

## **Group 9.12**

Aditya Jadeja : 202001432

Het Patel : 202001434

---

### **Project : Student Reward System**

**Mentor name:** Rahul Vansh



# **Software Requirement Specification for Students reward system**

## **Index:**

1. Introduction
  - 1.1. Purpose
  - 1.2. Intended audience and reading suggestion
  - 1.3. Product scope
  - 1.4. Description
2. Document the Requirements Collection/ Fact Finding Phase
  - 2.1. Readings and Description
  - 2.2. Interviews
  - 2.3. Questionnaires
  - 2.4. Observations
3. Fact Finding Chart
4. List requirements
5. User categories and Privileges
6. Assumptions
7. Business Constraints

# 1. Introduction:

## What is the student reward system?

A Students reward system is a motivation tactic that organizations use to help their Students feel encouraged to complete high-quality learning in every domain.

### 1.1 Purpose

To recognize excellence in all areas of student life, including academics, the arts, sports, perseverance, planning, self-assurance, resilience, social skills, and compassion.

- Encourage students to concentrate on their studies
- To ensure effective engagement in extracurricular activities
- To maintain an organized record of every student's learning behavior
- to give teachers a simple interface for managing each student's profile.
- Organize rewards distribution effectively
- Encourage peer-to-peer recognition to increase students' collaborative learning

### 1.2. Intended Audience and Reading Suggestions

that the document is intended for, such as developers, project managers, mentors, marketing staff, users, testers, and documentation writers.

### 1.3 Product Scope

- This reward system is specifically designed for the use of mentors,teachers,competition organizers and students, and their parents.
- Reward system will be designed to maximize the user(teacher,mentor,competition organizer) productivity by providing tools to assist in maintaining the track of rewards of respective students and also in maintaining the progress of each student, which would otherwise have to be performed manually.
- Student reward system is basically updating the reward information of students into application so that students can know their rewards on the basis of daily/weekly quizzes ,question-answer during lecture,semester wise exam,non academic activities etc.

## 1.4 Description

### What we provide:

The Student reward system will provide an interface for organizations to manage students' learning behavior through rewards and much more. It will keep track of students' learning behavior. Students and Teachers both will have dashboards at their ends Parents of each student will also have dashboards. Teachers will be able to decide the reward scheme and students can get access to it. The main aspect of the "Students Reward System" are as following:

- Students can Check their performance, test scores , and rewards history at their end.
- Parents are able to see progress &,Online Result of their child and attendance of the child.
- Teachers can add test scores , give rewards and view each student's progress.

### The relations that we are planning to implement for this system:

#### **Student:**

- It contains student's information such as name, ID, phone number, address.

#### **Instructor:**

- Relation represents instructor's personal information such as instructor ID, course ID, phone number, address.

#### **Course :**

- It contains information about course name , instructor name , instructor ID, course ID.

#### **Reward :**

- It contains information about reward type, activity type, course ID, instructor ID.

#### **Academic\_Rewards:**

- Data such as Student\_ID, course\_ID, Activity type, obtained marks, total marks, Reward points

#### **Non-academic\_Rewards:**

- Data such as Student\_ID, Activity type, Reward points

#### **Scale of Academic Performance:**

- It contains data of Instructor\_ID , course\_ID , activity type, scale.

**Scale of Non- Academic Performance:**

- It contains data of Instructor\_ID , activity type, scale.

**Scale of combined Performance:**

- It contains data of Instructor\_ID , academic scale,non- academic scale

**Performance:**

- It has attributes of student\_ID, Instructor\_ID , performance .

**Parent :**

- It contains information about parent name, parent email, student ID.

**Alert :**

- It has attributes such as student ID, alert type , alert message.

**Admin :**

- It has attributes such as admin name, admin id.

**Functions that planning to implement:**

**Add student:** adding new student and information of that student

**Add course:** adding up a new course

**Add instructor :** the function add the new instructor into the database

**Update instructor:** updating the information of instructor into the database

**Remove old student :** the function delete the information of old student into the database

**Remove course :** delete the course into the database. This function is used by the admin and instructor.

**Remove instructor :** remove the instructor details from the database which use by admin only

**Add reward :** This function adds the reward information like reward type, activity type and which instructor gives this reward into the database.

**Removes reward :** remove the reward details given by the instructor from the database.

**Give academic reward :** add student id and reward information by instructor into the database

**Update academic reward info :** update the student reward information into the database

**Remove academic reward :** removes the student id and reward information from the database

**Give non academic reward :** add student id and reward information by instructor into the database

**Update non academic reward info** : update the student reward information into the database

**Remove non academic reward** : removes the student id and reward information from the database

**Add scale to course** : instructor uses this function to define his/her course structure for the semester how to divide percentages among the academic activities.

**Update scale to course** : instructor uses this function to update his/her course structure for the semester and how to divide percentages among the academic activities.

**Remove course scale** : to remove course scale when course is removed by instructor or instructor is removed.

**Add scale for non academic activity** : This function adds scale to non academic activities

Like coding,sports,music etc. into the database by instructor.

**Update non academic activity** : update the existing scale of non academic activity.

**Removes non academic scale** : This function deletes the scale of non academic activities.

**Add overall performance scale** : add the scale of academic and non academic for the semester by instructor to calculate the performance of the student.

**Removes the performance scale** : removes the instructor-added performance scale from the database.

**Alert parent** : when student's attendance goes below 75% this trigger function adds the alert message and alert type into the database.

**Add alert** : function adds the alert type, alert message and student ID by instructor into the database.

### Workflow for Student reward system:

- An existing student, who is already in the system. There will be a separate view for students so that they can check out their basic information, academic information like marks of particular course ,performance of the particular course and overall performance or an alert by the instructor.
- If the instructor is new in the college/school , the admin will add the instructor and their course details into the database. If the instructor leaves the college/school, the admin will delete the instructor details and course details of that instructor.
- Instructor view has access to add the student to their course and update the student performance information.also instructor view has access to set the performance scale of that course.
- All the rewards that are given to students by the instructor can be viewed by student,parent and access or update by the instructor.
- Overall calculated performance for every student is added to the performance table and available to see for all students.
- If an instructor decides to inform a student about some academic or non academic activity to all students that information is added to alert.

## 2. Fact-Finding Phase

### 2.1 Background Reading :

#### 1. College management system from

link: [College-ERP/DBMS report.pdf at master](#)

From this, we referred to what are the functions and roles that we need to maintain and the overall relationship between them. In short, we referred to the structure of the database.

#### **Requirements:**

- Each user shall be able to view information in the database based on their user class.
- The administrator shall be able to view all the information in the database.
- Teachers shall be able to view, update and edit the attendance and marks of the students, part of their class.
- Students shall be able to communicate with their instructor by sending personal messages.

### 2.2 Interviews:

#### Interview Plan 1: (Instructor Interview(**Roleplay**))

**Project Reference :** SF/SJ/2022/09

**Interviewee:** Jim Kurose (**Roleplay**)      **Designation:** Instructor at unacademy

**Interviewer:** Aditya Jadeja      **Designation:** System Developer

Het Patel      **Designation:** System Developer

**Date:** 27/9/2022    **Time :** 14:30

**Duration :** 30 minutes    **Place:** Phone call

#### **Purpose of Interview :**

- Preliminary meeting to identify problems and requirements regarding students reward system for instructors point of view.

#### **Agenda :**

- Basic intro to our idea
- Discussion about how the system can help to motivate the students for learning

- What functions instructors will need at their dashboard.
  - How to calculate overall performance of a student

## **Documents to be brought to the interview :**

- Rough plan of building

## Interview Summary 1:

**Project Reference : SF/SJ/2022/09**

**Interviewee:** Jim Kurose([Roleplay](#))      **Designation:** Teacher at unacademy

**Interviewer:** Aditya Jadeja      **Designation:** System Developer

Het Patel      **Designation:**      System Developer

Date: 27/9/2022 Time : 14:30

**Duration :** 30 minutes      **Place:** Phone call

## **Purpose of Interview :**

- Preliminary meeting to identify problems and requirements regarding students reward system for instructors point of view.

### **Results of interview:**

- After the interview, we got to know the needs of an instructor from the system and basic idea of how to design the instructor-side dashboard.
  - Using the pen-paper (register) or excel sheet leads to very tiring work for instructors for tracking the performance of each student.
  - A well organized database and easy to use interface should solve the problem.
  - Instructor agrees with the reward system idea to motivate students and believes that it will help both - students and teachers .
  - Instructor needs a simple procedure for entries of attendance , marks and points of non-academic activities.
  - different academic activities should have their corresponding weightage and same goes for non-academic activities for calculating combined performance of a student.
  - Instructors should be able to choose the scales of each component for evaluating final performance.
  - Students can be sorted according to their performance to give rewards to them.

### Interview Plan 2: (Student Interview)

**Project Reference :** SF/SJ/2022/09

<b>Interviewee:</b>	Manan Dankhara	<b>Designation:</b>	Student at DA-IICT
<b>Interviewer:</b>	Aditya Jadeja	<b>Designation:</b>	System Developer
	Het Patel	<b>Designation:</b>	System Developer

**Date:** 27/9/2022    **Time :** 15:30

**Duration :** 30 minutes    **Place:** Cafetaria

#### **Purpose of Interview :**

- Preliminary meeting to identify problems and requirements regarding students reward system for students point of view.

#### **Agenda :**

- Basic intro to our idea
- Discussion about how the system can help to motivate the students for learning
- What functions students will need at their dashboard.
- Which type of reward do they want?

#### **Documents to be brought to the interview :**

- Rough plan of building

### Interview Summary - 2:

**Project Reference :** SF/SJ/2022/09

<b>Interviewee:</b>	Manan Dankhara	<b>Designation:</b>	Student at DA-IICT
<b>Interviewer:</b>	Aditya Jadeja	<b>Designation:</b>	System Developer
	Het Patel	<b>Designation:</b>	System Developer

**Date:** 27/9/2022    **Time :** 15:30

**Duration :** 30 minutes    **Place:**Cafetaria

## **Purpose of Interview :**

- Preliminary meeting to identify problems and requirements regarding students reward system for students point of view.

## **Results of interview:**

- After the interview, we got to know the needs of a student from the system and basic idea of how to design the student-side dashboard.
  - Students want to have a system in which they are able to get information about their attendance, performance of each activity (academic and non-academic).
  - This system will also ensure transparency in grading and students will be able to know how much and in which direction they need to improve their performance.
  - Each student should have their personal view of their performance records.
  - The leaderboards should be visible to all students so that they can compare their performance and which will lead to healthy competition among students.

### **Interview Plan 3: (Parent Interview)**

**Project Reference : SF/SJ/2022/09**

**Interviewee:** Darshit Bhakhar    **Designation:** Parent

**Interviewer:** Aditya Jadeja      **Designation:** System Developer

Het Patel **Designation:** System Developer

Date: 27/9/2022 Time : 16:30

**Duration :** 30 minutes      **Place:** Phone call

### **Purpose of Interview :**

- Preliminary meeting to identify problems and requirements regarding students' reward system from parents point of view.

## **Agenda :**

- Basic intro to our idea
  - Discussion about how the system can help to motivate the students for learning
  - What functions parents will need at their dashboard.
  - How to communicate with parents about the performance of their child.
  - Concerns about attendance

## **Documents to be brought to the interview :**

- Rough plan of building

**Interview Summary-3:****Project Reference :** SF/SJ/2022/09

<b>Interviewee:</b>	Darshit Bhakhar	<b>Designation:</b>	Parent
<b>Interviewer:</b>	Aditya Jadeja	<b>Designation:</b>	System Developer
	Het Patel	<b>Designation:</b>	System Developer

**Date:** 27/9/2022    **Time :** 16: 30**Duration :** 30 minutes    **Place:** Phone call**Purpose of Interview :**

- Preliminary meeting to identify problems and requirements regarding students reward system for instructors point of view.

**Results of interview:**

- After the interview, we got to know the needs of a parent from the system and basic idea of how to design the instructor-side dashboard.
- Usually parents are unaware of the performance of their child because students don't discuss everything with parents.
- The system has a student's view where every detail of academic,non-academic performance and attendance are displayed . This same view can be used for parents but for parents it is not optimal to check each activity and its performance with their busy schedule.
- To solve this problem we can implement an alert system which will notify the parents/ keep a log of all activity of students with activity type tags, which will ensure ease for parents to stay updated with minimal efforts.
- The parents are also concerned about attendance of their children. So we will also implement an alert for the same which will notify the parents if attendance goes down below 75 percent (which will be implemented by triggers).

**List the combined Requirements**

- Instructor needs a simple procedure for entries of attendance , marks and points of non-academic activities.
- different academic activities should have their corresponding weightage and same goes for non-academic activities for calculating combined performance of a student.
- Instructors should be able to choose the scales of each component for evaluating final performance.
- Students can be sorted according to their performance to give rewards to them.

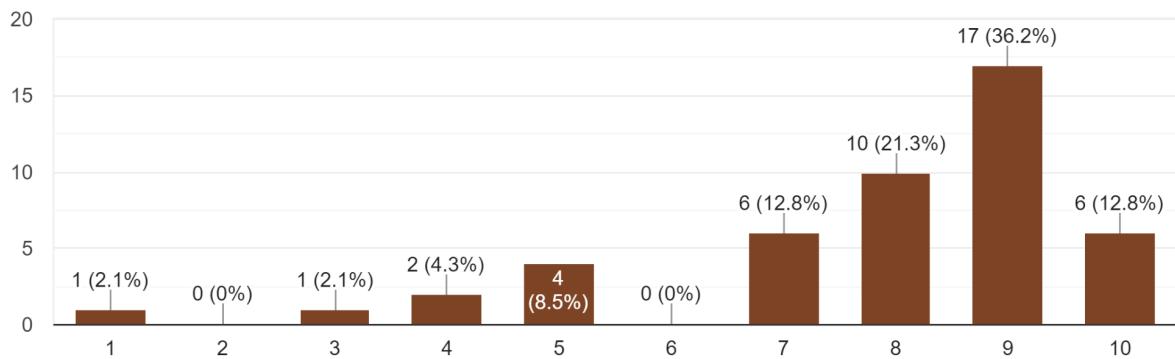
- Students want to have a system in which they are able to get information about their attendance, performance of each activity (academic and non-academic).
- This system will also ensure transparency in grading and students will be able to know how much and in which direction they need to improve their performance.
- Each student should have their personal view of their performance records.
- The leaderboards should be visible to all students so that they can compare their performance.
- an alert system which will notify the parents/ keep a log of all activity of students with activity type tags,
- an alert which will notify the parents if attendance goes down below 75 percent

## 2.3 Questionnaire/s

### Question-1:

How much the Student Reward System will impact on learning behavior of students on the scale of 10?

47 responses



### **Intent of this question:**

- To get to know how effective this system will be .

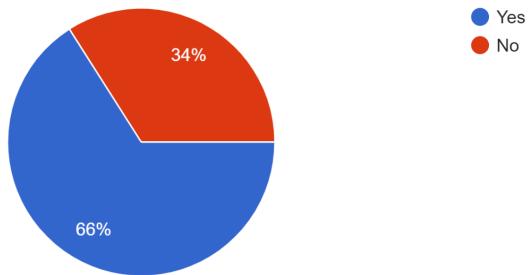
### **Observation from the response:**

- The majority of the responders believe that this system will be useful and will create a positive impact

**Question-2:**

As a student, do you believe that informing your parents about your performance will help you to improve better?

47 responses

**Intent of this question:**

- To get to know if we should notify parents about students' performance or not.

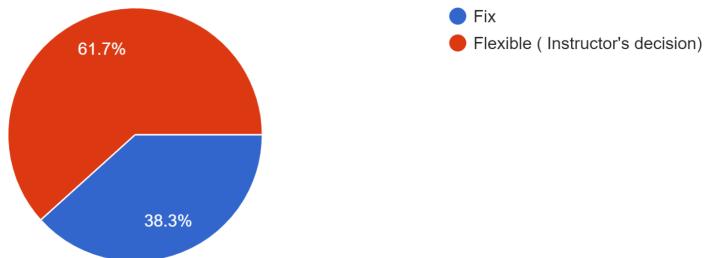
**Observation from the response:**

- The majority of the responders say that notifying parents will help to improve performance of students, so will implement the parent alert functionality.

**Question-3:**

The weightage of academic components (Weekly quiz, Monthly exams, Semester exam, attendance) should be fixed or flexible according to you?

47 responses



**Intent of this question:**

- To get to know if the weightage scale of academic activities should be fixed or flexible.

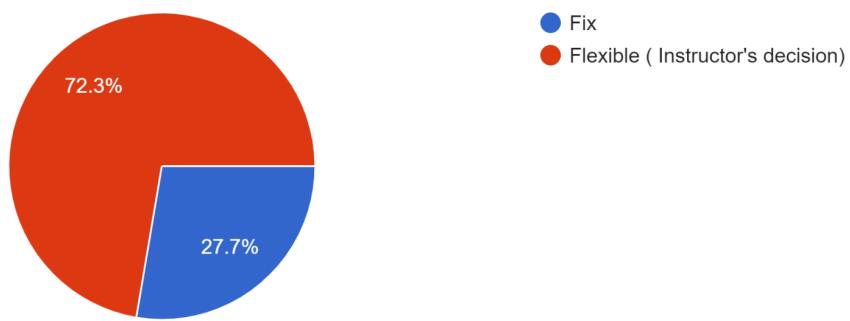
**Observation from the response:**

- The majority of the responders suggest that scale should be flexible so we will provide the feature of entering the scale of academic activities.

**Question-4:**

The weightage of non-academic components(Music, Sports, Coding etc.) should be fixed or flexible according to you?

47 responses

**Intent of this question:**

- To get to know if the weightage scale of non-academic activities should be fixed or flexible.

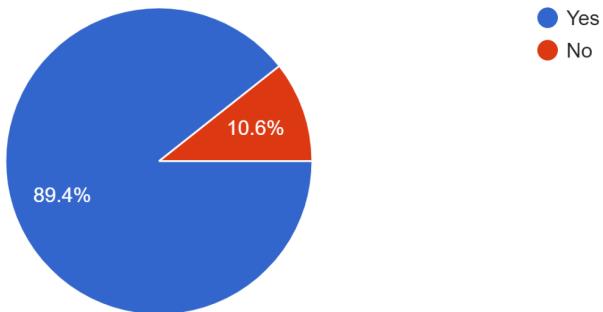
**Observation from the response:**

- The majority of the responders suggest that scale should be flexible so we will provide the feature of entering the scale of non-academic activities.

**Question-5:**

Do you believe that general leaderboard will help the students for creating healthy competition among them?

47 responses

**Intent of this question:**

- To get to know if the general leaderboard would be useful or not.

**Observation from the response:**

- The majority of the responders suggest that scale would be useful and it will create a positive impact so we will make a general leaderboard for all the students.

**List the combined Requirements from Questionaries:**

- Instructor needs a simple procedure for entries of attendance , marks and points of non-academic activities.
- different academic activities should have their corresponding weightage and same goes for non-academic activities for calculating combined performance of a student.
- Instructors should be able to choose the scales of each component for evaluating final performance.
- Students can be sorted according to their performance to give rewards to them.
- Students want to have a system in which they are able to get information about their attendance, performance of each activity (academic and non-academic).
- This system will also ensure transparency in grading and students will be able to know how much and in which direction they need to improve their performance.
- Each student should have their personal view of their performance records.
- The leaderboards should be visible to all students so that they can compare their performance.
- an alert system which will notify the parents/ keep a log of all activity of students with activity type tags,

- an alert which will notify the parents if attendance goes down below 75 percent.

## 2.4 Observations

## **System :** University database

**Project Reference : SF/SJ/2003/12**

**Observations by:** Aditya Jadeja      **Designation:** System Developer

Het Patel      **Designation:**      System Developer

Date: 04/10/2022 Time: 11:00

**Duration:** 45 minutes      **Place:** Faculty office

## Observations:

- Real-time updating of all student's attendance in every course lecture session.
  - Not all data was accessible to everyone, privacy and security of the data were given priority.
  - Instructor can see only the details of those students, who have registered for that particular instructor's course.
  - Instructors and students are able to see their profile anywhere on site.
  - Instructors are able to update final grades for the course only. So All instructor's are maintaining separate marks spreadsheets for lab exam,viva,mid semester exam,Final semester exam and quiz etc.
  - Having a proper internet connection is very important.

#### **Combined requirements from Observations:**

- The database is updated in real-time.
  - The instructor will be able to view their respective students and update their attendance and marks using an effortless interface.
  - Different user classes should be given different authorization to access or update the system.
  - All the users can connect to the database server from anywhere and have access to their information.
  - The database contains sensitive information of all the students and staff. Therefore, optimal security measures must be taken to ensure data is safe from unauthorized users.

### 3. Fact Finding Chart

Objective	Technique	Subject	Time Commitment
To get background knowledge on the Student Reward System	Background readings	Similar projects on internet, websites	1 day
To know what functions instructors will need at their dashboard.	Interview	Instructor(Roleplay)	1 hour
To know how to calculate overall performance of a student	Interview, Qutionary	Instructor(Roleplay), Students	1 day
To know what functions students will need at their dashboard.	Interview	Any student	1 hour
To know how to communicate with parents about the performance of their child.	Interview, Qutionary	Any parent, students	1 day
To get to know if the weightage scale of academic activities or non-academic should be fixed or flexible.	Qutionary	Students	1 day

## 4. List Requirements

- Instructors should be able to choose the scales of each component for evaluating final performance.(Interview, Questionnaire)
- The leaderboards should be visible to all students so that they can compare their performance and which will lead to health competition among students. Each student will be able to see each component or course performance leaderboard as well.(Interview, Questionnaire)
- An alert system which will notify the parents/ keep a log of all activity of students with activity type tags and attendance log also, which will ensure ease for parents to stay updated with minimal efforts.(Interview, Questionnaire)
- Students shall be able to communicate with their instructor by sending personal messages.(Background reading)
- Students want to have a system in which they are able to get information about their attendance, performance of each activity (academic and non-academic). (Interview)
- This system will also ensure transparency in grading and students will be able to know how much and in which direction they need to improve their performance.(Interview)
- Different user classes should be given different authorization to access or update the system.(Observation)
- All the users can connect to the database server from anywhere and have access to their information.(Observation)
- The database contains sensitive information of all the students and staff. Therefore, optimal security measures must be taken to ensure data is safe from unauthorized users.(Observation)

## 5. User Classes and Characteristics

### List of the user-class:

- **Admin:** Admin will have access to the entire database and will be in charge of managing it.
- **Students:** Students can access the database to view their personal data, performance, and rewards.
- **Teachers:** Teachers will be able to view and update each student's details, as well as add attendance, choose performance scale, and assign rewards.
- **Parents:** parents will receive notifications on every activity of their child and monitor progress.

### List of the functions which can be used by different user classes:

#### 1. Admin:

- Add course
- Add instructor
- Update instructor
- Remove course
- Remove instructor
- All the functions which can be accessed by students,instructors,and parents.

#### 2. Instructor:

- Add student
- Add student info
- Update student info
- Remove old student
- Remove old student info
- Add reward
- Removes reward
- Give academic reward
- Update academic reward info
- Remove academic reward
- Give non academic reward
- Update non academic reward info
- Remove non academic reward
- Add scale to course
- Update scale to course
- Remove course scale
- Add scale for non academic activity
- Update non academic activity
- Removes non academic scale
- Add overall performance scale.
- Removes the performance scale
- Alert parent

- Add alert

**3. Student:**

- View student profile
- Check attendance
- Check Rewards
- View Leaderboard

**4. Parents:**

- Check alerts log
- All the functions which can be accessed by students

## 6. Assumption

- The hardware and software required to execute this application are considered to be available to users.
- It is also believed that the database's data is constantly updated with the right information.
- The users will have access to alternate resources in the event of any unpreventable circumstance.
- Users must be able to use a computer and have a basic understanding of how the system works.

## 7. Business Constraints

- If the user pool is quite huge, the aforementioned database's scalability may become a problem. For it, distributed database systems are required.
- The information must be kept on a server. It could be costly to store big volumes of data on cloud servers.
- Highly scaled systems require specialized employees to handle and maintain the entire system.

