To fetch the information first from our database we will first use the query given below:

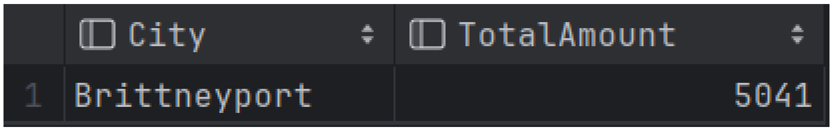
use makemytrip;

Query 1

```
SELECT users.City, SUM(tickets.Amount) AS TotalAmount
FROM users
INNER JOIN tickets ON users.userid = tickets.userid
GROUP BY users.City
HAVING SUM(tickets.Amount) = (
        SELECT MAX(TotalSum)
        FROM (
        SELECT SUM(Amount) AS TotalSum
        FROM tickets,users where tickets.userid=users.userid
        GROUP BY users.City
        ) AS MaxSums
);
```

Function: The above query retrieves the cities and the total sum of the tickets in each city. The tables “users” and “tickets” are joined on the “userid” column using an inner join, and only those tickets are included whose ticket amount is equal to the maximum ticket amount over all cities.

Output:



	City	TotalAmount
1	Brittneyport	5041

Relational Algebra

```
π City, TotalAmount (
          ρ users (Users ⋈ Tickets),
          γ City; SUM(Amount) → TotalAmount (
             ρ users (Users ⋈ Tickets),
             γ City; SUM(Amount) (
                ρ users (Users ⋈ Tickets)
             )
          ),
          TotalAmount = (
             ρ MaxSums (
                γ MAX(TotalSum) → TotalSum (
                   γ City; SUM(Amount) → TotalSum (
                      ρ users (Users ⋈ Tickets),
                      γ City; SUM(Amount) (
                          ρ users (Users ⋈ Tickets)
                        )
                      )
                    )
                  )
                )
              )
```

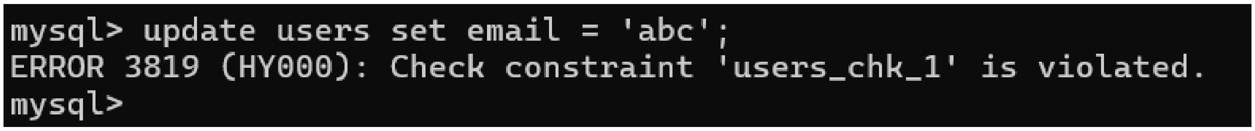
Query 2

update users set email = 'abc';

Note: Violates Integrity constraint.

Function: On running the above query then we expect to get an error message to show up as the output as the above query violates the constraints of our database, as we have the constraint in our inputs such that our emails entered should have an '@' and a .com or similar domain name in it.

Output:



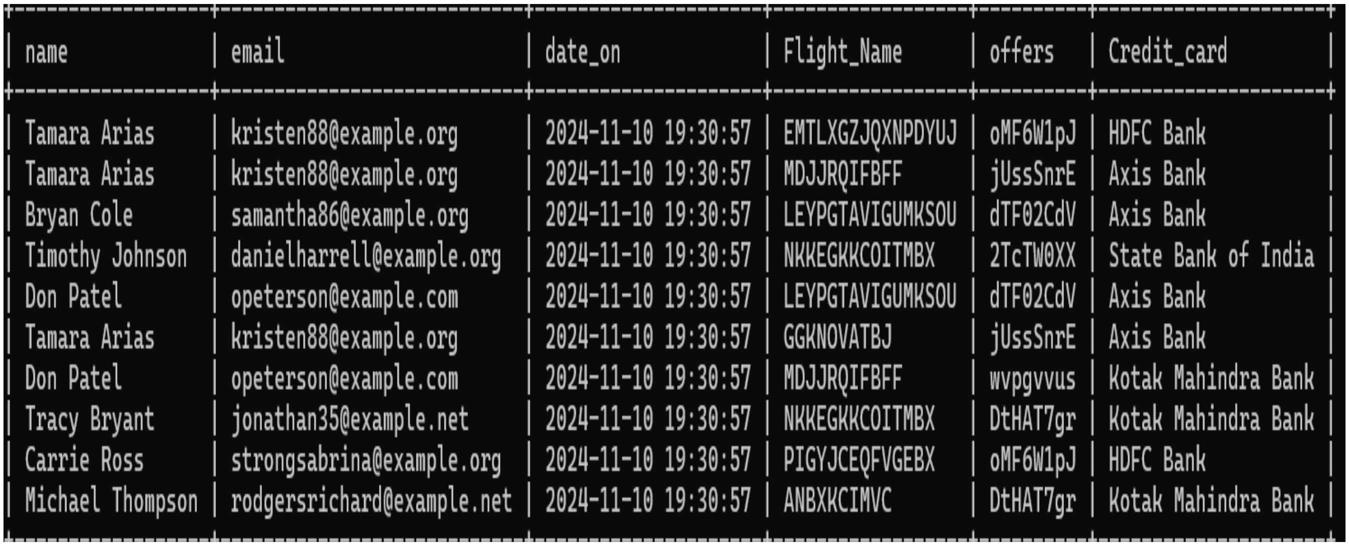
```
mysql> update users set email = 'abc';
ERROR 3819 (HY000): Check constraint 'users_chk_1' is violated.
mysql>
```

