CS 432: Databases

Assignment 2: DESIGNING THE DBMS

Group - 6
GREP

Question 1

Screenshots for created tables and populated data

1. Trainers Entity set

```
create table Trainers (
email_id varchar(50) primary key,
fee int default null,
name varchar(50) not null,
age int not null,
gender enum('Male', 'Female', 'Others') not null
);
```

Trainers Populated data

email_id	fee	name	age	gender
trainer1@gmail.com	2000	Rajesh Kumar	35	Male
trainer10@gmail.com	3200	Meera Shah	26	Female
trainer2@gmail.com	2500	Anjali Sharma	28	Female
trainer3@gmail.com	1800	Vikram Singh	42	Male
trainer4@gmail.com	3000	Sneha Patel	31	F
trainer5@gmail.com	2200	Gaurav Chauhan	29	Male
trainer6@gmail.com	2700	Priya Gupta	27	Female
trainer7@gmail.com	1900	Ravi Mishra	36	Male
trainer8@gmail.com	2800	Neha Singh	33	Female

Female

2. Beneficiary Entity set

```
create table Beneficiary (
aadhar_id bigint primary key,
name varchar(50) not null,
date_of_birth date not null,
gender enum('Male', 'Female', 'Others') not null,
marital_status varchar(10) not null,
education varchar(50) not null,
photo longblob default null,
employed varchar(20) default null,
photo_caption VARCHAR(255) DEFAULT NULL
);
```

Beneficiary Populated data

aadhar_id	name	date_of_birth	gender	marital_status	education	photo	employed	photo_caption
123450987651	Sanjay Kumar	1979-01-31	Male	Married	Master of Arts	BLOB	Yes	Photo of Sanjay Kumar
123456789011	Asha Sharma	1980-05-12	Female	Married	Master of Business Administration	BLOB	Yes	Photo of Asha Sharma
234567890121	Rahul Singh	1995-08-02	Male	Unmarried	Bachelor of Engineering	BLOB	Yes	Photo of Rahul Singh
345678901231	Priya Patel	1988-12-25	Female	Married	Doctor of Medicine	BLOB	No	NULL
456789012341	Rajesh Kumar	1976-09-18	Male	Married	Bachelor of Science	BLOB	Yes	Photo of Rajesh Kumar
567890123451	Sneha Gupta	1992-04-01	Female	Unmarried	Master of Computer Applications	BLOB	No	NULL
678901234561	Vikram Singh	1985-06-21	Male	Married	Bachelor of Commerce	BLOB	Yes	Photo of Vikram Singh
789012345671	Anjali Reddy	1990-03-15	Female	Unmarried	Bachelor of Arts	BLOB	No	NULL
890123456781	Amit Sharma	1983-11-08	Male	Married	Master of Science	BLOB	Yes	Photo of Amit Sharma

3. Teams Entity set

```
create table Teams (
employee_id varchar(20) primary key,
name varchar(50) not null,
email_id varchar(50) unique,
salary int not null,
position varchar(50) not null,
year_of_joining date not null,
year_of_leaving date default null,
reason_of_leaving text default null
);
```

Teams Populated data

employee_id	name	email_id	salary	position	year_of_joining	year_of_leaving	reason_of_leaving
E001	Ravi Kumar	ravi.kumar@example.com	50000	Manager	2015-01-01	NULL	NULL
E002	Priya Sharma	priya.sharma@example.com	40000	Developer	2016-02-01	NULL	NULL
E003	Amit Singh	amit.singh@example.com	45000	Developer	2017-03-01	NULL	NULL
E004	Neha Gupta	neha.gupta@example.com	55000	Designer	2018-04-01	NULL	NULL
E005	Rajesh Khanna	rajesh.khanna@example.com	60000	Manager	2019-05-01	NULL	NULL
E006	Anjali Verma	anjali.verma@example.com	35000	Developer	2020-06-01	NULL	NULL
E007	Suresh Menon	suresh.menon@example.com	45000	Developer	2021-07-01	NULL	NULL
E008	Nisha Rawat	nisha.rawat@example.com	50000	Designer	2014-08-01	2022-08-31	Personal reasons
E009	Rahul Singhania	rahul.singhania@example.com	60000	Manager	Manager	NULL	NULL

4. Projects Entity set

```
create table Projects (
event_name varchar(50),
start_date date,
primary key (event_name, start_date),
types varchar(50) not null,
budget int not null,
no_of_participants int not null,
duration int not null,
collection int default null,
total_expense int not null
);
```

Projects Populated data

event_name	start_date	types	budget	no_of_participants	duration	collection	total_expense
Project A	2022-01-01	Conference	500000	100	5	600000	400000
Project B	2022-02-01	Workshop	300000	50	3	NULL	200000
Project C	2022-03-01	Seminar	200000	30	2	NULL	150000
Project D	2022-04-01	Exhibition	800000	200	7	900000	700000
Project E	2022-05-01	Hackathon	400000	80	4	NULL	300000
Project F	2022-06-01	Training	600000	120	6	700000	500000
Project G	2022-07-01	Competition	500000	50	2	NULL	400000
Project H	2022-08-01	Expo	1000000	300	10	1200000	900000
Project I	2022-09-01	Webinar	100000	200	1	NULL	80000

5. ProjectPhotos Entity set

```
CREATE TABLE ProjectPhotos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  project_event_name VARCHAR(50),
  project_start_date DATE,
  photo_url longblob NOT NULL,
  caption VARCHAR(255) DEFAULT NULL,
  FOREIGN KEY (project_event_name, project_start_date)
    REFERENCES Projects(event_name, start_date)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

ProjectPhotos Populated data

id	project_event_name	project_start_date	photo_url	caption
1	Project A	2022-01-01	BLOB	Conference photo 1
2	Project A	2022-01-01	BLOB	Conference photo 2
3	Project B	2022-02-01	BLOB	Workshop photo 1
4	Project C	2022-03-01	BLOB	NULL
5	Project D	2022-04-01	BLOB	Exhibition photo 1
6	Project E	2022-05-01	BLOB	NULL
7	Project F	2022-06-01	BLOB	Training photo 1
8	Project G	2022-07-01	BLOB	NULL
9	Project H	2022-08-01	BLOB	Expo photo 1

6. Volunteers Entity set

```
create table Volunteers (
email_id varchar(50) primary key,
name varchar(50) not null,
phone_number bigint not null,
date_of_birth date default null,
gender enum('Male', 'Female', 'Others') not null
);
```

Volunteers Populated data

email_id	name	phone_number	date_of_birth	gender
amit.kumar@example.com	Amit Kumar	9876543214	1991-02-10	Male
avinash.singh@example.com	Avinash Singh	9876543218	1992-03-22	Male
divya.mishra@example.com	Divya Mishra	9876543215	1993-07-05	Female
jane.doe@example.com	Jane Doe	9876543211	1992-08-25	Female
john.doe@example.com	John Doe	9876543211	1990-05-15	Male
neha.patel@example.com	Neha Patel	9876543219	1997-06-08	Female
pooja.shah@example.com	Pooja Shah	9876543217	1996-12-18	Female
priya.rai@example.com	Priya Rai	9876543213	1995-11-20	Female
rahul.gupta@example.com	Rahul Gupta	9876543216	1994-09-12	Male

7. Funding Entity set

```
create table Funding (
email_id varchar(50) primary key,
amount int not null,
funder_name varchar(50) not null,
date date not null
);
```

Funding populated data

email_id	amount	funder_name	date
abc@gmail.com	50000	Rohit Sharma	2022-02-01
abc1@gmail.com	40000	Manisha Singh	2022-02-25
def@gmail.com	25000	Neha Gupta	2022-02-03
ghi@gmail.com	100000	Rajesh Singh	2022-02-06
jkl@gmail.com	75000	Priya Patel	2022-02-10
mno@gmail.com	30000	Siddharth Reddy	2022-02-12
pqr@gmail.com	150000	Amit Kumar	2022-02-15
stu@gmail.com	5000	Shreya Gupta	2022-02-17
vwx@gmail.com	80000	Anjali Sharma	2022-02-20

8. Venue Entity set

```
create table Venue (
venue_id varchar(20) primary key,
place varchar(100) not null,
pincode int not null,
district varchar (50) not null,
state varchar (50) not null
);
```

Venue Populated data

venue_id	place	pincode	district	state
VEN001	Maharashtra Nagar	400010	Mumbai	Maharashtra
VEN002	Shalimar Bagh	110088	North Delhi	Delhi
VEN003	Park Street	700016	Kolkata	West Bengal
VEN004	Whitefield	560066	Bangalore	Karnataka
VEN005	Banjara Hills	[illegible]	Hyderabad	Telangana
VEN006	Gomti Nagar	226010	Lucknow	Uttar Pradesh
VEN007	Alwarpet	600018	Chennai	Tamil Nadu
VEN008	Bodakdev	380054	Ahmedabad	Gujarat
VEN009	Race Course	390007	Vadodara	Gujarat

9. VillageProfile Entity set

```
create table VillageProfile (
pincode int primary key,
name varchar(50) not null,
no_of_beneficiaries int not null,
no_of_primary_health_center int not null,
no_of_primary_school int not null,
transport varchar(100) default null,
infrastructure varchar(100) default null,
major_occupation varchar(100) default null,
technical_literacy text default null,
year date not null
);
```

VillageProfile Populated data

pincode	name	no_of_beneficiaries	no_of_primary_health_center	no_of_primary_school	transport	infrastructure	major_occupation	technical_litera
380001	Ambawadi	2500	2	3	Bus, Train	Electricity, Water, Roads	Agriculture	Basic Computer
380002	Vastrapur	3500	1	4	Bus, Auto	Electricity, Water, Drainage	Education	Basic Computer
380003	Navrangpura	4000	3	2	Bus, Train, Auto	Electricity, Water, Roads	Manufacturing	Advanced Comp
380004	Naranpura	3000	2	3	Bus, Auto	Electricity, Water, Roads	Service	Basic Computer
380005	Satellite	5000	1	4	Bus, Auto	Electricity, Water, Roads	Software Development	Advanced Comp
382330	Gandhinagar	4500	3	2	Bus, Train	Electricity, Water, Roads	Agriculture	Basic Computer
382340	Kalol	2000	1	4	Bus, Auto	Electricity, Water, Drainage	Education	Basic Computer
382345	Adalaj	3000	2	3	Bus, Auto	Electricity, Water, Roads	Service	Basic Computer

10. Goods Entity set

```
create table Goods (
event_name varchar(50),
start_date date,
item_name varchar(50),
quantity int,
amount int,
primary key (event_name, start_date, item_name, quantity, amount),
foreign key (event_name, start_date)
references Projects(event_name, start_date) on delete cascade on update cascade
);
```

Goods Populated data

event_name	start_date	item_name	quantity	amount
Project A	2022-01-01	Laptop	10	100000
Project A	2022-01-01	Projector	2	50000
Project B	2022-02-01	Marker	50	1000
Project C	2022-03-01	Notebook	30	500
Project D	2022-04-01	Painting	5	1000000
Project E	2022-05-01	Raspberry Pi	20	20000
Project F	2022-06-01	Whiteboard	2	30000
Project G	2022-07-01	Chess Set	5	5000
Project H	2022-08-01	Microphone	10	50000

11. ProjectExpense Entity set

```
create table ProjectExpense (
event_name varchar(50),
start_date date,
description varchar(255),
amount int,
primary key (event_name, start_date, description, amount),
foreign key (event_name, start_date)
references Projects(event_name, start_date) on delete cascade on update cascade
);
```

ProjectExpense Populated data

event_name	start_date	description	amount
Project A	2022-01-01	Catering expenses	150000
Project A	2022-01-01	Speaker fees	200000
Project B	2022-02-01	Equipment rental	50000
Project B	2022-02-01	Venue rental	1(100000
Project C	2022-03-01	Printing and design	80000
Project C	2022-03-01	Speaker fees	70000
Project D	2022-04-01	Shipping expenses	150000
Project E	2022-05-01	Prize money	100000
Project F	2022-06-01	Speaker fees	300000

12. TrainerPhone Entity set

```
create table TrainerPhoneEntity (
email_id varchar(50),
phone_number bigint not null,
primary key (email_id, phone_number),
foreign key (email_id) references Trainers(email_id) on delete cascade on update cascade
);
```

TrainerPhone Entity Populated data

email_id	phone_number
trainer1@gmail.com	8765432109
trainer1@gmail.com	9876543210
trainer10@gmail.com	1234567890
trainer2@gmail.com	6543210987
trainer2@gmail.com	7654321098
trainer3@gmail.com	5432109876
trainer4@gmail.com	4321098765
trainer5@gmail.com	3210987654
trainer6@gmail.com	2109876543

13. BeneficiaryPhoneEntity set

```
create table BeneficiaryPhoneEntity (
aadhar_id bigint not null,
phone_number bigint not null,
primary key (aadhar_id, phone_number),
foreign key (aadhar_id) references Beneficiary(aadhar_id) on delete cascade on update cascade
);
```

BeneficiaryPhoneEntity Populated data

aadhar_id	phone_number
123450987654	1234567890
123456789012	9876543210
234567890123	8765432109
345678901234	7654321098
456789012345	6543210987
567890123456	5432109876
678901234567	4321098765
789012345678	3210987654
890123456789	2109876543

14. TeamPhone Relation set

```
create table TeamPhone (
employee_id varchar(20) not null,
phone_number bigint not null,
location text default null,
primary key (employee_id, phone_number),
foreign key (employee_id) references Teams(employee_id) on delete cascade on update cascade
);
```

TeamPhone Populated data

employee_id	phone_number	location
E001	9876543210	9876543210
E001	9898989898	Mumbai
E002	8765432109	Kolkata
E003	7894561230	Bangalore
E003	9865327410	Chennai
E004	9876543210	Pune
E005	8765432109	Hyderabad
E006	7894561230	Mumbai
E007	9865327410	Chennai

15. Organize relation set