## Noun & Verb Analysis

Sr. No.	Noun	Verbs
1	Reward	motivate
2	Student	focus
3	semester	study
4	academic	think
5	non-academic	given
6	points	contain
7	activity	include
8	questions	Gets
9	answers	can
10	lecture	Has
11	quiz	likewise
12	exam	see
13	teachers	will
14	progress	to
15	dashboard	manage
16	class	learning
17	prize	Notify
18	information	decide
19	database	takes

20	system	access
21	way	Decides
22	details	add
23	interface	alert
24	organization	update
25	behavior	Has
26	parents	be
27	scheme	Observes
28	Login id	exist
29	Password	Check
30	performance	Remove
31	test score	maintains
32	Reward history	Provide
33	result	receives
34	attendance	notify
35	child	Able
36	student_info	
37	instructor	
38	course	
39	Student name	
40	Student id	
41	Phone no	
42	address	

43	marks	
44	Course id	
45	Instructor id	
46	Course name	
47	Instructor name	
48	Reward type	
49	Activity type	
50	Academic reward	
51	Non-academic reward	
52	Obtained marks	
53	Total marks	
54	Reward points	
55	Scale of academic performance	
56	Scale of non-academic performance	
57	scale	
58	Scale of combined performance	
59	Parent name	
60	Parent email	
61	Academic scale	
62	Non-academic scale	
63	alert	

64	Alert type	
65	Alert message	
66	admin	
67	Admin name	
68	Admin id	

## 2. Entity - Attribute:

Candidate entity set	Candidate attribute set	Candidate relationship set
Student	<u>Student ID</u> , student name, phone no, address	Teaches,Has_parent, Receives, Gets, Student_Course,Observes
Instructor	<u>Instructor ID</u> , instructor name, course ID, phone no, address	Teaches, Assigns ,Maintains , Decides,Takes
Course	Instructor name, <u>course id</u> , course name, instructor id	Takes , Notify , Contains,Student_Course, Course rewards
Reward	<u>reward name</u> , <u>course id</u> , instructor id,Reward type	Contains
Academic Rewards	<u>Student_ID</u> , <u>course_ID</u> , <u>Activity type</u> , obtained marks, total marks, Reward points	Assigns , Academic performance ,Course rewards
Non-acadamic_Rewards:	<u>Student_ID</u> , <u>Activity type</u> , Instructor IDReward points	Assigns, Non-academic rewards
Scale of Academic Performance	<u>instructor_ID</u> , <u>course_ID</u> , <u>activity type</u> , scale.	Course scale,decides
Scale of Non- Academic Performance	<u>instructor_ID</u> , <u>activity type</u> , scale.	Decides
Scale of combined Performance	<u>Instructor_ID</u> , academic scale,non- academic scale	Decides
Performance	<u>student_ID</u> ,overall score	Academic performance, non-academic performance
Parent	<u>student ID</u> , parent name, parent email,	Has_parent

Alert	<u>Alert no.</u> (student ID, course id,date), alert type , alert message,	Notify, receives
Admin	<u>admin id</u> ,admin name	Maintains

### 3. Rejected Nouns and Verbs :

Noun	Reason
semester	Attribute
Reward points	Attribute
activity	Duplicate
questions	irrelevant
answers	irrelevant
lecture	irrelevant
quiz	Duplicate
exam	Duplicate
teachers	Duplicate
progress	duplicate
dashboard	irrelevant
class	duplicate
prize	duplicate
information	general
database	general
system	general

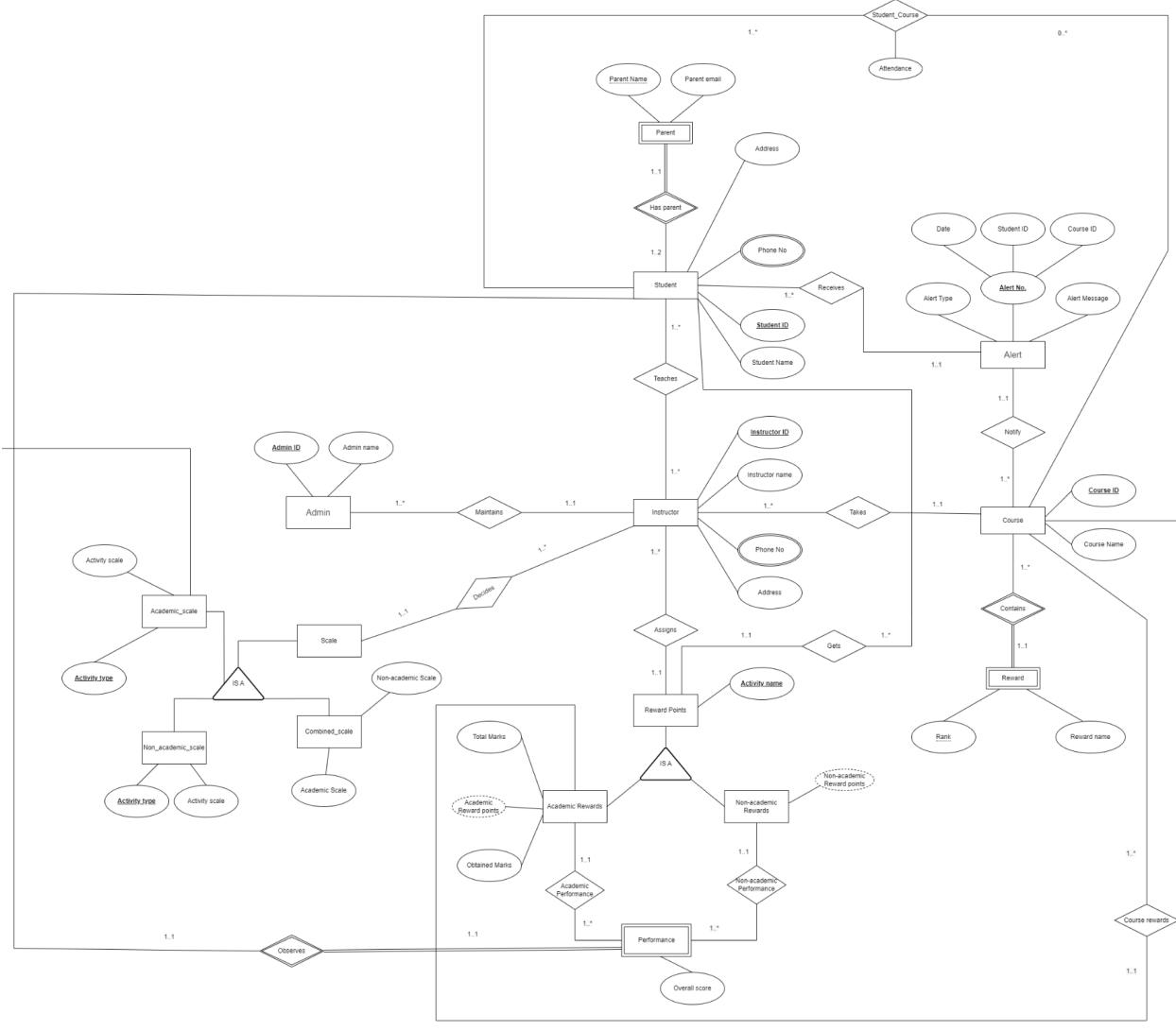
way	vague
details	general
interface	general
organization	irrelevant
behavior	irrelevant
scheme	vague
Login id	attribute
Password	attribute
test score	Duplicate
Reward history	Irrelevant
result	general
attendance	attribute
child	irrelevant
Student name	attribute
Student id	attribute
Phone no	attribute
address	attribute
marks	attribute
Course id	attribute
Instructor id	attribute
Course name	attribute
Instructor name	attribute
Reward type	attribute

Activity type	attribute
Obtained marks	attribute
Total marks	attribute
Reward points	attribute
scale	attribute
Parent name	attribute
Parent email	attribute
Academic scale	attribute
Non-academic scale	attribute
Alert type	attribute
Alert message	attribute
Admin name	attribute
Admin id	attribute

Verbs	Reason
motivate	Irrelevant
focus	Irrelevant
study	General
think	Irrelevant
given	General
contain	vague
include	duplicate
get	general

can	vague
be	vague
likewise	vague
see	general
will	vague
provide	general
manage	general
learning	general
track	irrelevant
decide	irrelevant
able	vague
access	general
check	general
add	association
alert	general
update	association
remove	association
exist	vague

## ❖ ER Diagram (Final Version):



## ❖ Mapping ER model to Relational model:

1. **Student(Student\_ID, student name, address)**  
Primary Key: Student\_ID
2. **Student\_phone(Student\_ID, Phone no)**  
Primary Key: (composite): Student ID, phone no  
Foreign key: Student\_ID references **Student**
3. **Instructor(Instructor\_ID, instructor name, address,Admin\_ID)**  
Primary Key: Instructor\_ID  
Foreign Key: Admin\_ID references **Admin**
4. **Instructor\_phone(Instructor\_ID, Phone no)**  
Primary Key: (composite): Instructor ID, phone no  
Foreign key: Instructor\_ID references **Instructor**
5. **Teches(Instructor\_ID, Student\_ID)**  
Primary Key: (composite) : Instructor ID, Student ID  
Foreign Key: Instructor ID references **Instructor**  
Foreign Key: Student ID references **Student**
6. **Course(Course\_ID, course name, Instructor\_ID)**  
Primary Key: Course\_ID  
Foreign Key: Instructor\_ID references **Instructor**
7. **Sudent\_course(student\_ID, Course\_ID, Attendance)**  
Primary Key: (composite) : Student\_ID, Course\_ID.  
Foreign Key: Student ID references **Student**  
Foreign Key: Course ID references **Course**
8. **Reward(Rank,Reward\_name)**  
Primary Key: : Rank

9. **Academic\_rewards**(Student\_ID, Course\_ID, Activity\_name, obtained\_marks, Total\_marks, Reward\_points, Instructor\_ID)  
 Primary Key: (composite) : Student\_ID, Course\_ID, activity\_name  
 Foreign Key: Student\_ID references **Student**  
 Foreign Key: Course\_ID references **Course**  
 Foreign Key: Instructor\_ID references **Instructor**
10. **Non\_academic\_rewards**(Student\_ID, Activity\_name, ,Instructor ID,Reward points)  
 Primary Key: (composite) : Student\_ID,Activity\_name  
 Foreign Key: Student\_ID references **Student**  
 Foreign Key: Instructor\_ID references **Instructor**
11. **Academic\_scale**(Instructor\_ID , Course\_ID , Activity\_type, scale)  
 Primary Key: (composite) : Instructor\_ID, Course\_ID, activity\_type  
 Foreign Key: Instuctor\_ID references **Instructor**  
 Foreign Key: Course\_ID references **Course**
12. **Non\_academic\_scale**(Instructor\_ID , Activity\_type, scale)  
 Primary Key: (composite) : Instructor\_ID, activity\_type  
 Foreign Key: Instuctor\_ID references **Instructor**
13. **Combined\_scale**(Instructor\_ID , academic scale,non- academic scale)  
 Primary Key : Instructor\_ID  
 Foreign Key: Instuctor\_ID references **Instructor**
14. **Performance**(Student\_ID ,overall score)  
 Primary Key : Student\_ID  
 Foreign Key: Student\_ID references **Student**
15. **Parent**(Student\_ID , parent\_name, parent email)  
 Primary Key: (composite) Student\_ID, parent name  
 Foeign key: Student\_ID references **Student**

16. **Alert**(Student\_ID, Course\_ID,date, alert type , alert message)

Primary Key: student ID, Course\_ID,date

Foeign key: Student\_ID references **Student**

Foeign key: Course\_ID references **Course**

17. **Admin**(Admin\_ID ,admin name)

Primary Key: Admin\_ID

### ❖ Functional Dependencies:

1. **Student**(Student\_ID → student name,  
Student\_ID → address)

2. **Student\_phone** : No Dependencies

3. **Instructor**(Instructor\_ID → instructor name,  
Instructor\_ID→address,  
Instructor\_ID →Admin\_ID)

4. **Instructor\_phone**: No Dependencies

5. **Teches** : No Dependencies

6. **Course**(Course\_ID→course name,  
Course\_ID→ Instructor\_ID)

7. **Sudent\_course**(Student\_ID, Course\_ID → Attendance)

8. **Reward**:No Dependencies

9. **Academic\_rewards**(

Student\_ID, Course\_ID, Activity\_name→ obtained\_marks,  
Course\_ID, Activity\_name→ Total\_marks,  
Student\_ID, Course\_ID, Activity\_name→ Reward\_points)

10. **Non\_academic\_rewards(**

Student\_ID, Activity\_name → Instructor ID,  
 Student\_ID, Activity\_name → Reward points)

11. **Academic\_scale(** Course\_ID , Activity\_type → scale)12. **Non\_academic\_scale(**Instructor\_ID , Activity\_type → scale)13. **Combined\_scale(**Instructor\_ID → academic scale,  
 Instructor\_ID → non-academic scale)14. **Performance(**Student\_ID → overall score)15. **Parent(**Student\_ID , parent\_name → parent email)16. **Alert(**Student\_ID, Course\_ID, date → alert type ,  
 Student\_ID, Course\_ID, date → alert message)17. **Admin(**Admin\_ID → admin name)

❖ **Redundancies and anomalies:**

1. **Student(**Student\_ID, student\_name, address)

- **Redundancy:** it is possible to have the same student\_name multiple times with different Student\_IDs.
- There are no anomalies in this relation.

2. **Student\_phone(**Student\_ID, Phone\_no)

- Foreign key: Student\_ID references **Student**
- **Redundancy:** It is possible to have the same student\_ID multiple times with different Phone\_no.
- There are no anomalies in this relation.

3. **Instructor(Instructor\_ID, instructor name, address,Admin\_ID)**

- Foreign Key: Admin\_ID references **Admin**
- **Redundancy:** It is possible to have the same Instructor\_name multiple times with different Student\_IDs.
- There are no anomalies in this relation.

4. **Instructor\_phone(Instructor\_ID, Phone no)**

- Foreign key: Instructor\_ID references **Instructor**
- **Redundancy:** It is possible to have the same Instructor\_ID multiple times with different Phone\_no.
- There are no anomalies in this relation.

5. **Teches(Instructor\_ID, Student\_ID)**

- Foreign Key: Instructor ID references **Instructor**
- Foreign Key: Student ID references **Student**
- There are no anomalies in this relation.
- **Redundancy:** It is possible to have the same Instructor\_ID multiple times with different Student\_ID and same Student\_ID multiple times with different Instructor\_ID.

6. **Course(Course\_ID, course name, Instructor\_ID)**

- Foreign Key: Instructor\_ID references **Instructor**
- **Redundancy:** It is possible to have the same Instructor\_ID multiple times with different Course\_ID.
- There are no anomalies in this relation.

7. **Sudent\_course(student\_ID, Course\_ID, Attendance)**

- Foreign Key: Student ID references **Student**
- Foreign Key: Course ID references **Course**
- There are no anomalies in this relation.

8. **Reward(Reward\_name, Course\_ID, Reward type)**

- Foreign Key: Course\_ID references **Course**
- There are no anomalies in this relation.

9. **Academic\_rewards**(Student\_ID, Course\_ID, Activity\_name, obtained\_marks, Total\_marks, Reward\_points)
- Foreign Key: Student\_ID references **Student**
  - Foreign Key: Course\_ID references **Course**
  - **FD =** (Course\_ID, Activity\_name → Total\_marks)
  - This is NOT in 2NF as it has a partial dependency.
  - **Anomalies:**
    - **Insertion:** We cannot add Total\_marks until it is associated with any Student\_ID.
    - **Deletion:** If you delete the only Student\_ID , you lose data about the Total marks about particular activity of course.
    - **Update:** Total\_marks is repeated for every activity\_name for a particular course.
10. **Non\_academic\_rewards**(Student\_ID, Activity\_name, ,Instructor ID,Reward points)
- Foreign Key: Student\_ID references **Student**
  - Foreign Key: Instructor\_ID references **Instructor**
  - There are no anomalies in this relation.
11. **Academic\_scale**( Course\_ID , Activity\_type, scale)
- Foreign Key: Course\_ID references **Course**
  - There are no anomalies in this relation.
  - **Redundancy:** It is possible to have the same scale multiple times with different activities in the same Course\_ID.
12. **Non\_academic\_scale**(Instructor\_ID , Activity\_type, scale)
- Foreign Key: Instuctor\_ID references **Instructor**
  - **Redundancy:** It is possible to have the same scale multiple times with different activities of the same Instructor\_ID.
  - There are no anomalies in this relation.

13. **Combined\_scale(Instructor\_ID, academic scale,non- academic scale)**

- Foreign Key: Instructor\_ID references **Instructor**
- There are no anomalies in this relation.

14. **Performance(Student\_ID ,overall score)**

- Foreign Key: Student\_ID references **Student**
- There are no anomalies in this relation.
- **Redundancy:** It is possible to have the same overall\_score multiple times with different Student\_ID

15. **Parent(Student\_ID , parent\_name, parent email)**

- Foeign key: Student\_ID references **Student**
- There are no anomalies in this relation.

16. **Alert(Student\_ID, Course\_ID,date, alert type , alert message)**

- Foeign key: Student\_ID references **Student**
- Foeign key: Course\_ID references **Course**
- There are no anomalies in this relation.

17. **Admin(Admin\_ID ,admin name)**

- **Redundancy:** It is possible to have the same admin\_name multiple times with different Admin\_ID.
- There are no anomalies in this relation.

## ❖ Normalization of Schema up to 3NF/BCNF:

1. **Student(Student\_ID, student\_name, address)**
  - Since every attribute value is atomic the relation is in 1NF.
  - Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
  - Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF
2. **Student\_phone(Student\_ID, Phone no)**
  - Since this is all key relation, all dependencies are trivial dependencies.
  - So this relation is already in BCNF, This implies that it is in 3NF.
3. **Instructor(Instructor\_ID, instructor name, address, Admin\_ID)**
  - Since every attribute value is atomic the relation is in 1NF.
  - Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
  - Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF
4. **Instructor\_phone(Instructor\_ID, Phone no)**
  - Since this is all key relation, all dependencies are trivial dependencies.
  - So this relation is already in BCNF, This implies that it is in 3NF.
5. **Teches(Instructor\_ID, Student\_ID)**
  - Since every attribute value is atomic the relation is in 1NF.
  - Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
  - Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

6. **Course(Course\_ID, course name, Instructor\_ID)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

7. **Sudent\_course(student\_ID, Course\_ID, Attendance)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

8. **Reward(Rank,Reward\_name)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

9. **Academic\_rewards(Student\_ID, Course\_ID, Activity\_name,**

obtained\_marks, Total\_marks, Reward\_points)

- Since every attribute value is atomic the relation is in 1NF.
- There is one partial dependency so we will decompose the table as follows to get it into 2NF.
  - **Academic\_rewards(Student\_ID, Course\_ID, Activity\_name,**
  - obtained\_marks, Reward\_points)
  - **Total\_marks(Course\_ID, Activity\_name,** Total\_marks)
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

10. **Non\_academic\_rewards(Student\_ID, Activity\_name,**

Instructor ID,Reward points)

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

11. **Academic\_scale( Course\_ID , Activity\_type, scale)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

12. **Non\_academic\_scale(Instructor\_ID , Activity\_type, scale)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

13. **Combined\_scale(Instructor\_ID , academic scale,non- academic scale)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

14. **Performance(Student\_ID ,overall score)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

15. **Parent(Student\_ID , parent\_name, parent email)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

16. **Alert(Student\_ID, Course\_ID,date, alert type , alert message)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF

17. **Admin(Admin\_ID ,admin name)**

- Since every attribute value is atomic the relation is in 1NF.
- Since there is only one value in PK and also there is only one candidate key the relation is in 2NF.
- Every candidate key is on LHS of the functional dependencies, so the relation is in BCNF. This implies that the relation is also in 3NF.

## ❖ List of Final relations with schema:

1. **Student(Student\_ID, student\_name, address)**
2. **Student\_phone(Student\_ID, Phone no)**
3. **Instructor(Instructor\_ID, instructor name, address,Admin\_ID)**
4. **Instructor\_phone(Instructor\_ID, Phone no)**
5. **Teches(Instructor\_ID, Student\_ID)**
6. **Course(Course\_ID, course name, Instructor\_ID)**
7. **Student\_course(student\_ID, Course\_ID, Attendance)**
8. **Reward(Rank,Reward\_name)**
9. **Academic\_rewards(Student\_ID, Course\_ID, Activity\_name, obtained\_marks, Reward\_points)**
10. **Non\_academic\_rewards(Student\_ID, Activity\_name, Instructor ID,Reward points)**
11. **Academic\_scale( Course\_ID , Activity\_type, scale)**
12. **Non\_academic\_scale(Instructor\_ID , Activity\_type, scale)**
13. **Combined\_scale(Instructor\_ID , academic scale,non- academic scale)**
14. **Performance(Student\_ID ,overall score)**
15. **Parent(Student\_ID , parent\_name, parent email)**
16. **Alert(Student\_ID, Course\_ID,date, alert type , alert message)**
17. **Admin(Admin\_ID ,admin name)**
18. **Total\_marks(Course\_ID, Activity\_name, Total\_marks)**

## ❖ DDL Scripts and Data entries of each table :

### 1.Student

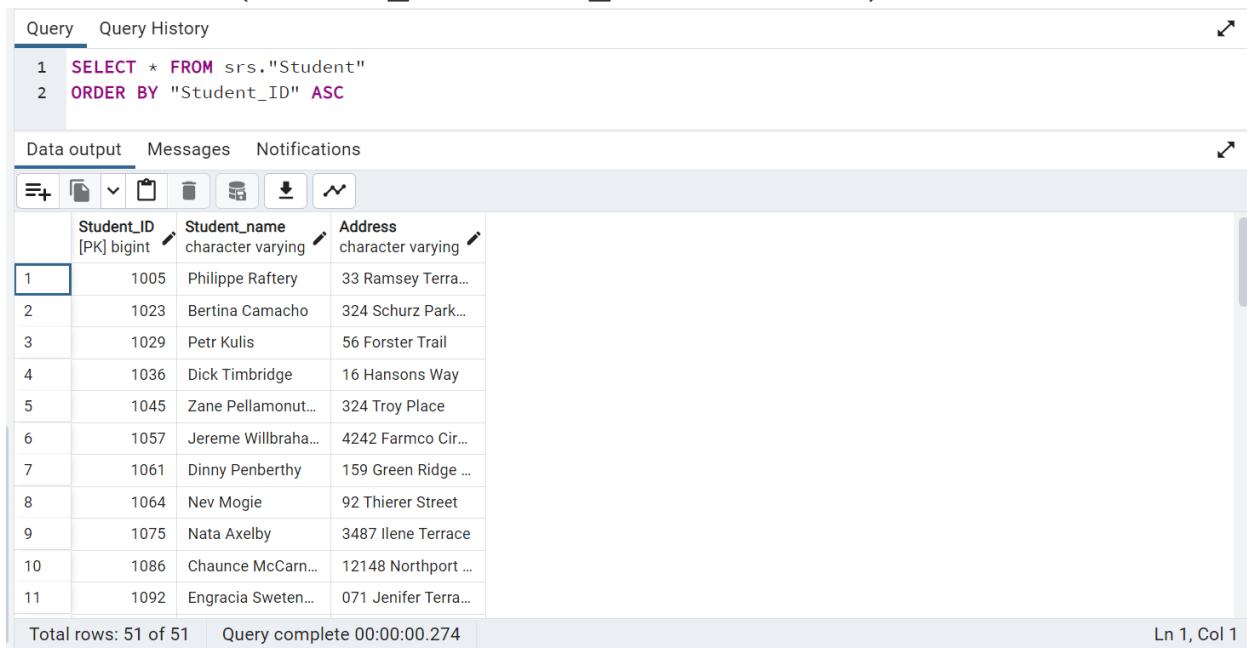
```

CREATE TABLE IF NOT EXISTS srs."Student"
(
    "Student_ID" bigint NOT NULL,
    "Student_name" character varying COLLATE pg_catalog."default",
    "Address" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Student_pkey" PRIMARY KEY ("Student_ID")
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Student"
OWNER to postgres;

```

- **Student(Student\_ID, student\_name, address)**



The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for 'Query' (which is selected) and 'Query History'. Below the tabs, the SQL query is displayed:

```

1  SELECT * FROM srs."Student"
2  ORDER BY "Student_ID" ASC

```

Below the query, there are tabs for 'Data output' (selected), 'Messages', and 'Notifications'. The data output section displays the results of the query as a table:

	Student_ID [PK] bigint	Student_name character varying	Address character varying
1	1005	Philippe Raftery	33 Ramsey Terra...
2	1023	Bertina Camacho	324 Schurz Park...
3	1029	Petr Kulis	56 Forster Trail
4	1036	Dick Timbridge	16 Hansons Way
5	1045	Zane Pellamonut...	324 Troy Place
6	1057	Jereme Willbraha...	4242 Farmco Cir...
7	1061	Dinny Penberthy	159 Green Ridge ...
8	1064	Nev Mogie	92 Thierer Street
9	1075	Nata Axelby	3487 Ilene Terrace
10	1086	Chounce McCarn...	12148 Northport ...
11	1092	Engracia Sweten...	071 Jenifer Terra...

