | Monroe        | Engineer |   10 | 2000-03-12 |  30000 | NULL |             2 |
|          14 | Roosevelt     | CPA      |    9 | 1995-12-10 |  35000 | NULL |             1 |
|          15 | Hancock       | Sales1   |    2 | 1990-02-03 |  27500 | NULL |             3 |
+-------------+---------------+----------+------+------------+--------+------+---------------+
→
```

Problem Statement-2: Employee Name and Hire Date Sorted by Hire Date(Recent to old).

SQl Program:

```
select Employee_Name,Hiredate from Employee order by
Hiredate Desc;
```

Output:



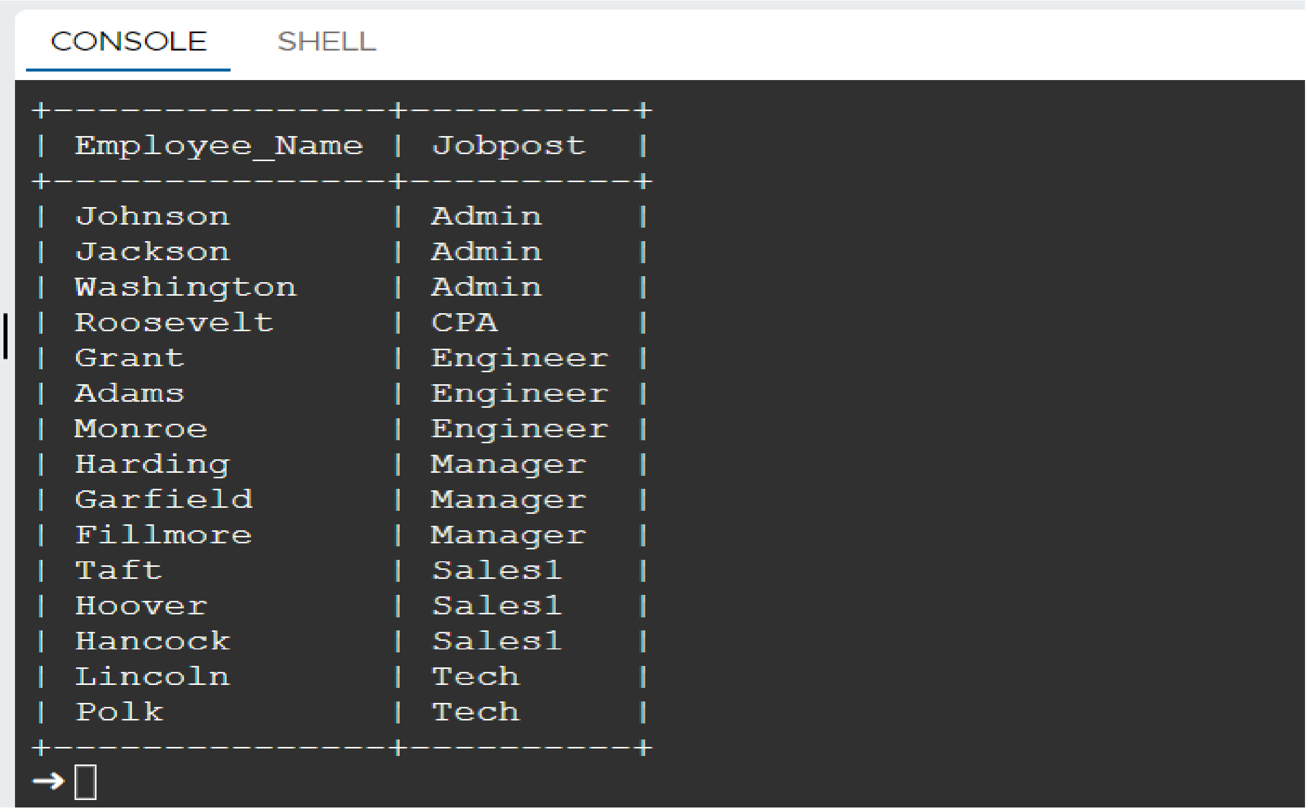
```
+---------------+------------+
| Employee_Name | Hiredate   |
+---------------+------------+
| Monroe        | 2000-03-12 |
| Washington    | 1998-04-16 |
| Harding       | 1998-02-02 |
| Polk          | 1997-09-22 |
| Grant         | 1997-03-30 |
| Adams         | 1996-03-15 |
| Taft          | 1996-02-01 |
| Roosevelt     | 1995-12-10 |
| Fillmore      | 1994-09-08 |
| Lincoln       | 1994-06-23 |
| Garfield      | 1993-05-01 |
| Johnson       | 1990-12-17 |
| Hoover        | 1990-04-02 |
| Hancock       | 1990-02-03 |
| Jackson       | 1990-01-01 |
+---------------+------------+
→
```

Problem Statement-3: Employee Name and Jobpost Sorted by Job(Alphabetically).

SQl Program:

```
SELECT Employee_Name,Jobpost from Employee order by
Jobpost Asc;
```

Output:



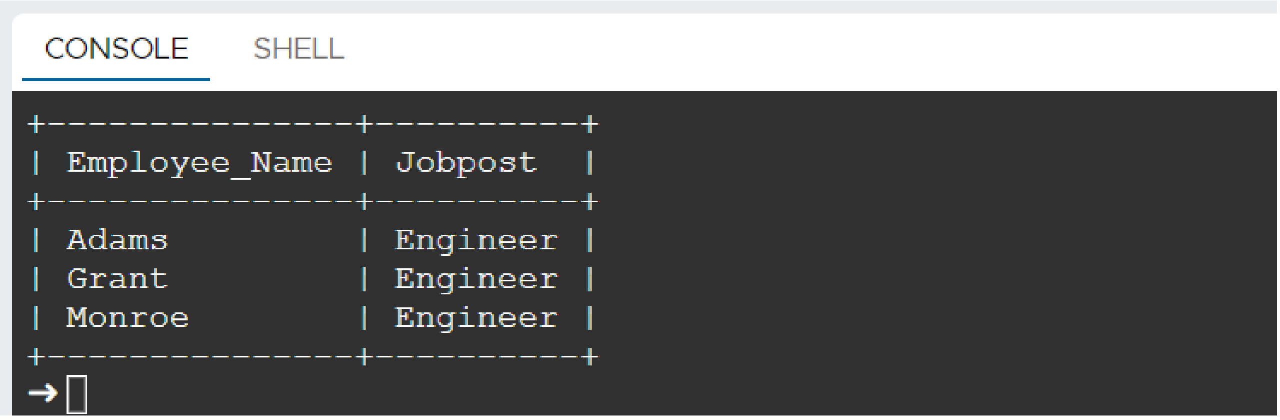
```
+----------------+-----------+
| Employee_Name  | Jobpost   |
+----------------+-----------+
| Johnson        | Admin     |
| Jackson        | Admin     |
| Washington     | Admin     |
| Roosevelt      | CPA       |
| Grant          | Engineer  |
| Adams          | Engineer  |
| Monroe         | Engineer  |
| Harding        | Manager   |
| Garfield       | Manager   |
| Fillmore       | Manager   |
| Taft           | Sales1    |
| Hoover         | Sales1    |
| Hancock        | Sales1    |
| Lincoln        | Tech      |
| Polk           | Tech      |
+----------------+-----------+
➜
```

Problem Statement-4: Employee Name and Jobpost for all Engineers, Sorted by Employee Name(Alphabetically).

SQl Program:

```
SELECT Employee_Name,Jobpost from Employee where
Jobpost="Engineer" order by Employee_Name Asc;
```

Output:



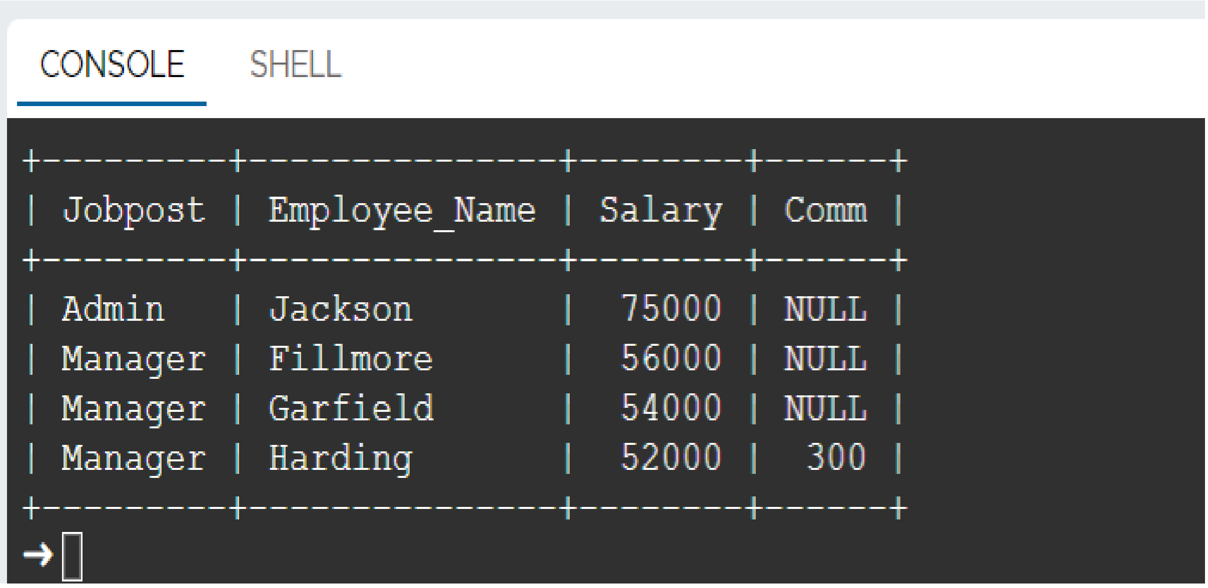
```
+----------------+-----------+
| Employee_Name  | Jobpost   |
+----------------+-----------+
| Adams          | Engineer  |
| Grant          | Engineer  |
| Monroe         | Engineer  |
+----------------+-----------+
➜
```

Problem Statement-5: Job, Employee Name, Salary and Commission for employees with salary over 50000 sorted by Salary (Largest to Smallest).

SQL Program:

SELECT Jobpost,Employee_Name,Salary,Comm from Employee where Salary>50000 order by Salary Desc;

Output:



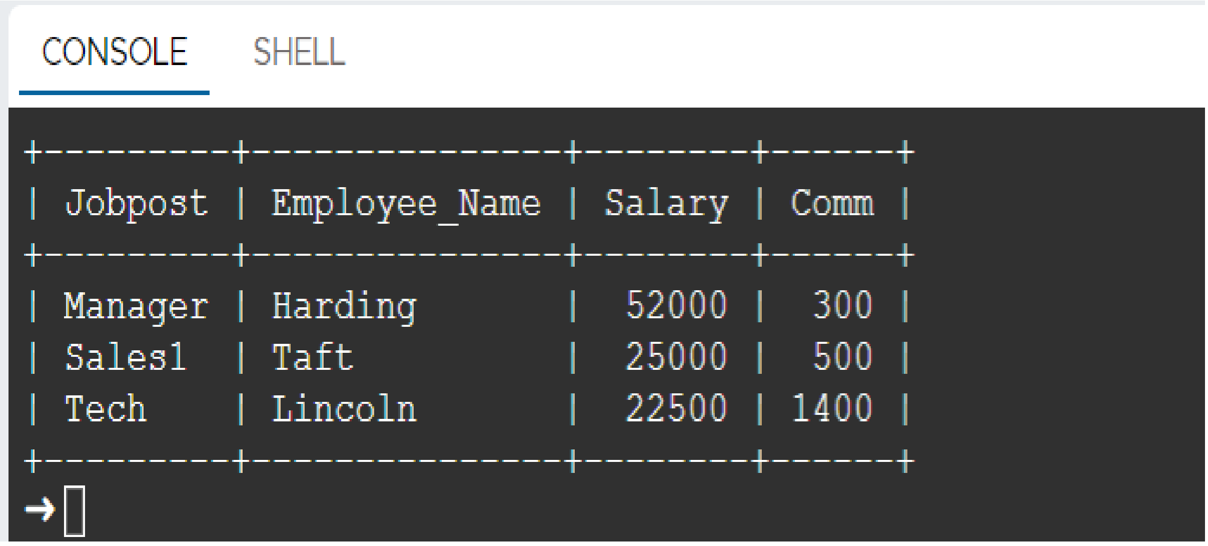
```
+---------+---------------+--------+------+
| Jobpost | Employee_Name | Salary | Comm |
+---------+---------------+--------+------+
| Admin   | Jackson       |  75000 | NULL |
| Manager | Fillmore      |  56000 | NULL |
| Manager | Garfield      |  54000 | NULL |
| Manager | Harding       |  52000 |  300 |
+---------+---------------+--------+------+
➜
```

Problem Statement-6: Job, Employee Name, Salary and Commission for employees with a Commission sorted by Salary (Largest to Smallest).

SQL Program:

SELECT Jobpost,Employee_Name,Salary,Comm from Employee where not comm="null" order by Salary Desc;

Output:



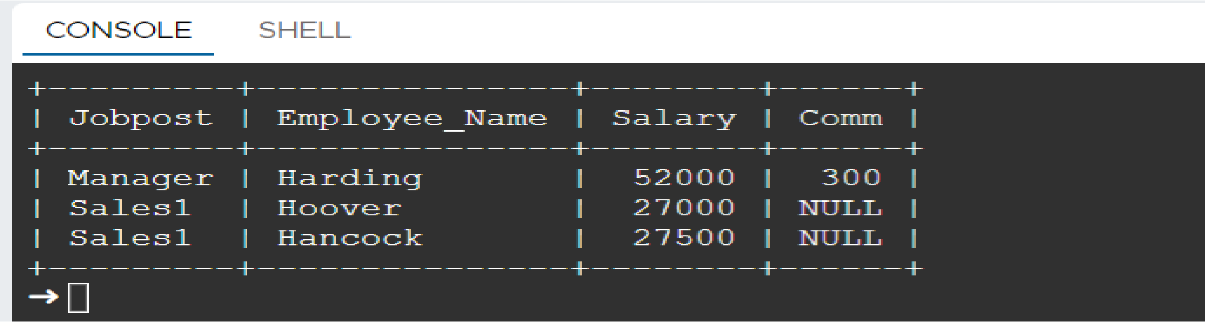
```
+---------+---------------+--------+------+
| Jobpost | Employee_Name | Salary | Comm |
+---------+---------------+--------+------+
| Manager | Harding       |  52000 |  300 |
| Sales1  | Taft          |  25000 |  500 |
| Tech    | Lincoln       |  22500 | 1400 |
+---------+---------------+--------+------+
➜
```

Problem Statement-7: Job, Employee Name, Salary and Commission for employees whose name starts with the letter H.

SQL Program:

SELECT Jobpost,Employee_Name,Salary,Comm from Employee where Employee_Name like "H%";

Output:



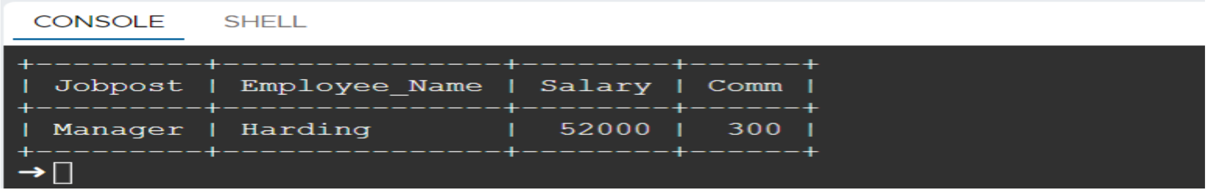
```
+---------+---------------+---------+-------+
| Jobpost | Employee_Name | Salary  | Comm  |
+---------+---------------+---------+-------+
| Manager | Harding       |  52000  |  300  |
| Sales1  | Hoover        |  27000  | NULL  |
| Sales1  | Hancock       |  27500  | NULL  |
+---------+---------------+---------+-------+
→
```

Problem Statement-8: Job, Employee Name, Salary and Commission for employees whose name starts with the letter H and who do not get commission.

SQL Program:

SELECT Jobpost,Employee_Name,Salary,Comm from Employee where not comm="null" and Employee_Name like "H%";

Output:



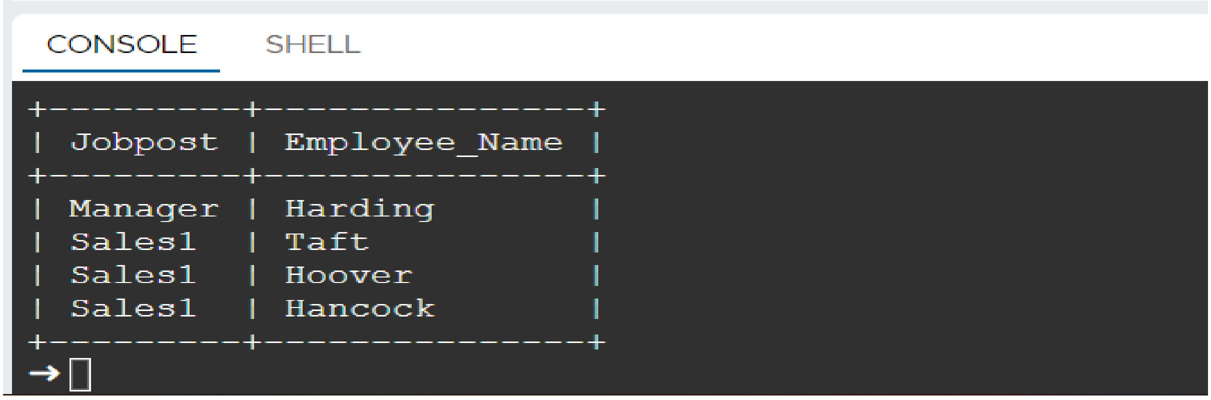
```
+---------+---------------+---------+-------+
| Jobpost | Employee_Name | Salary  | Comm  |
+---------+---------------+---------+-------+
| Manager | Harding       |  52000  |  300  |
+---------+---------------+---------+-------+
→
```

Problem Statement-9: Job, Employee Name for employees in Dept No. 3.

SQL Program:

SELECT Jobpost,Employee_Name from Employee where Department_no=3;

Output:



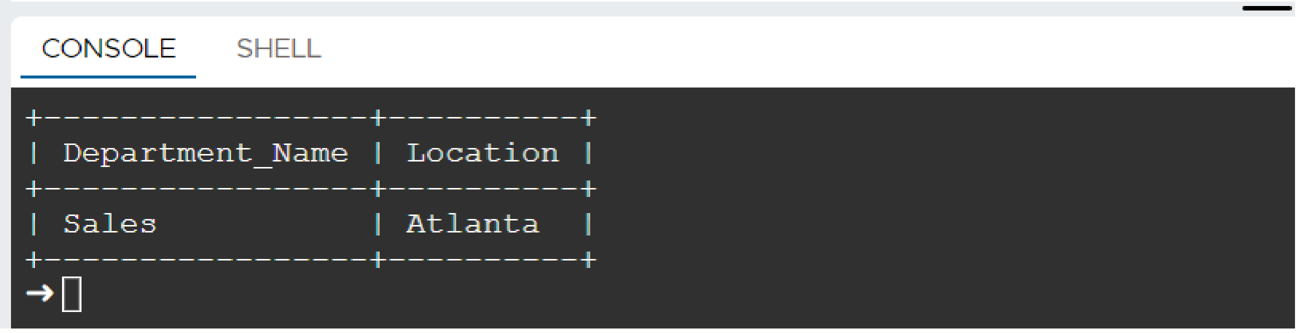
```
+---------+---------------+
| Jobpost | Employee_Name |
+---------+---------------+
| Manager | Harding       |
| Sales1  | Taft          |
| Sales1  | Hoover        |
| Sales1  | Hancock       |
+---------+---------------+
→
```

Problem Statement-10: Dept Name and Loc for employees in Dept No. 3.

SQL Program:

SELECT distinct Department.Department_Name,Department.Location from Employee Inner Join Department where Department.Department_no=3;

Output:



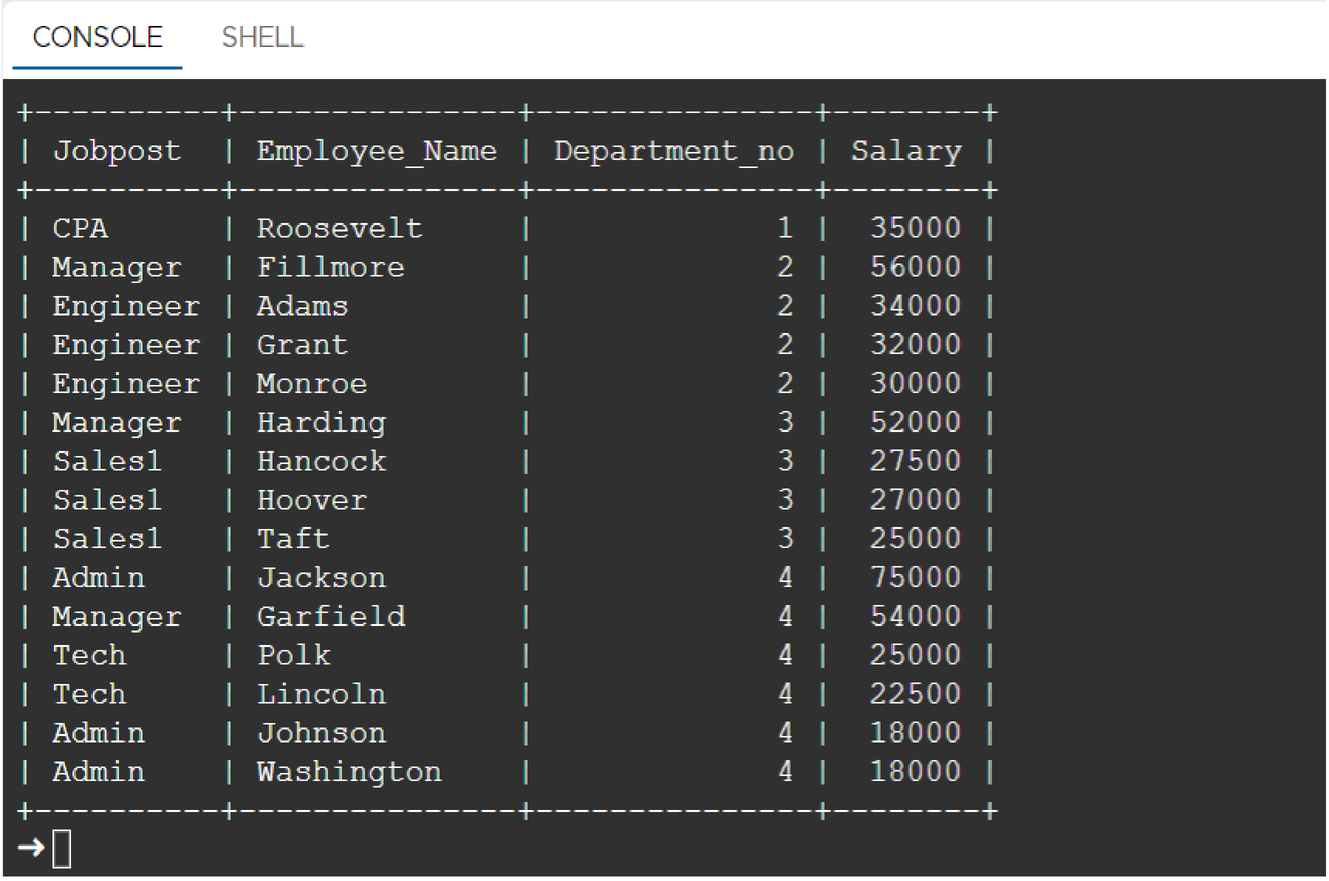
```
+-----------------+----------+
| Department_Name | Location |
+-----------------+----------+
| Sales           | Atlanta  |
+-----------------+----------+
→
```

Problem Statement-11: Job, Employee Name, Dept, Salary sorted first by Dept (Smallest to Largest) and then Salary (Largest to Smallest).

SQL Program:

SELECT Jobpost, Employee_Name,Department_no,Salary from Employee order by Department_no asc, salary desc;

Output:



```
+----------+---------------+---------------+--------+
| Jobpost  | Employee_Name | Department_no | Salary |
+----------+---------------+---------------+--------+
| CPA      | Roosevelt     |             1 |  35000 |
| Manager  | Fillmore      |             2 |  56000 |
| Engineer | Adams         |             2 |  34000 |
| Engineer | Grant         |             2 |  32000 |
| Engineer | Monroe        |             2 |  30000 |
| Manager  | Harding       |             3 |  52000 |
| Sales1   | Hancock       |             3 |  27500 |
| Sales1   | Hoover        |             3 |  27000 |
| Sales1   | Taft          |             3 |  25000 |
| Admin    | Jackson       |             4 |  75000 |
| Manager  | Garfield      |             4 |  54000 |
| Tech     | Polk          |             4 |  25000 |
| Tech     | Lincoln       |             4 |  22500 |
| Admin    | Johnson       |             4 |  18000 |
| Admin    | Washington    |             4 |  18000 |
+----------+---------------+---------------+--------+
→
```

EXPERIMENT - 5

Objective: To implement the concept of aggregating & grouping of Data.

Theory:

Group Functions: Group functions operate on a set of rows, the result is based on a group of rows rather than one result per row as returned by single row functions.

1)Avg: return average value of n.

Syntax: Avg ([distinct/all] n)

2)Min: return minimum value of expr.

Syntax: MIN([distinct/all] expr)

3) Count: Returns the no of rows where expr is not null

Syntax: Count ([distinct/all] expr)

Count(*) : Returns the no rows in the table, including duplicates and those with nulls.

4)Max: Return max value of expr

Syntax: Max ([distinct/all)expr)

5) Sum: Returns sum of values of n

Syntax: Sum ([distinct/all]n)

Grouping Data From Tables: There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples, we specify this wish in SQL using the group by clause. The attribute or attributes given in the group by clause are used to form a group. Tuples with the same value on all attributes in the group by clause are placed in one group.

Syntax: SELECT column name, column name FROM tablename GROUP BY column name;

HAVING clause: The HAVING clause can be used in conjunction with the GROUP BY clause. HAVING imposes a condition on the GROUP BY clause, this further filters the groups created by the GROUP BY clause. HAVING clauses can be used to find duplicates in a relation or in other words find unique values in the situations where DISTINCT cannot apply.

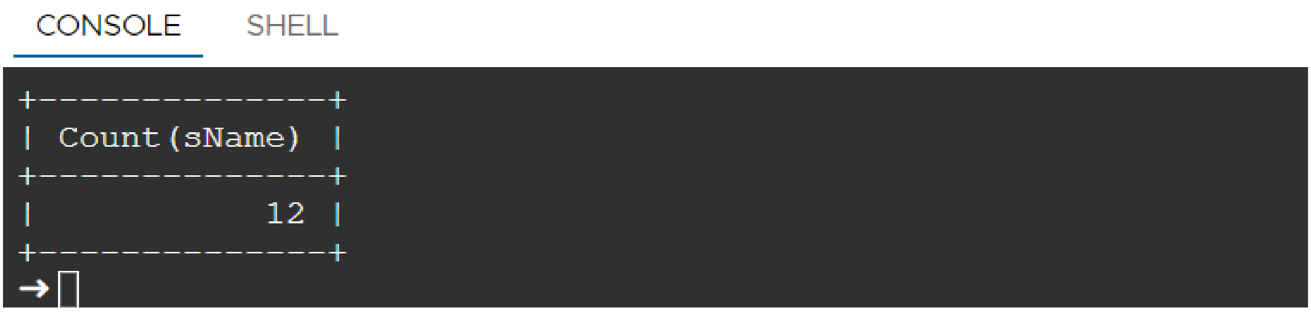
Exercise-6:

Problem Statement-1: Count the total number of students.

SQl Program:

```
Select Count(sName) from Student;
```

Output:



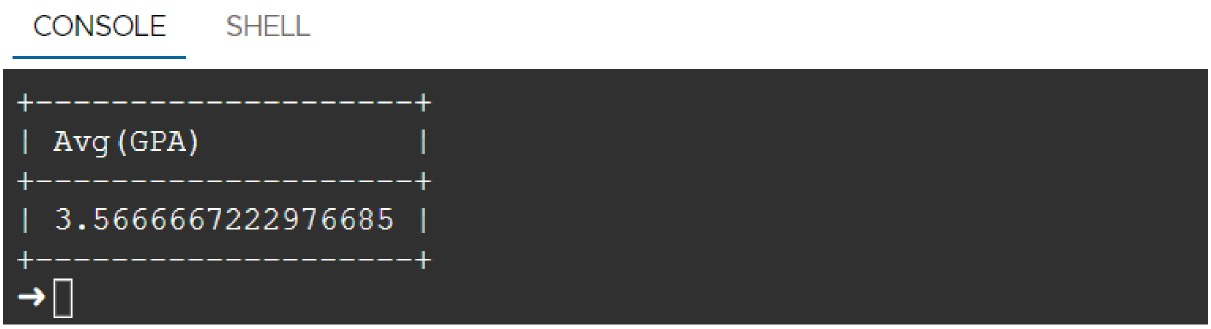
```
+--------------+
| Count(sName) |
+--------------+
|           12 |
+--------------+
→
```

Problem Statement-2: Calculate the average GPA of all students.

SQl Program:

Select Avg(GPA) from Student;

Output:



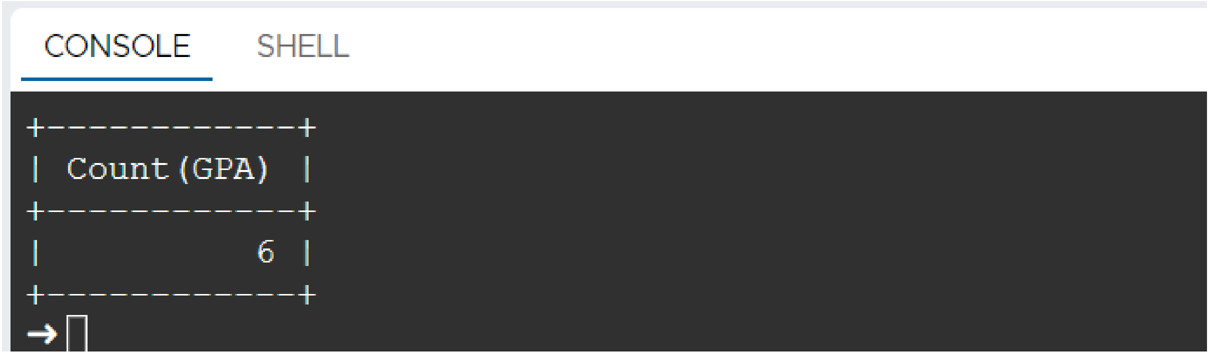
```
+--------------------+
| Avg(GPA)           |
+--------------------+
| 3.5666667222976685 |
+--------------------+
→
```

Problem Statement-3: Count the number of students having GPA greater than or equal to 3.7.

SQl Program:

Select Count(GPA) from Student where GPA>=3.7;

Output:



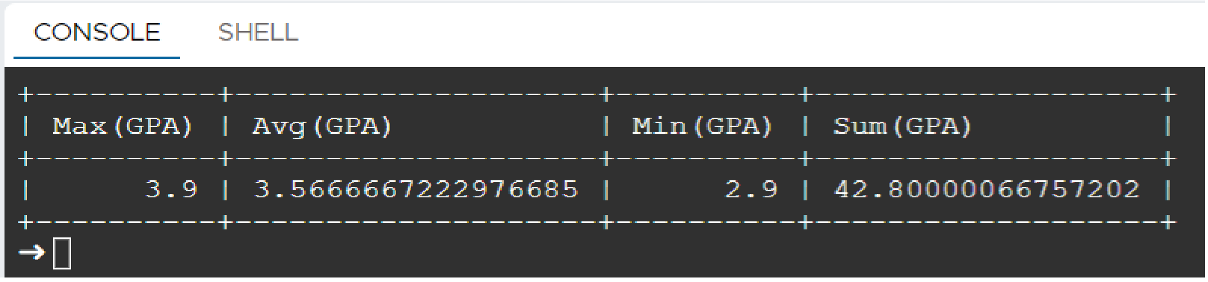
```
+------------+
| Count(GPA) |
+------------+
|          6 |
+------------+
→
```

Problem Statement-4: Find Maximum, Average, Minimum, total GPA of all students.

SQl Program:

Select Max(GPA),Avg(GPA),Min(GPA),Sum(GPA) from Student;

Output:



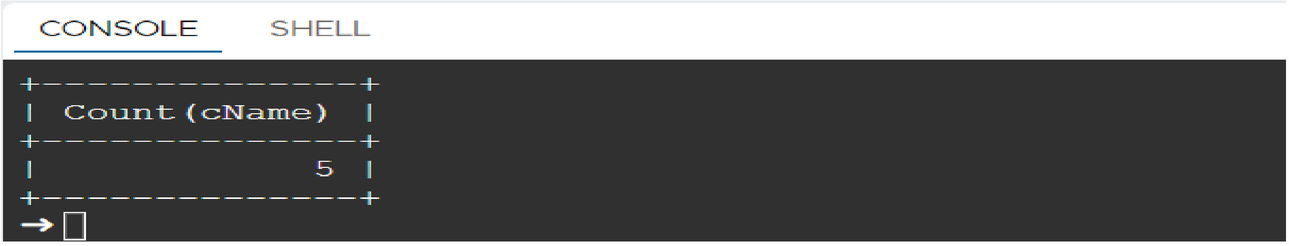
```
+----------+--------------------+----------+-------------------+
| Max(GPA) | Avg(GPA)           | Min(GPA) | Sum(GPA)          |
+----------+--------------------+----------+-------------------+
|      3.9 | 3.5666667222976685 |      2.9 | 42.80000066757202 |
+----------+--------------------+----------+-------------------+
→
```

Problem Statement-5: Find total number of colleges in our Application Database.

SQl Program:

Select Count(cName) from College;

Output:



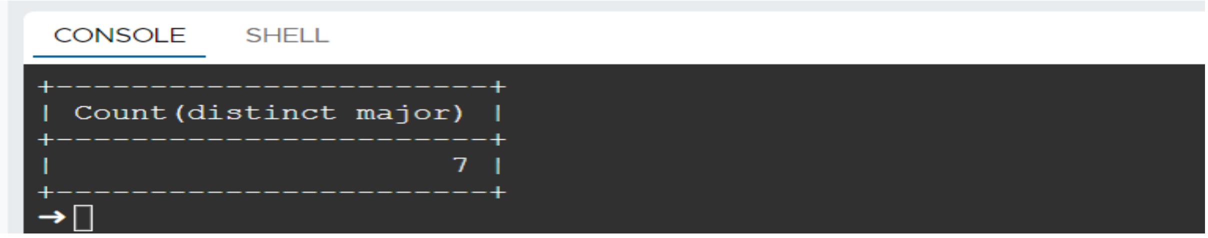
```
+--------------+
| Count(cName) |
+--------------+
|            5 |
+--------------+
→
```

Problem Statement-6: Find how many different majors students had applied in.

SQl Program:

Select Count(distinct major) from Applicant;

Output:



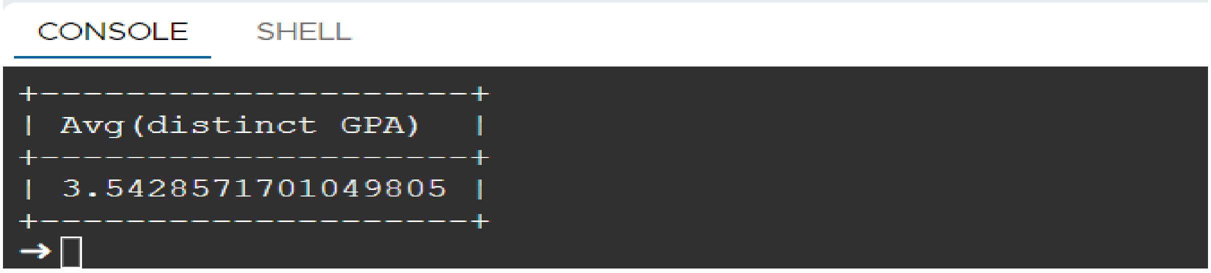
```
+-----------------------+
| Count(distinct major) |
+-----------------------+
|                     7 |
+-----------------------+
→
```

Problem Statement-7: Find average of all distinct GPA.

SQl Program:

Select Avg(distinct GPA) from Student;

Output:



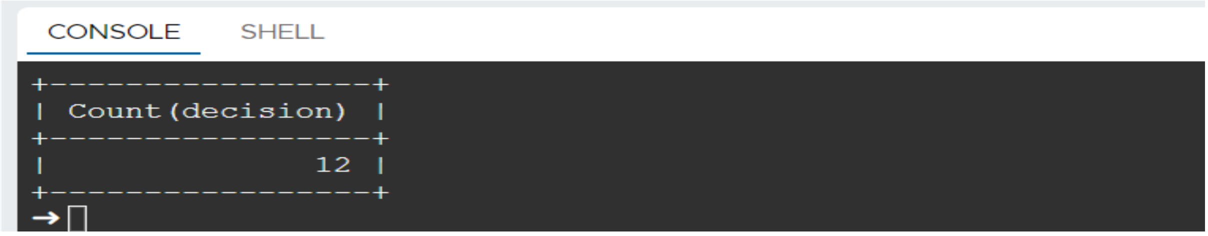
```
+---------------------+
| Avg(distinct GPA)   |
+---------------------+
| 3.5428571701049805  |
+---------------------+
→
```

Problem Statement-8: Display the total number of applications accepted.

SQl Program:

Select Count(decision) from Applicant where decision="Y";

Output:



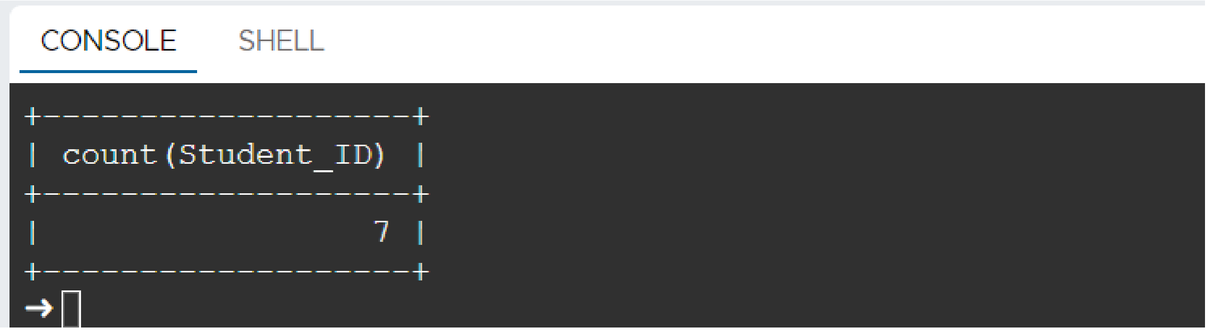
```
+-----------------+
| Count(decision) |
+-----------------+
|              12 |
+-----------------+
→
```

Problem Statement-9: Find how many students applied to a particular major (show count(sid) as No of applications).

SQl Program:

Select count(Student_ID) from Applicant where major="CS";

Output:



```
+-------------------+
| count(Student_ID) |
+-------------------+
|                 7 |
+-------------------+
→
```

EXPERIMENT - 6

Objective: To implement the concept of Triggers.

Theory:

A Trigger in Structured Query Language is a set of procedural statements which are executed automatically when there is any response to certain events on the particular table in the database. Triggers are used to protect the data integrity in the database.

The trigger is always executed with the specific table in the database. If we remove the table, all the triggers associated with that table are also deleted automatically.

In Structured Query Language, triggers are called only either before or after the below events:

- **INSERT Event:** This event is called when the new row is entered in the table.
- **UPDATE Event:** This event is called when the existing record is changed or modified in the table.
- **DELETE Event:** This event is called when the existing record is removed from the table.

Types of Triggers:

Following are the six types of triggers in SQL:

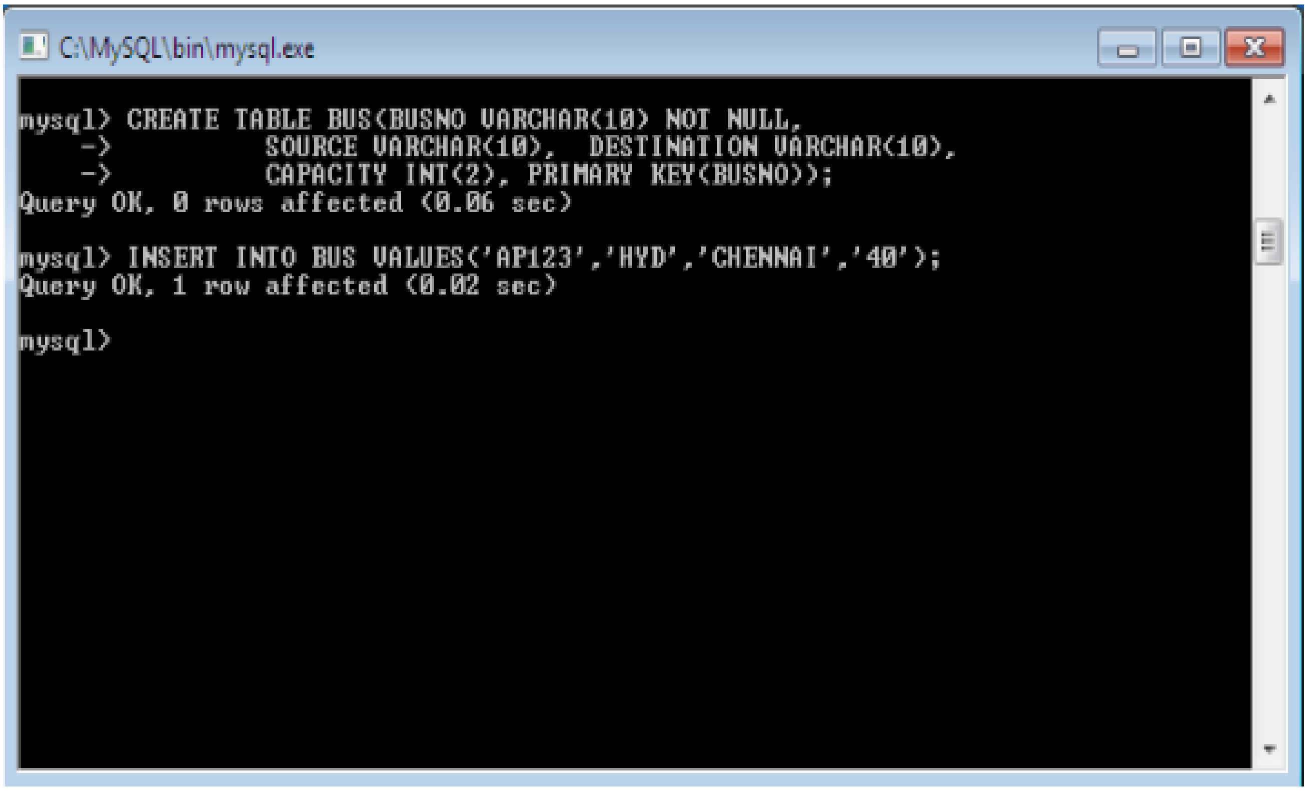
1. **AFTER INSERT Trigger**
   This trigger is invoked after the insertion of data in the table.
2. **AFTER UPDATE Trigger**
   This trigger is invoked in SQL after the modification of the data in the table.
3. **AFTER DELETE Trigger**
   This trigger is invoked after deleting the data from the table.
4. **BEFORE INSERT Trigger**
   This trigger is invoked before inserting the record in the table.
5. **BEFORE UPDATE Trigger**
   This trigger is invoked before updating the record in the table.
6. **BEFORE DELETE Trigger**
   This trigger is invoked before deleting the record from the table.

Syntax of Trigger:

```
CREATE TRIGGER Trigger_Name
[ BEFORE | AFTER ]  [ Insert | Update | Delete]
ON [Table_Name]
[ FOR EACH ROW | FOR EACH COLUMN ]
AS
Set of SQL Statement
```

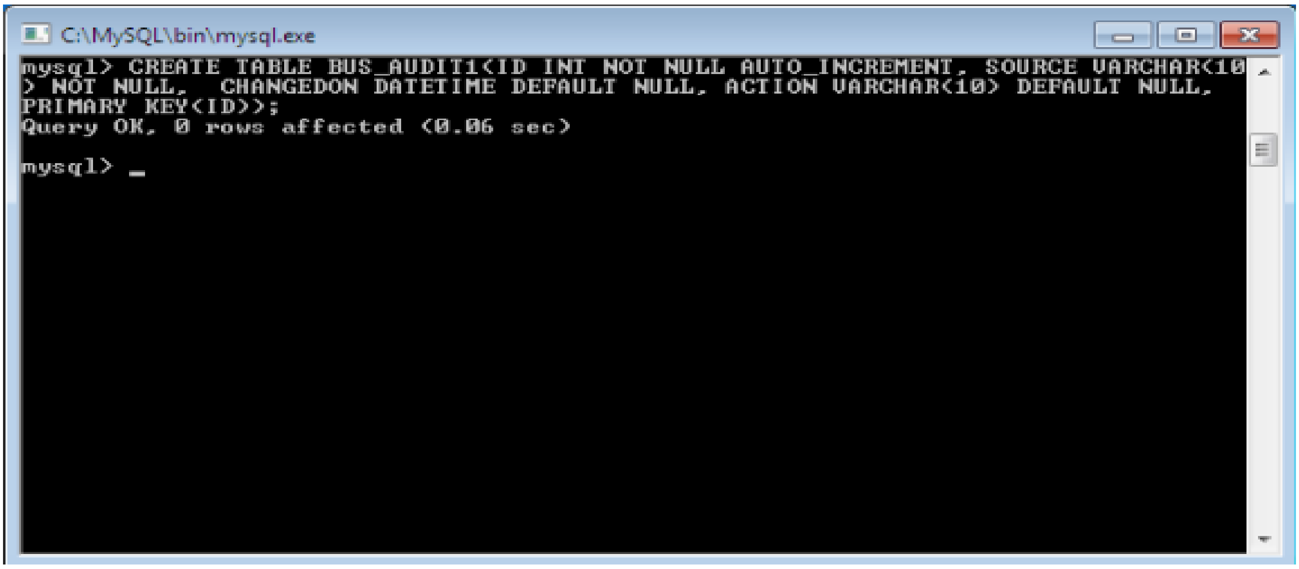
Examples:

MySQL>CREATE TABLE BUS(BUSNO VARCHAR(10) NOT NULL, SOURCE VARCHAR(10), DESTINATION VARCHAR(10), CAPACITY INT(2), PRIMARY KEY(BUSNO)); MySQL>INSERT INTO BUS VALUES('AP123','HYD','CHENNAI','40');



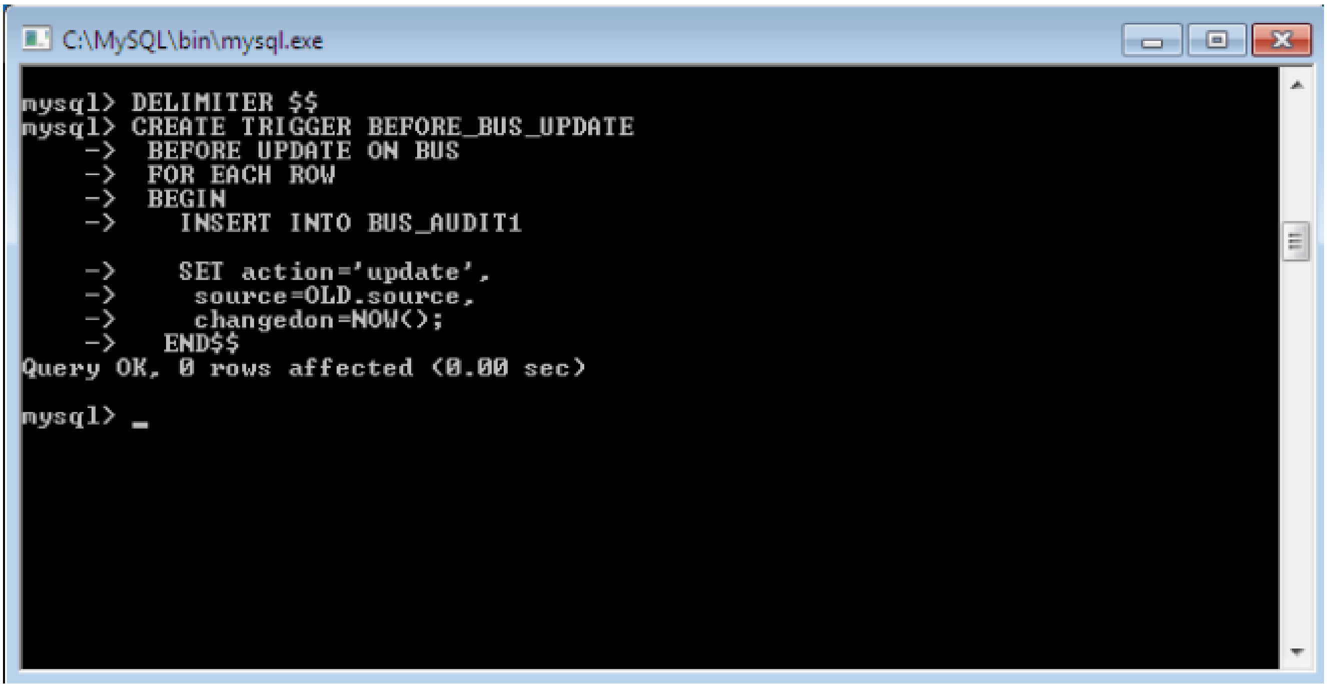
```
mysql> CREATE TABLE BUS(BUSNO VARCHAR(10) NOT NULL,
    ->          SOURCE VARCHAR(10),  DESTINATION VARCHAR(10),
    ->          CAPACITY INT(2), PRIMARY KEY(BUSNO));
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO BUS VALUES('AP123','HYD','CHENNAI','40');
Query OK, 1 row affected (0.02 sec)

mysql>
```

CREATE TABLE BUS_AUDIT1(ID INT NOT NULL AUTO_INCREMENT, SOURCE VARCHAR(10) NOT NULL, CHANGEDON DATETIME DEFAULT NULL, ACTION VARCHAR(10) DEFAULT NULL, PRIMARY KEY(ID));



```
mysql> CREATE TABLE BUS_AUDIT1(ID INT NOT NULL AUTO_INCREMENT, SOURCE VARCHAR(10
) NOT NULL,  CHANGEDON DATETIME DEFAULT NULL, ACTION VARCHAR(10) DEFAULT NULL,
PRIMARY KEY(ID));
Query OK, 0 rows affected (0.06 sec)

