Database Management and Database Design

Final Project

AIRBNB Booking and Renting Database

Name : Sanika Sambhaji Bhagat

NUID : 001889030

About AIRBNB :

- AirBNB is a rental and hospitality website that is used all over the world in more than 191 countries for short-term lodging such as vacations.
- The company that was founded in 2008 has a lot of customers who rent the houses of the hosts who put up their houses on rent.
- This means that AIRBNB does not own any houses, it just acts as a broker between the customer and the hosts.
- AirBNB believes in connecting people all over the world and strives to provide excellent customer service.

Why AIRBNB ?

- The purpose of choosing AIRBNB database is to dig deeper into learning how a hospitality enterprise functions as AIRBNB has revolutionized lodging and travelling.
- It can serve as a blueprint for a lot of other industries which function on the principle of renting instead of ownership.

- This business model is the future and it will be intriguing to learn about the data that the AirBNB needs to store in their database so that it is used in meaningful ways to best serve the people associated with the enterprise.

How Does AIRBNB Work?

- AIRBNB has a number of users which are either customers or hosts.
- The customers select the location of their choice, the dates of their choice choice and book one of the houses that are listed.
- Upon booking, the customer makes the payment out of which part of the payment goes to the AIRBNB enterprise and the other part goes to the Host whose house is been booked by the Customer.
- Customers can book a house, cancel a booking and if a booking is cancelled then the payment is refunded.
- Hosts can put up any number of houses on rent and can take them of anytime.

Business Rules

As a User :

- User makes an account on the AIRBNB Website
- The User can then decide whether he wants to become a Customer to rent the houses or he wants to become the Host to put up his house on the rent.
- One User can either be a Customer, Host or both Customer and a Host.
- The Customer can make any number of bookings under his account.
- The Host can put up any number of houses on rent under his account.
- The user (both Customer and Host) can give feedbacks to the AIRBNB Enterprise.
- The Customer can rate the Host upon his experience and the Host can rate the Customer upon his experience.
- The Customer can review the house he had rented.
- The person can deactivate or activate his account any number of times.

As an Enterprise :

- The AirBNB receive feedbacks from their customers and hosts. The feedback can be regarding the services, the website or the enterprise.

- The AirBNB can view all the statistics such as the total number of customers, total number of hosts, total number of houses, total number of houses in each locations, customer who has the highest number of bookings, host who has the maximum number of houses up for rent, host who has received good ratings, the customers who have received bad ratings and so on.

- AIRBNB maintains a backup of all the data that it has in its database, be it regarding the person information, the booking information or the payment information.

- Different employees of AIRBNB will have different levels of access to the database.

Other Business Rules :

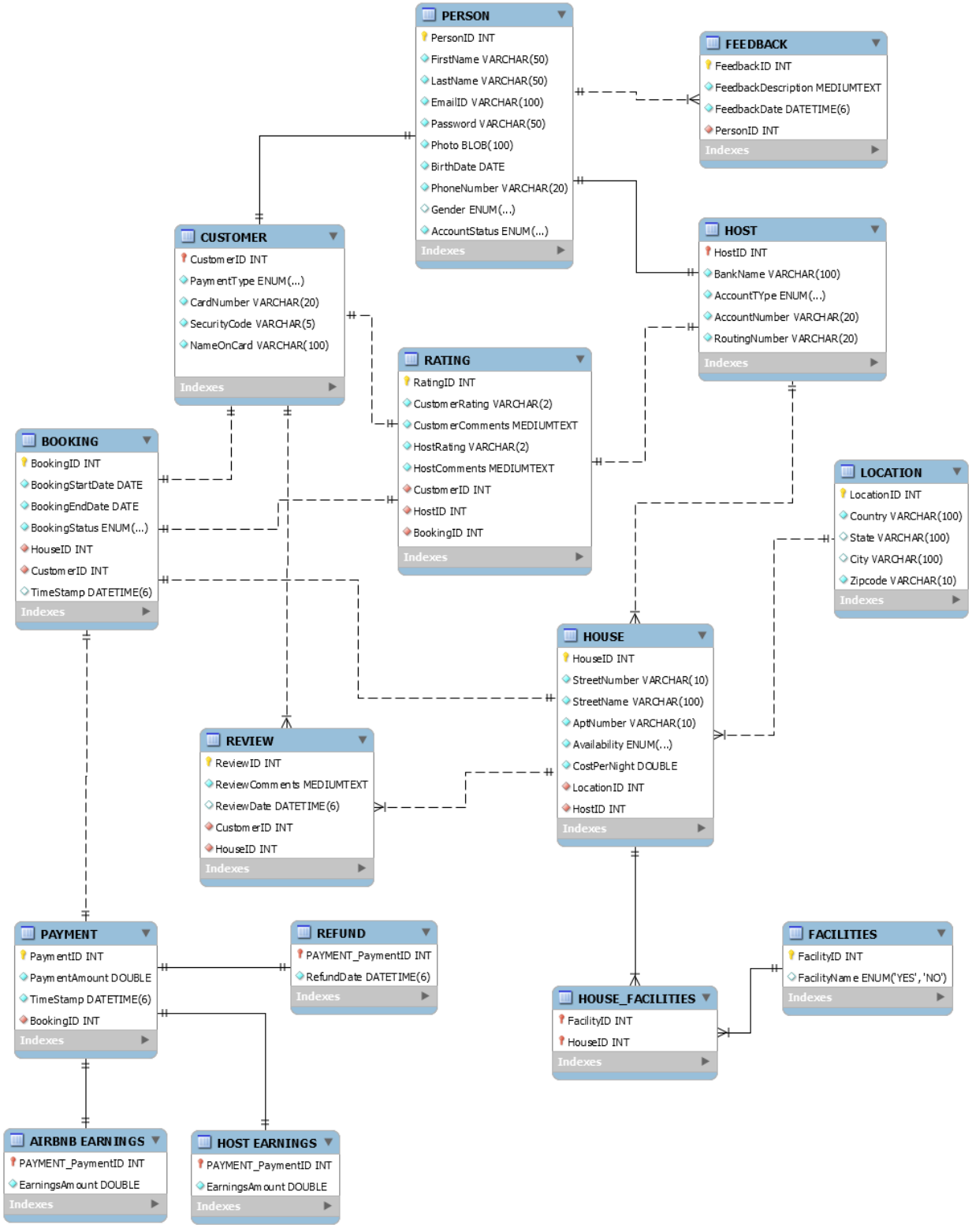
- If a person is a customer and wants to delete his account, then the AIRBNB should check whether the customer has any forthcoming bookings under his name. If he does, then he would not be able to delete his account.

- If a person is a host and wants to deactivate his account, then the AIRBNB should automatically delete the house listings so that no one can book the houses whose host does not exist.
- If a booking is made, then the payment table should automatically be populated and the earnings of the Enterprise and the Host should be segregated.

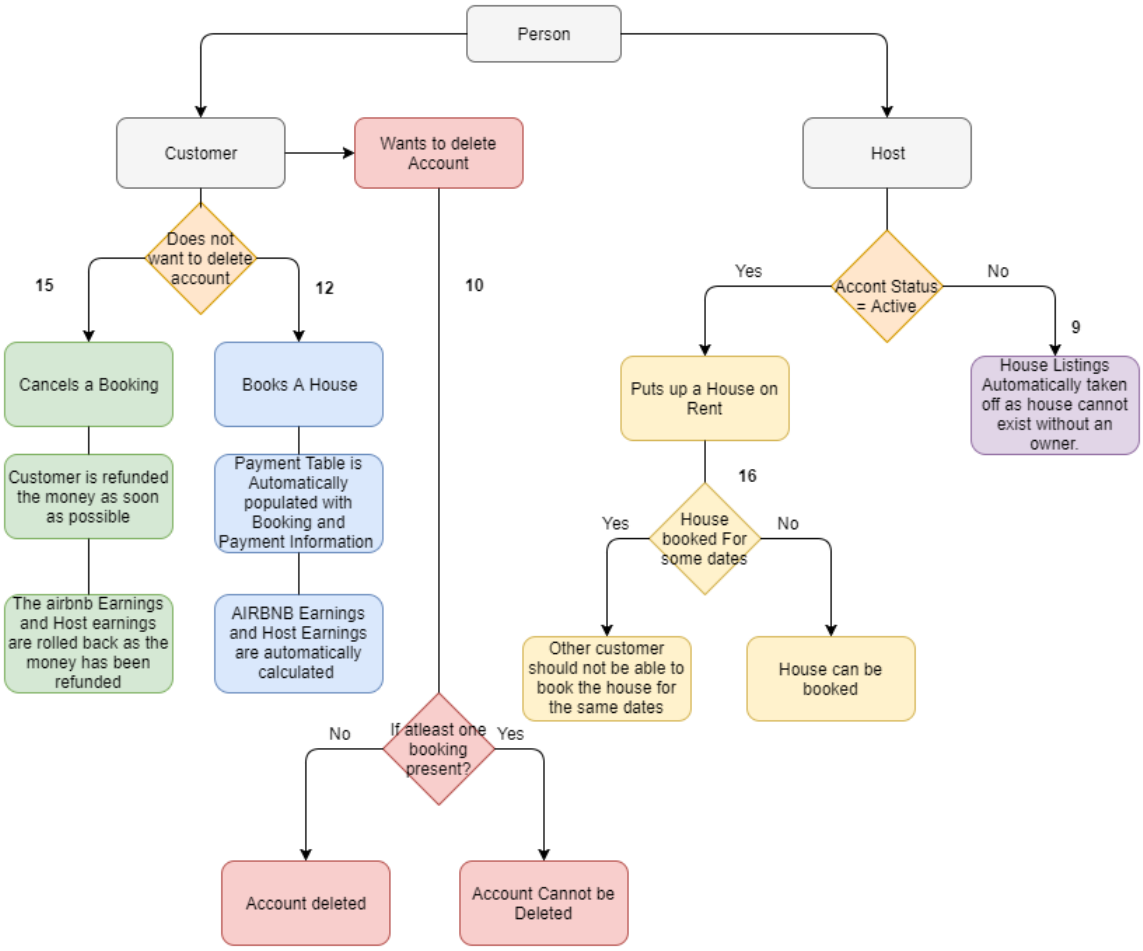
- If the booking is cancelled by the customer the payment made by the customer should be refunded immediately.

- If a house has been booked by the customer for a range of days, then some other customer should not be able to book the same house within that range.

ER Diagram :

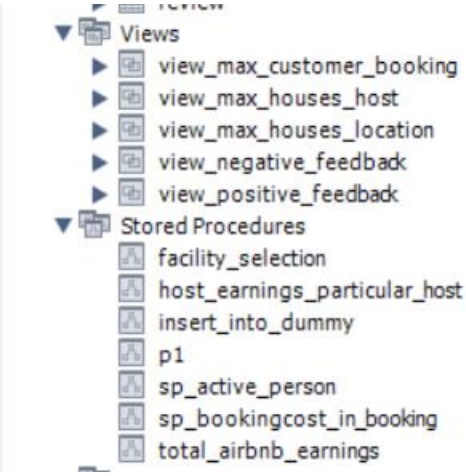

PERSON
PersonID INT
FirstName VARCHAR(50)
LastName VARCHAR(50)
EmailID VARCHAR(100)
Password VARCHAR(50)
Photo BLOB(100)
BirthDate DATE
PhoneNumber VARCHAR(20)
Gender ENUM(...)
AccountStatus ENUM(...)
Indexes
FEEDBACK
FeedbackID INT
FeedbackDescription MEDIUMTEXT
FeedbackDate DATETIME(6)
PersonID INT
Indexes
CUSTOMER
CustomerID INT
PaymentType ENUM(...)
CardNumber VARCHAR(20)
SecurityCode VARCHAR(5)
NameOnCard VARCHAR(100)
Indexes
HOST
HostID INT
BankName VARCHAR(100)
AccountTYpe ENUM(...)
AccountNumber VARCHAR(20)
RoutingNumber VARCHAR(20)
Indexes
RATING
RatingID INT
CustomerRating VARCHAR(2)
CustomerComments MEDIUMTEXT
HostRating VARCHAR(2)
HostComments MEDIUMTEXT
CustomerID INT
HostID INT
BookingID INT
Indexes
BOOKING
BookingID INT
BookingStartDate DATE
BookingEndDate DATE
BookingStatus ENUM(...)
HouseID INT
CustomerID INT
TimeStamp DATETIME(6)
Indexes
LOCATION
LocationID INT
Country VARCHAR(100)
State VARCHAR(100)
City VARCHAR(100)
Zipcode VARCHAR(10)
Indexes
HOUSE
HouseID INT
StreetNumber VARCHAR(10)
StreetName VARCHAR(100)
AptNumber VARCHAR(10)
Availability ENUM(...)
CostPerNight DOUBLE
LocationID INT
HostID INT
Indexes
REVIEW
ReviewID INT
ReviewComments MEDIUMTEXT
ReviewDate DATETIME(6)
CustomerID INT
HouseID INT
Indexes
PAYMENT
PaymentID INT
PaymentAmount DOUBLE
TimeStamp DATETIME(6)
BookingID INT
Indexes
REFUND
PAYMENT_PaymentID INT
RefundDate DATETIME(6)
Indexes
FACILITIES
FacilityID INT
FacilityName ENUM('YES','NO')
Indexes
HOUSE_FACILITIES
FacilityID INT
HouseID INT
Indexes
AIRBNB EARNINGS
PAYMENT_PaymentID INT
EarningsAmount DOUBLE
Indexes
HOST EARNINGS
PAYMENT_PaymentID INT
EarningsAmount DOUBLE
Indexes

Workflow :



List Of Tables :

List of Stored Procedures And Views :



List Of Triggers :

- Booking_validation (booking)
- Refund_policy (Booking)
- Insert_into_paymenttable (booking_dummy)
- Airbnb_trigger (payment)
- Host_earnings_trigger (payment)
- Delete_house_of_host (person)
- Delete_customer_without_booking (person)

- Create_person_record_logs (person)
- Update_person_record (person)
- Delete Airbnb_host_earnings (refund)

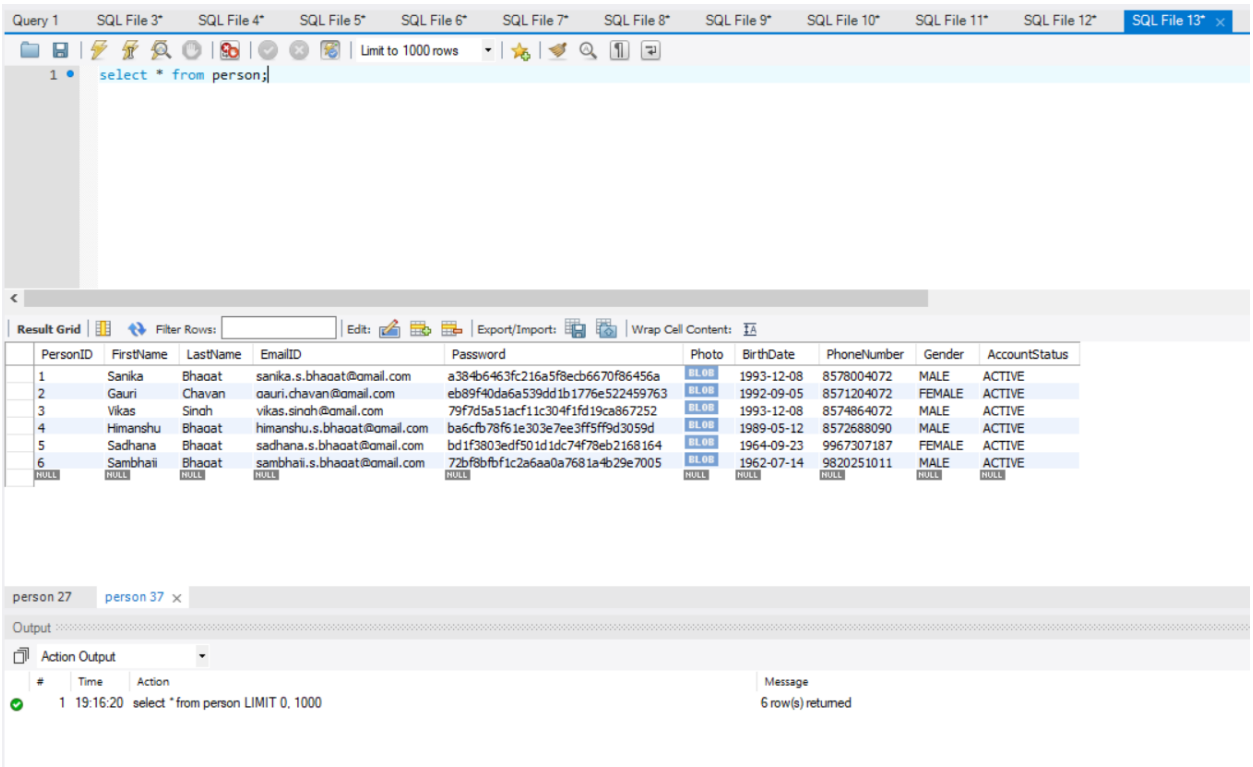
Scenarios :

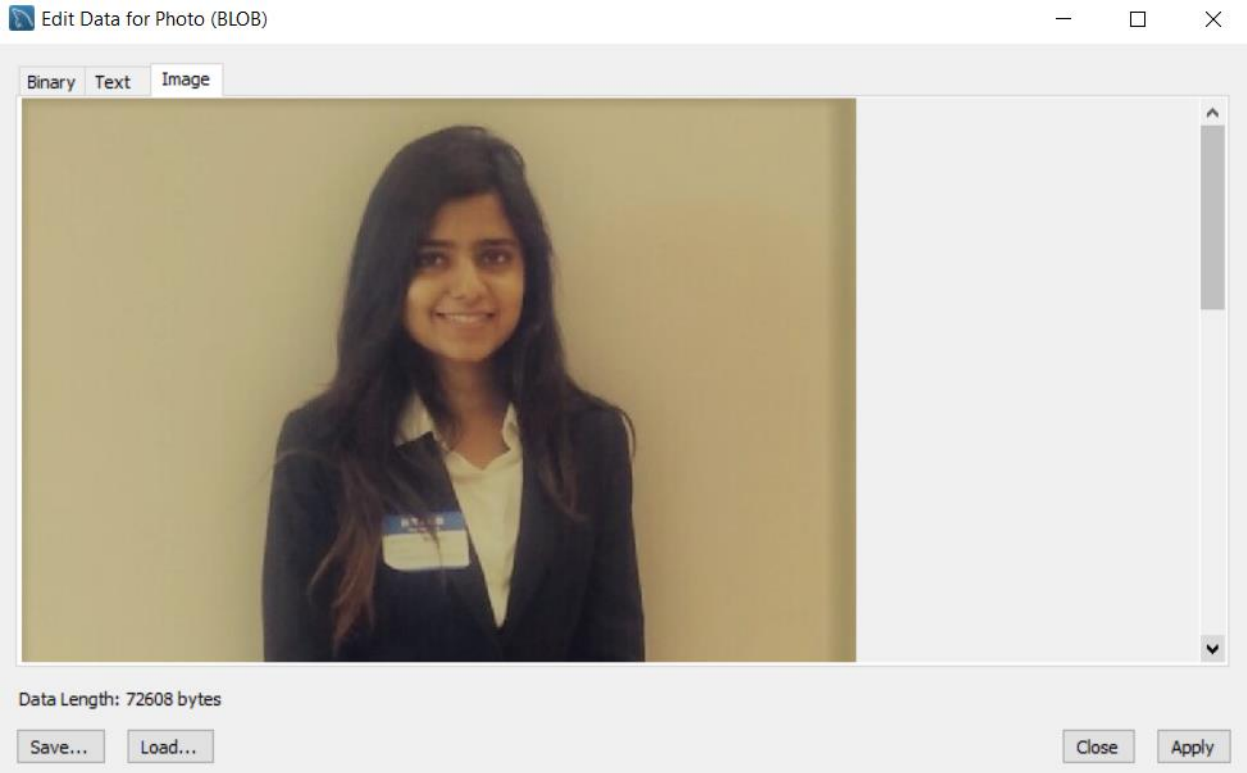
Scenario 1 :

To check the way the encryption and image is stored in the DB.