Total rows: 51 of 51    Query complete 00:00:00.274    Ln 1, Col 1

## 2.Student\_phone :

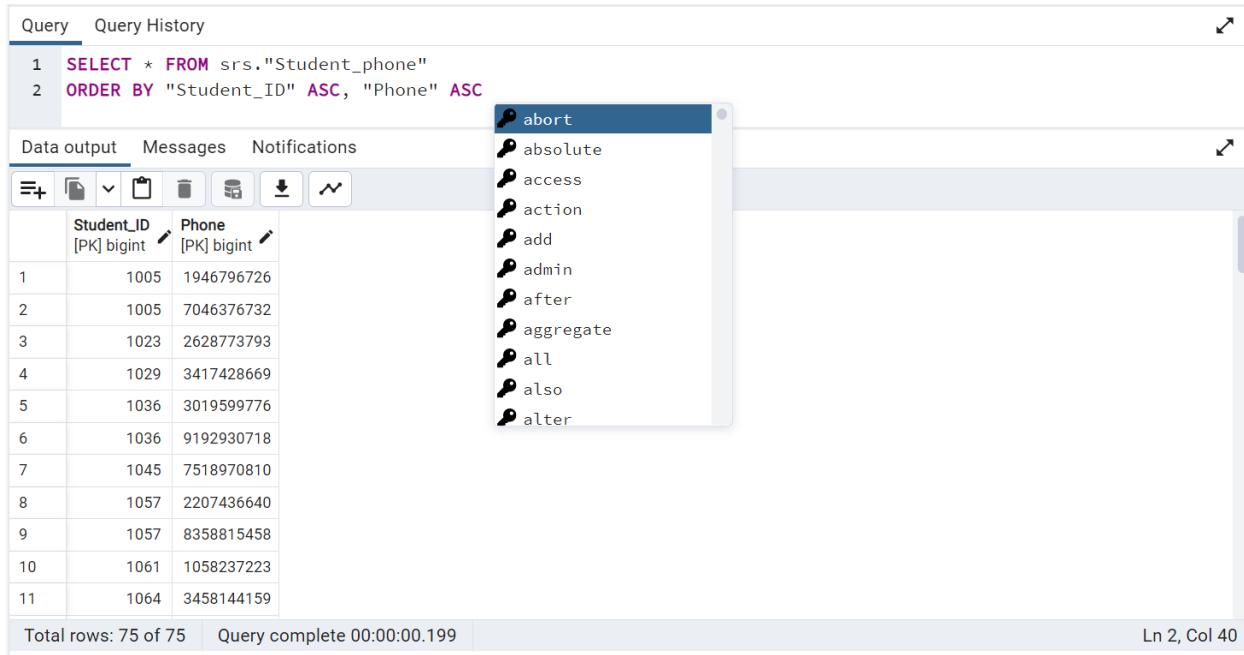
```

CREATE TABLE IF NOT EXISTS srs."Student_phone"
(
    "Student_ID" bigint NOT NULL,
    "Phone" bigint NOT NULL,
    CONSTRAINT "Student_phone_pkey" PRIMARY KEY ("Phone",
    "Student_ID"),
    CONSTRAINT "Student_ID" FOREIGN KEY ("Student_ID")
        REFERENCES srs."Student" ("Student_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Student_phone"
OWNER to postgres;

```

- **Student\_phone(Student\_ID, Phone no)**



The screenshot shows a PostgreSQL query tool interface. The top bar has tabs for 'Query' and 'Query History'. Below the tabs is a code editor with two lines of SQL:

```

1 SELECT * FROM srs."Student_phone"
2 ORDER BY "Student_ID" ASC, "Phone" ASC

```

Below the code editor are three buttons: 'Data output', 'Messages', and 'Notifications'. Underneath these buttons is a toolbar with icons for new query, copy, paste, delete, refresh, download, and search.

The main area displays a table with two columns: 'Student\_ID' and 'Phone'. The 'Student\_ID' column is marked as the primary key [PK]. The table contains 11 rows of data:

	Student_ID	Phone
1	1005	1946796726
2	1005	7046376732
3	1023	2628773793
4	1029	3417428669
5	1036	3019599776
6	1036	9192930718
7	1045	7518970810
8	1057	2207436640
9	1057	8358815458
10	1061	1058237223
11	1064	3458144159

At the bottom left, it says 'Total rows: 75 of 75' and 'Query complete 00:00:00.199'. At the bottom right, it says 'Ln 2, Col 40'. A context menu is open over the first row of the table, with the 'alter' option highlighted.

### 3.Instructor :

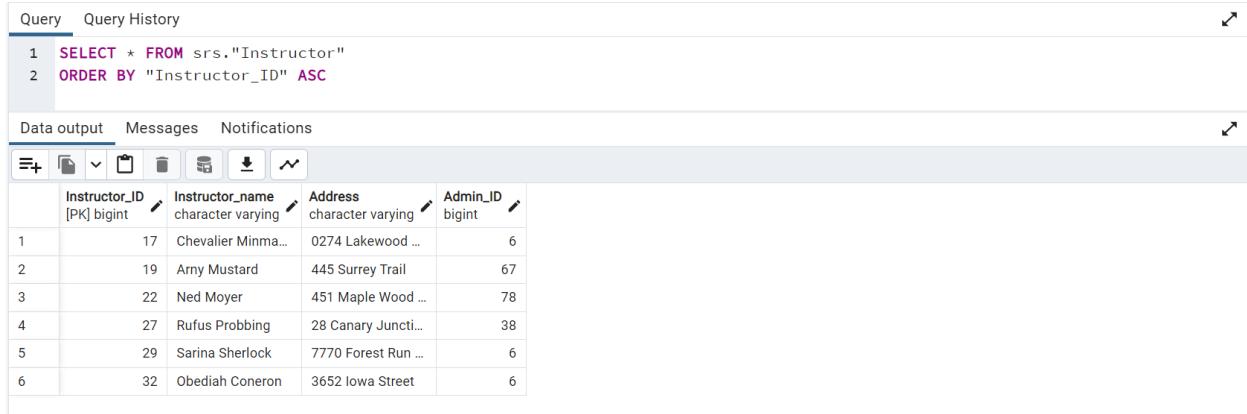
```

CREATE TABLE IF NOT EXISTS srs."Instructor"
(
    "Instructor_ID" bigint NOT NULL,
    "Instructor_name" character varying COLLATE pg_catalog."default"
NOT NULL,
    "Address" character varying COLLATE pg_catalog."default",
    "Admin_ID" bigint,
    CONSTRAINT "Instructor_pkey" PRIMARY KEY ("Instructor_ID"),
    CONSTRAINT "Admin_ID" FOREIGN KEY ("Admin_ID")
        REFERENCES srs."Admin" ("Admin_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Instructor"
OWNER to postgres;

```

- **Instructor(Instructor\_ID, instructor name, address,Admin\_ID)**



The screenshot shows a PostgreSQL query tool interface. The top section displays a SQL query:

```

1 SELECT * FROM srs."Instructor"
2 ORDER BY "Instructor_ID" ASC

```

The bottom section shows the resulting data output in a table format:

	Instructor_ID [PK] bigint	Instructor_name character varying	Address character varying	Admin_ID bigint
1	17	Chevalier Minima...	0274 Lakewood ...	6
2	19	Amy Mustard	445 Surrey Trail	67
3	22	Ned Moyer	451 Maple Wood ...	78
4	27	Rufus Probbing	28 Canary Juncti...	38
5	29	Sarina Sherlock	7770 Forest Run ...	6
6	32	Obediah Coneron	3652 Iowa Street	6

#### 4.Instructor\_phone :

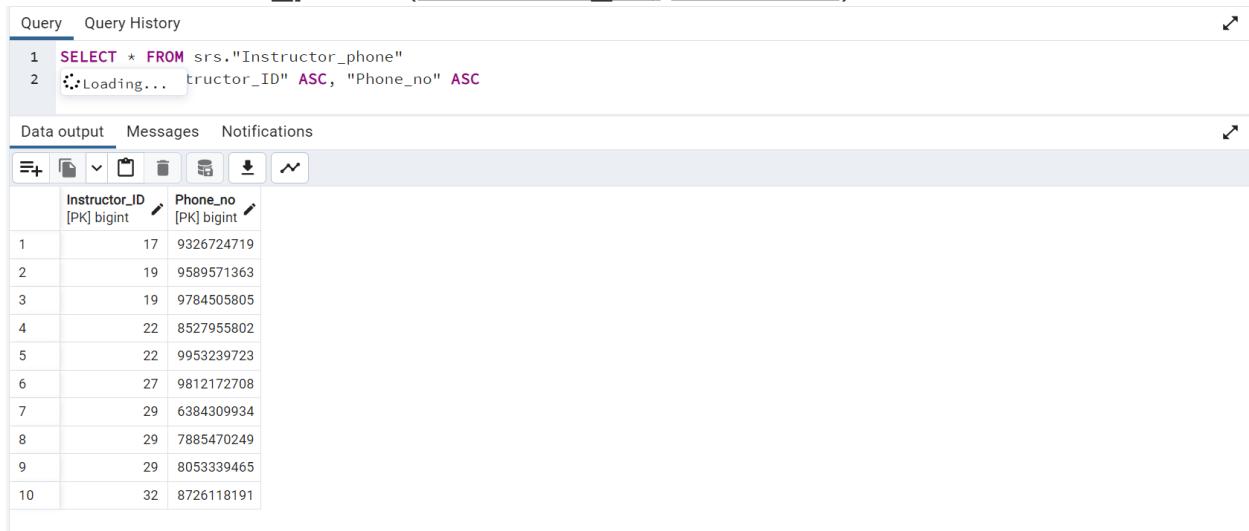
```

CREATE TABLE IF NOT EXISTS srs."Instructor_phone"
(
    "Instructor_ID" bigint NOT NULL,
    "Phone_no" bigint NOT NULL,
    CONSTRAINT "Instructor_phone_pkey" PRIMARY KEY
    ("Instructor_ID", "Phone_no"),
    CONSTRAINT "Instructor_ID" FOREIGN KEY ("Instructor_ID")
        REFERENCES srs."Instructor" ("Instructor_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Instructor_phone"
OWNER to postgres;

```

- **Instructor\_phone(Instructor\_ID, Phone no)**



The screenshot shows a PostgreSQL query tool interface. The top section displays a query window with the following content:

```

Query   Query History
1 SELECT * FROM srs."Instructor_phone"
2 Loading... "Instructor_ID" ASC, "Phone_no" ASC

```

The bottom section shows the results of the query in a data grid:

	Instructor_ID	Phone_no
1	17	9326724719
2	19	9589571363
3	19	9784505805
4	22	8527955802
5	22	9953239723
6	27	9812172708
7	29	6384309934
8	29	7885470249
9	29	8053339465
10	32	8726118191

## 5.Teaches :

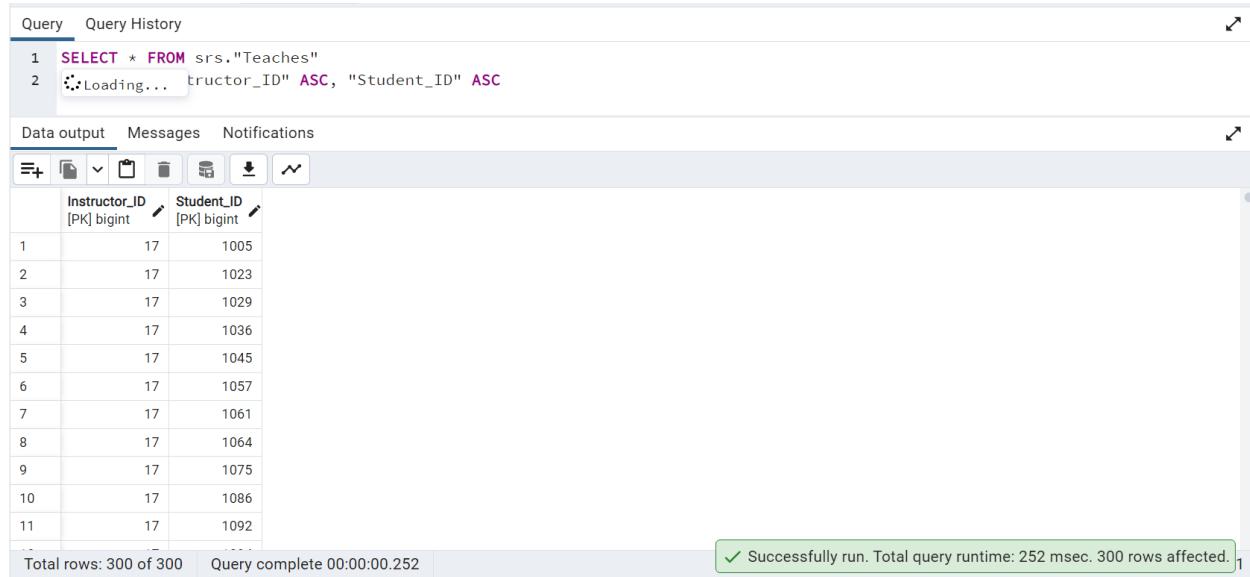
```

CREATE TABLE IF NOT EXISTS srs."Teaches"
(
    "Instructor_ID" bigint NOT NULL,
    "Student_ID" bigint NOT NULL,
    CONSTRAINT "Teaches_pkey" PRIMARY KEY ("Instructor_ID",
    "Student_ID"),
    CONSTRAINT "Instructor_ID" FOREIGN KEY ("Instructor_ID")
        REFERENCES srs."Instructor" ("Instructor_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT "Student_ID" FOREIGN KEY ("Student_ID")
        REFERENCES srs."Student" ("Student_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Teaches"
OWNER to postgres;

```

- Teches(Instructor\_ID, Student\_ID)



The screenshot shows a PostgreSQL query editor interface. The query window contains the following code:

```

1 SELECT * FROM srs."Teaches"
2 ⏺ Loading... "Instructor_ID" ASC, "Student_ID" ASC

```

The data output window displays the results of the query:

	Instructor_ID	Student_ID
1	17	1005
2	17	1023
3	17	1029
4	17	1036
5	17	1045
6	17	1057
7	17	1061
8	17	1064
9	17	1075
10	17	1086
11	17	1092

At the bottom of the interface, a status bar indicates:

Total rows: 300 of 300    Query complete 00:00:00.252    ✓ Successfully run. Total query runtime: 252 msec. 300 rows affected.

## 6. Course :

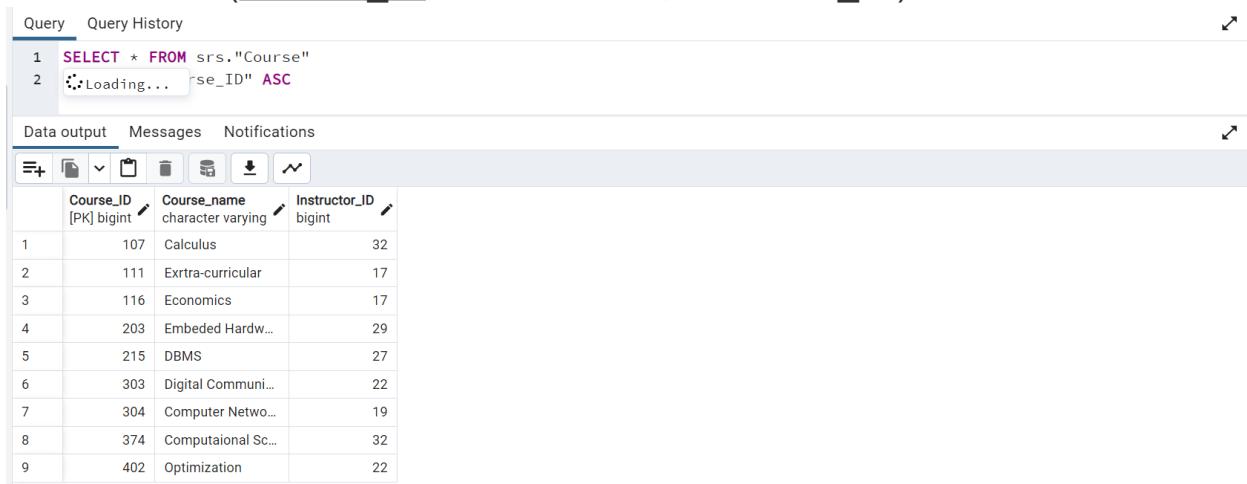
```

CREATE TABLE IF NOT EXISTS srs."Course"
(
    "Course_ID" bigint NOT NULL,
    "Course_name" character varying COLLATE pg_catalog."default" NOT
NULL,
    "Instructor_ID" bigint,
    CONSTRAINT "Course_pkey" PRIMARY KEY ("Course_ID"),
    CONSTRAINT "Instructor_ID" FOREIGN KEY ("Instructor_ID")
        REFERENCES srs."Instructor" ("Instructor_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Course"
OWNER to postgres;

```

- Course(Course\_ID, course name, Instructor\_ID)



The screenshot shows a PostgreSQL query tool interface. The top bar has tabs for 'Query' (selected) and 'Query History'. Below the tabs, there are two lines of SQL code:

```

1 SELECT * FROM srs."Course"
2 :Loading... Course_ID ASC

```

The bottom section is titled 'Data output' and contains a table with the following data:

	Course_ID [PK] bigint	Course_name character varying	Instructor_ID bigint
1	107	Calculus	32
2	111	Exrtra-curricular	17
3	116	Economics	17
4	203	Embedded Hardw...	29
5	215	DBMS	27
6	303	Digital Communi...	22
7	304	Computer Netwo...	19
8	374	Computaional Sc...	32
9	402	Optimization	22

## 7.Student\_course :

```

CREATE TABLE IF NOT EXISTS srs."Student_course"
(
    "Student_ID" bigint NOT NULL,
    "Course_ID" bigint NOT NULL,
    "Attendance" integer,
    CONSTRAINT "Student_course_pkey" PRIMARY KEY ("Student_ID",
    "Course_ID"),
    CONSTRAINT "Course_ID" FOREIGN KEY ("Course_ID")
        REFERENCES srs."Course" ("Course_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT "Student_ID" FOREIGN KEY ("Student_ID")
        REFERENCES srs."Student" ("Student_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

```

```

ALTER TABLE IF EXISTS srs."Student_course"
OWNER to postgres;

```

- **Student\_course(student\_ID, Course\_ID, Attendance)**

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for 'Query' (which is selected) and 'Query History'. Below the tabs, there are two numbered code snippets:

```

1 SELECT * FROM srs."Student_course"
2 ⏷ Loading... "Student_ID" ASC, "Course_ID" ASC

```

Below the code, there are three tabs: 'Data output' (selected), 'Messages', and 'Notifications'. The 'Data output' tab displays a table with the following data:

	Student_ID	Course_ID	Attendance
1	1005	107	0
2	1005	111	0
3	1005	203	0
4	1005	215	0
5	1005	303	0
6	1005	304	0
7	1005	374	0
8	1005	402	0
9	1023	107	0
10	1023	111	0
11	1023	203	0

At the bottom of the interface, there are status messages: 'Total rows: 360 of 360' and 'Query complete 00:00:00.298'. To the right, a green success message says 'Successfully run. Total query runtime: 298 msec. 360 rows affected.'

## 8.Reward :

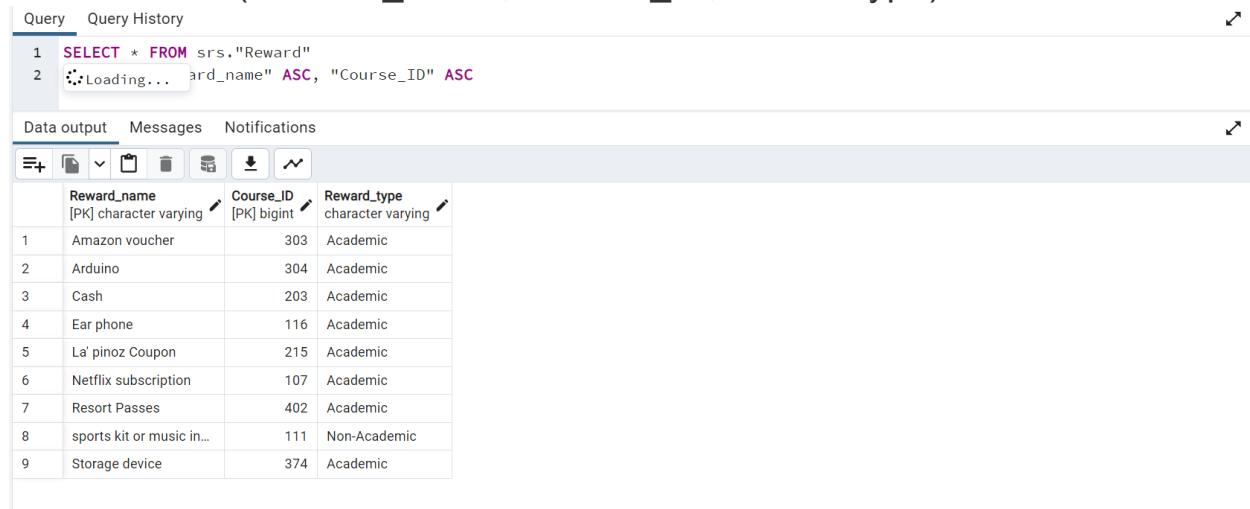
```

CREATE TABLE IF NOT EXISTS srs."Reward"
(
    "Reward_name" character varying COLLATE pg_catalog."default" NOT
NULL,
    "Course_ID" bigint NOT NULL,
    "Reward_type" character varying COLLATE pg_catalog."default",
        CONSTRAINT "Reward_pkey" PRIMARY KEY ("Reward_name",
"Course_ID"),
    CONSTRAINT "Course_ID" FOREIGN KEY ("Course_ID")
        REFERENCES srs."Course" ("Course_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Reward"
OWNER to postgres;

```

- Reward(Reward\_name, Course\_ID, Reward type)



The screenshot shows a PostgreSQL query editor interface with the following details:

- Query Tab:** Contains the SQL code for creating and altering the 'Reward' table.
- Data output Tab:** Displays the contents of the 'Reward' table.
- Table Data:** The 'Reward' table has three columns: 'Reward\_name' (character varying), 'Course\_ID' (bigint), and 'Reward\_type' (character varying). The data is as follows:

	Reward_name	Course_ID	Reward_type
1	Amazon voucher	303	Academic
2	Arduino	304	Academic
3	Cash	203	Academic
4	Ear phone	116	Academic
5	La' pinoy Coupon	215	Academic
6	Netflix subscription	107	Academic
7	Resort Passes	402	Academic
8	sports kit or music in...	111	Non-Academic
9	Storage device	374	Academic

**9.Academic\_reward :**

```
CREATE TABLE IF NOT EXISTS srs."Academic_reward"
(
    "Student_ID" bigint NOT NULL,
    "Course_ID" bigint NOT NULL,
    "Activity_name" character varying COLLATE pg_catalog."default" NOT
NULL,
    "Obtained_marks" double precision,
    "Reward_points"double precision,
    CONSTRAINT "Academic_reward_pkey" PRIMARY KEY
("Student_ID", "Course_ID", "Activity_name"),
    CONSTRAINT "Course_ID" FOREIGN KEY ("Course_ID")
        REFERENCES srs."Course" ("Course_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT "Student_ID" FOREIGN KEY ("Student_ID")
        REFERENCES srs."Student" ("Student_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Academic_reward"
OWNER to postgres;
```

