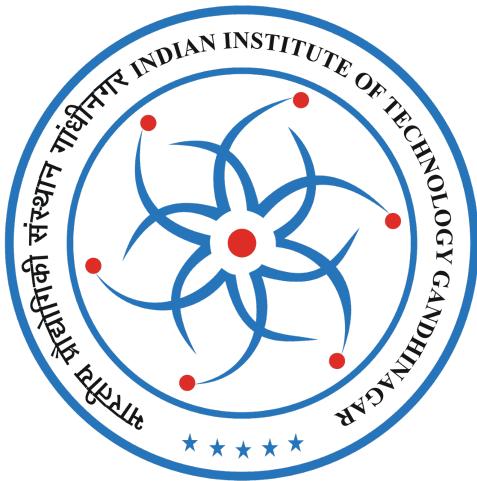


CS 432 - DATABASES

ASSIGNMENT -2



TEAM: DB11
HOSTEL MANAGEMENT

Assumptions :

- email_id length: varchar(320), the maximum email address length possible is 320 characters only.
- RESIDENT_ID: numeric(8), assuming resident_id is of length 8.
- all FIRST_NAME: varchar(15), the maximum length of first_name, middle_name, and last_name each are restricted to a maximum length of 15 characters.
- city_name: varchar(85), the maximum length of the name of a city can be of length 85 characters.
- payment_amount in the ALLOCATION table is of type int.
- Phone number data type is 10 digit numeric.

Responsibility of G1:

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.

Populating the tables with random data by following the ACID properties and all the defined constraints(e.g. Foreign key constraint).

ACID - Atomicity, Consistency, Isolation, Durability.

- Atomicity: Each transaction must be an indivisible unit of work. That is, everything happens or nothing happens. For example, if we are performing any transaction that involves multiple steps then if in between anything fails then the transaction rollbacks.
- Consistency: The database should remain valid even after the transaction.
- Isolation: Each transaction should happen independently. For example, if we are updating any tuple in a table then any other transaction should not access the same tuples.
- Durability: After a transaction is completed, the changes it made should be permanent. That is, the changes should be saved in disk so that the data could survive any failures such as crashes, etc.,

We took care to ensure that our tables follow ACID properties.

The queries to generate random data are written in population.sql and TableCreationSQL.sql.

Populating CARETAKER:

```
1 •  SELECT * FROM hostelmng.caretakers;
```

Result Grid							
	caretaker_id	first_name	middle_name	last_name	gender	office_no	email_id
▶	10000001	caretakername1	caretakermname1	caretakername1	M	a181	caretakeremail1@iitgn.ac.in
	10000002	caretakername2	caretakermname2	caretakername2	F	b199	caretakeremail2@iitgn.ac.in
	10000003	caretakername3	caretakermname3	caretakername3	M	c160	caretakeremail3@iitgn.ac.in
	10000004	caretakername4	caretakermname4	caretakername4	M	d146	caretakeremail4@iitgn.ac.in
	10000005	caretakername5	caretakermname5	caretakername5	M	e135	caretakeremail5@iitgn.ac.in
	10000006	caretakername6	caretakermname6	caretakername6	M	f157	caretakeremail6@iitgn.ac.in
	10000007	caretakername7	caretakermname7	caretakername7	F	g175	caretakeremail7@iitgn.ac.in

Populating HOSTEL:

Result Grid							
	hostel_name	contact	total_rooms	total_students	energy_consumption	water_consumption	caretaker_id
▶	a	1000000000	301	494	146.13	85.51	10000001
	b	1000000001	301	499	689.18	478.47	10000002
	c	1000000002	301	466	188.67	968.91	10000003
	d	1000000003	301	463	486.02	594.43	10000004
	e	1000000004	301	475	787.11	393.33	10000005
	f	1000000005	301	480	846.66	417.28	10000006
	g	1000000006	301	477	480.42	762.72	10000007
	h	1000000007	301	468	573.64	751.12	10000001
	i	1000000008	301	451	919.97	495.85	10000002
	j	1000000009	301	485	109.14	387.71	10000003
	k	1000000010	301	480	892.41	628.75	10000004
	l	1000000011	301	473	446.28	831.87	10000005

Populating CARETAKER_PHONE:

Result Grid		
	phone_no	caretaker_id
	7000000007	10000002
	7000000008	10000003
	7000000009	10000003
	7000000010	10000003
	7000000011	10000004
	7000000012	10000005
	7000000013	10000005
	7000000014	10000005
	7000000015	10000006
	7000000016	10000006
	7000000017	10000007
	7000000018	10000007

populating GUARD:

```
1 •  SELECT * FROM hostelmng.guard;
```

security_id	first_name	middle_name	last_name
80000001	guardFName1	guardMName1	guardLName1
80000002	guardFName2	guardMName2	guardLName2
80000003	guardFName3	guardMName3	guardLName3
80000004	guardFName4	guardMName4	guardLName4
80000005	guardFName5	guardMName5	guardLName5
80000006	guardFName6	guardMName6	guardLName6
80000007	guardFName7	guardMName7	guardLName7
80000008	guardFName8	guardMName8	guardLName8
80000009	guardFName9	guardMName9	guardLName9
80000010	guardFName10	guardMName...	guardLName10

populating GUARD_PHONE:

phone_no	security_id
7000000007	80000003
7000000008	80000004
7000000009	80000005
7000000010	80000005
7000000011	80000005
7000000012	80000006
7000000013	80000006
7000000014	80000007
7000000015	80000007
7000000016	80000008
7000000017	80000008
7000000018	80000009
7000000019	80000009
7000000020	80000009
7000000021	80000010

populating RESIDENT:

resident_id	first_name	middle_name	last_name	gender	blood_group	email_id	city	postal_code	home_contact	resident_type	on_campus
20000000	residentfn1	residentmn1	residentln1	M	b-	residentemail1@iitgn.ac.in	city1	100000	5000000000	faculty	1
20000001	residentfn2	residentmn2	residentln2	M	o+	residentemail2@iitgn.ac.in	city2	100001	5000000001	visitor	1
20000002	residentfn3	residentmn3	residentln3	F	a+	residentemail3@iitgn.ac.in	city3	100002	5000000002	faculty	1
20000003	residentfn4	residentmn4	residentln4	F	b+	residentemail4@iitgn.ac.in	city4	100003	5000000003	student	0
20000004	residentfn5	residentmn5	residentln5	F	o+	residentemail5@iitgn.ac.in	city5	100004	5000000004	visitor	0
20000005	residentfn6	residentmn6	residentln6	M	b+	residentemail6@iitgn.ac.in	city6	100005	5000000005	visitor	1
20000006	residentfn7	residentmn7	residentln7	M	o+	residentemail7@iitgn.ac.in	city7	100006	5000000006	visitor	0
20000007	residentfn8	residentmn8	residentln8	F	b-	residentemail8@iitgn.ac.in	city8	100007	5000000007	visitor	0
20000008	residentfn9	residentmn9	residentln9	F	b+	residentemail9@iitgn.ac.in	city9	100008	5000000008	faculty	1
20000009	residentfn10	residentmn10	residentln10	F	b+	residentemail10@iitgn.ac.in	city10	100009	5000000009	student	0
20000010	residentfn11	residentmn11	residentln11	F	o+	residentemail11@iitgn.ac.in	city11	100010	5000000010	faculty	1
20000011	residentfn12	residentmn12	residentln12	F	a+	residentemail12@iitgn.ac.in	city12	100011	5000000011	faculty	1
20000012	residentfn13	residentmn13	residentln13	F	b+	residentemail13@iitgn.ac.in	city13	100012	5000000012	student	0
20000013	residentfn14	residentmn14	residentln14	M	o-	residentemail14@iitgn.ac.in	city14	100013	5000000013	faculty	0
20000014	residentfn15	residentmn15	residentln15	M	-	residentemail15@iitgn.ac.in	city15	100014	5000000014	visitor	1

populating ACADEMIC_PERIOD:

	semester	year
1		2014
1		2015
1		2016
1		2017
1		2018
1		2019
1		2020
1		2021
1		2022
2		2008
2		2009
2		2010
2		2011
2		2012
2		2013
2		2014

populating DEGREE:

	program	branch
p1	b1	
p1	b2	
p1	b3	
p1	b4	
p1	b5	
p1	b6	
p2	b1	
p2	b2	
p2	b3	
p2	b4	
p2	b5	
p2	b6	
p3	b1	
p3	b2	
p3	b3	
n3	b4	

populating RESIDENT_PHONE:

	phone_no	resident_id
	5000000124	20000049
	5000000125	20000050
	5000000126	20000050
	5000000127	20000051
	5000000128	20000051
	5000000129	20000051
	5000000130	20000052
	5000000131	20000052
	5000000132	20000053
	5000000133	20000053
	5000000134	20000053
	5000000135	20000053
	5000000136	20000054
	5000000137	20000055
	5000000138	20000056
	5000000139	20000056

populating OUTLET:

A screenshot of a database result grid titled "Result Grid". The grid has a header row with columns: outlet_name, open_time, close_time, contact, owner_first_name, owner_middle_name, owner_last_name, and hostel_name. Below the header, there are six data rows corresponding to outlet1 through outlet6. The data shows various times and names for each outlet.

	outlet_name	open_time	close_time	contact	owner_first_name	owner_middle_name	owner_last_name	hostel_name
▶	outlet1	12:00:00	01:00:00	4000000000	ownerfname1	ownermname1	ownername1	e
	outlet2	12:00:00	02:00:00	4000000001	ownerfname2	ownermname2	ownername2	e
	outlet3	05:00:00	01:00:00	4000000002	ownerfname3	ownermname3	ownername3	a
	outlet4	10:00:00	00:00:00	4000000003	ownerfname4	ownermname4	ownername4	a
	outlet5	09:00:00	01:00:00	4000000004	ownerfname5	ownermname5	ownername5	h
	outlet6	06:00:00	00:00:00	4000000005	ownerfname6	ownermname6	ownername6	h

populating OUTLET_PHONE:

A screenshot of a database result grid titled "Result Grid". The grid has a header row with columns: phone_no and outlet_name. Below the header, there are twelve data rows, each mapping a unique phone number to an outlet name. The outlet names repeat for outlets 2, 3, 4, 5, and 6.

	phone_no	outlet_name
	4100000002	outlet1
	4100000003	outlet2
	4100000004	outlet2
	4100000005	outlet3
	4100000006	outlet3
	4100000007	outlet4
	4100000008	outlet4
	4100000009	outlet5
	4100000010	outlet5
	4100000011	outlet6
	4100000012	outlet6

populating OUTLET_OWNER_PHONE:

A screenshot of a database result grid titled "Result Grid". The grid has a header row with columns: phone_no and outlet_name. Below the header, there are twelve data rows, each mapping a unique phone number to an outlet name. The outlet names repeat for outlets 2, 3, 4, 5, and 6.

	phone_no	outlet_name
	4200000003	outlet2
	4200000004	outlet2
	4200000005	outlet3
	4200000006	outlet3
	4200000007	outlet4
	4200000008	outlet4
	4200000009	outlet5
	4200000010	outlet5
	4200000011	outlet6
	4200000012	outlet6

populating ROOM:

	room_no	hostel_name	room_type	occupied
	100	j	bathroom	0
	100	k	bathroom	0
	101	a	triple	0
	101	b	triple	0
	101	c	triple	0
	101	d	triple	0
	101	e	triple	0
	101	f	triple	0
	101	g	triple	0
	101	h	triple	0
	101	i	triple	0
	101	j	triple	1
	101	k	triple	0
	102	a	single	0
...	.	.	.	-

populating ALLOCATION:

```
1 •  SELECT * FROM hostelmng.allocation;
```

	semester	year	resident_id	room_no	hostel_name	entry_date	exit_date	payment_status	due_amount	due_status	payment_amount
	2	2021	20000274	386	a	2021-01-02	2021-04-30	5000	0	0	5000
	2	2021	20000285	397	j	2021-01-02	2021-04-30	5000	0	0	5000
	2	2021	20000289	402	j	2021-01-02	2021-04-30	5000	0	0	5000
	2	2021	20000290	403	a	2021-01-02	2021-04-30	5000	0	0	5000
	2	2021	20000294	407	j	2021-01-02	2021-04-30	5000	0	0	5000
	3	2020	2000003	104	a	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000016	117	a	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000017	118	a	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000018	119	j	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000019	120	j	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000022	123	j	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000024	126	123	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000031	133	j	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000032	134	a	2020-09-01	2020-11-30	5000	0	0	5000
	3	2020	20000037	139	j	2020-09-01	2020-11-30	5000	0	0	5000
3	2020	20000029	141	=	=	2020-09-01	2020-11-30	5000	0	0	5000

populating SECURITY:

```
1 •   SELECT * FROM hostelmng.security;
```

	security_id	hostel_name	start_time	end_time
	80000002	j	05:00:00	19:00:00
	80000003	e	16:00:00	07:00:00
	80000003	h	16:00:00	07:00:00
	80000004	a	04:00:00	21:00:00
	80000004	l	04:00:00	21:00:00
	80000005	b	05:00:00	18:00:00
	80000005	f	05:00:00	18:00:00
	80000006	d	03:00:00	17:00:00
	80000006	e	03:00:00	17:00:00
	80000007	g	12:00:00	02:00:00
	80000007	i	12:00:00	02:00:00
	80000008	g	17:00:00	00:00:00
	80000008	k	17:00:00	00:00:00
	80000009	f	17:00:00	05:00:00
	80000009	l	17:00:00	05:00:00
	80000010	e	23:00:00	01:00:00

populating FURNITURE:

```
1 •   SELECT * FROM hostelmng.furniture;
```

	furniture_id	status	room_no	hostel_name
	furniture102b	1	102	b
	furniture102c	1	102	c
	furniture102d	1	102	d
	furniture102e	1	102	e
	furniture102f	1	102	f
	furniture102g	1	102	g
	furniture102h	1	102	h
	furniture102i	1	102	i
	furniture102j	1	102	j
	furniture102k	1	102	k
	furniture103a	1	103	a
	furniture103b	1	103	b
	furniture103c	1	103	c
	furniture103d	1	103	d
	furniture103e	1	103	e

Populating ENROLLED_IN:

```
1 •   SELECT * FROM hostelmng.enrolled_in;
```

	resident_id	program	branch
▶	20000015	p1	b1
	20000060	p1	b1
	20000075	p1	b1
	20000090	p1	b1
	20000120	p1	b1
	20000180	p1	b1
	20000240	p1	b1
	20000051	p1	b2
	20000081	p1	b2
	20000096	p1	b2
	20000126	p1	b2
	20000246	p1	b2
	20000291	p1	b2
	20000012	p1	b3
	20000027	p1	b3

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.

In general, we chose the search key on the following basis:

1. Uniqueness: A search key should be unique for each record in the table so that it can be used to uniquely identify and retrieve a specific record.
2. Stability: A search key should be stable over time and not frequently change to be used consistently to locate a record.
3. Minimality: A search key should be as small as possible to reduce storage requirements and search time.
4. Accessibility: A search key should be easily accessible and efficiently searchable to retrieve records quickly.
5. Relevance: To optimize performance, a search key should be relevant to the queries and operations commonly performed on the table.

NOTE: Wherever we are using the primary key as the search key, the reason is **Efficiency**: Since the primary key is typically indexed automatically by the database management system, searches using the primary key as the search key can be performed efficiently and quickly, without requiring a full table scan.

Sr no	Schema name	Search Key (Indexing)	Reason	SQL Implementation
1.	Hostel	hostel_name	Since a hostel is uniquely identified by and mostly used by its name, selecting hostel_name would be the most suitable choice.	<p>Index: <code>hostel_name</code></p> <p>Definition:</p> <pre>Type BTREE Unique Yes Visible Yes Columns hostel_name</pre>
2.	Outlet	outlet_name	Since an outlet is most commonly called by its name.	<p>Index: <code>outlet_name</code></p> <p>Definition:</p> <pre>Type BTREE Unique Yes Visible Yes Columns outlet_name</pre>
3.	Caretaker	caretaker_id	The caretaker_id attribute satisfies the properties of a good search key, as it is unique for each record in the table, stable, minimal, accessible, and relevant to the queries that are commonly performed on the table.	<p>Index: <code>caretaker_id</code></p> <p>Definition:</p> <pre>Type BTREE Unique Yes Visible Yes Columns caretaker_id</pre>

4.	Room	room_no	For easy accessibility, more frequently addressed and is unique to a particular room.	Index: room_no Definition: Type BTREE Unique No Visible Yes Columns room_no
5.	degree	program	Since the program(ex: Btech) covers a wide range of data(students) involved rather than a specific degree (ex, CSE).	Index: program Definition: Type BTREE Unique No Visible Yes Columns program
6.	furniture	hostel_name	We would be checking furniture hostel-wise. It would be convenient to use it.	Index: hostel_name Definition: Type BTREE Unique No Visible Yes Columns hostel_name room_no
7.	Allocation	room_no	Since the room allocation would be done based on the room no, all associated information(like allocated to whom) can be accessed by knowing the room_no.	Index: room_no Definition: Type BTREE Unique No Visible Yes Columns room_no
8.	academic_period	year	A acad period can be accessed by the year since a year contains only 2 semesters, but semesters would repeat each and every year.	Index: year Definition: Type BTREE Unique No Visible Yes Columns year