Query :

Select * from person;



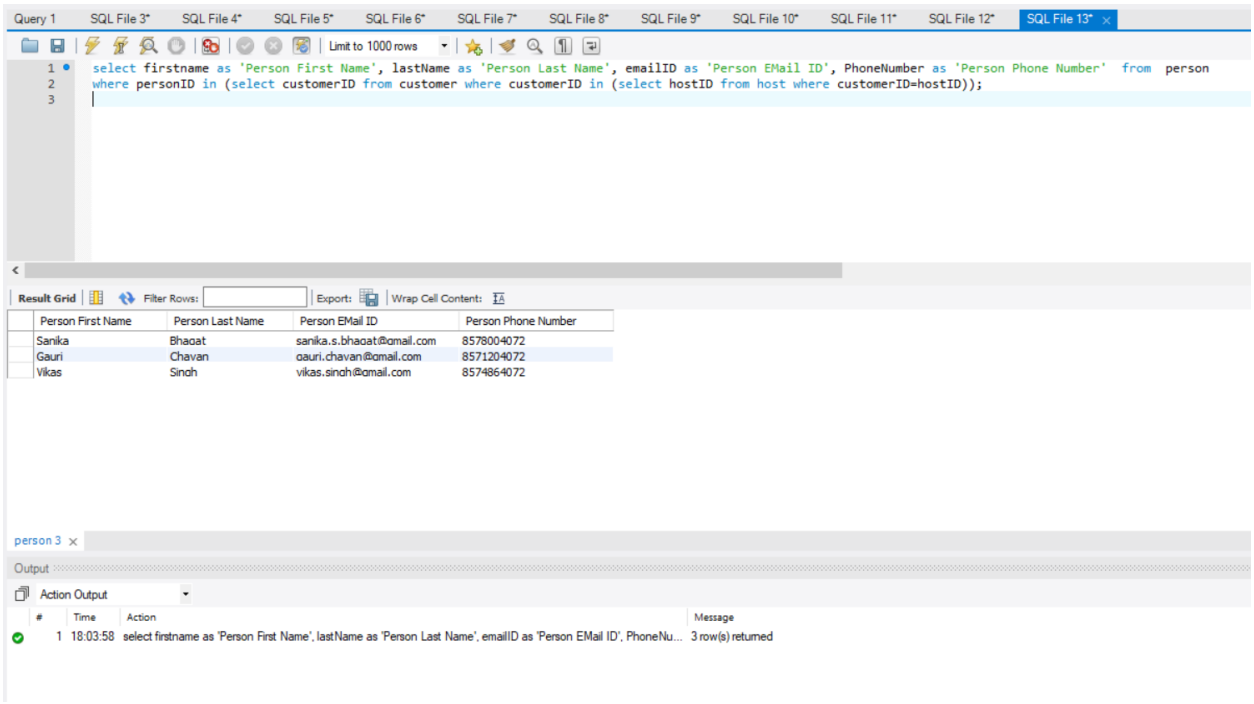


Scenario 2:

Check how many Users are both Customers and Hosts with their contact information

Query :

```
select firstname as 'Person First Name', lastName as 'Person Last Name', emailID as 'Person Email ID', phoneNumber as 'Person Phone Number' from  person
where personID in (select customerID from customer where customerID in (select hostID from host where customerID=hostID));
```



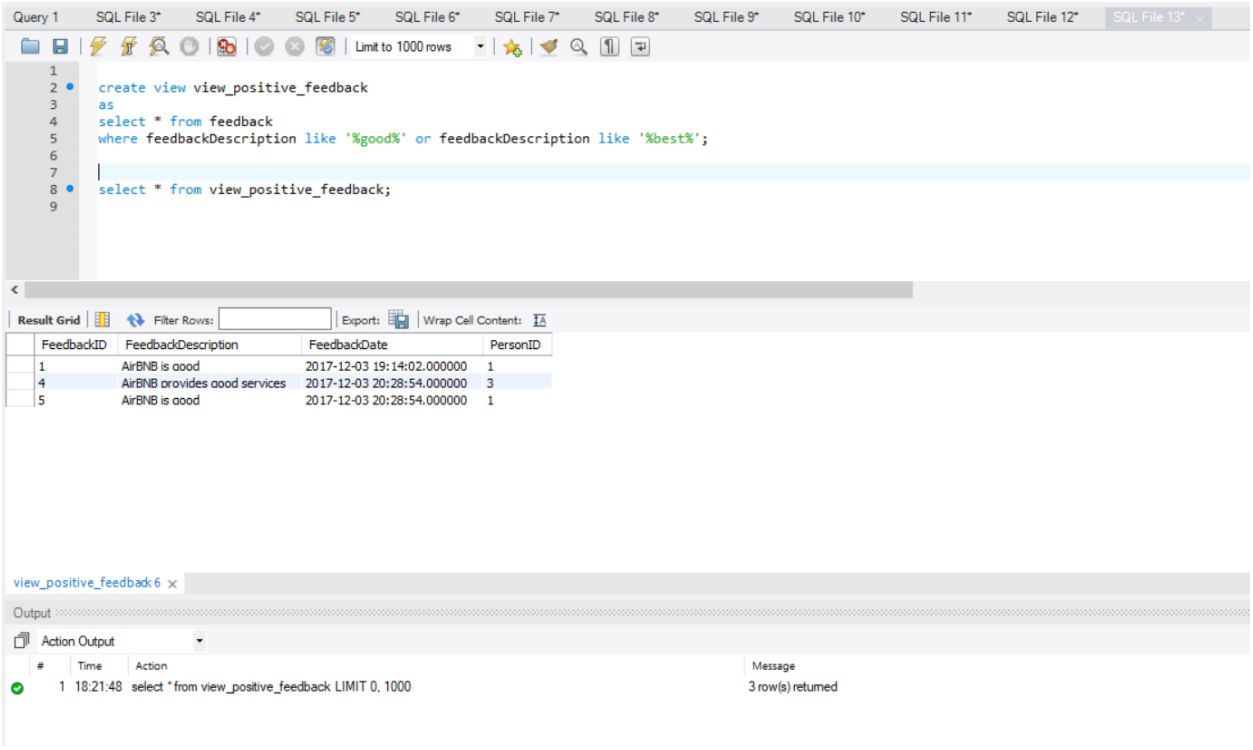
Scenario 3 :

Find out the number of positive feedbacks received by AirBNB Enterprise.

Query :

```
create view view_positive_feedback
as
select * from feedback
where feedbackDescription like '%good%' or feedbackDescription like '%best%';
```

```
select * from view_positive_feedback;
```



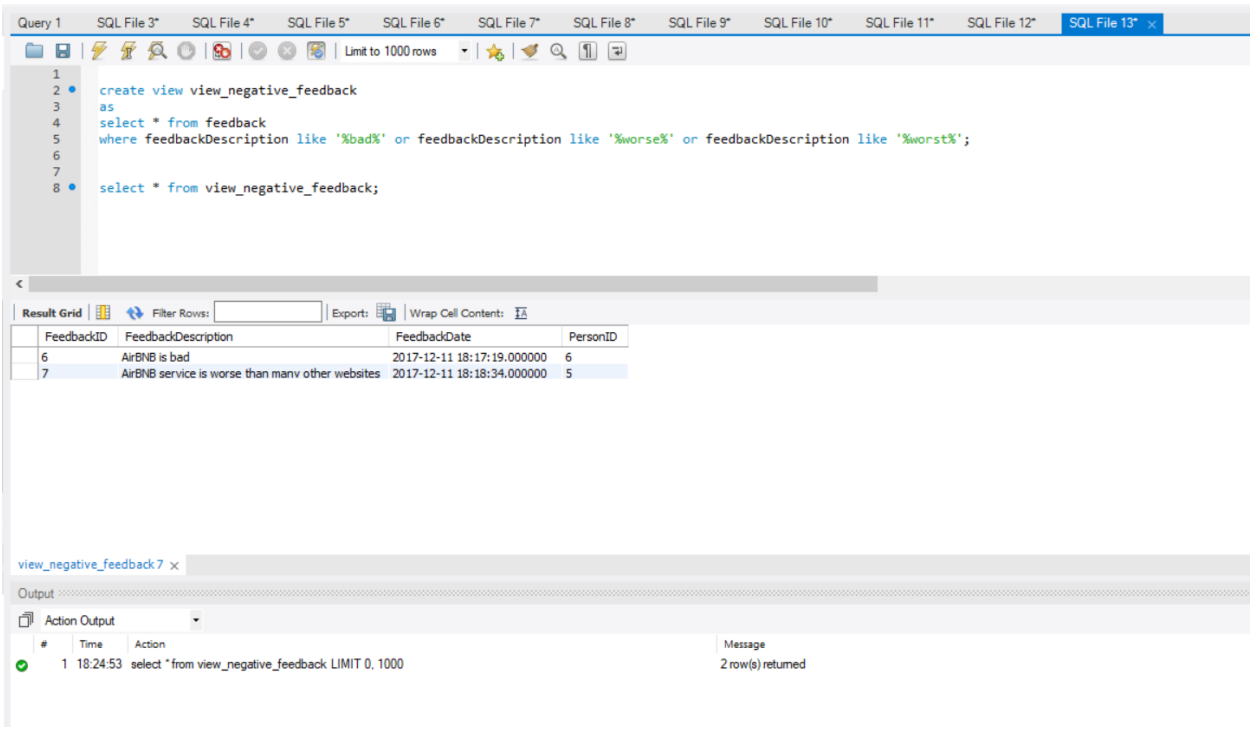
Scenario 4 :

Find out the number of negative feedback received by AirBNB Enterprise.

Query:

```
create view view_negative_feedback
as
select * from feedback
where feedbackDescription like '%bad%' or feedbackDescription like '%worse%';
```

```
select * from view_negative_feedback;
```

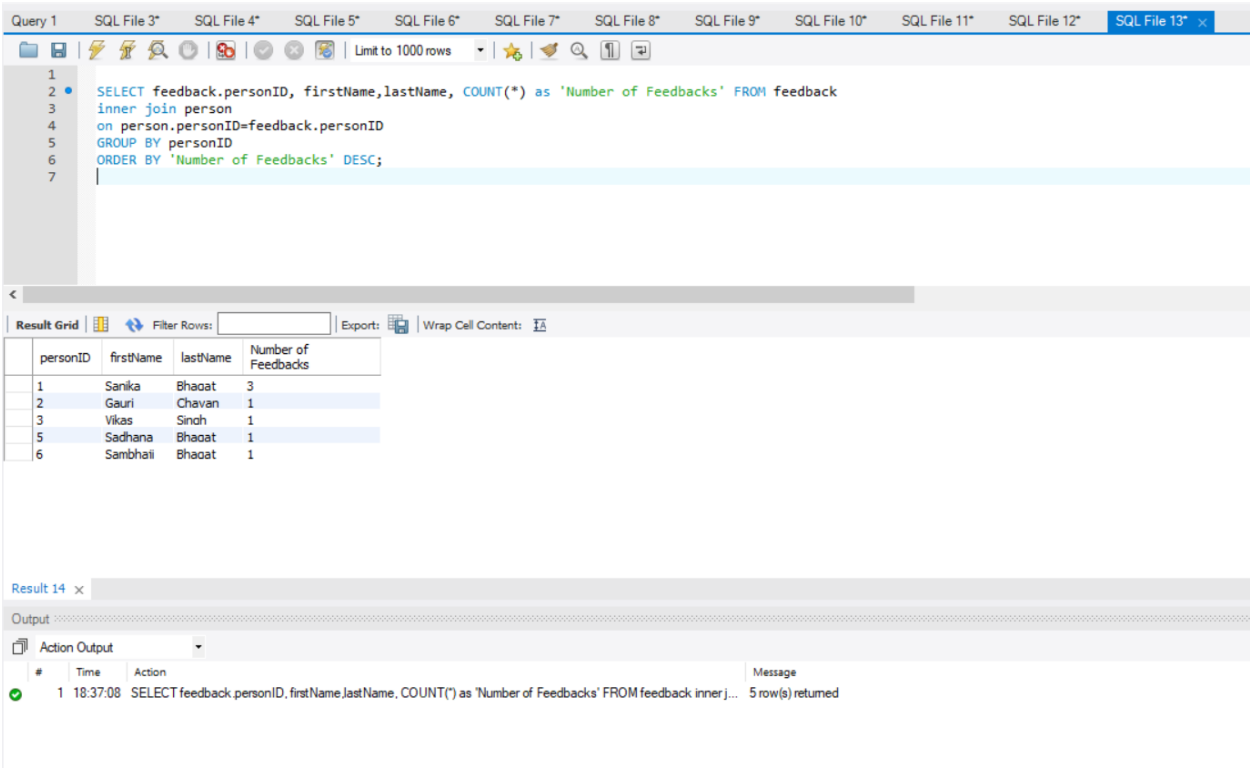


Scenario 5 :

Find the Number of feedbacks given by each person to the AIRBNB enterprise.

Query :

```
SELECT feedback.personID, firstName,lastName, COUNT(*) as 'Number of Feedbacks' FROM feedback
inner join person
on person.personID=feedback.personID
GROUP BY personID
ORDER BY 'Number of Feedbacks' DESC;
```



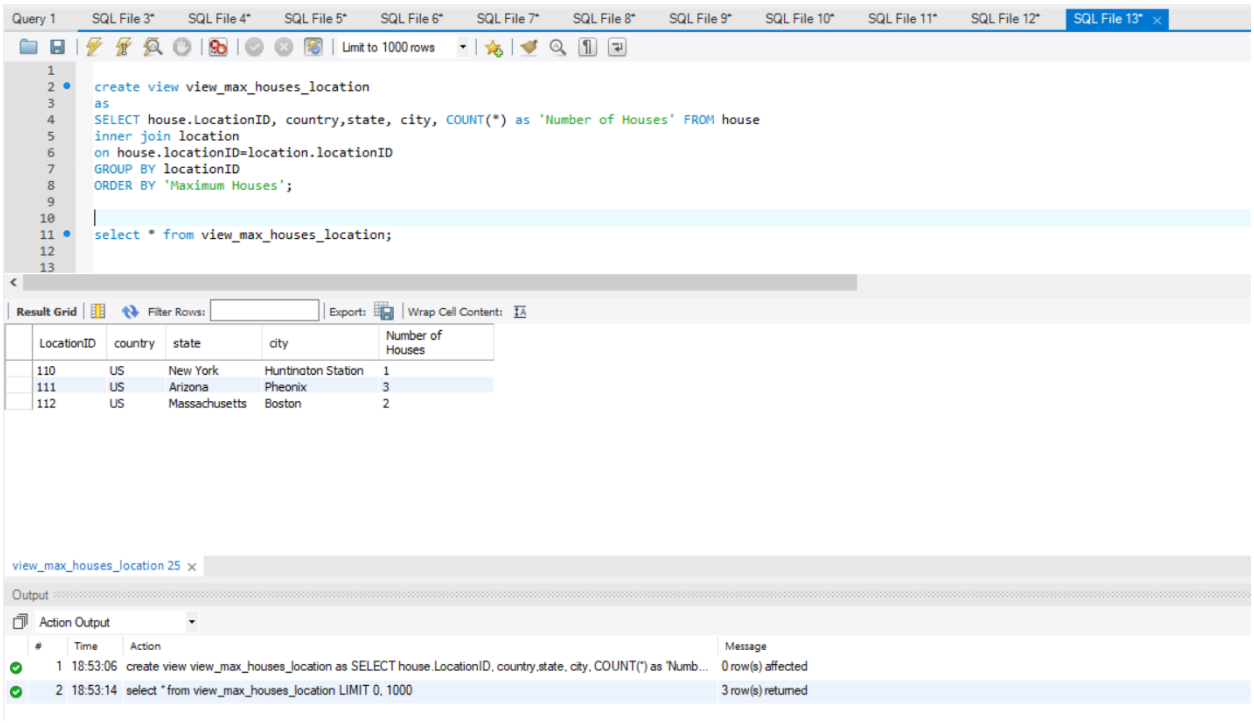
Scenario 6 :

To find the number of houses in each location.

Query:

```
create view view_max_houses_location
as
SELECT house.LocationID, country,state, city, COUNT(*) as 'Number of Houses' FROM house
inner join location
on house.locationID=location.locationID
GROUP BY locationID
ORDER BY 'Maximum Houses';
```

```
select * from view_max_houses_location;
```



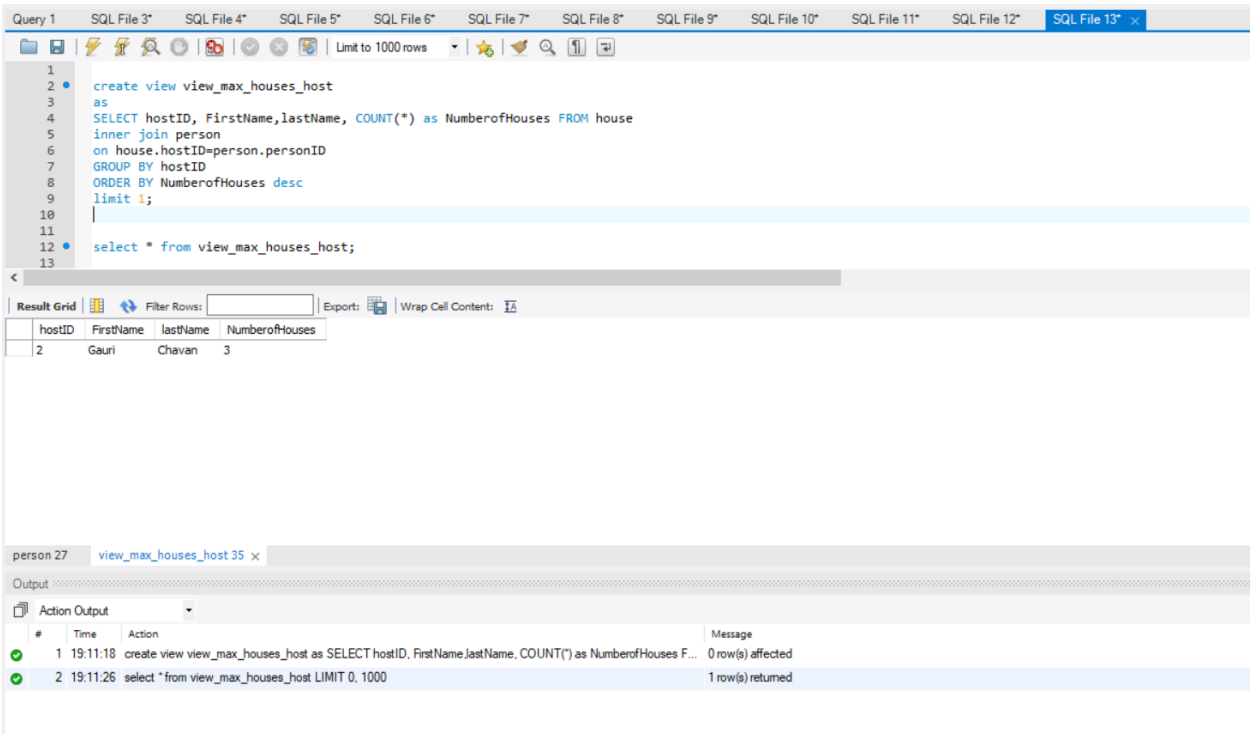
Scenario 7 :

Find the host who has maximum number of houses put up on rent

Query :

```
create view view_max_houses_host
as
SELECT hostID, FirstName,lastName, COUNT(*) as NumberofHouses FROM house
inner join person
on house.hostID=person.personID
GROUP BY hostID
ORDER BY NumberofHouses desc
limit 1;
```

```
select * from view_max_houses_host;
```



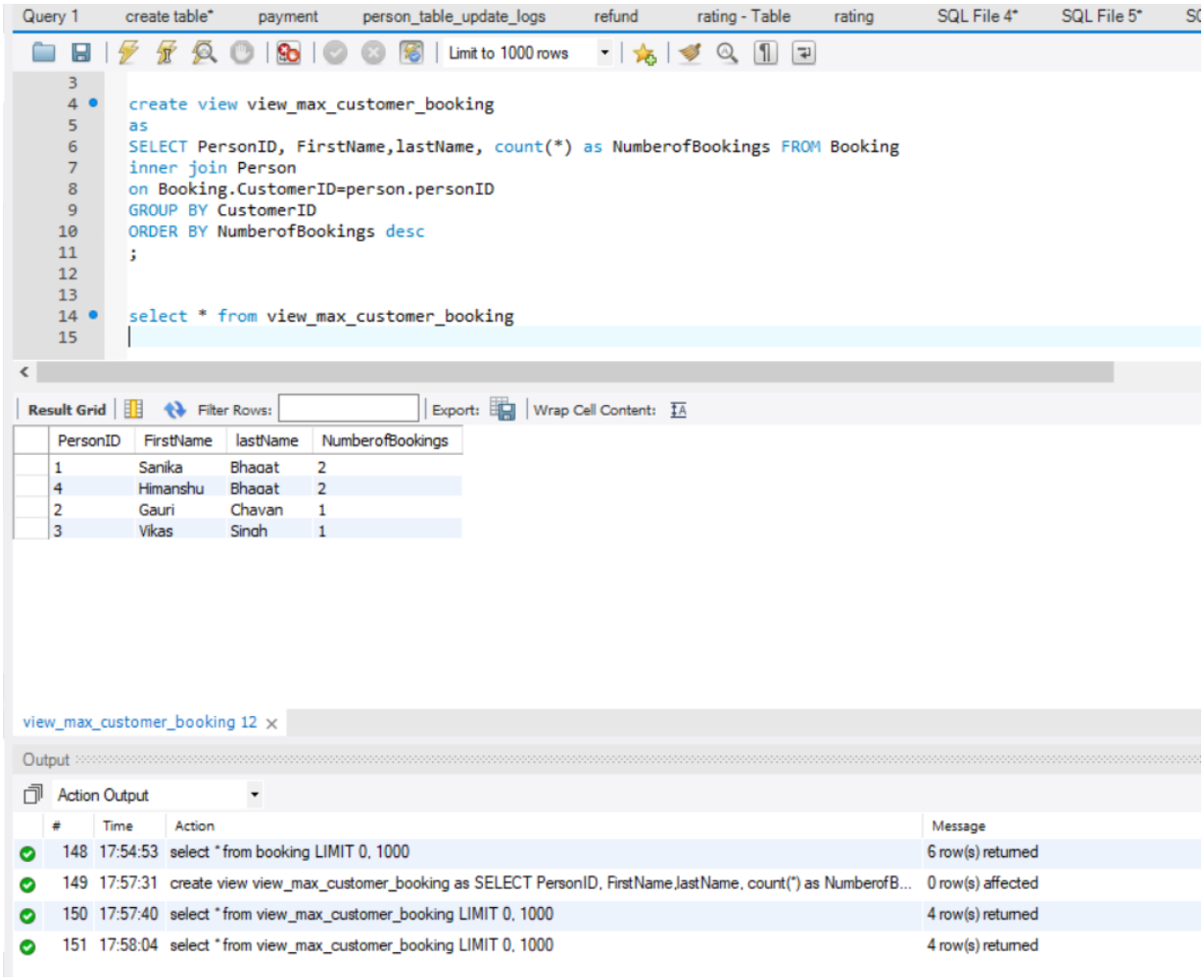
Scenario 8 :

Find the Customer who has the highest number of bookings

Query :

```
create view view_max_customer_booking
as
SELECT PersonID, FirstName,lastName, count(*) as NumberofBookings FROM Booking
inner join Person
on Booking.CustomerID=person.personID
GROUP BY CustomerID
ORDER BY NumberofBookings desc;
```

```
select * from view_max_customer_booking;
```



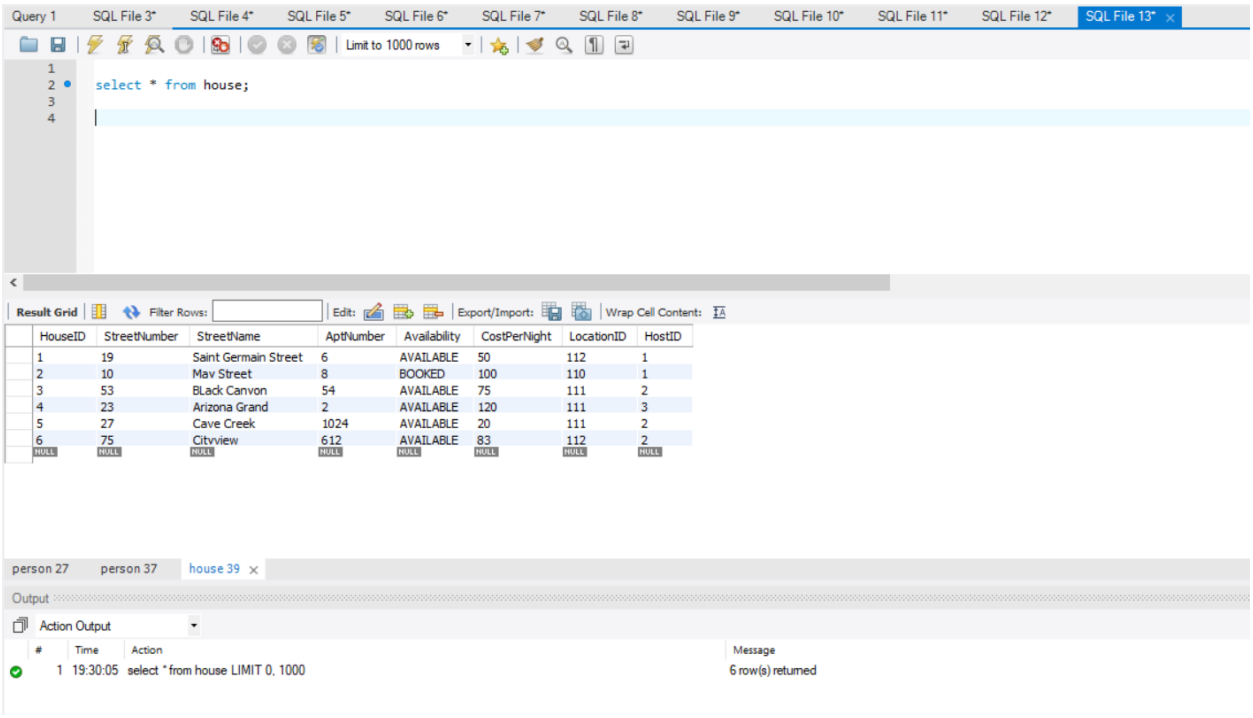
Scenario 9 :