- Academic\_rewards(Student\_ID, Course\_ID, Activity\_name, obtained\_marks, Reward\_points)

Query    Query History

```
1 SELECT * FROM srs."Academic_reward"
2 ORDER BY "Student_ID" ASC, "Course_ID" ASC, "Activity_name" ASC
```

Data output    Messages    Notifications

Total rows: 1000 of 1550    Query complete 00:00:00.179    Ln 1, Col 1

## 10. Non\_academic\_reward :

```
CREATE TABLE IF NOT EXISTS srs."Non_academic_reward"
(
    "Student_ID" bigint NOT NULL,
    "Activity_name" character varying COLLATE pg_catalog."default" NOT
NULL,
    "Instructor_ID" bigint,
    "Reward_points" double precision,
    CONSTRAINT "Non_academic_reward_pkey" PRIMARY KEY
("Student_ID", "Activity_name"),
    CONSTRAINT "Instructor_ID" FOREIGN KEY ("Instructor_ID")
        REFERENCES srs."Instructor" ("Instructor_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT "Student_ID" FOREIGN KEY ("Student_ID")
        REFERENCES srs."Student" ("Student_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
```

```
TABLESPACE pg_default;
```

```
ALTER TABLE IF EXISTS srs."Non_academic_reward"
OWNER to postgres;
```

- **Non\_academic\_rewards(Student\_ID, Activity\_name, ,Instructor ID,Reward points)**

Query    Query History

```
1 SELECT * FROM srs."Non_academic_reward"
2 ⏷ Loading... "Student_ID" ASC, "Activity_name" ASC
```

Data output    Messages    Notifications

	Student_ID [PK] bigint	Activity_name [PK] character varying	Instructor_ID bigint	Reward_points double precision
1	1005	Coding	17	0
2	1005	Dance	17	0
3	1005	Drama	17	20
4	1005	Music	17	0
5	1005	Sports	17	20
6	1023	Coding	17	0
7	1023	Dance	17	20
8	1023	Drama	17	20
9	1023	Music	17	0
10	1023	Sports	17	20
11	1029	Coding	17	0

Total rows: 250 of 250    Query complete 00:00:00.195    Ln 1, Col 1

## 11. Academic\_scale :

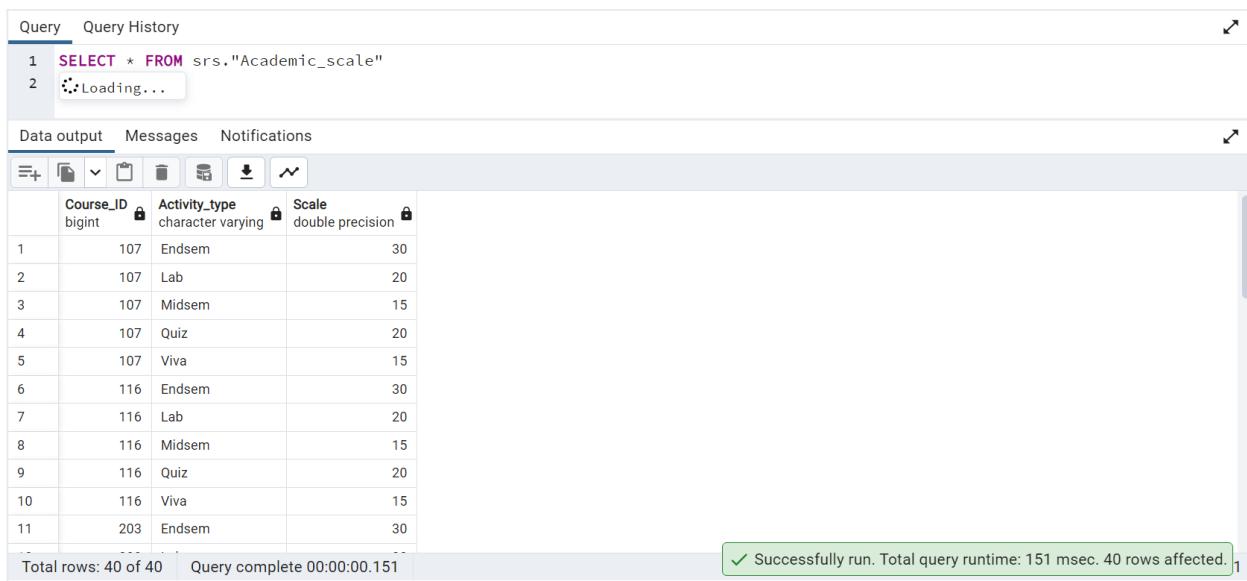
```

CREATE TABLE IF NOT EXISTS srs."Academic_scale"
(
    "Course_ID" bigint NOT NULL,
    "Activity_type" character varying COLLATE pg_catalog."default" NOT
NULL,
    "Scale" double precision,
    CONSTRAINT "Course_ID" FOREIGN KEY ("Course_ID")
        REFERENCES srs."Course" ("Course_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Academic_scale"
OWNER to postgres;

```

- Academic\_scale( Course\_ID , Activity\_type, scale)



The screenshot shows a PostgreSQL query tool interface. The top navigation bar has tabs for 'Query' (which is selected) and 'Query History'. Below the tabs, there are two numbered rows: '1 SELECT \* FROM srs."Academic\_scale"' and '2 Loading...'. The main area is titled 'Data output' and contains a table with the following data:

	Course_ID	Activity_type	Scale
1	107	Endsem	30
2	107	Lab	20
3	107	Midsem	15
4	107	Quiz	20
5	107	Viva	15
6	116	Endsem	30
7	116	Lab	20
8	116	Midsem	15
9	116	Quiz	20
10	116	Viva	15
11	203	Endsem	30

At the bottom of the interface, it says 'Total rows: 40 of 40' and 'Query complete 00:00:00.151'. A green success message box indicates 'Successfully run. Total query runtime: 151 msec. 40 rows affected.'

## 12. Non\_academic\_scale :

```

CREATE TABLE IF NOT EXISTS srs."Non_academic_scale"
(
    "Instructor_ID" bigint NOT NULL,
    "Activity_type" character varying COLLATE pg_catalog."default" NOT
NULL,
    "Scale" double precision,
    CONSTRAINT "Non_academic_scale_pkey" PRIMARY KEY
("Instructor_ID", "Activity_type"),
    CONSTRAINT "Instructor_ID" FOREIGN KEY ("Instructor_ID")
        REFERENCES srs."Instructor" ("Instructor_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Non_academic_scale"
OWNER to postgres;

```

- **Non\_academic\_scale(Instructor\_ID , Activity\_type, scale)**

The screenshot shows a PostgreSQL query editor interface. At the top, there is a 'Query' tab and a 'Query History' tab. Below the tabs, the SQL code for creating the table is displayed. A progress bar indicates that the query is 'Loading...'. The table structure is shown below the code. In the bottom half of the interface, there is a 'Data output' tab which is currently selected, showing a table with five rows of data.

	Instructor_ID [PK] bigint	Activity_type [PK] character varying	Scale double precision
1	17	Coding	20
2	17	Dance	20
3	17	Drama	20
4	17	Music	20
5	17	Sports	20

### **13.Combined\_scale :**

```

CREATE TABLE IF NOT EXISTS srs."Combined_scale"
(
    "Instructor_ID" bigint NOT NULL,
    "Non_academic_scale" double precision,
    "Academic_scale" double precision,
    CONSTRAINT "Combined_scale_pkey" PRIMARY KEY
    ("Instructor_ID"),
    CONSTRAINT "Instructor_ID" FOREIGN KEY ("Instructor_ID")
        REFERENCES srs."Instructor" ("Instructor_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Combined_scale"
OWNER to postgres;

```

- **Combined\_scale(Instructor\_ID, academic scale,non- academic scale)**

Query    Query History

```

1 SELECT * FROM srs."Combined_scale"
2 ⏺ Loading... Instructor_ID" ASC

```

Data output    Messages    Notifications

	Instructor_ID [PK] bigint	Non_academic_scale double precision	Academic_scale double precision
1	22	40	60

## 14.Performance :

```

CREATE TABLE IF NOT EXISTS srs."Performance"
(
    "Student_ID" bigint NOT NULL,
    "Overall_score" double precision,
    CONSTRAINT "Performance_pkey" PRIMARY KEY ("Student_ID"),
    CONSTRAINT "Student_ID" FOREIGN KEY ("Student_ID")
        REFERENCES srs."Student" ("Student_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Performance"
OWNER to postgres;

```

- **Performance(Student\_ID ,overall score)**



The screenshot shows a PostgreSQL query editor interface. The query window contains:

```

1 SELECT * FROM srs."Performance"
2 ⏷ Loading... Student_ID" ASC

```

The data output window displays the following table:

	Student_ID	Overall_score
1	1005	179.2
2	1023	160.399999999999
3	1029	140.200000000000
4	1036	135.8
5	1045	172.700000000000
6	1057	149.7
7	1061	149.9
8	1064	150.099999999999
9	1075	141.399999999999
10	1086	153.1
11	1092	164.399999999999

At the bottom of the interface, status messages indicate:

- Total rows: 50 of 50
- Query complete 00:00:00.319
- Successfully run. Total query runtime: 319 msec. 50 rows affected.

## 15.Parent :

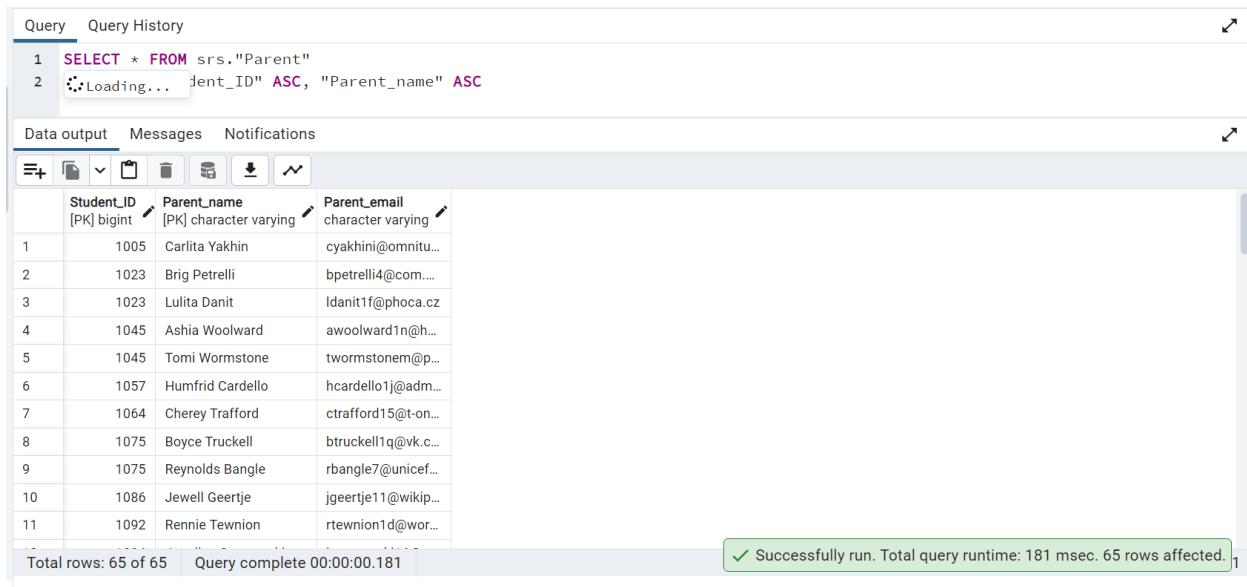
```

CREATE TABLE IF NOT EXISTS srs."Parent"
(
    "Student_ID" bigint NOT NULL,
    "Parent_name" character varying COLLATE pg_catalog."default" NOT
NULL,
    "Parent_email" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Parent_pkey" PRIMARY KEY ("Student_ID",
"Parent_name"),
    CONSTRAINT "Student_ID" FOREIGN KEY ("Student_ID")
        REFERENCES srs."Student" ("Student_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Parent"
OWNER to postgres;

```

- Parent(Student\_ID , parent\_name, parent email)



The screenshot shows a PostgreSQL query editor interface. The top part displays the SQL query:

```

1 SELECT * FROM srs."Parent"
2 ⏷ Loading... "Student_ID" ASC, "Parent_name" ASC

```

The bottom part shows the results of the query in a table format:

	Student_ID	Parent_name	Parent_email
1	1005	Carlita Yakhin	cyakhini@omnitu...
2	1023	Brig Petrelli	bpetrelli4@com...
3	1023	Lulita Danit	ldanit1f@phoca.cz
4	1045	Ashia Woolward	awoolward1n@h...
5	1045	Tomi Wormstone	twormstonem@p...
6	1057	Humfrid Cardello	hcardello1j@admin...
7	1064	Cherey Trafford	ctrafford15@t-on...
8	1075	Boyce Truckell	btruckell1q@vk.c...
9	1075	Reynolds Bangle	rbangle7@unicef...
10	1086	Jewell Geertje	jgeertje11@wikip...
11	1092	Rennie Tewnion	rtewnion1d@wor...

Total rows: 65 of 65    Query complete 00:00:00.181    ✓ Successfully run. Total query runtime: 181 msec. 65 rows affected.

**16.Alert :**

```

CREATE TABLE IF NOT EXISTS srs."Alert"
(
    "Student_ID" bigint NOT NULL,
    "Course_ID" bigint NOT NULL,
    "Date" date NOT NULL,
    "Alert_type" character varying COLLATE pg_catalog."default",
    "Alert_message" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Alert_pkey" PRIMARY KEY ("Student_ID",
    "Course_ID", "Date"),
    CONSTRAINT "Course_ID" FOREIGN KEY ("Course_ID")
        REFERENCES srs."Course" ("Course_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT "Student_ID" FOREIGN KEY ("Student_ID")
        REFERENCES srs."Student" ("Student_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Alert"
OWNER to postgres;

```

- Alert(Student\_ID, Course\_ID, date, alert type , alert message)

The screenshot shows a PostgreSQL database interface. At the top, there is a 'Query' tab and a 'Query History' tab. Below the tabs, a query is displayed:

```

1 SELECT * FROM srs."Alert"
2 Loading... "Student_ID" ASC, "Course_ID" ASC, "Date" ASC

```

Below the query, there are tabs for 'Data output', 'Messages', and 'Notifications'. The 'Data output' tab is selected, showing a table with the following data:

	Student_ID [PK] bigint	Course_ID [PK] bigint	Date [PK] date	Alert_type character varying	Alert_message character varying
1	1005	303	2022-11-15	Attendance related	Your attendance i...
2	1036	374	2022-11-20	lab related	Please submit yo...
3	1057	215	2022-10-10	quiz related	You missed your ...
4	1119	402	2022-11-16	meeting with par...	meeting start at ...
5	1150	203	2022-10-11	course related	focus on studies ...

## 17. Admin :

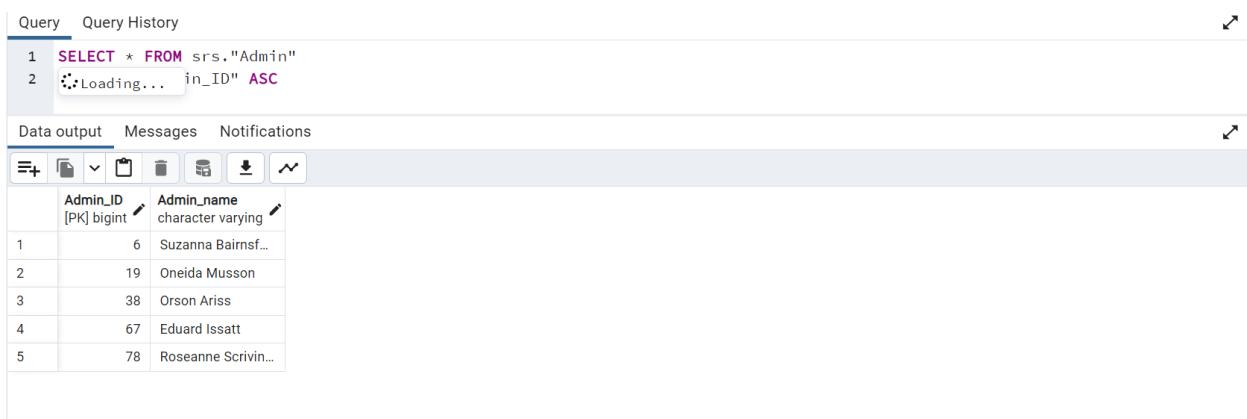
```

CREATE TABLE IF NOT EXISTS srs."Admin"
(
    "Admin_ID" bigint NOT NULL,
    "Admin_name" character varying COLLATE pg_catalog."default" NOT
NULL,
    CONSTRAINT "Admin_pkey" PRIMARY KEY ("Admin_ID")
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Admin"
OWNER to postgres;

```

- Admin(Admin\_ID ,admin name)



The screenshot shows a PostgreSQL query interface with the following details:

- Query History:** Shows the executed SQL command: `SELECT * FROM srs."Admin"`.
- Data output:** A table titled "Data output" showing the contents of the "Admin" table.
- Table Structure:** The table has two columns: "Admin\_ID" (bigint) and "Admin\_name" (character varying).
- Data:** Five rows of data are displayed:
 

	Admin_ID	Admin_name
1	6	Suzanna Bairnsf...
2	19	Oneida Musson
3	38	Orson Ariss
4	67	Eduard Issatt
5	78	Roseanne Scrivin...

**18.Total\_marks :**

```

CREATE TABLE IF NOT EXISTS srs."Total_marks"
(
    "Course_ID" bigint NOT NULL,
    "Activity_name" character varying COLLATE pg_catalog."default" NOT
NULL,
    "Total_marks" integer,
    CONSTRAINT "Total_marks_pkey" PRIMARY KEY ("Course_ID",
"Activity_name"),
    CONSTRAINT "Course_ID" FOREIGN KEY ("Course_ID")
        REFERENCES srs."Course" ("Course_ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS srs."Total_marks"
OWNER to postgres;

```

- **Total\_marks(Course\_ID, Activity\_name, Total\_marks)**

The screenshot shows a PostgreSQL query tool interface. At the top, there are tabs for 'Query' and 'Query History'. Below the tabs, two lines of SQL code are displayed:

```

1 SELECT * FROM srs."Total_marks"
2 ORDER BY "Course_ID" ASC, "Activity_name" ASC

```

Below the code, there are three buttons: 'Data output', 'Messages', and 'Notifications'. The 'Data output' button is selected. A table is displayed with the following data:

	Course_ID [PK] bigint	Activity_name [PK] character varying	Total_marks double precision
1	107	Endsem	60
2	107	Lab	30
3	107	Midsem	30
4	107	Quiz	20
5	107	Viva	10
6	116	Endsem	60
7	116	Lab	30
8	116	Midsem	30
9	116	Quiz	20
10	116	Viva	10
11	203	Endsem	60

At the bottom of the table area, it says 'Total rows: 40 of 40 Query complete 00:00:00.165 Ln 2, Col 47'.

A context menu is open over the primary key constraint 'Total\_marks\_pkey'. The menu items listed are: abort, absolute, access, action, add, admin, after, aggregate, all, also, and alter. The 'abort' option is currently selected.

## Queries:

1. Show the number of students studying under each Instructor.

```
SELECT "Instructor_ID",COUNT(*)  
FROM srs."Teaches"  
GROUP BY "Instructor_ID"
```

The screenshot shows a database query interface with the following details:

- Query History:** The tab is selected, showing the executed query:

```
1 select "Instructor_ID" , count(*)  
2 from srs."Teaches"  
3 group by "Instructor_ID"
```
- Data output:** This tab is also selected, displaying the results of the query in a table format.
- Messages:** No messages are present.
- Notifications:** No notifications are present.
- Table Results:**

	Instructor_ID	count
1	19	50
2	27	50
3	17	50
4	29	50
5	22	50
6	32	50
- Footer:** Shows "Total rows: 6 of 6", "Query complete 00:00:11.515", and "Ln 3, Col 25".

2. Show the highest number of courses taken by 1 student in the semester.

```
SELECT COUNT(*) FROM srs."Student_course"
GROUP BY "Student_ID"
ORDER BY COUNT(*) DESC LIMIT 1
```

The screenshot shows a database query interface. At the top, there is a toolbar with 'Query' and 'Query History' tabs, and a button labeled 'Execute/Refresh' (F5). Below the toolbar is the SQL query:

```
1 select count(*)
2 from srs."Student_course"
3 group by "Student_ID"
4 order by count(*) DESC limit 1
```

Underneath the query is a data grid titled 'Data output'. It has a single row with one column labeled 'count' and a value of 8. The grid includes icons for sorting and filtering. At the bottom of the interface, status bars show 'Total rows: 1 of 1', 'Query complete 00:00:04.861', and 'Ln 4, Col 31'.

3. Show all details of academic activities of Student whose Student\_Id = 1075.

```
SELECT *
FROM srs."Academic_reward"
WHERE "Student_ID" = 1075
```

The screenshot shows a database query interface with a query editor and a data output viewer.

**Query Editor:**

```

1 select *
2 from srs."Academic_reward"
3 where "Student_ID" = 1075
  
```

**Data Output:**

	Student_ID bigint	Course_ID bigint	Activity_name character varying	Obtained_marks double precision	Reward_points double precision
1	1075	203	Lab	24	80
2	1075	203	Midsem	23	76.6666666666667
3	1075	203	Endsem	19	31.6666666666667
4	1075	203	Quiz	5	25
5	1075	203	Viva	2	20
6	1075	215	Endsem	43	71.6666666666667
7	1075	215	Lab	4	13.3333333333333
8	1075	215	Midsem	19	63.3333333333333
9	1075	215	Quiz	2	10

Total rows: 30 of 30    Query complete 00:00:52.440    Ln 3, Col 26

#### 4. Show details of student with non-academic activities whose name starts with 'N'

```

SELECT *
FROM srs."Student" NATURAL JOIN srs."Non_academic_reward"
WHERE "Student"."Student_name" LIKE 'N%'
  
```

The screenshot shows a database query interface with a query editor and a data output viewer.

**Query Editor:**

```

1 select *
2 from srs."Student" natural join srs."Non_academic_reward"
3 where "Student"."Student_name" like 'N%'
4 
  
```

**Data Output:**

	Student_ID bigint	Student_name character varying	Address character varying	Activity_name character varying	Instructor_ID bigint	Reward_points integer
1	1064	Nev Mogie	92 Thierer Street	Drama	17	20
2	1064	Nev Mogie	92 Thierer Street	Coding	17	0
3	1064	Nev Mogie	92 Thierer Street	Dance	17	20
4	1064	Nev Mogie	92 Thierer Street	Sports	17	0
5	1064	Nev Mogie	92 Thierer Street	Music	17	20
6	1235	Nikita Sacker	2881 Shasta Cen...	Drama	17	0
7	1235	Nikita Sacker	2881 Shasta Cen...	Coding	17	20
8	1235	Nikita Sacker	2881 Shasta Cen...	Dance	17	0
9	1235	Nikita Sacker	2881 Shasta Cen...	Sports	17	20

Total rows: 15 of 15    Query complete 00:00:12.652    Ln 4, Col 1

5. Show all courses' course\_id and course\_name taken by the student whose Student\_ID is 1075.

```
SELECT "Course_ID", "Course_name"
FROM srs."Student_course" NATURAL JOIN srs."Course"
WHERE "Student_ID" = 1075
```

The screenshot shows a database query interface with the following details:

- Query Tab:** Contains the SQL code:
 

```
1 select "Course_ID", "Course_name"
2 from srs."Student_course" natural join srs."Course"
3 where "Student_ID" = 1075
4
```
- Data output Tab:** Displays the results of the query in a table format:
 

	Course_ID	Course_name
1	107	Calculus
2	374	Computational Sc...
3	304	Computer Netwo...
4	303	Digital Communi...
5	203	Embedded Hardw...
6	215	DBMS
7	111	Exrtra-curricular
- Status Bar:** Shows "Total rows: 7 of 7" and "Query complete 00:00:00.075".

