

# CS 432: Databases

## Assignment 2: DEVELOPING THE DBMS

Group Name: Bro Code

Project Name: Neev Management System

Here are the name & Roll no. of the Group members:

Dhru Jain Khilosiya 23120018

Diya Parashar 23120020

Himani Trivedi 23120022

Om Ambekar 23120028

Priya Darji 2312003

Responsibility of G1:

---

Question 1

---

1. Populate the tables that you created in the previous assignment with random data with the following constraints. All tables must follow the ACID properties and the previous constraints mentioned in Assignment 1

- = Following our ER Diagram we have created 6 Entities, namely:
  1. Students
  2. NEEV members
  3. Instructors
  4. Donor
  5. Volunteer
  6. Admin
  7. Course

**1. Populating the Student table**

```
>> use test;  
>> select * from students;
```

**Attributes used:** ( S\_ID, Name, email, Identification ID, Gender, Photo, family Background, address, M\_ID(f))

Note: Our populated table Student satisfies the ACID properties.

```

CREATE TABLE IF NOT EXISTS students (
    S_ID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Identification_ID VARCHAR(10),
    Gender ENUM('Male', 'Female'),
    Photo VARCHAR(255),
    Family_Background VARCHAR(255),
    Address VARCHAR(255),
    M_ID INT,
    FOREIGN KEY (M_ID) REFERENCES Members(M_ID)
);

```

```

INSERT INTO Persons (S_ID, Name, Email, Identification_ID, Gender, Photo, Family_Background, Address, M_ID) VALUES
(0001, 'Johny Pal', 'john@example.com', '1234567890', 'Male', 'photo.jpg', 'Upper class', '123 Main St, City', 1),
(0002, 'Emma Brown', 'emma@example.com', '0987654321', 'Female', 'pic.jpg', 'Middle class', '456 Elm St, Town', 2),
(0003, 'Michael Lee', 'michael@example.com', '4567890123', 'Male', 'image.jpg', 'Lower class', '789 Oak St, Village', 3),
(0004, 'Sarah Johnson', 'sarah@example.com', '9876543210', 'Female', 'img.jpg', 'Upper middle class', '321 Pine St, City', 4),
(0005, 'David Wilson', 'david@example.com', '1357902468', 'Male', 'photo2.jpg', 'Lower middle class', '654 Cedar St, Town', 5),
(0006, 'Emily Davis', 'emily@example.com', '2468013579', 'Female', 'pic2.jpg', 'Upper class', '987 Maple St, City', 1),
(0007, 'Daniel Martinez', 'daniel@example.com', '5791357024', 'Male', 'image2.jpg', 'Lower class', '258 Oak St, Village', 1),
(0008, 'Olivia Taylor', 'olivia@example.com', '7024579135', 'Female', 'img2.jpg', 'Middle class', '753 Elm St, Town', 3),
(0009, 'James Brown', 'james@example.com', '0147852369', 'Male', 'photo3.jpg', 'Lower middle class', '159 Cedar St, City', 3),
(0010, 'Sophia Garcia', 'sophia@example.com', '3691478520', 'Female', 'pic3.jpg', 'Upper middle class', '852 Pine St, Town', 3);

```

## 1. Students table

S_ID	Name	Email	Identification ID	Gender	Photo	Family Background	Address	M_ID (f)
0001	Johny Pal	john@example.com	1234567890	Male	photo.jpg	Upper class	123 Main St, City	1
0002	Emma Brown	emma@example.com	0987654321	Female	pic.jpg	Middle class	456 Elm St, Town	2
0003	Michael Lee	michael@example.com	4567890123	Male	image.jpg	Lower class	789 Oak St, Village	3
0004	Sarah Johnson	sarah@example.com	9876543210	Female	img.jpg	Upper middle class	321 Pine St, City	4
0005	David Wilson	david@example.com	1357902468	Male	photo2.jpg	Lower middle class	654 Cedar St, Town	5
0006	Emily Davis	emily@example.com	2468013579	Female	pic2.jpg	Upper class	987 Maple St, City	1
0007	Daniel Martinez	daniel@example.com	5791357024	Male	image2.jpg	Lower class	258 Oak St, Village	1

S_ID	Name	Email	Identification ID	Gender	Photo	Family Background	Address	M_ID (f)
0008	Olivia Taylor	olivia@example.com	7024579135	Female	img2.jpg	Middle class	753 Elm St, Town	3
0009	James Brown	james@example.com	0147852369	Male	photo3.jpg	Lower middle class	159 Cedar St, City	3
0010	Sophia Garcia	sophia@example.com	3691478520	Female	pic3.jpg	Upper middle class	852 Pine St, Town	3

	S_ID	Name	Email	Identification_ID	Gender	Photo	Family_Background	Address	M_ID
▶	1	Johny Pal	john@example.com	1234567890	Male	photo.jpg	Upper class	123 Main St, City	1
	2	Emma Brown	emma@example.com	987654321	Female	pic.jpg	Middle class	456 Elm St, Town	2
	3	Michael Lee	michael@example.com	4567890123	Male	image.jpg	Lower class	789 Oak St, Village	3
	4	Sarah Johnson	sarah@example.com	9876543210	Female	img.jpg	Upper middle class	321 Pine St, City	4
	5	David Wilson	david@example.com	1357902468	Male	photo2.jpg	Lower middle class	654 Cedar St, Town	5
	6	Emily Davis	emily@example.com	2468013579	Female	pic2.jpg	Upper class	987 Maple St, City	1
	7	Daniel Martinez	daniel@example.com	5791357024	Male	image2.jpg	Lower class	258 Oak St, Village	1
	8	Olivia Taylor	olivia@example.com	7024579135	Female	img2.jpg	Middle class	753 Elm St, Town	3
	9	James Brown	james@example.com	147852369	Male	photo3.jpg	Lower middle class	159 Cedar St, City	3
*	10	Sophia Garcia	sophia@example.com	3691478520	Female	pic3.jpg	Upper middle class	852 Pine St, Town	3
*	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

## 2. Populating the NEEV member's table

```
>> use test;
>> select * from NEEV members;
```

**Attributes used:**(M\_ID, name, email, Identification ID, gender, designation, photo, password, phone no., AD\_ID(f))

Note: Our populated table Student satisfies the ACID properties

```
CREATE TABLE IF NOT EXISTS NEEV_Members (
    M_ID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Identification_ID VARCHAR(10),
    Gender ENUM('Male', 'Female'),
    Designation VARCHAR(255),
    Photo VARCHAR(255),
    Password VARCHAR(255),
    Phone_No VARCHAR(20)
);
```

```

INSERT INTO Employees (M_ID, Name, Email, Identification_ID, Gender, Designation, Photo, Password, Phone_No)
VALUES
(1, 'John Smith', 'john@example.com', '1234567890', 'Male', 'Manager', 'link_to_photo1', 'password1', '123-456-7890'),
(2, 'Emily Jones', 'emily@example.com', '0987654321', 'Female', 'Developer', 'link_to_photo2', 'password2', '987-654-3210'),
(3, 'Michael Lee', 'michael@example.com', '1357924680', 'Male', 'Analyst', 'link_to_photo3', 'password3', '456-789-1230'),
(4, 'Sarah Brown', 'sarah@example.com', '2468013579', 'Female', 'Designer', 'link_to_photo4', 'password4', '789-123-4560'),
(5, 'David Kim', 'david@example.com', '3692581470', 'Male', 'Intern', 'link_to_photo5', 'password5', '321-654-9870');

```

## 2. NEEV Members

M_ID	Name	Email	Identification ID	Gender	Designation	Photo	Password	Phone No
1	John Smith	john@example.com	1234567890	Male	Manager	(link to photo)	password1	123-456-7890
2	Emily Jones	emily@example.com	0987654321	Female	Developer	(link to photo)	password2	987-654-3210
3	Michael Lee	michael@example.com	1357924680	Male	Analyst	(link to photo)	password3	456-789-1230
4	Sarah Brown	sarah@example.com	2468013579	Female	Designer	(link to photo)	password4	789-123-4560
5	David Kim	david@example.com	3692581470	Male	Intern	(link to photo)	password5	321-654-9870