Query 3

```
SELECT u.name, u.email, l.date_on, f.name AS Flight_Name, bl.Offers_Code AS offers,
bl.Credit_card
  FROM Users u
   INNER JOIN Booked_Loungue l ON u.userid = l.userid
   INNER JOIN Flight f ON l.Flight_No = f.Flight_No
           INNER JOIN Loungue bl ON bl.Offers_code = l.Offers_code;
```

Function: This query retrieves the information about the people who have booked lounge while booking tickets for the flight. It joins multiple tables together keeping some constraints on the join. It retrieves the name, email date of booking and some other details of the bookings.

Output:



```
+------------------+----------------------------+---------------------+------------------+----------+---------------------+
| name             | email                      | date_on             | Flight_Name      | offers   | Credit_card         |
+------------------+----------------------------+---------------------+------------------+----------+---------------------+
| Tamara Arias     | kristen88@example.org      | 2024-11-10 19:30:57 | EMTLXGZJQXNPDYUJ | oMF6W1pJ | HDFC Bank           |
| Tamara Arias     | kristen88@example.org      | 2024-11-10 19:30:57 | MDJJRQIFBFF      | jUssSnrE | Axis Bank           |
| Bryan Cole       | samantha86@example.org     | 2024-11-10 19:30:57 | LEYPGTAVIGUMKSOU | dTF02CdV | Axis Bank           |
| Timothy Johnson  | danielharrell@example.org  | 2024-11-10 19:30:57 | NKKEGKKCOITMBX   | 2TcTW0XX | State Bank of India |
| Don Patel        | opeterson@example.com      | 2024-11-10 19:30:57 | LEYPGTAVIGUMKSOU | dTF02CdV | Axis Bank           |
| Tamara Arias     | kristen88@example.org      | 2024-11-10 19:30:57 | GGKNOVATBJ       | jUssSnrE | Axis Bank           |
| Don Patel        | opeterson@example.com      | 2024-11-10 19:30:57 | MDJJRQIFBFF      | wvpgvvus | Kotak Mahindra Bank |
| Tracy Bryant     | jonathan35@example.net     | 2024-11-10 19:30:57 | NKKEGKKCOITMBX   | DtHAT7gr | Kotak Mahindra Bank |
| Carrie Ross      | strongsabrina@example.org  | 2024-11-10 19:30:57 | PIGYJCEQFVGEBX   | oMF6W1pJ | HDFC Bank           |
| Michael Thompson | rodgersrichard@example.net | 2024-11-10 19:30:57 | ANBXKCIMVC       | DtHAT7gr | Kotak Mahindra Bank |
+------------------+----------------------------+---------------------+------------------+----------+---------------------+
```

Relational Algebra :

π(u.name, u.email, l.date_on, f.name AS Flight_Name, bl.Offers_Code AS offers, bl.Credit_card)

(

(σ(u.userid = l.userid AND l.Flight_No = f.Flight_No AND bl.Offers_code = l.Offers_code))

(ρ(u)(Users)) ⋈ (ρ(l)(Booked_Loungue)) ⋈ (ρ(f)(Flight)) ⋈ (ρ(bl)(Loungue))