```
create table Organize (
employee_id varchar(20),
event_name varchar(50),
start_date date,
role varchar(50) not null,
primary key (employee_id, event_name, start_date),
foreign key(employee_id) references Teams(employee_id) on delete cascade on update cascade,
foreign key(event_name, start_date) references Projects(event_name, start_date) on delete cascade
on update cascade
);
```

Organize Populated data

employee_id	event_name	start_date	role
E001	Project A	2022-01-01	
E001	Project E	2022-05-01	Manager
E001	Project I	2022-09-01	Manager
E002	Project A	2022-01-01	Developer
E002	Project E	2022-05-01	Developer
E002	Project I	2022-09-01	Developer
E003	Project A	2022-01-01	Developer
E003	Project E	2022-05-01	Developer
E003	Project I	2022-09-01	Developer

16. Sponsors relation set

```
create table sponsors(
email_id varchar(50),
event_name varchar(50),
start_date date,
primary key (email_id, event_name, start_date),
foreign key(email_id) references Funding(email_id) on delete cascade on update cascade,
foreign key(event_name, start_date) references Projects(event_name, start_date) on delete cascade
on update cascade
);
```

Sponsors Populated data

email_id	event_name	start_date
abc@gmail.com	Project B	2022-02-01
abc1@gmail.com	Project B	2022-02-01
def@gmail.com	Project B	2022-02-01
ghi@gmail.com	Project B	2022-02-01
jkl@gmail.com	Project B	2022-02-01
mno@gmail.com	Project B	2022-02-01
pqr@gmail.com	Project B	2022-02-01
stu@gmail.com	Project B	2022-02-01
vwx@gmail.com	Project B	2022-02-01

17. Volunteering relation set

```
create table volunteering(
email_id varchar(50),
event_name varchar(50),
start_date date,
primary key (email_id, event_name, start_date),
foreign key(email_id) references Volunteers(email_id) on delete cascade on update cascade,
foreign key(event_name, start_date) references Projects(event_name, start_date) on delete cascade
on update cascade
);
```

Volunteering Populated data

email_id	event_name	start_date
amit.kumar@example.com	Project A	2022-01-01
john.doe@example.com	Project A	2022-01-01
priya.rai@example.com	Project A	2022-01-01
jane.doe@example.com	Project B	2022-02-01
pooja.shah@example.com	Project B	2022-02-01
priya.rai@example.com	Project B	2022-02-01
jane.doe@example.com	Project C	2022-03-01
rahul.gupta@example.com	Project C	2022-03-01
john.doe@example.com	Project D	2022-04-01

18. HeldAt relation set

```
create table HeldAt(
venue_id varchar(20),
event_name varchar(50),
start_date date,
primary key (venue_id, event_name, start_date),
foreign key(venue_id) references Venue(venue_id) on delete cascade on update cascade,
foreign key(event_name, start_date) references Projects(event_name, start_date) on delete cascade
on update cascade
);
```

HeldAt Populated data

venue_id	event_name	start_date
VEN001	Project A	2022-01-01
VEN002	Project B	2022-02-01
VEN003	Project C	2022-03-01
VEN004	Project D	2022-04-01
VEN005	Project E	2022-05-01
VEN006	Project F	2022-06-01
VEN007	Project G	2022-07-01
VEN008	Project H	2022-08-01
VEN009	Project I	2022-09-01

19. Trains relation set

```
create table trains(
email_id varchar(50),
event_name varchar(50),
start_date date,
primary key (email_id, event_name, start_date),
foreign key(email_id) references Trainers(email_id) on delete cascade on update cascade,
foreign key(event_name, start_date) references Projects(event_name, start_date) on delete cascade
on update cascade
);
```

Trains Populated data

email_id	event_name	start_date
trainer1@gmail.com	Project A	2022-01-01
trainer2@gmail.com	Project A	2022-01-01
trainer3@gmail.com	Project B	2022-02-01
trainer4@gmail.com	Project B	2022-02-01
trainer5@gmail.com	Project C	2022-03-01
trainer6@gmail.com	Project C	2022-03-01
trainer7@gmail.com	Project D	2022-04-01
trainer8@gmail.com	Project D	2022-04-01
trainer9@gmail.com	Project E	2022-05-01

20. Participants Populated data

```
create table participants(
aadhar_id bigint,
event_name varchar(50),
start_date date,
primary key (aadhar_id, event_name, start_date),
foreign key(aadhar_id) references Beneficiary(aadhar_id) on delete cascade on update cascade,
foreign key(event_name, start_date) references Projects(event_name, start_date) on delete cascade
on update cascade
);
```

Participants Populated data

aadhar_id	event_name	start_date
123456789012	Project A	2022-01-01
234567890123	Project A	2022-01-01
345678901234	Project A	2022-01-01
456789012345	Project B	2022-02-01
567890123456	Project B	2022-02-01
678901234567	Project C	2022-03-01
789012345678	Project D	2022-04-01
890123456789	Project E	2022-05-01
901234567890	Project F	2022-06-01

21. TrainerBeneficiary relation set

```
create table TrainerBeneficiary(
aadhar_id bigint,
email_id varchar(50),
primary key (aadhar_id),
foreign key(aadhar_id) references Beneficiary(aadhar_id) on delete cascade on update cascade,
foreign key(email_id) references Trainers(email_id) on delete cascade
on update cascade
);
```

TrainerBeneficiary Populated data

aadhar_id	email_id
123456789012	trainer1@gmail.com
678901234567	[illegible]
890123456789	trainer3@gmail.com
456789012345	trainer5@gmail.com
123450987654	trainer7@gmail.com
234567890123	trainer9@gmail.com

22. Assessment relation set

```
create table assessment(
aadhar_id bigint,
event_name varchar(50),
start_date date,
Date date NOT NULL,
present_or_absent enum('Present', 'Absent') NOT NULL,
primary key (aadhar_id),
foreign key(aadhar_id) references Beneficiary(aadhar_id) on delete cascade on update cascade,
foreign key(event_name, start_date) references Projects(event_name, start_date) on delete cascade
on update cascade
);
```

Assessment Populated set

aadhar_id	event_name	start_date	Date	present_or_absent
123450987654	Project F	2022-06-01	2022-06-01	Present
123456789012	Project A	2022-01-01	2022-01-01	Present
234567890123	Project A	2022-01-01	2022-01-01	Absent
345678901234	Project B	2022-02-01	2022-02-01	Present
456789012345	Project B	2022-02-01	2022-02-01	Absent
567890123456	Project C	2022-03-01	2022-03-01	Present
678901234567	Project D	2022-04-01	2022-04-01	Present
789012345678	Project D	2022-04-01	2022-04-01	Absent
890123456789	Project E	2022-05-01	2022-05-01	Absent

23. Belongs relation set

```
create table belongs(
aadhar_id bigint,
pincode int,
primary key (aadhar_id),
foreign key(aadhar_id) references Beneficiary(aadhar_id) on delete cascade on update cascade,
foreign key(pincode) references VillageProfile(pincode) on delete cascade
on update cascade
);
```

Belongs Populated data

aadhar_id	pincode
123456789012	380002
345678901234	380002
890123456789	380002
456789012345	380004
234567890123	380005
123450987654	382330
567890123456	382330
789012345678	382340
901234567890	382340

Question 2

Used enum for user defined datatype.
Here gender is user defined as it can take only ‘Male’, ‘Female’ and ‘Other’.

```
create table Volunteers (
email_id varchar(50) primary key,
name varchar(50) not null,
phone_number bigint not null,
date_of_birth date default null,
gender enum('Male', 'Female', 'Others') not null
);
```

Here peresent_or_absent is also user defined datatype as it can take ‘Present’ and ‘Absent’.

```
create table assessment(
aadhar_id bigint,
event_name varchar(50),
start_date date,
Date date NOT NULL,
present_or_absent enum('Present', 'Absent') NOT NULL,
primary key (aadhar_id),
foreign key(aadhar_id) references Beneficiary(aadhar_id) on delete cascade on update cascade,
foreign key(event_name, start_date) references Projects(event_name, start_date) on delete cascade
on update cascade
);
```

Indexing:

Indexing for table Beneficiary:

To perform time analysis so that we can observe results of indexing we generated random data containing 50000 records for Beneficiary table. Python script for generating random data can be found here.

Indexing Query : We have performed indexing on the *'name'* attribute of the '*Beneficiary'* relation.

```
create index bene_name_index on Beneficiary (name);
```

Query	Runtime without Indexing	Average Runtime without Indexing(s)	Runtime with Indexing	Average Runtime with Indexing(s)	Ratio of run time of without indexing to with indexing
`select * from Beneficiary where name = 'Amit';`	0.011 sec / 0.058 sec 0.011 sec / 0.050 sec 0.011 sec / 0.053 sec	0.011	0.0027 sec / 0.019 sec 0.0037 sec / 0.021 sec 0.0029 sec / 0.018 sec	0.0031	3.54
`select * from Beneficiary where name = "Amit" and gender = "Male";`	0.019 sec / 0.055 sec 0.018 sec / 0.048 sec 0.015 sec / 0.051 sec	0.0173	0.0077 sec / 0.017 sec 0.0050 sec / 0.015 sec 0.0049 sec / 0.015 sec	0.0058	2.98

`select * from Beneficiary where name = "Amit" and date_of_birth between '1990-01-01' and '2000-12-31';`	0.067 sec / 0.015 sec 0.073 sec / 0.017 sec 0.066 sec / 0.014 sec	0.0686	0.016 sec / 0.0011 sec 0.015 sec / 0.00077 s... 0.019 sec / 0.0011 sec	0.0166	4.13

From the above observations we can clearly see that queries involving the attribute ‘name’ are processed 3-4 times faster with indexing when compared to same queries performed without indexing.

Indexing for table volunteers:

To perform time analysis so that we can observe results of indexing we generated random data containing 50000 records for volunteers table. Python script for generating random data can be found here.

Indexing Query : We have performed indexing on the *‘name’* attribute of the ‘*volunteers’* relation.

```
CREATE INDEX volunteer_name_index ON Volunteers (name);
```

Query	Runtime without Indexing	Average Runtime without Indexing(s)	Runtime with Indexing	Average Runtime with Indexing(s)	Ratio of run time of without indexing to with indexing
`select * from Volunteers where name = 'Asha';`	0.0094 sec / 0.032 sec 0.012 sec / 0.031 sec 0.0097 sec / 0.030 sec	0.0103	0.0066 sec / 0.017 sec 0.0069 sec / 0.022 sec 0.0059 sec / 0.015 sec	0.00064	1.609
`select * from Volunteers where name = "Asha" and gender = "Female";`	0.033 sec / 0.038 sec 0.020 sec / 0.024 sec 0.024 sec / 0.026 sec	0.0256	0.0061 sec / 0.0071 sec 0.0044 sec / 0.0072 sec 0.0057 sec / 0.0055 sec	0.0054	4.74
`select * from Volunteers where name = "Asha" and date_of_birth between '1990-01-01' and '2000-12-31';`	0.041 sec / 0.0047 sec 0.033 sec / 0.0041 sec 0.037 sec / 0.0038 sec	0.037	0.013 sec / 0.0016 sec 0.014 sec / 0.0016 sec 0.013 sec / 0.0018 sec	0.0133	2.78

From the above observations we can clearly see that queries involving the attribute ‘name’ are processed 2-5 times faster with indexing when compared to same queries performed without indexing.

Table Extensions:

1. Query for creating a new table ‘feamle_beneficiary’ with the same schema as table ‘beneficiary’.

```
create table female_beneficiary like Beneficiary;
```

aadhar_id	name	date_of_bir...	gender	marital_stat...	education	photo	employed	photo_capti...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2. Query for creating a new tabel ‘female_beneficiary’ with same schema as table ‘benficiary’ and populating beneficiaries having gender ‘Female’.

```
create table female_beneficiary as
(select * from Beneficiary where gender='Female');
```

aadhar_id	name	date_of_bir...	gender	marital_stat...	education	photo	employed	photo_caption
123456789012	Asha Sharma	1980-05-12	Female	Married	Master of Business Administration	NULL	Yes	Photo of Asha Sharma
345678901234	Priya Patel	1988-12-25	Female	Married	Doctor of Medicine	NULL	No	NULL
567890123456	Sneha Gupta	1992-04-01	Female	Unmarried	Master of Computer Applications	NULL	No	NULL
789012345678	Anjali Reddy	1990-03-15	Female	Unmarried	Bachelor of Arts	NULL	No	NULL
901234567890	Shalini Verma	1987-07-14	Female	Married	Bachelor of Business Administration	NULL	No	NULL

3. Query for creating table of projects started during summer months from table ‘projects’ with data population.

