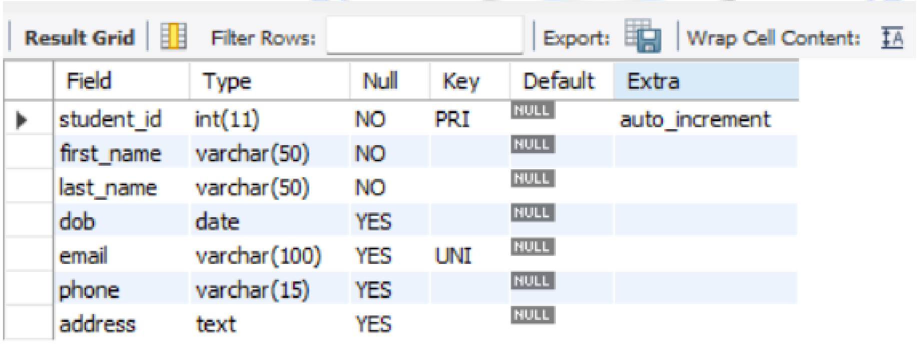
1. Creation of tables

a) Create Students Table

```
CREATE DATABASE StudentManagement;

USE StudentManagement;


-- Creating Students Table

CREATE TABLE Students (

student_id INT PRIMARY KEY AUTO_INCREMENT,

first_nameVARCHAR(50) NOT NULL,

last_nameVARCHAR(50) NOT NULL,

    dob DATE,

    email VARCHAR(100) UNIQUE,

    phone VARCHAR(15),

    address TEXT

);


select * from students;

desc students;
```

Output:

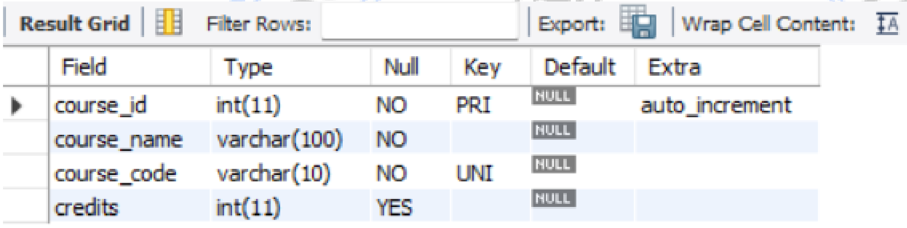


Field	Type	Null	Key	Default	Extra
student_id	int(11)	NO	PRI	NULL	auto_increment
first_name	varchar(50)	NO		NULL	
last_name	varchar(50)	NO		NULL	
dob	date	YES		NULL	
email	varchar(100)	YES	UNI	NULL	
phone	varchar(15)	YES		NULL	
address	text	YES		NULL	

b) Create Course Table

```
-- Creating Courses Table
CREATE TABLE Courses (
course_id INT PRIMARY KEY AUTO_INCREMENT,
course_nameVARCHAR(100) NOT NULL,
course_codeVARCHAR(10) UNIQUE NOT NULL,
    credits INT CHECK (credits BETWEEN 1 AND 6)
);
select * from Courses;
desc Courses;
```

Output:

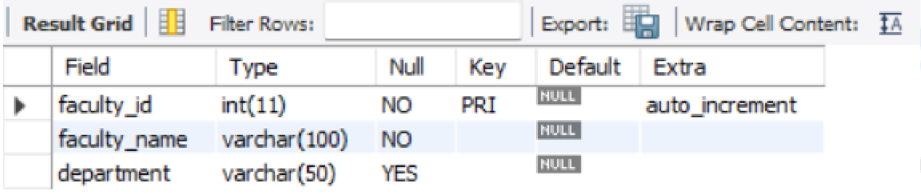


Field	Type	Null	Key	Default	Extra
course_id	int(11)	NO	PRI	NULL	auto_increment
course_name	varchar(100)	NO		NULL	
course_code	varchar(10)	NO	UNI	NULL	
credits	int(11)	YES		NULL	

c) Create Faculty Table

```
-- Creating Faculty Table
CREATE TABLE Faculty (
faculty_id INT PRIMARY KEY AUTO_INCREMENT,
faculty_nameVARCHAR(100) NOT NULL,
    department VARCHAR(50)
);
select * from Faculty;
desc Faculty;
```

Output:

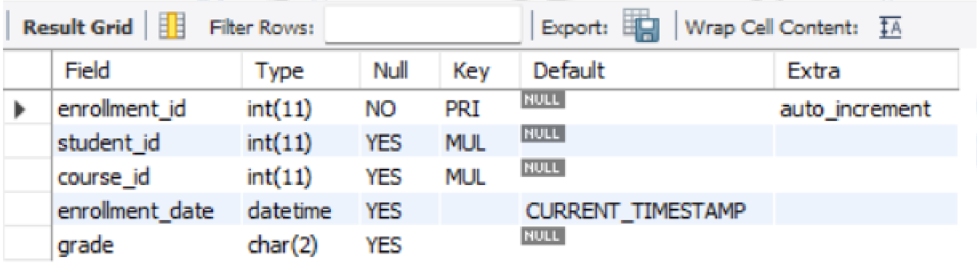


Field	Type	Null	Key	Default	Extra
faculty_id	int(11)	NO	PRI	NULL	auto_increment
faculty_name	varchar(100)	NO		NULL	
department	varchar(50)	YES		NULL	

2) Applying integrity constraints to tables.

```
-- Creating Enrollment Table (Many-to-Many Relationship)
CREATE TABLE Enrollment (
enrollment_id INT PRIMARY KEY AUTO_INCREMENT,
student_id INT,
course_id INT,
enrollment_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    grade CHAR(2),
    FOREIGN KEY (student_id) REFERENCES Students(student_id) ON DELETE CASCADE,
    FOREIGN KEY (course_id) REFERENCES Courses(course_id) ON DELETE CASCADE
);
select * from Enrollment;
descEnrollment;
```

Output:



Field	Type	Null	Key	Default	Extra
enrollment_id	int(11)	NO	PRI	NULL	auto_increment
student_id	int(11)	YES	MUL	NULL	
course_id	int(11)	YES	MUL	NULL	
enrollment_date	datetime	YES		CURRENT_TIMESTAMP	
grade	char(2)	YES		NULL	

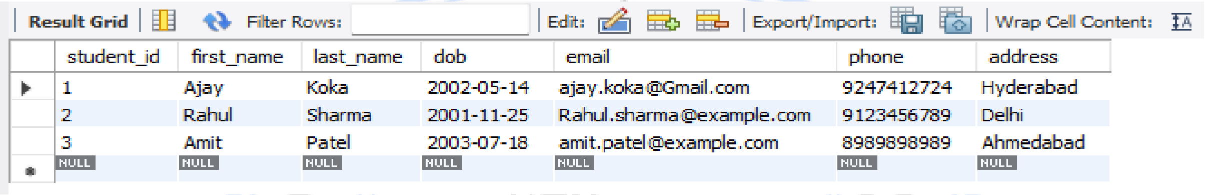
3) Application of INSERT, DELETE & UPDATE commands.

INSERT Command

Adding Data into the students Table

```
INSERT INTO students (first_name, last_name, dob, email, phone, address)

VALUES

('Ajay', 'Koka', '2002-05-14', 'ajay,koka@Gmail.com', '9247412724', 'Hyderabad'),

('Rahul', 'Sharma', '2001-11-25', 'Rahul.sharma@example.com', '9123456789', 'Delhi'),

('Amit', 'Patel', '2003-07-18', 'amit.patel@example.com', '8989898989', 'Ahmedabad');
```

Output:



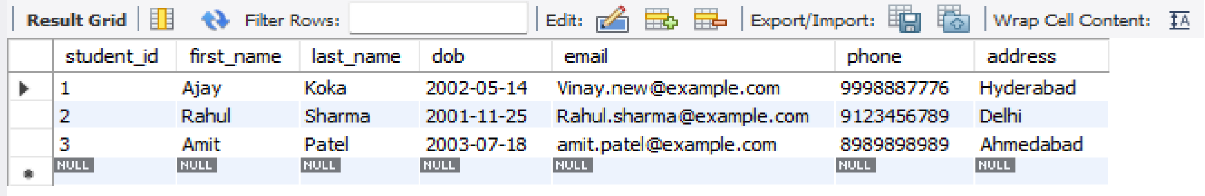
student_id	first_name	last_name	dob	email	phone	address
1	Ajay	Koka	2002-05-14	ajay.koka@Gmail.com	9247412724	Hyderabad
2	Rahul	Sharma	2001-11-25	Rahul.sharma@example.com	9123456789	Delhi
3	Amit	Patel	2003-07-18	amit.patel@example.com	8989898989	Ahmedabad
NULL	NULL	NULL	NULL	NULL	NULL	NULL

UPDATE Command

Modifying a Student's Email and Phone Number

```
UPDATE students

SET email = 'Vinay.new@example.com', phone = '9998887776'

WHERE student_id = 1;
```

