Project Title
College Management System

"A Smart Database Solution for Managing Students, Professors, Courses, and Enrollments Efficiently".

Prepared By: Akshata Shinde

Module: SQL

Introduction

In today’s fast-paced academic environment, managing college operations manually has become inefficient and error-prone. The **College Management System** aims to provide a streamlined, automated solution that simplifies the administration of academic, student, and faculty-related data. This SQL-based system serves as a virtual backbone for the institution, organizing critical information such as student records, course, professor details, and enrollment data in a structured and accessible way.

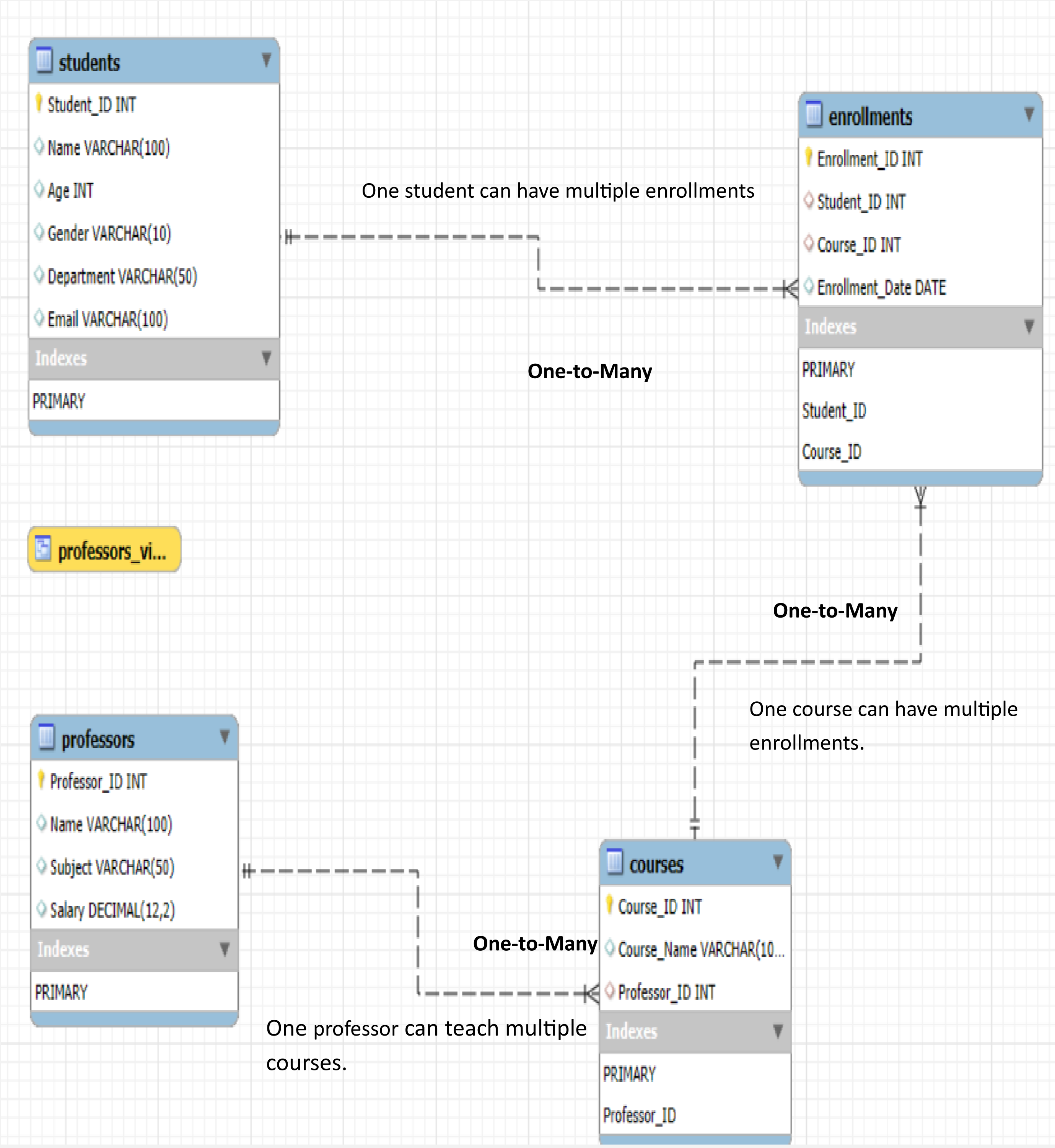
The system is designed to help administrative staff and faculty members focus more on their core activities by reducing the time spent on paperwork and repetitive tasks. Through well-defined relational tables and carefully constructed queries, it offers efficient data storage, rapid retrieval, and insightful reporting.

From a technical standpoint, the project emphasizes practical application of database management concepts such as table creation, data integrity enforcement, relational constraints, and advanced data querying techniques. Users can perform various operations, including inserting new records, updating existing data, and generating reports to track student progress, faculty workload, and course popularity.

Moreover, this system aligns with analytical thinking by enabling statistical insights from the stored data. For example, trends in department enrollments, professor workload analysis, and course success rates can be easily derived, supporting informed decision-making by college management.

This project demonstrates a fusion of database design with real-world academic needs, showcasing the power of SQL in building solutions that are not only functional but also insightful and efficient.

ER DIAGRAM



One student can have multiple enrollments

One-to-Many

One-to-Many

One course can have multiple enrollments.

One-to-Many

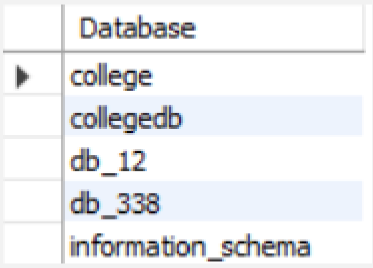
One professor can teach multiple courses.