```
create table summer_projects as
(select * from Projects where month (start_date) in (4,5,6,7));
```

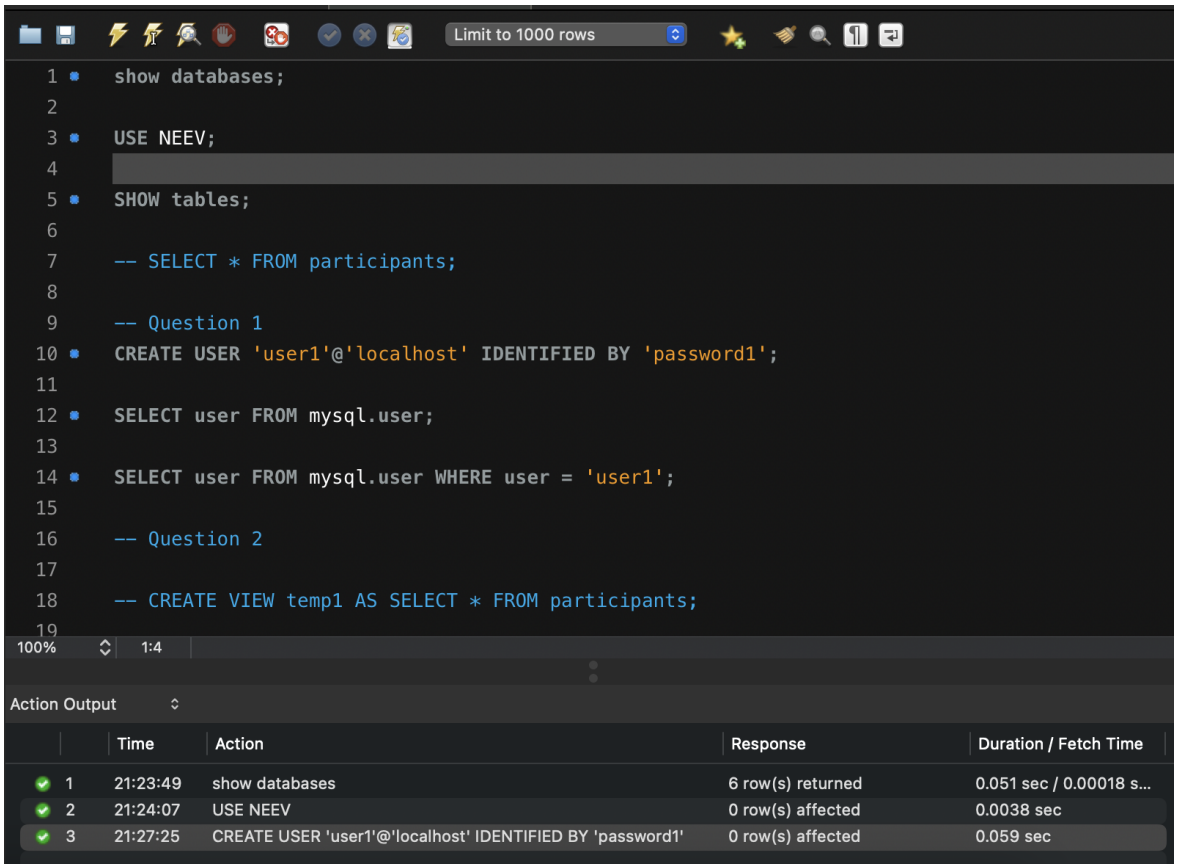
event_name	start_date	types	budget	no_of_participa...	duration	collecti...	total_expense
Project D	2022-04-01	Exhibition	800000	200	7	900000	700000
Project E	2022-05-01	Hackathon	400000	80	4	NULL	300000
Project F	2022-06-01	Training	600000	120	6	700000	500000
Project G	2022-07-01	Competition	500000	50	2	NULL	400000

Query Questions for G2:

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

```
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';
```

This query creates a new user with username “user1” and the user can login to the database with password as “password1”. The login screenshot for user1 will be shown below.



We have written Lines 12 and 14 to select user1 from the available users in mysql.server.

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

In this we have considered the tables volunteers and volunteering for making view1. Since we cannot directly add a column to a view, we first need to create a new table named “temp_table” using INNER JOIN. Once we have the table, then we will convert that to view1.

Make a table named “temp_table” and use INNER JOIN on the tables volunteers and volunteering. The query used is shown below:
Query Used:

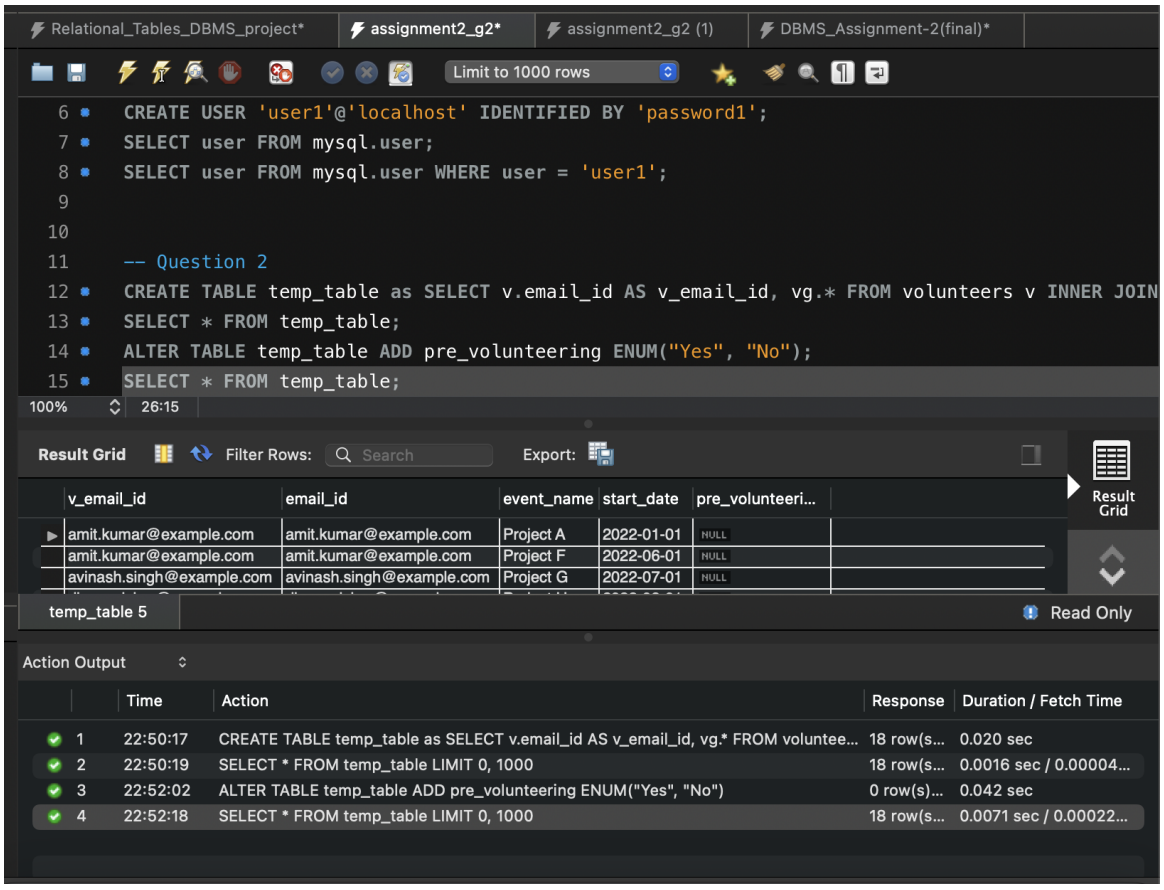
```
CREATE TABLE temp_table as SELECT v.email_id AS v_email_id, vg.*
FROM volunteers v INNER JOIN volunteering vg ON v.email_id = vg.email_id;
```

Add new Column:

We have added a new user defined column named “pre_volunteering” using ENUM which takes the value as “Yes” and “No”. Since ENUM is a user defined datatype, we could use that to make a new column.

Query Used:

```
ALTER TABLE temp_table ADD pre_volunteering ENUM("Yes", "No");
```

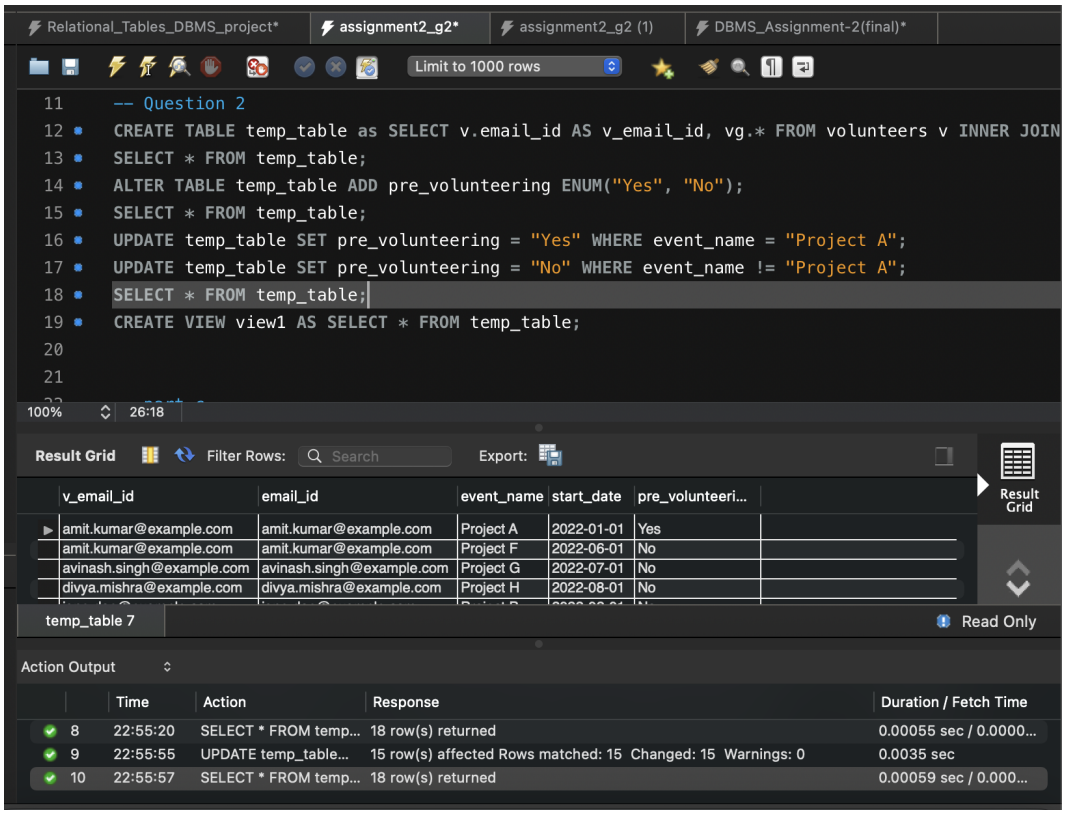


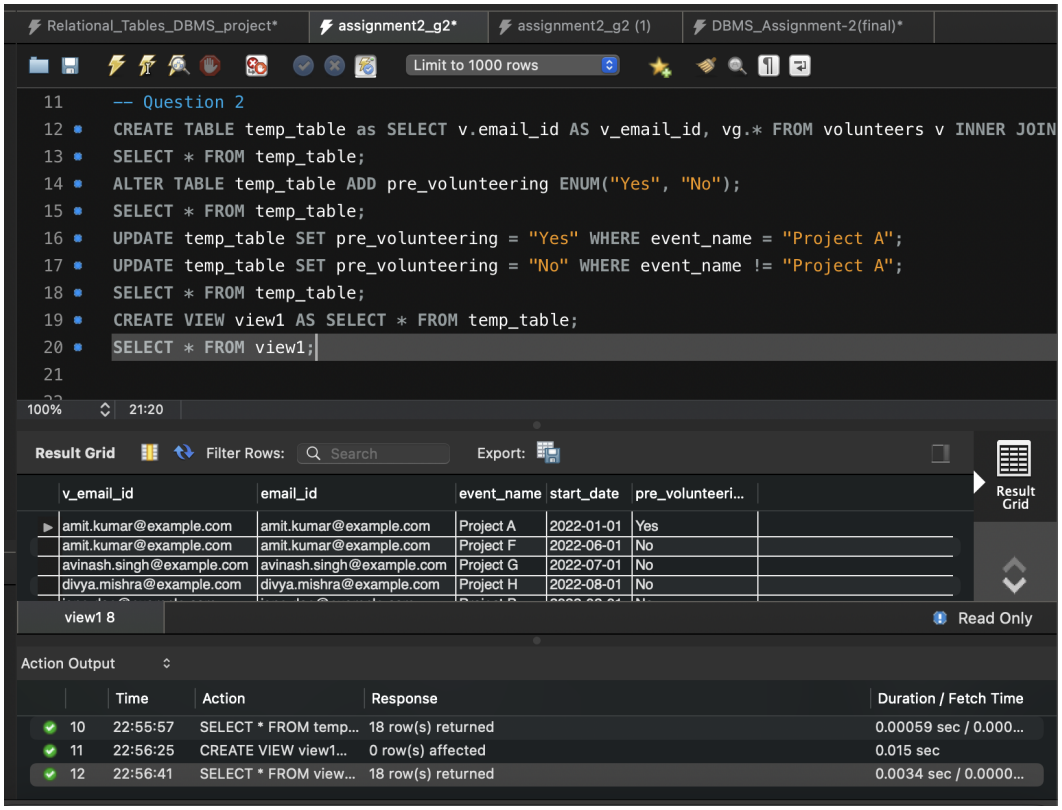
Make a view

Create a view named view1 using the following query.

Query Used:

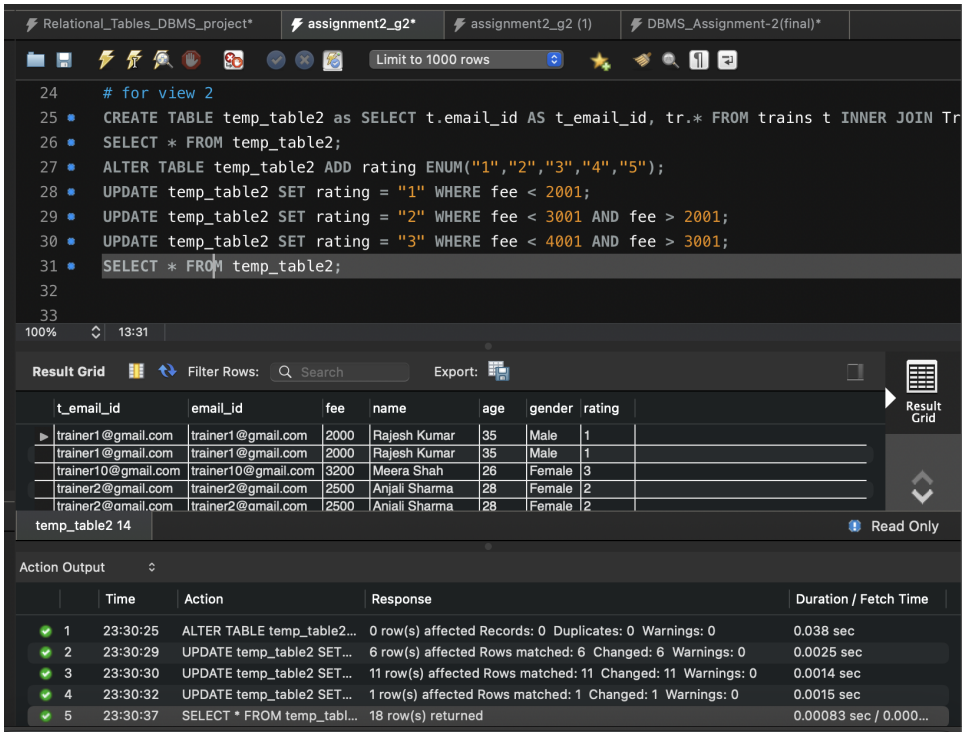
```
SELECT * FROM temp_table;
CREATE VIEW view1 AS SELECT * FROM temp_table;
```

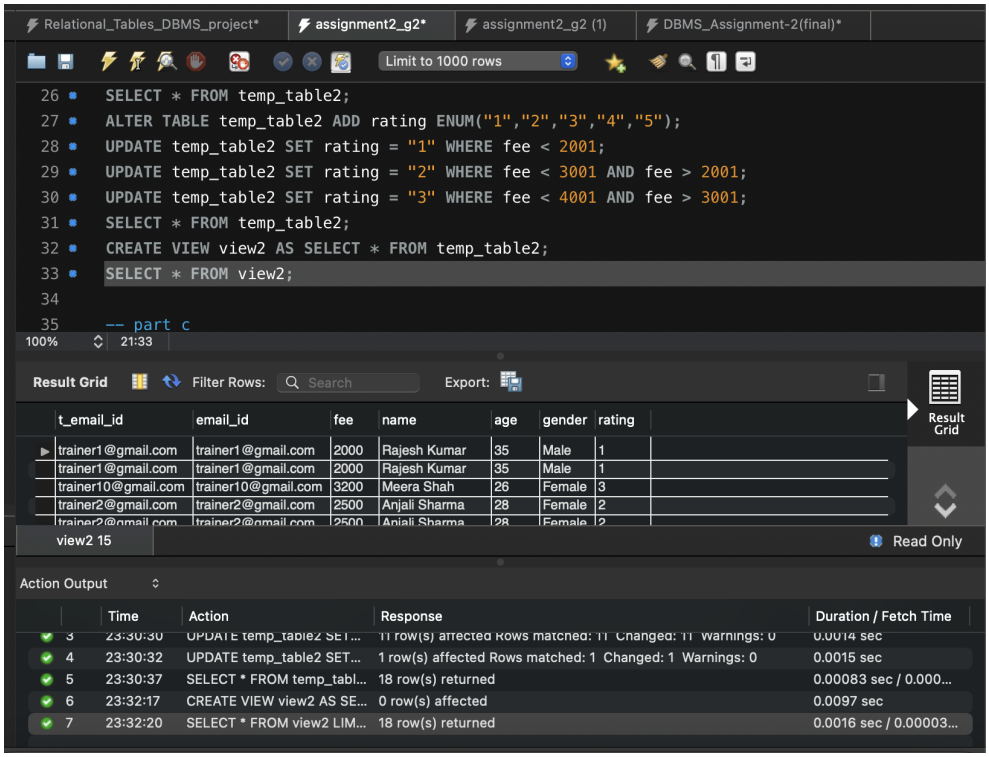




On View 2:

Here, we have taken two tables, Trainers and trains, through an inner join on a common column, ‘email_id’. We have created a temporary table temp_table2, which is used for capturing the inner join and adding the user defined datatype, enum(‘1’,’2’,’3’,’4’,’5’). At last, we created the view2 upon the temp_table2.

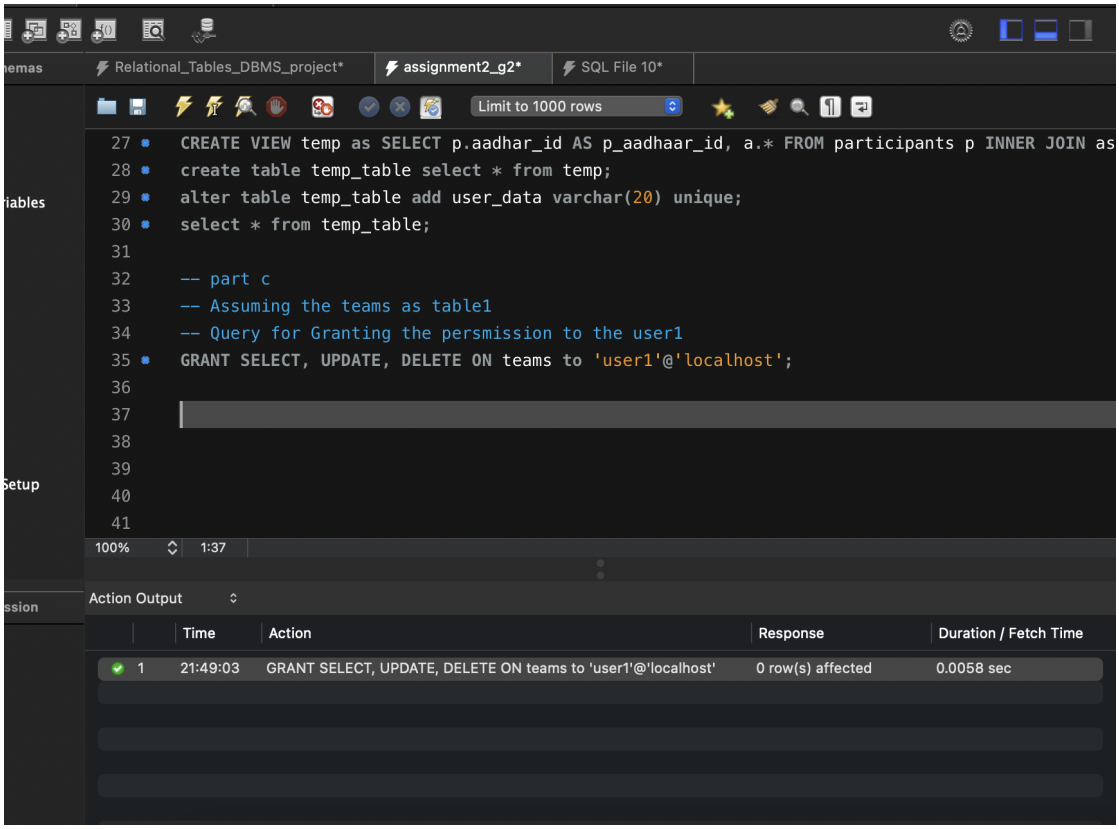


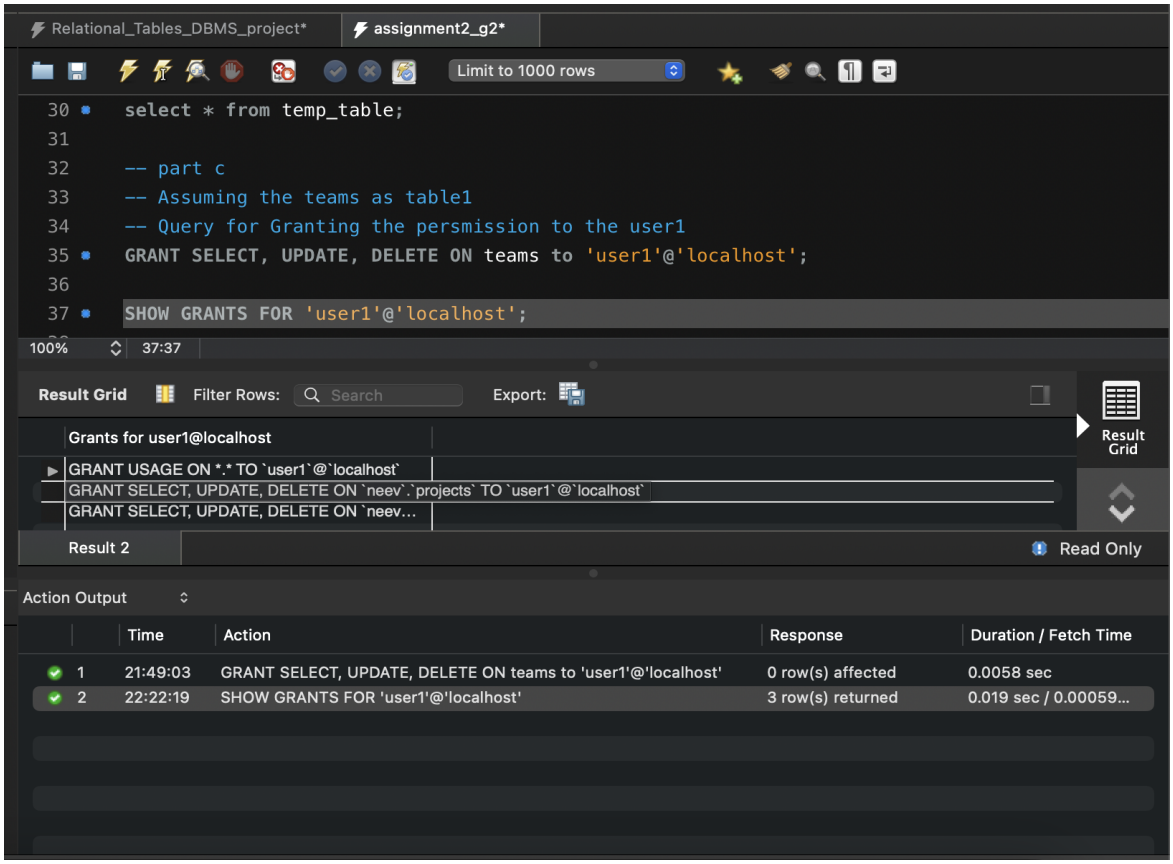


3. Grant "user1" the following permissions on "table1":
    - **SELECT**
    - **UPDATE**
    - **DELETE**

The following permissions have been granted to user1 after considering the table teams from our database. The screenshot and the user query are both displayed below.

```
GRANT SELECT, UPDATE, DELETE ON teams to 'user1'@'localhost';
```



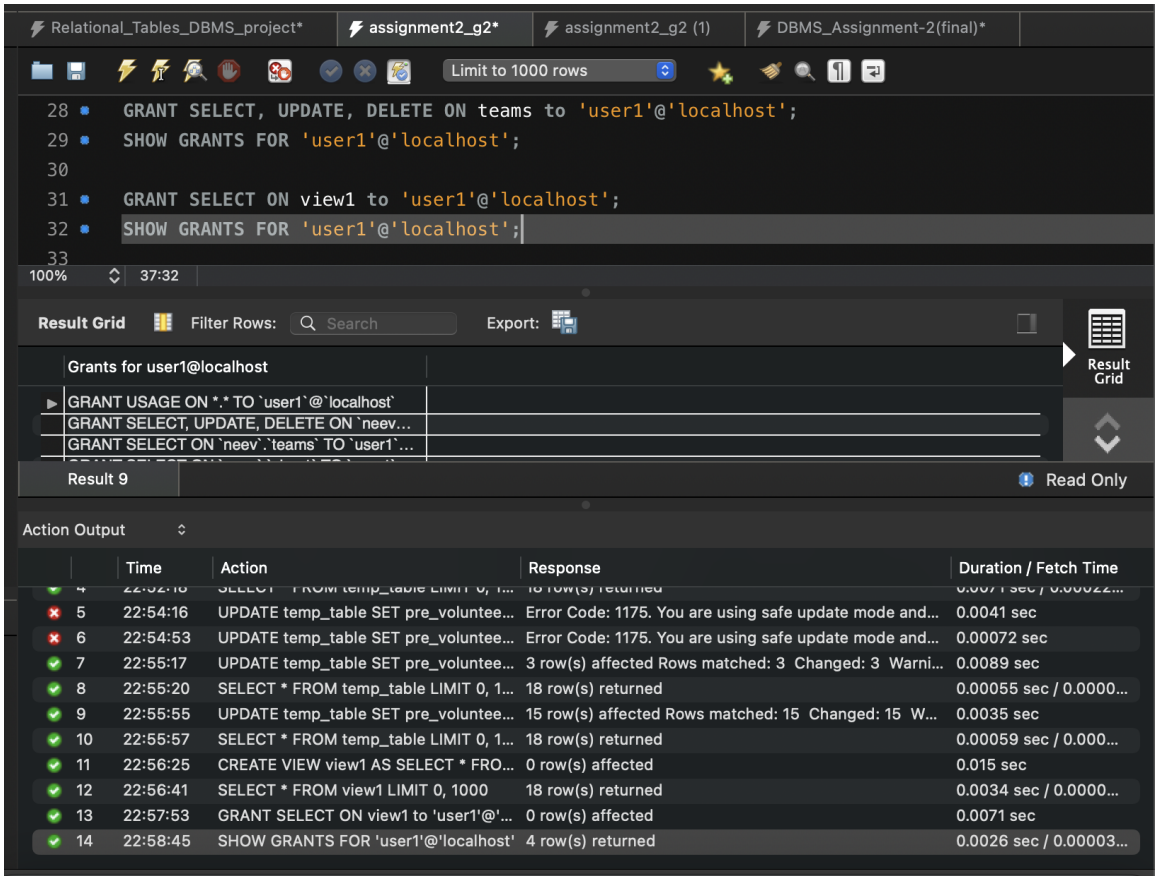


4. Grant "user1" the following permissions on "view1": SELECT

Grant user1 the permissions for view1 with the following query:-

Query Used:

```
GRANT SELECT ON view1 to 'user1'@'localhost';
```

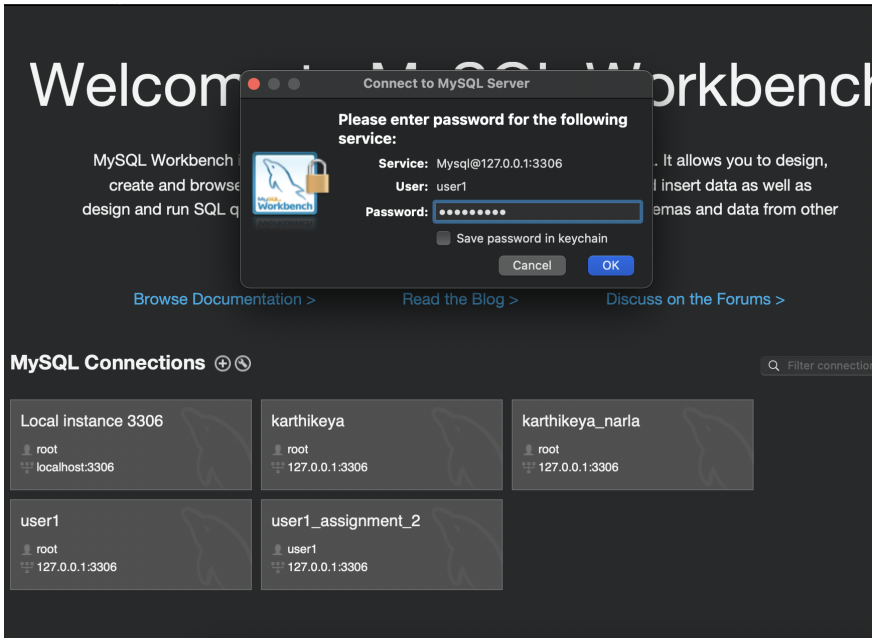


By using SHOW GRANTS, we could see the permissions that the user1 will have. As of now, we could see from the result grid that we have granted SELECT to user1.

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

Operations on Table

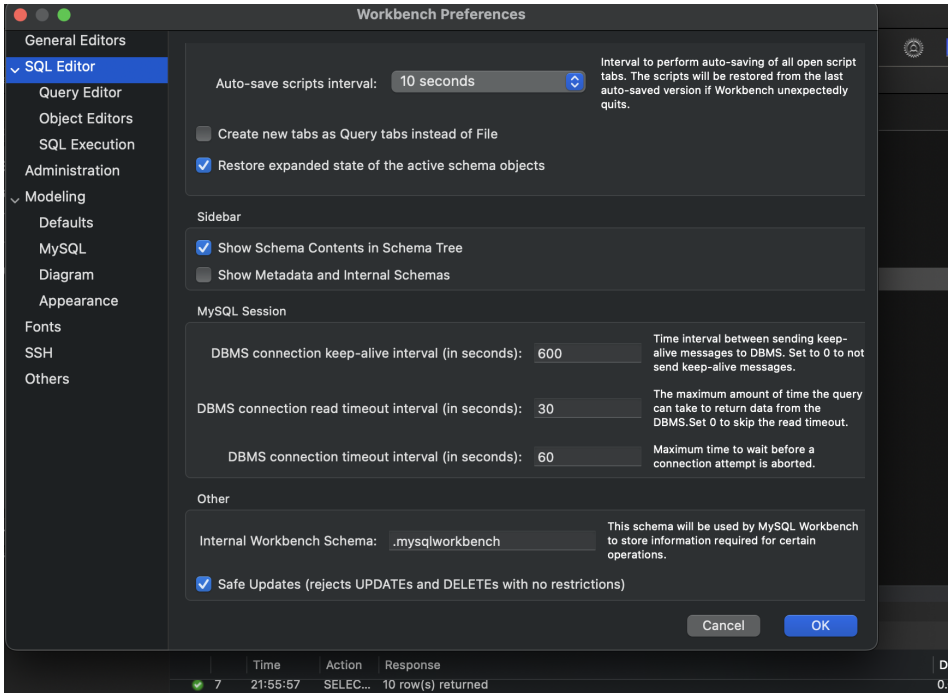
Now login into the database as user1, the screenshot is shown below shows the login page for user1. Login as user1.



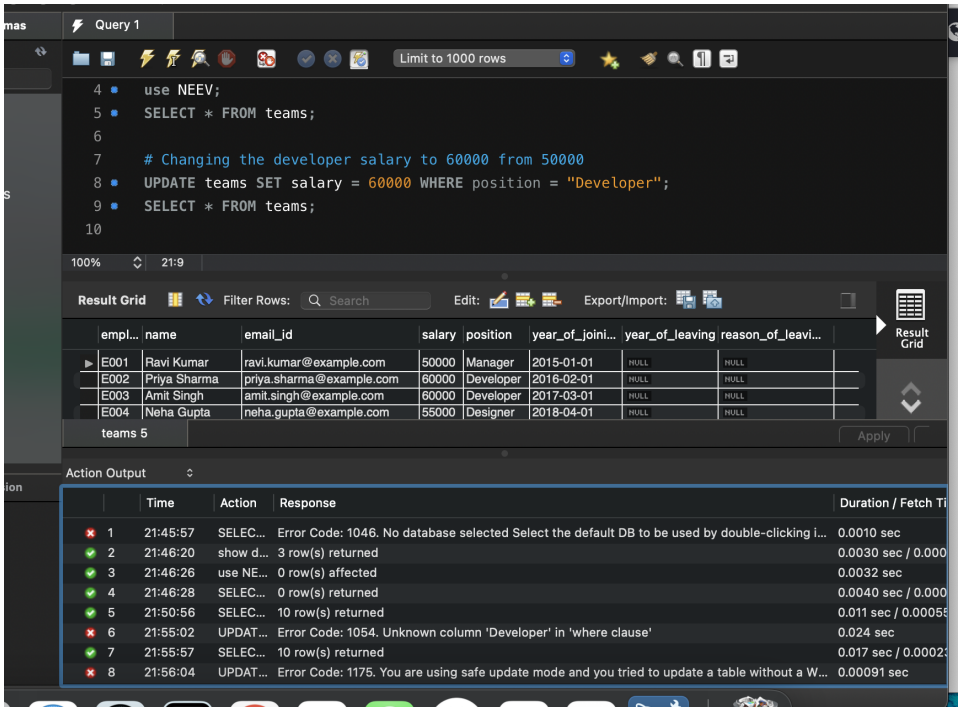
Below screenshot shows the 3 operations on table1. We have selected everything from the table teams and then performed an update operation that is Setting the Salary to 60000 where the position is “Developer”. So the below is the query, we executed for the following operation.