6. Print Student\_id and Student\_name whose overall\_score is greater than avg overall\_score.

```
SELECT "Student"."Student_ID", "Student"."Student_name"
FROM srs."Performance" NATURAL JOIN srs."Student"
WHERE "Performance"."Overall_score">>(SELECT
AVG("Performance"."Overall_score") FROM srs."Performance")
```

The screenshot shows a database query interface. The query in the editor is:

```

1 select "Student"."Student_ID","Student"."Student_name"
2 from srs."Performance" natural join srs."Student"
3 where "Performance"."Overall_score">>(select avg("Performance"."Overall_score") from srs."Performance")

```

The results table has two columns: Student\_ID [PK] bigint and Student\_name character varying. The data is:

	Student_ID	Student_name
1	1150	Giselle Tarry
2	1345	Shelley Osgorby
3	1045	Zane Pellamonuten
4	1218	Dorian Craxford
5	1086	Chunce McCarney
6	1061	Dinny Penberthy
7	1131	Seline Jeeves
8	1147	Pammie O'Lochan
9	1286	Klement Teggart

Total rows: 27 of 27 Query complete 00:00:00.073 Ln 1, Col 55

## 7. Display names of Students in upper case.

```

SELECT UPPER("Student"."Student_name")
FROM srs."Student"

```

The screenshot shows a database query interface. The query in the editor is:

```

1 select upper("Student"."Student_name")
2 from srs."Student"
3

```

The results table has one column: upper text. The data is:

upper
JACK
ZANE PELLAMONUTEN
GISELLE TARRY
BROOKE DYNE
TYMO THY DUBOIS
DINNY PENBERTHY
PETR KULIS
PHILIPPE RAFTERY
JOYE VILA

Total rows: 51 of 51 Query complete 00:00:00.106 Ln 1, Col 15

8. Display the Course\_ID and Course\_name which are taken by instructor whose Instructor\_ID is 32.

```
SELECT "Course"."Course_ID", "Course"."Course_name"
FROM srs."Instructor" NATURAL JOIN srs."Course"
WHERE "Instructor"."Instructor_ID" = 32
```

	Course_ID [PK] bigint	Course_name character varying
1	107	Calculus
2	374	Computational Science

Total rows: 2 of 2    Query complete 00:00:00.084    Ln 4, Col 1

9. Show final academic score for every student with use of view.

```
CREATE OR REPLACE VIEW academic_score_final AS
SELECT "Student_ID", SUM("Reward_points") AS
"final_academic_score"
FROM srs."Academic_reward"
GROUP BY "Student_ID"

SELECT * FROM academic_score_final
```

Query    Query History

```

1 create or replace view academic_score_final AS
2 select "Student_ID" , sum("Reward_points") as "final_acadmic_score"
3 from srs."Academic_reward"
4 group by "Student_ID"
5
6 select * from academic_score_final
7 
```

• Loading...

Data output    Messages    Notifications

	Student_ID	final_acadmic_score
1	1150	379.666666666667
2	1045	366.5
3	1345	342.5
4	1257	254.5
5	1119	263.666666666667
6	1218	300
7	1397	288
8	1288	278

Total rows: 50 of 50    Query complete 00:00:00.070    Ln 6, Col 35

## 10. Display Total numbers of Students.

```
SELECT COUNT(*) FROM srs."Student"
```

Query    Query History

```

1 select count(*)
2 from srs."Student"
3 
```

Data output    Messages    Notifications

	count
1	51

Total rows: 1 of 1    Query complete 00:00:00.119    Ln 2, Col 19

11. Display all the phone numbers of student id 1005.

```
SELECT "Phone" FROM srs."Student_phone"
WHERE "Student_ID" = 1005
```

The screenshot shows a database query interface with the following details:

- Query History:** Shows the executed query: `select "Phone" from srs."Student_phone" where "Student_ID" = 1005`.
- Data output:** A table with one column named "Phone" (datatype bigint) containing two rows of data.
- Table Data:**

	Phone
1	1946796726
2	7046376732
- Statistics:** Total rows: 2 of 2 | Query complete 00:00:01.182 | Ln 2, Col 25

12. Show total non academic reward points earned by every student using view.

```
CREATE OR REPLACE VIEW non_academic_score_final AS
SELECT "Student_ID" , SUM("Reward_points") AS
"final_non_acadmic_score"
FROM srs."Non_academic_reward"
GROUP BY "Student_ID"

SELECT * FROM non_academic_score_final
```

The screenshot shows a database interface with a query editor and a results viewer.

**Query Editor:**

```

1 create or replace view non_academic_score_final AS
2 select "Student_ID" , sum("Reward_points") as "final_non_acdmic_score"
3 from srs."Non_academic_reward"
4 group by "Student_ID"
5
6 select * from non_academic_score_final
7

```

**Data Output:**

	Student_ID bigint	final_non_acdmic_score double precision
1	1150	40
2	1345	80
3	1257	80
4	1045	60
5	1119	80
6	1397	20
7	1218	80
8	1288	40
...	1076	60

Total rows: 50 of 50    Query complete 00:00:00.086    Ln 6, Col 39

13.Run the query that updates or calculates the overall performance table with use of all academic and non academic data of each student.

```

UPDATE srs."Performance"
set "Overall_score"=final_score.os
from (select fp."Student_ID", (((fp."arp" * fp."Academic_scale")
/ 100) + ((fp."Non_academic_scale" * fp."nrp") / 100))
as os
from (((select "Student_ID", sum("Reward_points") as ARP from
(Select
ar."Student_ID", ar."Course_ID", ar."Activity_name", ar."Obtained_marks",
ar."Reward_points", ars."Scale", tm."Total_marks"
From srs."Academic_reward" as ar,srs."Academic_scale" as
ars,srs."Total_marks" as tm
Where ar."Course_ID"=ars."Course_ID" and
ar."Activity_name"=ars."Activity_type"

```

```

and tm."Course_ID"=ar."Course_ID" and
tm."Activity_name"=ar."Activity_name") as academic_reward group
by "Student_ID")
as afs
natural join
(select n."Student_ID",sum(n."Reward_points") as NRP from
(select
nar."Student_ID",nar."Activity_name",nar."Reward_points"
from srs."Non_academic_reward" as nar join
srs."Non_academic_scale" as nas
on nar."Instructor_ID"=nas."Instructor_ID" and
nar."Activity_name"=nas."Activity_type") as n group by
n."Student_ID") as
nfs)
cross join srs."Combined_scale" as cs) as fp) as final_score;

select * from srs."Performance"

```

```

1 UPDATE srs."Performance"
2 set "Overall_score"=final_score.os
3 from (select fp."Student_ID",(((fp."arp" * fp."Academic_scale") / 100) + ((fp."Non_academic_scale" * fp."nrp") / 100)) as os
4 from ((select "Student_ID",sum("Reward_points") as ARP from (Select ar."Student_ID",ar."Course_ID",ar."Activity_name",ar."Ob:
5 From srs."Academic_reward" as ar,srs."Academic_scale" as ars,srs."Total_marks" as tm
6 Where ar."Course_ID"=ars."Course_ID" and ar."Activity_name"=ars."Activity_type"
7 and tm."Course_ID"=ar."Course_ID" and tm."Activity_name"=ar."Activity_name") as academic_reward group by "Student_ID") as afs
8 natural join
9 (select n."Student_ID",n(n."Reward_points") as NRP from (select nar."Student_ID",nar."Activity_name",nar."Reward_points"
10 from srs."Non_academic_reward" as nar join srs."Non_academic_scale" as nas
11 on nar."Instructor_ID"=nas."Instructor_ID" and nar."Activity_name"=nas."Activity_type") as n group by n."Student_ID") as nfs
12 cross join srs."Combined_scale" as cs) as fp) as final_score;
13
14 select * from srs."Performance"

```

The screenshot shows a database interface with a toolbar at the top labeled "Data output", "Messages", and "Notifications". Below the toolbar is a table with two columns: "Student\_ID" and "overall\_score". The table contains 15 rows of data. The "Student\_ID" column has values ranging from 1 to 15. The "overall\_score" column has values such as 155.5, 182, 133.0999999999999, 172.7000000000000, 146.4000000000000, 134.3, 163.4, 147.3, 134, 130.7000000000000, 153.1, 149.9, 164.3, 95.5000000000000, and 149.6000000000000. The bottom status bar indicates "Total rows: 50 of 50" and "Query complete 00:00:00.097".

	Student_ID	overall_score
1	1150	155.5
2	1345	182
3	1257	133.0999999999999
4	1045	172.7000000000000
5	1119	146.4000000000000
6	1397	134.3
7	1218	163.4
8	1288	147.3
9	1376	134
10	1341	130.7000000000000
11	1086	153.1
12	1061	149.9
13	1131	164.3
14	1160	95.5000000000000
15	1147	149.6000000000000

#### 14. Display Top 5 student of this semester who score high in overall semester.

```
SELECT "Student_ID", "Overall_score"
FROM srs."Performance" ORDER BY "Overall_score" DESC LIMIT 5
```

The screenshot shows a database interface with a toolbar at the top labeled "Data output", "Messages", and "Notifications". Below the toolbar is a table with two columns: "Student\_ID" and "Overall\_score". The table contains 5 rows of data. The "Student\_ID" column has values ranging from 1 to 5. The "Overall\_score" column has values such as 182, 179.2, 179.0000000000000, 172.7000000000000, and 170.7000000000000. The bottom status bar indicates "Total rows: 50 of 50" and "Query complete 00:00:00.097".

	Student_ID	Overall_score
1	1345	182
2	1005	179.2
3	1222	179.0000000000000
4	1045	172.7000000000000
5	1229	170.7000000000000

15. List 10 brilliant student who score higher under Instructor ID='22'.

```
SELECT
"Academic_reward"."Reward_points", "Academic_reward"."Course_ID",
"Academic_reward"."Student_ID"
FROM (srs."Academic_reward" NATURAL JOIN (srs."Teaches" NATURAL
JOIN srs."Course"))
WHERE "Teaches"."Instructor_ID"=22
ORDER BY "Academic_reward"."Reward_points" DESC LIMIT 10
```

```
1 select "Academic_reward","Reward_points","Academic_reward"."Course_ID","Academic_reward"."Student_ID"
2 from (srs."Academic_reward" natural join (srs."Teaches" natural join srs."Course"))
3 where "Teaches"."Instructor_ID"=22
4 order by "Academic_reward"."Reward_points" DESC limit 10
```

Data output    Messages    Notifications

	Reward_points	Course_ID	Student_ID
1	20	402	1131
2	20	402	1169
3	20	402	1180
4	20	303	1229
5	20	402	1256
6	19.3333333333333	303	1075
7	19.3333333333333	303	1086
8	19	303	1141
9	19	402	1023
10	19	402	1119

Total rows: 10 of 10    Query complete 00:00:00.925    Ln 3, Col 35

16. Display all courses average scores in semester.

```
SELECT "Course_ID", AVG ("Reward_points")
FROM srs."Academic_reward"
GROUP BY "Course_ID"
```

The screenshot shows a database query interface with the following details:

- Query Tab:** Contains the SQL code: 

```
1 SELECT "Course_ID", AVG("Reward_points")
2 FROM srs."Academic_reward"
3 GROUP BY "Course_ID"
```
- Data output Tab:** Displays the results of the query in a table format.
- Table Headers:** Course\_ID (bigint), avg (double precision).
- Table Data:**

	Course_ID	avg
1	116	5.945
2	402	7.06666666666666
3	374	7.52166666666666
4	215	6.64133333333333
5	107	6.45933333333333
6	304	6.63266666666666
7	203	6.634
8	303	7.64200000000000
- Bottom Status Bar:** Total rows: 8 of 8, Query complete 00:00:00.000, Ln 3, Col 21.

## 17. Display number of courses taken by every instructor.

```
SELECT "Instructor_ID" , COUNT("Course_ID") AS c1
FROM srs."Course"
GROUP BY "Instructor_ID"
ORDER BY c1 DESC
```

The screenshot shows a database query interface with the following details:

- Query Tab:** Contains the SQL code: 

```
1 select "Instructor_ID" , count("Course_ID") as c1
2 from srs."Course"
3 group by "Instructor_ID"
4 order by c1 desc|
```
- Data output Tab:** Displays the results of the query in a table format.
- Table Headers:** Instructor\_ID (bigint), c1 (bigint).
- Table Data:**

	Instructor_ID	c1
1	22	2
2	17	2
3	32	2
4	19	1
5	29	1
6	27	1
- Bottom Status Bar:** Total rows: 6 of 6, Query complete 00:00:36.654, Ln 4, Col 17.

**18. Display all alerts received in the month of November-2022.**

```
SELECT * FROM srs."Alert"
WHERE "Date" BETWEEN '01-11-2022' AND '30-11-2022'
```

The screenshot shows a database query interface with a query editor and a results viewer. The query editor contains the SQL code for selecting alerts from November 2022. The results viewer displays a table with three rows of data, showing student IDs, course IDs, dates, alert types, and messages.

	Student_ID [PK] bigint	Course_ID [PK] bigint	Date [PK] date	Alert_type character varying	Alert_message character varying
1	1005	303	2022-11-15	Attendance related	Your attendance i...
2	1036	374	2022-11-20	lab related	Please submit yo...
3	1119	402	2022-11-16	meeting with par...	meeting start at ...

**19. Display details of parents whose Student\_id is 1147.**

```
SELECT * FROM srs."Parent"
WHERE "Student_ID" = 1147
```

The screenshot shows a database query interface with a query editor and a results viewer. The query editor contains the SQL code for selecting parents where the student ID is 1147. The results viewer displays a table with two rows of data, showing parent names and emails.

	Student_ID [PK] bigint	Parent_name [PK] character varying	Parent_email character varying
1	1147	Thatcher Coulthard	tcoulthardc@newsvine.com
2	1147	Diahann Prendergrast	dprendergrast1a@usda.gov

Total rows: 2 of 2    Query complete 00:00:08.423    Ln 3, Col 26

20. Create a function for sending alert to top 5 students for getting reward.

```
create or replace function give_reward_notification()
returns void
as
$$
declare

BEGIN
insert into srs."Alert"
("Student_ID", "Course_ID", "Date", "Alert_type", "Alert_message")
(select "Student_ID", 111, current_timestamp, 'reward_related', 'You
have been rewarded for top being in top 5 performers'
from srs."Reward" as r cross join (select "Student_ID", rank()
over (order by "Overall_score" desc) from
srs."Performance" order by "Overall_score" desc limit 5) as sid
where r."Rank" = sid.rank);

end;
$$
language 'plpgsql'

select give_reward_notification()
```

Query    Query History

---

```
1  create or replace function give_reward_notification()
2  returns void
3  as
4  $$
5  declare
6
7  BEGIN
8      insert into srs."Alert" ("Student_ID", "Course_ID", "Date", "Alert_type", "Alert_message")
9      (select "Student_ID", 0, current_timestamp, 'reward_related', 'You have been rewarded for top being in top
10     from srs."Reward" as r cross join (select "Student_ID", rank() over (order by "Overall_score" desc) fro
11     where r."Rank" = sid.rank);
12
13 end;
14 $$
15 language 'plpgsql'
16
17
```

The screenshot shows a database interface with a query editor and a results viewer.

**Query Editor:**

```

1 SELECT * FROM srs."Alert"
2 ORDER BY "Course_ID" ASC, "Student_ID" ASC, "Date" ASC
  
```

**Data Output:**

	Student_ID [PK] bigint	Course_ID [PK] bigint	Date [PK] timestamp without time zone	Alert_type character varying	Alert_message character varying
1	1005	0	2022-11-19 22:55:56.064062	reward_related	You have been rewarded for top being in top 5 performers
2	1045	0	2022-11-19 22:55:56.064062	reward_related	You have been rewarded for top being in top 5 performers
3	1150	0	2022-11-19 22:55:56.064062	reward_related	You have been rewarded for top being in top 5 performers
4	1152	0	2022-11-19 22:55:56.064062	reward_related	You have been rewarded for top being in top 5 performers
5	1256	0	2022-11-19 22:55:56.064062	reward_related	You have been rewarded for top being in top 5 performers
6	1005	107	2022-11-18 00:00:00	Attendance Relat...	Your attendance is low in this course!!
7	1023	107	2022-11-18 00:00:00	Attendance Relat...	Your attendance is low in this course!!

Total rows: 206 of 206    Query complete 00:00:00.190    Ln 2, Col 57

21. Create a trigger-function for calculating the Reward\_points on inserting or updating tuples in the table Academic\_rewards.

```

create or replace function
update_before_Academic_Reward_Points()
returns trigger
as
$$
declare
total double precision;
scale_temp double precision;
BEGIN
select "Total_marks" into total
from srs."Total_marks"
where "Activity_name"=new."Activity_name" and "Course_ID" =
new."Course_ID" ;
select "Scale" into scale_temp
from srs."Academic_scale"
where "Activity_type"=new."Activity_name" and "Course_ID" =
new."Course_ID" ;
new."Reward_points" = new."Obtained_marks"*scale_temp/total;
  
```

```

return new;
end;
$$
language 'plpgsql'

create trigger trigger_update_before_academic_reward_points
before update or insert on srs."Academic_reward"
for each row
execute procedure update_before_Academic_Reward_Points();

```

Query    Query History

```

1  create or replace function update_before_Academic_Reward_Points()
2  returns trigger
3  as
4  $$
5  declare
6  total double precision;
7  scale_temp double precision;
8  BEGIN
9    select "Total_marks" into total
10   from srs."Total_marks"
11  where "Activity_name"=new."Activity_name" and "Course_ID" = new."Course_ID" ;
12  select "Scale" into scale_temp
13   from srs."Academic_scale"
14  where "Activity_type"=new."Activity_name" and "Course_ID" = new."Course_ID" ;
15  new."Reward_points" = new."Obtained_marks"*scale_temp/total;
16  return new;
17 end;
18 $$ 
19 language 'plpgsql'
20 create trigger trigger_update_before_academic_reward_points
21 before update or insert on srs."Academic_reward"
22 for each row
23 execute procedure update_before_Academic_Reward_Points();

```

Query    Query History

```

1 UPDATE srs."Academic_reward"
2     SET "Obtained_marks"=50
3      WHERE "Student_ID"=1005 and "Course_ID"=107 and "Activity_name"='Endsem'
4
5 select * from srs."Academic_reward" WHERE "Student_ID"=1005 and "Course_ID"=107 and "Activity_name"='Endsem'
6
7 Loading...

```

Data output    Messages    Notifications

	Student_ID [PK] bigint	Course_ID [PK] bigint	Activity_name [PK] character varying	Obtained_marks double precision	Reward_points double precision
1	1005	107	Endsem	50	25

22. Create a trigger-function for calculating the Reward\_points on inserting or updating tuples in the table Non\_academic\_rewards.

```
create or replace function
update_before_Non_academic_Reward_Points()
returns trigger
as
$$
declare
scale_temp double precision;
BEGIN
select "Scale" into scale_temp
from srs."Non_academic_scale"
where "Activity_type"=new."Activity_name" ;
new."Reward_points" = new."Reward_points"*scale_temp/100;
return new;
end;
$$
language 'plpgsql'

create trigger trigger_update_before_non_academic_reward_points
before update or insert on srs."Non_academic_reward"
for each row
execute procedure update_before_Non_academic_Reward_Points();
```

```

Query Query History
1 create or replace function update_before_Non_academic_Reward_Points()
2 returns trigger
3 as
4 $$*
5 declare
6 scale_temp double precision;
7 BEGIN
8     select "Scale" into scale_temp
9     from srs."Non_academic_scale"
10    where "Activity_type"=new."Activity_name" ;
11    new."Reward_points" = new."Reward_points" *scale_temp/100;
12    return new;
13 end;
14 $$*
15 language 'plpgsql'
16
17
18 create trigger trigger_update_before_non_academic_reward_points
19 before update or insert on srs."Non_academic_reward"
20 for each row
21 execute procedure update_before_Non_academic_Reward_Points();
22

```

```

Query Query History
1 UPDATE srs."Non_academic_reward"
2      SET "Reward_points"=100
3      WHERE "Student_ID"=1005 and "Activity_name"='Coding'
4
5 select * from srs."Non_academic_reward" WHERE "Student_ID"=1005 and "Activity_name"='Coding' | Loading...
6
7
Data output Messages Notifications

```

	Student_ID [PK] bigint	Activity_name [PK] character varying	Instructor_ID bigint	Reward_points double precision
1	1005	Coding	17	20

23.Create a trigger-function for sending alerts to the students who have attendance less than 80 percent in a particular course on updating the value of attendance.

```

create or replace function update_after_alert_Attendance()
returns trigger
as
$$
declare

BEGIN
    if new."Attendance" < 80 then
        INSERT INTO srs."Alert"(

```

```

"Student_ID", "Course_ID", "Date", "Alert_type",
"Alert_message")
      VALUES (new."Student_ID", new."Course_ID", current_date,
'Attendance Related', 'Your attendance is low in this
course!!');
      return new;
    end if;
      return new;
end;
$$
language 'plpgsql'

create trigger trigger_update_after_alert_attendance
after update or insert on srs."Student_course"
for each row
execute procedure update_after_alert_Attendance();

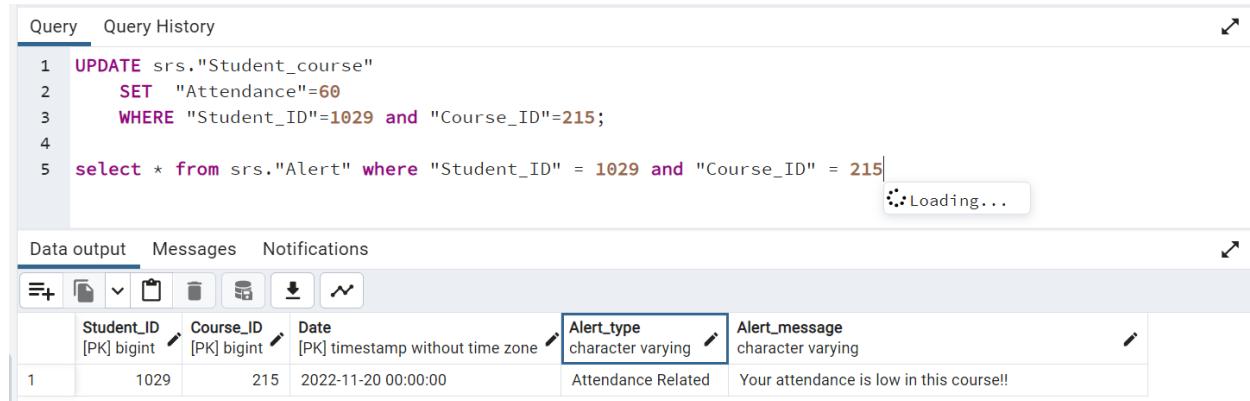
```

The screenshot shows a database query editor window with two tabs: 'Query' and 'Query History'. The 'Query' tab is active and contains the following PL/pgSQL code:

```

1 create or replace function update_after_alert_Attendance()
2 returns trigger
3 as
4 $$ 
5 declare
6
7 BEGIN
8   if new."Attendance" < 80 then
9     INSERT INTO srs."Alert"(
10    "Student_ID", "Course_ID", "Date", "Alert_type", "Alert_message")
11    VALUES (new."Student_ID", new."Course_ID", current_date, 'Attendance Related', 'Your attendance is low in this course!!');
12   return new;
13 end if;
14   return new;
15 end;
16 $$ 
17 language 'plpgsql'
18
19 create trigger trigger_update_after_alert_attendance
20 after update or insert on srs."Student_course"
21 for each row
22 execute procedure update_after_alert_Attendance();

```



```

1 UPDATE srs."Student_course"
2   SET "Attendance"=60
3 WHERE "Student_ID"=1029 and "Course_ID"=215;
4
5 select * from srs."Alert" where "Student_ID" = 1029 and "Course_ID" = 215

```

Loading...

Student_ID [PK] bigint	Course_ID [PK] bigint	Date [PK] timestamp without time zone	Alert_type character varying	Alert_message character varying
1029	215	2022-11-20 00:00:00	Attendance Related	Your attendance is low in this course!!