Databases :

```
Create Database CollegeDB;

use CollegeDB;

Show databases;
```



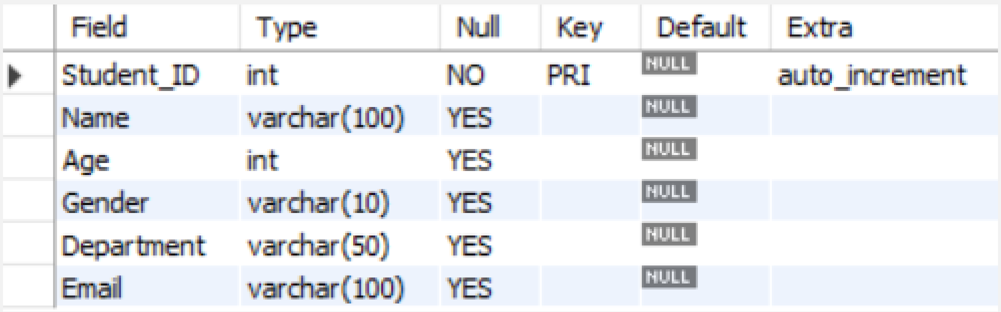
	Database
▶	college
	collegedb
	db_12
	db_338
	information_schema

DATA DEFINITION LANGUAGE (DDL) :

1. Create Tables :

a) Students

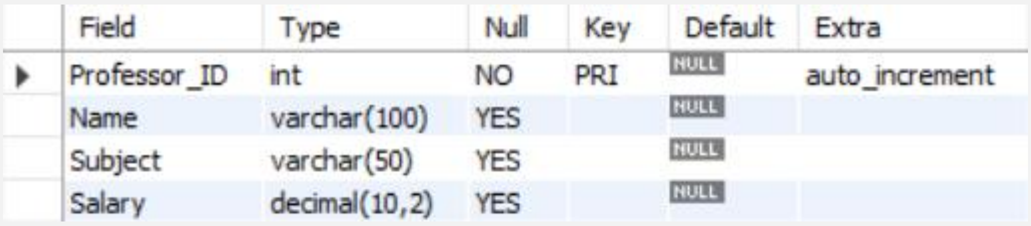
```
CREATE TABLE Students (
    Student_ID  INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Department VARCHAR(50),
    Email VARCHAR(100) );
desc Students;
```



	Field	Type	Null	Key	Default	Extra
▶	Student_ID	int	NO	PRI	NULL	auto_increment
	Name	varchar(100)	YES		NULL	
	Age	int	YES		NULL	
	Gender	varchar(10)	YES		NULL	
	Department	varchar(50)	YES		NULL	
	Email	varchar(100)	YES		NULL	

b) Professors

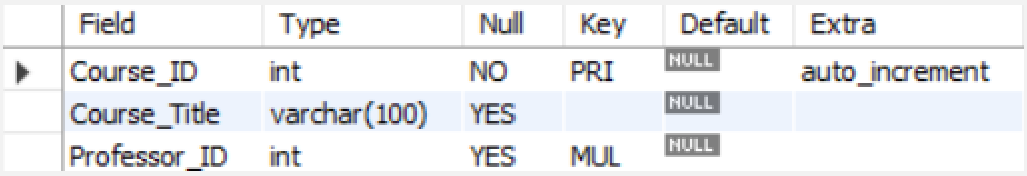
```
CREATE TABLE Professors (
    Professor_ID  INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100),
    Subject VARCHAR(50),
    Salary DECIMAL(10,2) );
desc Professors;
```



	Field	Type	Null	Key	Default	Extra
▶	Professor_ID	int	NO	PRI	NULL	auto_increment
	Name	varchar(100)	YES		NULL	
	Subject	varchar(50)	YES		NULL	
	Salary	decimal(10,2)	YES		NULL	

c) Courses

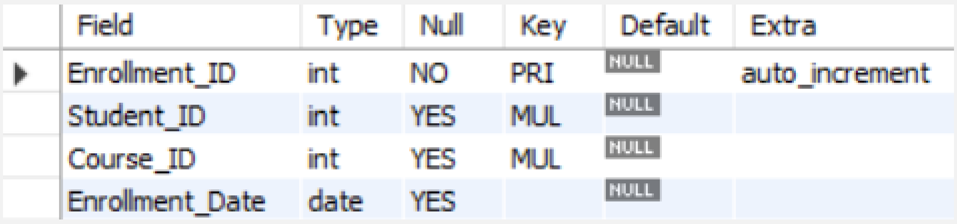
```
CREATE TABLE Courses (
    Course_ID INT AUTO_INCREMENT PRIMARY KEY,
    Course_Title VARCHAR(100),
    Professor_ID  INT,  FOREIGN  KEY  (Professor_ID)  REFERENCES
Professors(Professor_ID) );
desc Courses;
```



	Field	Type	Null	Key	Default	Extra
▶	Course_ID	int	NO	PRI	NULL	auto_increment
	Course_Title	varchar(100)	YES		NULL	
	Professor_ID	int	YES	MUL	NULL	

d) Enrollments

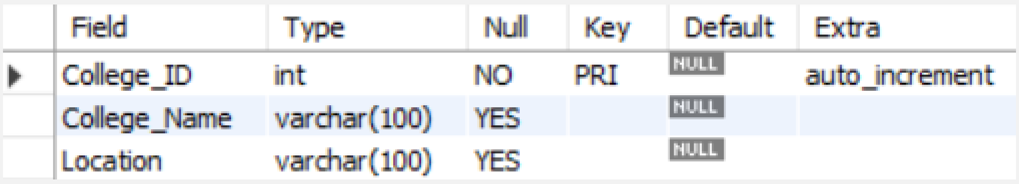
```
CREATE TABLE Enrollments (
    Enrollment_ID INT AUTO_INCREMENT PRIMARY KEY,
    Student_ID INT,
    Course_ID INT,
    Enrollment_Date DATE,
    FOREIGN KEY (Student_ID) REFERENCES Students(Student_ID),
    FOREIGN KEY (Course_ID) REFERENCES Courses(Course_ID)
);
desc Enrollments;
```



Field	Type	Null	Key	Default	Extra
Enrollment_ID	int	NO	PRI	NULL	auto_increment
Student_ID	int	YES	MUL	NULL	
Course_ID	int	YES	MUL	NULL	
Enrollment_Date	date	YES		NULL	

e) College_Info

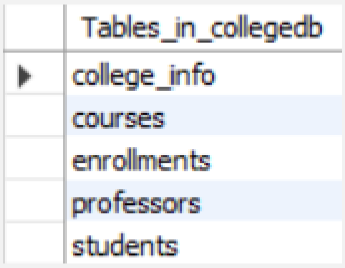
```
CREATE TABLE College_Info (
    College_ID INT AUTO_INCREMENT PRIMARY KEY,
    College_Name VARCHAR(100),
    Location VARCHAR(100));
desc College_Info;
```



Field	Type	Null	Key	Default	Extra
College_ID	int	NO	PRI	NULL	auto_increment
College_Name	varchar(100)	YES		NULL	
Location	varchar(100)	YES		NULL	