Output:



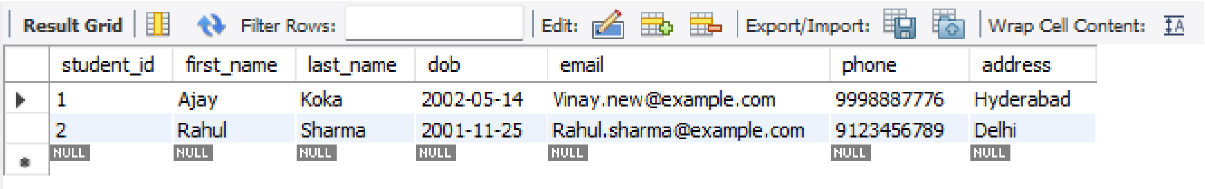
student_id	first_name	last_name	dob	email	phone	address
1	Ajay	Koka	2002-05-14	Vinay.new@example.com	9998887776	Hyderabad
2	Rahul	Sharma	2001-11-25	Rahul.sharma@example.com	9123456789	Delhi
3	Amit	Patel	2003-07-18	amit.patel@example.com	8989898989	Ahmedabad
NULL	NULL	NULL	NULL	NULL	NULL	NULL

DELETE Command

Removing a Student Record

```
DELETE FROM students

WHERE student_id = 3;
```

Output:

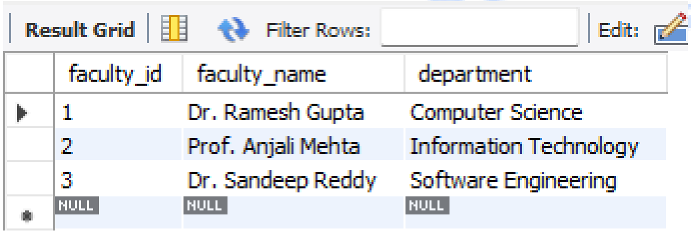


student_id	first_name	last_name	dob	email	phone	address
1	Ajay	Koka	2002-05-14	Vinay.new@example.com	9998887776	Hyderabad
2	Rahul	Sharma	2001-11-25	Rahul.sharma@example.com	9123456789	Delhi
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Inserting into Faculty Table

```
INSERT INTO Faculty (faculty_name, department) VALUES

('Dr. Ramesh Gupta', 'Computer Science'),

('Prof. Anjali Mehta', 'Information Technology'),

('Dr. Sandeep Reddy', 'Software Engineering');

select * from Faculty;
```

Output:



faculty_id	faculty_name	department
1	Dr. Ramesh Gupta	Computer Science
2	Prof. Anjali Mehta	Information Technology
3	Dr. Sandeep Reddy	Software Engineering
NULL	NULL	NULL

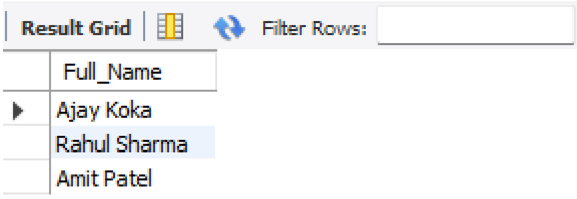
4. Applying built-in functions.

a) String Functions

Concatenation of First and Last Name

SELECT CONCAT(first_name, ' ', last_name) AS Full_Name FROM Students;

Output:

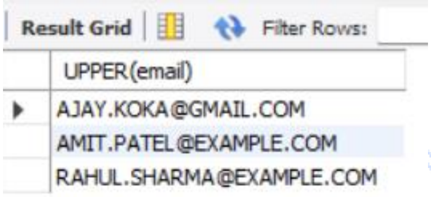


Full_Name
Ajay Koka
Rahul Sharma
Amit Patel

Convert Email to Uppercase

SELECT UPPER(email) FROM Students;

Output:

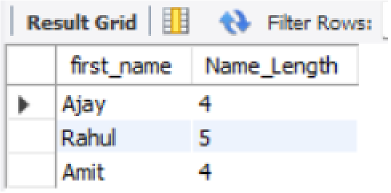


UPPER(email)
AJAY.KOKA@GMAIL.COM
AMIT.PATEL@EXAMPLE.COM
RAHUL.SHARMA@EXAMPLE.COM

Length of Student's Name

SELECT first_name, LENGTH(first_name) AS Name_Length FROM Students;

Output:



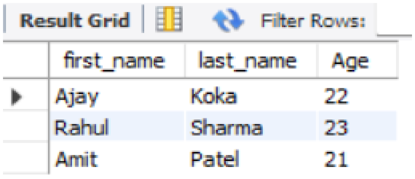
first_name	Name_Length
Ajay	4
Rahul	5
Amit	4

b) Date Functions

Finding Age of Students

SELECT first_name, last_name, TIMESTAMPDIFF(YEAR, dob, CURDATE()) AS Age FROM Students;

Output:



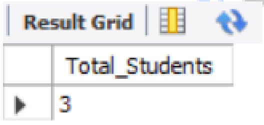
first_name	last_name	Age
Ajay	Koka	22
Rahul	Sharma	23
Amit	Patel	21

c) Aggregate Functions

Count the Total Number of Students

SELECT COUNT(*) AS Total_Students FROM Students;

Output:

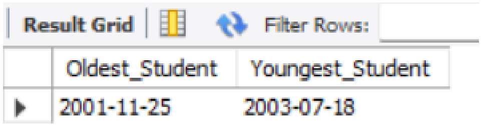


Total_Students
3

Find the Earliest and Latest Date of Birth

SELECT MIN(dob) AS Oldest_Student, MAX(dob) AS Youngest_Student FROM Students;

Output:



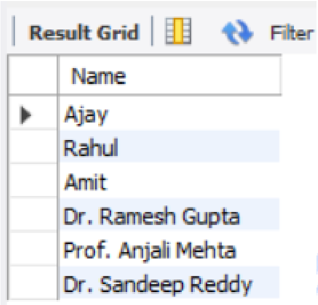
Oldest_Student	Youngest_Student
2001-11-25	2003-07-18

5. Queries Using Set Operators

1. UNION: Combining Students and Faculty Names

```
SELECT first_name AS Name FROM Students

UNION

SELECT faculty_name FROM Faculty;
```

Output:

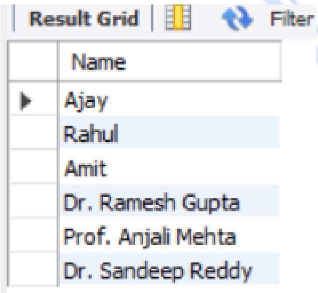


Name
Ajay
Rahul
Amit
Dr. Ramesh Gupta
Prof. Anjali Mehta
Dr. Sandeep Reddy

2. UNION ALL: Keeping Duplicate Names

```
SELECT first_name AS Name FROM Students

UNION ALL

SELECT faculty_name FROM Faculty;
```

Output:



Name
Ajay
Rahul
Amit
Dr. Ramesh Gupta
Prof. Anjali Mehta
Dr. Sandeep Reddy

6. Queries using various types of joins.

INNER JOIN to find common names between Students and Faculty.

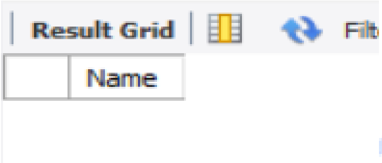
SELECT first_name AS Name

FROM Students

INNER JOIN Faculty

ON Students.first_name = Faculty.faculty_name;

Output:



Name

LEFT JOIN to get all Students and their Courses, Even If Not Enrolled.

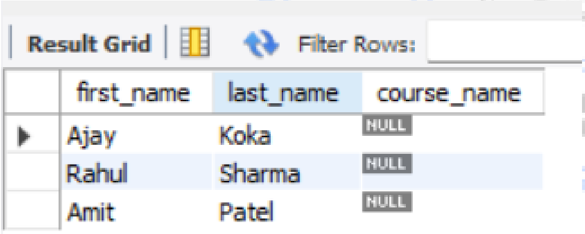
SELECT Students.first_name, Students.last_name, Courses.course_name

FROM Students

LEFT JOIN Enrollment ON Students.student_id = Enrollment.student_id

LEFT JOIN Courses ON Enrollment.course_id = Courses.course_id;

Output:



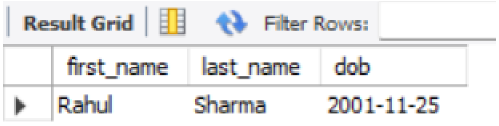
first_name	last_name	course_name
Ajay	Koka	NULL
Rahul	Sharma	NULL
Amit	Patel	NULL

7. Selecting data using subqueries.

Find the Oldest Student (Using MIN in a Subquery)

```
SELECT first_name, last_name, dob

FROM Students

WHERE dob = (SELECT MIN(dob) FROM Students);
```

Output:

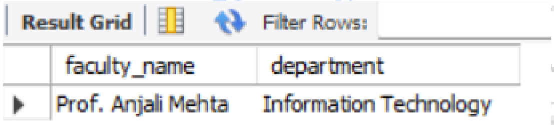


first_name	last_name	dob
Rahul	Sharma	2001-11-25

Find Faculty Members Belonging to the Largest Department (Using MAX)

```
SELECT faculty_name, department

FROM Faculty

WHERE department = (SELECT department FROM Faculty GROUP BY department
ORDER BY COUNT(*) DESC LIMIT 1);
```

Output:



faculty_name	department
Prof. Anjali Mehta	Information Technology

8. Problems related to database management.

1. Handling Orphan Records in the Enrollment Table

Problem:
If a student is deleted from the Students table, their enrollments in the Enrollment table become orphaned (pointing to a non-existent student).

Solution:
You already have ON DELETE CASCADE in the Enrollment table, which automatically deletes enrollments when a student is removed.

To verify this behavior:

```
DELETE FROM Students WHERE student_id = 2;
```

```
SELECT * FROM Enrollment; -- Check if related records are deleted
```

2. Preventing Duplicate Student Entries

Problem:
The **Students** table allows NULL values in the **email** column. However, we want to ensure that every student must have an email address.

How can we modify the table to prevent NULL values in the email column?

Solution:
We can use the ALTER TABLE statement to modify the email column and set it as NOT NULL, ensuring that every student has a valid email.

```
ALTER TABLE Students MODIFY email VARCHAR(100) NOT NULL;
```

3. Handling Null Values in Student Contact Information

Problem:
Some students might not provide an email or phone number, leading to incomplete data.

Solution:
Use NOT NULL constraints and provide default values if needed.

```
ALTER TABLE Students MODIFY email VARCHAR(100) NOT NULL;
```

```
ALTER TABLE Students MODIFY phone VARCHAR(15) NOT NULL DEFAULT 'Not Provided';
```

4. Finding Students Without Enrollment (Data Consistency)

Problem:
Some students might exist in the Students table but have **never enrolled** in any course.

Solution:
Use a LEFT JOIN to find such students.

```
SELECT s.student_id, s.first_name, s.last_name
FROM Students s
LEFT JOIN Enrollment e ON s.student_id = e.student_id
WHERE e.student_id IS NULL;
```

5. Identifying Courses Without Enrollments

Problem:
Some courses might exist in the Courses table but have **no students enrolled**.

Solution:
Use NOT EXISTS to find such courses.

```
SELECT c.course_id, c.course_name
FROM Courses c
WHERE NOT EXISTS (SELECT 1 FROM Enrollment e WHERE e.course_id = c.course_id);
```

6. Normalization Issue – Storing Repetitive Faculty Data

Problem:
If the same faculty name appears multiple times in different departments, it leads to **data redundancy**.

Solution:
Create a Departments table and use department_id as a foreign key in the Faculty table.

```
CREATE TABLE Departments (
   department_id INT PRIMARY KEY AUTO_INCREMENT,
   department_name VARCHAR(100) UNIQUE NOT NULL);
```

```
ALTER TABLE Faculty ADD COLUMN department_id INT;
ALTER TABLE Faculty ADD CONSTRAINT fk_department FOREIGN KEY
(department_id) REFERENCES Departments(department_id);
```

7. Handling Case Sensitivity in Searches

Problem:
If users search for a student with SELECT * FROM Students WHERE first_name = 'ajay';, it may not return results if names are stored with capitalized letters.

Solution:
Use LOWER() or COLLATE for case-insensitive searches.

```
SELECT * FROM Students WHERE LOWER(first_name) = LOWER('ajay');
```

8. Checking for Duplicate Faculty Names

Problem:
Two faculty members with the **same name but different departments** may cause confusion.

Solution:
Find duplicate names using GROUP BY.

```
SELECT faculty_name, COUNT(*) AS count

FROM Faculty

GROUP BY faculty_name

HAVING COUNT(*) > 1;
```

9. Improving Query Performance Using Indexes

Problem:
If the database grows, queries may slow down due to full table scans.

Solution:
Add indexes on frequently searched columns.

```
CREATE INDEX idx_student_name ON Students(first_name, last_name);
```

```
CREATE INDEX idx_course_code ON Courses(course_code);
```

10. Backing Up the Database Regularly

Problem:
Data loss due to accidental deletions or system failures.

Solution:
Use MySQL’s backup feature (mysqldump) to export data.

```
mysqldump -u root -p StudentManagement > backup.sql
```