	M_ID	Name	Email	Identification_ID	Gender	Designation	Photo	Password	Phone_No
▶	1	John Smith	john@example.com	1234567890	Male	Manager	link_to_photo	password1	123-456-7890
	2	Emily Jones	emily@example.com	987654321	Female	Developer	link_to_photo	password2	987-654-3210
	3	Michael Lee	michael@example.com	1357924680	Male	Analyst	link_to_photo	password3	456-789-1230
	4	Sarah Brown	sarah@example.com	2468013579	Female	Designer	link_to_photo	password4	789-123-4560
*	5	David Kim	david@example.com	3692581470	Male	Intern	link_to_photo	password5	321-654-9870
	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

## 3. Populating the Instructors table

```

>> use test;
>> select * from instructors;

```

**Attributes used:** (I\_ID, name, email, phone no., gender, Identification ID, type, S\_ID(f), M\_ID(f))

Note: Our populated table Student satisfies the ACID properties.

```

    ↳ CREATE TABLE IF NOT EXISTS Instructors(
        I_ID INT NOT NULL,
        Name VARCHAR(255) NOT NULL,
        Email VARCHAR(255) NOT NULL,
        Phone_No VARCHAR(12) NOT NULL,
        Gender ENUM('Male', 'Female') NOT NULL,
        Identification_ID VARCHAR(9) NOT NULL,
        Type ENUM('Full', 'Part') NOT NULL,
        S_ID VARCHAR(4) NOT NULL,
        M_ID INT NOT NULL,
        PRIMARY KEY (I_ID)
    );

```

```

INSERT INTO Instructors (I_ID, Name, Email, Phone_No, Gender, Identification_ID, Type, S_ID, M_ID) VALUES
(101, 'John Doe', 'johndoe@example.com', '123-456-789', 'Male', '123456789', 'Full', '0001', 1),
(102, 'Jane Smith', 'janeshmith@example.com', '987-654-321', 'Female', '987654321', 'Part', '0007', 1),
(103, 'Alex Lee', 'alexlee@example.com', '456-789-012', 'Male', '456789012', 'Full', '0002', 1),
(104, 'Emily Chen', 'emilychen@example.com', '741-852-963', 'Female', '741852963', 'Full', '0003', 2),
(105, 'Mark Kim', 'markkim@example.com', '369-258-147', 'Male', '369258147', 'Part', '0010', 1);

```

### 3. Instructors table

I_ID	Name	Email	Phone No.	Gender	Identification ID	Type	S_ID (f)	M_ID (f)
101	John Doe	johndoe@example.com	123-456-789	Male	123456789	Full	0001	1
102	Jane Smith	janeshmith@example.com	987-654-321	Female	987654321	Part	0007	1
103	Alex Lee	alexlee@example.com	456-789-012	Male	456789012	Full	0002	1
104	Emily Chen	emilychen@example.com	741-852-963	Female	741852963	Full	0003	2
105	Mark Kim	markkim@example.com	369-258-147	Male	369258147	Part	0010	1

	I_ID	Name	Email	Phone_No	Gender	Identification_ID	Type	S_ID	M_ID
▶	101	John Doe	johndoe@example.com	123-456-789	Male	123456789	Full	1	1
	102	Jane Smith	janeshmith@example.com	987-654-321	Female	987654321	Part	7	1
	103	Alex Lee	alexlee@example.com	456-789-012	Male	456789012	Full	2	1
	104	Emily Chen	emilychen@example.com	741-852-963	Female	741852963	Full	3	2
*	105	Mark Kim	markkim@example.com	369-258-147	Male	369258147	Part	10	1
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

### 4. Populating the Donation table

>> use test;

```
>> select * from Donation;
```

**Attributes used:** (D\_ID, donor\_name, amount, details, phone no., email)

Note: Our populated table Student satisfies the ACID properties

- `CREATE TABLE Donation (`  
 `D_ID INT PRIMARY KEY,`  
 `Donor_Name VARCHAR(255),`  
 `Amount DECIMAL(10, 2),`  
 `Details VARCHAR(255),`  
 `Phone_No VARCHAR(15),`  
 `Email VARCHAR(255)`  
`);`
- `INSERT INTO Donation (D_ID, Donor_Name, Amount, Details, Phone_No, Email) VALUES`  
`(11, 'Joey Tribianni', 100, 'Transaction ID', '123-456-789', 'johntribianni@example.com'),`  
`(12, 'Jane Smith', 250, 'Transaction ID', '987-654-321', 'janesmith@example.com'),`  
`(23, 'Alice Lee', 50, 'Transaction ID', '555-123-456', 'alicelee@example.com'),`  
`(54, 'Bob Johnson', 150, 'Transaction ID', '111-222-333', 'bobjohnson@example.com'),`  
`(55, 'Sarah Brown', 75, 'Transaction ID', '444-555-666', 'sarahbrown@example.com');`

#### 4. Donation table

D_ID	Donor Name	Amount	Details	Phone No.	Email
11	Joey Tribianni	100	Transaction ID	123-456-789	johntribianni@example.com
12	Jane Smith	250	Transaction ID	987-654-321	janesmith@example.com
23	Alice Lee	50	Transaction ID	555-123-456	alicelee@example.com
54	Bob Johnson	150	Transaction ID	111-222-333	bobjohnson@example.com
55	Sarah Brown	75	Transaction ID	444-555-666	sarahbrown@example.com

	D_ID	Donor_Name	Amount	Details	Phone_No	Email
▶	11	Joey Tribianni	100	Transaction ID	123-456-789	johntribianni@example.com
	12	Jane Smith	250	Transaction ID	987-654-321	janesmith@example.com
	23	Alice Lee	50	Transaction ID	555-123-456	alicelee@example.com
	54	Bob Johnson	150	Transaction ID	111-222-333	bobjohnson@example.com
*	55	Sarah Brown	75	Transaction ID	444-555-666	sarahbrown@example.com
	NULL	NULL	NULL	NULL	NULL	NULL

#### 5. Populating the Volunteer table

```
>> use test;
>> select * from Volunteer;
```

**Attributes used:** (V\_ID, name, email, Identification ID, gender, photo, phone no., M\_ID(f), address)

Note: Our populated table Student satisfies the ACID properties

- `CREATE TABLE Persons (
 V_ID INT AUTO_INCREMENT PRIMARY KEY,
 Name VARCHAR(255) NOT NULL,
 Email VARCHAR(255) NOT NULL,
 Identification_ID BIGINT NOT NULL,
 Gender ENUM('Male', 'Female') NOT NULL,
 Photo VARCHAR(255),
 Phone_No VARCHAR(20) NOT NULL,
 M_ID INT,
 Address VARCHAR(255),
 FOREIGN KEY (M_ID) REFERENCES Courses(C_ID)
);`
- `INSERT INTO Members (Name, Email, Identification_ID, Gender, Photo, Phone_No, V_ID, Address) VALUES
('Steve Smith', 'steve@example.com', '123456789', 'Male', '[Link]', '123-456-789', 1, '123 Main St, City'),
('Emily Johnson', 'emily@example.com', '987654321', 'Female', '[Link]', '987-654-321', 2, '456 Oak Ave, Town'),
('Michael Brown', 'michael@example.com', '567890123', 'Male', '[Link]', '567-890-123', 1, '789 Elm St, Village'),
('Sarah Davis', 'sarah@example.com', '456789012', 'Female', '[Link]', '456-789-012', 4, '321 Pine St, County'),
('David Wilson', 'david@example.com', '234567890', 'Male', '[Link]', '234-567-890', 1, '654 Maple Dr, Township');`