```
UPDATE teams SET salary = 60000 WHERE position = "Developer";
```

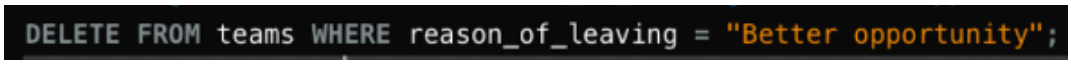
But there is one problem here since we are logged as user1. We need to switch off the safe mode to use other operations and then reconnect to the server. So we toggled the option off and then performed the procedures.

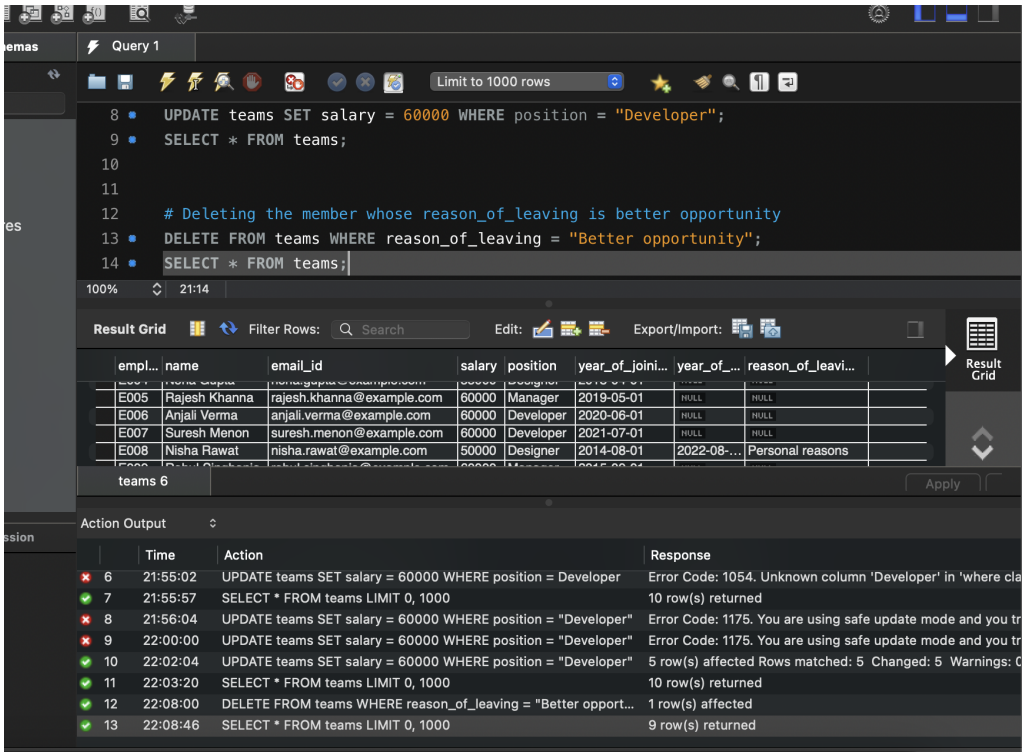


So now, let's perform the UPDATE and SELECT operations.



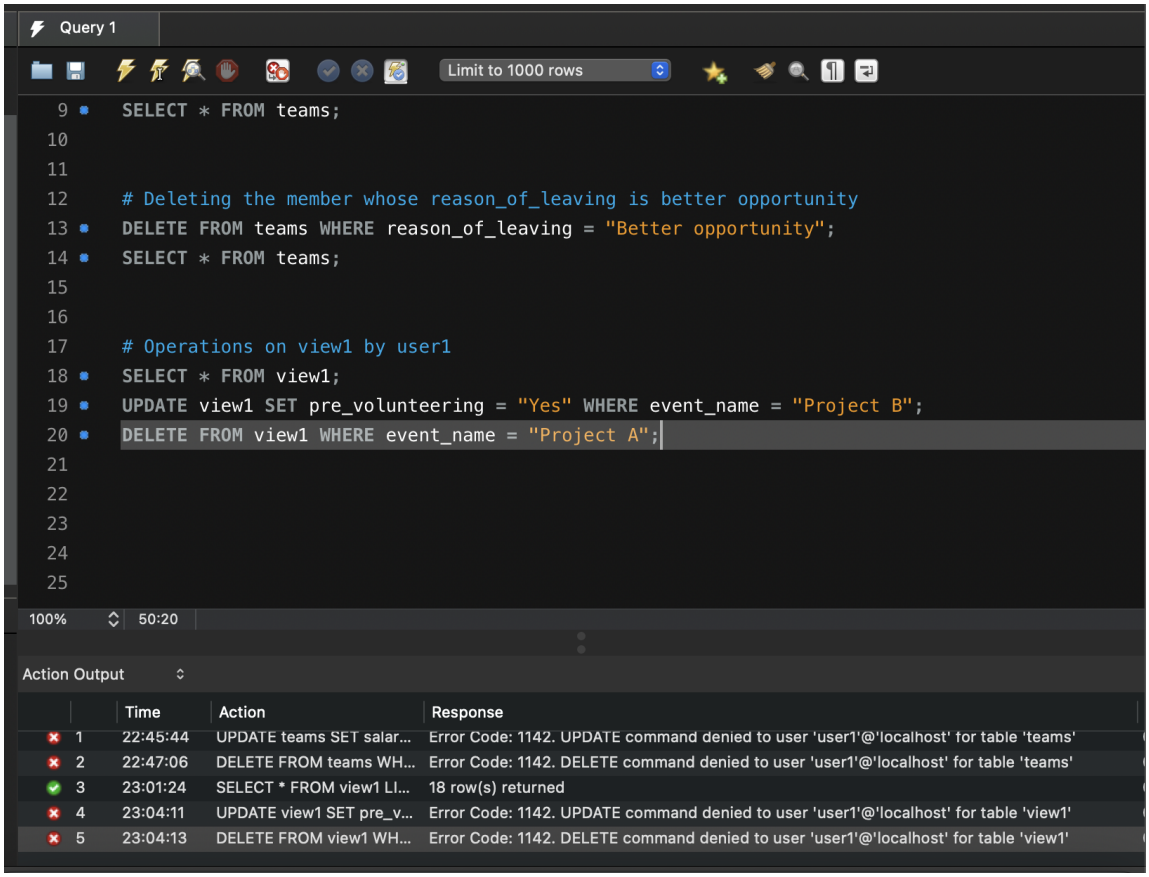
Delete operation:-





Operations on view1
Below is the screenshot showing the SELECT, UPDATE and DELETE operations.

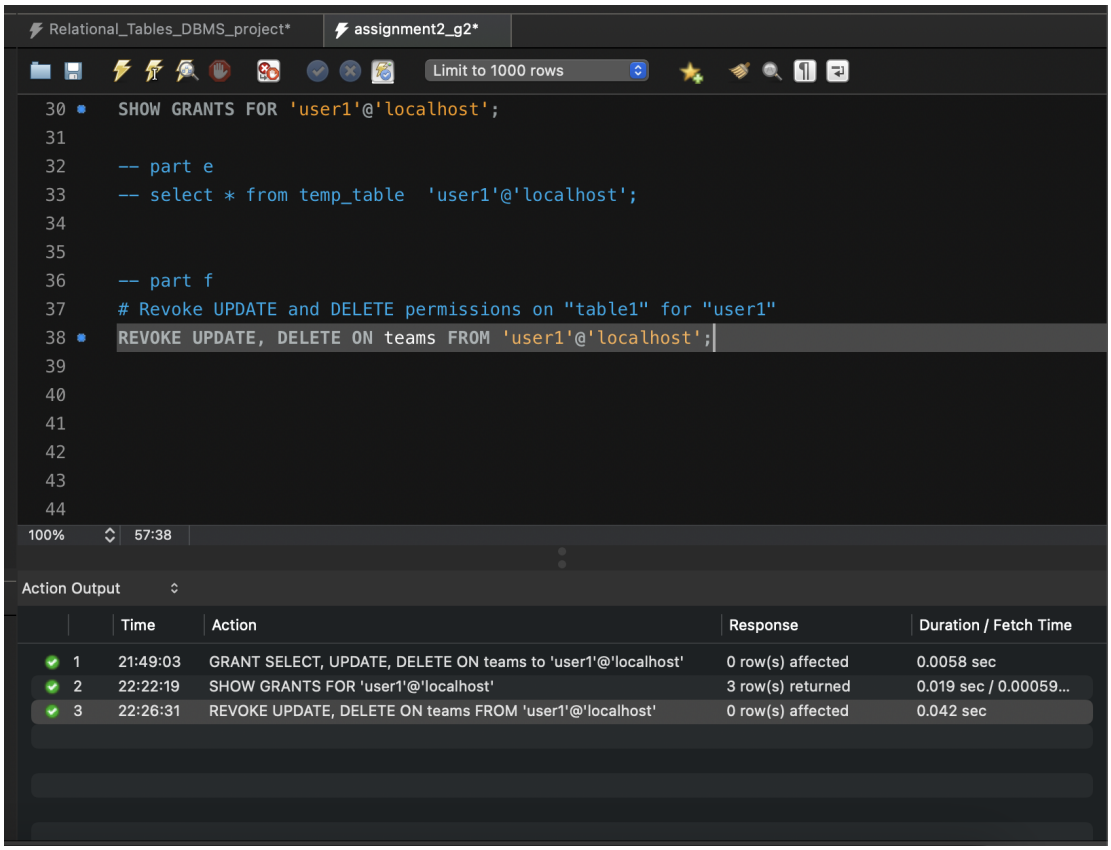
Findings: UPDATE and **DELETE** operations wont work on a view. **The SELECT** operation is still working. The reason being in Part 5, we give only **SELECT** operation to user1.



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

ON TABLE
Here, we are removing the privileges from user ‘user1’ corresponding to the commands, UPDATE and DELETE for the table ‘table1’. This will not allow the user to modify or remove the tuples from that table.

Query Used: **REVOKE UPDATE, DELETE ON teams FROM ‘user1@localhost’**



ON VIEW

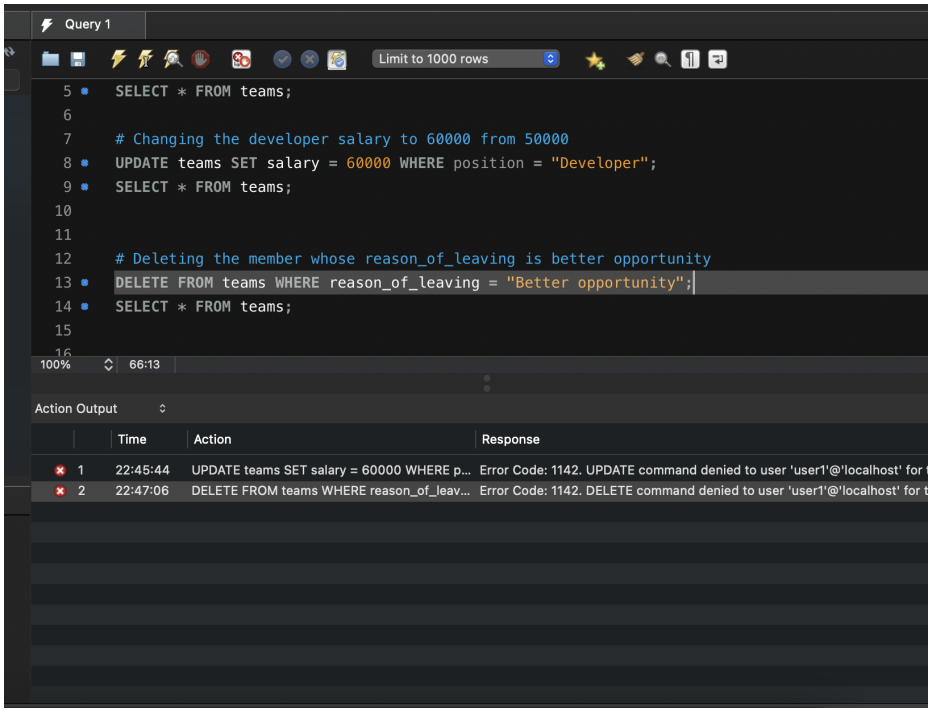
We were not asked to REVOKE any operations for user1 for the view1. So no revoke query was run on view1 here as per question.

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

On table1

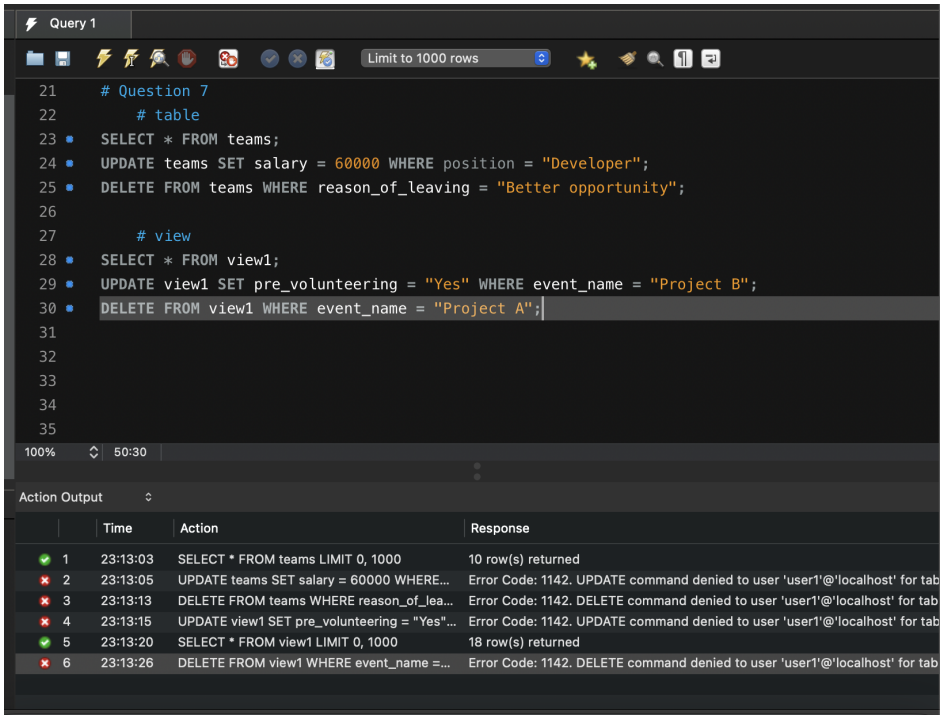
Now, we will perform the same operation that is DELETE and UPDATE, it can be seen from the Action Output that the queries cannot be executed. Below is the screenshot showing the failed query for **DELETE and UPDATE.**

Findings: **DELETE** and **UPDATE** won't work and the **SELECT** operation will work as we have only revoked the **DELETE** and **UPDATE** in Part 7.



On view1

Findings: DELETE AND **UPDATE** will not work and the **SELECT** will work. Though we have not revoked the grants in Part 6 as of the table but still the operations will not work because on a view you cannot perform **DELETE** AND **UPDATE**



2) A situation that violates referential integrity:

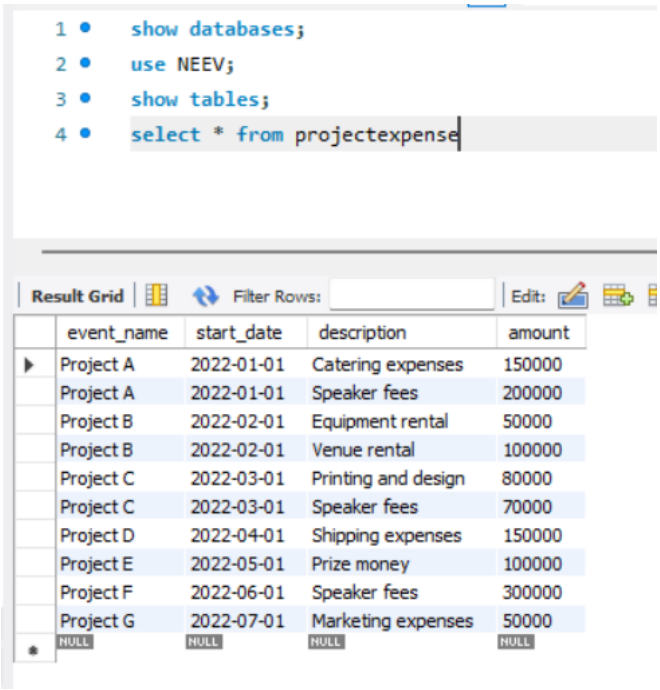
Referential integrity is said to have been violated in the relation **Expense.**

In this relation, the entity set *Project* is a strong entity set while the entity set *ProjectExpense* is weak.

As a result, if we delete a project record from the NEEV database, we must also delete the item for that project in the "ProjectExpense" table. If not, it will have financial information for a project not listed outside the NEEV database.

Testing in MySQL

Here is the table *Project* Expense's starting state, and it is evident that *ProjectExpense* initially has an entry labelled *"Project A."*



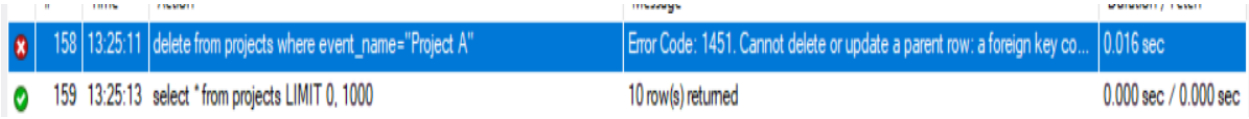
event_name	start_date	description	amount
Project A	2022-01-01	Catering expenses	150000
Project A	2022-01-01	Speaker fees	200000
Project B	2022-02-01	Equipment rental	50000
Project B	2022-02-01	Venue rental	100000
Project C	2022-03-01	Printing and design	80000
Project C	2022-03-01	Speaker fees	70000
Project D	2022-04-01	Shipping expenses	150000
Project E	2022-05-01	Prize money	100000
Project F	2022-06-01	Speaker fees	300000
Project G	2022-07-01	Marketing expenses	50000
NULL	NULL	NULL	NULL

Deletion

The following query attempts to remove *"Project A"* from the projects:

```
1 •  show databases;
2 •  use NEEV;
3 •  show tables;
4 •  select * from projectexpense;
5 •  delete from projects where event_name='Project A';
6 •  select * from projectexpense
```

The following error occurs when referential integrity is violated when we Use the above Query:



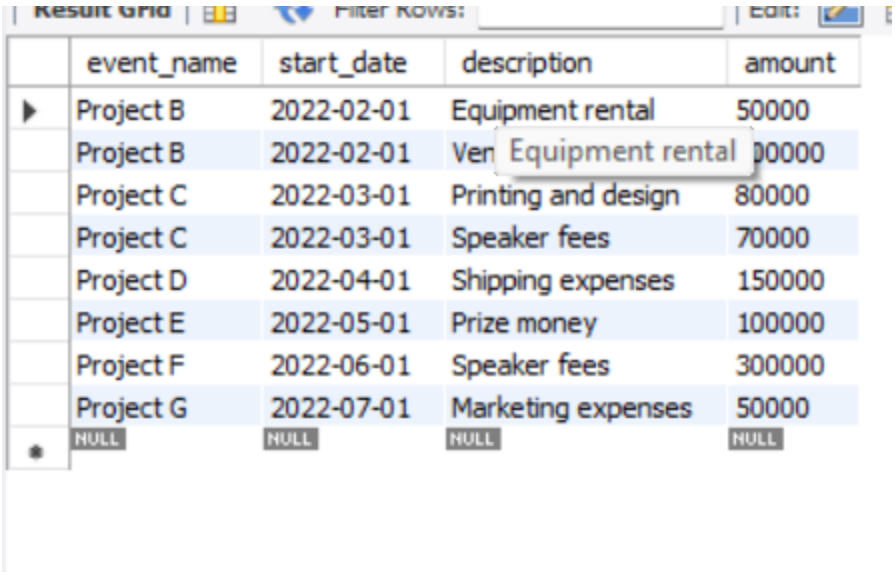
✖	158	13:25:11	delete from projects where event_name="Project A"	Error Code: 1451. Cannot delete or update a parent row: a foreign key co...	0.016 sec
✔	159	13:25:13	select * from projects LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec

When we attempted to delete *Project A* from the *Project* database, referential integrity with the "ProjectExpense" field was breached since the data for the costs associated with *Project A* was still available. Hence, MySQL forbids such a query. Here we can see that error shows that **we cannot delete or update a parent row Foreign Key constraint.**

Solution: We add **on delete cascade** on the entity set that references the table *Projects*

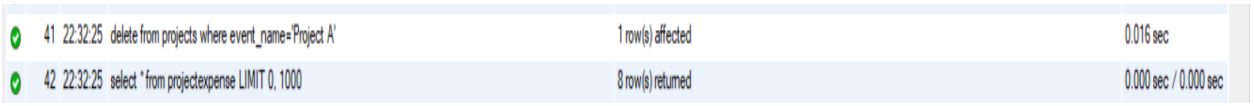
```
create table ProjectExpense (
event_name varchar(50),
start_date date,
description varchar(255) not null,
amount int not null,
primary key (event_name, start_date, description, amount),
foreign key (event_name, start_date)
references Projects(event_name, start_date) on delete cascade
);
```

As can be seen, using **on delete cascade** has resulted in removing the rows relating to *Project A*.



event_name	start_date	description	amount
Project B	2022-02-01	Equipment rental	50000
Project B	2022-02-01	Ven	00000
Project C	2022-03-01	Printing and design	80000
Project C	2022-03-01	Speaker fees	70000
Project D	2022-04-01	Shipping expenses	150000
Project E	2022-05-01	Prize money	100000
Project F	2022-06-01	Speaker fees	300000
Project G	2022-07-01	Marketing expenses	50000
NULL	NULL	NULL	NULL

Here we can see the previous error is resolved.



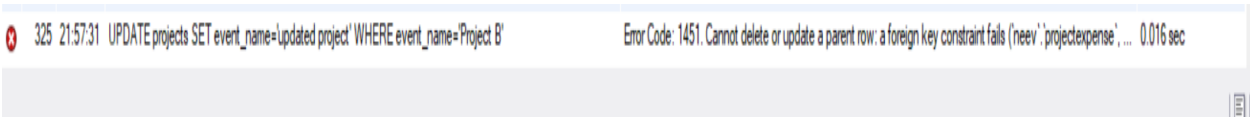
41	22:32:25	delete from projects where event_name='Project A'	1 row(s) affected	0.016 sec
42	22:32:25	select * from projectexpense LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec

Updating

The following query attempts to update *"Project A"* from the projects.