9.	resident	resident_id	As there are more residents, we use mostly resident_id's rather than names to recognize them easily based on the need.	Index: resident_id Definition: Type BTREE Unique Yes Visible Yes Columns resident_id
10.	resident_phone	resident_id	In the database, we go to the resident's phone through his resident_id.	Index: resident_id Definition: Type BTREE Unique No Visible Yes Columns resident_id
11.	guard	security_id	We recognize a guard with his security_id.	Index: security_id Definition: Type BTREE Unique Yes Visible Yes Columns security_id
12.	caretaker_phone	caretaker_id	In a database, we go through the resident's phone through caretaker_id.	Index: caretaker_id Definition: Type BTREE Unique No Visible Yes Columns caretaker_id
13.	security	hostel_name	If we query the data primarily based on the guards assigned to each hostel, then the hostel_name attribute is a more preferred search key.	Index: hostel_name Definition: Type BTREE Unique No Visible Yes Columns hostel_name
14.	enrolled_in	program	We'll search for the program and then look up who is enrolled in a particular program.	Index: program Definition: Type BTREE Unique No Visible Yes Columns program branch

15.	guard_phone	security_id	In the database, we go through the guard's phone through his security_id.	Index: security_id Definition: Type BTREE Unique No Visible Yes Columns security_id
16.	outlet_owner_phone	outlet_name	In the database, we go through the outlet's owner's phone through the outlet_name.	Index: outlet_name Definition: Type BTREE Unique No Visible Yes Columns outlet_name
17.	outlet_phone	outlet_name	In the database, we go through the outlet's phone through oulet_name.	Index: outlet_name Definition: Type BTREE Unique No Visible Yes Columns outlet_name

User Defined Datatypes:

MySQL does not support creating type statements to execute user defined datatype queries, but we use JSON to implement user defined data types.

We define user defined datatypes on email_id's.

```
) WHILE (counter <= 300) DO -- change 10 to the desired number of tuples to insertw

  SET resident_id = resident_id ; -- generates random hostel name
  SET first_name = concat("residentfn" , CAST(counter as char)) ;
  SET middle_name = concat("residentmn" , CAST(counter as char)) ;
  SET last_name = concat("residentln" , CAST(counter as char)) ;
  SET gender = CASE WHEN ROUND(RAND()*10,0) % 2 =0 THEN 'M' ELSE 'F' END;
  SET blood_group = concat(elt(floor(1 + rand() * 3), "o", "a", "b"), elt(floor(1 + rand() * 2), "+", "-"));
  SET email_id = JSON_OBJECT("name", concat("residentemail", cast(counter as CHAR)), "domain", "@iitgn.ac.in");
  SET city = concat("city", CAST(counter as char)) ;
  SET postal_code = postal_code ;
  SET home_contact = home_contact ;
  SET
```

```
create table RESIDENT(
    resident_id numeric(8) PRIMARY KEY NOT NULL UNIQUE,
    first_name varchar(15) NOT NULL,
    middle_name varchar(15),
    last_name varchar(15) NOT NULL,
    gender char(1) NOT NULL,
    blood_group char(3) NOT NULL,
    email_id JSON NOT NULL UNIQUE,
    city varchar(85) NOT NULL,
    postal_code numeric(6, 0) NOT NULL,
    home_contact numeric(10, 0) NOT NULL UNIQUE,
    resident_type varchar(15) NOT NULL,
    on_campus bool NOT NULL
);
```

Table Extensions:

The query for creating table temp_resident which has the same schema as resident.

```
create table temp_resident like RESIDENT;
```

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Schemas: RESIDENT, RESIDENT_PHONE, ROOM, SECURITY, temp_resident

Table: temp_resident

Columns: resident_id

Query:

```
1 • SELECT * FROM hostelmng.temp_resident;
```

Result Grid:

#	resident_ic	first_name	middle_name	last_name	gender	blood_group	email_id	city	postal_code	home_contact	resident_type	on_campus
1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

temp_resident 1

The query for creating and populating table t1 which has same schema as resident and gender female.

```
create table t1 as
( select * from RESIDENT where gender = 'F' ) ;
```

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Schemas: RESIDENT, RESIDENT_PHONE, ROOM, SECURITY, t1, temp_resident

Table: t1

Columns: resident_id

Query:

```
1 • SELECT * FROM hostelmng.t1;
```

Result Grid:

#	resident_ic	first_name	middle_name	last_name	gender	blood_group	email_id	city	postal_code	home_contact	resident_type	on_campus
1	20000002	residentfn3	residentmn3	residentln3	F	o+	residentemail3@iitgn.ac.in	city3	100002	5000000002	visitor	0
2	20000003	residentfn4	residentmn4	residentln4	F	o+	residentemail4@iitgn.ac.in	city4	100003	5000000003	visitor	1
3	20000005	residentfn6	residentmn6	residentln6	F	o-	residentemail6@iitgn.ac.in	city6	100005	5000000005	visitor	1
4	20000006	residentfn7	residentmn7	residentln7	F	b+	residentemail7@iitgn.ac.in	city7	100006	5000000006	visitor	0
5	20000008	residentfn9	residentmn9	residentln9	F	a-	residentemail9@iitgn.ac.in	city9	100008	5000000008	visitor	1
6	20000009	residentfn10	residentmn10	residentln10	F	a+	residentemail10@iitgn.ac.in	city10	100009	5000000009	visitor	1
7	20000010	residentfn11	residentmn11	residentln11	F	b-	residentemail11@iitgn.ac.in	city11	100010	5000000010	faculty	1
8	20000019	residentfn20	residentmn20	residentln20	F	a-	residentemail20@iitgn.ac.in	city20	100019	5000000019	faculty	0
9	20000022	residentfn23	residentmn23	residentln23	F	a+	residentemail23@iitgn.ac.in	city23	100022	5000000022	visitor	0
10	20000023	residentfn24	residentmn24	residentln24	F	b+	residentemail24@iitgn.ac.in	city24	100023	5000000023	faculty	1
11	20000024	residentfn25	residentmn25	residentln25	F	o-	residentemail25@iitgn.ac.in	city25	100024	5000000024	student	1
12	20000027	residentfn28	residentmn28	residentln28	F	o+	residentemail28@iitgn.ac.in	city28	100027	5000000027	visitor	0
13	20000028	residentfn29	residentmn29	residentln29	F	a-	residentemail29@iitgn.ac.in	city29	100028	5000000028	faculty	0
14	20000031	residentfn32	residentmn32	residentln32	F	a+	residentemail32@iitgn.ac.in	city32	100031	5000000031	visitor	1
15	20000033	residentfn34	residentmn34	residentln34	F	b+	residentemail34@iitgn.ac.in	city34	100033	5000000033	student	0

t1 1

3.2 Responsibility of G2:

1. Create a user named "user1" with the password "password1".

Created a user named "user1" with password Lunch@4u. According to the question we should set the password as "password1" but it is rejected by mySQL as it is trivial.

```
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'Lunch@4u';
```

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.

View1:

Creating view1 on the tables RESIDENT and ENROLLED_IN, the columns that we select from RESIDENT are resident_id, first_name, email_id, the columns that we select from ENROLLED_IN are program, branch. The additional column with user defined data type is resident email_id.

```
CREATE VIEW view1 AS
SELECT r.resident_id, r.first_name, r.email_id, ei.program, ei.branch
FROM RESIDENT as r
INNER JOIN ENROLLED_IN as ei
ON ei.resident_id = r.resident_id;
```

View2:

Creating view2 on the tables CARETAKER and HOSTEL, the columns that we select from CARETAKER are caretaker_id, first_name, email_id, the columns that we select from HOSTEL are hostel_name, contact. The additional column with user defined data type is caretaker email_id.

```
CREATE VIEW view2 AS
SELECT c.caretaker_id, c.first_name, c.email_id, h.hostel_name, h.contact
FROM CARETAKER as c
INNER JOIN HOSTEL as h
ON c.caretaker_id = h.caretaker_id;
```

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

- SELECT
- UPDATE
- DELETE

Granting user1 SELECT, UPDATE, DELETE permissions on table RESIDENT.

We use the RESIDENT table as “table1”.

```
# Q3 granting permissions on table
GRANT SELECT, UPDATE, DELETE ON RESIDENT TO "user1"@"localhost";
```

4. **Grant "user1" the following permissions on "view1":**

- SELECT

Granting user1 SELECT permission on view1.

```
# Q4 granting permissions on view
GRANT SELECT ON view1 TO "user1"@"localhost";
```

5. **Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" and report your findings.**

On RESIDENT(table1):

Since the user1 has been given permissions to SELECT, UPDATE, and DELETE on RESIDENT, when we execute the query, we should get no error.