## 5. Volunteer table

V_ID	Name	Email	Identification ID	Gender	Photo	Phone No.	M_ID (f)	Address
1	steve Smith	steve@example.com	123456789	Male	[Link]	123-456-789	1	123 Main St, City
2	Emily Johnson	emily@example.com	987654321	Female	[Link]	987-654-321	2	456 Oak Ave, Town
3	Michael Brown	michael@example.com	567890123	Male	[Link]	567-890-123	1	789 Elm St, Village
4	Sarah Davis	sarah@example.com	456789012	Female	[Link]	456-789-012	4	321 Pine St, County
5	David Wilson	david@example.com	234567890	Male	[Link]	234-567-890	1	654 Maple Dr, Township

	V_ID	Name	Email	Identification_ID	Gender	Photo	Phone_No	M_ID	Address
▶	1	Steve Smith	steve@example.com	123456789	Male	[Link]	123-456-789	1	123 Main St, City
2	Emily Johnson	emily@example.com	987654321	Female	[Link]	987-654-321	2	456 Oak Ave, Town	
3	Michael Brown	michael@example.com	567890123	Male	[Link]	567-890-123	1	789 Elm St, Village	
4	Sarah Davis	sarah@example.com	456789012	Female	[Link]	456-789-012	4	321 Pine St, County	
5	David Wilson	david@example.com	234567890	Male	[Link]	234-567-890	1	654 Maple Dr, Township	
*	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

## 6. Populating the Admin table

```
>> use test;
>> select * from Admin;
```

**Attributes used:** (AD\_ID, email, name, password)

Note: Our populated table Student satisfies the ACID properties

```
▶ Ⓜ CREATE TABLE Admin (
    AD_ID INT AUTO_INCREMENT PRIMARY KEY,
    Email VARCHAR(255) NOT NULL,
    Name VARCHAR(255) NOT NULL,
    Password VARCHAR(255) NOT NULL
);
•   INSERT INTO Admin (Email, Name, Password)
    VALUES ('admin1@example.com', 'John Doe', 'johndoe123');
```

## 6. Admin Table

AD_ID	Email	Name	Password
1	admin1@example.com	John Doe	johndoe123

	AD_ID	Email	Name	Password
▶	1	admin1@example.com	John Doe	johndoe123
*	HULL	HULL	HULL	HULL

## 7. Populating the course table

```
>> use test;
>> select * from Course;
```

**Attributes used:** (C\_ID, Course\_name, Details, Venue)

Note: Our populated table Student satisfies the ACID properties

- `CREATE TABLE Course (`  
    `C_ID INT PRIMARY KEY,`  
    `Course_name VARCHAR(255),`  
    `Details VARCHAR(255),`  
    `Venue VARCHAR(255)`  
`);`
- `INSERT INTO Course (C_ID, Course_name, Details, Venue) VALUES`  
    `(101, 'Mathematics', 'Introduction to Algebra', 'Room 201'),`  
    `(102, 'Literature', 'Shakespearean Plays Analysis', 'Auditorium'),`  
    `(103, 'History', 'World War II Overview', 'Room 103'),`  
    `(104, 'Computer Science', 'Python Programming Basics', 'Lab 301'),`  
    `(105, 'Biology', 'Cellular Biology', 'Room 202');`

## 7. Course table

C_ID	Course_name	Details	Venue
101	Mathematics	Introduction to Algebra	Room 201
102	Literature	Shakespearean Plays Analysis	Auditorium
103	History	World War II Overview	Room 103
104	Computer Science	Python Programming Basics	Lab 301
105	Biology	Cellular Biology	Room 202

	C_ID	Course_name	Details	Venue
▶	101	Mathematics	Introduction to Algebra	Room 201
	102	Literature	Shakespearean Plays Analysis	Auditorium
	103	History	World War II Overview	Room 103
	104	Computer Science	Python Programming Basics	Lab 301
*	105	Biology	Cellular Biology	Room 202
	NULL	NULL	NULL	NULL

## Question 2

---

Please explain and implement the indexing over one of the columns (where the search needs to be optimized), user-defined data types, and table extensions.

### ● Indexing

#### 1. Admin:

An index was created named admin\_info. It was indexed over one column of the Admin table. The below are the edited query code of question G1Q1 (same applied to all the tables) action output to the following code. The optimization may happen by creating the index by fraction seconds.

```
1 •   select * from admin;
2
3 •   alter table admin add index admin_info(Name);
```

66	17:01:30	select * from admin LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
67	17:06:33	alter table admin add index admin_info(Name)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.078 sec

#### 2. Course:

An index was created named course\_info. It was indexed over two columns of the Course table in which C\_ID is primary key.

```
5 •   select * from course;
6
7 •   alter table course add index course_info(C_ID,Course_name);
```

68	17:11:57	select * from course LIMIT 0, 1000	5 row(s) returned	
69	17:13:52	alter table course add index course_info(C_ID,Course_name)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	

#### 3. Donation:

An index was created named donation\_info. It was indexed over two columns of the Course table in which D\_ID is primary key.

```
9 •   select * from donation;
10
11 •   alter table donation add index donation_info(D_ID,Donor_Name);
```

70	17:17:39	select * from donation LIMIT 0, 1000	5 row(s) returned	
71	17:19:36	alter table donation add index donation_info(D_ID,Donor_Name)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	

#### 4. Instructors:

An index was created named instructors\_info. It was indexed over two columns of the Instructor table in which I\_ID is primary key.

```
13 •   select * from instructors;
14
15 •   alter table instructors add index instructors_info(I_ID,Name);
```

```
72 17:21:19 select *from instructors LIMIT 0, 1000          0 row(s) returned
73 17:23:26 alter table instructors add index instructors_info(I_ID,Name) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
```

#### 5. neev\_members:

An index was created named neev\_members\_info. It was indexed over two columns of the neev\_members table in which M\_ID is primary key.

```
17 •   select * from neev_members;
18
19 •   alter table neev_members add index neev_members_info(M_ID,Name);
20
```

```
74 17:25:10 select *from neev_members LIMIT 0, 1000          5 row(s) returned
75 17:25:26 alter table neev_members add index neev_members_info(M_ID,Name) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
```

#### 6. Students:

An index was created named students\_info. It was indexed over two columns of the students table in which S\_ID is a primary key.

```
21 •   select * from students;
22
23 •   alter table students add index students_info(S_ID,Name);
24
```

```
76 17:26:49 select *from students LIMIT 0, 1000          10 row(s) returned
77 17:27:46 alter table students add index students_info(S_ID,Name) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
```

#### 7. Volunteer :

An index was created named volunteer\_info. It was indexed over two columns of the volunteer table in which V\_ID is primary key.

```
25 •   select * from volunteer;
26
27 •   alter table volunteer add index volunteer_info(V_ID,Name);
```

#### • User-defined data types:

```
78 17:29:00 select *from volunteer LIMIT 0, 1000          5 row(s) returned
79 17:29:30 alter table volunteer add index volunteer_info(V_ID,Name) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
```

Here we implemented user-defined data type phone\_no which has 2 fields , phone no and country code and p\_id which stores the id of the person whose phone no is to stored.

The MySQL query for this:

```
24
25      #user defined data type phone_no which stores the code,nd number
26 •  create table phone_no(
27      P_ID INT NOT NULL,
28      country_code varchar(3) not null,
29      Phone_No VARCHAR(10) NOT NULL
30  );
31
32 •  insert into phone_no values(101,'079','123456789'),(101,'079','123458889'),(102,'079','123456789');
33 •  select * from phone_no;
34
35
```

The screenshot shows the MySQL Workbench interface. The top menu bar includes Server, Tools, Scripting, Help, and several icons. Below the menu is a toolbar with various icons. The main window title is 'Ques\_1\_DB-2'. The SQL editor contains the provided MySQL script. The results grid at the bottom displays the data inserted into the 'phone\_no' table.