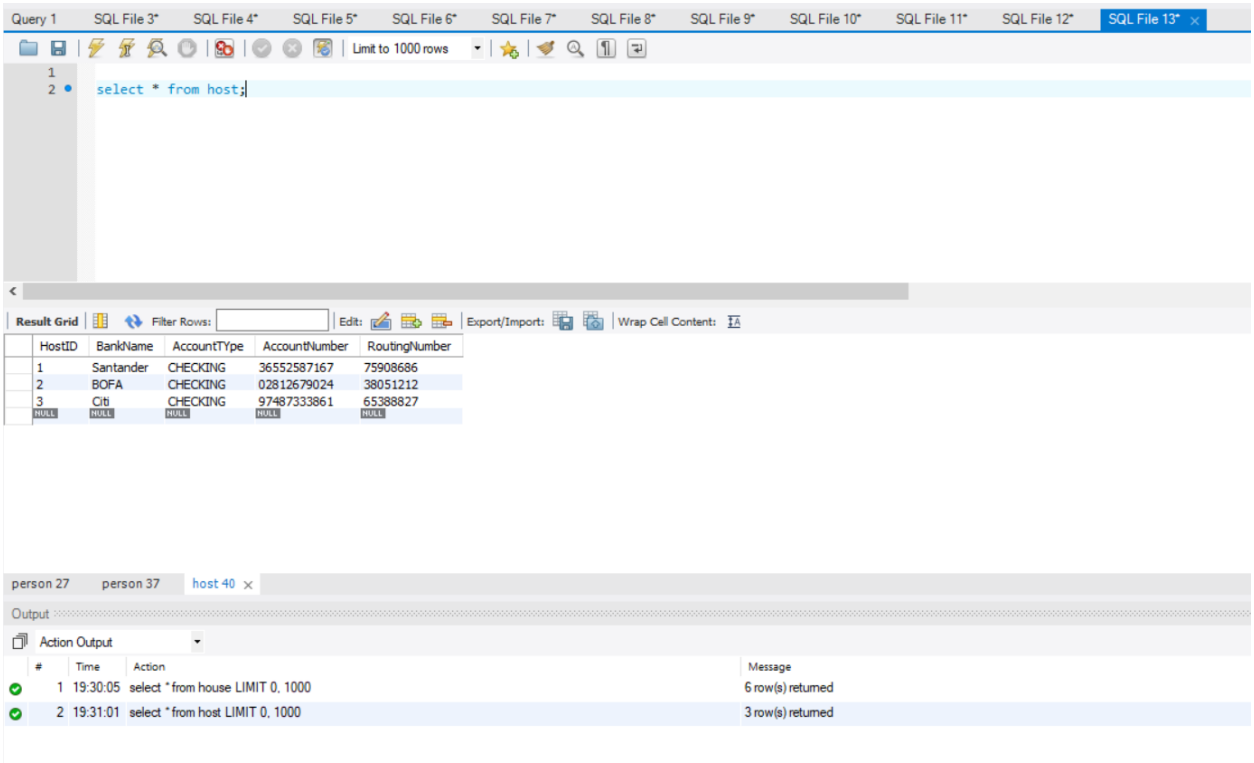
If a person deactivates his account and if the person is a host, then the house listing posted by the host should be deleted.

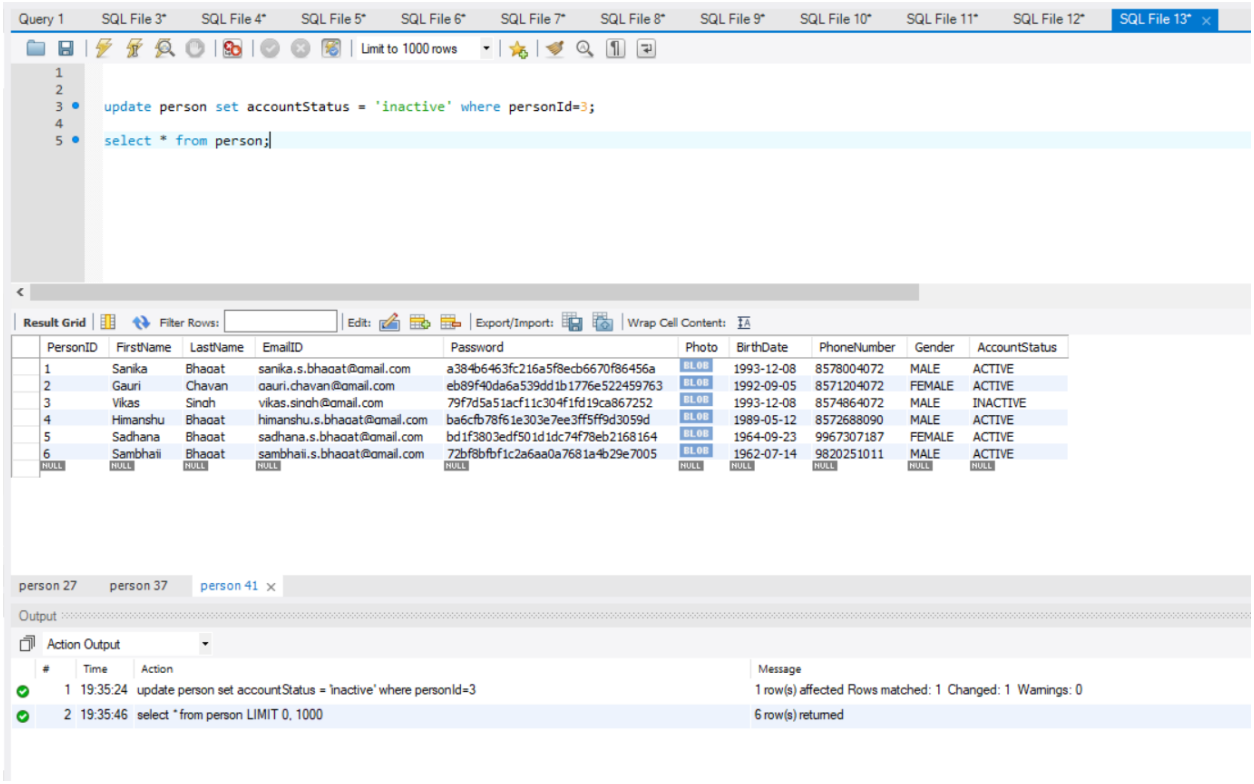
Query :

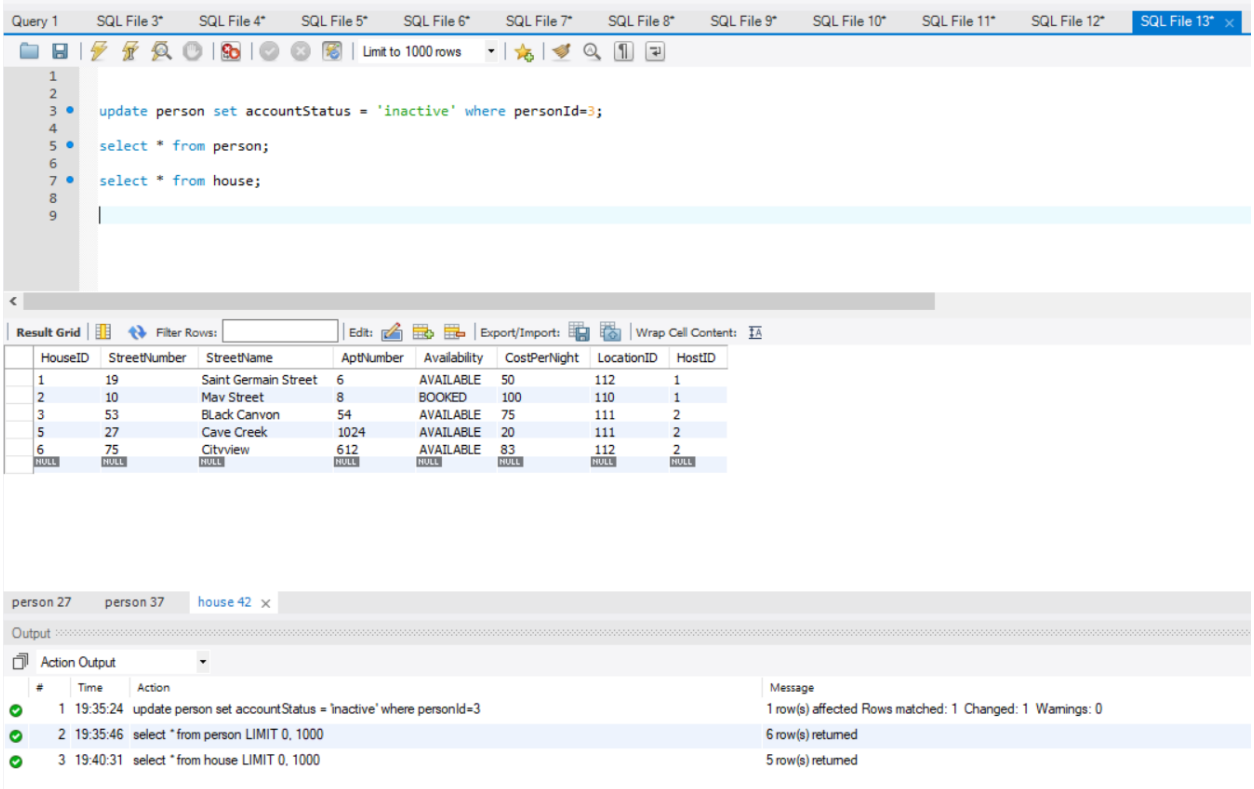
```
delimiter //
create trigger delete_house_of_host
after update
on person
for each row
begin
if new.accountStatus= 'inactive'
then
delete from house where house.hostID=new.personID;
end if;
end
//
```

```
select * from house;
select * from person;
update person set accountStatus = 'inactive' where personId=3;
select * from person;
select * from house;
```






Query 1 SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13*
Limit to 1000 rows
update person set accountStatus = 'inactive' where personId=3;
select * from person;
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
PersonID FirstName LastName EmailID Password Photo BirthDate PhoneNumber Gender AccountStatus
1 Sanika Bhagat sanika.s.bhagat@gmail.com a384b6463fc216a5f8ecb6670f86456a BLOB 1993-12-08 8578004072 MALE ACTIVE
2 Gauri Chavan gauri.chavan@gmail.com eb89f40da6a539dd1b1776e522459763 BLOB 1992-09-05 8571204072 FEMALE ACTIVE
3 Vikas Singh vikas.singh@gmail.com 79f7d5a51acf11c304f1fd19ca867252 BLOB 1993-12-08 8574864072 MALE INACTIVE
4 Himanshu Bhagat himanshu.s.bhagat@gmail.com ba6cfb78f61e303e7ee3ff5ff9d3059d BLOB 1989-05-12 8572688090 MALE ACTIVE
5 Sadhana Bhagat sadhana.s.bhagat@gmail.com bd1f3803edf501d1dc74f78eb2168164 BLOB 1964-09-23 9967307187 FEMALE ACTIVE
6 Sambhaji Bhagat sambhaji.s.bhagat@gmail.com 72bf8bfbf1c2a6aa0a7681a4b29e7005 BLOB 1962-07-14 9820251011 MALE ACTIVE
person 27 person 37 person 41
Output
Action Output
Time Action Message
1 19:35:24 update person set accountStatus = 'inactive' where personId=3 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
2 19:35:46 select * from person LIMIT 0, 1000 6 row(s) returned


Query 1 SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13*
Limit to 1000 rows
update person set accountStatus = 'inactive' where personId=3;
select * from person;
select * from house;
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
HouseID StreetNumber StreetName AptNumber Availability CostPerNight LocationID HostID
1 19 Saint Germain Street 6 AVAILABLE 50 112 1
2 10 Mav Street 8 BOOKED 100 110 1
3 53 BLack Canvon 54 AVAILABLE 75 111 2
5 27 Cave Creek 1024 AVAILABLE 20 111 2
6 75 Citvview 612 AVAILABLE 83 112 2
person 27 person 37 house 42
Output
Action Output
Time Action Message
1 19:35:24 update person set accountStatus = 'inactive' where personId=3 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
2 19:35:46 select * from person LIMIT 0, 1000 6 row(s) returned
3 19:40:31 select * from house LIMIT 0, 1000 5 row(s) returned

Scenario 10 :

If a person wants to delete his account and the person is a customer, then he should not be able to delete his account if he has made atleast one booking .

Query :

```
delimiter //
create trigger delete_customer_without_booking
after update
on person
for each row
begin
if new.accountStatus= 'inactive'
then
if (new.personID in (Select customerID from booking))
then
insert into message values ('Cannot delete customer');
end if;
else
delete from booking where booking.customerID=new.personID;
end if;
end
//
```

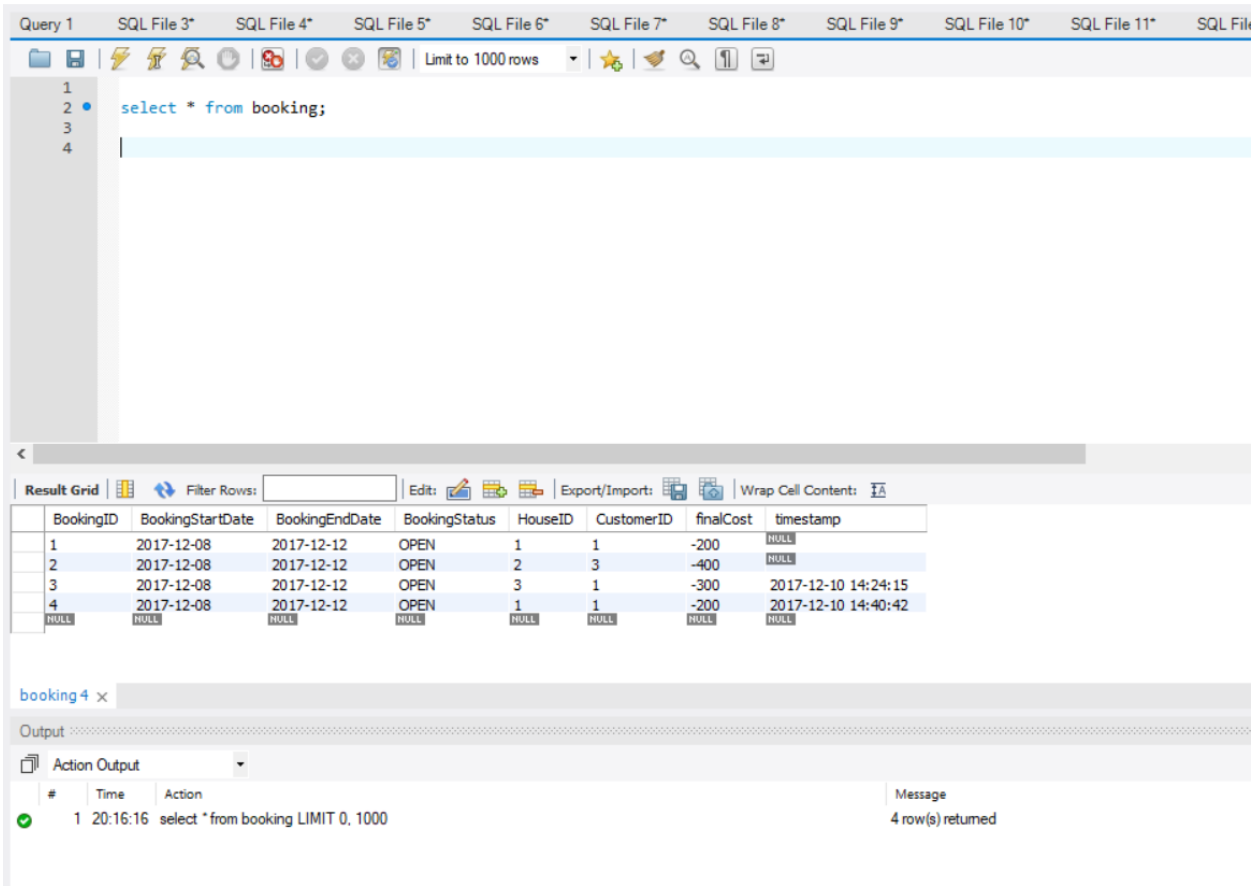
```
create table message
(
messagename varchar(200)
);
```

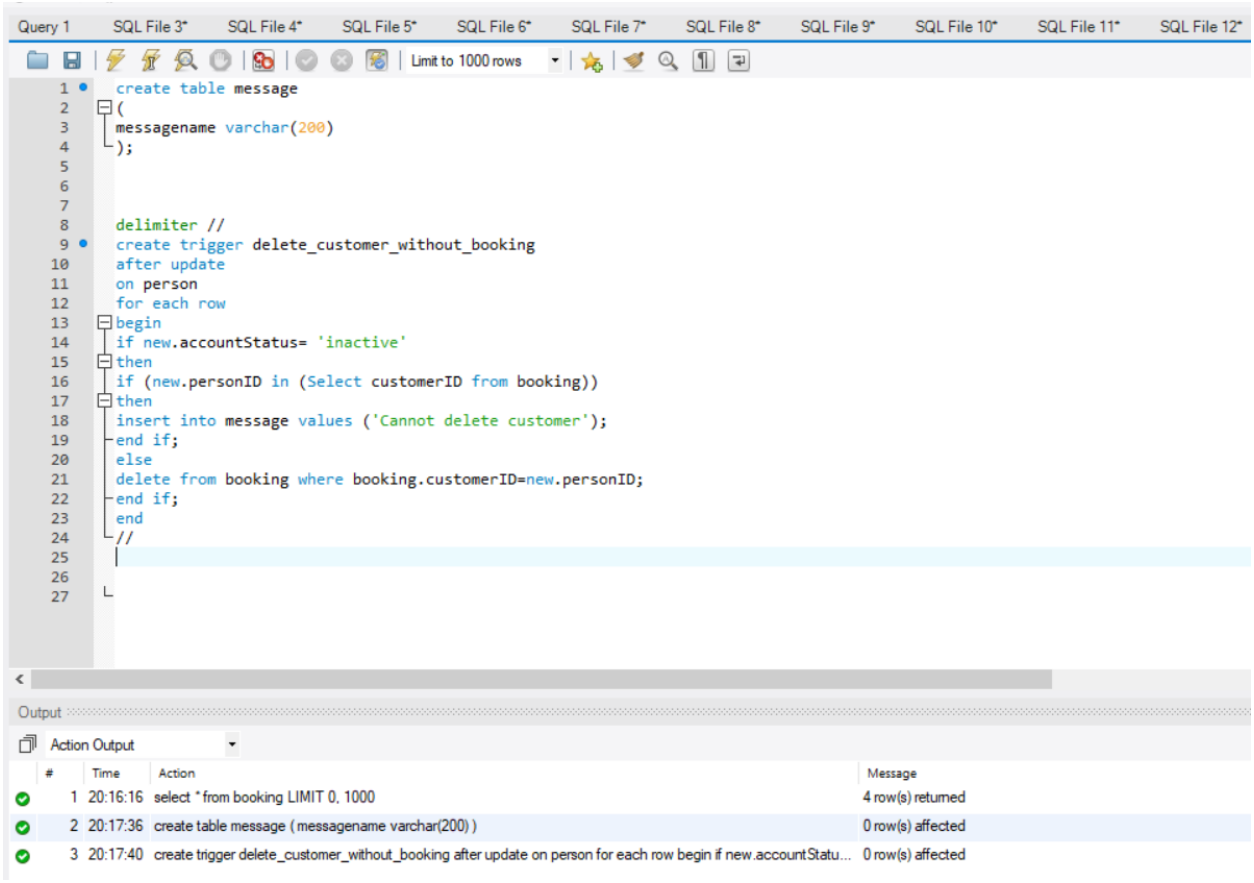
```
select * from booking
```

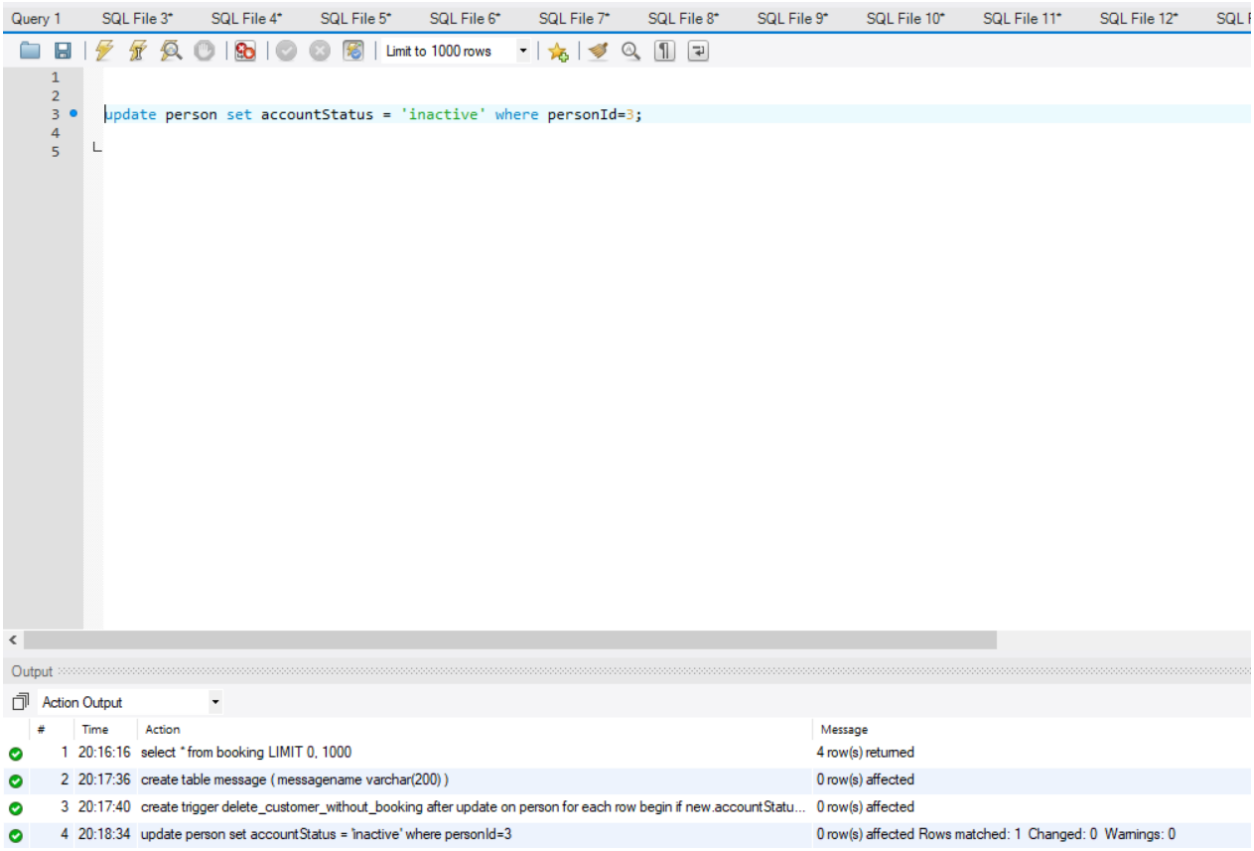
```
update person set accountStatus = 'inactive' where personId=3;
```