Tables in CollegeDB

```
SHOW TABLES;
```

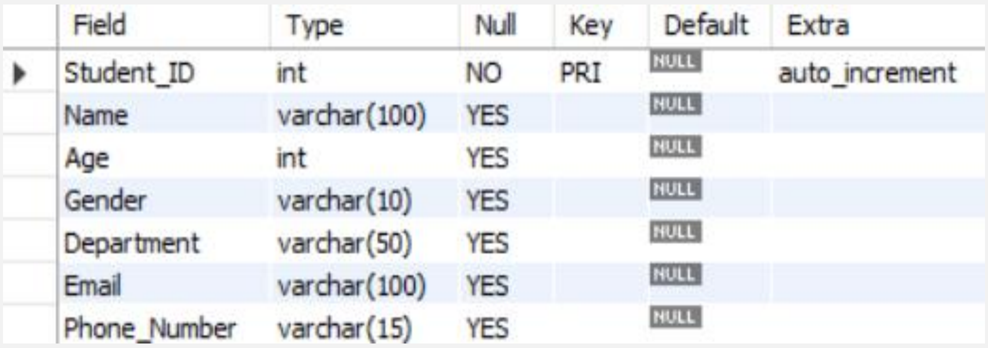


Tables_in_collegedb
college_info
courses
enrollments
professors
students

2. Alter Table:

➢ Add Column: Add Phone_Number column to Students table

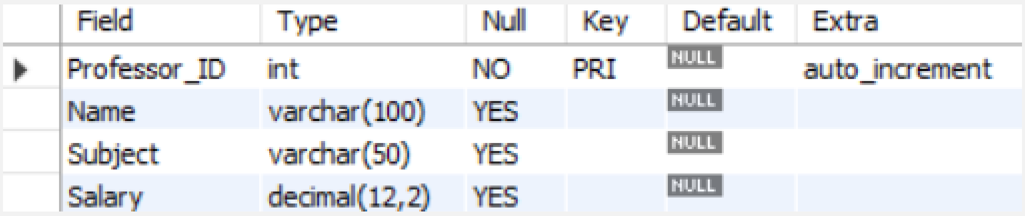
```
ALTER TABLE Students ADD COLUMN Phone_Number
VARCHAR(15);
```



	Field	Type	Null	Key	Default	Extra
▶	Student_ID	int	NO	PRI	NULL	auto_increment
	Name	varchar(100)	YES		NULL	
	Age	int	YES		NULL	
	Gender	varchar(10)	YES		NULL	
	Department	varchar(50)	YES		NULL	
	Email	varchar(100)	YES		NULL	
	Phone_Number	varchar(15)	YES		NULL	

➢ Modify Column: Modify Salary column in Professors table

```
ALTER TABLE Professors MODIFY Salary DECIMAL(12,2);
```

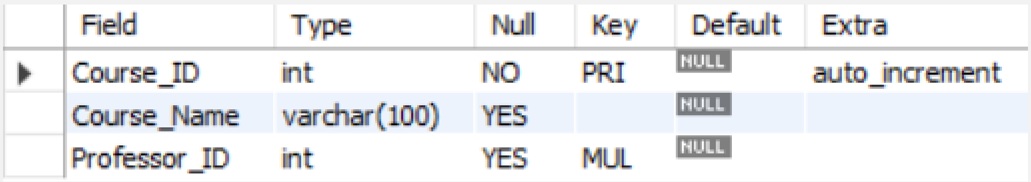


	Field	Type	Null	Key	Default	Extra
▶	Professor_ID	int	NO	PRI	NULL	auto_increment
	Name	varchar(100)	YES		NULL	
	Subject	varchar(50)	YES		NULL	
	Salary	decimal(12,2)	YES		NULL	

➢ Rename Column:

Rename column Course_Title to Course_Name in Courses table

```
ALTER TABLE Courses RENAME COLUMN Course_Title TO
Course_Name;
```

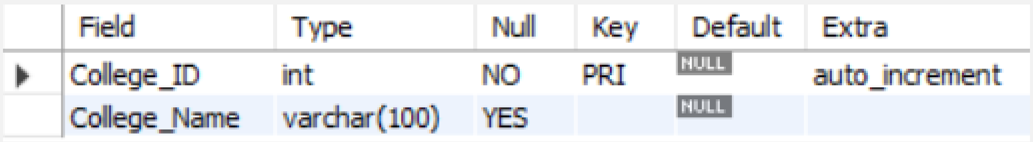


	Field	Type	Null	Key	Default	Extra
▶	Course_ID	int	NO	PRI	NULL	auto_increment
	Course_Name	varchar(100)	YES		NULL	
	Professor_ID	int	YES	MUL	NULL	

- **Drop column:**

Drop column Location from College_Info.

ALTER TABLE College_Info DROP COLUMN Location;



	Field	Type	Null	Key	Default	Extra
▶	College_ID	int	NO	PRI	NULL	auto_increment
	College_Name	varchar(100)	YES		NULL	

Drop the Phone_Number column from the Student table

ALTER TABLE Students DROP COLUMN Phone_Number;

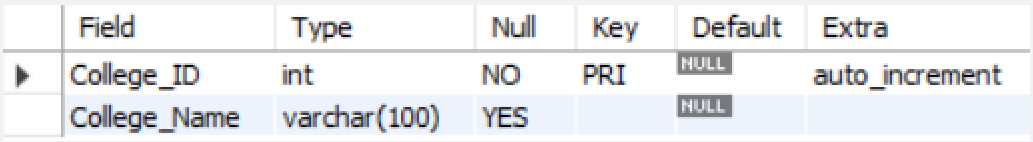
select*from Students;

- **Rename Table:**

Rename College_Info table to College_Details.