P_ID	country_code	Phone_No
101	124	123456789
101	124	123458889
102	123	123456789

- Table Extension

MySQL has an extension for the CREATE TABLE clause that can be used to generate a new table with the same schema as an existing table in the database. This is typically used to temporarily store the output of complex queries in a new table. The resulting table has identical column names and data types as the original table.

Here we implemented the table extension over volunteer. Our extension creates new table having three attributes from the table volunteer.

Separate table to of volunteers.

```
)  
• CREATE TABLE extendeds_table AS  
L   SELECT V_id, Name, Gender  
:   FROM volunteer;  
)  
• select * from extendeds_table;
```

	V_id	Name	Gender
▶	1	Steve Smith	Male
	2	Emily Johnson	Female
	3	Michael Brown	Male
	4	Sarah Davis	Female
	5	David Wilson	Male

0 45 19:16:42 CREATE TABLE extendeds\_table AS SELECT V\_id, Name, Gender FROM volunteer

5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

## Responsibility of G2:

---

### Question 3

---

To demonstrate the process of creating a user, granting different permissions on tables and views, and revoking the permissions in a database management system. You are supposed to use the database created by the G1.

1. Create a user named "user1" with the password "password1".  $1 \times 1 = 1$  Pts

```
CREATE USER 'user1'@'127.0.0.1' IDENTIFIED BY 'password1';
select * from mysql.user;
```

The screenshot shows the MySQL Workbench interface with a query editor window titled 'Ques\_1\_DB-2'. The SQL pane contains the following code:

```
1 #use test;
2 #show tables;
3
4 • select * from mysql.user;
5 • CREATE USER 'user1'@'127.0.0.1' IDENTIFIED BY 'password1';
6 • select * from mysql.user;
7
```

The Result Grid pane displays the contents of the 'mysql.user' table. The table has columns: Host, User, Password, Select\_priv, Insert\_priv, Update\_priv, Delete\_priv, Create\_priv, Drop\_priv, Reload\_priv, Shutdown\_priv, Process, and Super\_priv. The data shows the creation of a user 'user1' with password 'password1' at host '127.0.0.1'.

Host	User	Password	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv	Process	Super_priv
localhost	root		Y	Y	N	Y	Y	Y	Y	Y	Y	N
%	root		N	N	N	N	N	N	N	N	N	N
127.0.0.1	root		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
::1	root		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	test	*54B3077CC291D368DC0A82C89143DEA6166...	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	Himani	*54B3077CC291D368DC0A82C89143DEA6166...	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
127.0.0.1	user1	"668425423DB5193AF921380129F465A64252..."	N	N	N	N	N	N	N	N	N	N

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	19:16:01	select * from mysql.user LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
2	19:16:01	CREATE USER 'user1'@'127.0.0.1' IDENTIFIED BY 'password1'	0 row(s) affected	0.000 sec
3	19:16:01	select * from mysql.user LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec

2. Create Views on any of the two tables formed by G1 as view1 and view2. And make sure that views contain columns from at least two tables and one additional column with the user-defined data type. 6 x 1 = 6 Pts

```
create view view2 as
select instructors.I_ID, Name as Instructor_Name ,Email,country_code, phone_no.Phone_NO,
C_ID from phone_no join instructors, active_courses where phone_no.P_ID=instructors.I_ID and
active_courses.I_ID=instructors.I_ID;
```

- Phone\_no is user defined data type which stores the phone no of each person with it's country code.

The screenshot shows the MySQL Workbench interface. The SQL editor window displays the following code:

```
#view2
drop view view2;
create view view2 as
select instructors.I_ID, Name as Instructor_Name ,Email,country_code, phone_no.Phone_NO,
C_ID from phone_no join instructors, active_courses where phone_no.P_ID=instructors.I_ID and
active_courses.I_ID=instructors.I_ID;
```

The Result Grid shows the following data:

I_ID	Instructor_Name	Email	country_code	Phone_NO	C_ID
101	John Doe	john.doe@example.com	079	123456789	102
101	John Doe	john.doe@example.com	079	123456889	102
102	Jane Smith	jane.smith@example.com	079	123456789	103

The Action Output window shows the execution log:

#	Time	Action	Message	Duration / Fetch
48	22:12:22	select instructors.I_ID,Name as Instructor_Name ,Email,country_code, phone_no.Phone_NO,C_ID from phone_no join instructors, active_courses where phone_no.P_ID=instructors.I_ID and active_courses.I_ID=instructors.I_ID	3 row(s) returned	0.000 sec / 0.000 sec
49	22:14:11	use test	0 row(s) affected	0.000 sec
50	22:14:11	create view view2 as select instructors.I_ID, Name as Instructor_Name ,Email,country_code, phone_no.Phone_NO, C_ID from phone_no join instructors, active_courses where phone_no.P_ID=instructors.I_ID and active_courses.I_ID=instructors.I_ID	0 row(s) affected	0.016 sec
51	22:15:53	use test	0 row(s) affected	0.000 sec
52	22:15:53	drop view view2	0 row(s) affected	0.000 sec
53	22:15:53	create view view2 as select instructors.I_ID, Name as Instructor_Name ,Email,country_code, phone_no.Phone_NO, C_ID from phone_no join instructors, active_courses where phone_no.P_ID=instructors.I_ID and active_courses.I_ID=instructors.I_ID	0 row(s) affected	0.015 sec
54	22:15:53	select * from view2 LIMIT 0, 1000	3 row(s) returned	0.016 sec / 0.000 sec

- Instructors and active\_courses are 2 tables from which view is created.

```
create view view1 as
select instructors.I_ID, Name as Instructor_Name ,course.Course_name as teaches ,
country_code, phone_no.Phone_NO
from phone_no join instructors, course,active_courses where phone_no.P_ID=instructors.I_ID and active_courses.I_ID=instructors.I_ID
and course.C_ID=active_courses.C_ID;
select * from view1;
```

MySQL Workbench

Local instance MySQL80 - W... db\_ass2 - Warning - not supported X

File Edit View Query Database Server Tools Scripting Help

Navigator: ass\_2\* | Ques\_1\_DB-2\*

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Administration Schemas

Information: No object selected

Result Grid | Filter Rows: Export Wrap Cell Content: Result Grid Read Only Context Help Snippets

```

34
35
36 -- #view1
37 • create view view1 as
38     select instructors.I_ID, Name as Instructor_Name , course.Course_name as teaches ,
39     country_code, phone_no.Phone_NO
40     from phone_no join instructors, course,active_courses where phone_no.P_ID=instructors.I_ID and active_courses.I_ID=instr
41     and course.C_ID=active_courses.C_ID;
42 • select * from view1;
43

```

I_ID	Instructor_Name	teaches	country_code	Phone_NO
101	John Doe	Literature	079	123456789
101	John Doe	Literature	079	123458889
102	Jane Smith	History	079	123456789

view1 38 x

Output: Action Output

#	Time	Action	Message	Duration / Fetch
62	22:23:08	select instructors.I_ID, Name as Instructor_Name ,course.Course_name as teaches ,country_code, phone_no.Phone_NO	3 row(s) returned	0.000 sec / 0.000 sec
63	22:23:24	selected instructors.I_ID, Name as Instructor_Name ,course.Course_name as teaches ,country_code, phone_no.Phone_NO	3 row(s) returned	0.016 sec / 0.000 sec
64	22:23:50	select instructors.I_ID, Name as Instructor_Name ,course.Course_name as teaches ,country_code, phone_no.Phone_NO	3 row(s) returned	0.000 sec / 0.000 sec
65	22:25:37	select * from view1 LIMIT 0, 1000	Error Code: 1146. Table test.view1 doesn't exist	0.000 sec
66	22:25:57	use test	0 row(s) affected	0.000 sec
67	22:25:57	create view view1 as select instructors.I_ID, Name as Instructor_Name ,course.Course_name as teaches ,country_code, phone_no.Phone_NO	0 row(s) affected	0.016 sec
68	22:25:57	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.016 sec / 0.000 sec