SELECT operation:

Activities Mar 1 04:23 MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Administration Schemas

SCHEMAS Filter objects

hostelmng Tables Views view1 resident_id first_name email_id program branch

Stored Procedures Functions

Query 1 g2* Limit to 1000 rows

```

21 # Q4 granting permissions on view
22 • GRANT SELECT ON view1 TO 'user1'@'localhost';
23
24 # Q5
25 # ON Table
26 • use hostelmng;
27 • select * from RESIDENT where gender = "F";
28

```

Result Grid Filter Rows: A Edit: Export/Import: Wrap Cell Content: Result Grid Form Editor

#	resident_id	first_name	middle_name	last_name	gender	blood_group	email_id	city	postal_code	home_contact	resident_type
1	20000000	residentfn1	residentmn1	residentln1	F	a-	residentemail1@iitgn.ac.in	city1	100000	5000000000	visitor
2	20000001	residentfn2	residentmn2	residentln2	F	b-	residentemail2@iitgn.ac.in	city2	100001	5000000001	student
3	20000004	residentfn5	residentmn5	residentln5	F	b+	residentemail5@iitgn.ac.in	city5	100004	5000000004	faculty
4	20000005	residentfn6	residentmn6	residentln6	F	a+	residentemail6@iitgn.ac.in	city6	100005	5000000005	visitor
5	20000006	residentfn7	residentmn7	residentln7	F	b+	residentemail7@iitgn.ac.in	city7	100006	5000000006	visitor
6	20000007	residentfn8	residentmn8	residentln8	F	a-	residentemail8@iitgn.ac.in	city8	100007	5000000007	faculty

Action Output

#	Time	Action	Message	Duration / Fetch
1	04:21:57	select * from RESIDENT where gender = "F" LIMIT 0, 1000	Error Code: 1046. No database selected Select the default DB to be used by double-clicking...	0.00050 sec
2	04:22:16	use hostelmng	0 row(s) affected	0.00043 sec
3	04:22:19	select * from RESIDENT where gender = "F" LIMIT 0, 1000	151 row(s) returned	0.0013 sec / 0.0002...

Object Info Session

Query Completed

UPDATE operation:

Activities Mar 1 04:29 MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Administration Schemas

SCHEMAS Filter objects

hostelmng Tables Views view1 resident_id first_name email_id program branch

Stored Procedures Functions

Query 1 g2* population

```

25 # ON Table
26 • use hostelmng;
27 • select * from RESIDENT where gender = "F";
28
29 • SET SQL_SAFE_UPDATES = 0;
30 • update RESIDENT|
31 set on_campus = 1
32 where on_campus = 0;
33
34 • delete from RESIDENT
35 where on_campus = 0;
36
37 # On view
38 • select * from view1 where gender = "F";
39

```

Action Output

#	Time	Action	Message	Duration / Fetch
1	04:28:30	SET SQL_SAFE_UPDATES = 0	0 row(s) affected	0.00038 sec
2	04:28:32	update RESIDENT set on_campus = 1 where on_campus = 0	138 row(s) affected Rows matched: 138 Changed: 138 Warnings: 0	0.0066 sec

Object Info Session

Query Completed

DELETE operation:

The screenshot shows the MySQL Workbench interface. The top bar displays 'Activities' and the date 'Mar 1 04:30'. The main window has a title bar 'user1' and tabs for 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help'. The 'Schemas' tab is selected. In the left sidebar, under 'hostelmg', there are 'Tables', 'Views', and 'Stored Procedures'. The 'Views' section contains 'view1' which is expanded to show columns: resident_id, first_name, email_id, program, and branch. The 'Query Editor' tab 'Query 1' contains the following SQL code:

```

25 # ON Table
26 • use hostelmg;
27 • select * from RESIDENT where gender = "F";
28
29 • SET SQL_SAFE_UPDATES = 0;
30 • update RESIDENT
31 set on_campus = 1
32 where on_campus = 0;
33
34 • delete from RESIDENT
35 where on_campus = 0;
36
37 # On view
38 • select * from view1 where gender = "F";
39

```

The 'Action Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	04:28:30	SET SQL_SAFE_UPDATES = 0	0 row(s) affected	0.00038 sec
2	04:28:32	update RESIDENT set on_campus = 1 where on_campus = 0	138 row(s) affected Rows matched: 138 Changed: 138 Warnings: 0	0.0066 sec
3	04:30:16	delete from RESIDENT where on_campus = 0	0 row(s) affected	0.0011 sec

The 'Result Grid' pane shows the data from 'view1' with 8 rows:

#	resident_id	first_name	email_id	program	branch
1	20000075	residentfn76	residentemail76@iitgn.ac.in	p1	b1
2	20000105	residentfn106	residentemail106@iitgn.ac.in	p1	b1
3	20000150	residentfn151	residentemail151@iitgn.ac.in	p1	b1
4	20000180	residentfn181	residentemail181@iitgn.ac.in	p1	b1
5	20000066	residentfn67	residentemail67@iitgn.ac.in	p1	b2
6	20000126	residentfn127	residentemail127@iitgn.ac.in	p1	b2
7	20000201	residentfn202	residentemail202@iitgn.ac.in	p1	b2
8	20000261	residentfn262	residentemail262@iitgn.ac.in	p1	b2

As predicted, the observations correlated with our expectations.

On VIEW1:

Since the user1 has been given permissions to only SELECT on view1, when we execute the query, we should get no error for this, but for update and delete operation, we should get an error.

SELECT operation:

The screenshot shows the MySQL Workbench interface. The top bar displays 'Activities' and the date 'Mar 1 04:30'. The main window has a title bar 'user1' and tabs for 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help'. The 'Schemas' tab is selected. In the left sidebar, under 'hostelmg', there are 'Tables', 'Views', and 'Stored Procedures'. The 'Views' section contains 'view1' which is expanded to show columns: resident_id, first_name, email_id, program, and branch. The 'Query Editor' tab 'Query 1' contains the following SQL code:

```

34 • delete from RESIDENT
35 where on_campus = 0;
36
37 # On view
38 • select * from view1 where gender = "F";
39

```

The 'Action Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	04:28:30	SET SQL_SAFE_UPDATES = 0	0 row(s) affected	0.00038 sec
2	04:28:32	update RESIDENT set on_campus = 1 where on_campus = 0	138 row(s) affected Rows matched: 138 Changed: 138 Warnings: 0	0.0066 sec
3	04:30:16	delete from RESIDENT where on_campus = 0	0 row(s) affected	0.0011 sec
4	04:31:21	select * from view1 where gender = "F" LIMIT 0, 1000	Error Code: 1054, Unknown column 'gender' in 'w...'	0.00036 sec
5	04:31:43	select * from view1 where program = 'p1' LIMIT 0, 1000	28 row(s) returned	0.00085 sec / 0.000...

The 'Result Grid' pane shows the data from 'view1' with 8 rows:

#	resident_id	first_name	email_id	program	branch
1	20000075	residentfn76	residentemail76@iitgn.ac.in	p1	b1
2	20000105	residentfn106	residentemail106@iitgn.ac.in	p1	b1
3	20000150	residentfn151	residentemail151@iitgn.ac.in	p1	b1
4	20000180	residentfn181	residentemail181@iitgn.ac.in	p1	b1
5	20000066	residentfn67	residentemail67@iitgn.ac.in	p1	b2
6	20000126	residentfn127	residentemail127@iitgn.ac.in	p1	b2
7	20000201	residentfn202	residentemail202@iitgn.ac.in	p1	b2
8	20000261	residentfn262	residentemail262@iitgn.ac.in	p1	b2

UPDATE operation:

The screenshot shows the MySQL Workbench interface. The left sidebar shows the schema 'hostelmg' with a view named 'view1' selected. The main area contains a SQL editor with the following code:

```
32 where on_campus = 0;
33
34 • delete from RESIDENT
35 where on_campus = 0;
36
37 # On view
38 • select * from view1 where program = "p1";
39
40 • update view1
41 set on_campus = 1
42 where on_campus = 0;
43
44 • delete from view1
45 where on_campus = 0;
46
```

The Action Output pane at the bottom shows a single log entry:

#	Time	Action	Message	Duration / Fetch
1	04:33:23	update view1 set on_camp...	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'	0.00031 sec

DELETE operation:

The screenshot shows the MySQL Workbench interface. The left sidebar shows the schema 'hostelmg' with a view named 'view1' selected. The main area contains a SQL editor with the following code:

```
32 where on_campus = 0;
33
34 • delete from RESIDENT
35 where on_campus = 0;
36
37 # On view
38 • select * from view1 where program = "p1";
39
40 • update view1
41 set on_campus = 1
42 where on_campus = 0;
43
44 • delete from view1
45 where on_campus = 0;
46
```

The Action Output pane at the bottom shows two log entries:

#	Time	Action	Message	Duration / Fetch
1	04:33:23	update view1 set on_camp...	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'	0.00031 sec
2	04:34:22	delete from view1 where o...	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'	0.00018 sec

As predicted, the observations match with our expectations.