mysql> _
```

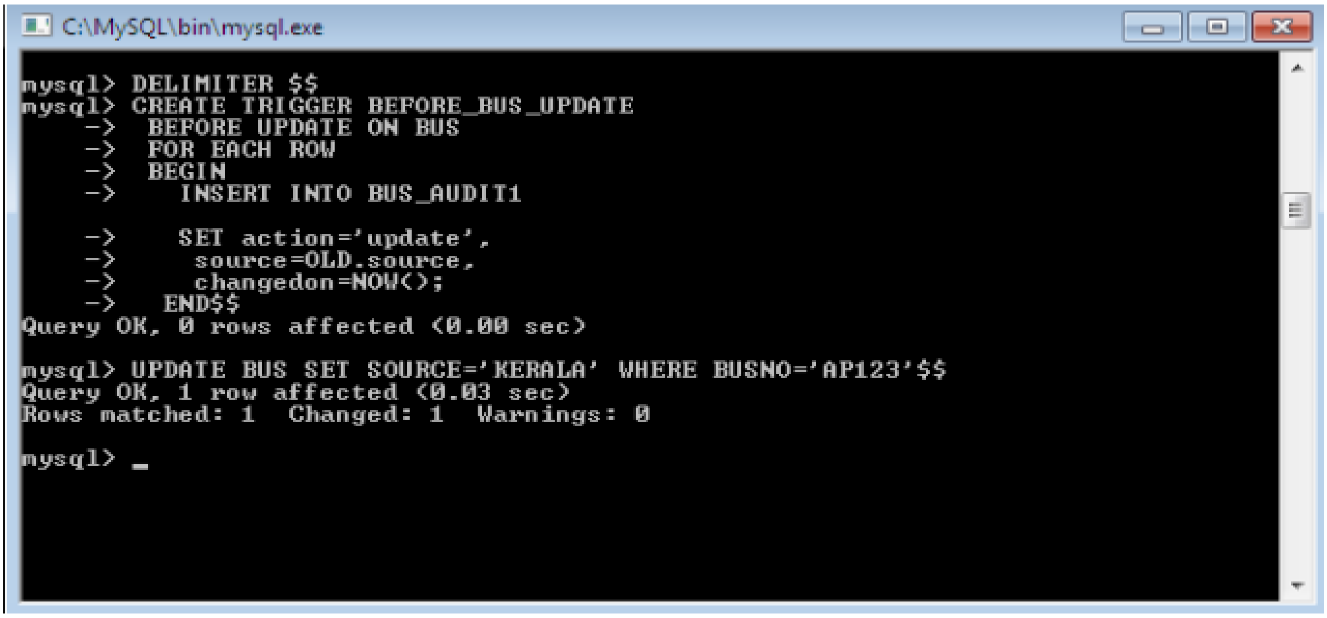
CREATE TRIGGER BEFORE_BUS_UPDATE BEFORE UPDATE ON BUS FOR EACH ROW BEGIN INSERT INTO BUS_AUDIT1 SET action='update', source=OLD.source, changedon=NOW(); END$$




```
mysql> DELIMITER $$
mysql> CREATE TRIGGER BEFORE_BUS_UPDATE
    ->  BEFORE UPDATE ON BUS
    ->  FOR EACH ROW
    ->  BEGIN
    ->    INSERT INTO BUS_AUDIT1

    ->    SET action='update',
    ->     source=OLD.source,
    ->     changedon=NOW();
    ->   END$$
Query OK, 0 rows affected (0.00 sec)

mysql> _
```

UPDATE:

MySQL>UPDATE BUS SET SOURCE='KERALA' WHERE BUSNO='AP123'$$

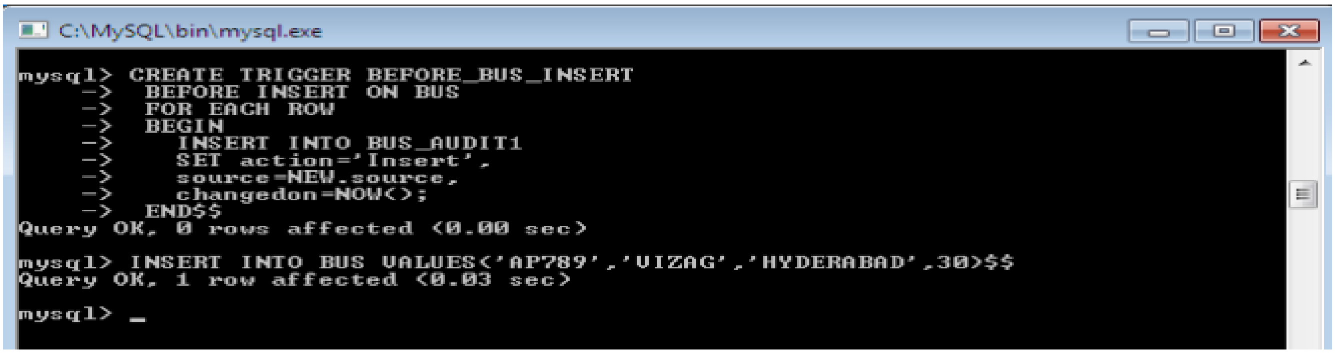


```
mysql> DELIMITER $$
mysql> CREATE TRIGGER BEFORE_BUS_UPDATE
    ->  BEFORE UPDATE ON BUS
    ->  FOR EACH ROW
    ->  BEGIN
    ->    INSERT INTO BUS_AUDIT1

    ->    SET action='update',
    ->     source=OLD.source,
    ->     changedon=NOW();
    ->   END$$
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE BUS SET SOURCE='KERALA' WHERE BUSNO='AP123'$$
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> _
```

SNo	Source	Changedon	Action
1	Banglore	2014:03:23 12:51:00	Insert
2	Kerela	2014:03:25:12:56:00	Update
3	Mumbai	2014:04:26:12:59:02	Delete

INSERT:

CREATE TRIGGER BEFORE_BUS_INSERT BEFORE INSERT ON BUS FOR EACH ROW BEGIN INSERT INTO BUS_AUDIT1 SET action='Insert', source=NEW.source, changedon=NOW(); END$$ MYSQL>INSERT INTO BUS VALUES('AP789','VIZAG','HYDERABAD',30)$$



```
mysql> CREATE TRIGGER BEFORE_BUS_INSERT
    ->  BEFORE INSERT ON BUS
    ->  FOR EACH ROW
    ->  BEGIN
    ->    INSERT INTO BUS_AUDIT1
    ->    SET action='Insert',
    ->    source=NEW.source,
    ->    changedon=NOW();
    ->  END$$
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO BUS VALUES('AP789','VIZAG','HYDERABAD',30)$$
Query OK, 1 row affected (0.03 sec)