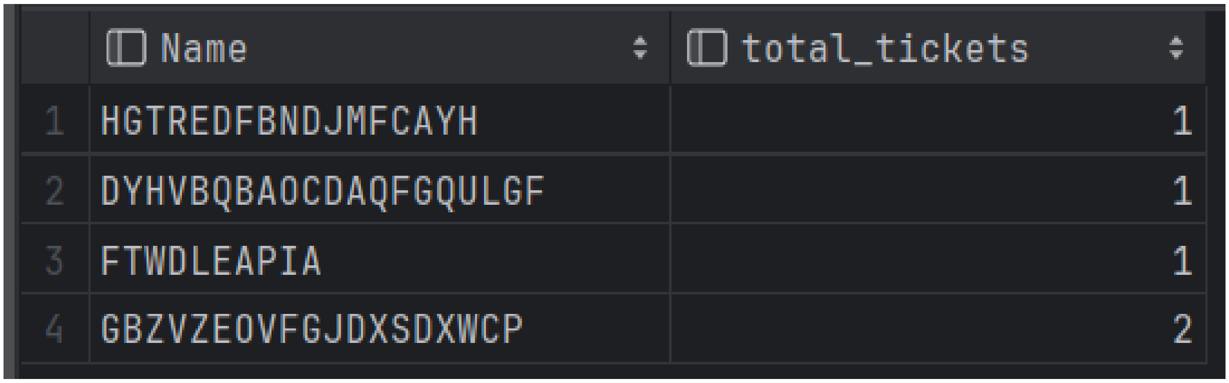
)

Query 4

Select Name , COUNT(ti.Ticket_No) as total_tickets from Trains t , Tickets ti where t.Train_No = ti.Train_No Group By T.name;

Function : displays name of the train and Count total number of tickets sold for each train.

Output:



	Name	total_tickets
1	HGTREDFBNDJMFCAYH	1
2	DYHVBQBAOCDAQFGQULGF	1
3	FTWDLEAPIA	1
4	GBZVZEOVFGJDXSDXWCP	2

Relational Algebra:
π(Name, COUNT(ti.Ticket_No) → total_tickets) (

γ(T.name)

(**Trains**) ⋈ T.Train_No = ti.Train_No(**Tickets**))

)

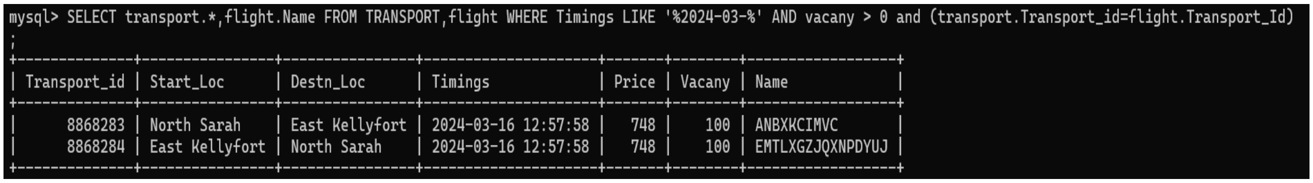
<u>Query 5</u>

SELECT transport.*,flight.Name FROM TRANSPORT,flight WHERE Timings LIKE '%2024-03-%' AND vacany > 0 and (transport.Transport_id=flight.Transport_Id) ;

Function: This query returns information about available transport options available on timings “%2024-03-%” and checks if there is a vacancy available on the transport. The query also joins the flight and transport tables based on the common transport ids columns in both.

Relational Algebra:
π(transport.*, flight.name) ((σ(Timings LIKE '%2024-03-%' AND vacany > 0)(
 (**Transport**) ⋈ (transport.Transport_id = flight.Transport_Id)(**Flight**))

Output:



```
mysql> SELECT transport.*,flight.Name FROM TRANSPORT,flight WHERE Timings LIKE '%2024-03-%' AND vacany > 0 and (transport.Transport_id=flight.Transport_Id)
;
+--------------+----------------+----------------+---------------------+-------+--------+------------------+
| Transport_id | Start_Loc      | Destn_Loc      | Timings             | Price | Vacany | Name             |
+--------------+----------------+----------------+---------------------+-------+--------+------------------+
|      8868283 | North Sarah    | East Kellyfort | 2024-03-16 12:57:58 |   748 |    100 | ANBXKCIMVC       |
|      8868284 | East Kellyfort | North Sarah    | 2024-03-16 12:57:58 |   748 |    100 | EMTLXGZJQXNPDYUJ |
+--------------+----------------+----------------+---------------------+-------+--------+------------------+
```