Object Info Session

Query Completed

3. Grant "user1" the following permissions on "table1":

- SELECT
- UPDATE
- DELETE

```
9
10
11 • GRANT select,update,delete ON * TO 'user1'@'localhost';
12
```

Output: Action Output

#	Time	Action	Message	Duration / Fetch
25	20:00:32	select * from table1 LIMIT 0, 1000	5 row(s) returned	0.015 sec / 0.000 sec
26	20:12:03	use test	0 row(s) affected	0.000 sec
27	20:12:07	GRANT select,update,delete ON users TO 'user1'@'localhost'	Error Code: 1146. Table 'test.users' doesn't exist	0.000 sec
28	20:12:32	GRANT select,update,delete ON user TO 'user1'@'localhost'	Error Code: 1146. Table 'test.user' doesn't exist	0.000 sec
29	20:13:14	use test	0 row(s) affected	0.000 sec
30	20:13:31	use test	0 row(s) affected	0.000 sec
31	20:13:31	GRANT select,update,delete ON * TO 'user1'@'localhost'	0 row(s) affected	0.015 sec

4. Grant "user1" the following permissions on "view1": 2 x 1 = 2 Pts
- SELECT

```
grant select On view1 To 'user1'@'localhost';
```

98 01:41:17 use test	0 row(s) affected	0.000 sec
99 01:41:17 grant select On view1 To 'user1'@'localhost'	0 row(s) affected	0.016 sec

```
show grants for 'user1'@'localhost';
```

The screenshot shows the MySQL Workbench interface. In the SQL editor, the command `show grants for 'user1'@'localhost';` is run. The results are displayed in a Result Grid:

```

26 • show grants for 'user1'@'localhost';
27
28
29
Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
Grants for user1@localhost
GRANT USAGE ON *.* TO 'user1'@'localhost'
GRANT SELECT, UPDATE, DELETE ON `test`.* ...
▶ GRANT SELECT, UPDATE, DELETE ON `test`.`*` ...
GRANT SELECT, UPDATE, DELETE ON `test`.`*`...
GRANT SELECT ON `test`.`view1` TO 'user1'...

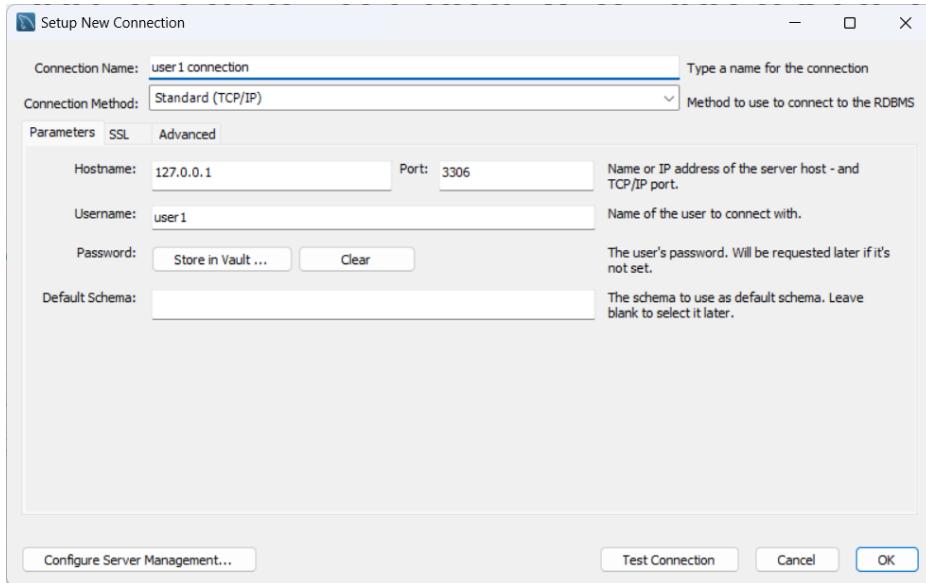
```

Below the grid, the Action Output pane shows the history of actions:

#	Time	Action	Message	Duration / Fetch
104	01:44:09	use test	0 row(s) affected	0.000 sec
105	01:44:09	revoke update on * from user1	Error Code: 1141. There is no such grant defined for user 'user1' on host '%'	0.015 sec
106	01:44:41	use test	0 row(s) affected	0.000 sec
107	01:44:41	revoke update,delete on * from 'user1'@'localhost'	Error Code: 1141. There is no such grant defined for user 'user1' on host 'localhost'	0.000 sec
108	01:45:12	revoke update,delete from 'user1'@'localhost'	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MariaDB s...	0.000 sec
109	01:45:53	use test	0 row(s) affected	0.000 sec
110	01:45:53	show grants for 'user1'@'localhost'	5 row(s) returned	0.000 sec / 0.000 sec

5. Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" and report your findings. 6 x 1 = 6 Pts

- I have taken "table1" as "course" table for this question.
- Connected as user1:



- Select:

```
select * from course;
```

The screenshot shows the MySQL Workbench interface. The results grid displays the following data:

C_ID	Course_name	Details	Venue
101	Mathematics	Introduction to Algebra	Room 201
102	Literature	Shakespearean Plays Analysis	Auditorium
103	History 3	World War II Overview	Room 103
104	Computer Science	Python Programming Basics	Lab 301
105	Biology	Cellular Biology	Room 202
NULL	NULL	NULL	NULL

The SQL history pane shows the following log:

#	Time	Action	Message	Duration / Fetch
6	10:31:49	selected * from view1 LIMIT 0, 1000	Error Code: 1142. SELECT command denied to user 'user1'@'localhost' for table 'view1'	0.000 sec
7	10:32:18	use test	0 row(s) affected	0.000 sec
8	10:32:18	selected * from course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
9	10:32:18	selected * from view1 LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
10	10:34:19	use test	0 row(s) affected	0.016 sec
11	10:34:19	selected * from course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
12	10:34:20	selected * from view1 LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec

```
select * from view1;
```

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: user1\_query

**MANAGEMENT**

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

**INSTANCE**

- Administration Schemas

No object selected

```

1 -- show databases;
2 * use test;
3
4 -- show tables;
5
6 * select * from view1;
7

```

Result Grid | Filter Rows: Export: Wrap Cell Content: □

I_ID	Instructor_Name	teaches	country_code	Phone_No
101	John Doe	Literature	079	123456789
101	John Doe	Literature	079	123458889
102	Jane Smith	History	079	123456789

view1.x

Action Output

#	Time	Action	Message	Duration / Fetch
1	22:31:26	show tables	Error Code: 1046. No database selected Select the default DB to be used by double-clicking its name in the SCH...	0.000 sec
2	22:31:29	use test	0 row(s) affected	0.000 sec
3	22:31:29	show tables	16 row(s) returned	0.000 sec / 0.000 sec
4	22:31:43	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

Query Completed

- Update:

```

update course set Course_name="History II" where c_id=103;
select * from course;

```

Local instance MySQL80 - Wamin.x

File Edit View Query Database Server Tools Scripting Help

Query 1

```

6
7 -- select * from course;
8 -- select * from view1;
9
10
11 * update course set Course_name="History II" where c_id=103;
12 * select * from course;
13

```

Result Grid | Filter Rows: Export/Import: Wrap Cell Content: □

C_ID	Course_name	Details	Venue
101	Mathematics	Introduction to Algebra	Room 201
102	Literature	Shakespearean Plays Analysis	Auditorium
103	History II	World War II Overview	Room 103
104	Computer Science	Python Programming Basics	Lab 301
105	Biology	Cellular Biology	Room 202

course 7

Action Output

#	Time	Action	Message	Duration / Fetch
9	10:32:18	select * from view1 LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
10	10:34:19	use test	0 row(s) affected	0.016 sec
11	10:34:19	select * from course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
12	10:34:20	select * from view1 LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
13	10:38:26	use test	0 row(s) affected	0.000 sec
14	10:38:26	update course set Course_name="History II" where c_id=103	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.015 sec
15	10:38:27	select * from course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

```
update view1 set country_code='123' where I_Id=101;  
select * from view1;
```

- This will update the view only not the base table.