6. Revoke the UPDATE and DELETE permissions on "table1" for "user1" and report your findings.

Revoking the UPDATE and DELETE permissions on RESIDENT for user1.

REVOKE UPDATE, DELETE ON RESIDENT FROM "user1"@"localhost";

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the connection is set to 'Local instance 3306'. The main area displays a SQL editor with the following code:

```
34 • delete from RESIDENT
35   where on_campus = 0;
36
37 # On view
38 • select * from view1 where program = "p1";
39
40 • update view1
41   set on_campus = 1
42   where on_campus = 0;
43
44 • delete from view1
45   where on_campus = 0;
46
47 # Q6 Revoking permissions
48 • REVOKE UPDATE, DELETE ON RESIDENT FROM "user1"@"localhost";
```

Below the editor, the 'Action Output' pane shows a single log entry:

#	Time	Action	Message	Duration / Fetch
1	04:37:51	REVOKE UPDATE, DELETE ON RESIDENT FROM "user1"@"localhost";	0 row(s) affected	0.0028 sec

The bottom left corner of the interface shows 'Query Completed'.

7. Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" again.

On RESIDENT(table1):

Now that we removed the UPDATE and DELETE permissions on the RESIDENT table for user1, the executions should give errors for UPDATE and DELETE operations. SELECT operation should give no error.

SELECT operation:

Activities Mar 1 04:41 MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Administration Schemas

Schemas

hostelmng

- Tables
- Views
- Stored Procedures
- Functions

Query 1 g2

```

25 # ON Table
26 • use hostelmng;
27 • select * from RESIDENT where gender = "F";
28

```

Result Grid Limit to 1000 rows

#	resident_id	first_name	middle_name	last_name	gender	blood_group	email_id	city	postal_code	home_contact	resident_type
1	20000000	residentf1	residentmn1	residentl1	F	a+	residentemail1@iitgn.ac.in	city1	100000	5000000000	visitor
2	20000001	residentf2	residentmn2	residentl2	F	b-	residentemail2@iitgn.ac.in	city2	100001	5000000001	student
3	20000004	residentf5	residentmn5	residentl5	F	b+	residentemail5@iitgn.ac.in	city5	100004	5000000004	faculty
4	20000005	residentf6	residentmn6	residentl6	F	a+	residentemail6@iitgn.ac.in	city6	100005	5000000005	visitor
5	20000006	residentf7	residentmn7	residentl7	F	b+	residentemail7@iitgn.ac.in	city7	100006	5000000006	visitor
6	20000007	residentf8	residentmn8	residentl8	F	a-	residentemail8@iitgn.ac.in	city8	100007	5000000007	faculty
7	20000008	residentf9	residentmn9	residentl9	F	a+	residentemail9@iitgn.ac.in	city9	100008	5000000008	faculty
8	20000012	residentf13	residentmn13	residentl13	F	a-	residentemail13@iitgn.ac.in	city13	100012	5000000012	faculty

RESIDENT 1

Action Output

#	Time	Action	Message	Duration / Fetch
1	04:41:05	use hostelmng	0 row(s) affected	0.00041 sec
2	04:41:09	select * from RESIDENT where gender = "F" LIMIT 0, 1000	151 row(s) returned	0.0013 sec / 0.0002...

Object Info Session

No object selected

Query Completed

UPDATE operation:

Activities Mar 1 04:42 MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Administration Schemas

Schemas

hostelmng

- Tables
- Views
- Stored Procedures
- Functions

Query 1 g2

```

25 # ON Table
26 • use hostelmng;
27 • select * from RESIDENT where gender = "F";
28
29 • SET SQL_SAFE_UPDATES = 0;
30 • update RESIDENT
31 set on_campus = 1
32 where on_campus = 0;
33
34 • delete from RESIDENT
35 where on_campus = 0;
36
37 # On view
38 • select * from view1 where program = "p1";
39

```

Action Output

#	Time	Action	Message	Duration / Fetch
1	04:41:05	use hostelmng	0 row(s) affected	0.00041 sec
2	04:41:09	select * from RESI...	151 row(s) returned	0.0013 sec / 0.0002...
3	04:42:25	SET SQL_SAFE_UP...	0 row(s) affected	0.00035 sec
4	04:42:29	update RESIDENT ...	Error Code: 1142; UPDATE command denied to user 'user1'@'localhost' for table 'RESIDENT'	0.00038 sec

Object Info Session

No object selected

Query interrupted

DELETE operation:

The screenshot shows the MySQL Workbench interface. The top bar displays 'Activities' and the date 'Mar 1 04:43'. The main window has two tabs: 'user1' and 'g2'. The 'user1' tab is active, showing a query editor with the following SQL code:

```

26 • use hostelmng;
27 • select * from RESIDENT where gender = "F";
28
29 • SET SQL_SAFE_UPDATES = 0;
30 • update RESIDENT
31 set on_campus = 1
32 where on_campus = 0;
33
34 • delete from RESIDENT
35 where on_campus = 0;
36
37 # On view
38 • select * from view1 where program = "p1";
39
40 • update view1

```

Below the query editor is an 'Action Output' table:

#	Time	Action	Message	Duration / Fetch
1	04:41:05	use hostelmng	0 row(s) affected	0.00041 sec
2	04:41:09	select * from RESI...	151 row(s) returned	0.0013 sec / 0.0002...
3	04:42:25	SET SQL_SAFE_UP...	0 row(s) affected	0.00035 sec
4	04:42:29	update RESIDENT ...	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'RESIDENT'	0.00038 sec
5	04:43:23	delete from RESID...	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'RESIDENT'	0.00039 sec

The bottom status bar indicates 'Query interrupted'.

As predicted, the observations correlated with our expectations.

On view1:

We haven't made any changes to view1 permissions. So, they should work the same as they were in question 5.

SELECT operation:

The screenshot shows the MySQL Workbench interface. The top bar displays 'Activities' and the date 'Mar 1 04:44'. The main window has two tabs: 'user1' and 'g2'. The 'user1' tab is active, showing a query editor with the following SQL code:

```

36
37 # On view
38 • select * from view1 where program = "p1";
39

```

Below the query editor is a 'Result Grid' table:

#	resident_ic	first_name	email_id	program	branch
1	20000075	residentfn76	residentemail76@iltgn.ac.in	p1	b1
2	20000105	residentfn106	residentemail106@iltgn.ac.in	p1	b1
3	20000150	residentfn151	residentemail151@iltgn.ac.in	p1	b1
4	20000180	residentfn181	residentemail181@iltgn.ac.in	p1	b1
5	20000066	residentfn67	residentemail67@iltgn.ac.in	p1	b2
6	20000126	residentfn127	residentemail127@iltgn.ac.in	p1	b2
7	20000201	residentfn202	residentemail202@iltgn.ac.in	p1	b2
8	20000261	residentfn262	residentemail262@iltgn.ac.in	p1	b2

The bottom status bar indicates 'Query Completed'.

UPDATE operation:

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** Activities, Mar 1 04:45, MySQL Workbench.
- Schemas:** Schemas tab selected, showing the schema `hostelmng`.
- Query Editor:** Two tabs: Query 1 (containing the failed UPDATE statement) and g2.
- Action Output:** Shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
1	04:41:05	use hostelmng	0 row(s) affected	0.00041 sec
2	04:41:09	select * from RESIDENT	151 row(s) returned	0.0013 sec / 0.0002...
3	04:42:25	SET SQL_SAFE_UP...	0 row(s) affected	0.00035 sec
4	04:42:29	update RESIDENT ...	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'RESIDENT'	0.00038 sec
5	04:43:23	delete from RESID...	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'RESIDENT'	0.00039 sec
6	04:44:25	select * from view1	28 row(s) returned	0.0012 sec / 0.0000...
7	04:45:04	update view1 set o...	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'	0.00030 sec

- Object Info:** No object selected.