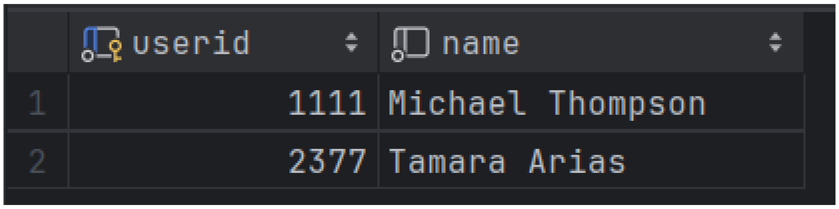
<u>Query 6</u>

select users.userid,users.name from tickets,users where tickets.userid=users.userid AND Train_No is not NULL
intersect
select users.userid,users.name from tickets,users where tickets.userid=users.userid AND Flight_No is not NULL;

Function: This query retrieves the user ID and name of those users who have booked tickets in trains and flights. It retrieves the people who have booked these tickets individually and then takes their intersection.

Relational Algebra:
(π(users.userid, users.name) ((σ(Train_No IS NOT NULL)(**tickets**)) ⋈ (tickets.userid = users.userid) (**users**)))
∩
(π(users.userid, users.name) ((σ(Flight_No IS NOT NULL)(**tickets**)) ⋈ (tickets.userid = users.userid) (**users**)))

Output:



	userid	name
1	1111	Michael Thompson
2	2377	Tamara Arias

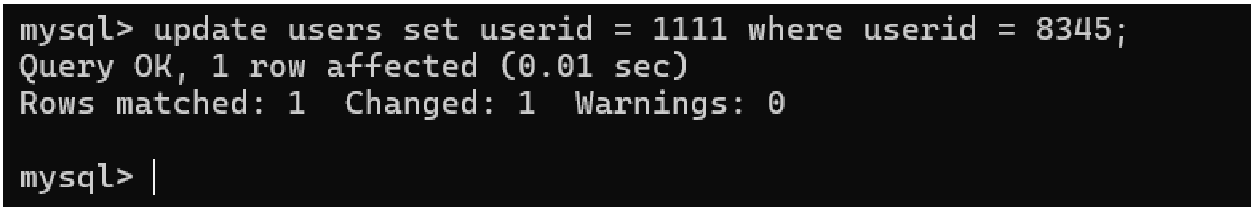
Query 7

update users set userid = 1111 where userid = 8345;

Function: The above query changes the userid of that customers who had “8345” user ID to change it to “1111”. It represents our constraint of on update cascade as given that if any user data is updated then we need to change its requested data every where , wherever it is Foreign key as associated , Clearly it Works

Relational Algebra: NA

Output:



```
mysql> update users set userid = 1111 where userid = 8345;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> |
```

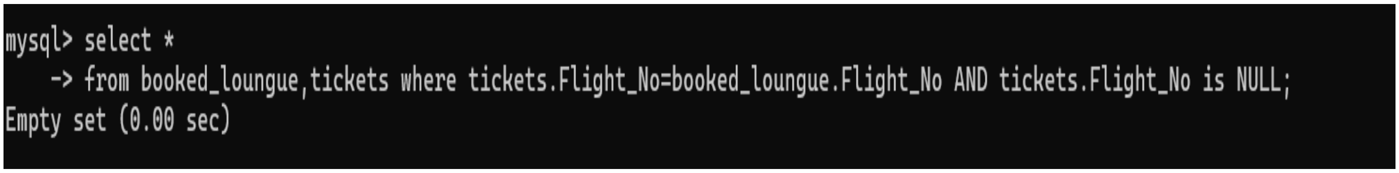
Query 8

SELECT *
FROM   booked_loungue,tickets
WHERE tickets.Flight_No=booked_loungue.Flight_No AND tickets.Flight_No is NULL;

Function: This query returns those entries from the booked_loungues, tickets table where the flight numbers match in tickets and also takes care of the cases where those entries got included where the flight number is NULL because of a product of two tables we are considering here. Here it must be Empty as there must not be any entry in booked lounges without any corresponding Flight (P.S. No passenger is allowed to use lounge without booking any flight).