## 24. Create a function for calculating the final performance of all the students.

```

create or replace function calculate_final_performance()
returns trigger
as
$$
declare
aascore double precision;
nascore double precision;
aascale double precision;
nascale double precision;
BEGIN
select academic_score_final.final_acadmic_score into aascore
from academic_score_final
where "Student_ID" = new."Student_ID";
select non_academic_score_final.final_non_acadmic_score into
nascore
from non_academic_score_final
where "Student_ID" = new."Student_ID";
select "Academic_scale" into aascale
from srs."Combined_scale"
where "Instructor_ID" = 22;
select "Non_academic_scale" into nascale
from srs."Combined_scale"
where "Instructor_ID" = 22;

```

```

update srs."Performance" set "Overall_score"=((aascore*aascale)
+ (nascore*nascale))/100
where "Student_ID" = new."Student_ID";
return new;
end;
$$
language 'plpgsql'

```

Query    Query History

```

1  create or replace function calculate_final_performance()
2  returns trigger
3  as
4  $$
5  declare
6  aascore double precision;
7  nascore double precision;
8  aascale double precision;
9  nascale double precision;
10 ▼ BEGIN
11      select academic_score_final.final_acdmic_score into aascore
12      from academic_score_final
13      where "Student_ID" = new."Student_ID";
14      select non_academic_score_final.final_non_acadmic_score into nascore
15      from non_academic_score_final
16      where "Student_ID" = new."Student_ID";
17      select "Academic_scale" into aascale
18      from srs."Combined_scale"
19      where "Instructor_ID" = 22;
20      select "Non_academic_scale" into nascale
21      from srs."Combined_scale"
22      where "Instructor_ID" = 22;
23      update srs."Performance" set "Overall_score"=((aascore*aascale) + (nascore*nascale))/100
24      where "Student_ID" = new."Student_ID";
25      return new;
26  end;
27  $$
28 language 'plpgsql'
29

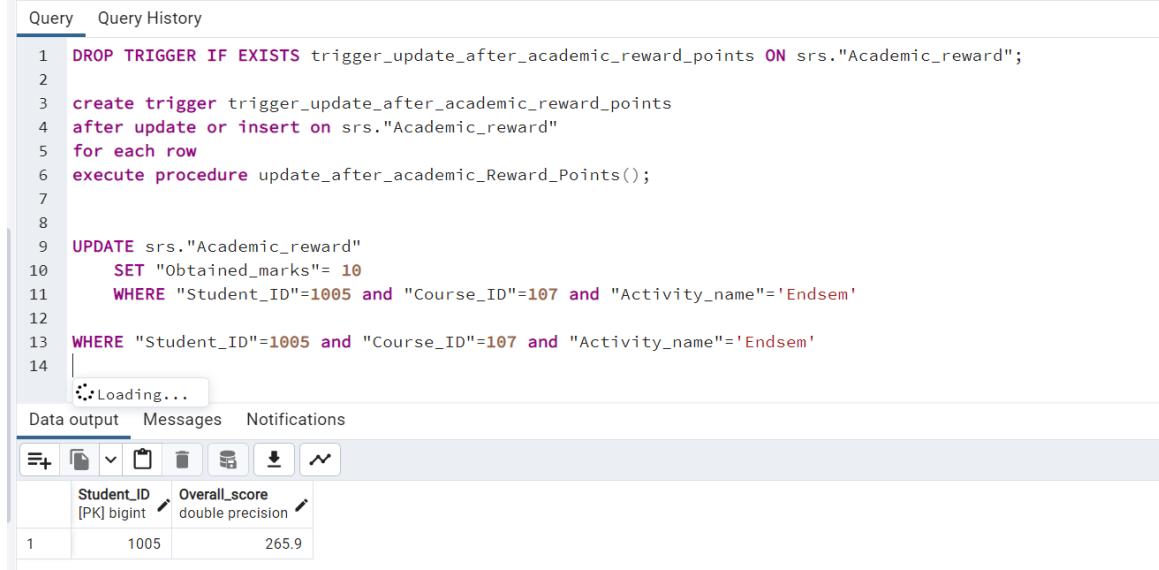
```

25. Create a trigger for updating the final performance of a particular student on updating or inserting data in the academic reward table.(which uses the function ‘calculate\_final\_performance()’)

```

create trigger trigger_update_after_academic_reward_points
after update on srs."Academic_reward"
for each row
execute procedure update_after_academic_Reward_Points();

```



```

1 DROP TRIGGER IF EXISTS trigger_update_after_academic_reward_points ON srs."Academic_reward";
2
3 create trigger trigger_update_after_academic_reward_points
4 after update or insert on srs."Academic_reward"
5 for each row
6 execute procedure update_after_academic_Reward_Points();
7
8
9 UPDATE srs."Academic_reward"
10    SET "Obtained_marks"= 10
11 WHERE "Student_ID"=1005 and "Course_ID"=107 and "Activity_name"='Endsem'
12
13 WHERE "Student_ID"=1005 and "Course_ID"=107 and "Activity_name"='Endsem'
14

```

• Loading...

Data output Messages Notifications

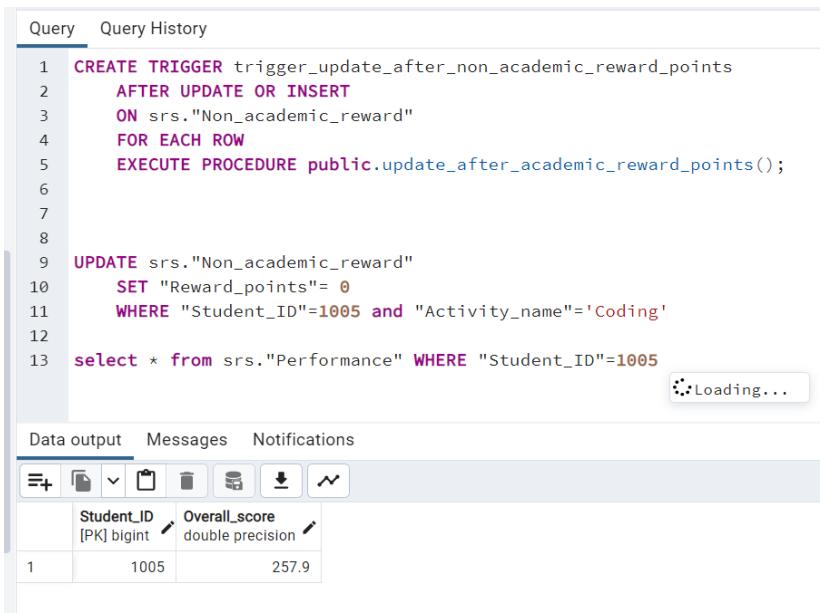
	Student_ID [PK] bigint	Overall_score double precision
1	1005	265.9

26. Create a trigger for updating the final performance of a particular student on updating or inserting data in the non\_academic\_reward table.(which uses the function ‘calculate\_final\_performance()’)

```

create trigger trigger_update_after_non_academic_reward_points
after update on srs."Non_academic_reward"
for each row
execute procedure update_after_academic_Reward_Points();

```



```

1 CREATE TRIGGER trigger_update_after_non_academic_reward_points
2     AFTER UPDATE OR INSERT
3     ON srs."Non_academic_reward"
4     FOR EACH ROW
5     EXECUTE PROCEDURE public.update_after_academic_reward_points();
6
7
8
9 UPDATE srs."Non_academic_reward"
10    SET "Reward_points"= 0
11 WHERE "Student_ID"=1005 and "Activity_name"='Coding'
12
13 select * from srs."Performance" WHERE "Student_ID"=1005

```

• Loading...

Data output Messages Notifications

	Student_ID [PK] bigint	Overall_score double precision
1	1005	257.9

# Project Code with output screenshots

## ❖ Backend Code:

**Connects frontend with backend (Using Django):**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'srs_app_DB',
        'USER': 'postgres',
        'PASSWORD': 'admin',
        'HOST': 'localhost',
        'PORT': '5432'
    }
}
```

**Code for setting up the URL:**

```
urlpatterns = [
    path('', views.showhomepage, name='homepage'),

    path('performance/', views.performanceall, name='performance_all'),
    ,

    path('performance/<str:var>/', views.performance_sort, name='performance_sort'),
    path('adminview/', views.showadmin, name='admin_view'),

    path('adminview/instructor/', views.instructor_form, name='instructor_insert'),
```

```
path('adminview/instructor/<int:id>',views.instructor_form,name='instructor_update'),  
  
path('adminview/instructor/delete/<int:id>',views.instructor_delete,name='instructor_delete'),  
  
path('adminview/instructor/list/<str:var>',views.instructor_sort,name='instructor_sort'),  
  
path('adminview/instructor/list/',views.instructor_list,name='instructor_list'),  
  
path('adminview/course/',views.course_form,name='course_insert'),  
,  
  
path('adminview/course/<int:id>',views.course_form,name='course_update'),  
  
path('adminview/course/delete/<int:id>',views.course_delete,name='course_delete'),  
  
path('adminview/course/list/<str:var>',views.course_sort,name='course_sort'),  
  
path('adminview/course/list/',views.course_list,name='course_list'),  
    path('student/',views.student_form,name='student_insert'),  
  
path('<int:student_ID>',views.student_form,name='student_update'),  
    path('list/',views.student_list,name='student_list'),
```

```

path('delete/<int:id>',views.student_delete,name='student_delete'),
    path('<str:var>/',views.student_sort,name='student_sort'),

path('q/studentview/',views.studentviewquery,name='show_studentview'),

path('q/instructorview/alert',views.sendalert,name='send_alert')
,

path('q/instructorview/',views.instructorviewquery,name='show_instructorview'),

path('q/instructorview/update',views.updatemarks,name='update_marks'),


]

```

### **Code for setting up the modules used for frontend:**

```

class student(models.Model):
    student_ID = models.IntegerField(primary_key=True)
    student_name = models.CharField(max_length=60)
    address = models.CharField(max_length=60)

    def __str__(self):
        return str(self.student_ID)


class admin(models.Model):
    admin_ID = models.IntegerField(primary_key=True)
    admin_name = models.CharField(max_length=60)

```

```

def __str__(self):
    return str(self.admin_ID)

class instructor(models.Model):
    instructor_ID = models.IntegerField(primary_key=True)
    instructor_name = models.CharField(max_length=60)
    instructor_address = models.CharField(max_length=60)
    admin_ID = models.ForeignKey(admin, on_delete=models.CASCADE)

    def __str__(self):
        return str(self.instructor_ID)

class course(models.Model):
    course_ID = models.IntegerField(primary_key=True)
    course_name = models.CharField(max_length=60)
    instructor_ID =
models.ForeignKey(instructor, on_delete=models.CASCADE)

class reward(models.Model):
    rank = models.IntegerField(primary_key=True)
    reward_name = models.CharField(max_length=60)

class combined_scale(models.Model):
    instructor_ID =
models.ForeignKey(instructor, on_delete=models.CASCADE, primary_key=True)
    academic_scale = models.FloatField()
    non_academic_scale = models.FloatField()

class performance(models.Model):

```

```

    student_ID =
models.ForeignKey(student, on_delete=models.CASCADE, primary_key=True)
performance = models.FloatField()

```

**Code for fetching, editing, deleting, sorting and running queries in the database:**

```

def showhomepage(request):
    return render(request, "srsapp/homepage.html")

def showadmin(request):
    return render(request, "srsapp/admin.html")

def student_list(request):
    context = {'student_list':student.objects.all()}
    return render(request, "srsapp/student_list.html", context)

def instructor_list(request):
    context = {'instructor_list':instructor.objects.all()}
    return render(request, "srsapp/instructor_list.html", context)

def student_form(request, student_ID=0):
    if request.method == "GET":
        if student_ID==0:
            form = studentForm()
        else:
            students=student.objects.get(pk=student_ID)
            form = studentForm(instance=students)
        return
    render(request, "srsapp/student_form.html", {'form':form})
    else:
        if student_ID==0:
            form = studentForm(request.POST)
        else:

```

```

        students=student.objects.get(pk=student_ID)
        form = studentForm(request.POST, instance=students)
        if form.is_valid():
            form.save()
        return redirect('/student/list')

def instructor_form(request,id=0):
    if request.method == "GET":
        if id==0:
            form = instructorForm()
        else:
            instructors=instructor.objects.get(pk=id)
            form = instructorForm(instance=instructors)
        return
    render(request,"srsapp/instructor_form.html",{'form':form})
    else:
        if id==0:
            form = instructorForm(request.POST)
        else:
            instructors=instructor.objects.get(pk=id)
            form =
instructorForm(request.POST,instance=instructors)
        if form.is_valid():
            form.save()
        return redirect('/adminview/instructor/list')

def student_delete(request,id):
    students=student.objects.get(pk=id)
    students.delete()
    return redirect('/student/list')

def student_sort(request,var):
    students=student.objects.order_by(var)
    context={'student_list' : students}

```

```

    return render(request,"srsapp/student_list.html",context)

def instructor_delete(request,id):
    instructors=instructor.objects.get(pk=id)
    instructors.delete()
    return redirect('/adminview/instructor/list')

def instructor_sort(request,var):
    instructors=instructor.objects.order_by(var)
    context={'instructor_list':instructors}
    return render(request,"srsapp/instructor_list.html",context)

def course_list(request):
    context = {'course_list':course.objects.all()}
    return render(request,"srsapp/course_list.html",context)

def course_delete(request,id):
    courses=course.objects.get(pk=id)
    courses.delete()
    return redirect('/adminview/course/list')

def course_sort(request,var):
    courses=course.objects.order_by(var)
    context={'course_list':courses}
    return render(request,"srsapp/course_list.html",context)

def course_form(request,id=0):
    if request.method == "GET":
        if id==0:
            form = courseForm()
        else:
            courses=course.objects.get(pk=id)
            form = courseForm(instance=courses)

```

```

        return
render(request,"srsapp/course_form.html",{'form':form})
else:
    if id==0:
        form = courseForm(request.POST)
    else:
        courses=course.objects.get(pk=id)
        form = courseForm(request.POST,instance=courses)
    if form.is_valid():
        form.save()
    return redirect('/adminview/course/list')

def performanceall(request):
    context = {'performance_all':performance.objects.all() }
    return render(request,"srsapp/performanceall.html",context)

def performance_sort(request,var):
    performances=performance.objects.order_by(var)
    context={'performance_all' : performances}
    return render(request,"srsapp/performanceall.html",context)

def studentviewquery(request):
    if request.method == 'POST':
        search_id = request.POST.get('textfield', None)
        raw_query1 = "select * from srsapp_student where
\"student_ID\" = %s ;"
        cursor = connection.cursor()
        cursor.execute(raw_query1, (search_id,))
        alldata1=cursor.fetchall()

        raw_query2 = "select * from srsapp_performance where
\"student_ID_id\")==%s ;"
        cursor = connection.cursor()
        cursor.execute(raw_query2, (search_id,))

```

```

alldata2=cursor.fetchall()

    raw_query3 = "select * from academic_score_final natural
join non_academic_score_final where
academic_score_final.\"student_ID\" = %s;"
    cursor = connection.cursor()
    cursor.execute(raw_query3, (search_id,))
alldata3=cursor.fetchall()

    raw_query4 = "select * from srsapp_alert where
\"student_ID\" = %s;"
    cursor = connection.cursor()
    cursor.execute(raw_query4, (search_id,))
alldata4=cursor.fetchall()

    raw_query5 = "SELECT \"course_ID\",
sum(reward_points)from srsapp_academic_reward    where
\"student_ID\"=%s group by \"course_ID\";"
    cursor = connection.cursor()
    cursor.execute(raw_query5, (search_id,))
alldata5=cursor.fetchall()

    raw_query6 = "select * from srsapp_non_academic_reward
where \"student_ID\"=%s"
    cursor = connection.cursor()
    cursor.execute(raw_query6, (search_id,))
alldata6=cursor.fetchall()

    return
render(request,'srsapp/student_view.html',{'data':alldata1,'data
2':alldata2,'data3':alldata3,'data4':alldata4,'data5':alldata5,'
data6':alldata6})

else:

```

```

        return render(request, 'srsapp/studentviewip.html')

def sendalert(request):
    if request.method == 'POST':
        s_id = request.POST.get('id', None)
        c_id= request.POST.get('c_id', None)
        type_1 = request.POST.get('type', None)
        msg = request.POST.get('msg', None)

        raw_query1 = "INSERT INTO
public.srsapp_alert(\"student_ID\", \"course_ID\", date,
alert_type, alert_message) VALUES (%s, %s,CURRENT_TIMESTAMP,
%s,%s);"
        cursor = connection.cursor()
        cursor.execute(raw_query1, (s_id,c_id,type_1,msg))

        return redirect('show_instructorview')

    else:
        return render(request, 'srsapp/alert_input.html')

def instructorviewquery(request):
    if request.method == 'POST':
        search_id = request.POST.get('textfield', None)
        raw_query1 = "select * from srsapp_course natural join
srsapp_academic_reward where srsapp_course.\"instructor_ID_id\""
        "= %s ;"
        cursor = connection.cursor()
        cursor.execute(raw_query1, (search_id,))
        alldata1=cursor.fetchall()

        raw_query2 = "select *from public.srsapp_instructor
where \"instructor_ID\" = %s ;"
        cursor = connection.cursor()

```

```

        cursor.execute(raw_query2, (search_id,))
        alldata2=cursor.fetchall()

    return

render(request,'srsapp/instructorview.html',{'data':alldata1,'da-
ta2':alldata2})

else:
    return render(request, 'srsapp/instructor_ip.html')

def updatemarks(request):
    if request.method == 'POST':
        s_id = request.POST.get('id', None)
        c_id= request.POST.get('c_id', None)
        type_1 = request.POST.get('type', None)
        mark = request.POST.get('marks', None)

        raw_query1 = "UPDATE public.srsapp_academic_reward SET
obtained_marks= %s WHERE \"student_ID\\"=%s and \"course_ID\\"=%s
and activity_name=%s;"
        cursor = connection.cursor()
        cursor.execute(raw_query1, (mark,s_id,c_id,type_1))
        return redirect('show_instructorview')

    else:
        return render(request, 'srsapp/acad_input.html')

```

**Code for setting up model-forms:**

```
class studentForm(forms.ModelForm):
```

```

class Meta:
    model = student
    fields = ('student_ID', 'student_name', 'address')
    labels = {
        'student_ID': 'Student ID',
        'student_name': 'Student Name',
        'address': 'Address'
    }

def __init__(self, *args, **kwargs):
    super(studentForm, self).__init__(*args, **kwargs)
    self.fields['address'].required = False

class instructorForm(forms.ModelForm):

    class Meta:
        model = instructor
        fields = [
            'instructor_ID', 'instructor_name', 'instructor_address', 'admin_ID'
        ]
        labels = {
            'instructor_ID': 'Instructor ID',
            'instructor_name': 'Instructor Name',
            'instructor_address': 'Address',
            'admin_ID': 'Admin_ID'
        }

    def __init__(self, *args, **kwargs):
        super(instructorForm, self).__init__(*args, **kwargs)
        self.fields['admin_ID'].empty_label = "select"

class courseForm(forms.ModelForm):

    class Meta:

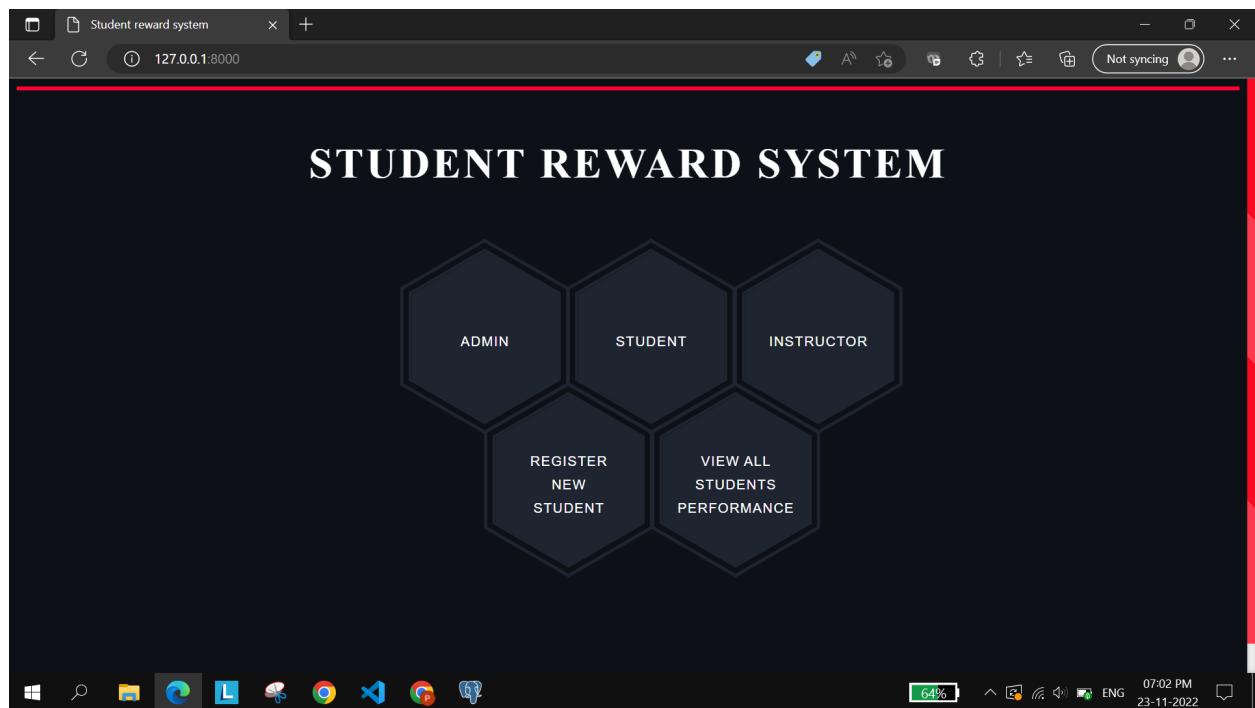
```

```
model = course
fields = [ 'course_ID', 'course_name', 'instructor_ID' ]
labels = {
    'course_ID': 'Course ID',
    'course_name': 'Course Name',
    'instructor_ID': 'Instructor ID'
}