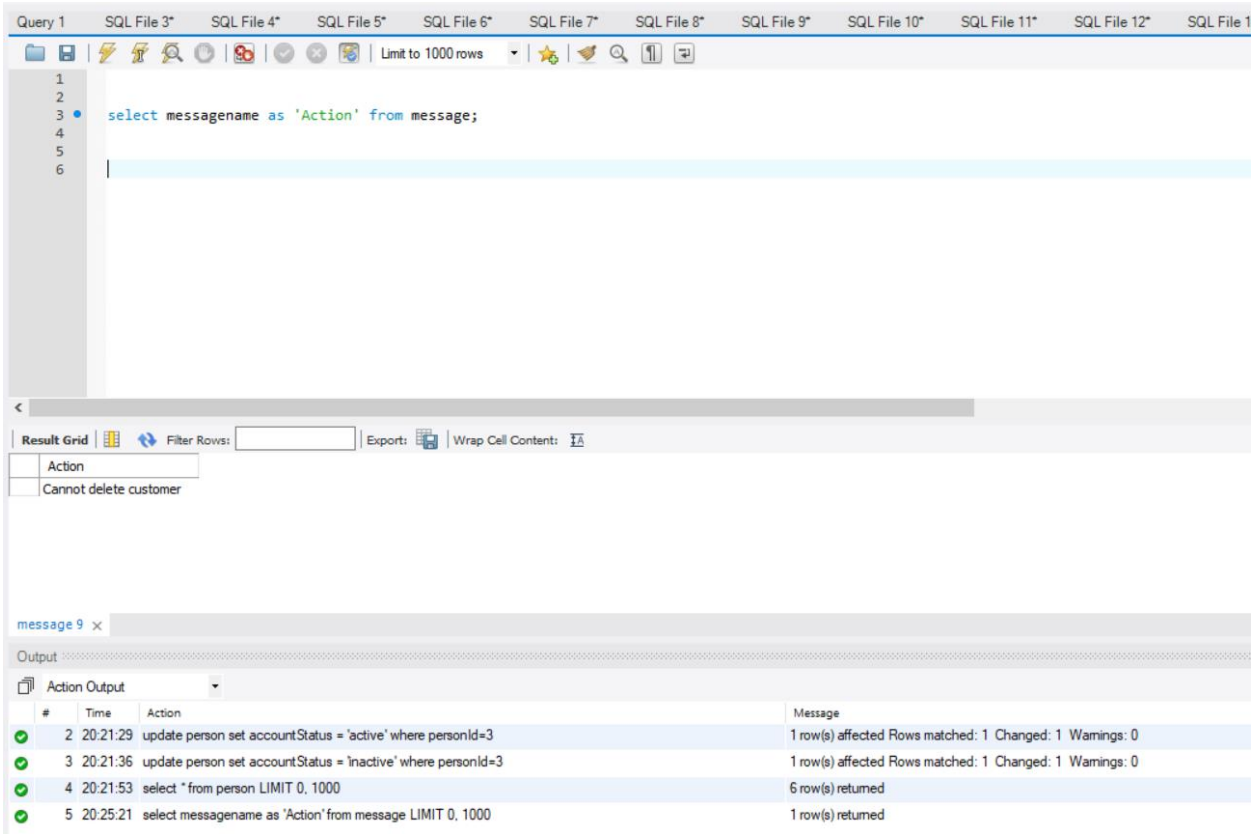
```
select messageName as 'Action' from message;
```

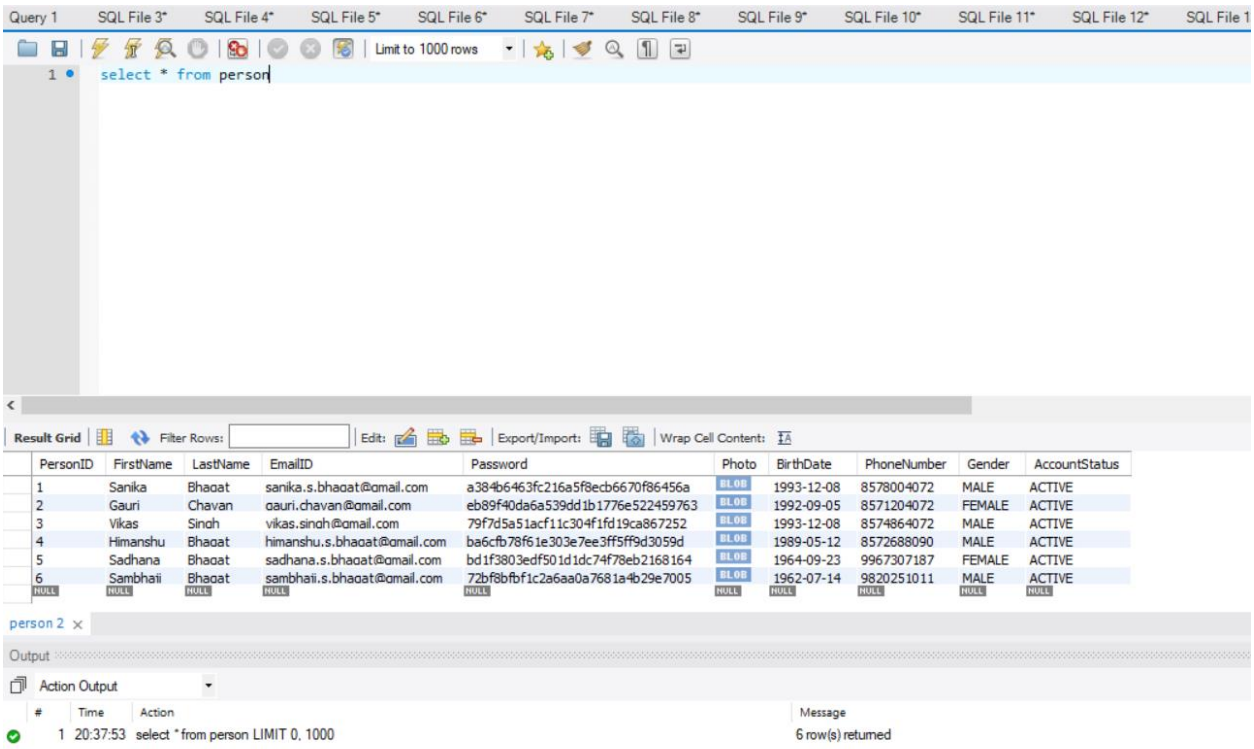
```
select * from person;
```


Query 1
SQL File 3*
SQL File 4*
SQL File 5*
SQL File 6*
SQL File 7*
SQL File 8*
SQL File 9*
SQL File 10*
SQL File 11*
Limit to 1000 rows
select * from booking;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
BookingID BookingStartDate BookingEndDate BookingStatus HouseID CustomerID finalCost timestamp
1 2017-12-08 2017-12-12 OPEN 1 1 -200 NULL
2 2017-12-08 2017-12-12 OPEN 2 3 -400 NULL
3 2017-12-08 2017-12-12 OPEN 3 1 -300 2017-12-10 14:24:15
4 2017-12-08 2017-12-12 OPEN 1 1 -200 2017-12-10 14:40:42
booking 4
Output
Action Output
Time Action Message
1 20:16:16 select * from booking LIMIT 0, 1000 4 row(s) returned


Query 1
SQL File 3*
SQL File 4*
SQL File 5*
SQL File 6*
SQL File 7*
SQL File 8*
SQL File 9*
SQL File 10*
SQL File 11*
SQL File 12*
Limit to 1000 rows
create table message
(
messagename varchar(200)
);
delimiter //
create trigger delete_customer_without_booking
after update
on person
for each row
begin
if new.accountStatus= 'inactive'
then
if (new.personID in (Select customerID from booking))
then
insert into message values ('Cannot delete customer');
end if;
else
delete from booking where booking.customerID=new.personID;
end if;
end
//
Output
Action Output
Time Action Message
1 20:16:16 select * from booking LIMIT 0, 1000 4 row(s) returned
2 20:17:36 create table message (messagename varchar(200)) 0 row(s) affected
3 20:17:40 create trigger delete_customer_without_booking after update on person for each row begin if new.accountStatu... 0 row(s) affected


Query 1
SQL File 3*
SQL File 4*
SQL File 5*
SQL File 6*
SQL File 7*
SQL File 8*
SQL File 9*
SQL File 10*
SQL File 11*
SQL File 12*
Limit to 1000 rows
update person set accountStatus = 'inactive' where personId=3;
Output
Action Output
Time Action Message
1 20:16:16 select * from booking LIMIT 0, 1000 4 row(s) returned
2 20:17:36 create table message (messagename varchar(200)) 0 row(s) affected
3 20:17:40 create trigger delete_customer_without_booking after update on person for each row begin if new.accountStatu... 0 row(s) affected
4 20:18:34 update person set accountStatus = 'inactive' where personId=3 0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0


Query 1
SQL File 3*
SQL File 4*
SQL File 5*
SQL File 6*
SQL File 7*
SQL File 8*
SQL File 9*
SQL File 10*
SQL File 11*
SQL File 12*
Limit to 1000 rows
select messagename as 'Action' from message;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Action
Cannot delete customer
message 9
Output
Action Output
Time Action Message
2 20:21:29 update person set accountStatus = 'active' where personId=3 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
3 20:21:36 update person set accountStatus = 'inactive' where personId=3 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
4 20:21:53 select * from person LIMIT 0, 1000 6 row(s) returned
5 20:25:21 select messagename as 'Action' from message LIMIT 0, 1000 1 row(s) returned



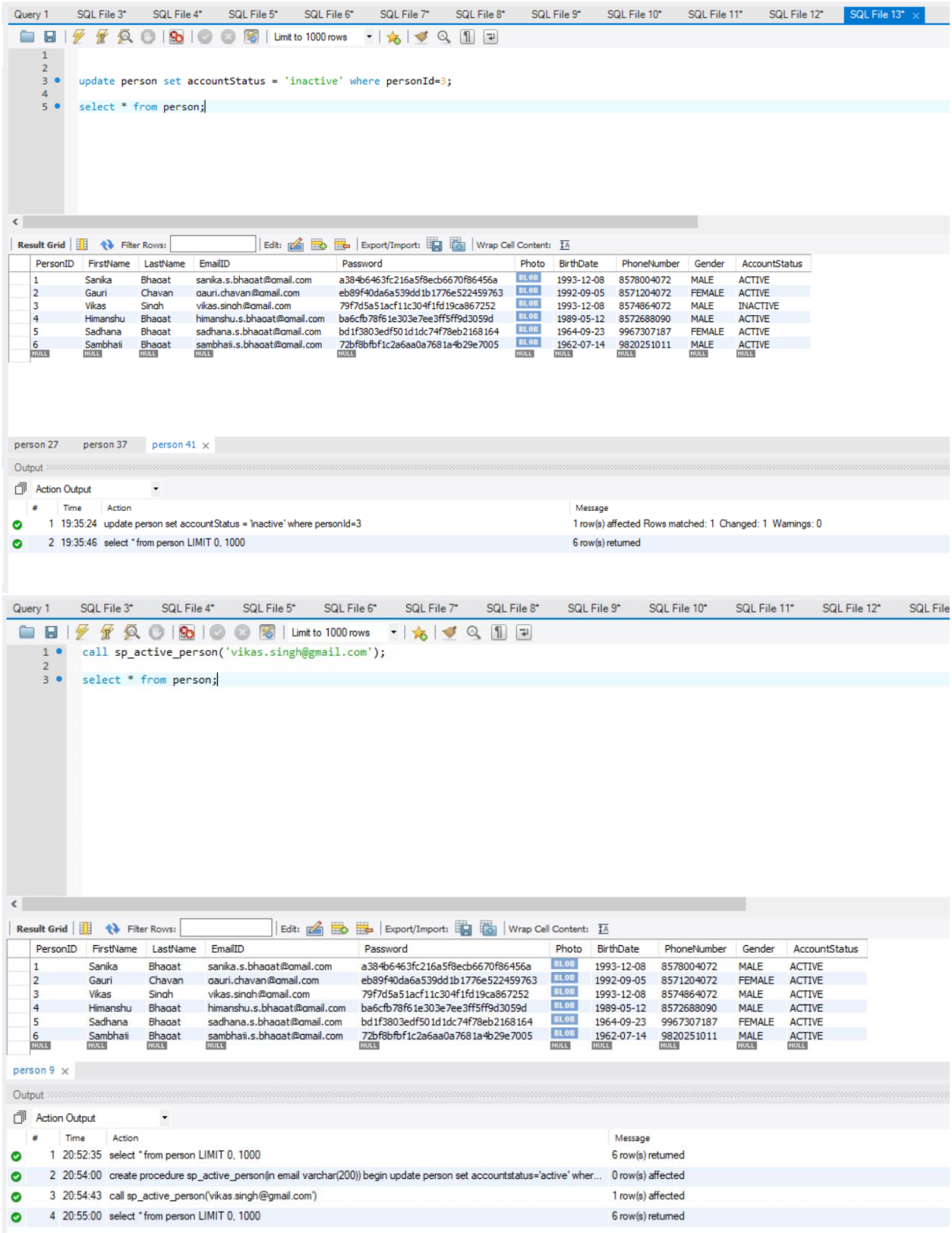
Scenario 11 :

Suppose a person deleted his account and want to activate it again. Then a new row should not be created but the status of the person in the person table should be made active.

Query :

```
delimiter //
create procedure sp_active_person(in email varchar(200))
begin
update person set accountstatus='active'
where EMailID=email;
end
//
```

```
Select * from person
call sp_active_person('vikas.singh@gmail.com');
```



```
update person set accountStatus = 'inactive' where personId=3;

select * from person;
```

PersonID	FirstName	LastName	EmailID	Password	Photo	BirthDate	PhoneNumber	Gender	AccountStatus
1	Sanika	Bhagat	sanika.s.bhagat@gmail.com	a384b6463fc216a5f8ecb6670f86456a	BLOB	1993-12-08	8578004072	MALE	ACTIVE
2	Gauri	Chavan	gauri.chavan@gmail.com	eb89f40da6a539dd1b1776e522459763	BLOB	1992-09-05	8571204072	FEMALE	ACTIVE
3	Vikas	Singh	vikas.singh@gmail.com	79f7d5a51acf11c304f1fd19ca867252	BLOB	1993-12-08	8574864072	MALE	INACTIVE
4	Himanshu	Bhagat	himanshu.s.bhagat@gmail.com	ba6cfb78f61e303e7ee3ff5ff9d3059d	BLOB	1989-05-12	8572688090	MALE	ACTIVE
5	Sadhana	Bhagat	sadhana.s.bhagat@gmail.com	bd1f3803edf501d1dc74f78eb2168164	BLOB	1964-09-23	9967307187	FEMALE	ACTIVE
6	Sambhaji	Bhagat	sambhaji.s.bhagat@gmail.com	72bf8bfbf1c2a6aa0a7681a4b29e7005	BLOB	1962-07-14	9820251011	MALE	ACTIVE
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

person 27 | person 37 | person 41

Output

Action Output

#	Time	Action	Message
1	19:35:24	update person set accountStatus = 'inactive' where personId=3	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
2	19:35:46	select * from person LIMIT 0, 1000	6 row(s) returned


```
call sp_active_person('vikas.singh@gmail.com');

select * from person;
```

PersonID	FirstName	LastName	EmailID	Password	Photo	BirthDate	PhoneNumber	Gender	AccountStatus
1	Sanika	Bhagat	sanika.s.bhagat@gmail.com	a384b6463fc216a5f8ecb6670f86456a	BLOB	1993-12-08	8578004072	MALE	ACTIVE
2	Gauri	Chavan	gauri.chavan@gmail.com	eb89f40da6a539dd1b1776e522459763	BLOB	1992-09-05	8571204072	FEMALE	ACTIVE
3	Vikas	Singh	vikas.singh@gmail.com	79f7d5a51acf11c304f1fd19ca867252	BLOB	1993-12-08	8574864072	MALE	ACTIVE
4	Himanshu	Bhagat	himanshu.s.bhagat@gmail.com	ba6cfb78f61e303e7ee3ff5ff9d3059d	BLOB	1989-05-12	8572688090	MALE	ACTIVE
5	Sadhana	Bhagat	sadhana.s.bhagat@gmail.com	bd1f3803edf501d1dc74f78eb2168164	BLOB	1964-09-23	9967307187	FEMALE	ACTIVE
6	Sambhaji	Bhagat	sambhaji.s.bhagat@gmail.com	72bf8bfbf1c2a6aa0a7681a4b29e7005	BLOB	1962-07-14	9820251011	MALE	ACTIVE
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

person 9

Output

Action Output

#	Time	Action	Message
1	20:52:35	select * from person LIMIT 0, 1000	6 row(s) returned
2	20:54:00	create procedure sp_active_person(in email varchar(200)) begin update person set accountstatus='active' wher...	0 row(s) affected
3	20:54:43	call sp_active_person('vikas.singh@gmail.com')	1 row(s) affected
4	20:55:00	select * from person LIMIT 0, 1000	6 row(s) returned

Scenario 12 :

If a booking is done, the payment table should get populated and the after the payment table, the airbnb_earnings and host_earnings should be populated.

Query :

To get the booking cost in the booking table

```
delimiter //
create procedure sp_bookingcost_in_booking ()
begin
/*create view view_booking*/
update booking join
(select booking.houseid,(booking.BookingStartDate-booking.BookingEndDate)*house.CostPerNight as bookingAmount from
booking inner join house on house.houseid=booking.houseid) temp
on booking.houseid=temp.houseid
set finalCost=temp.bookingAmount;
select bookingId,houseID,bookingStartDate,bookingEndDate,finalCost from Booking;
end
//
```

Creating a dummy table booking_dummy

```
create table booking_dummy
        (
        BookingID int,
        BookingStartDate date,
        BookingEndDate date,
        BookingStatus varchar(20),
        HouseID int,
        CustomerID int,
        FinalCost double,
        timestamp datetime
        );
```

Populating the dummy table

```
delimiter //
create procedure insert_into_dummy()
begin
insert into booking_dummy
select * from booking order by timestamp desc limit 1;
end;
//
```

```
delimiter //
create trigger insert_into_paymenttable
after insert
on booking_dummy
for each row
begin
```

```
insert into payment(paymentAmount, timeStamp,BookingID) values (new.finalCost,now(),new.BookingID);
end
//
```

```
delimiter //
create trigger airbnb_trigger
after insert
on payment
for each row
begin
insert into `airbnb earnings` values (new.paymentID,new.paymentAmount*0.7);
end
//
```

```
delimiter //
create trigger host_earnings_trigger
after insert
on payment
for each row
begin
insert into `host earnings` values (new.paymentID,new.paymentAmount*0.3);
end
//
```

```
insert into booking(bookingID,BookingStartDAte,BookingEndDate,BookingStatus,HouseID,CustomerID,timestamp) values
(4,'2017-12-08','2017-12-12','OPEN','1','1',now());

call sp_bookingcost_in_booking();

call insert_into_dummy();
```

Query 1 | SQL File 3* | SQL File 4* | SQL File 5* | SQL File 6* | SQL File 7* | SQL File 8* | SQL File 9* | SQL File 10* | SQL File 11* | SQL File 12* | SQL

Limit to 1000 rows

```
1  delimiter //
2  create procedure sp_bookingcost_in_booking()
3  begin
4  /*create view view_booking*/
5  update booking join
6  (select booking.houseid,(booking.BookingEndDate-booking.BookingStartDate)*house.CostPerNight
7  as bookingAmount from booking inner join house on house.houseid=booking.houseid) temp
8  on booking.houseid=temp.houseid
9  set finalCost=temp.bookingAmount;
10 select bookingId,houseID,bookingStartDate,bookingEndDate,finalCost from Booking;
11 end
12 //
13
14
15 delimiter //
16 create procedure insert_into_dummy()
17 begin
18 insert into booking_dummy
19 select * from booking order by timestamp desc limit 1;
20 end;
21 //
22
23
24
```

Output

Action Output

#	Time	Action	Message
1	23:22:13	create procedure sp_bookingcost_in_booking() begin /*create view view_booking*/ update booking join (select ...	0 row(s) affected
2	23:22:17	create procedure insert_into_dummy() begin insert into booking_dummy select * from booking order by timestamp ...	0 row(s) affected

Query 1 | create table* | payment | person_table_update_logs | refund | rating - Table | rating | SQL File 4* | SQL File 5* | SQL File 6*

Limit to 1000 rows

```
1  call sp_bookingcost_in_booking();
2
3  call insert_into_dummy();
4
5          *       booking_dummy
6
7  select * from payment
8
9  select * from `airbnb earnings`
10
11
12
```