Relational Algebra:
π (*) (σ (tickets.Flight_No is NULL) (**booked_loungue** ⋈ **tickets**))

Output:



```
mysql> select *
    -> from booked_loungue,tickets where tickets.Flight_No=booked_loungue.Flight_No AND tickets.Flight_No is NULL;
Empty set (0.00 sec)
```

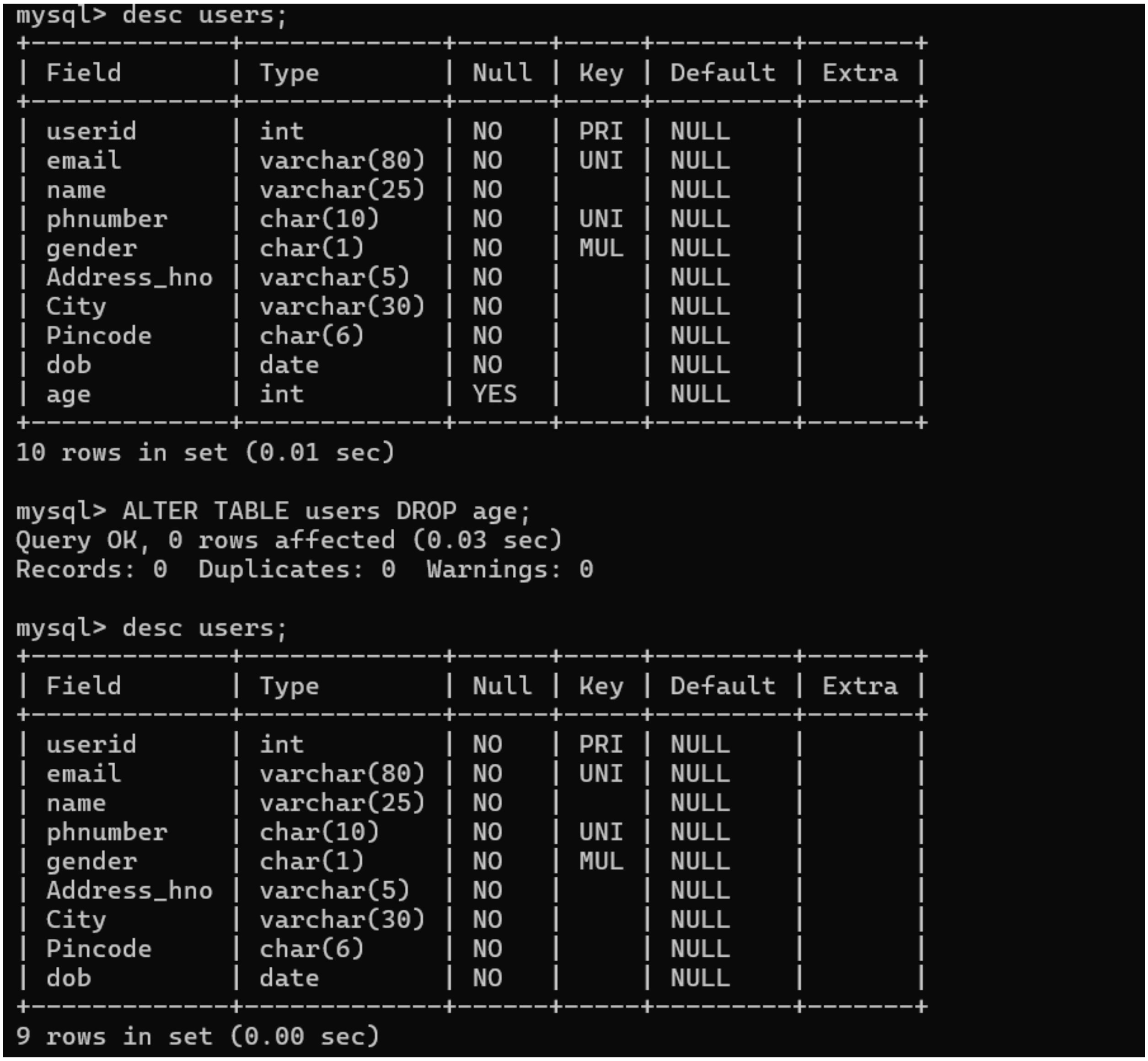
<u>Query 9</u>

ALTER TABLE users DROP age;

Please Note will result in Error if there is No column age
Function: It drops the redundant table age as we don't need this column we can directly calculate using DOB and currdate as age = YEARS(currdate-dob) :- Provided currdate >= dob

Relational Algebra: NA

Output:



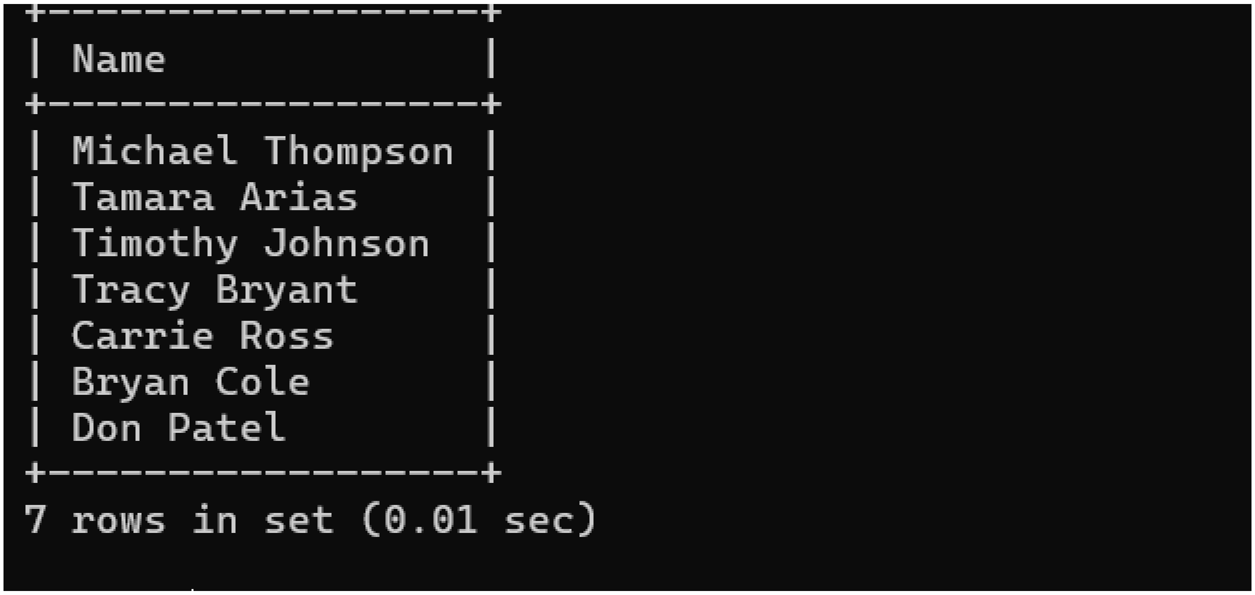
```
mysql> desc users;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| userid      | int         | NO   | PRI | NULL    |       |
| email       | varchar(80) | NO   | UNI | NULL    |       |
| name        | varchar(25) | NO   |     | NULL    |       |
| phnumber    | char(10)    | NO   | UNI | NULL    |       |
| gender      | char(1)     | NO   | MUL | NULL    |       |
| Address_hno | varchar(5)  | NO   |     | NULL    |       |
| City        | varchar(30) | NO   |     | NULL    |       |
| Pincode     | char(6)     | NO   |     | NULL    |       |
| dob         | date        | NO   |     | NULL    |       |
| age         | int         | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
10 rows in set (0.01 sec)

mysql> ALTER TABLE users DROP age;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc users;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| userid      | int         | NO   | PRI | NULL    |       |
| email       | varchar(80) | NO   | UNI | NULL    |       |
| name        | varchar(25) | NO   |     | NULL    |       |
| phnumber    | char(10)    | NO   | UNI | NULL    |       |
| gender      | char(1)     | NO   | MUL | NULL    |       |
| Address_hno | varchar(5)  | NO   |     | NULL    |       |
| City        | varchar(30) | NO   |     | NULL    |       |
| Pincode     | char(6)     | NO   |     | NULL    |       |
| dob         | date        | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
9 rows in set (0.00 sec)
```

Query 10

```
SELECT u.Name
FROM Users u
WHERE EXISTS (
        SELECT 1
        FROM Booked_Loungue l
        WHERE u.userid = l.userid
);
```

Function: Printing users who have booked lounge. It checks using exists method to retrieve only those .

Relational Algebra:
π(u.Name) (σ(EXISTS (π(1) (σ(u.userid = l.userid) (ρ(l)(**Booked_Loungue)**)))) (ρ(u)(**Users)**))