DELETE operation:

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** Activities, Mar 1 04:45, MySQL Workbench.
- Schemas:** Schemas tab selected, showing the schema `hostelmng`.
- Query Editor:** Two tabs: Query 1 (containing the failed DELETE statement) and g2.
- Action Output:** Shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
2	04:41:09	select * from RESIDENT	151 row(s) returned	0.0013 sec / 0.0002...
3	04:42:25	SET SQL_SAFE_UP...	0 row(s) affected	0.00035 sec
4	04:42:29	update RESIDENT ...	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'RESIDENT'	0.00038 sec
5	04:43:23	delete from RESID...	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'RESIDENT'	0.00039 sec
6	04:44:25	select * from view1	28 row(s) returned	0.0012 sec / 0.0000...
7	04:45:04	update view1 set o...	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'	0.00030 sec
8	04:45:42	delete from view1 ...	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'	0.00044 sec

- Object Info:** No object selected.

As predicted, our expectations meet our observations.

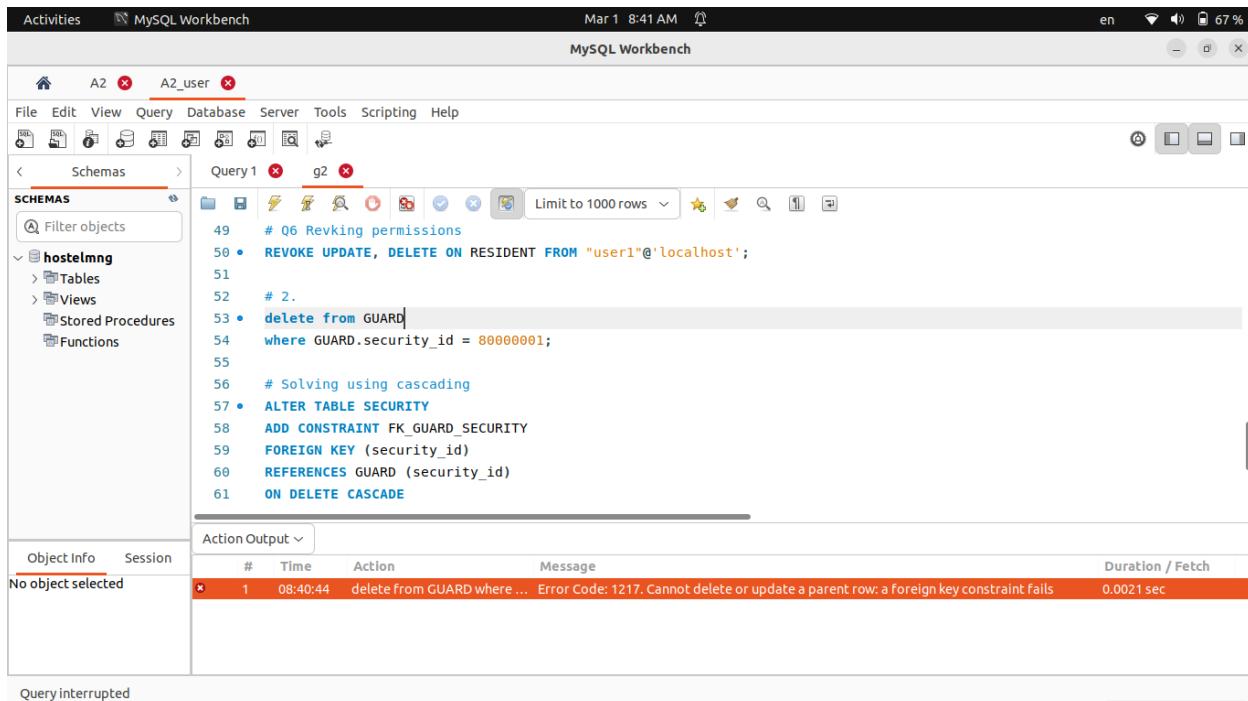
2. Mention the situation that violates the referential integrity, show the updates in the table, and how we can solve such problems.

If a GUARD record is deleted, there may be GUARD_PHONE records that reference the deleted GUARD record through their security_id foreign key. This can result in orphaned records that do not have a corresponding GUARD record.

Here's an example code to add cascading delete and update actions to the foreign key constraints:

```
delete from GUARD where GUARD.security_id = 80000001;
```

Let's say we want to delete a guard record whose security_id is 80000001, but if we execute it, we get the following error message.



The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'hostelmng' schema, the 'Tables' node is selected. In the main pane, a query editor window titled 'Query 1' contains the following SQL code:

```
49 # Q6 Revoking permissions
50 • REVOKE UPDATE, DELETE ON RESIDENT FROM "user1"@"localhost";
51
52 # 2.
53 • delete from GUARD
54 where GUARD.security_id = 80000001;
55
56 # Solving using cascading
57 • ALTER TABLE SECURITY
58 ADD CONSTRAINT FK_GUARD_SECURITY
59 FOREIGN KEY (security_id)
60 REFERENCES GUARD (security_id)
61 ON DELETE CASCADE
```

Below the query editor, the 'Action Output' pane displays the results of the execution. A single row is shown in red, indicating an error:

#	Time	Action	Message	Duration / Fetch
x	1	08:40:44	delete from GUARD where ... Error Code: 1217. Cannot delete or update a parent row: a foreign key constraint fails	0.0021 sec

A status message at the bottom of the interface reads 'Query interrupted'.

This means the record whose security_id is 80000001 will not be deleted, and can be viewed below.

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Schemas hostelmng Tables GUARD GUARD_PHONE RESIDENT Views Stored Procedures Functions

Object Info Session Table: GUARD Columns: security_id

Query 1 g2 GUARD

Result Grid

#	security_id	first_name	middle_name	last_name
1	80000001	guardFName1	guardMName1	guardLName1
2	80000002	guardFName2	guardMName2	guardLName2
3	80000003	guardFName3	guardMName3	guardLName3
4	80000004	guardFName4	guardMName4	guardLName4
5	80000005	guardFName5	guardMName5	guardLName5
6	80000006	guardFName6	guardMName6	guardLName6
7	80000007	guardFName7	guardMName7	guardLName7
8	80000008	guardFName8	guardMName8	guardLName8
9	80000009	guardFName9	guardMName9	guardLName9

GUARD 1

Query Completed

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Schemas hostelmng Tables GUARD GUARD_PHONE RESIDENT Views Stored Procedures Functions

Object Info Session Table: GUARD_PHONE Columns: phone_no decimal(10,0)

Query 1 g2 GUARD_PHONE

Result Grid

#	phone_no	security_id
1	7000000001	80000001
2	7000000002	80000001
3	7000000003	80000001
4	7000000004	80000001
5	7000000005	80000002
6	7000000006	80000003
7	7000000007	80000003
8	7000000008	80000003
9	7000000009	80000003

GUARD_PHONE 1

Query Completed

To avoid these referential integrity issues, we can set up the foreign key constraints with cascading delete and update actions. This means that when a parent record is deleted or updated, the corresponding child records will also be deleted or updated automatically to maintain referential integrity.

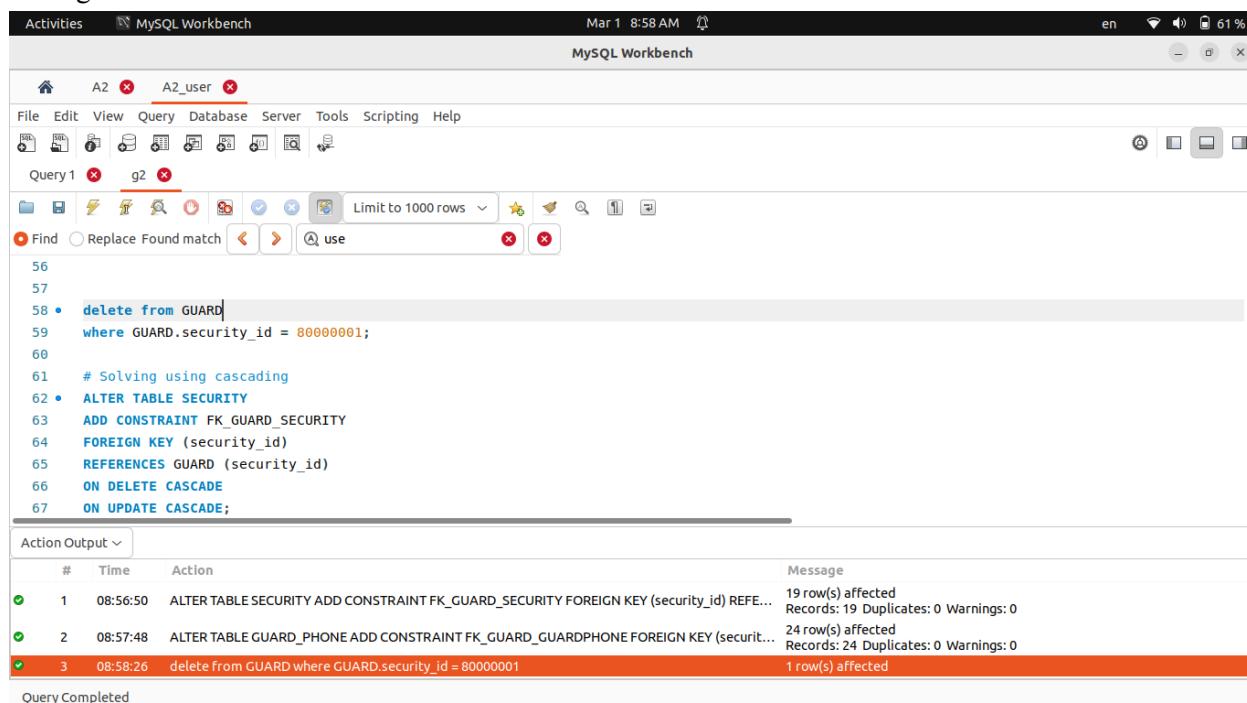
```

# Solving using cascading
ALTER TABLE SECURITY
ADD CONSTRAINT FK_GUARD_SECURITY
FOREIGN KEY (security_id)
REFERENCES GUARD (security_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE GUARD_PHONE
ADD CONSTRAINT FK_GUARD_GUARDPHONE
FOREIGN KEY (security_id)
REFERENCES GUARD (security_id)
ON DELETE CASCADE
ON UPDATE CASCADE;