The screenshot shows the MySQL Workbench interface. In the central query editor window, the following SQL code is displayed:

```
1 -- show databases;  
2 * use test;  
3 -- show tables;  
4 * select * from view1;  
5  
6 * update view1 set country_code='123' where I_Id=101;  
7 * select * from view1;  
8
```

Below the code, the results grid displays the data from the view:

I_ID	Instructor_Name	teaches	country_code	Phone_NO
101	John Doe	Literature	123	123456789
101	John Doe	Literature	123	123458889
102	Jane Smith	History	123	123456789

The output pane at the bottom shows the execution log:

Action	Time	Action	Message	Duration / Fetch
36	22:44:09	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
37	22:44:09	update view1 set country_code='123' where I_Id=102	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0	0.016 sec
38	22:44:09	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
39	22:44:31	use test	0 row(s) affected	0.000 sec
40	22:44:31	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.016 sec / 0.000 sec
41	22:44:31	update view1 set country_code='123' where I_Id=101	2 row(s) affected Rows matched: 2 Changed: 2 Warnings: 0	0.000 sec
42	22:44:31	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

- Delete:

```
delete from course where C_ID=105;
```

```
select * from course;
```

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the database is set to 'db\_ass2 - Warning - not supp...' and the connection is 'Local instance MySQL80 - Wamin.x'. The main area has a tab titled 'Query 1' containing the following SQL code:

```
1 -- show databases;
2 • use test;
3
4 -- show tables;
5
6 • delete from course where C_ID=105;
7 • select * from course;
```

Below the code, the 'Result Grid' shows the following data:

C_ID	Course_name	Details	Venue
101	Mathematics	Introduction to Algebra	Room 201
102	Literature	Shakespearean Plays Analysis	Auditorium
103	History	World War II Overview	Room 103
104	Computer Science	Python Programming Basics	Lab 301

The 'course 9' tab shows the execution history:

#	Time	Action	Message	Duration / Fetch
11	19:49:04	select * from courses LIMIT 0, 1000	Error Code: 1146. Table 'test.courses' doesn't exist.	0.000 sec
12	19:49:09	select * from course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
13	19:54:09	use test	0 row(s) affected	0.000 sec
14	19:54:18	select * from course LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
15	19:54:28	use test	0 row(s) affected	0.000 sec
16	19:54:28	delete from course where C_ID=105	1 row(s) affected	0.016 sec
17	19:54:28	select * from course LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec

The status bar at the bottom indicates 'Query Completed'.

```
delete from view1 where I_Id=102;
select * from view1;
```

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the database is set to 'db\_ass2 - Warning - not supp...' and the connection is 'Local instance MySQL80 - Wamin.x'. The main area has a tab titled 'user1\_query' containing the following SQL code:

```
1 -- show databases;
2 • use test;
3 -- show tables;
4 • select * from view1;
5
6 -- update view1 set country_code='123' where I_Id=101;
7
8 • delete from view1 where I_Id=102;
9 • select * from view1;
10
```

Below the code, the 'Result Grid' shows the following data:

I_ID	Instructor_Name	teaches	country_code	Phone_NO
101	John Doe	Literature	123	123456789
101	John Doe	Literature	123	123458889
102	Jane Smith	History	123	123456789

The 'view1\_20' tab shows the execution history:

#	Time	Action	Message	Duration / Fetch
40	22:44:31	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.016 sec / 0.000 sec
41	22:44:31	update view1 set country_code='123' where I_Id=101	2 row(s) affected Rows matched: 2 Changed: 2 Warnings: 0	0.000 sec
42	22:44:31	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
43	22:52:27	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
44	22:52:29	use test	0 row(s) affected	0.000 sec
45	22:52:29	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
46	22:52:29	delete from view1 where I_Id=102	Error Code: 1395. Can not delete from join view 'test.view1'	0.000 sec

The status bar at the bottom indicates 'SQL script saved to 'D:\Himani\IT\DB\SQL\user1\_query.sql''.

6. Revoke the UPDATE and DELETE permissions on "table1" for "user1" and report your findings.  $4 \times 1 = 4$  Pts

- Again connected to root account and revoke the permissions:
- After revoking, user1 is not able to make changes to the table course and view1.

```
revoke update,delete on test.course from 'user1'@'127.0.0.1';
show grants for 'user1'@'127.0.0.1';
```

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: ass\_2 x Ques\_1\_DB-2

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs

Administration Schemas

No object selected

```

10    -- show grants for 'user1'@'127.0.0.1';
11
12 •  revoke update,delete on test.course from 'user1'@'127.0.0.1';
13 •  show grants for 'user1'@'127.0.0.1';
14
15
16    -- drop view view1;

```

Result Grid | Filter Rows: Export: Wrap Cell Content: SQLAdditions: Jump to

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result 6 Result 7 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
10	19:57:11	use test	0 row(s) affected	0.000 sec
11	19:57:11	show tables	7 row(s) returned	0.000 sec / 0.000 sec
12	19:57:11	GRANT select,update,DELETE ON test.course TO 'user1'@'127.0.0.1';	0 row(s) affected	0.000 sec
13	19:57:23	use test	0 row(s) affected	0.000 sec
14	19:57:23	show tables	7 row(s) returned	0.000 sec / 0.000 sec
15	19:57:23	revoke update,delete on test.course from user1@'127.0.0.1'	0 row(s) affected	0.015 sec
16	19:57:23	show grants for 'user1'@'127.0.0.1'	3 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

Query Completed

revoke update,delete on view1 from 'user1'@'127.0.0.1';

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: ass\_2 x Ques\_1\_DB-2\*

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown

Administration Schemas

No object selected

```

52    #grant on view1
53    -- GRANT select,update,DELETE ON test.view1 TO 'user1'@'127.0.0.1';
54
55
56
57 •  revoke update,delete on view1 from 'user1'@'127.0.0.1';
58
59
60
61
62
63    -- drop view view1;
64    -- create view view1 as
65    -- select S_ID,students.M_ID,students.Name from neev_members inner join students on neev_members.M_ID = students.M_ID;
66    -- select * from view1;
67

```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
66	22:25:57	use test	0 row(s) affected	0.000 sec
67	22:25:57	create view view1 as select instructors.I_ID, Name as Instructor_Name ,course.Course_name as teaches ,co ...	3 row(s) affected	0.016 sec
68	22:25:57	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.016 sec / 0.000 sec
69	22:30:57	GRANT select,update,DELETE ON test.view1 TO 'user1'@'127.0.0.1';	0 row(s) affected	0.016 sec
70	22:43:58	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
71	22:45:51	select * from view1 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
72	22:55:23	revoke update,delete on view1 from user1@'127.0.0.1'	0 row(s) affected	0.016 sec

Object Info Session

Query Completed

7. Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" again.  $3 \times 1 = 3$  Pts
- Select:

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: user1\_query

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE Administration Schemas

Information: No object selected

```

1 -- show databases;
2 * use test;
3 -- show tables;
4
5 * select * from course;
6
7
8
9
10
11

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid

C_ID	Course_name	Details	Venue
101	Mathematics	Introduction to Algebra	Room 201
102	Literature	Shakespearean Plays Analysis	Auditorium
103	History	World War II Overview	Room 103
104	Computer Science	Python Programming Basics	Lab 301

course31 x

Output:

#	Time	Action	Message	Duration / Fetch
61	22:59:19	update view1 set country_code='124' where I_Id=101	0 row(s) affected Rows matched: 2 Changed: 0 Warnings: 0	0.000 sec
62	22:59:19	select * from view1 LIMIT 0,1000	3 row(s) returned	0.000 sec / 0.000 sec
63	22:59:20	use test	0 row(s) affected	0.000 sec
64	22:59:22	select * from view1 LIMIT 0,1000	3 row(s) returned	0.000 sec / 0.000 sec
65	22:59:22	update view1 set country_code='124' where I_Id=101	0 row(s) affected Rows matched: 2 Changed: 0 Warnings: 0	0.000 sec
66	22:59:22	select * from view1 LIMIT 0,1000	3 row(s) returned	0.000 sec / 0.000 sec
67	23:01:12	select * from course LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

Query Completed

- update:

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: user1\_query

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE Administration Schemas

Information: No object selected

```

1 -- show databases;
2 * use test;
3 -- show tables;
4
5 * update view1 set country_code='124' where I_Id=101;
6 * select * from view1;
7 * update course set Venue='Room 111' where C_Id=101;
8 * select * from course;
9
10
11
12
13
14
15
16
17
18
19

```

Output:

#	Time	Action	Message	Duration / Fetch
1	23:09:28	use test	0 row(s) affected	0.000 sec
2	23:09:28	show grants for user1'@'127.0.0.1'	Error Code: 1044. Access denied for user 'user1'@'localhost' to database 'mysql'	0.000 sec
3	23:09:40	use test	0 row(s) affected	0.000 sec
4	23:09:40	update view1 set country_code='124' where I_Id=101	Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a ...	0.016 sec

Object Info Session

- Delete:



## Question 2

Mention the situation that violates the referential integrity, show the updates in the table, and how we can solve such problems.

Here in our NEEV system, we have certain referential keys like S\_ID in the student\_course, instructors table, M\_ID in the Volunteer table, C\_ID in the student\_course, active\_courses and Volunteer\_course,I\_ID in the active\_courses, AD\_ID in the funds, V\_ID in the volunteer\_courses.

A database error will be occurred here because there are certain column that is referenced by other tables through referential integrity constraints. As a result, it is not possible to update or delete rows from the tables that are referenced by the other tables, as it would violate the constraints.

So, to get rid of the problem we use the below given script to fetch/update the records without violating the referential integrity problems. The above SQL statements adds a rule to the Student\_course table. This rule ensures that whenever we add a new row to

Student\_course, the value in the S\_ID column must already exist in the Student table's S\_ID column.The ON DELETE CASCADE part means that if we delete a student record from the Student table, all corresponding rows in the Student\_course table will also be deleted.

Similarly, ON UPDATE CASCADE means that if we change a student's ID in the Student table, all related records in the Student\_course table will also have their student IDs updated to match. Overall, this rule keeps things tidy and ensures that every S\_ID in Student\_course points to a valid student in the Student table.

```
-- Student_course
ALTER TABLE `Student_course`
ADD CONSTRAINT `fk_student_course_student`
FOREIGN KEY (`S_ID`)
REFERENCES `Students` (`S_ID`)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Output:

SHOW TABLES	11 row(s) returned
ALTER TABLE 'Student_course' ADD CONSTRAINT 'fk_student_course_student' FOREIGN KEY ('S_ID') R...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0

Here, we will do the same for as the above query by taking every individual schema into account and apply same kind of the query to make sure no violation happens.

```
ALTER TABLE `Student_course`  
ADD CONSTRAINT `fk_student_course_course`  
FOREIGN KEY (`C_ID`)  
REFERENCES `Course` (`C_ID`)  
ON DELETE CASCADE  
ON UPDATE CASCADE;
```

Output:

```
ALTER TABLE 'Student_course' ADD CONSTRAINT 'fk_student_course_course' FOREIGN KEY ('C_ID') RE... 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
```

```
-- Active_courses  
ALTER TABLE `Active_courses`  
ADD CONSTRAINT `fk_active_courses_course`  
FOREIGN KEY (`C_ID`)  
REFERENCES `Course` (`C_ID`)  
ON DELETE CASCADE  
ON UPDATE CASCADE;
```

Output:

```
ALTER TABLE 'Active_courses' ADD CONSTRAINT 'fk_active_courses_course' FOREIGN KEY ('C_ID') RE... 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
```

```
-- Active_courses
ALTER TABLE `Active_courses`
ADD CONSTRAINT `fk_active_courses_instructor`
FOREIGN KEY (`I_ID`)
REFERENCES `Instructors` (`I_ID`)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Output:

```
ALTER TABLE `Active_courses` ADD CONSTRAINT `fk_active_courses_instructor` FOREIGN KEY (`I_ID`) R... 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
```

Here, we can also have instructors table depended on both S\_ID and M\_ID, the below given case is for the case where the foreign key is taken as S\_ID.

```
-- Instructors
ALTER TABLE `Instructors`
ADD CONSTRAINT `fk_instructor_student`
FOREIGN KEY (`S_ID`)
REFERENCES `Students` (`S_ID`)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Output:

#	Time	Action	Message
113	22:22:27	ALTER TABLE `Instructors` ADD CONSTRAINT `fk_instructor_student` FOREIGN KEY (`S_ID`) REFERENCE...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

```
-- Volunteer

ALTER TABLE `Volunteer`
ADD CONSTRAINT `fk_volunteer_member`
FOREIGN KEY (`M_ID`)
REFERENCES `NEEV_Members` (`M_ID`)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Output:

```
ALTER TABLE 'Volunteer' ADD CONSTRAINT 'fk_volunteer_member' FOREIGN KEY ('M_ID') REFERENCE... 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
```

```
-- Volunteer_course

ALTER TABLE `Volunteer_course`
ADD CONSTRAINT `fk_volunteer_course_volunteer`
FOREIGN KEY (`V_ID`)
REFERENCES `Volunteer` (`V_ID`)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

```
-- Volunteer_course

ALTER TABLE `Volunteer_course`
ADD CONSTRAINT `fk_volunteer_course_course`
FOREIGN KEY (`C_ID`)
REFERENCES `Course` (`C_ID`)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Output:

```
✓ 118 22:24:47 ALTER TABLE 'Volunteer_course' ADD CONSTRAINT `fk_volunteer_course_volunteer` FOREIGN KEY (`V_ID`) REFERENCES `Volunteer` (`ID`) 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
✓ 119 22:25:16 ALTER TABLE 'Volunteer_course' ADD CONSTRAINT `fk_volunteer_course_course` FOREIGN KEY (`C_ID`) REFERENCES `Course` (`ID`) 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
```

-- Funds

```
ALTER TABLE `Funds`
ADD CONSTRAINT `fk_funds_admin`
FOREIGN KEY (`AD_ID`)
REFERENCES `Admin` (`AD_ID`)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Output:

```
✓ 119 22:25:16 ALTER TABLE 'Volunteer_course' ADD CONSTRAINT `fk_volunteer_course_course` FOREIGN KEY (`C_ID`) REFERENCES `Course` (`ID`) 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
✓ 120 22:25:51 ALTER TABLE 'Funds' ADD CONSTRAINT `fk_funds_admin` FOREIGN KEY (`AD_ID`) REFERENCES `Admin` (`AD_ID`) 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
```

### 3.3 Responsibility of G1 & G2:

---

Using the optimized ER diagram, write **five** operations, their corresponding **nested** SQL queries, and tuple relational calculus/relational algebra with the following specifications:

1. Two queries must throw an error due to violation of constraints specified by **G1**.  
Let the five operations be denoted by F1,F2,F3,F4 and F5.