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

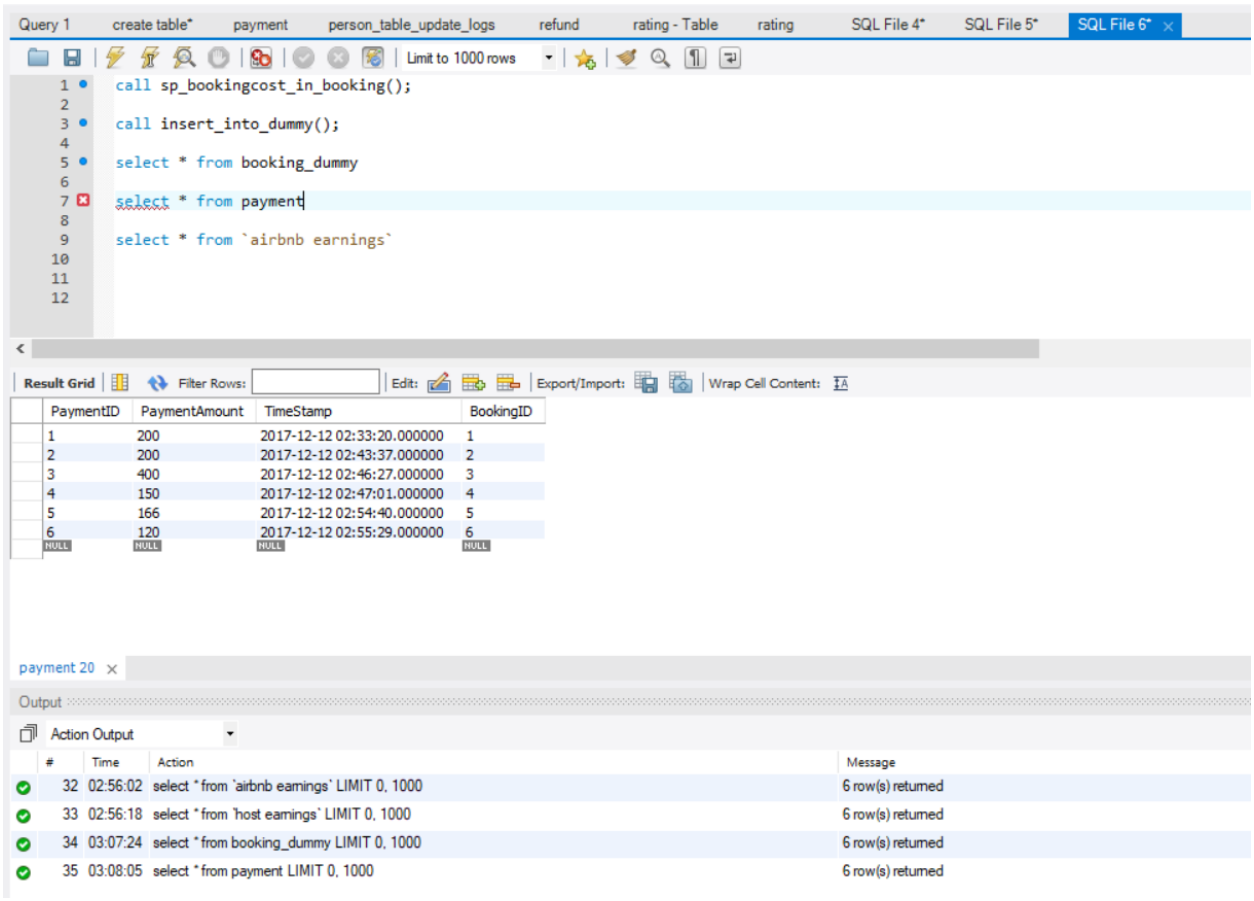
BookingID	BookingStartDate	BookingEndDate	BookingStatus	HouseID	CustomerID	FinalCost	timestamp
1	2017-12-08	2017-12-12	OPEN	1	1	200	2017-12-12 02:31:02
2	2018-01-01	2018-01-03	OPEN	1	1	200	2017-12-12 02:43:19
3	2018-03-08	2018-03-12	OPEN	2	2	400	2017-12-12 02:45:41
4	2017-12-23	2017-12-25	OPEN	3	3	150	2017-12-12 02:46:39
5	2017-12-29	2017-12-31	OPEN	6	4	166	2017-12-12 02:54:13
6	2018-02-08	2018-02-14	OPEN	5	4	120	2017-12-12 02:55:11

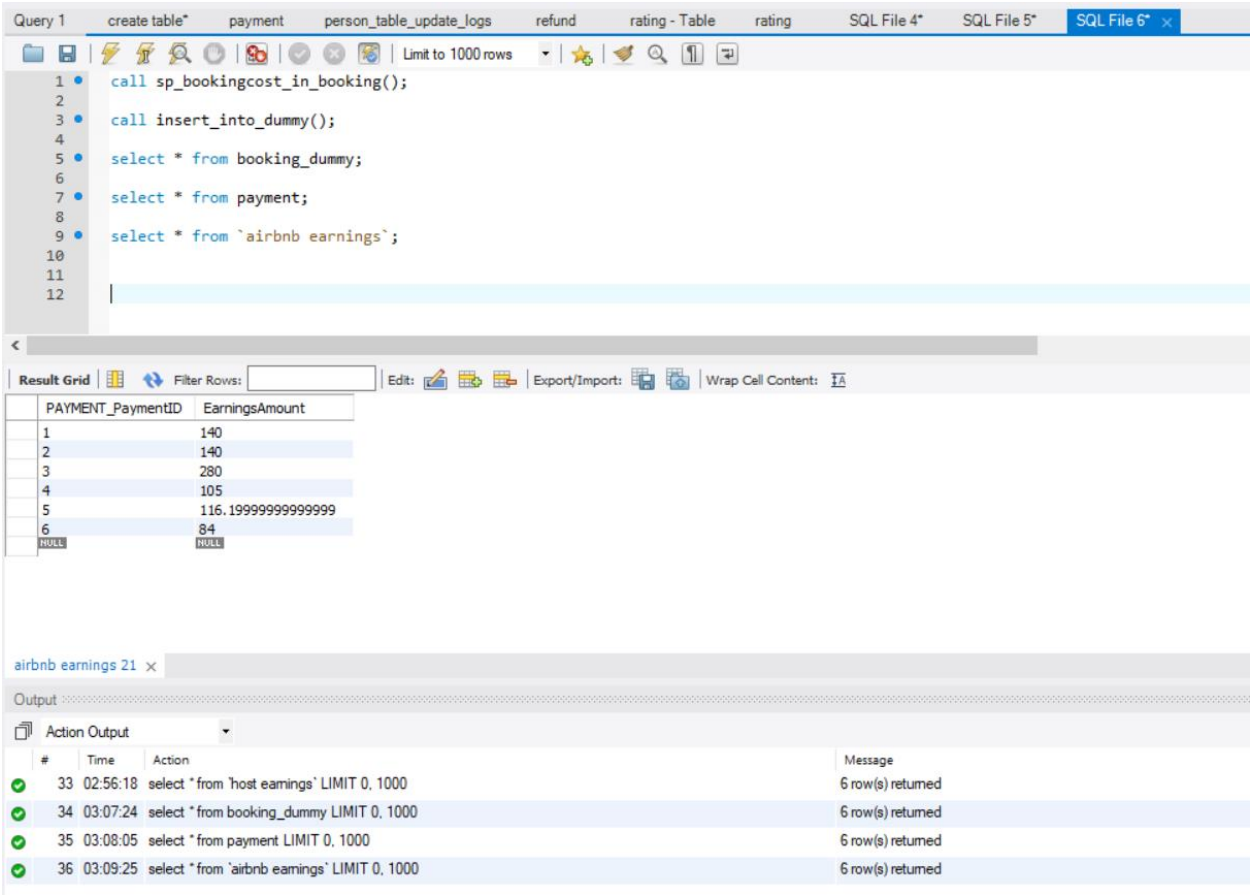
booking_dummy 19

Output

Action Output

#	Time	Action	Message
31	02:55:39	select * from payment LIMIT 0, 1000	6 row(s) returned
32	02:56:02	select * from `airbnb earnings` LIMIT 0, 1000	6 row(s) returned
33	02:56:18	select * from `host earnings` LIMIT 0, 1000	6 row(s) returned
34	03:07:24	select * from booking_dummy LIMIT 0, 1000	6 row(s) returned


Query 1
create table*
payment
person_table_update_logs
refund
rating - Table
rating
SQL File 4*
SQL File 5*
SQL File 6*
Limit to 1000 rows
call sp_bookingcost_in_booking();
call insert_into_dummy();
select * from booking_dummy
select * from payment
select * from `airbnb earnings`
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
PaymentID PaymentAmount TimeStamp BookingID
1 200 2017-12-12 02:33:20.000000 1
2 200 2017-12-12 02:43:37.000000 2
3 400 2017-12-12 02:46:27.000000 3
4 150 2017-12-12 02:47:01.000000 4
5 166 2017-12-12 02:54:40.000000 5
6 120 2017-12-12 02:55:29.000000 6
payment 20
Output
Action Output
Time Action Message
32 02:56:02 select * from `airbnb earnings` LIMIT 0, 1000 6 row(s) returned
33 02:56:18 select * from `host earnings` LIMIT 0, 1000 6 row(s) returned
34 03:07:24 select * from booking_dummy LIMIT 0, 1000 6 row(s) returned
35 03:08:05 select * from payment LIMIT 0, 1000 6 row(s) returned



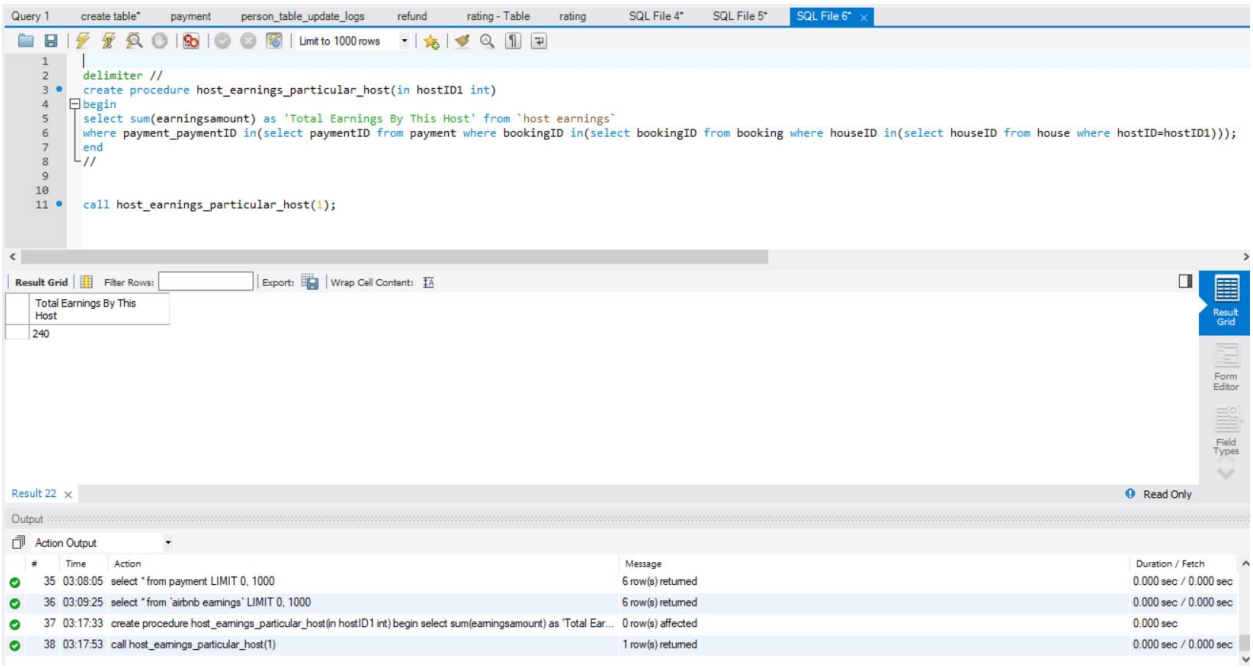
Scenario 13:

Find out the total earnings of the host till date

Query :

```
delimiter //
create procedure host_earnings_particular_host(in hostID1 int)
begin
select sum(earningsamount) as 'Total Earnings By This Host' from `host earnings`
where payment_paymentID in(select paymentID from payment where bookingID in(select bookingID from booking where
houseID in(select houseID from house where hostID=hostID1)));
end
//
```

```
call host_earnings_particular_host(1);
```



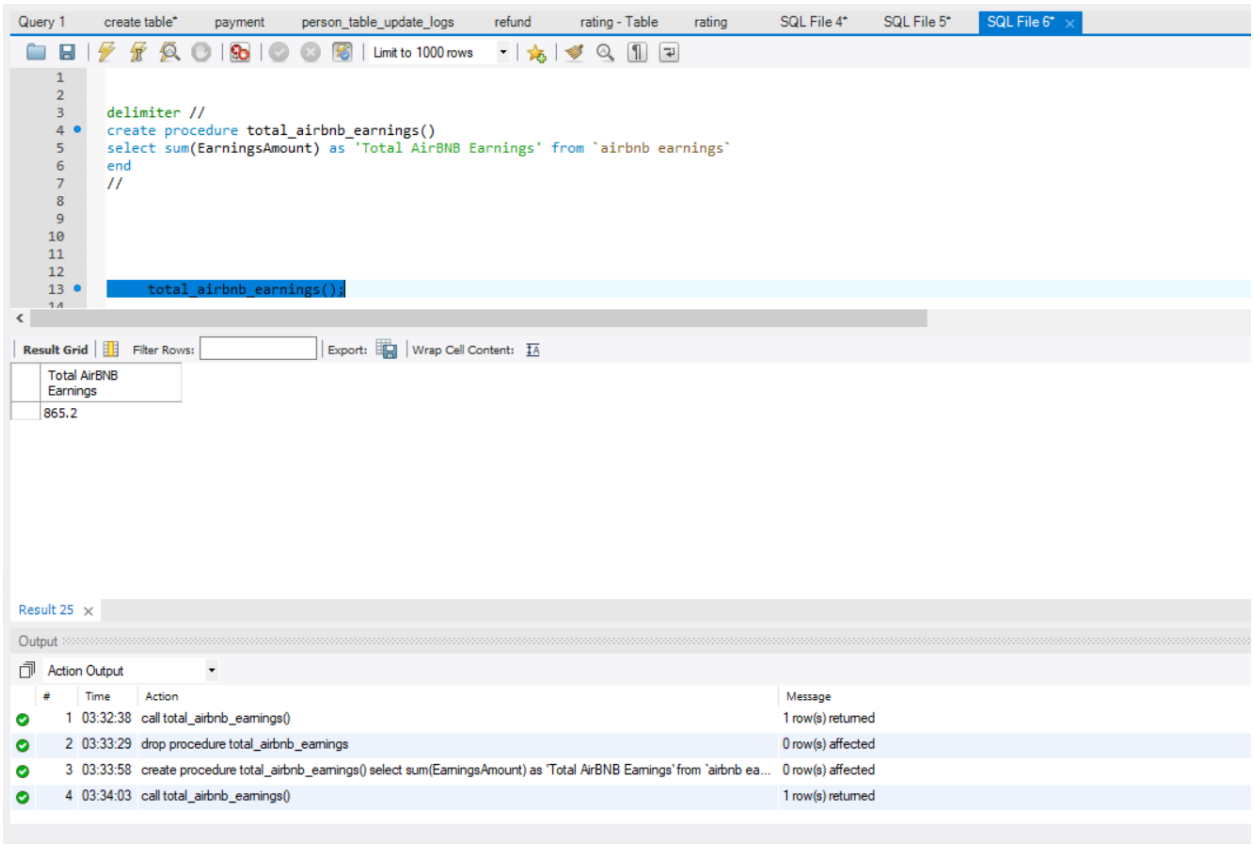
Scenario 14 :

Find the total earnings of AirBNB

Query :

```
delimiter //
create procedure total_airbnb_earnings()
select sum(EarningsAmount) as 'Total AirBNB Earnings' from `airbnb earnings`
end
//
```

```
call total_airbnb_earnings();
```



Scenario 15 :

If a booking is cancelled, then the refund table should be populated and the records from airbnb and host tables should be deleted as well

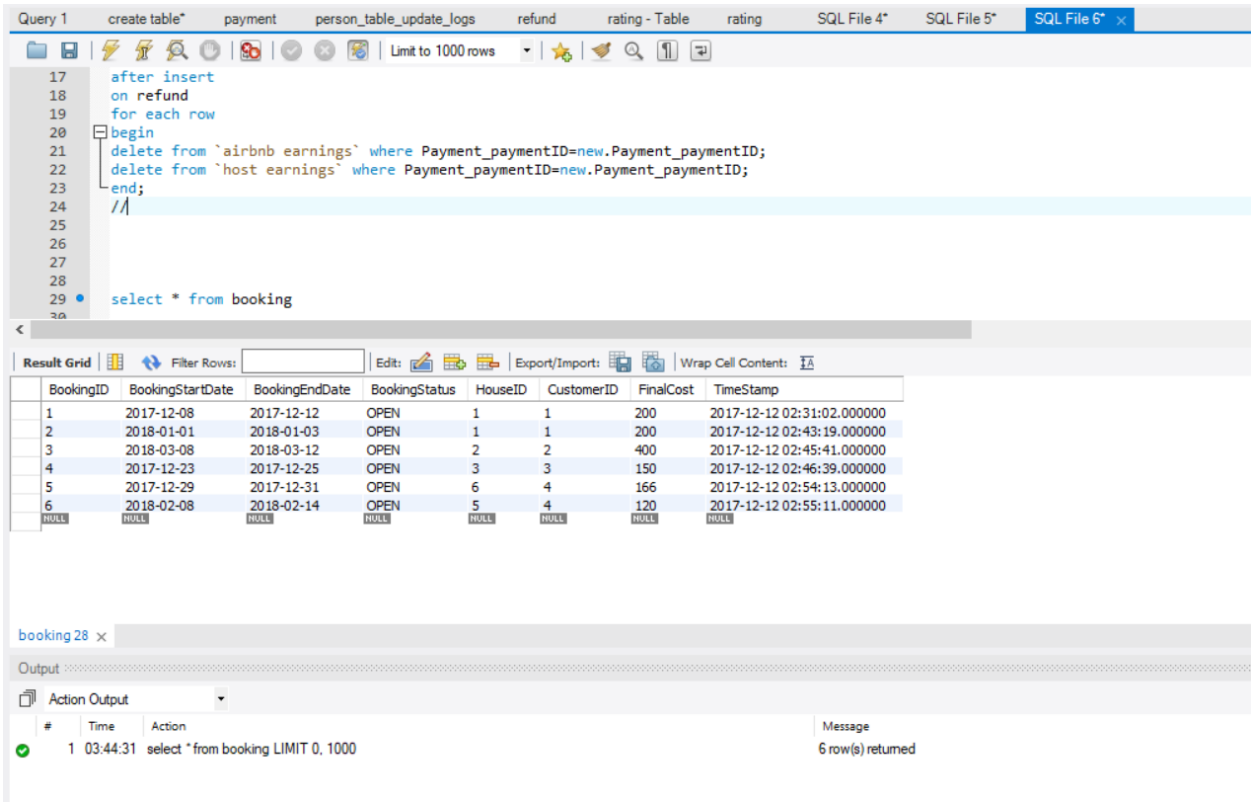
Query :

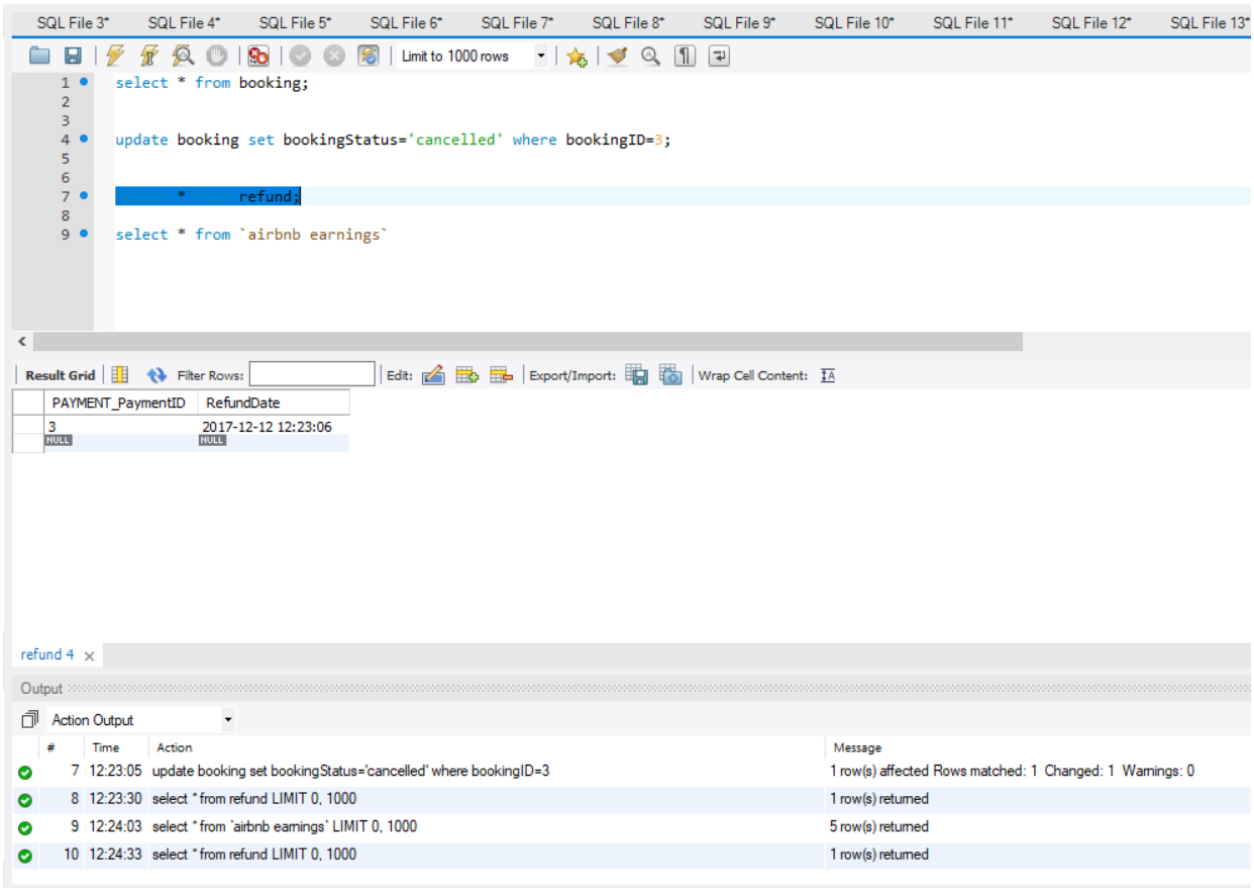
```
delimiter //
create trigger refund_policy
after update
on booking
for each row
begin
if new.bookingStatus= 'cancelled'
then
insert into refund (payment_paymentID) select (paymentID)from payment where payment.bookingID=new.bookingID ;
end if;
end
//
```

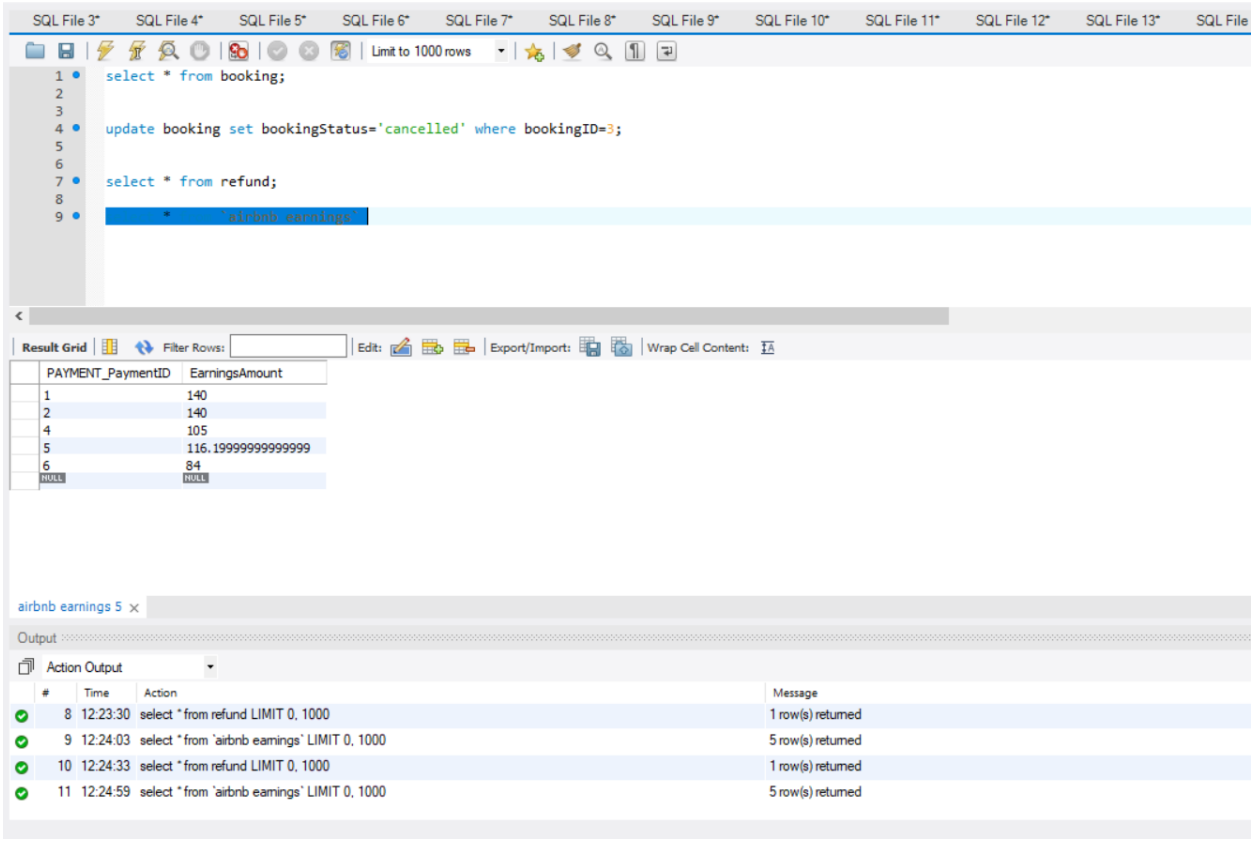
```
delimiter //
create trigger delete_airbnb_host_earnings
after insert
on refund
for each row
begin
```