```
7  •  UPDATE projects
8     SET event_name='updated project'
9     WHERE event_name='Project B';
10 •  select * from projectexpense
```

Here we attempt to replace *Project B* with *updated project* in *Projects*.



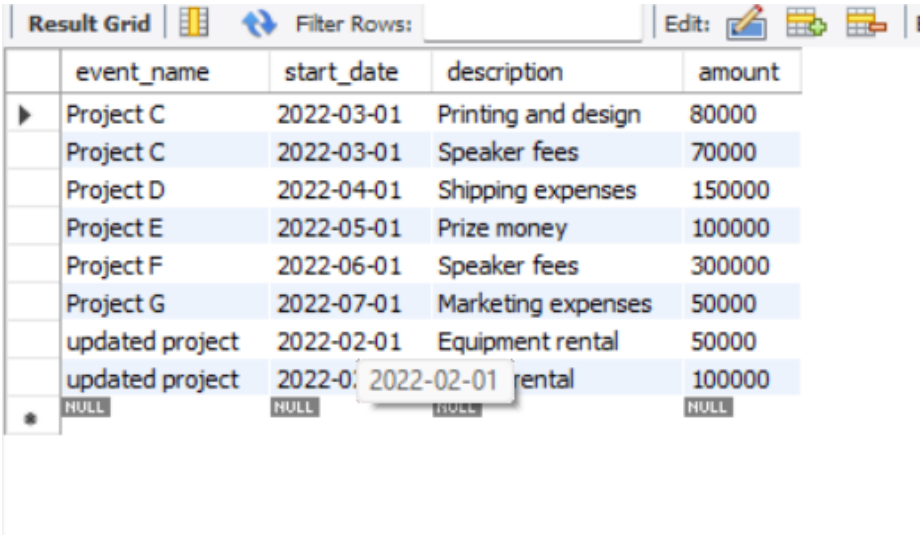
325	21:57:31	UPDATE projects SET event_name='updated project' WHERE event_name='Project B'	Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`neev`.`projectexpense`, ...	0.016 sec

Referential integrity with the "ProjectExpense" column was violated when we tried to delete "Project A" from the "Project" database since the information for the expenses related to "Project A" was still available. As a result, MySQL prohibits such a query. Here, it is evident that the issue indicates that **we cannot delete or alter a parent row: Foreign Key Constraint.**

Solution: We use **on update cascade** similar to **on delete cascade**

```
create table ProjectExpense (
event_name varchar(50),
start_date date,
description varchar(255) not null,
amount int not null,
primary key (event_name, start_date, description, amount),
foreign key (event_name, start_date)
references Projects(event_name, start_date) on delete cascade on update cascade
);
```

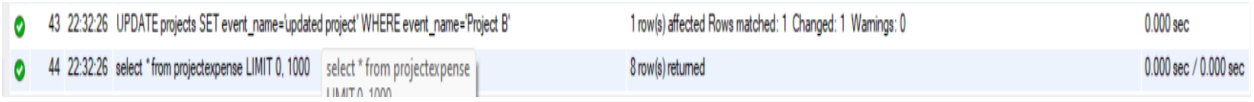
As we see here, the name *Project B* updates to *updated project*



Result Grid | Filter Rows: | Edit:

event_name	start_date	description	amount
Project C	2022-03-01	Printing and design	80000
Project C	2022-03-01	Speaker fees	70000
Project D	2022-04-01	Shipping expenses	150000
Project E	2022-05-01	Prize money	100000
Project F	2022-06-01	Speaker fees	300000
Project G	2022-07-01	Marketing expenses	50000
updated project	2022-02-01	Equipment rental	50000
updated project	2022-02-01	rental	100000
NULL	NULL	NULL	NULL

Here We can see the previous error is resolved .

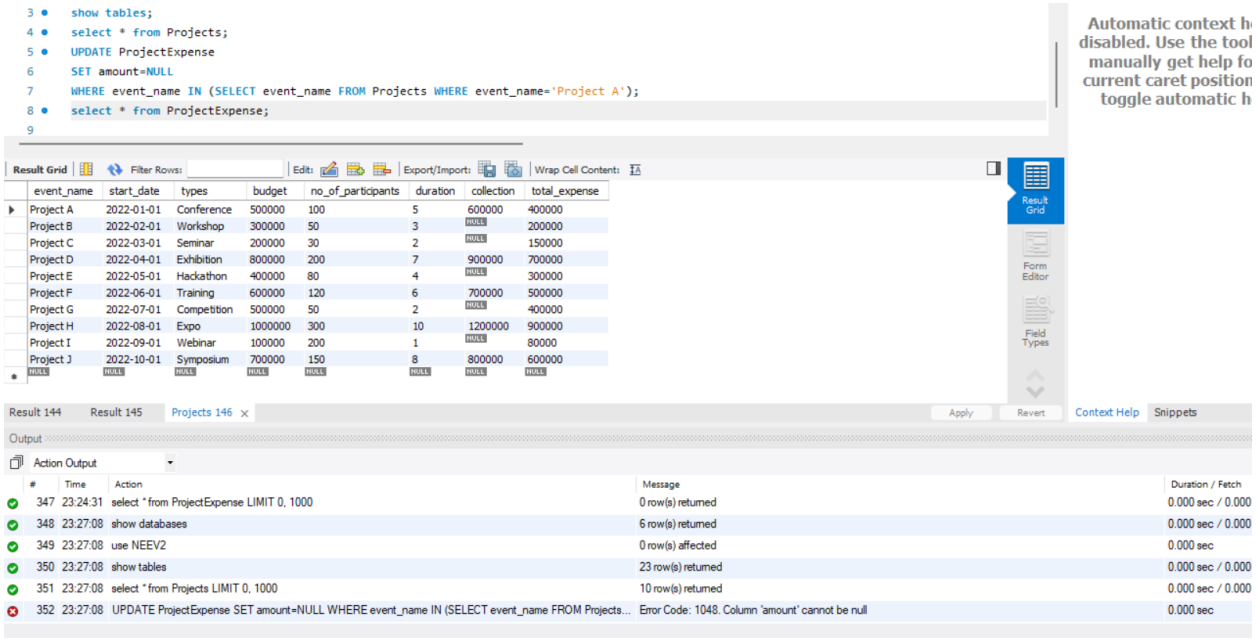


43	22:32:26	UPDATE projects SET event_name='updated project' WHERE event_name='Project B'		1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
44	22:32:26	select * from projectexpense LIMIT 0, 1000	select * from projectexpense LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec

G1 + G2

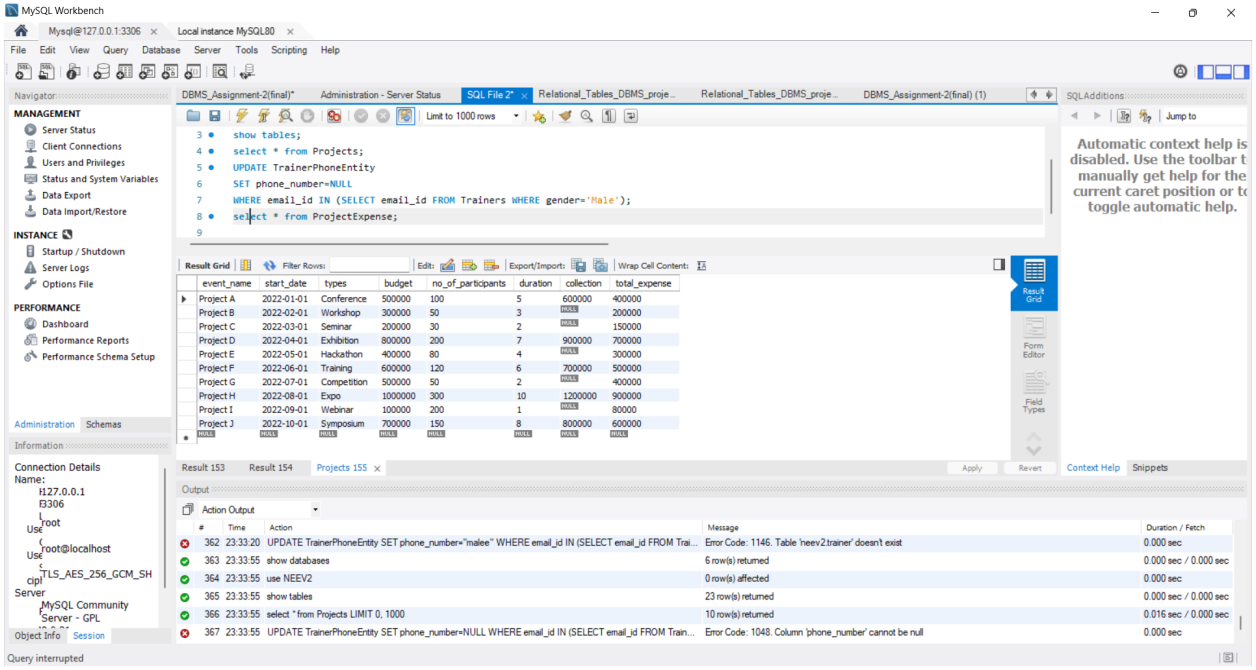
A)

Error 1:



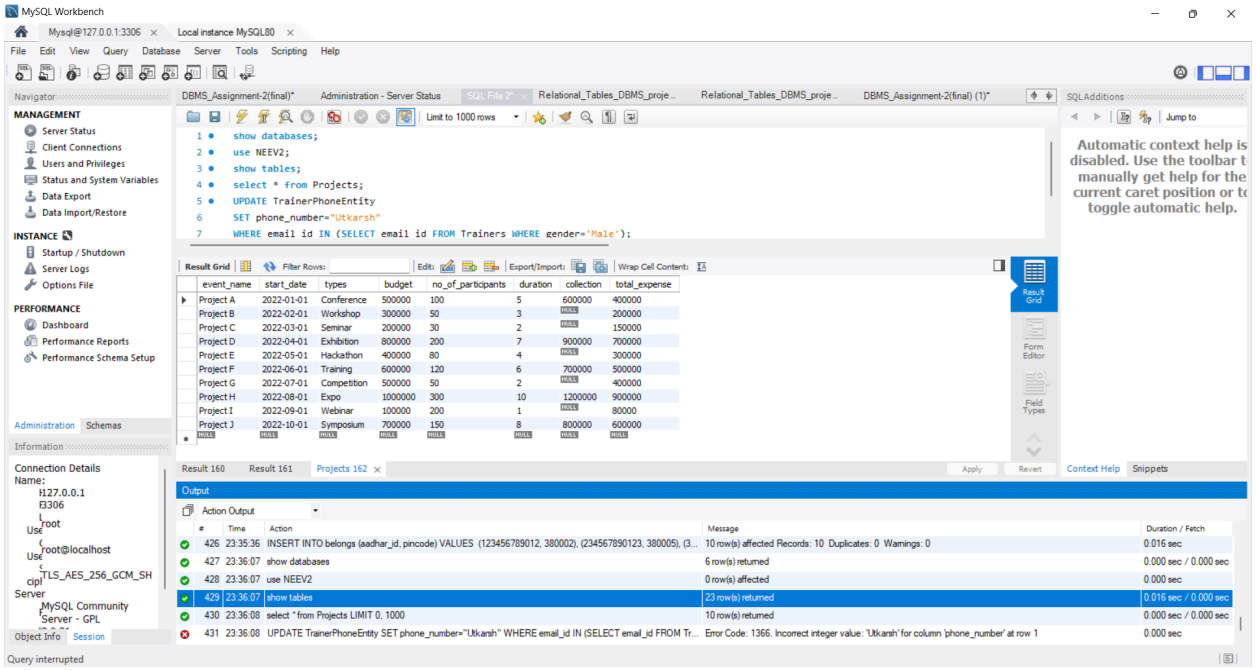
Here we tried to make *amount* as NULL, but that was not allowed because of our constraint

Error 2:



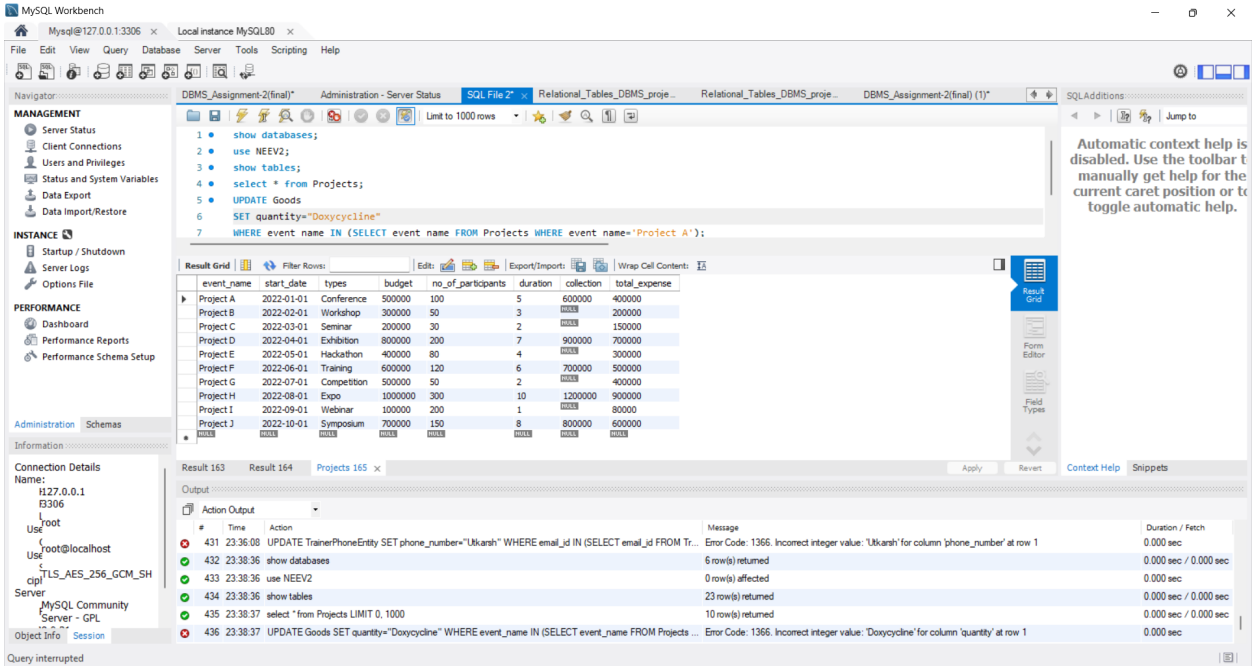
Here we tried to make Phone number of trainers as NULL but it was disallowed

Error 3:



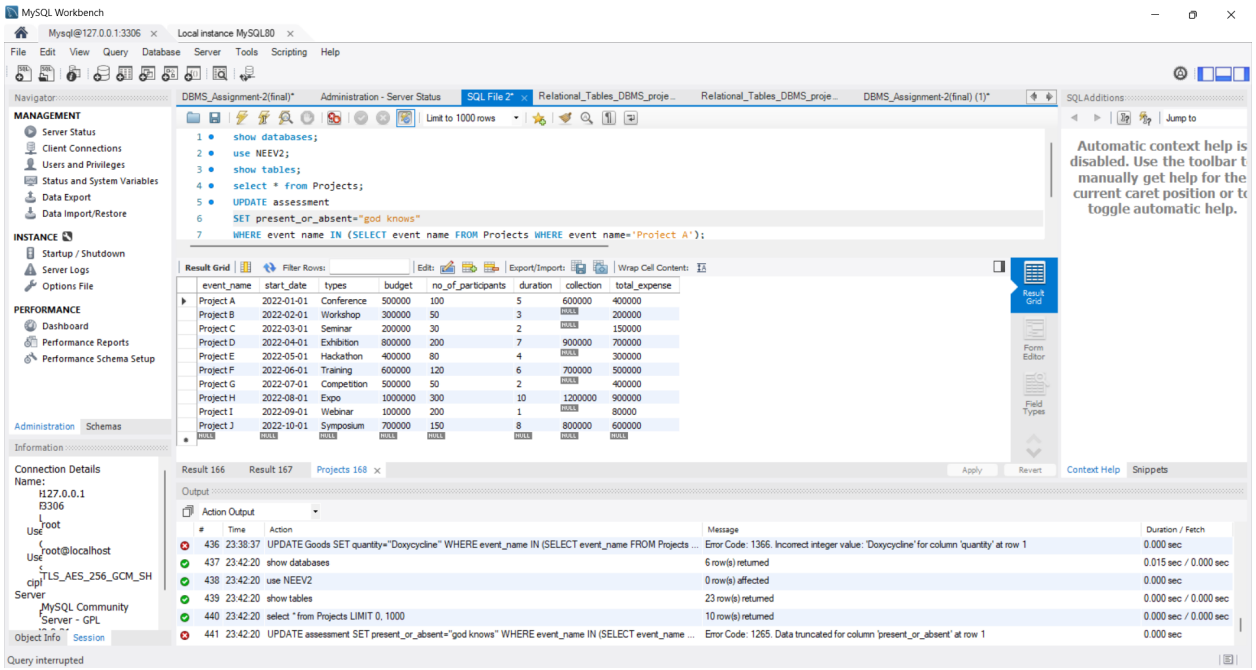
Here we tried entering a team member's name in phone number column but it failed, because non integer values are disallowed

Error 4:



Here, we tried changing the quantity column to a string but SQL did not allow it

Error 5:



Here we tried changing the status of attendance to something other than “present” or “absence”,

but it did not allow it.

b)

Storing the Images and cations in the database