ALTER TABLE College_Info RENAME TO College_Details;

desc College_Details;



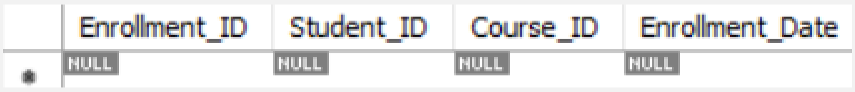
	Field	Type	Null	Key	Default	Extra
▶	College_ID	int	NO	PRI	NULL	auto_increment
	College_Name	varchar(100)	YES		NULL	

3. **Truncate table :**

Truncate Enrollments table to remove all data

TRUNCATE TABLE Enrollments;

SELECT * FROM Enrollments;



	Enrollment_ID	Student_ID	Course_ID	Enrollment_Date
*	NULL	NULL	NULL	NULL

4. Drop Table:

Drop College_Details table completely

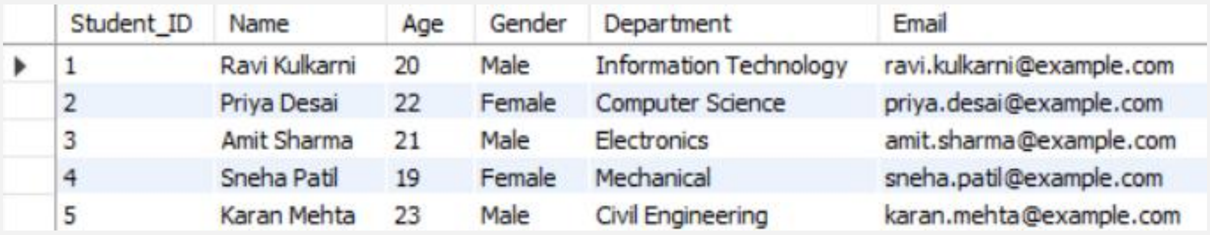
```
DROP TABLE College_Details;
```

DATA MANIPULATION LANGUAGE (DML)

1. Insert into table :

```
INSERT INTO Students (Name, Age, Gender, Department, Email)VALUES
('Karan Mehta', 23, 'Male', 'Civil Engineering', 'karan.mehta@example.com');
```

```
select*from Students;
```

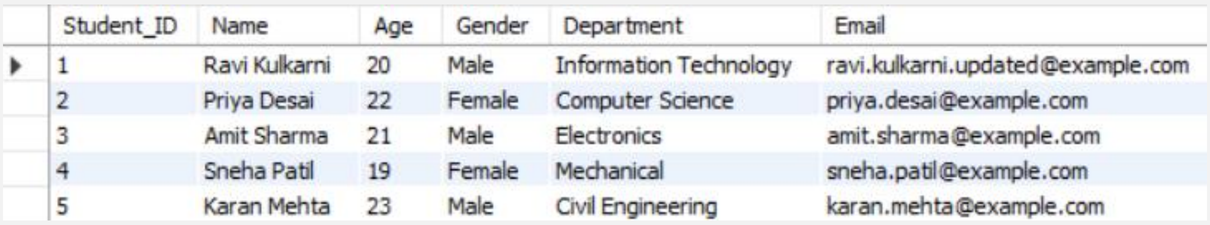


Student_ID	Name	Age	Gender	Department	Email
1	Ravi Kulkarni	20	Male	Information Technology	ravi.kulkarni@example.com
2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com
4	Sneha Patil	19	Female	Mechanical	sneha.patil@example.com
5	Karan Mehta	23	Male	Civil Engineering	karan.mehta@example.com

2. Update into Table :

Update Ravi Kulkarni’s Email in Students table.

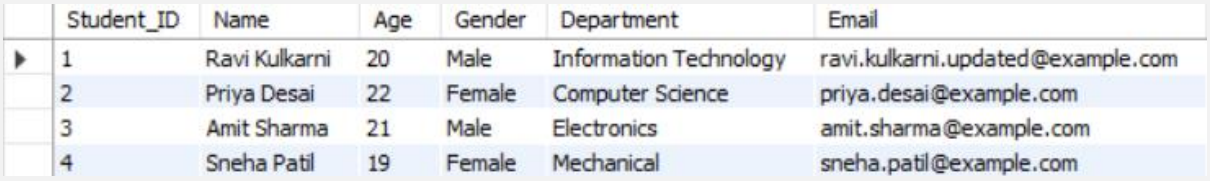
```
UPDATE Students SET Email = 'ravi.kulkarni.updated@example.com' WHERE
Student_ID = 1;
```



Student_ID	Name	Age	Gender	Department	Email
1	Ravi Kulkarni	20	Male	Information Technology	ravi.kulkarni.updated@example.com
2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com
4	Sneha Patil	19	Female	Mechanical	sneha.patil@example.com
5	Karan Mehta	23	Male	Civil Engineering	karan.mehta@example.com

3. Delete from table :

```
DELETE FROM Students WHERE Student_ID=5;
```



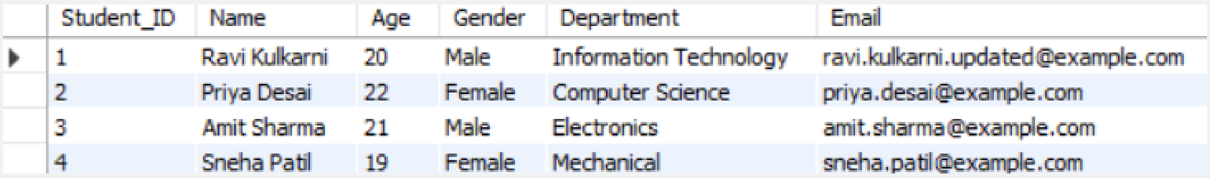
	Student_ID	Name	Age	Gender	Department	Email
▶	1	Ravi Kulkarni	20	Male	Information Technology	ravi.kulkarni.updated@example.com
	2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
	3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com
	4	Sneha Patil	19	Female	Mechanical	sneha.patil@example.com

DATA QUERY LANGUAGE (DQL)

1.Select Queries

a) Select Query for entire data

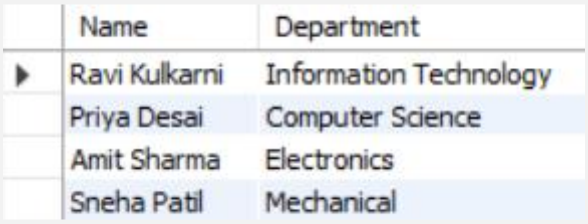
```
SELECT * FROM Students;
```



	Student_ID	Name	Age	Gender	Department	Email
▶	1	Ravi Kulkarni	20	Male	Information Technology	ravi.kulkarni.updated@example.com
	2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
	3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com
	4	Sneha Patil	19	Female	Mechanical	sneha.patil@example.com

b) Select Specific columns

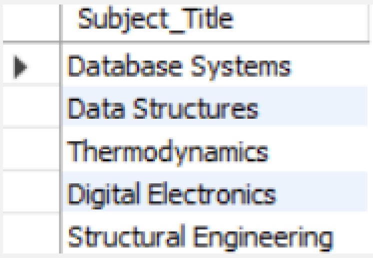
```
SELECT Name, Department FROM Students;
```