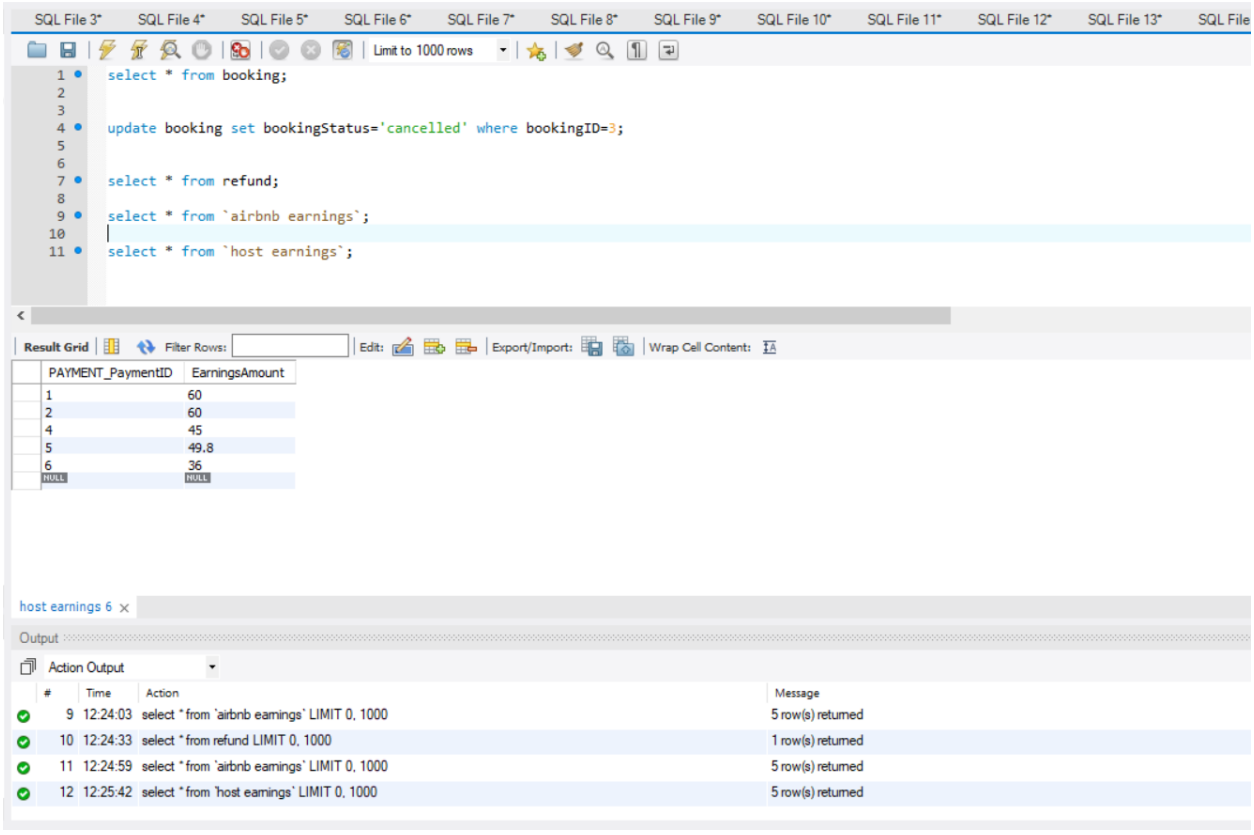
```
delete from `airbnb earnings` where Payment_paymentID=new.Payment_paymentID;
delete from `host earnings` where Payment_paymentID=new.Payment_paymentID;
end;
//
```

```
update booking set bookingStatus='cancelled' where bookingID=3;
```




SQL File 3*
SQL File 4*
SQL File 5*
SQL File 6*
SQL File 7*
SQL File 8*
SQL File 9*
SQL File 10*
SQL File 11*
SQL File 12*
SQL File 13*
Limit to 1000 rows
select * from booking;
update booking set bookingStatus='cancelled' where bookingID=3;
* refund;
select * from `airbnb earnings`
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
PAYMENT_PaymentID
RefundDate
3
2017-12-12 12:23:06
NULL
NULL
refund 4
Output
Action Output
Time Action Message
7 12:23:05 update booking set bookingStatus='cancelled' where bookingID=3 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
8 12:23:30 select * from refund LIMIT 0, 1000 1 row(s) returned
9 12:24:03 select * from `airbnb earnings` LIMIT 0, 1000 5 row(s) returned
10 12:24:33 select * from refund LIMIT 0, 1000 1 row(s) returned


SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13* SQL File
Limit to 1000 rows
1 select * from booking;
4 update booking set bookingStatus='cancelled' where bookingID=3;
7 select * from refund;
9 * `airbnb earnings`
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
PAYMENT_PaymentID EarningsAmount
1 140
2 140
4 105
5 116.19999999999999
6 84
NULL NULL
airbnb earnings 5
Output
Action Output
Time Action Message
8 12:23:30 select * from refund LIMIT 0, 1000 1 row(s) returned
9 12:24:03 select * from `airbnb earnings` LIMIT 0, 1000 5 row(s) returned
10 12:24:33 select * from refund LIMIT 0, 1000 1 row(s) returned
11 12:24:59 select * from `airbnb earnings` LIMIT 0, 1000 5 row(s) returned

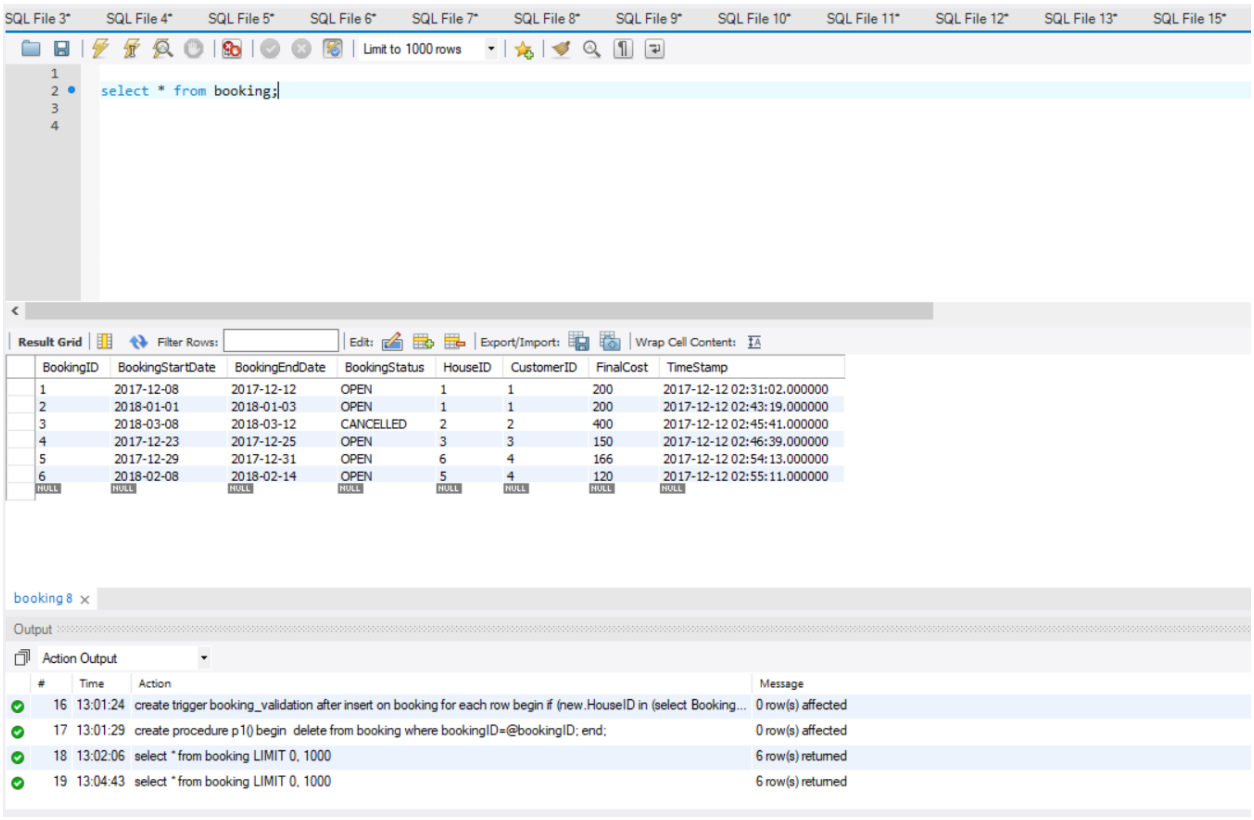

SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13* SQL File
Limit to 1000 rows
1 select * from booking;
4 update booking set bookingStatus='cancelled' where bookingID=3;
7 select * from refund;
9 select * from `airbnb earnings`;
11 select * from `host earnings`;
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
PAYMENT_PaymentID EarningsAmount
1 60
2 60
4 45
5 49.8
6 36
NULL NULL
host earnings 6
Output
Action Output
Time Action Message
9 12:24:03 select * from `airbnb earnings` LIMIT 0, 1000 5 row(s) returned
10 12:24:33 select * from refund LIMIT 0, 1000 1 row(s) returned
11 12:24:59 select * from `airbnb earnings` LIMIT 0, 1000 5 row(s) returned
12 12:25:42 select * from `host earnings` LIMIT 0, 1000 5 row(s) returned

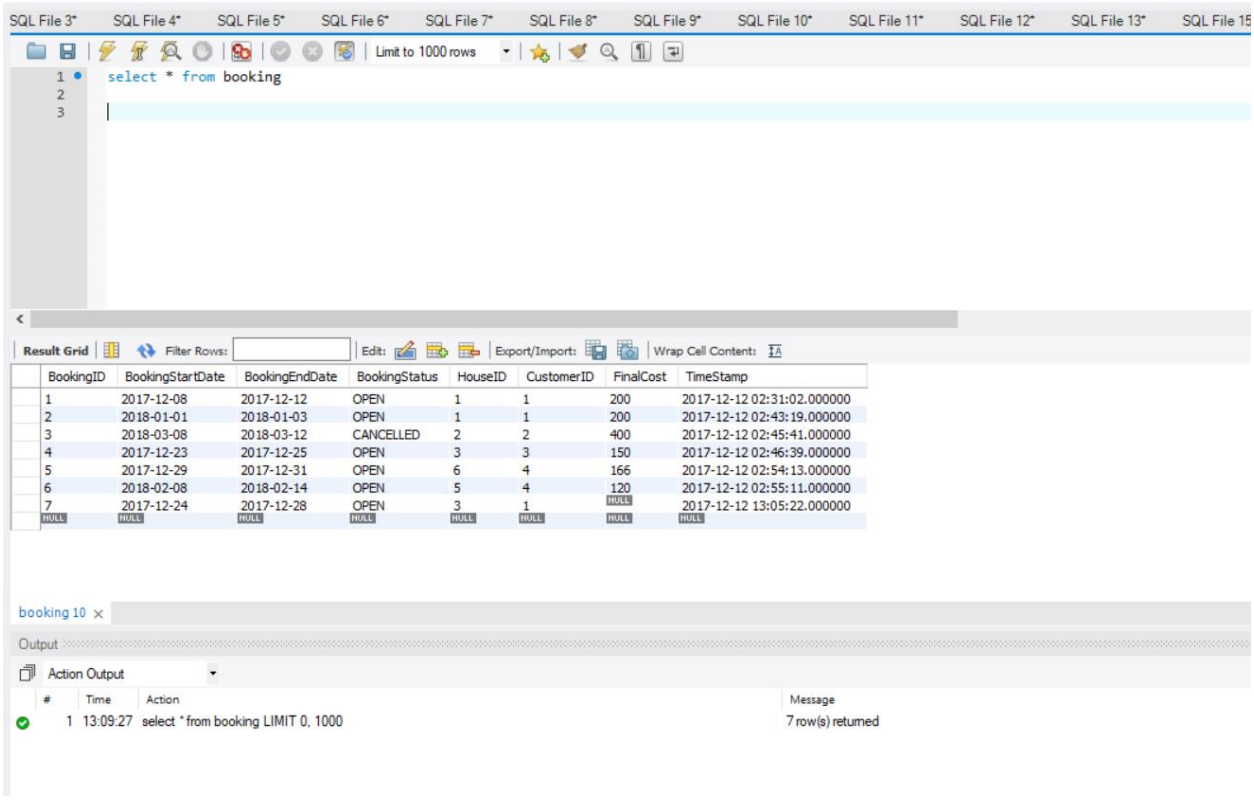
Scenario 16 :

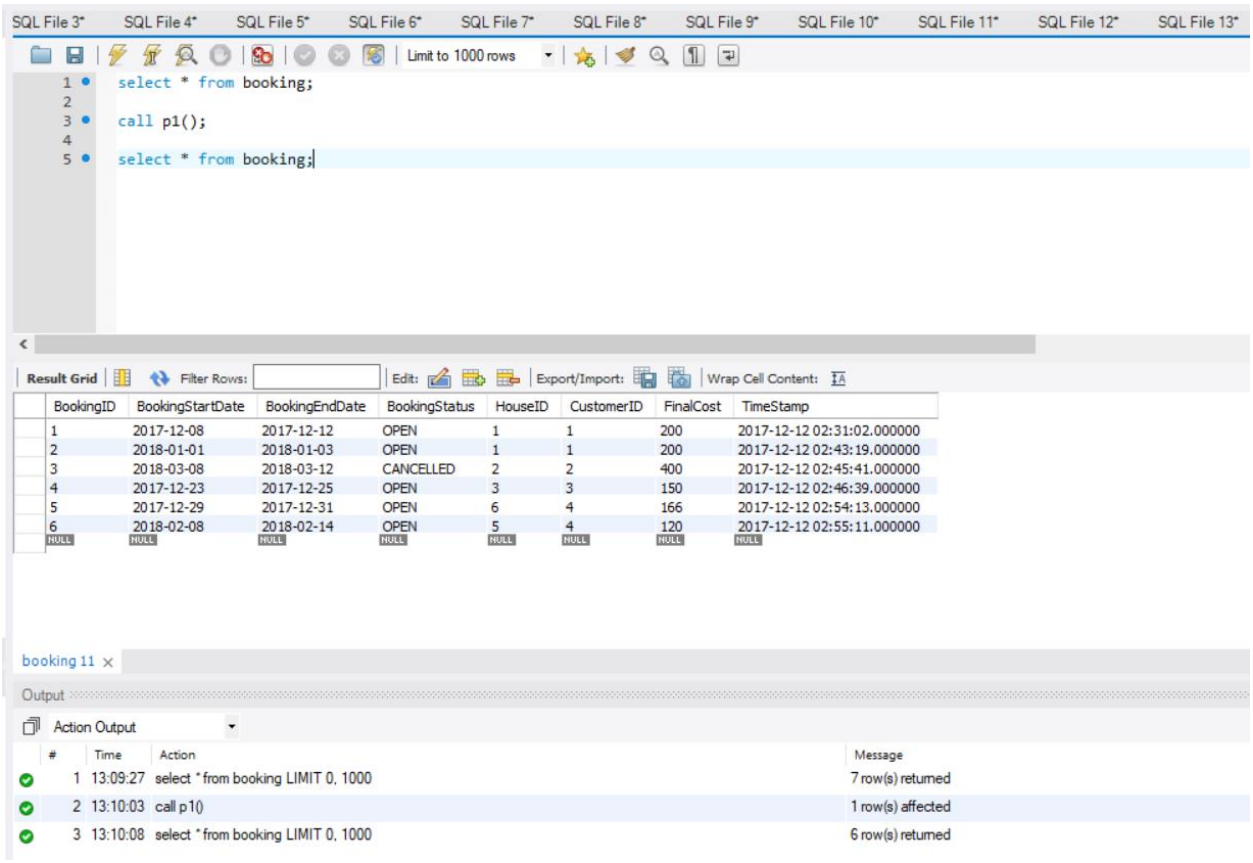
If a house has been booked for a number of days by a customer, the same house cannot be booked for the same number of days by any other customers.

Query :

```
delimiter //
create trigger booking_validation
after insert
on booking
for each row
begin
if (new.HouseID in (select BookingID from booking))
then
if ((new.BookingstartDate < any (select BookingEndDate from Booking)) and (new.BookingStartDate > any(select
BookingStartDAte from booking)))
then
set @bookingID=new.BookingID;
end if;
end if;
end
//
-----------------------------------------------------------------------------------------------------------------------------------------------
delimiter //
create procedure p1()
begin
delete from booking where bookingID=@bookingID;
end;
//
-----------------------------------------------------------------------------------------------------------------------------------------------
select * from booking;

insert into booking values()

insert into booking(BookingID,bookingStartDate,BookingEndDate,bookingStatus,houseID,CustomerID,timestamp) values
(7,'2017/12/24','2017/12/26','open',4,4,now());

call p1();
```


SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13* SQL File 15*
Limit to 1000 rows
select * from booking;
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
BookingID BookingStartDate BookingEndDate BookingStatus HouseID CustomerID FinalCost TimeStamp
1 2017-12-08 2017-12-12 OPEN 1 1 200 2017-12-12 02:31:02.000000
2 2018-01-01 2018-01-03 OPEN 1 1 200 2017-12-12 02:43:19.000000
3 2018-03-08 2018-03-12 CANCELLED 2 2 400 2017-12-12 02:45:41.000000
4 2017-12-23 2017-12-25 OPEN 3 3 150 2017-12-12 02:46:39.000000
5 2017-12-29 2017-12-31 OPEN 6 4 166 2017-12-12 02:54:13.000000
6 2018-02-08 2018-02-14 OPEN 5 4 120 2017-12-12 02:55:11.000000
booking 8
Output
Action Output
Time Action Message
16 13:01:24 create trigger booking_validation after insert on booking for each row begin if (new.HouseID in (select Booking... 0 row(s) affected
17 13:01:29 create procedure p1() begin delete from booking where bookingID=@bookingID; end; 0 row(s) affected
18 13:02:06 select * from booking LIMIT 0, 1000 6 row(s) returned
19 13:04:43 select * from booking LIMIT 0, 1000 6 row(s) returned


SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* SQL File 10* SQL File 11* SQL File 12* SQL File 13* SQL File 15
Limit to 1000 rows
select * from booking
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
BookingID BookingStartDate BookingEndDate BookingStatus HouseID CustomerID FinalCost TimeStamp
1 2017-12-08 2017-12-12 OPEN 1 1 200 2017-12-12 02:31:02.000000
2 2018-01-01 2018-01-03 OPEN 1 1 200 2017-12-12 02:43:19.000000
3 2018-03-08 2018-03-12 CANCELLED 2 2 400 2017-12-12 02:45:41.000000
4 2017-12-23 2017-12-25 OPEN 3 3 150 2017-12-12 02:46:39.000000
5 2017-12-29 2017-12-31 OPEN 6 4 166 2017-12-12 02:54:13.000000
6 2018-02-08 2018-02-14 OPEN 5 4 120 2017-12-12 02:55:11.000000
7 2017-12-24 2017-12-28 OPEN 3 1 2017-12-12 13:05:22.000000
booking 10
Output
Action Output
Time Action Message
1 13:09:27 select * from booking LIMIT 0, 1000 7 row(s) returned

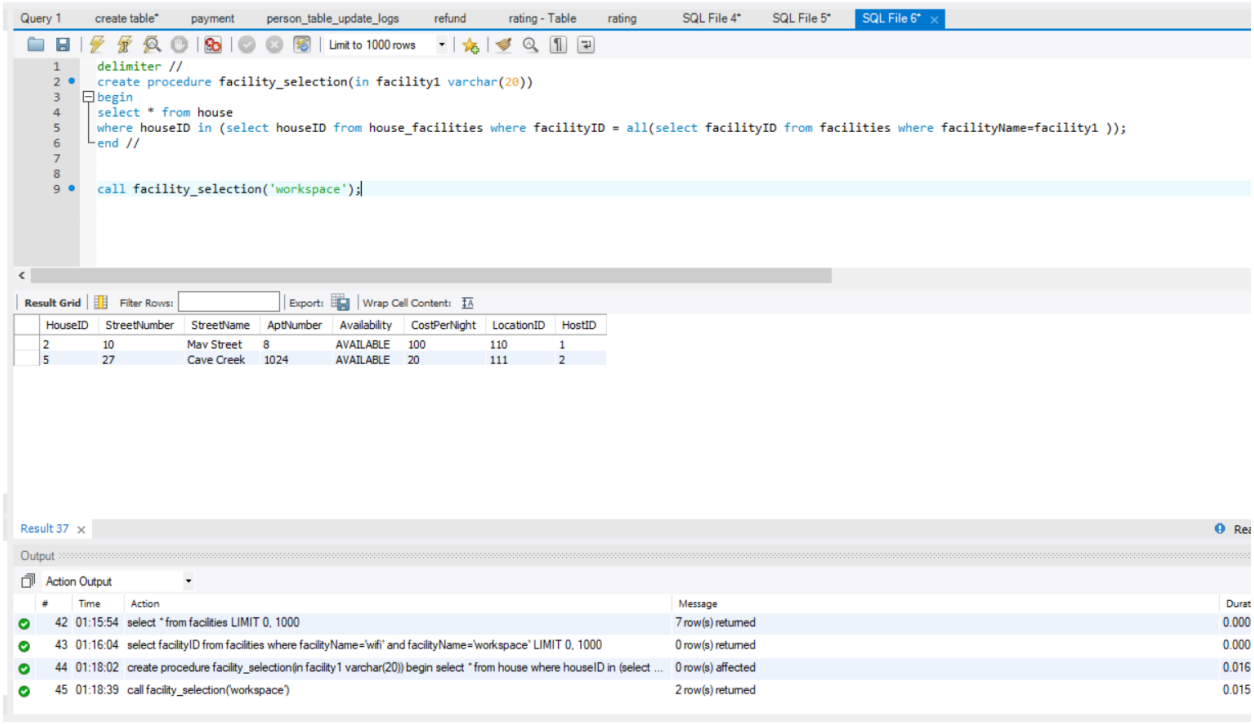


Scenario 17 :

A person can choose whatever facilities he is looking for in a house and then the houses with only those facilities will be displayed to the user.