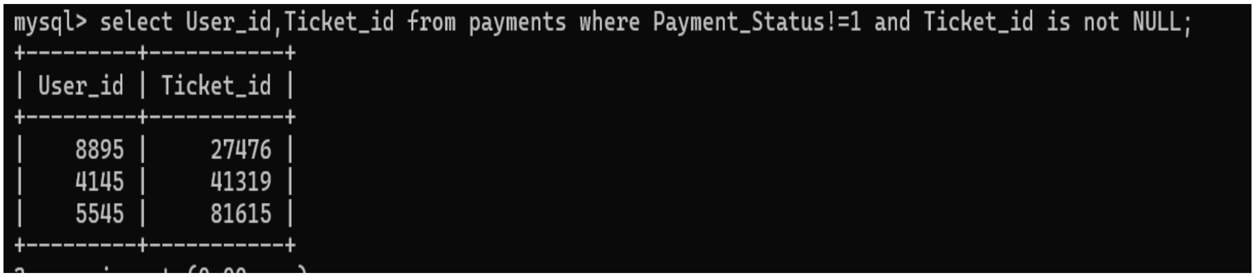
```
+------------------+
| Name             |
+------------------+
| Michael Thompson |
| Tamara Arias     |
| Timothy Johnson  |
| Tracy Bryant     |
| Carrie Ross      |
| Bryan Cole       |
| Don Patel        |
+------------------+
7 rows in set (0.01 sec)
```

Query 11

select User_id,Ticket_id from payments where Payment_Status!=1 and Ticket_id is not NULL;

Function: Printing user_id and tickets whose ticket payments are not cleared and if not cleared then they are not allowed to board the train.

Relational Algebra:
π(User_id, Ticket_id)(σ(Payment_Status ≠ 1 ∧ Ticket_id IS NOT NULL)(**Payments**))



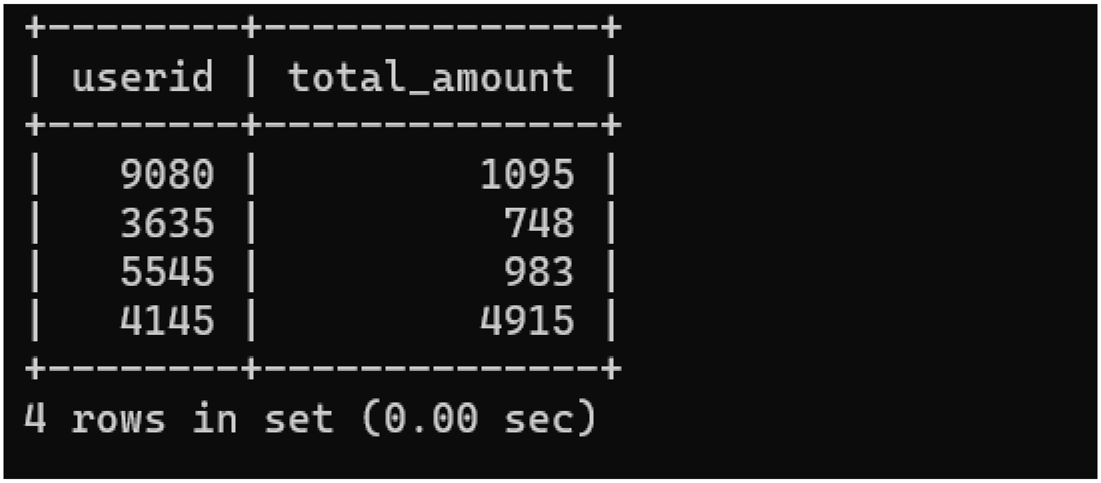
```
mysql> select User_id,Ticket_id from payments where Payment_Status!=1 and Ticket_id is not NULL;
+---------+-----------+
| User_id | Ticket_id |
+---------+-----------+
|    8895 |     27476 |
|    4145 |     41319 |
|    5545 |     81615 |
+---------+-----------+
```

Query 12

```
SELECT u.userid, SUM(t.Amount) AS total_amount
FROM Users u
JOIN Payments p ON u.userid = p.User_id
JOIN Tickets t ON p.Ticket_id = t.Ticket_No
where Payment_Status=1
GROUP BY u.userid;
```

Function : To retrieve how much each Each user spend on tickets (only valid payments)

Relational Algebra:
(γ(u.userid)
(
π(u.userid, total_amount)
(σ (Payment_Status=1)
((ρ(u)**Users**)⋈(u.userid = p.user_id)(ρ(p)**Payments**)⋈p.Ticket_id =
t.Ticket_No(ρ(T)(**Tickets**)))
)



```
+--------+--------------+
| userid | total_amount |
+--------+--------------+
|   9080 |         1095 |
|   3635 |          748 |
|   5545 |          983 |
|   4145 |         4915 |
+--------+--------------+
4 rows in set (0.00 sec)
```