F1:

```
INSERT INTO NEEV_Members (M_ID, Name, Email, Identification_ID, Gender, Designation, Photo, Password, Phone_No)
VALUES
(89, 'John Smith', 'john@example.com', 1234567890, 'other', 'Manager', 'link_to_photo', 'password1', '123-456-7890')
```

 9 19:57:36 INSERT INTO NEEV\_Members (M\_ID, Name, Email, Identification\_ID, Gender, Designation, Photo, Password, Phone\_No) VALUES ... Error Code: 1265. Data truncated for column 'Gender' at row 1

This error occurs because we had specified a constraint for Gender in the table NEEV\_Members, i.e. it can either be "Male" or "Female". So here, it will not accept the Gender as "other" as we have shown in our query and hence, it will give an error.

F2:

```
INSERT INTO Volunteer (V_ID, Name, Email, Identification_ID, Gender, Photo, Phone_No, M_ID, Address) VALUES
(898954, 'Steve Smith', 'steve@example.com', 123456789, 'other', '[Link]', '123-456-789', 'Hjashd', '123 Main St, City');
```

 26 20:11:06 INSERT INTO Volunteer (V\_ID, Name, Email, Identification\_ID, Gender, Photo, Phone\_No, M\_ID, Address) V... Error Code: 1366. Incorrect integer value: 'Hjashd' for column 'M\_ID' at row 1

This error is caused because it violates the constraint as specified for M\_ID in Volunteer which can only take an integer value. Here, we wrote random characters 'Hjashd' such that it gave the error.

Both F1 and F2 satisfy specification 1.

[As there is no relational algebra/tuple calculus for the insert operation we have not included it. ]

2. One of the functions should involve storage of an image along with a caption into the database.

```
create table student_profile
(
    profile_id int not null,
    profile_photo longblob not null,
    caption varchar(30) ,
    S_ID int not null,
    foreign key (S_ID) references students(S_ID),
    primary key(profile_id)
);
```

	profile_id	profile_photo	caption	S_ID
▶	1	BLOB	Profile photo of user	1
	2	BLOB	Profile photo of user	2
	3	BLOB	Profile photo of user	3
*	NULL	NULL	NULL	NULL

F3 satisfies Specification 2.

[As there is no relational algebra/tuple calculus for the insert operation we have not included it.]

3. Include cases of natural join, outer join, renaming, two or more different kinds of aggregate functions, and case statements in one or more queries.

Here are five operations with corresponding nested SQL queries and their relational algebra :

F4: Retrieve the total amount donated by each donor along with their names.

SQL Query:

sql

```
SELECT Donor_Name, SUM(Amount) AS Total_Donation
```

```
FROM Donation
```

```
GROUP BY Donor_Name;
```

```
198 •   SELECT Donor_Name, SUM(Amount) AS Total_Donation
199     FROM Donation
200     GROUP BY Donor_Name;
201
202
```

	Donor_Name	Total_Donation
▶	Alice Lee	50.00
	Bob Johnson	150.00
	Jane Smith	250.00
	Joey Tribianni	100.00
	Sarah Brown	75.00

Relational Algebra:

$$\pi_{\text{Donor\_Name}, \text{SUM}(\text{Amount})}(\sigma_{\text{Donation}.\text{Donor\_Name} = \text{Donation}.\text{Donor\_Name}}(\text{Donation}))$$

F5: Operation: List all students along with their corresponding courses, if any.

SQL Query:

sql

```
SELECT S.Name AS Student_Name, C.Course_name  
FROM Students S  
LEFT JOIN STUDENT_COURSE SC ON S.S_ID = SC.S_ID  
LEFT JOIN Course C ON SC.C_ID = C.C_ID;
```

```
202 •  SELECT S.Name AS Student_Name, C.Course_name  
203     FROM Students S  
204     LEFT JOIN STUDENT_COURSE SC ON S.S_ID = SC.S_ID  
205     LEFT JOIN Course C ON SC.C_ID = C.C_ID;  
206  
207
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	Student_Name	Course_name		
▶	Johny Pal	Mathematics		
	Emma Brown	Mathematics		
	Michael Lee	NULL		
	Sarah Johnson	NULL		
	David Wilson	NULL		
	Emily Davis	NULL		
	Daniel Martinez	NULL		
	Olivia Taylor	NULL		
	James Brown	NULL		
	Sonhia Garcia	NULL		

Relational Algebra:

$$\pi \text{Name, Course\_name}(\text{Students} \bowtie (\text{SC} \bowtie \text{Course}))$$

F6: Operation: Retrieve the instructors who are teaching courses along with the number of courses each instructor is teaching.

SQL Query:

sql

```
SELECT I.Name AS Instructor_Name, COUNT(AC.C_ID) AS Number_of_Courses  
FROM Instructors I  
LEFT JOIN Active_courses AC ON I.I_ID = AC.I_ID  
GROUP BY I.Name;
```

```
202  
203 •   SELECT I.Name AS Instructor_Name, COUNT(AC.C_ID) AS Number_of_Courses  
204     FROM Instructors I  
205     LEFT JOIN Active_courses AC ON I.I_ID = AC.I_ID  
206     GROUP BY I.Name;  
207  
208
```

Result Grid | Filter Rows: \_\_\_\_\_ | Export: \_\_\_\_\_ | Wrap Cell Content:

Instructor_Name	Number_of_Courses
John Doe	1
Jane Smith	1
Alex Lee	0
Emily Chen	0
Mark Kim	0

Relational Algebra:

$$\pi \text{Name, COUNT(C_ID)}(\text{Instructors} \bowtie \text{Active_courses})$$

F7: Operation: Retrieve the courses along with the count of students enrolled in each course.

SQL Query:

```
sql
SELECT C.Course_name, COUNT(SC.S_ID) AS Number_of_Students
FROM Course C
LEFT JOIN STUDENT_COURSE SC ON C.C_ID = SC.C_ID
GROUP BY C.Course_name;
```

The screenshot shows a database interface with the following details:

- SQL Editor:** Displays the SQL query with line numbers 209 through 213. The code is identical to the one above.
- Result Grid:** A table showing the results of the query. The columns are "Course\_name" and "Number\_of\_Students". The data is as follows:

Course_name	Number_of_Students
Mathematics	2
Literature	0
History	0
Computer Science	0
Biology	0

Relational Algebra:

$$\pi \text{Course\_name, COUNT(S\_ID)}(\text{Course} \bowtie \text{STUDENT\_COURSE})$$

F8: Operation: Retrieve the volunteers along with the courses they are assisting in, including those who are not currently assisting in any course.

SQL Query:

sql

```
SELECT V.Name AS Volunteer_Name, COALESCE(C.Course_name, 'Not Assisting') AS Course_Assisting  
FROM Volunteer V  
LEFT JOIN VOLUNTEER_COURSE VC ON V.V_ID = VC.V_ID  
LEFT JOIN Course C ON VC.C_ID = C.C_ID;
```

```
220  
221 •  SELECT V.Name AS Volunteer_Name, COALESCE(C.Course_name, 'Not Assisting') AS Course_Assisting  
222   FROM Volunteer V  
223   LEFT JOIN VOLUNTEER_COURSE VC ON V.V_ID = VC.V_ID  
224   LEFT JOIN Course C ON VC.C_ID = C.C_ID;  
225  
??6
```

Result Grid	
Volunteer_Name	Course_Assisting
Steve Smith	Mathematics
Emily Johnson	Literature
Michael Brown	Not Assisting
Sarah Davis	Not Assisting
David Wilson	Not Assisting

Relational Algebra:

$$\pi \text{Name, COALESCE}(\text{Course\_name}, \text{'Not Assisting'})(\text{Volunteer} \sqcap \text{Volunteer\_Course} \sqcap \text{Course})$$

ALL F4-F8 Operations follow specification 3.

Contribution:

G1Q1: Om Ambekar

G2Q2: Dhru Jain Khilosiya

G2 Q1(1-7): Himani Trivedi

G2 Q2: Priya Darji

G1&G2 Q1(1-3): Diya Parashar