Query :

```
delimiter //
create procedure facility_selection(in facility1 varchar(20))
begin
select * from house
where houseID in (select houseID from house_facilities where facilityID = all(select facilityID from facilities where
facilityName=facility1 ));
end //
-------------------------------------------------------------------------------------------------------------------------------
call facility_selection('workspace');
```

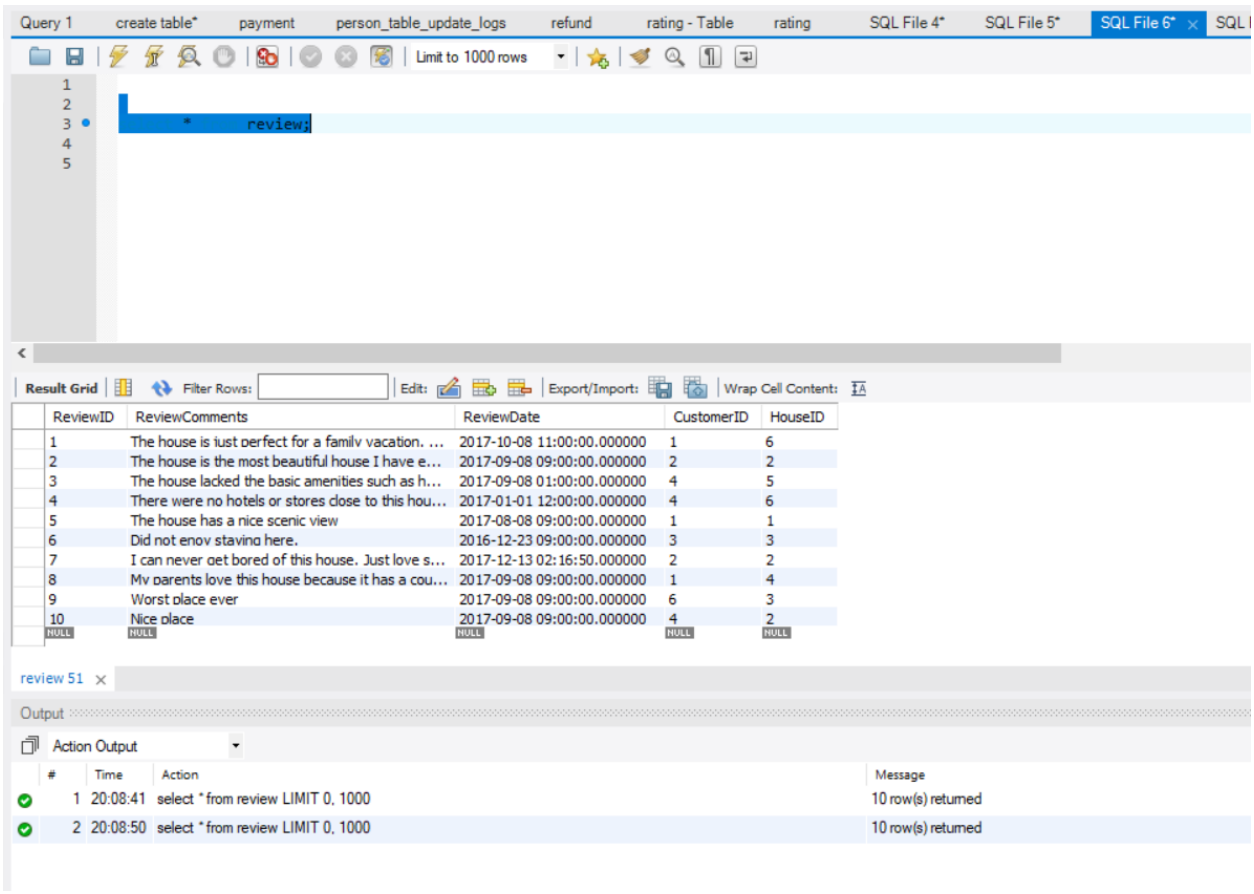


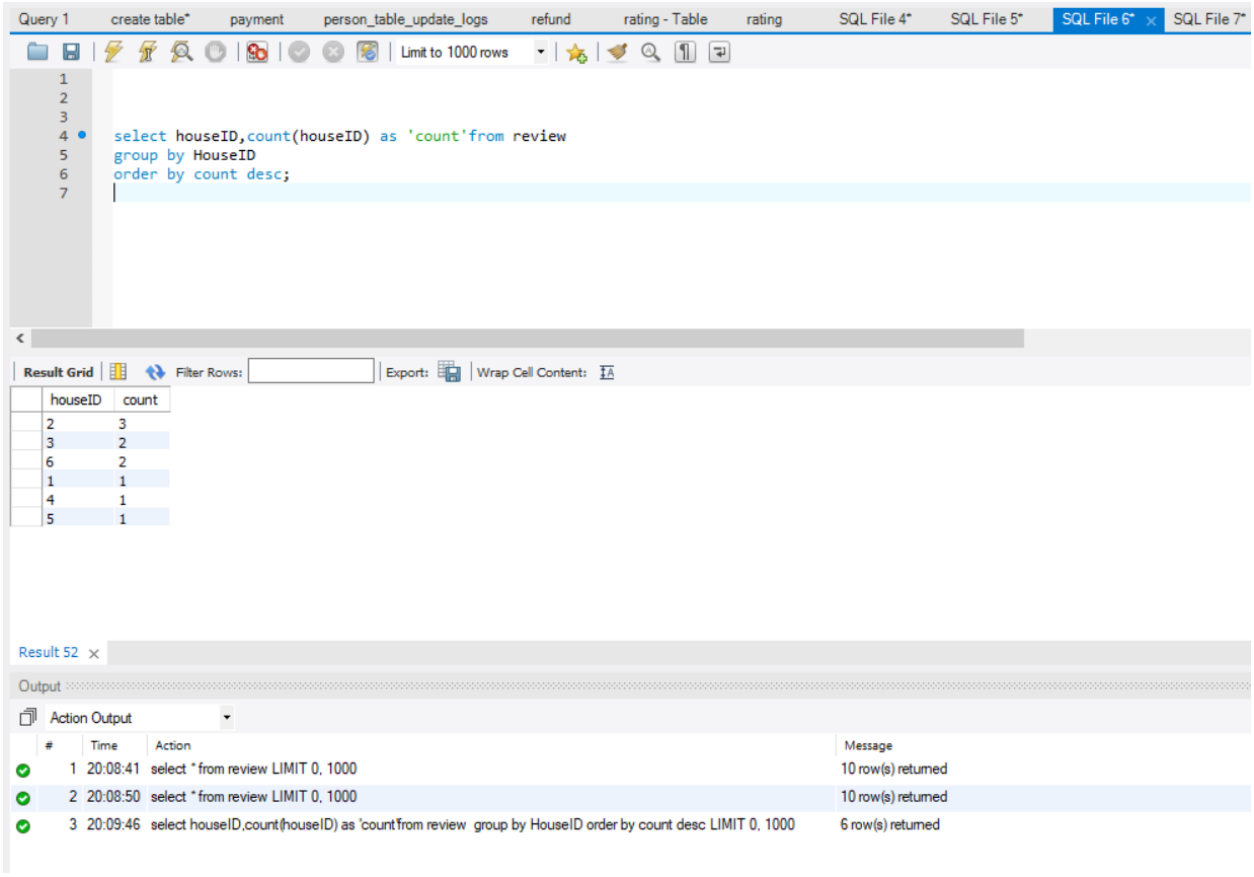
Scenario 18 :

Find the house which has got the maximum reviews :

Query:

```
select * from review;

select houseID,count(*) as 'count'from review
group by HouseID
order by count desc;
```


Query 1
create table*
payment
person_table_update_logs
refund
rating - Table
rating
SQL File 4*
SQL File 5*
SQL File 6*
Limit to 1000 rows
1
2
3
4
5
* review;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
ReviewID
ReviewComments
ReviewDate
CustomerID
HouseID
1 The house is just perfect for a family vacation. ... 2017-10-08 11:00:00.000000 1 6
2 The house is the most beautiful house I have e... 2017-09-08 09:00:00.000000 2 2
3 The house lacked the basic amenities such as h... 2017-09-08 01:00:00.000000 4 5
4 There were no hotels or stores close to this hou... 2017-01-01 12:00:00.000000 4 6
5 The house has a nice scenic view 2017-08-08 09:00:00.000000 1 1
6 Did not enoy staying here. 2016-12-23 09:00:00.000000 3 3
7 I can never get bored of this house. Just love s... 2017-12-13 02:16:50.000000 2 2
8 My parents love this house because it has a cou... 2017-09-08 09:00:00.000000 1 4
9 Worst place ever 2017-09-08 09:00:00.000000 6 3
10 Nice place 2017-09-08 09:00:00.000000 4 2
NULL
review 51
Output
Action Output
Time Action Message
1 20:08:41 select * from review LIMIT 0, 1000 10 row(s) returned
2 20:08:50 select * from review LIMIT 0, 1000 10 row(s) returned

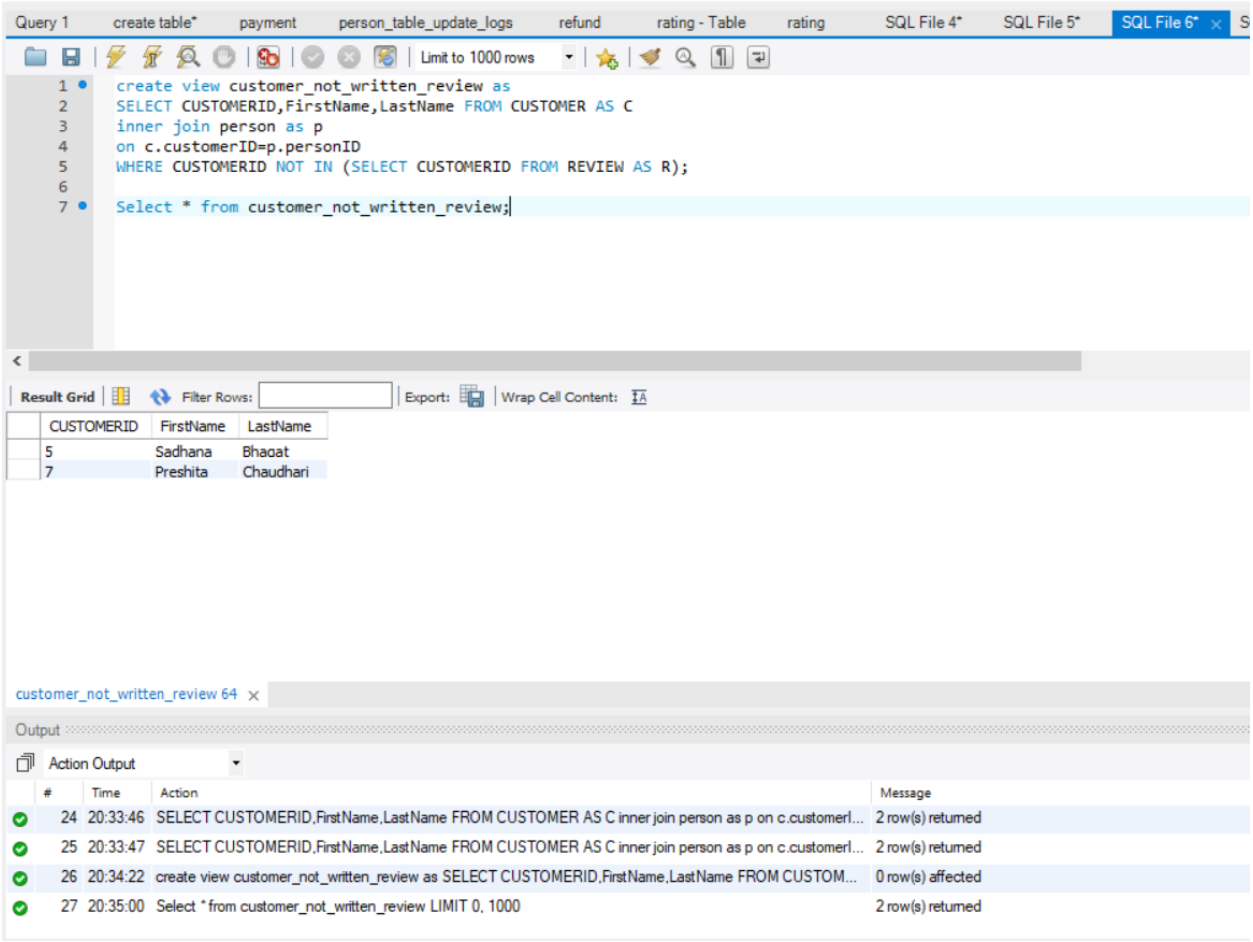


Scenario 19 :

Find out the customers who have not written any reviews.

Query:

```
create view customer_not_written_review as
SELECT CUSTOMERID,FirstName,LastName FROM CUSTOMER AS C
inner join person as p
on c.customerID=p.personID
WHERE CUSTOMERID NOT IN (SELECT CUSTOMERID FROM REVIEW AS R)

Select * from customer_not_written_review;
```

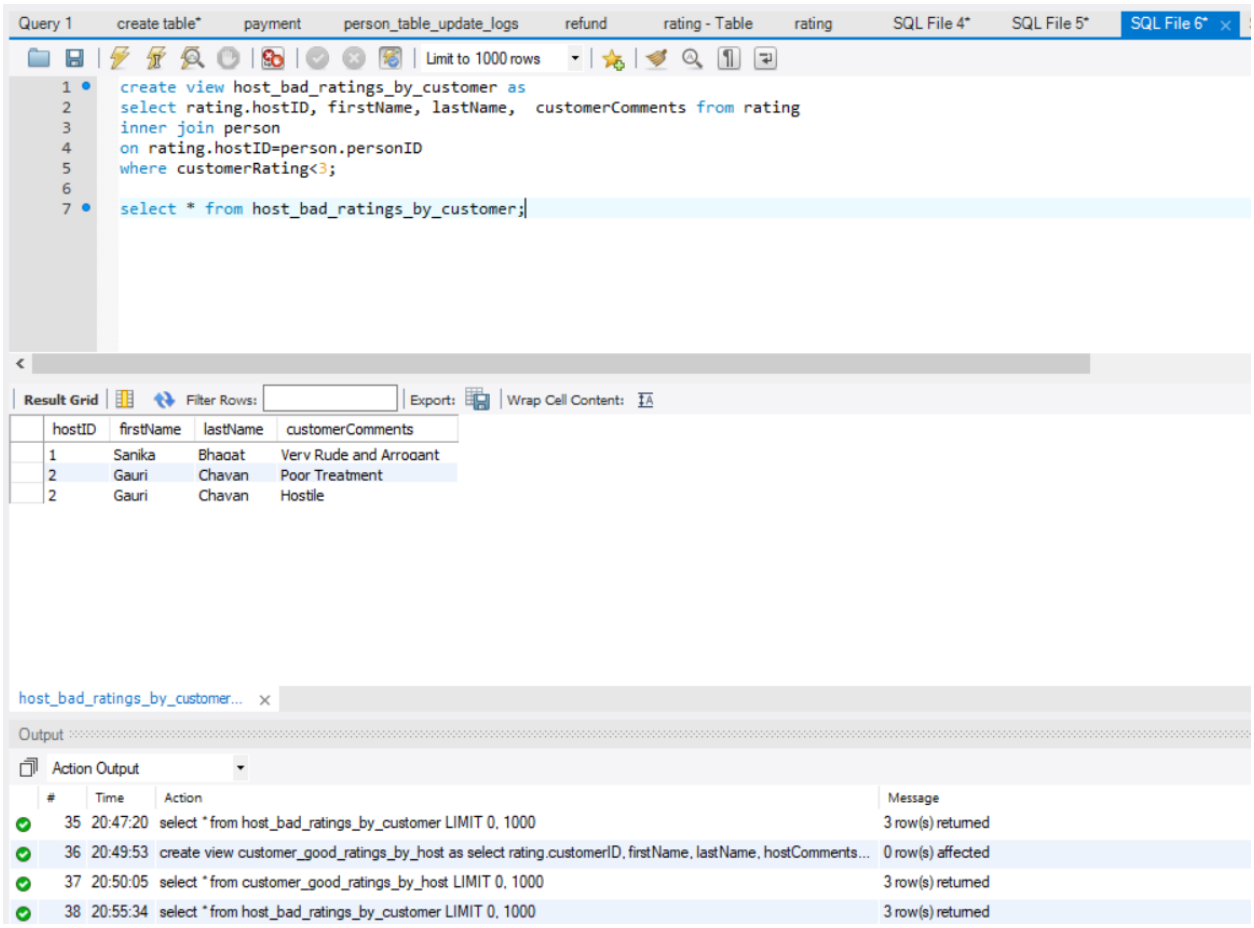


Scenario 20 :

Find out the hosts who have received poor ratings from the customer.

Query :

```
create view host_bad_ratings_by_customer as
select rating.hostID, firstName, lastName,  customerComments from rating
inner join person
on rating.hostID=person.personID
where customerRating<3;

select * from host_bad_ratings_by_customer;
```

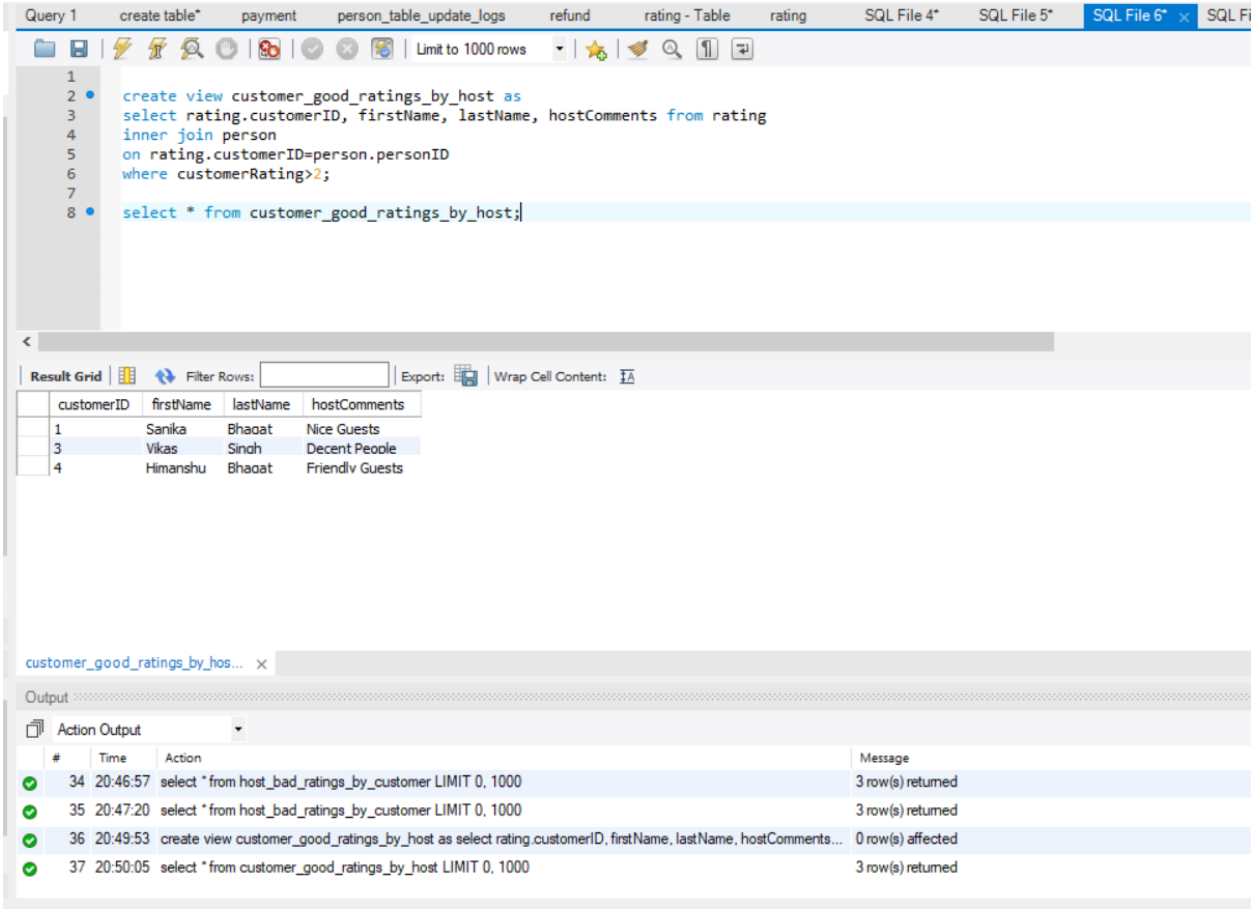


Scenario 21 :

Find out the customers who have received good ratings from host.