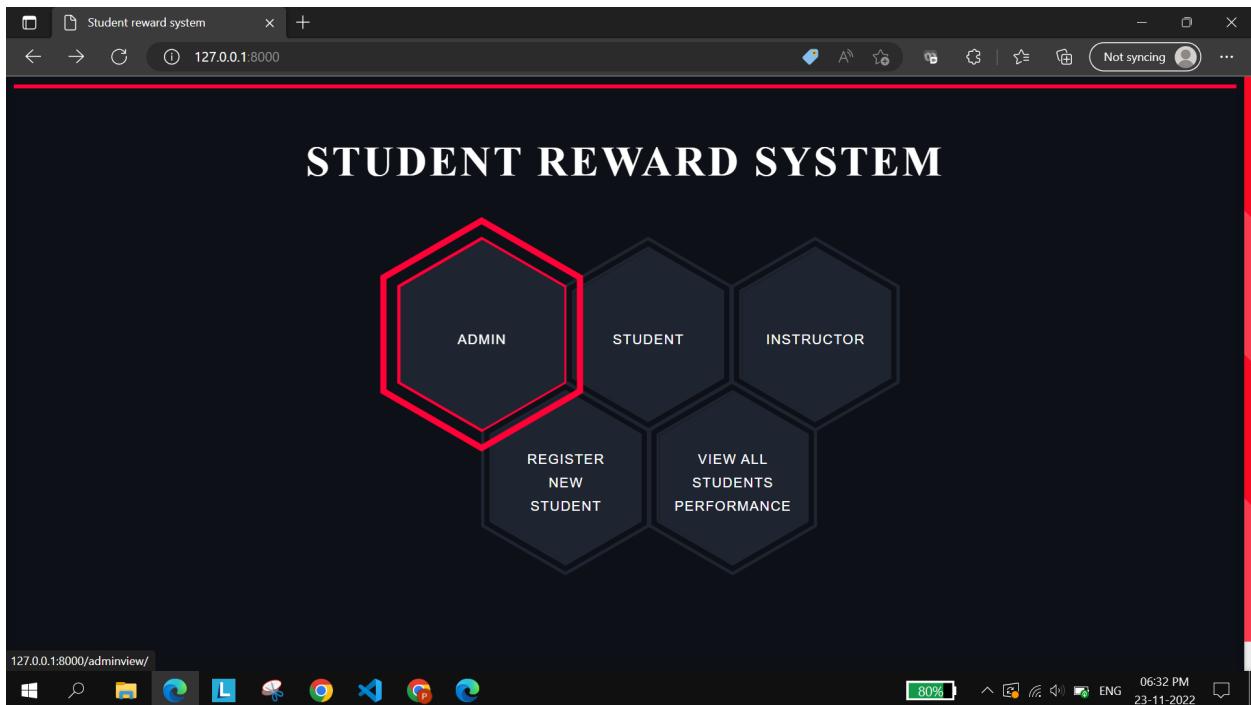
def __init__(self, *args, **kwargs):
    super(courseForm, self).__init__(*args, **kwargs)
    self.fields['instructor_ID'].empty_label = "select"
```

## ❖ Screenshots of the Website that connects the Database:

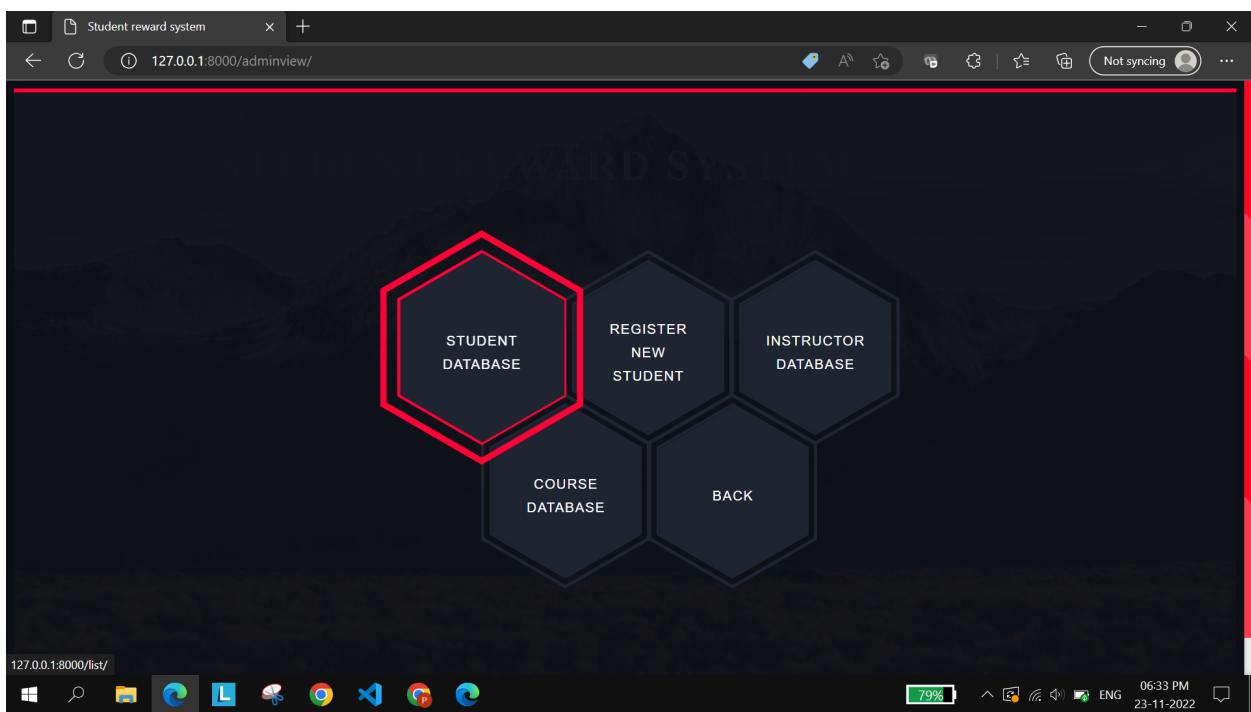
### 1) Homepage:



## 2) Admin-view:



→ Inside Admin view :

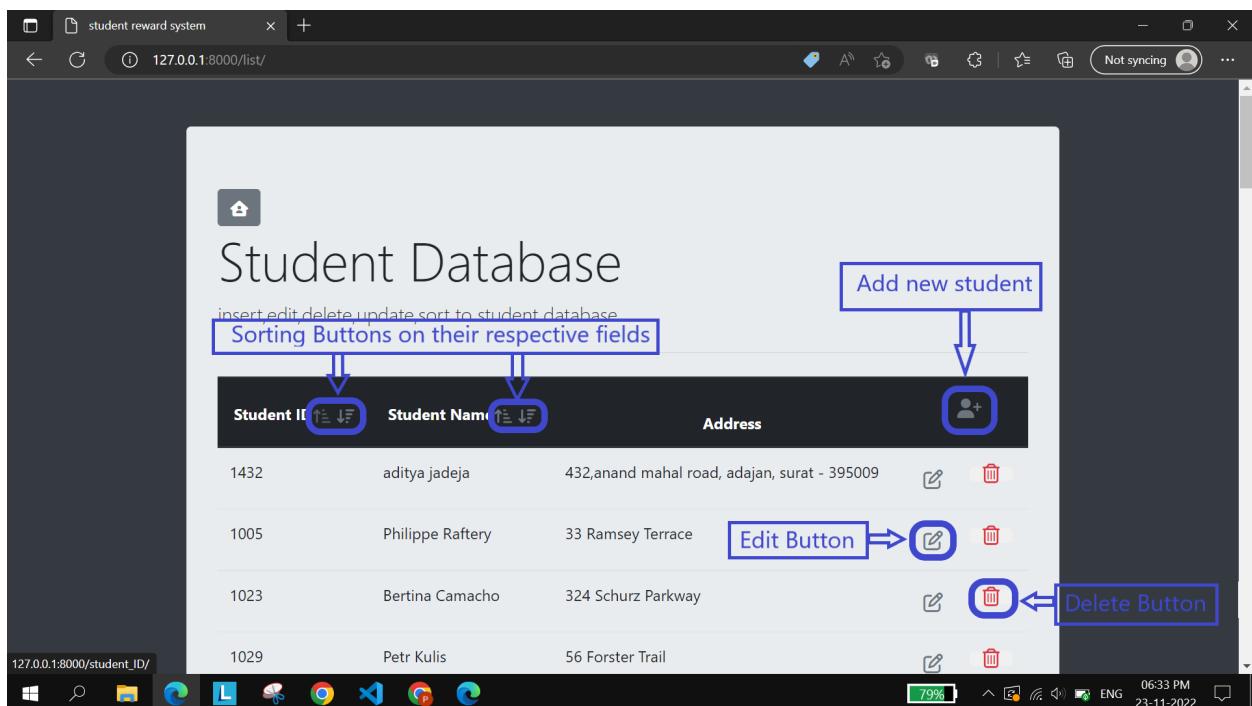


→ We will have the following functionalities:

- i) Student Database
- ii) Register New Student
- iii) Instructor Database
- iv) Course Database

### i) Student Database:

- Admin will be able to see the details of each student and can edit, insert and delete the information of students.
- The sorting buttons will sort the table based on respective fields.



### a) Insertion of student:

→ Entering details of new students:

Student Database

insert,edit,delete,update,sort to student database.

Student ID*	Student Name*
1	Gaurang Parmar
Address	
121,acpc street	

**submit**      **Back to list**

→ New student data is inserted into the table.

Student Database

insert,edit,delete,update,sort to student database.

Student ID	Student Name	Address	
1	Gaurang Parmar	121,acpc street	
1005	Philippe Raftery	111,bhagunagar socity,surat	
1023	Bertina Camacho	324 Schurz Parkway	
1029	Petr Kulis	56 Forster Trail	

**b) Deletion of student:**

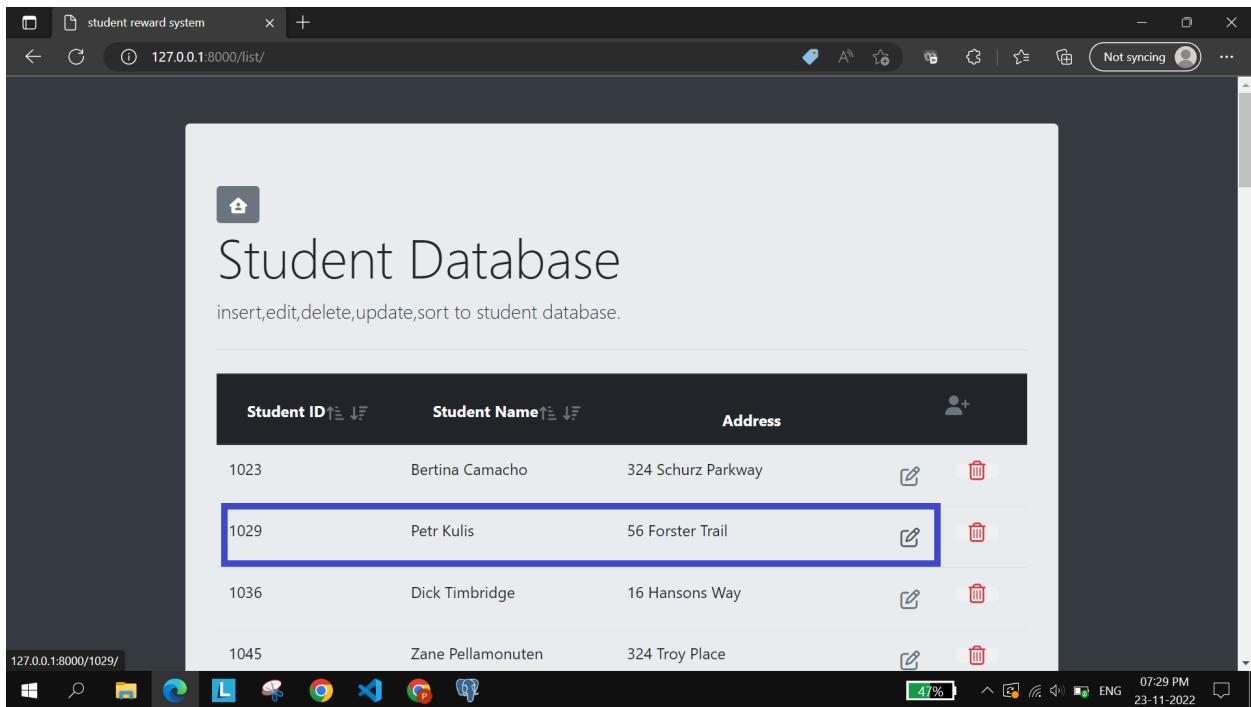
→ Deleting the following tuple:

Student ID	Student Name	Address	Action
1	Gaurang Parmar	121,acpc street	
1005	Philippe Raftery	111,bhagunagar socity,surat	
1023	Bertina Camacho	324 Schurz Parkway	
1029	Petr Kulis	56 Forster Trail	

→ Table after deletion:

Student ID	Student Name	Address	Action
1023	Bertina Camacho	324 Schurz Parkway	
1029	Petr Kulis	56 Forster Trail	
1036	Dick Timbridge	16 Hansons Way	
1045	Zane Pellamonuten	324 Troy Place	

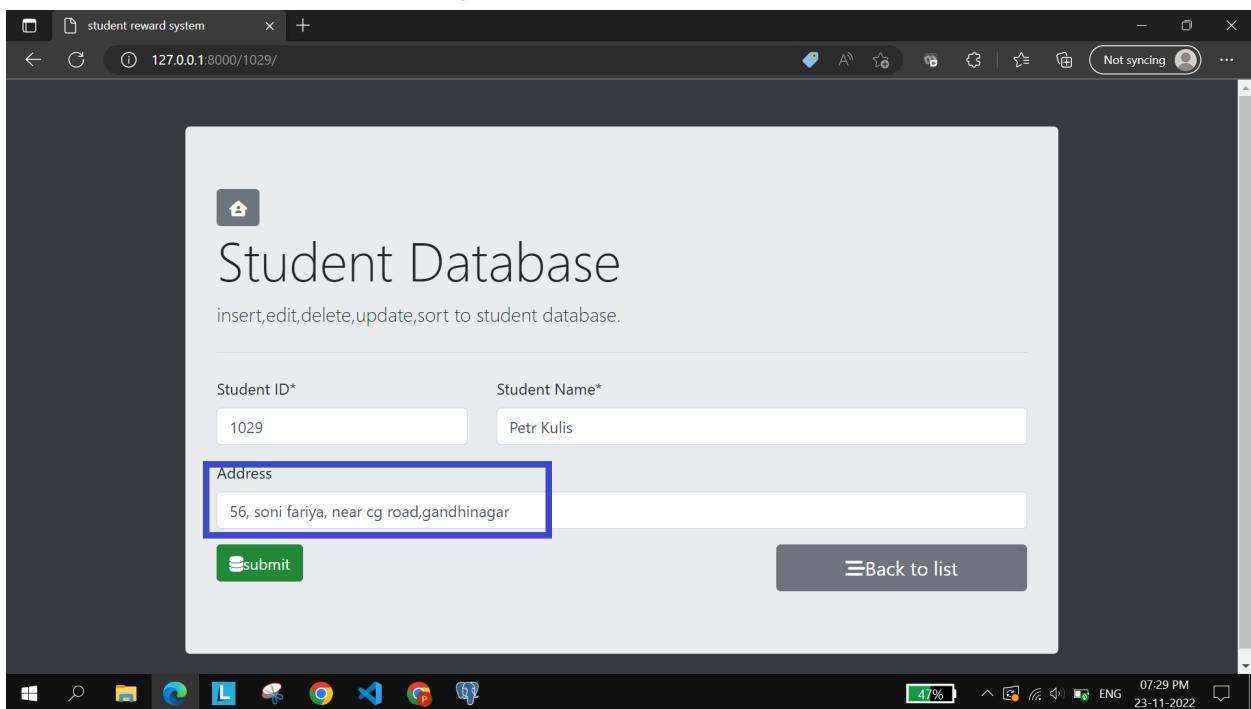
c) updating Student information:  
 → Updating data of following student:



The screenshot shows a web application titled "Student Database". The URL in the address bar is 127.0.0.1:8000/list/. The page displays a table with columns: "Student ID", "Student Name", and "Address". There are edit and delete icons for each row. The row for student ID 1029, name Petr Kulis, and address 56 Forster Trail is selected and highlighted with a blue box.

Student ID	Student Name	Address
1023	Bertina Camacho	324 Schurz Parkway
1029	Petr Kulis	56 Forster Trail
1036	Dick Timbridge	16 Hansons Way
1045	Zane Pellamonuten	324 Troy Place

→ The address field is being updated.



The screenshot shows a web application titled "Student Database". The URL in the address bar is 127.0.0.1:8000/1029/. The page displays a form for updating student information. The "Student ID\*" field contains 1029, and the "Student Name\*" field contains Petr Kulis. The "Address" field is highlighted with a blue box and contains the new value "56, soni fariya, near cg road,gandhinagar".

Student ID*	Student Name*
1029	Petr Kulis
Address	
56, soni fariya, near cg road,gandhinagar	

**submit**      **Back to list**

→ The updated information is stored in the database.

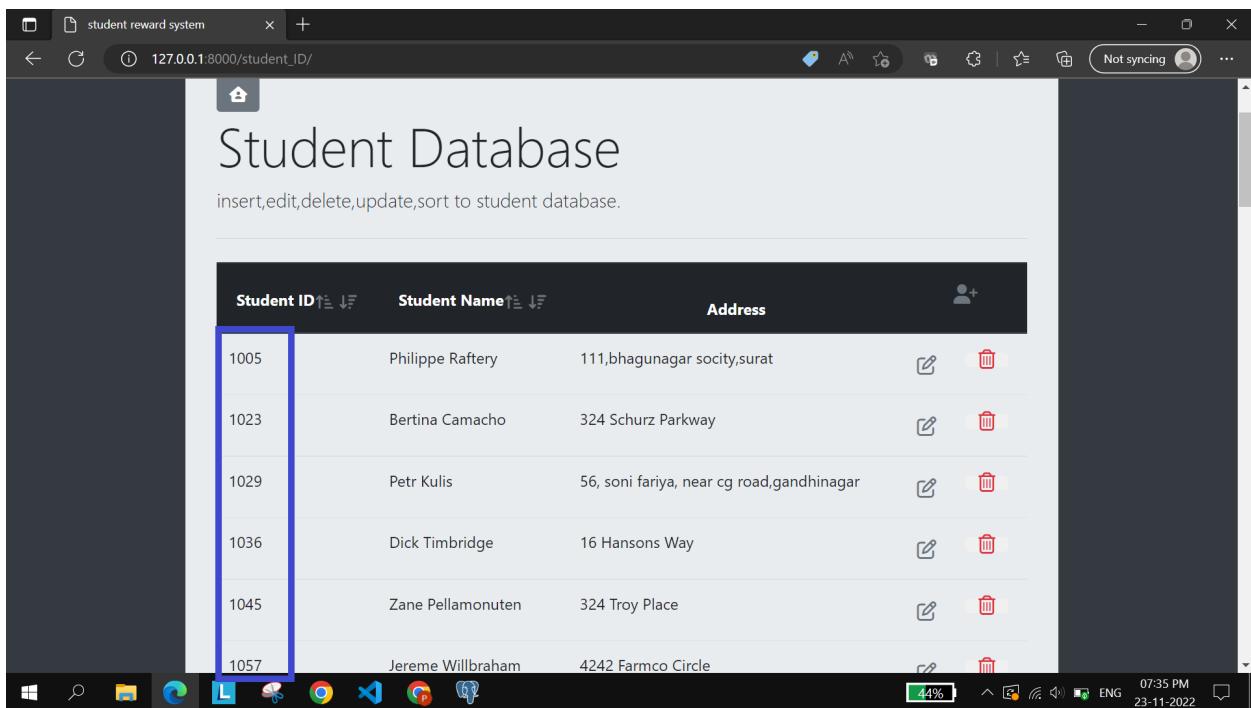
Student ID	Student Name	Address
1005	Philippe Raftery	111,bhagunagar socity,surat
1023	Bertina Camacho	324 Schurz Parkway
1029	Petr Kulis	56, soni fariya, near cg road,gandhinagar
1036	Dick Timbridge	16 Hansons Way

#### d) Sorting The Table:

→ Sorting the table Student\_ID wise:

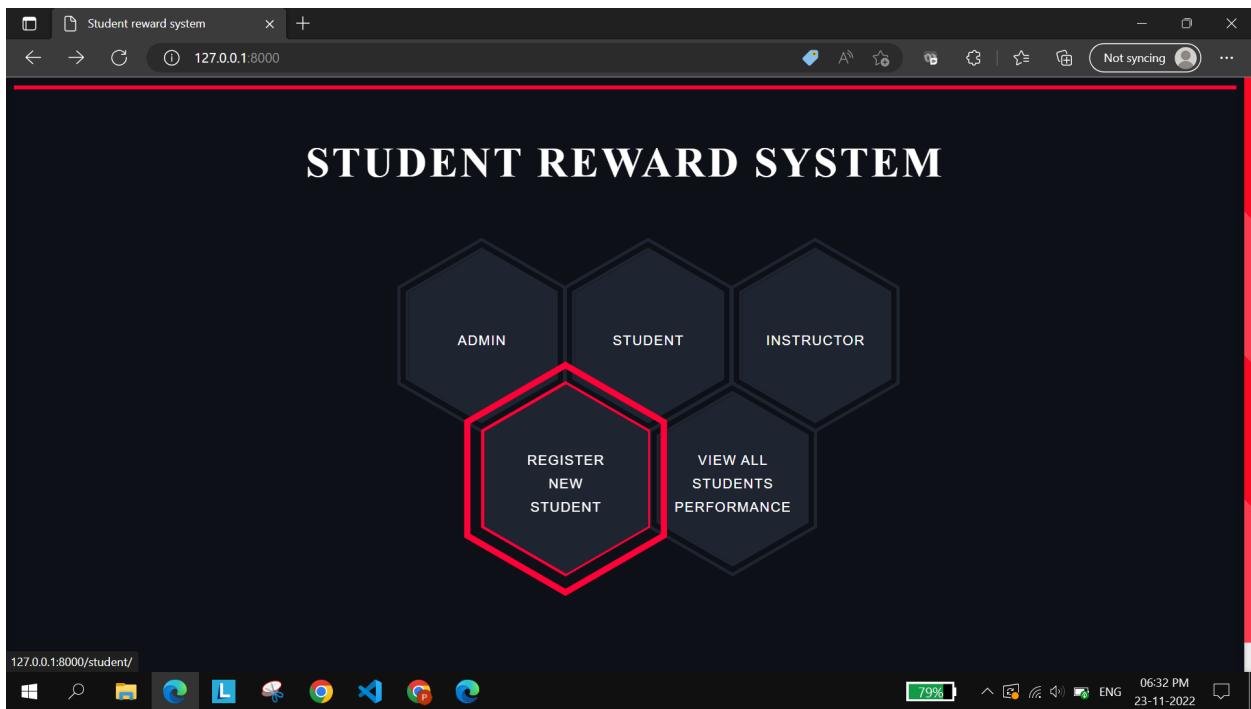
Student ID	Student Name	Address
1045	Zane Pellamonuten	324 Troy Place
1261	Victor Pickervance	744 Loftsgordon Pass
1288	Tymothy Dubois	7 Shopko Plaza
1174	Tuck Rickersy	05336 Bunker Hill Alley
1367	Trev Dellenbrok	201 Manufacturers Court
1251	Tammy Sprulls	41616 Fallview Way

→ Sorted table (Student\_ID wise):



Student ID ↑ ↓	Student Name ↑ ↓	Address	
1005	Philippe Raftery	111,bhagunagar socity,surat	 
1023	Bertina Camacho	324 Schurz Parkway	 
1029	Petr Kulis	56, soni fariya, near cg road,gandhinagar	 
1036	Dick Timbridge	16 Hansons Way	 
1045	Zane Pellamonuten	324 Troy Place	 
1057	Jereme Willbraham	4242 Farmco Circle	 

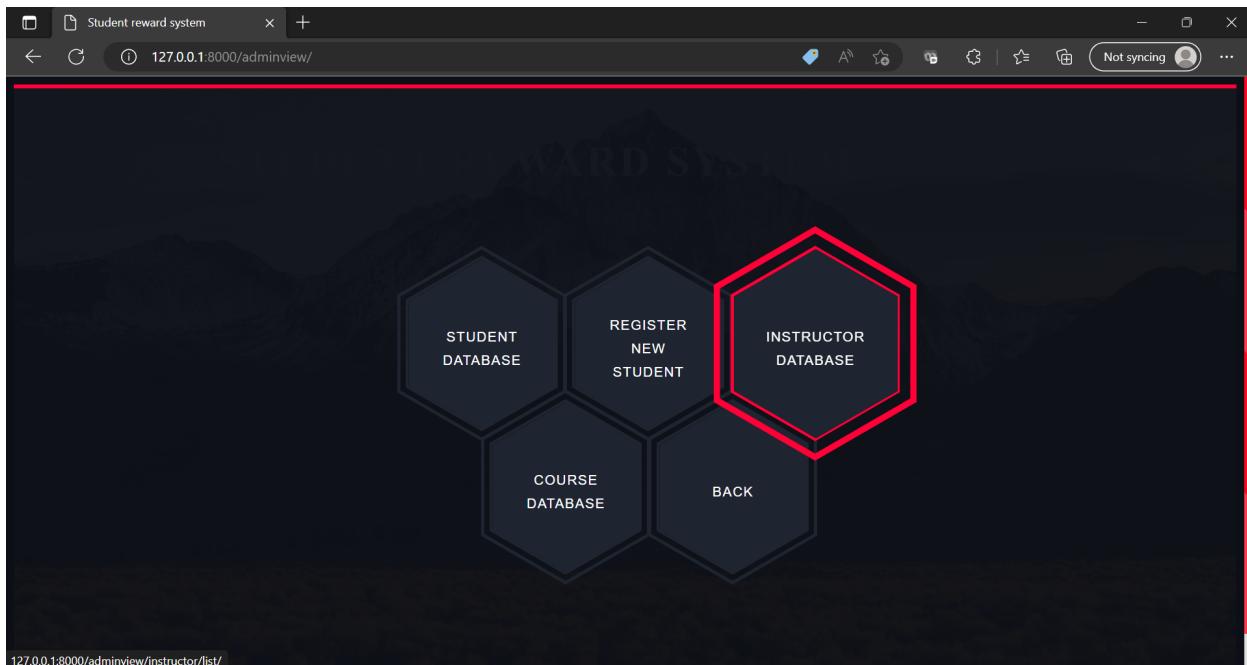
ii) Register New Student:



→ Admin can directly register new students in the database by this functionality and it will be the same as insertion in the Student database.

### iii) Instructor Database:

- Admin will be able to see the details of each instructor and can edit, insert and delete the information of instructors.
- The sorting buttons will sort the table based on respective fields.



The screenshot shows a table titled 'Instructor Database' with the following data:

Instructor ID	Instructor Name	Address	Admin ID
19	Arny Mustard	445 Surrey Trail	67
22	Ned Moyer	451 Maple Wood Pass	78
27	Rufus Probbing	28 Canary Junction	38
29	Sarina Sherlock	7770 Forest Run Court	6
32	Obediah Coneron	3652 Iowa Street	6

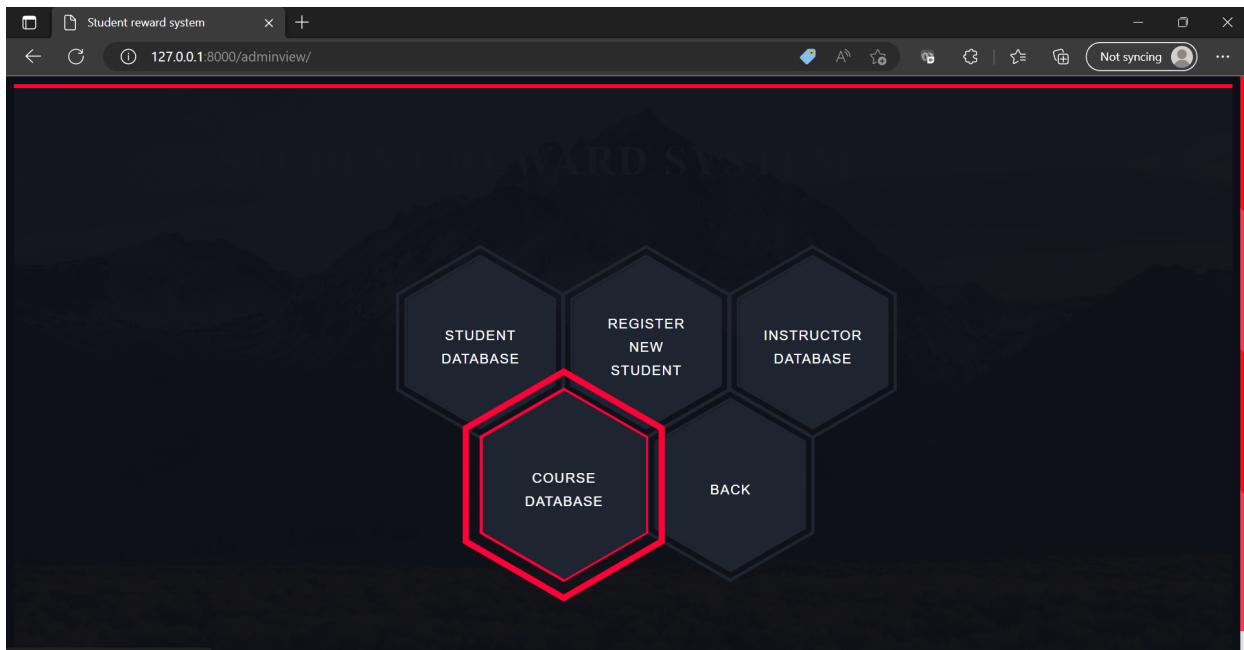
Annotations on the page include:

- A blue box labeled 'Sorting Buttons according to their fields' points to the sorting icons in the header row.
- A blue box labeled 'Add New Instructor' points to the '+' icon in the top right corner.
- A blue box labeled 'Edit Button' points to the edit icon (pencil) in the table row for Instructor ID 27.
- A blue box labeled 'Delete Button' points to the delete icon (trash bin) in the table row for Instructor ID 27.

→ All the operations can be performed in the same manner as described in the Student Database.

#### iv) Course Database:

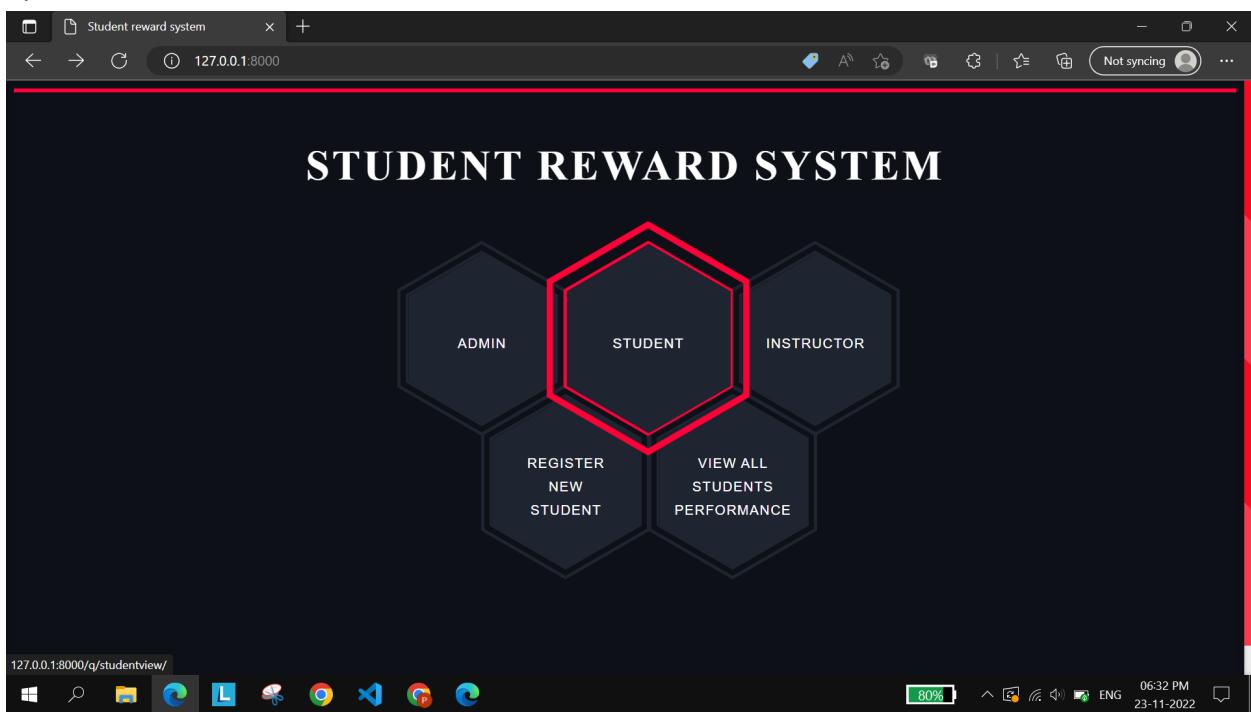
- Admin will be able to see the details of each Course and can edit, insert and delete the information of courses.