```

Now, after executing the above two queries, and executes the delete GUARD record, we get the following message.



The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** Displays the executed SQL code. The first query is a delete statement from the GUARD table where security_id = 80000001. The second query is the ALTER TABLE SECURITY query shown in the code block. The third query is the ALTER TABLE GUARD_PHONE query shown in the code block.
- Action Output:** Shows the results of the executed statements:

#	Time	Action	Message
1	08:56:50	ALTER TABLE SECURITY ADD CONSTRAINT FK_GUARD_SECURITY FOREIGN KEY (security_id) REFERENCES GUARD (security_id) ON DELETE CASCADE ON UPDATE CASCADE;	19 row(s) affected Records: 19 Duplicates: 0 Warnings: 0
2	08:57:48	ALTER TABLE GUARD_PHONE ADD CONSTRAINT FK_GUARD_GUARDPHONE FOREIGN KEY (security_id) REFERENCES GUARD (security_id) ON DELETE CASCADE ON UPDATE CASCADE;	24 row(s) affected Records: 24 Duplicates: 0 Warnings: 0
3	08:58:26	delete from GUARD where GUARD.security_id = 80000001	1 row(s) affected
- Status Bar:** Shows the date and time (Mar 1 8:58 AM), system status (en), and battery level (61%).

Activities MySQL Workbench Mar 1 8:59 AM en 59 %

MySQL Workbench

A2 A2_user

File Edit View Query Database Server Tools Scripting Help

Schemas hostelmng Tables GUARD GUARD_PHONE RESIDENT SECURITY Views Stored Procedures Functions

Object Info Session

Table: GUARD

Columns: security_id

Query 1 g2 GUARD

1 • `SELECT * FROM hostelmng.GUARD;`

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

#	security_id	first_name	middle_name	last_name
1	80000002	guardFName2	guardMName2	guardLName2
2	80000003	guardFName3	guardMName3	guardLName3
3	80000004	guardFName4	guardMName4	guardLName4
4	80000005	guardFName5	guardMName5	guardLName5
5	80000006	guardFName6	guardMName6	guardLName6
6	80000007	guardFName7	guardMName7	guardLName7
7	80000008	guardFName8	guardMName8	guardLName8
8	80000009	guardFName9	guardMName9	guardLName9
9	80000010	guardFName10	guardMName10	guardLName10

GUARD 1

Query Completed

Activities MySQL Workbench Mar 1 8:59 AM en 59 %

MySQL Workbench

A2 A2_user

File Edit View Query Database Server Tools Scripting Help

Schemas hostelmng Tables GUARD GUARD_PHONE RESIDENT SECURITY Views Stored Procedures Functions

Object Info Session

Table: GUARD_PHONE

Columns: phone_no decimal(10,0)

Query 1 g2 GUARD_PHONE

1 • `SELECT * FROM hostelmng.GUARD_PHONE;`

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

#	phone_no	security_id
1	700000005	80000002
2	700000006	80000003
3	700000007	80000003
4	700000008	80000003
5	700000009	80000003
6	700000010	80000004
7	700000011	80000004
8	700000012	80000005
9	700000013	80000005

GUARD_PHONE 1

Query Completed

As you can see from the above screenshots, the GUARD record whose security_id is 80000001 is deleted in GUARD and GUARD_PHONE.

3.3 Responsibility of G1 & G2:

The five nested sql queries f1,f2,f3,f4 and f5 are described below.

1. Two queries must throw an error due to a violation of constraints specified by G1.

- f1 and f2 are the two operations that satisfy these conditions.

- **f1:**

```
DELETE FROM RESIDENT
WHERE gender="F" and on_campus=1 and resident_id IN (
SELECT ENROLLED_IN.resident_id from ENROLLED_IN where
ENROLLED_IN.branch = "b1" );
```

In the below figure, the delete query, (where the cursor is placed) throws an error due to the violation of the foreign key constraint specified by G1.

The screenshot shows the MySQL Workbench interface with a query editor and object browser. The object browser displays the 'CARETAKER' table under the 'ALLOCATION' schema. The query editor contains the following SQL code:

```
1 * use hostelman;
2
3 * DELETE FROM RESIDENT
4 WHERE gender="F" and on_campus=1 and resident_id IN (
5   SELECT ENROLLED_IN.resident_id
6   from ENROLLED_IN
7   where ENROLLED_IN.branch = "b1"
8 );
9
10 * update ALLOCATION
11   SET exit_date = STR_TO_DATE('21,5,2013')
12 WHERE hostel_name="a" and resident_id IN (
13   select resident_id from
14   RESIDENT
15   where gender = "F"
```

The cursor is positioned at the start of the WHERE clause of the first query. In the 'Object Output' pane, there is one entry:

#	Time	Action	Message
1	09:54:51	DELETE FROM RESIDENT...	Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('hostelman', 'ALLOCATION')

A status message 'Query interrupted' is visible at the bottom left.

Relational algebra:

f1

RESIDENT - (gender = 'F' and (RESIDENT) \bowtie resident_id (Branch= b1 (ENROLLED_IN))
on_campus = 1)

• f2 :

```
update ALLOCATION
SET exit_date = STR_TO_DATE('21,5,2013')
WHERE hostel_name="a" and resident_id IN (
    select resident_id from
    RESIDENT
    where gender = "F"
);
```

In the below figure, the update query, (where the cursor is placed) throws an error due to the violation of the foreign key constraint on the exit-date attributes given by G1.

The screenshot shows the MySQL Workbench interface with the following details:

- Activities:** Local instance 3306
- File Edit View Query Database Server Tools Scripting Help**
- Schemas:** Filter objects, Tables (ACADEMIC_PERIOD, ALLOCATION, CARETAKER), Columns (caretaker_id, first_name, middle_name, last_name, gender, office_no, email_id, photo), Indexes.
- SQL Editor:** Contains the following SQL code:

```
1 • use hostellmng;
2
3 • DELETE FROM RESIDENT
4 ○ WHERE gender="F" and on_campus=1 and resident_id IN (
5     SELECT ENROLLED_IN.resident_id
6     from ENROLLED_IN
7     where ENROLLED_IN.branch = "b1"
8 );
9
10 • update ALLOCATION
11 SET exit_date = STR_TO_DATE('21,5,2013')
12 ○ WHERE hostel_name="a" and resident_id IN (
13     select resident_id from
14     RESIDENT
15     where gender = "F"
```
- Action Output:** Shows a single row with the message: "Error Code: 1582: Incorrect parameter count in the call to native function 'STR_TO_DATE'"
- Object Info:** Table: CARETAKER, Columns: caretaker_id, first_name, middle_name, last_name, gender, office_no, email_id, photo.