mysql> _
```

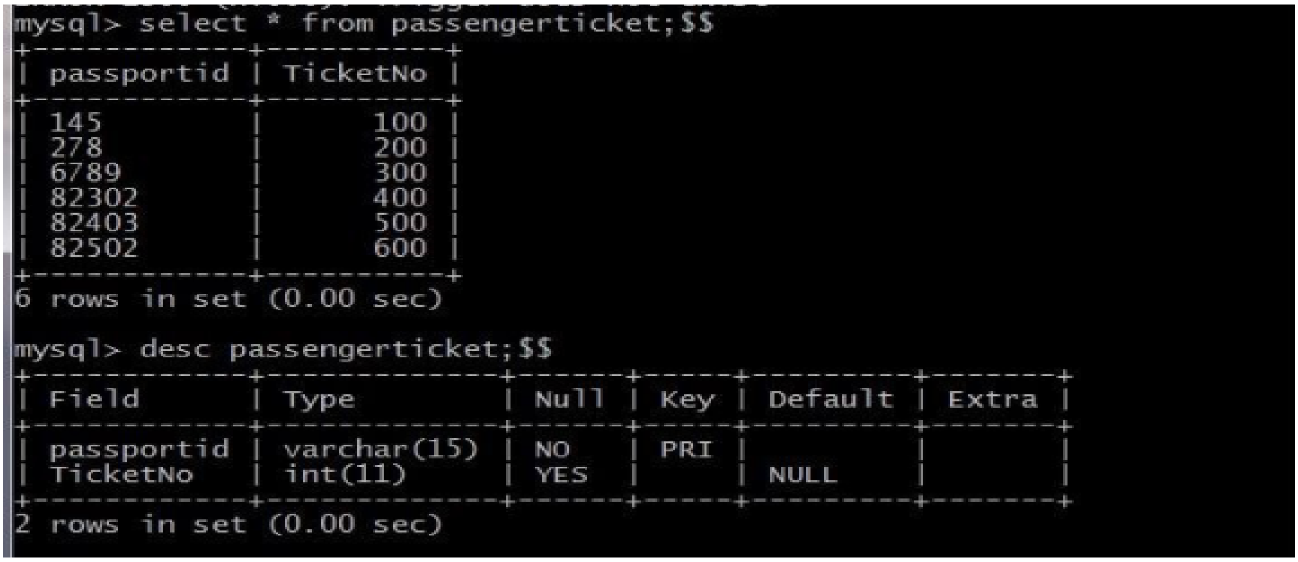
SNo	Source	Changedon	Action
1	Banglore	2014:03:23 12:51:00	Insert
2	Kerela	2014:03:25:12:56:00	Update
3	Mumbai	2014:04:26:12:59:02	Delete

CREATE TRIGGER BEFORE_BUS_DELETE BEFORE DELETE ON BUS FOR EACH ROW BEGIN DELETE FROM BUS_AUDIT1 SET action='Insert', source=NEW.source, changedon=NOW(); END$$ DELETE FROM BUS WHERE SOURCE='HYDERABAD'$$

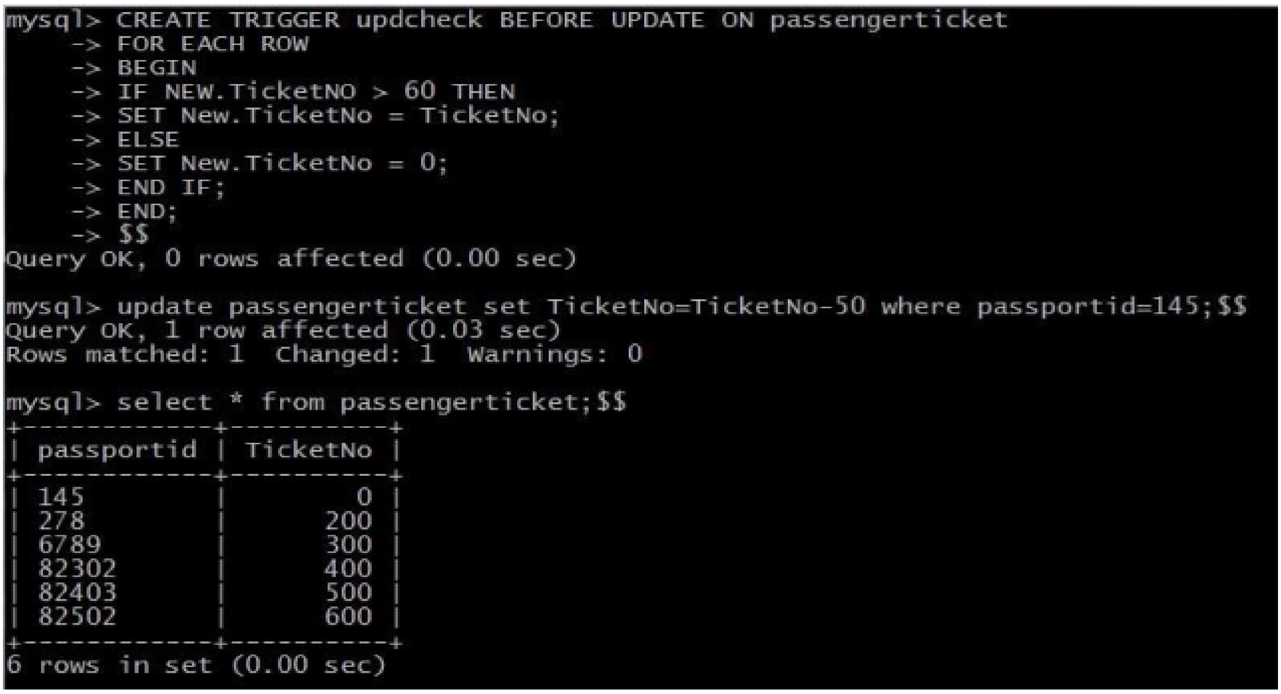
SNo	Source	Changedon	Action
1	Banglore	2014:03:23 12:51:00	Insert
2	Kerela	2014:03:25:12:56:00	Update
3	Mumbai	2014:04:26:12:59:02	Delete

Examples

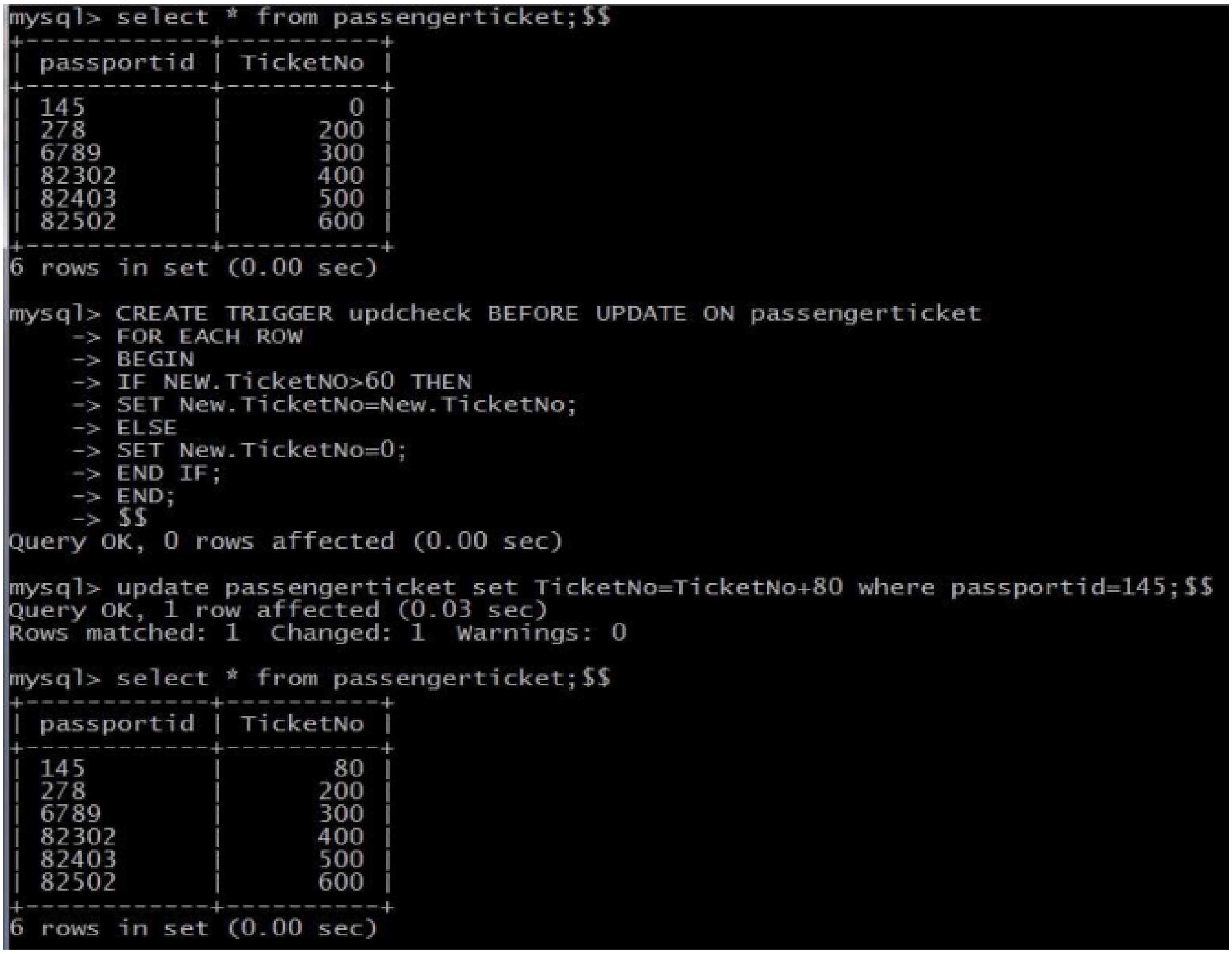
CREATE TRIGGER updcheck1 BEFORE UPDATE ON passengerticket FOR EACH ROW BEGIN IF NEW.TicketNO > 60 THEN SET New.TicketNo = New.TicketNo; ELSE SET New.TicketNo = 0; END IF; END;



```
mysql> select * from passengerticket;$$
+------------+----------+
| passportid | TicketNo |
+------------+----------+
| 145        |      100 |
| 278        |      200 |
| 6789       |      300 |
| 82302      |      400 |
| 82403      |      500 |
| 82502      |      600 |
+------------+----------+
6 rows in set (0.00 sec)

mysql> desc passengerticket;$$
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| passportid | varchar(15) | NO   | PRI |         |       |
| TicketNo   | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```



```
mysql> CREATE TRIGGER updcheck BEFORE UPDATE ON passengerticket
    -> FOR EACH ROW
    -> BEGIN
    -> IF NEW.TicketNO > 60 THEN
    -> SET New.TicketNo = TicketNo;
    -> ELSE
    -> SET New.TicketNo = 0;
    -> END IF;
    -> END;
    -> $$
Query OK, 0 rows affected (0.00 sec)

mysql> update passengerticket set TicketNo=TicketNo-50 where passportid=145;$$
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from passengerticket;$$
+------------+----------+
| passportid | TicketNo |
+------------+----------+
| 145        |        0 |
| 278        |      200 |
| 6789       |      300 |
| 82302      |      400 |
| 82403      |      500 |
| 82502      |      600 |
+------------+----------+
6 rows in set (0.00 sec)
```



```
mysql> select * from passengerticket;$$
+------------+----------+
| passportid | TicketNo |
+------------+----------+
| 145        |        0 |
| 278        |      200 |
| 6789       |      300 |
| 82302      |      400 |
| 82403      |      500 |
| 82502      |      600 |
+------------+----------+
6 rows in set (0.00 sec)

mysql> CREATE TRIGGER updcheck BEFORE UPDATE ON passengerticket
    -> FOR EACH ROW
    -> BEGIN
    -> IF NEW.TicketNO>60 THEN
    -> SET New.TicketNo=New.TicketNo;
    -> ELSE
    -> SET New.TicketNo=0;
    -> END IF;
    -> END;
    -> $$
Query OK, 0 rows affected (0.00 sec)

mysql> update passengerticket set TicketNo=TicketNo+80 where passportid=145;$$
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from passengerticket;$$
+------------+----------+
| passportid | TicketNo |
+------------+----------+
| 145        |       80 |
| 278        |      200 |
| 6789       |      300 |
| 82302      |      400 |
| 82403      |      500 |
| 82502      |      600 |
+------------+----------+
6 rows in set (0.00 sec)
```