	Name	Department
▶	Ravi Kulkarni	Information Technology
	Priya Desai	Computer Science
	Amit Sharma	Electronics
	Sneha Patil	Mechanical

c) Using Alias

```
SELECT Course_Name AS Subject_Title FROM Courses;
```

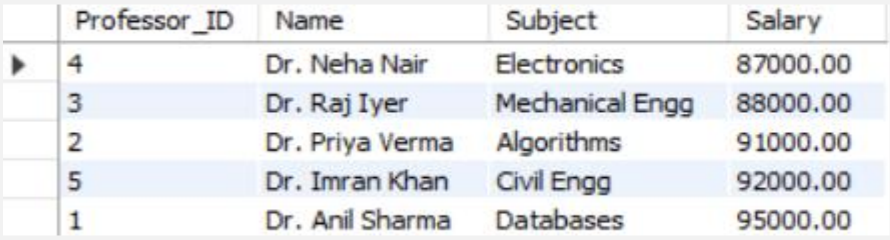


	Subject_Title
▶	Database Systems
	Data Structures
	Thermodynamics
	Digital Electronics
	Structural Engineering

2. Order By

a. Professors by salary (ascending)

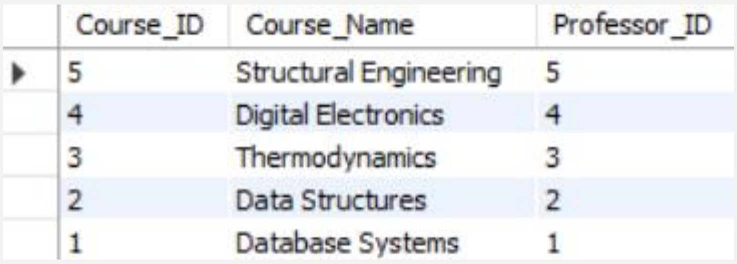
```
SELECT * FROM Professors ORDER BY Salary ASC;
```



	Professor_ID	Name	Subject	Salary
▶	4	Dr. Neha Nair	Electronics	87000.00
	3	Dr. Raj Iyer	Mechanical Engg	88000.00
	2	Dr. Priya Verma	Algorithms	91000.00
	5	Dr. Imran Khan	Civil Engg	92000.00
	1	Dr. Anil Sharma	Databases	95000.00

b. Courses in descending order by Course_ID

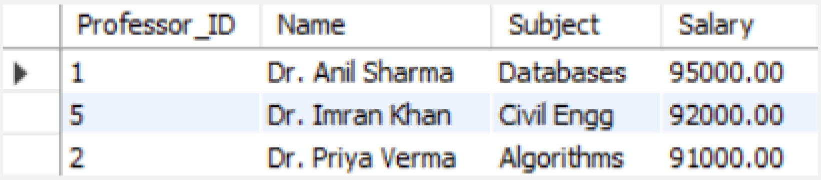
```
SELECT * FROM Courses ORDER BY Course_ID DESC;
```



	Course_ID	Course_Name	Professor_ID
▶	5	Structural Engineering	5
	4	Digital Electronics	4
	3	Thermodynamics	3
	2	Data Structures	2
	1	Database Systems	1

3. Limit Query: Display Top 3 highest-paid professors

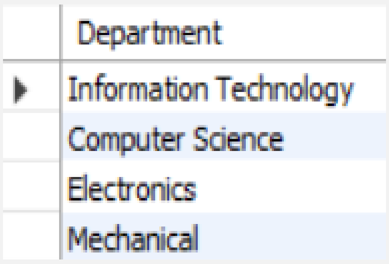
```
SELECT * FROM Professors ORDER BY Salary DESC LIMIT 3;
```



	Professor_ID	Name	Subject	Salary
▶	1	Dr. Anil Sharma	Databases	95000.00
	5	Dr. Imran Khan	Civil Engg	92000.00
	2	Dr. Priya Verma	Algorithms	91000.00

4.Distinct Query: Display Unique departments

```
SELECT DISTINCT Department FROM Students;
```



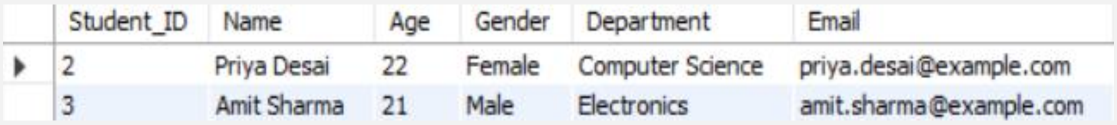
	Department
▶	Information Technology
	Computer Science
	Electronics
	Mechanical

5.Where Clause

i. Comparison Operator

- **Students older than 20**

```
SELECT * FROM Students WHERE Age > 20;
```



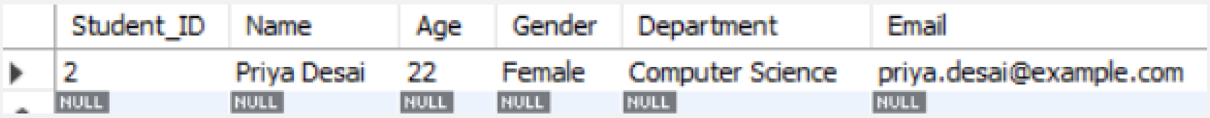
Student_ID	Name	Age	Gender	Department	Email
2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com

ii. Logical Operator

- **Using AND Operator**

Female students in CS

```
SELECT * FROM Students WHERE Gender= 'Female'  AND
Department= 'Computer Science';
```

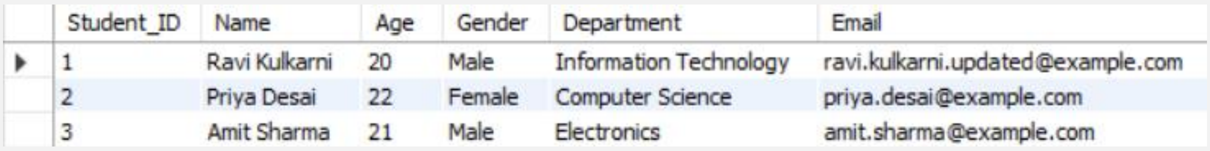


Student_ID	Name	Age	Gender	Department	Email
2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
NULL	NULL	NULL	NULL	NULL	NULL

- **Using BETWEEN operator**

Select Students whose Age is between 20 and 22.

```
SELECT * FROM Students WHERE Age BETWEEN 20 AND 22;
```