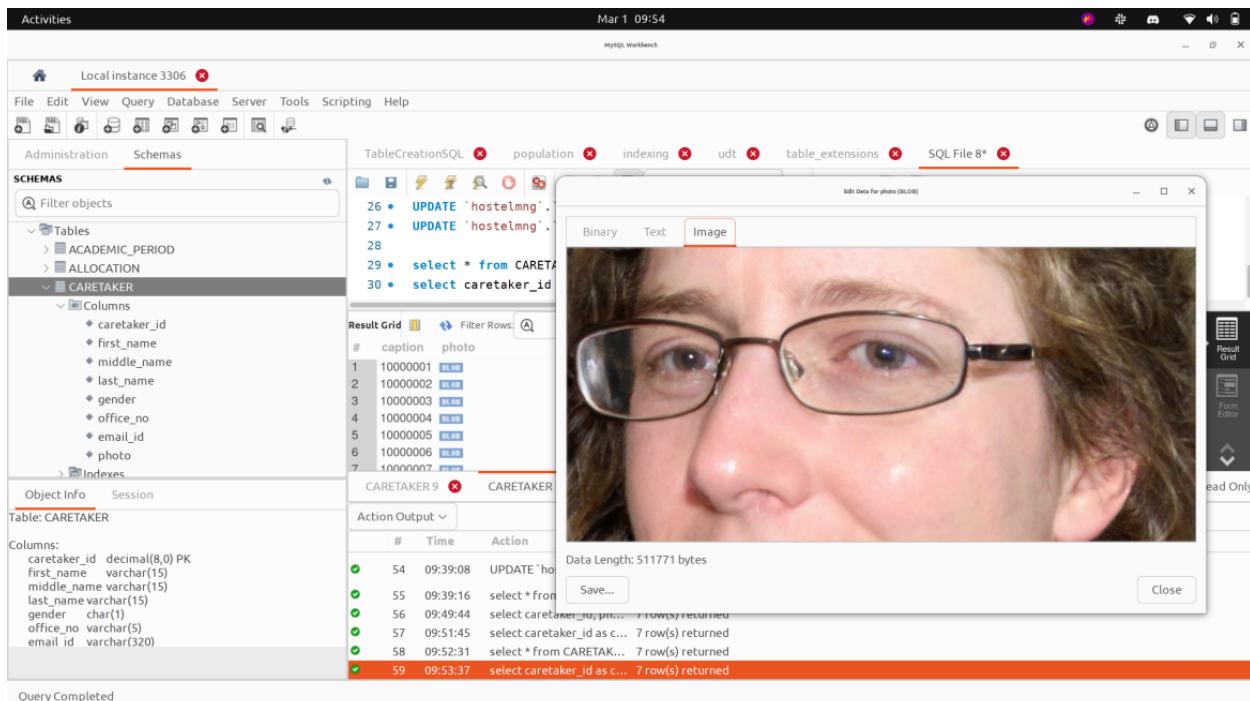
2. One function should involve storing an image and a caption in the database.

- f3 :

```
# Adding the image to CARETAKER ALTER table CARETAKER ADD COLUMN photo LONGBLOB;

UPDATE hostelmng.CARETAKER SET photo = LOAD_FILE("images/1.jpg") WHERE (caretaker_id = '10000001');
UPDATE hostelmng.CARETAKER SET photo = LOAD_FILE("images/2.jpg") WHERE (caretaker_id = '10000002');
UPDATE hostelmng.CARETAKER SET photo = LOAD_FILE("images/3.jpg") WHERE (caretaker_id = '10000003');
UPDATE hostelmng.CARETAKER SET photo = LOAD_FILE("images/4.jpg") WHERE (caretaker_id = '10000004');
UPDATE hostelmng.CARETAKER SET photo = LOAD_FILE("images/5.jpg") WHERE (caretaker_id = '10000005');
UPDATE hostelmng.CARETAKER SET photo = LOAD_FILE("images/6.jpg") WHERE (caretaker_id = '10000006');
UPDATE hostelmng.CARETAKER SET photo = LOAD_FILE("images/7.jpg") WHERE (caretaker_id = '10000007');
```

In this query, we are storing an image ‘photo’ in the caretaker entity. We are also using one of the attributes as its caption. The query and the output is shown in the below figure.



Relational algebra:

f3
 $\ell_x(\text{caption}, \text{photo}) \cap_{\text{caretaker_id}, \text{photo}} (\text{caretaker})$

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.

- f4 :

```
# GET details about all the allocated residents and their current room details
```

```
SELECT *
FROM (
    SELECT *
    FROM ALLOCATION
    NATURAL JOIN RESIDENT
) AS a
NATURAL JOIN ROOM;
```

#	room_no	hostel_name	resident_ic	semester	year	entry_date	exit_date	payment_status	due_amount	due_status	payment_amount	first_name	middle_name	last_name	gender	office_no	email_id	photo
1	101	a	20000000	1	2020	2020-09-01	2020-1...	5000	0	0	5000	resident1	resident1	resident1	resident1	resident1	resident1	resident1
2	101	a	20000000	2	2020	2020-01-02	2020-0...	5000	0	0	5000	resident1	resident1	resident1	resident1	resident1	resident1	resident1
3	101	a	20000000	3	2021	2021-09-01	2021-1...	5000	0	0	5000	resident1	resident1	resident1	resident1	resident1	resident1	resident1
4	101	a	20000000	4	2021	2021-01-02	2021-0...	5000	0	0	5000	resident1	resident1	resident1	resident1	resident1	resident1	resident1
5	101	a	20000000	5	2022	2022-11-30	2022-1...	5000	0	0	5000	resident1	resident1	resident1	resident1	resident1	resident1	resident1
6	102	a	20000001	1	2022	2022-11-30	2022-1...	5000	0	0	5000	resident2	resident2	resident2	resident2	resident2	resident2	resident2

Relational algebra:

f4

$$\Pi_{a.all} \left(\ell_a \left(\Pi_{\substack{\text{ALLOCATION.all} \cup \\ \text{RESIDENT.all}}} \right) \cap \text{Room} \right)$$

$t.all \rightarrow$ all attribute in table t.

- f5 :

GET the first name email id and the enrolled program for a give branch

```

SELECT s.first_name, s.email_id, e.program
FROM RESIDENTS
LEFTOUTERJOIN(
    SELECT * FROM ENROLLED_IN
    WHERE branch = 'b1'
) e ON s.resident_id = e.resident_id
where e.program IS NOT NULL;

```

Activities Local instance 3306 Mar 1 11:00 MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Administration Schemas

SCHEMAS Filter objects Tables ACADEMIC_PERIOD ALLOCATION CARETAKER Indexes Object Info Session

Table: CARETAKER

Columns:

- caretaker_id decimal(8,0) PK
- first_name varchar(15)
- middle_name varchar(15)
- last_name varchar(15)
- gender char(1)
- office_no varchar(5)
- email_id varchar(320)

```

40
41 # GET the first name email id and the enrolled program for a give branch
42 • SELECT s.first_name, s.email_id, e.program
43 FROM RESIDENT s
44 LEFT OUTER JOIN (
45     SELECT * FROM ENROLLED_IN
46     WHERE branch = 'b1'
47 ) e ON s.resident_id = e.resident_id
48 where e.program IS NOT NULL;
49

```

Result Grid Export: Wrap Cell Content: Result Grid Form Editor

#	first_name	email_id	program
1	residentfn76	residentemail76@iitgn.ac.in	p1
2	residentfn106	residentemail106@iitgn.ac.in	p1
3	residentfn151	residentemail151@iitgn.ac.in	p1
4	residentfn181	residentemail181@iitgn.ac.in	p1
5	residentfn26	residentemail26@iitgn.ac.in	p2
6	residentfn41	residentemail41@iitgn.ac.in	p2

Action Output

#	Time	Action	Message
1	10:58:57	SELECT * FROM (SEL...	1000 row(s) returned
2	10:59:43	SELECT s.first_name, s...	16 row(s) returned

Query Completed

Relational algebra:

$$\begin{array}{c}
 \text{f1} \quad \text{f2} \\
 \text{f3} \quad \text{f4} \quad \text{f5}
 \end{array}
 \text{f1: } \Pi_{s.\text{first_name}, s.\text{email_id}} \left(\rho_{e.\text{program} \neq \text{NULL}} \left(\rho_s(\text{RESIDENT}) \times_{s.\text{resident} = e.\text{resident_id}} \left(\rho_e (\sigma_{\text{branch} = 'b1'} (\text{ENROLLED_IN})) \right) \right) \right)$$

- f1 satisfies specification 1
- f2 satisfies specification 1
- f3 satisfies specification 2
- f4 and f5 together satisfy specification 3

Contributions :

Everyone in the group has equal contribution towards the project. Contributions of the people in each sub-group (G1 and G2) are mixed (several students are involved in each activity).

Group G2 Contribution:

- Bommisetty Siva Sai - 20110041
- Voorugonda Rajesh - 20110231
- Bhavini Korthi - 20110039
- Balu Karthik Ram - 20110036
- Kareena Beniwal - 20110095
- Rishabh Patidar - 20110165

Group G1 Contribution:

- Venkata SriMan Narayana Malli - 20110224
- Gali Sunny - 20110067
- S Sri Manish Goud - 20110174
- Talla Gnana Sai - 20110210
- Hamsini Kadali - 20110087
- Chaitanya Rao - 20110163