- The sorting buttons will sort the table based on respective fields.



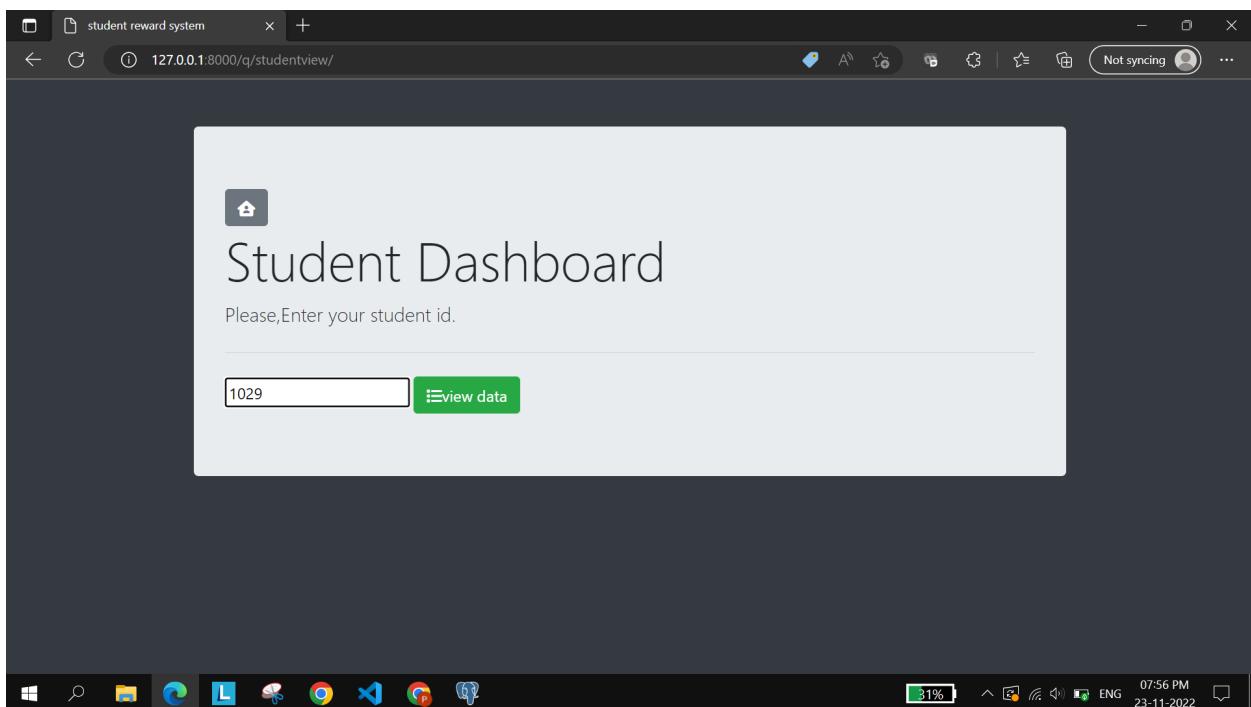
The screenshot shows a list of courses in a table format. The columns are "Course ID", "Course Name", and "Instructor ID". Each row contains course details: 111 (Extra-curricular), 116 (Economics), 203 (Embedded Hardware Design), 215 (DBMS), and 303 (Digital Communication). To the right of the table are "Edit" and "Delete" buttons for each row. Above the table, there is a header "Course Database" and a sub-instruction "insert,edit,delete,update,sort to Course database". A blue box highlights the "Sorting Buttons according to their respective fields" above the table, and another blue box highlights the "Delete Button" next to the "Edit" button in the first row.

Course ID	Course Name	Instructor ID	
111	Extra-curricular	17	
116	Economics	17	
203	Embedded Hardware Design	29	
215	DBMS	27	
303	Digital Communication	22	

## 3) Student View:



→ Enter Student\_ID to show its details:



→ Profile of Student with Student\_ID = 1029.

## Petr Kulis's Profile

current semester profile

Student ID	Student Name	Address
------------	--------------	---------

1029	Petr Kulis	56, soni fariya, near cg road,gandhinagar
------	------------	---

## Academic Profile

Course ID	Total Course Reward Points
-----------	----------------------------

402	49.16666667
116	33.166666667
215	60.16666667
107	42.5
304	51.83333333
203	48.166666667
303	33.333333333

## Non-Academic Profile

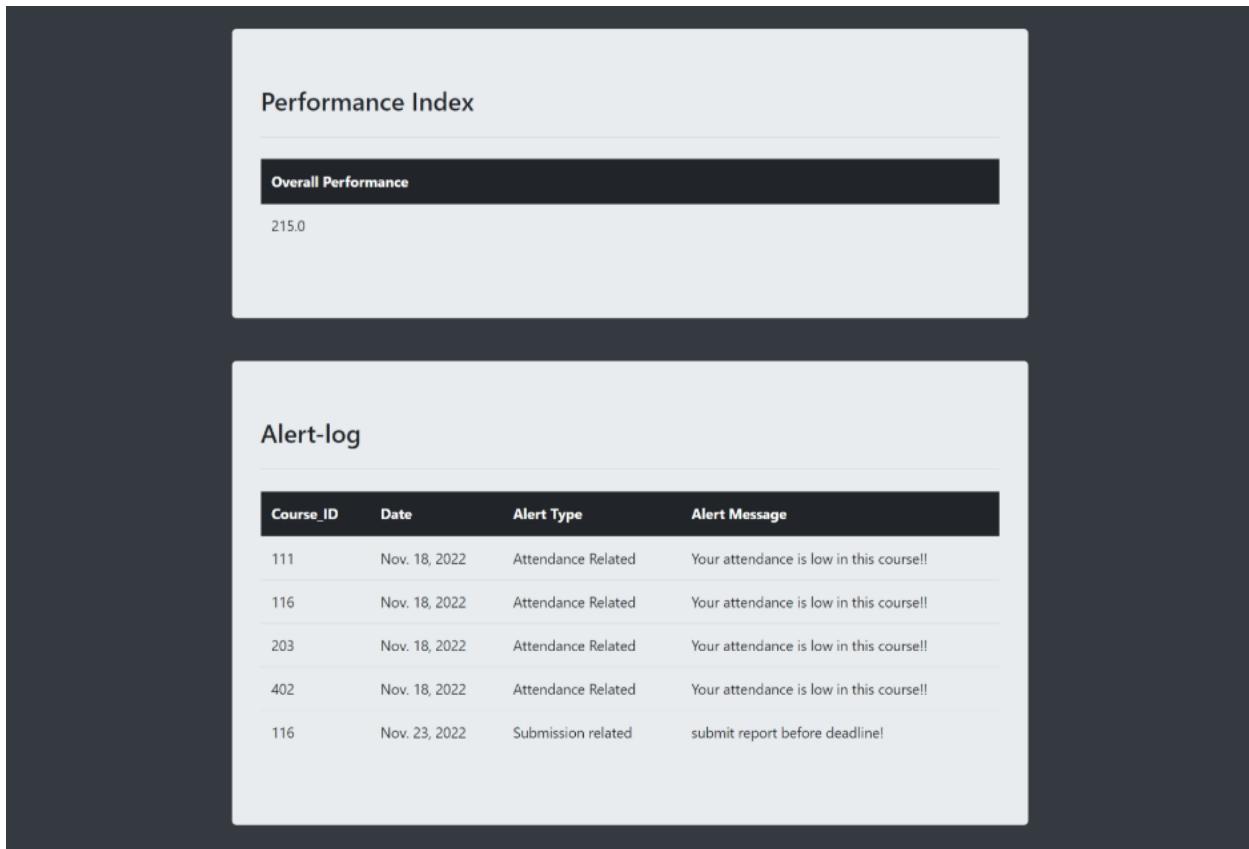
Activity Name	Activity Reward Points
---------------	------------------------

Coding	0.0
Dance	0.0
Drama	20.0
Music	20.0
Sports	20.0

## Combined Performance

Final Academic score	Final Non-Academic score
----------------------	--------------------------

318.333333337	60.0
---------------	------



→ Here all the data is displayed with the use of sql queries.

**Some of the sample queries used are as following.**

- 1) Show Academic Course-wise total Reward points of student whose Student\_ID = 1029.**

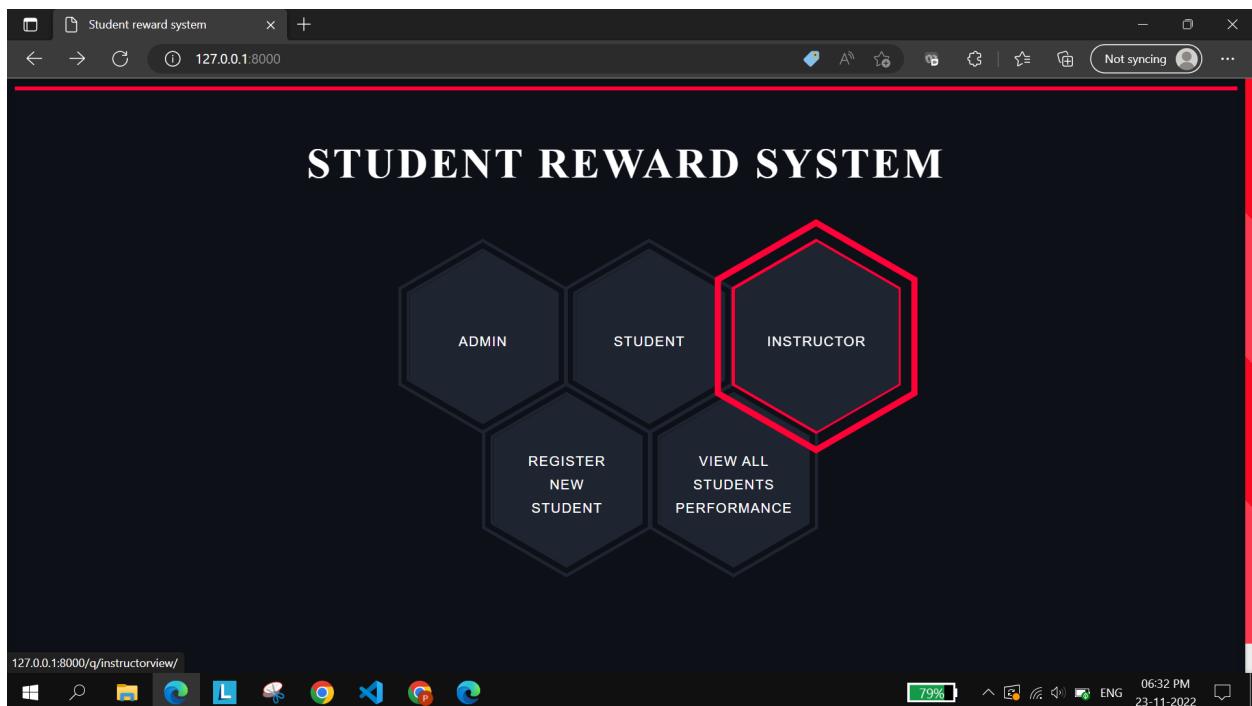
```
SELECT "course_ID", SUM(reward_points)
FROM srsapp_academic_reward
WHERE "student_ID"=1029
GROUP BY "course_ID";
```

- 2) Show Total Academic Reward points and total Non-academic reward points of student whose Student\_ID = 1029.**

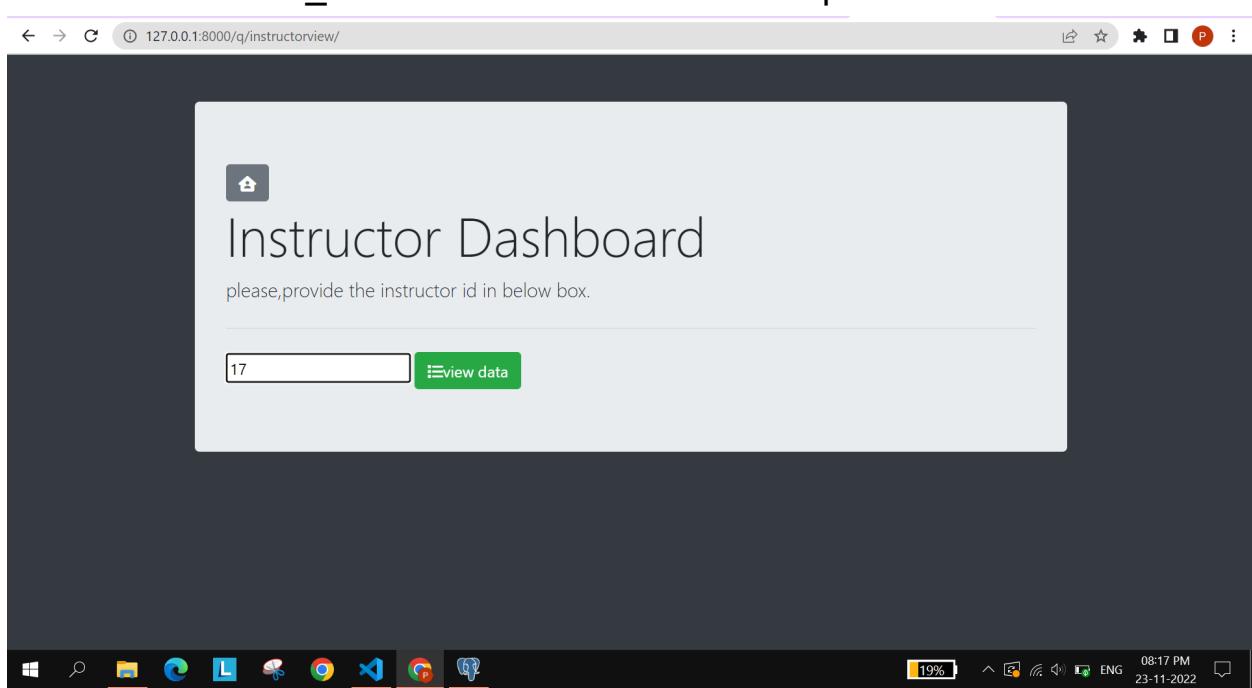
```
SELECT *
FROM academic_score_final NATURAL JOIN non_academic_score_final
```

WHERE academic\_score\_final."student\_ID" = 1029;

#### 4) Instructor View:



→ Enter Instructor\_ID to view the dashboard of a particular instructor.



- Dashboard of Instructor with Instructor\_ID = 17.
- Instructors can update mark send alerts to a student.

The screenshot shows a web browser window with the URL `127.0.0.1:8000/q/instructorview/`. The page title is "Chevalier Minmagh's Profile" and the subtext "Instructor\_ID = 17". At the top left, there are three icons: a house (Home), a pencil (Edit), and a bell (Alert). A blue box labeled "Update Button" points to the edit icon. Another blue box labeled "Alert Button" points to the bell icon. Below the title is a table with the following data:

Course ID	Student ID	Activity Name	Obtained marks	Reward Points
116	1029	Endsem	26.0	13.0
116	1029	Lab	1.0	0.666666667
116	1029	Midsem	10.0	5.0
116	1029	Quiz	4.0	4.0
116	1029	Viva	7.0	10.5
116	1057	Endsem	54.0	27.0

The browser status bar at the bottom shows "19%" battery, "08:17 PM", "ENG", and the date "23-11-2022".

### a) Updating Student's marks:

This screenshot shows the same dashboard as above, but the second row (Student ID 1029, Activity Lab) is highlighted with a blue selection box. The rest of the interface is identical to the first screenshot.

→ Student's marks are updated and the reward points are calculated accordingly with use of triggers.

Course ID	Student ID	Activity Name	Obtained marks	Reward Points
116	1029	Endsem	26.0	13.0
116	1029	Midsem	10.0	5.0
116	1029	Quiz	4.0	4.0
116	1029	Viva	7.0	10.5
116	1029	Lab	10.0	6.666666666666667
116	1057	Endsem	54.0	27.0
116	1057	Lab	3.0	2.0
116	1057	Midsem	18.0	9.0
116	1057	Quiz	7.0	7.0

## b) Sending Alerts to students:

Send Alert To Student

Student ID:  Course ID:

Alert Type:  Alert Message:

→ The alert is successfully sent and is stored in the alert-log of the respective student.

Course_ID	Date	Alert Type	Alert Message
107	Nov. 18, 2022	Attendance Related	Your attendance is low in this course!!
111	Nov. 18, 2022	reward_related	You have been rewarded for top being in top 5 performers
203	Nov. 18, 2022	Attendance Related	Your attendance is low in this course!!
303	Nov. 18, 2022	Attendance Related	Your attendance is low in this course!!
304	Nov. 18, 2022	Attendance Related	Your attendance is low in this course!!
303	Nov. 23, 2022	misc	PTM tomorrow
107	Nov. 23, 2022	misc	PTM tomorrow
215	Nov. 23, 2022	Lab	Please, submit your pending lab reports!

## 5) Performance Page:

→ We can see the final performance of students on this page and by using sort buttons, top performers and weak performers can be identified.

The screenshot shows a web browser window with the URL 127.0.0.1:8000/performance/. The title of the page is "Performance Database". A sub-instruction "Sorting Buttons according to their respective fields" is overlaid on the page, with arrows pointing to the sorting icons for "Student ID" and "Overall Score". The table displays student IDs and their overall scores:

Student ID	Overall Score
1023	210.8
1036	196.4
1045	243.9
1057	195.3
1061	215.4
1064	216.7
	217.0

### Github Repository Link:

[https://github.com/Het99/STUDENT\\_REWARD\\_SYSTEM](https://github.com/Het99/STUDENT_REWARD_SYSTEM)