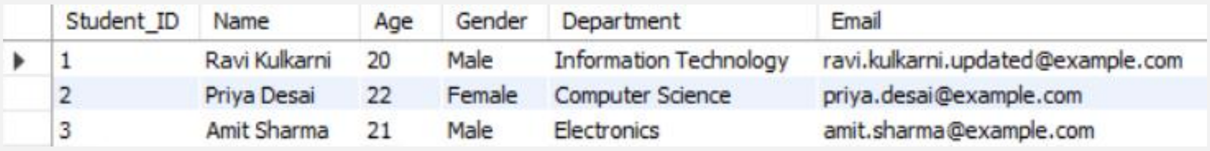


Student_ID	Name	Age	Gender	Department	Email
1	Ravi Kulkarni	20	Male	Information Technology	ravi.kulkarni.updated@example.com
2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com

- **Using NOT IN operator**

Select Students who are not from Mechanical or Civil.

```
SELECT * FROM Students WHERE Department NOT IN ('Mechanical',
'Civil Engineering');
```

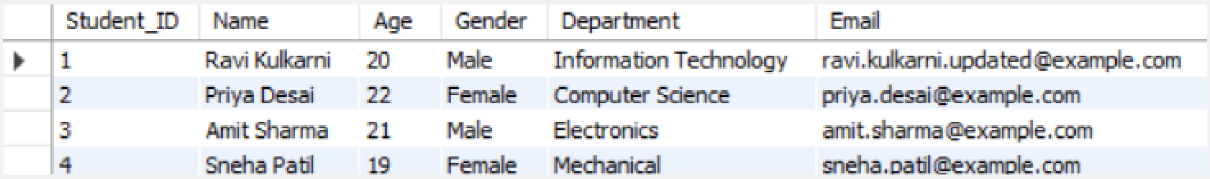


	Student_ID	Name	Age	Gender	Department	Email
▶	1	Ravi Kulkarni	20	Male	Information Technology	ravi.kulkarni.updated@example.com
	2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
	3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com

- **Using IS NOT NULL operator**

Select Students who have provided email.

```
SELECT * FROM Students WHERE Email IS NOT NULL;
```

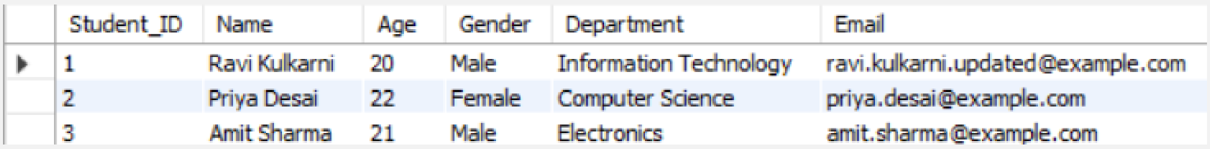


	Student_ID	Name	Age	Gender	Department	Email
▶	1	Ravi Kulkarni	20	Male	Information Technology	ravi.kulkarni.updated@example.com
	2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
	3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com
	4	Sneha Patil	19	Female	Mechanical	sneha.patil@example.com

- **Using ANY operator**

Select Students older than at least one female student.

```
SELECT * FROM Students
WHERE Age > ANY (SELECT Age FROM Students WHERE Gender =
'Female');
```



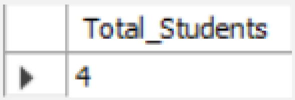
	Student_ID	Name	Age	Gender	Department	Email
▶	1	Ravi Kulkarni	20	Male	Information Technology	ravi.kulkarni.updated@example.com
	2	Priya Desai	22	Female	Computer Science	priya.desai@example.com
	3	Amit Sharma	21	Male	Electronics	amit.sharma@example.com

6. Aggregate Functions

a) Count Function:

Select count of students are there in the Students table

```
SELECT COUNT(*) AS Total_Students FROM Students;
```

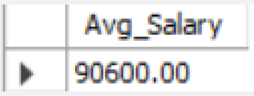


	Total_Students
▶	4

b) Average Function with round Function:

Find the average salary of all professors

```
SELECT ROUND(AVG(Salary),2) AS Avg_Salary FROM Professors;
```

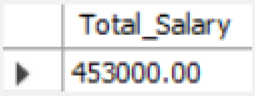


	Avg_Salary
▶	90600.00

c) Sum Function:

Find total salary paid to all professors

```
SELECT SUM(Salary) AS Total_Salary FROM Professors;
```

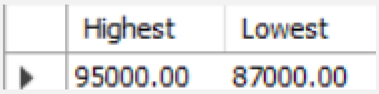


	Total_Salary
▶	453000.00

d) Max , Min Function:

Find the highest and lowest salary among professors.

```
SELECT MAX(Salary) AS Highest, MIN(Salary) AS Lowest FROM
Professors;
```

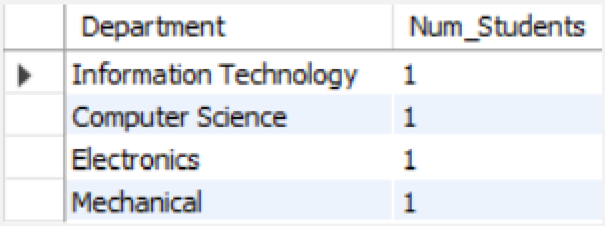


	Highest	Lowest
▶	95000.00	87000.00

7. Group by clause :

- **How many students are there in each department**

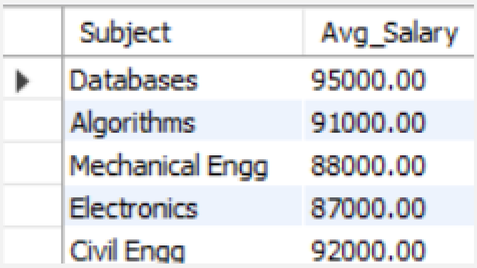
```
SELECT Department, COUNT(Student_ID) AS Num_Students
FROM Students
GROUP BY Department;
```



	Department	Num_Students
▶	Information Technology	1
	Computer Science	1
	Electronics	1
	Mechanical	1

- **What is the average salary of professors for each subject**

```
SELECT Subject, ROUND(AVG(Salary),2) AS Avg_Salary
FROM Professors
GROUP BY Subject;
```