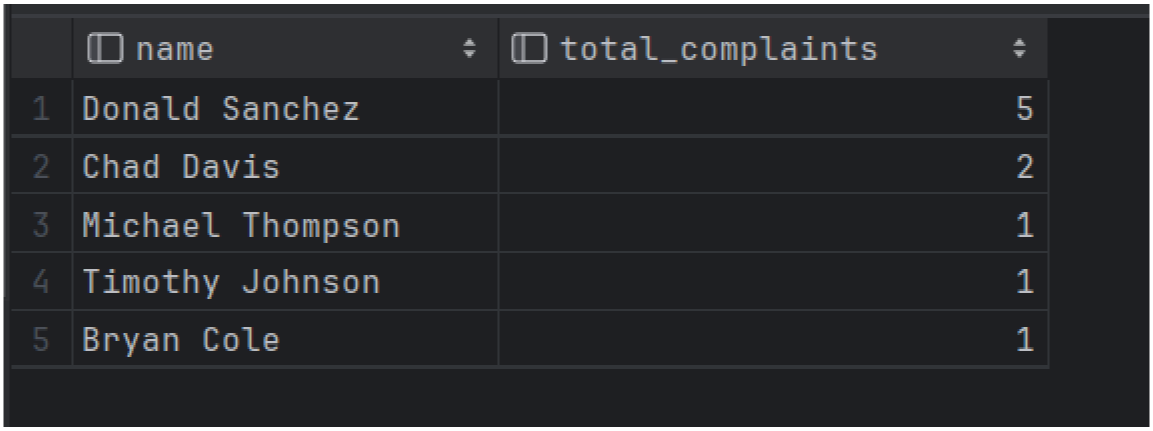
Query 13

```
SELECT  name,count(user_id) as total_complaints
FROM complaint,users
WHERE users.userid=complaint.user_id
GROUP BY user_id
ORDER BY count(user_id) desc;
```

Function: To Print each total complaints from each user by grouping them also ordering them on the basis of their complaint count that how many complaints he/she has filed.

Relational Algebra
τ {count(user_id) desc} (γ (user_id) ((π (name, count(user_id) → total_complaints))((**Complaints**) ⋈ **Users**)))

Output:



	name	total_complaints
1	Donald Sanchez	5
2	Chad Davis	2
3	Michael Thompson	1
4	Timothy Johnson	1
5	Bryan Cole	1

Relational Schema

1. User(**User_ID** , Name, Email, Phone No., Age(D.O.B , Year), Sex, Address(House No. , Address_City , Address_PIN))

2. Complaints (**Complaint_number,** *User_Id*(references))
   {Here User_Id is referenced from User Entity that cant be Null as forced Constraint of not Null as there must be a user_Id that needs to have for complaint (Foreign key).}

3. Transport(**Transport_ID** , Destn_Location , Timings , Start_Location , Price , Vacancy(Total , Filled))

4. Hotels(**Hotel_ID**, Vacancy(Total , Filled) , Pricing)

5. Train(***Transport_Id* , Train No ,** Name) { Transport_Id is a foreign key }

6. Flight(*Transport_Id (references)* **, Flight No ,** Name){ Transport_Id is a foreign key }

7. Booked_Lounge(Date,***User_Id***(references),Flight_no,Offer_code)
   { *Here User_Id is used as a foreignkey from user table to reference*}

8. Payments(**Payment_ID** , User_ID,Ticket_ID,Payment_Status):
   {User_ID and Ticket_ID is a foreign key}

9. Lounge(**Offers_code**, Timings, Place, Credit card accepted)

10. Tickets(**Ticket_no** , date_of_journey , quantity , Amount , Train_no, Flight_no, user_id)
    {Train_no is a foreign key and user_id is a foreign key,Flight_no is a foreign key}

11. Hotel_Invoice(Date , Hotel_ID) {Hotel_ID is a foreign key here}

12. Holiday_Package(Package_Id (Primary Key) , Ticket_no , Hotel_Id , Time_Period,Discounts)
13. Gender_ref (Gender Primary key)
14. Holiday_pay (package_id,Payment_id) Stores as relationship

Contributions 👏

1) Aditya Upadhyay (2022040) : - Written Sql queries, Helped in making Pdf, written relational algebra
2) Keshav Chhabra(2022247):- Written Sql queries, Helped in making Pdf, written relational algebra
3) Mann Nariya (2022278) :- Written Sql queries, Helped in making Pdf, written relational algebra
4) Dev Utkarsh (2022150) :- Written Sql queries, Helped in making Pdf, written relational algebra