```
CREATE TABLE ProjectPhotos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  project_event_name VARCHAR(50),
  project_start_date DATE,
  photo_url longblob NOT NULL,
  caption VARCHAR(255) DEFAULT NULL,
  FOREIGN KEY (project_event_name, project_start_date)
    REFERENCES Projects(event_name, start_date)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
INSERT INTO Beneficiary (aadhar_id, name, date_of_birth, gender, marital_status, education, photo, employed, photo_caption)
VALUES
(123456789011, 'Asha Sharma', '1980-05-12', 'Female', 'Married', 'Master of Business Administration', 'https://example.com/photo1.jpg', 'Yes',
(234567890121, 'Rahul Singh', '1995-08-02', 'Male', 'Unmarried', 'Bachelor of Engineering', 'https://example.com/photo1.jpg', 'Yes', 'Photo of
(345678901231, 'Priya Patel', '1988-12-25', 'Female', 'Married', 'Doctor of Medicine', 'https://example.com/photo1.jpg', 'No', NULL),
(456789012341, 'Rajesh Kumar', '1976-09-18', 'Male', 'Married', 'Bachelor of Science', 'https://example.com/photo1.jpg', 'Yes', 'Photo of Raje
(567890123451, 'Sneha Gupta', '1992-04-01', 'Female', 'Unmarried', 'Master of Computer Applications', 'https://example.com/photo1.jpg', 'No', I
(678901234561, 'Vikram Singh', '1985-06-21', 'Male', 'Married', 'Bachelor of Commerce', 'https://example.com/photo1.jpg', 'Yes', 'Photo of Vik
(789012345671, 'Anjali Reddy', '1990-03-15', 'Female', 'Unmarried', 'Bachelor of Arts', 'https://example.com/photo1.jpg', 'No', NULL),
(890123456781, 'Amit Sharma', '1983-11-08', 'Male', 'Married', 'Master of Science', 'https://example.com/photo1.jpg', 'Yes', 'Photo of Amit Sh
(901234567891, 'Shalini Verma', '1987-07-14', 'Female', 'Married', 'Bachelor of Business Administration', NULL, 'No', NULL),
(123450987651, 'Sanjay Kumar', '1979-01-31', 'Male', 'Married', 'Master of Arts', 'https://example.com/photo1.jpg', 'Yes', 'Photo of Sanjay Ku
```

id	project_event_name	project_start_date	photo_url	caption
1	Project A	2022-01-01	BLOB	Conference photo 1
2	Project A	2022-01-01	BLOB	Conference photo 2
3	Project B	2022-02-01	BLOB	Workshop photo 1
4	Project C	2022-03-01	BLOB	NULL
5	Project D	2022-04-01	BLOB	Exhibition photo 1
6	Project E	2022-05-01	BLOB	NULL
7	Project F	2022-06-01	BLOB	Training photo 1
8	Project G	2022-07-01	BLOB	NULL
9	Project H	2022-08-01	BLOB	Expo photo 1

C)

Natural Join

```
SELECT * FROM Beneficiary NATURAL JOIN BeneficiaryPhoneEntity;
```

aadhar_id	name	date_of_birth	gender	marital_status	education	photo	employed	photo_caption	phone_number
123450987654	Sanjay Kumar	1979-01-31	Male	Married	Master of Arts	NULL	Yes	Photo of Sanjay Kumar	1234567890
123456789012	Asha Sharma	1980-05-12	Female	Married	Master of Business Administration	NULL	Yes	Photo of Asha Sharma	9876543210
234567890123	Rahul Singh	1995-08-02	Male	Unmarried	Bachelor of Engineering	NULL	Yes	Photo of Rahul Singh	8765432109
345678901234	Priya Patel	1988-12-25	Female	Married	Doctor of Medicine	NULL	No	NULL	7654321098
456789012345	Rajesh Kumar	1976-09-18	Male	Married	Bachelor of Science	NULL	Yes	Photo of Rajesh Kumar	6543210987
567890123456	Sneha Gupta	1992-04-01	Female	Unmarried	Master of Computer Applications	NULL	No	NULL	5432109876
678901234567	Vikram Singh	1985-06-21	Male	Married	Bachelor of Commerce	NULL	Yes	Photo of Vikram Singh	4321098765
789012345678	Anjali Reddy	1990-03-15	Female	Unmarried	Bachelor of Arts	NULL	No	NULL	3210987654
890123456789	Amit Sharma	1983-11-08	Male	Married	Master of Science	NULL	Yes	Photo of Amit Sharma	2109876543
901234567890	Shalini Verma	1987-07-14	Female	Married	Bachelor of Business Administration	NULL	No	NULL	1098765432

Outer Join

```
SELECT * FROM Beneficiary LEFT OUTER JOIN BeneficiaryPhoneEntity
ON Beneficiary.aadhar_id = BeneficiaryPhoneEntity.aadhar_id;
```

aadhar_id	name	date_of_birth	gender	marital_status	education	photo	employed	photo_caption	aadhar_id	phone_number
123450987654	Sanjay Kumar	1979-01-31	Male	Married	Master of Arts	NULL	Yes	Photo of Sanjay Kumar	123450987654	1234567890
123456789012	Asha Sharma	1980-05-12	Female	Married	Master of Business Administration	NULL	Yes	Photo of Asha Sharma	123456789012	9876543210
234567890123	Rahul Singh	1995-08-02	Male	Unmarried	Bachelor of Engineering	NULL	Yes	Photo of Rahul Singh	234567890123	8765432109
345678901234	Priya Patel	1988-12-25	Female	Married	Doctor of Medicine	NULL	No	NULL	345678901234	7654321098
456789012345	Rajesh Kumar	1976-09-18	Male	Married	Bachelor of Science	NULL	Yes	Photo of Rajesh Kumar	456789012345	6543210987
567890123456	Sneha Gupta	1992-04-01	Female	Unmarried	Master of Computer Applications	NULL	No	NULL	567890123456	5432109876
678901234567	Vikram Singh	1985-06-21	Male	Married	Bachelor of Commerce	NULL	Yes	Photo of Vikram Singh	678901234567	4321098765
789012345678	Anjali Reddy	1990-03-15	Female	Unmarried	Bachelor of Arts	NULL	No	NULL	789012345678	3210987654
890123456789	Amit Sharma	1983-11-08	Male	Married	Master of Science	NULL	Yes	Photo of Amit Sharma	890123456789	2109876543

Renaming

```
SELECT name AS beneficiary_name, phone_number AS beneficiary_phone_number
FROM Beneficiary JOIN BeneficiaryPhoneEntity
ON Beneficiary.aadhar_id = BeneficiaryPhoneEntity.aadhar_id;
```

beneficiary_name	beneficiary_phone_number
Sanjay Kumar	1234567890
Asha Sharma	9876543210
Rahul Singh	8765432109
Priya Patel	7654321098
Rajesh Kumar	6543210987
Sneha Gupta	5432109876
Vikram Singh	4321098765
Anjali Reddy	3210987654
Amit Sharma	2109876543
Shalini Verma	1098765432

Two or more different kinds of aggregate functions

```
SELECT gender, COUNT(*) AS count, AVG(phone_number) AS avg_phone_number
FROM Beneficiary JOIN BeneficiaryPhoneEntity
ON Beneficiary.aadhar_id = BeneficiaryPhoneEntity.aadhar_id
GROUP BY gender;
```

gender	count	avg_phone_number
Male	5	4594837258.8000
Female	5	5454545454.0000

Case Statement

```
SELECT gender,
  COUNT(*) AS count,
  SUM(CASE WHEN BeneficiaryPhoneEntity.phone_number >= 1000000000
  AND BeneficiaryPhoneEntity.phone_number < 2000000000 THEN 1 ELSE 0 END) AS jio_count,
  SUM(CASE WHEN BeneficiaryPhoneEntity.phone_number >= 2000000000
  AND BeneficiaryPhoneEntity.phone_number < 3000000000 THEN 1 ELSE 0 END) AS airtel_count
FROM Beneficiary LEFT OUTER JOIN BeneficiaryPhoneEntity
ON Beneficiary.aadhar_id = BeneficiaryPhoneEntity.aadhar_id
GROUP BY gender;
```

gender	count	jio_count	airtel_count
Male	5	1	1
Female	5	1	0

Contribution

Group G1		
Roll No.	**Name**	**Participation**
20110002	Mahesh Abhale	Contributed in creating tables for all entity and relationship sets, including user-defined data types and data population, thereby facilitating seamless query execution. Also, documented the final report, and also reviewed the work of G2.
20110047	Chirag Sarda	Contributed in creating tables for all entity and relationship sets, including user-defined data types and data population, thereby facilitating seamless query execution. Also, documented the final report, and also reviewed the work of G2.
20110052	Sandeep Desai	Populated the table with random data values. Random data generation with python script for time analysis of indexing. Executed indexing and Table Extension.
20110034	Badal Chowdhary	Populated the table with random data values. Random data generation with python script for time analysis of indexing.- Executed indexing and Table Extension.
20110024	Ary Pratap Singh	Contributed in creating tables for all entity and relationship sets, including user-defined data types and data population, thereby facilitating seamless query execution. Also, documented the final report, and also reviewed the work of G2.

Group G2		
Roll No.	**Name**	**Participation**
20110129	Utkarsh Mittal	Contributed in the 3.2.2 part where referential integrity was violated, displayed changes in the table, and discussed solutions to these issues. Also contributed in Granting and revoking the permissions that UPDATE and DELETE Ran SQL queries violating constraints and took screenshots
20110156	Rahul Rai	Contributed in The First Seven Parts of 3.2.1, the creation of a user, the process of providing different permissions on tables and views, and the process of revoking the permissions in a database management system , and reviewed the work of the G1, G2 Documentation Part .
20110117	Narla Karthikeya	Contributed in The First Seven Parts of 3.2.1, the creation of a user, the process of providing different permissions on tables and views, and the process of revoking the permissions in a database management system , and reviewed the work of the G1, G2 Documentation Part .
20110242	Zeeshan Snehil Bhagat	Contributed in The First Seven Parts of 3.2.1, the creation of a user, the process of providing different permissions on tables and views, and the process of revoking the permissions in a database management system , and reviewed the work of the G1, G2 Documentation Part .
20110154	Rahul Lalani	Contributed in the 3.2.2 part where referential integrity was violated, displayed changes in the table, and discussed solutions to these issues. Also contributed in Granting and revoking the permissions that UPDATE and DELETE examined the G1, G2, and Final Documentation Part .

20110056	Dheeraj Yadav	Contributed in the 3.2.2 part where referential integrity was violated, displayed changes in the table, and discussed solutions to these issues. Also contributed in Granting and revoking the permissions that UPDATE and DELETE examined the G1, G2, and Final Documentation Part .