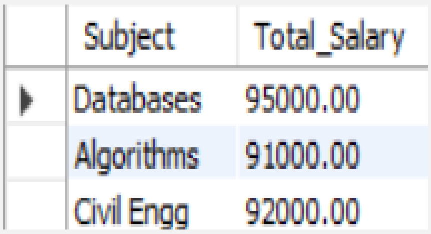


	Subject	Avg_Salary
▶	Databases	95000.00
	Algorithms	91000.00
	Mechanical Engg	88000.00
	Electronics	87000.00
	Civil Engg	92000.00

8. Having Clause

Find Professors earning more than 90k

```
SELECT Subject, SUM(Salary) AS Total_Salary
FROM Professors
GROUP BY Subject
HAVING Total_Salary > 90000;
```

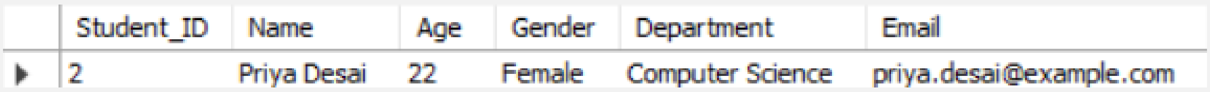


	Subject	Total_Salary
▶	Databases	95000.00
	Algorithms	91000.00
	Civil Engg	92000.00

9. Like Operator :

Students whose Name starts with 'P'

```
SELECT * FROM Students WHERE Name LIKE  'P%' ;
```



	Student_ID	Name	Age	Gender	Department	Email
▶	2	Priya Desai	22	Female	Computer Science	priya.desai@example.com

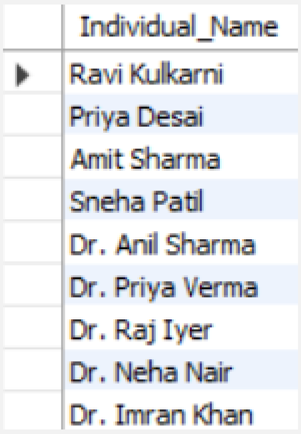
10.Union:

Union of Student Names and Professor Names.

```
SELECT Name AS Individual_Name FROM Students

 UNION

SELECT  Name AS Individual_Name FROM Professors;
```

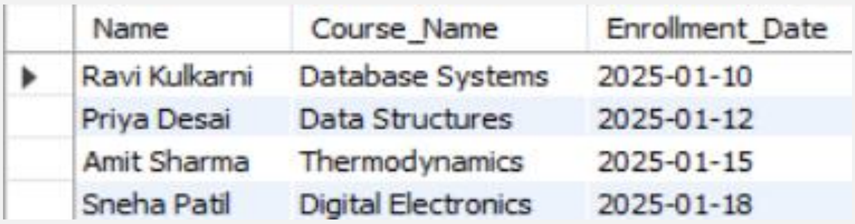


	Individual_Name
▶	Ravi Kulkarni
	Priya Desai
	Amit Sharma
	Sneha Patil
	Dr. Anil Sharma
	Dr. Priya Verma
	Dr. Raj Iyer
	Dr. Neha Nair
	Dr. Imran Khan

11. Joins:

➢ Show student and enrolled course

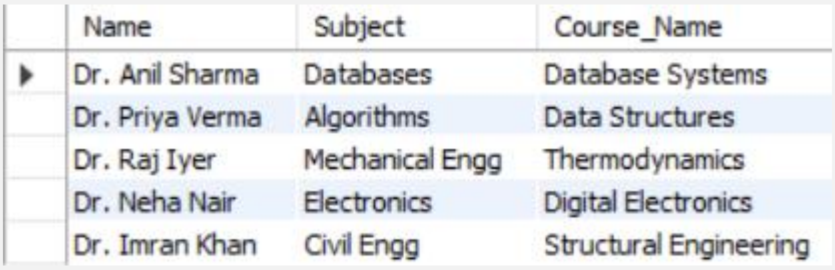
```
SELECT s.Name, c.Course_Name, e.Enrollment_Date
FROM Enrollments e
JOIN Students s ON e.Student_ID = s.Student_ID
JOIN Courses c ON e.Course_ID = c.Course_ID;
```



	Name	Course_Name	Enrollment_Date
▶	Ravi Kulkarni	Database Systems	2025-01-10
	Priya Desai	Data Structures	2025-01-12
	Amit Sharma	Thermodynamics	2025-01-15
	Sneha Patil	Digital Electronics	2025-01-18

➢ Show professor and their courses

```
SELECT p.Name, p.Subject, c.Course_Name
FROM Professors p
JOIN Courses c  ON p.Professor_ID = c.Professor_ID;
```

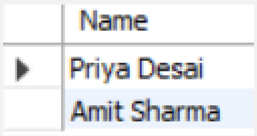


	Name	Subject	Course_Name
▶	Dr. Anil Sharma	Databases	Database Systems
	Dr. Priya Verma	Algorithms	Data Structures
	Dr. Raj Iyer	Mechanical Engg	Thermodynamics
	Dr. Neha Nair	Electronics	Digital Electronics
	Dr. Imran Khan	Civil Engg	Structural Engineering

12. Subqueries:

a) Single row Query:

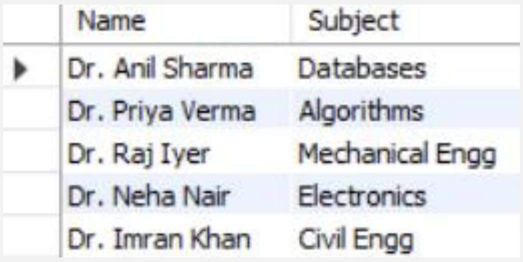
```
SELECT Name FROM Students WHERE Age > (SELECT AVG(Age)
FROM Students);
```



Name
Priya Desai
Amit Sharma

b) Multiple row:

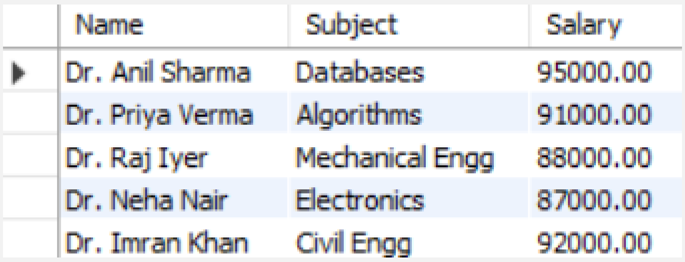
```
SELECT Name, Subject  FROM Professors  WHERE Professor_ID
IN ( SELECT DISTINCT Professor_ID FROM Courses);
```



Name	Subject
Dr. Anil Sharma	Databases
Dr. Priya Verma	Algorithms
Dr. Raj Iyer	Mechanical Engg
Dr. Neha Nair	Electronics
Dr. Imran Khan	Civil Engg

c) Multiple column: Highest-paid professor per subject

```
SELECT Name, Subject, Salary FROM Professors
 WHERE (Subject, Salary) IN ( SELECT Subject, MAX(Salary)
 FROM Professors
GROUP BY Subject);
```



Name	Subject	Salary
Dr. Anil Sharma	Databases	95000.00
Dr. Priya Verma	Algorithms	91000.00
Dr. Raj Iyer	Mechanical Engg	88000.00
Dr. Neha Nair	Electronics	87000.00
Dr. Imran Khan	Civil Engg	92000.00

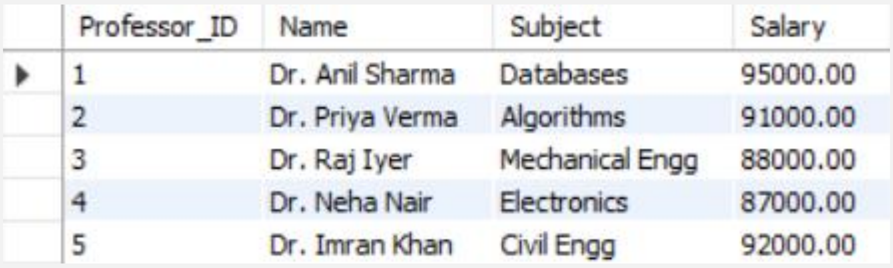
13. VIEW:

```
CREATE VIEW Professors_View AS

SELECT Professor_ID, Name, Subject, Salary

FROM Professors;

SELECT * FROM Professors_View;
```



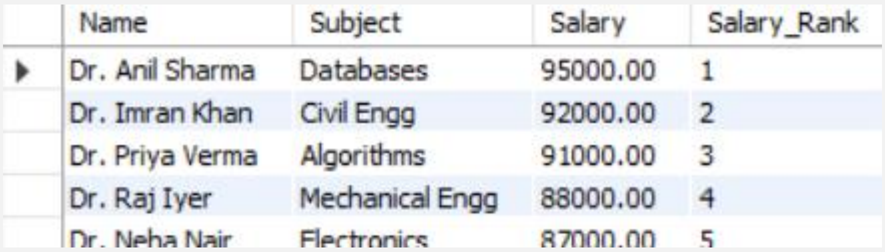
Professor_ID	Name	Subject	Salary
1	Dr. Anil Sharma	Databases	95000.00
2	Dr. Priya Verma	Algorithms	91000.00
3	Dr. Raj Iyer	Mechanical Engg	88000.00
4	Dr. Neha Nair	Electronics	87000.00
5	Dr. Imran Khan	Civil Engg	92000.00

14. Window Function

➢ RANK():

Ranking professors by salary

```
SELECT Name, Subject, Salary,
RANK() OVER (ORDER BY Salary DESC) AS Salary_Rank
FROM Professors;
```

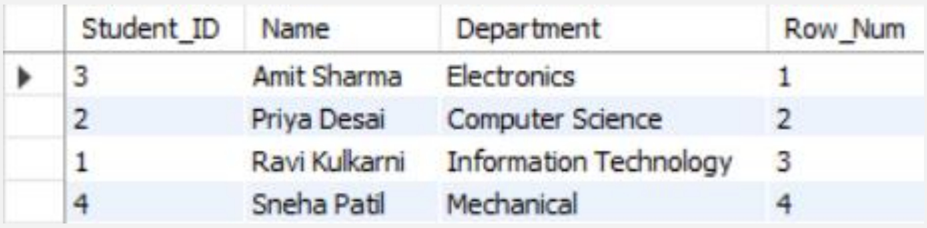


Name	Subject	Salary	Salary_Rank
Dr. Anil Sharma	Databases	95000.00	1
Dr. Imran Khan	Civil Engg	92000.00	2
Dr. Priya Verma	Algorithms	91000.00	3
Dr. Raj Iyer	Mechanical Engg	88000.00	4
Dr. Neha Nair	Electronics	87000.00	5

➢ ROW_NUMBER()

Rank professors by salary, but without gaps in ranking.

```
SELECT Student_ID, Name, Department,
ROW_NUMBER() OVER (ORDER BY Name ASC) AS Row_Num
FROM Students;
```

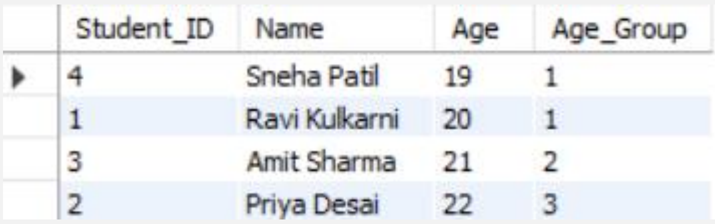


	Student_ID	Name	Department	Row_Num
▶	3	Amit Sharma	Electronics	1
	2	Priya Desai	Computer Science	2
	1	Ravi Kulkarni	Information Technology	3
	4	Sneha Patil	Mechanical	4

➢ **NTILE()**

Divide students into 3 groups based on their age.

```
SELECT Student_ID, Name, Age, NTILE(3) OVER (ORDER BY Age
ASC) AS Age_Group
FROM Students;
```

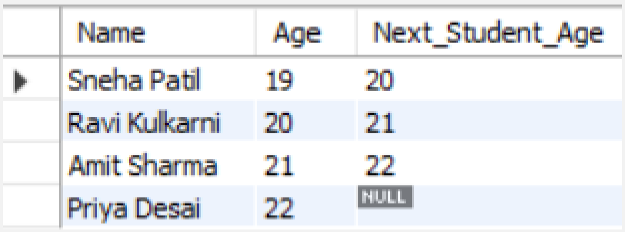


	Student_ID	Name	Age	Age_Group
▶	4	Sneha Patil	19	1
	1	Ravi Kulkarni	20	1
	3	Amit Sharma	21	2
	2	Priya Desai	22	3

➢ **LEAD()**

Compare each student's age with the next student's age.

```
SELECT Name, Age, LEAD(Age, 1) OVER (ORDER BY Age ASC) AS
Next_Student_Age
FROM Students;
```



	Name	Age	Next_Student_Age
▶	Sneha Patil	19	20
	Ravi Kulkarni	20	21
	Amit Sharma	21	22
	Priya Desai	22	NULL