Query :

```
create view customer_good_ratings_by_host as
select rating.customerID, firstName, lastName, hostComments from rating
inner join person
on rating.customerID=person.personID
where customerRating>2;

select * from customer_good_ratings_by_host;
```



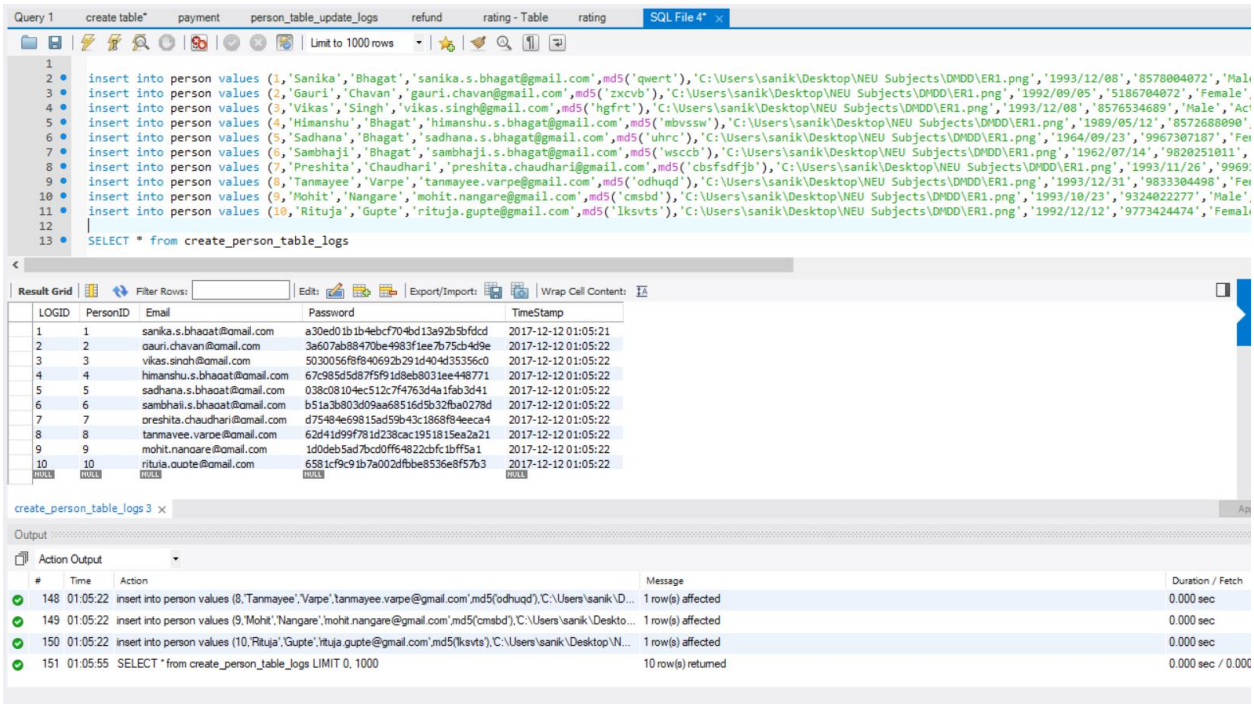
Backup Tables :

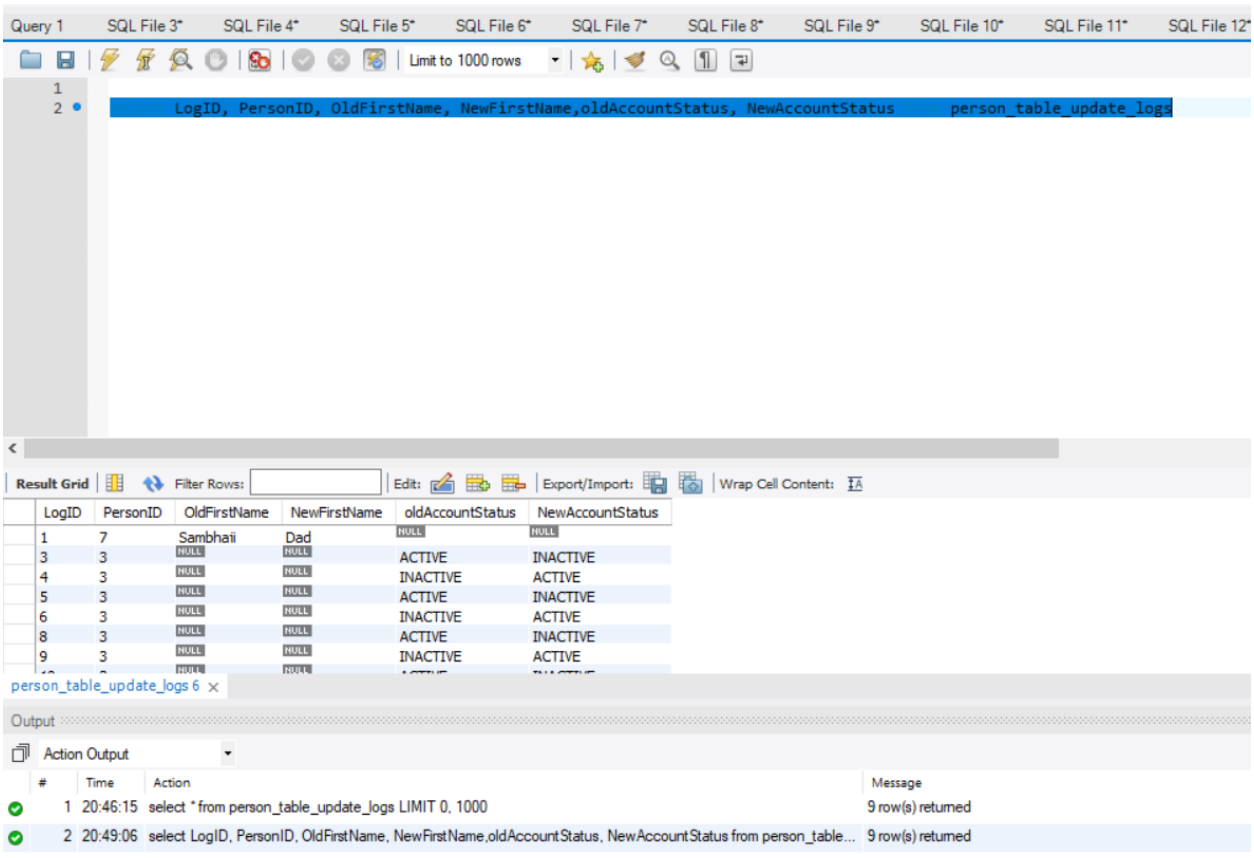
- Log Tables have been created to keep a record of the sensitive information of the organization and to track the changes made to any of the record.

- The log Tables for the Person table are create_person_table_logs which will log any data entry into the Person table.

- The second log table is the person_table_update_logs which will log the changes made to any of the record in the person table.

- This table will keep the old value as well as the new value in the log table for reference.


Query 1
create table*
payment
person_table_update_logs
refund
rating - Table
rating
SQL File 4*
Limit to 1000 rows
insert into person values (1,'Sanika','Bhagat','sanika.s.bhagat@gmail.com',md5('qwert'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1993/12/08','8578004072','Mal
insert into person values (2,'Gauri','Chavan','gauri.chavan@gmail.com',md5('zxcvb'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1992/09/05','5186704072','Female'
insert into person values (3,'Vikas','Singh','vikas.singh@gmail.com',md5('hgfrt'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1993/12/08','8576534689','Male','Ac
insert into person values (4,'Himanshu','Bhagat','himanshu.s.bhagat@gmail.com',md5('mbvssw'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1989/05/12','8572688090'
insert into person values (5,'Sadhana','Bhagat','sadhana.s.bhagat@gmail.com',md5('uhrc'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1964/09/23','9967307187','Fe
insert into person values (6,'Sambhaji','Bhagat','sambhaji.s.bhagat@gmail.com',md5('wsccb'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1962/07/14','9820251011',
insert into person values (7,'Preshita','Chaudhari','preshita.chaudhari@gmail.com',md5('cbsfsdfjb'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1993/11/26','9969
insert into person values (8,'Tanmayee','Varpe','tanmayee.varpe@gmail.com',md5('odhuqd'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1993/12/31','9833304498','Fe
insert into person values (9,'Mohit','Nangare','mohit.nangare@gmail.com',md5('cmsbd'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1993/10/23','9324022277','Male'
insert into person values (10,'Rituja','Gupte','rituja.gupte@gmail.com',md5('lksvts'),'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png','1992/12/12','9773424474','Femal
SELECT * from create_person_table_logs
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
LOGID PersonID Email Password TimeStamp
1 1 sanika.s.bhagat@gmail.com a30ed01b1b4ebcf704bd13a92b5bfdcd 2017-12-12 01:05:21
2 2 gauri.chavan@gmail.com 3a607ab88470be4983f1ee7b75cb4d9e 2017-12-12 01:05:22
3 3 vikas.singh@gmail.com 5030056f8f840692b291d404d35356c0 2017-12-12 01:05:22
4 4 himanshu.s.bhagat@gmail.com 67c985d5d87f5f91d8eb8031ee448771 2017-12-12 01:05:22
5 5 sadhana.s.bhagat@gmail.com 038c08104ec512c7f4763d4a1fab3d41 2017-12-12 01:05:22
6 6 sambhaji.s.bhagat@gmail.com b51a3b803d09aa68516d5b32fba0278d 2017-12-12 01:05:22
7 7 preshita.chaudhari@gmail.com d75484e69815ad59b43c1868f84eeca4 2017-12-12 01:05:22
8 8 tanmayee.varpe@gmail.com 62d41d99f781d238cac1951815ea2a21 2017-12-12 01:05:22
9 9 mohit.nangare@gmail.com 1d0deb5ad7bcd0ff64822cbfc1bff5a1 2017-12-12 01:05:22
10 10 rituja.gupte@gmail.com 6581cf9c91b7a002dfbbe8536e8f57b3 2017-12-12 01:05:22
create_person_table_logs 3
Output
Action Output
Time Action Message Duration / Fetch
148 01:05:22 insert into person values (8,'Tanmayee','Varpe','tanmayee.varpe@gmail.com',md5('odhuqd'),'C:\Users\sanik\D... 1 row(s) affected 0.000 sec
149 01:05:22 insert into person values (9,'Mohit','Nangare','mohit.nangare@gmail.com',md5('cmsbd'),'C:\Users\sanik\Deskto... 1 row(s) affected 0.000 sec
150 01:05:22 insert into person values (10,'Rituja','Gupte','rituja.gupte@gmail.com',md5('lksvts'),'C:\Users\sanik\Desktop\N... 1 row(s) affected 0.000 sec
151 01:05:55 SELECT * from create_person_table_logs LIMIT 0, 1000 10 row(s) returned 0.000 sec / 0.000


Query 1
SQL File 3*
SQL File 4*
SQL File 5*
SQL File 6*
SQL File 7*
SQL File 8*
SQL File 9*
SQL File 10*
SQL File 11*
SQL File 12*
Limit to 1000 rows
LogID, PersonID, OldFirstName, NewFirstName,oldAccountStatus, NewAccountStatus person_table_update_logs
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
LogID PersonID OldFirstName NewFirstName oldAccountStatus NewAccountStatus
1 7 Sambhaji Dad
3 3 ACTIVE INACTIVE
4 3 INACTIVE ACTIVE
5 3 ACTIVE INACTIVE
6 3 INACTIVE ACTIVE
8 3 ACTIVE INACTIVE
9 3 INACTIVE ACTIVE
person_table_update_logs 6
Output
Action Output
Time Action Message
1 20:46:15 select * from person_table_update_logs LIMIT 0, 1000 9 row(s) returned
2 20:49:06 select LogID, PersonID, OldFirstName, NewFirstName,oldAccountStatus, NewAccountStatus from person_table... 9 row(s) returned

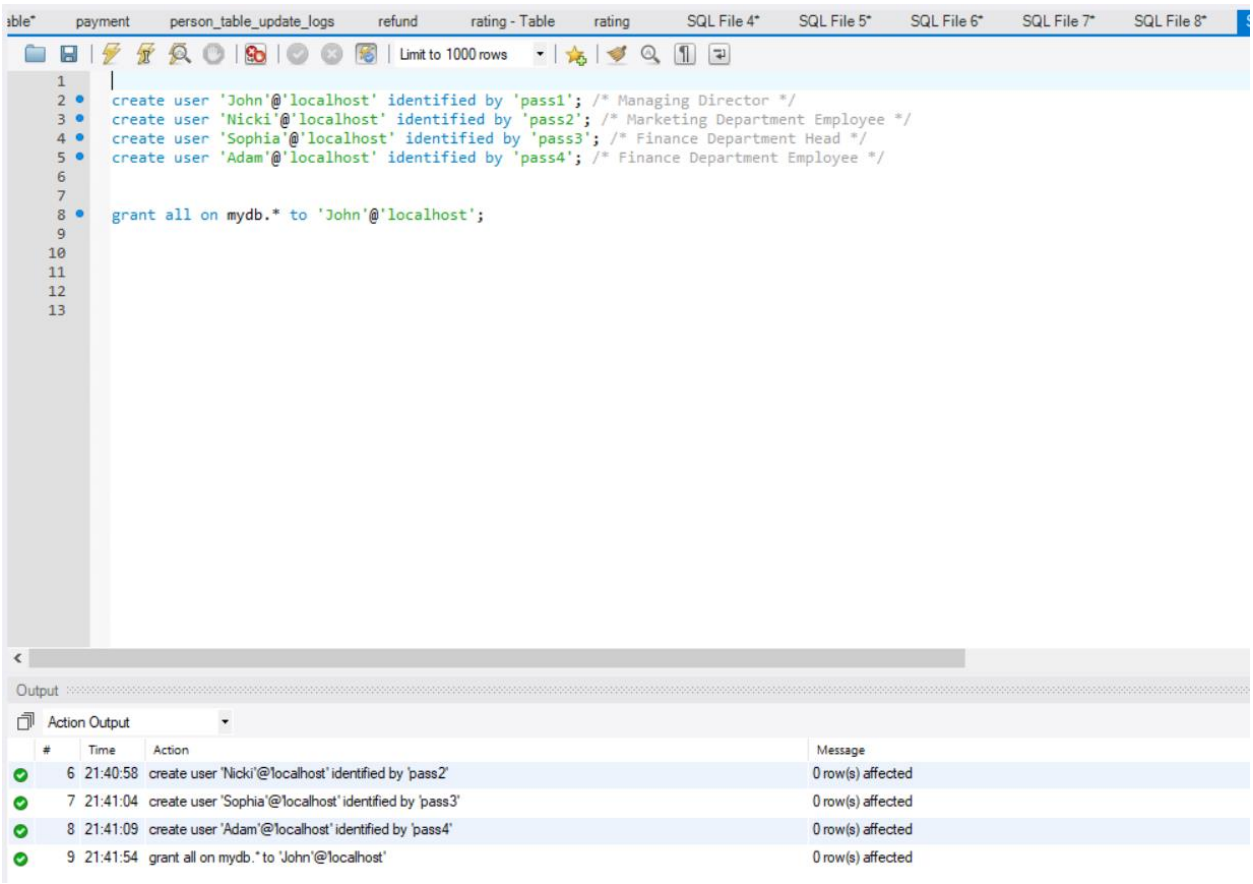
Users and Privileges:

The four Users of the AIRBNB Database John, Nicki, Sophia and Adam.

The users and created and they are identified by username and password.

Creating Users :

create user 'John'@'localhost' identified by 'pass1'; /* Managing Director */
create user 'Nicki'@'localhost' identified by 'pass2'; /* Marketing Department Employee */
create user 'Sophia'@'localhost' identified by 'pass3'; /* Finance Department Head */
create user 'Adam'@'localhost' identified by 'pass4'; /* Finance Department Employee */



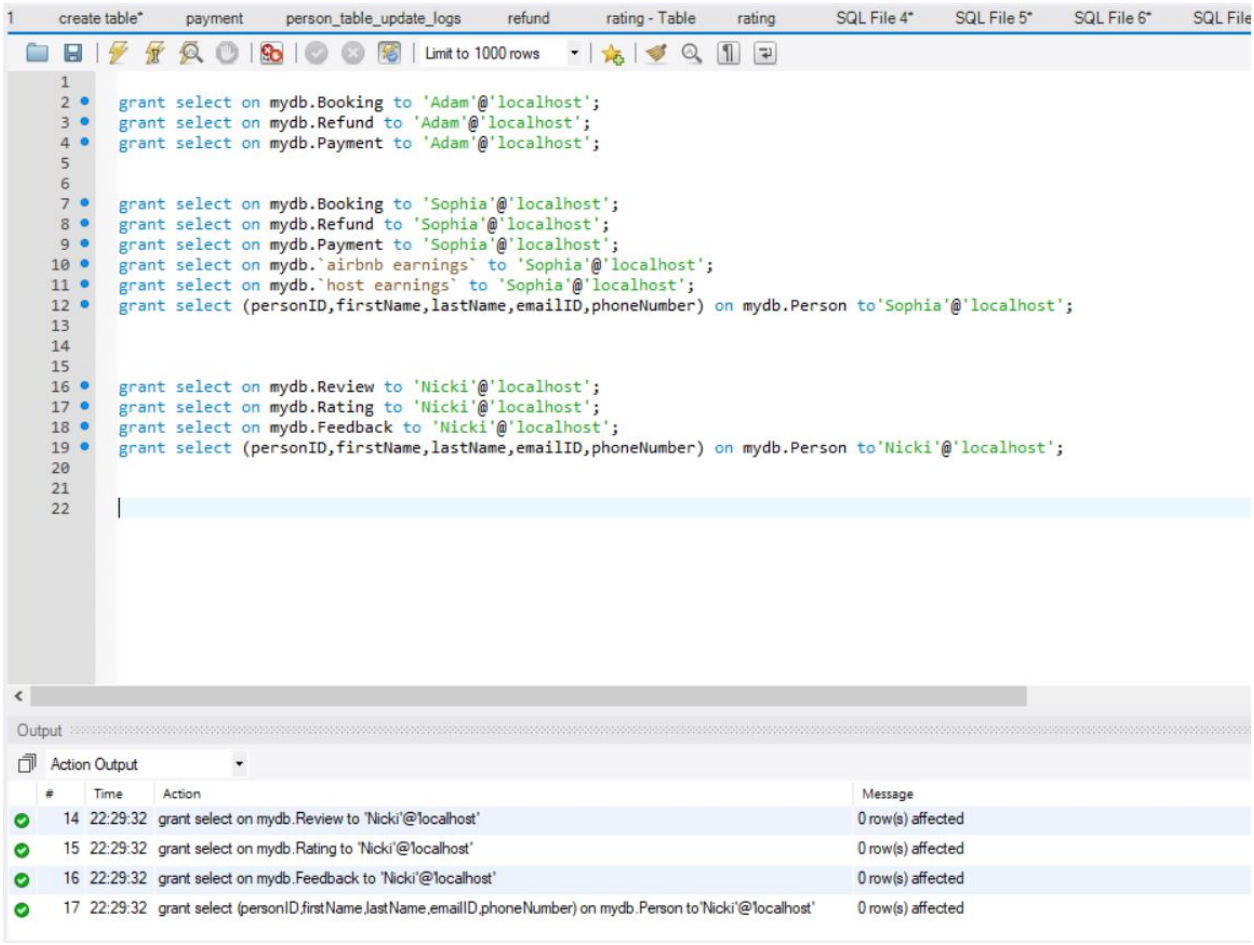
Now all the Users are granted privileges depending upon their Roles at AIRBNB.

Granting Privileges:

grant select on mydb.Booking to 'Adam'@'localhost';
grant select on mydb.Refund to 'Adam'@'localhost';
grant select on mydb.Payment to 'Adam'@'localhost';

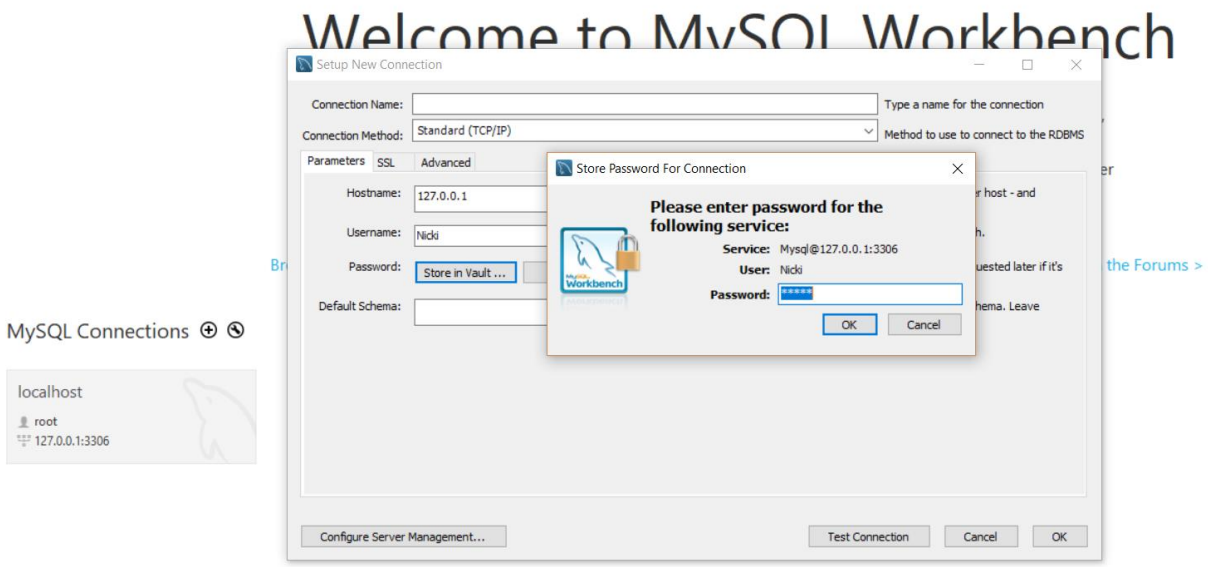
```
grant select on mydb.Booking to 'Sophia'@'localhost';
grant select on mydb.Refund to 'Sophia'@'localhost';
grant select on mydb.Payment to 'Sophia'@'localhost';
grant select on mydb.`airbnb earnings` to 'Sophia'@'localhost';
grant select on mydb.`host earnings` to 'Sophia'@'localhost';
grant select (personID,firstName,lastName,emailID,phoneNumber) on mydb.Person to'Sophia'@'localhost';
```

```
grant select on mydb.Review to 'Nicki'@'localhost';
grant select on mydb.Rating to 'Nicki'@'localhost';
grant select on mydb.Feedback to 'Nicki'@'localhost';
grant select (personID,firstName,lastName,emailID,phoneNumber) on mydb.Person to'Nicki'@'localhost';
```



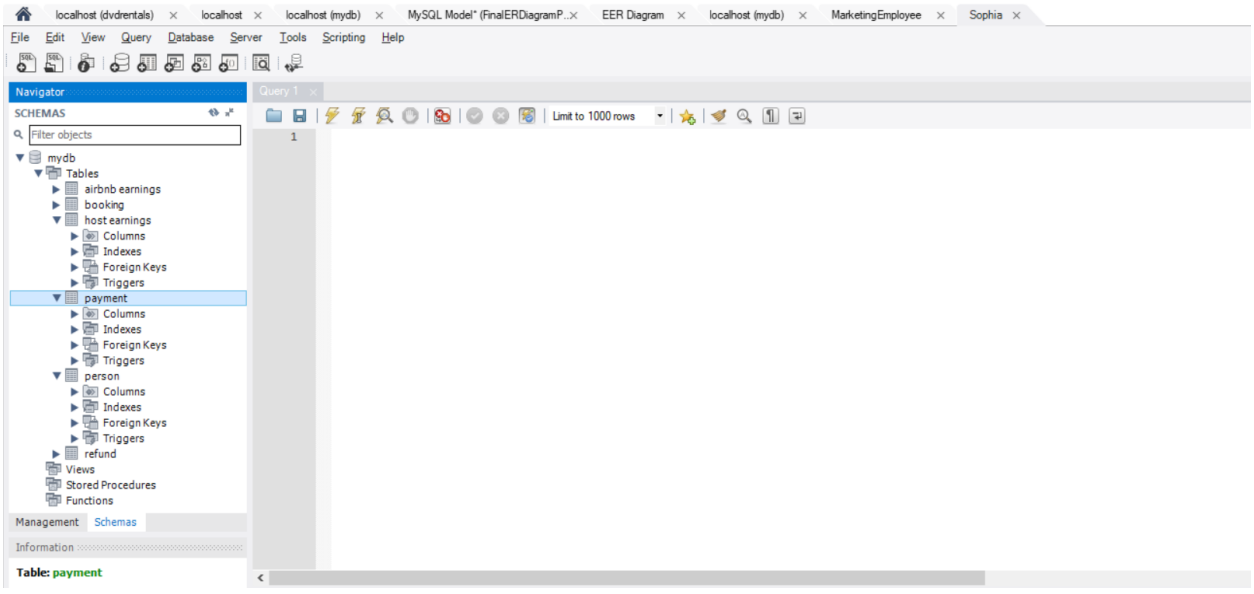
Now when the Nicki logs in, her username and password is accepted.

Login As a User :

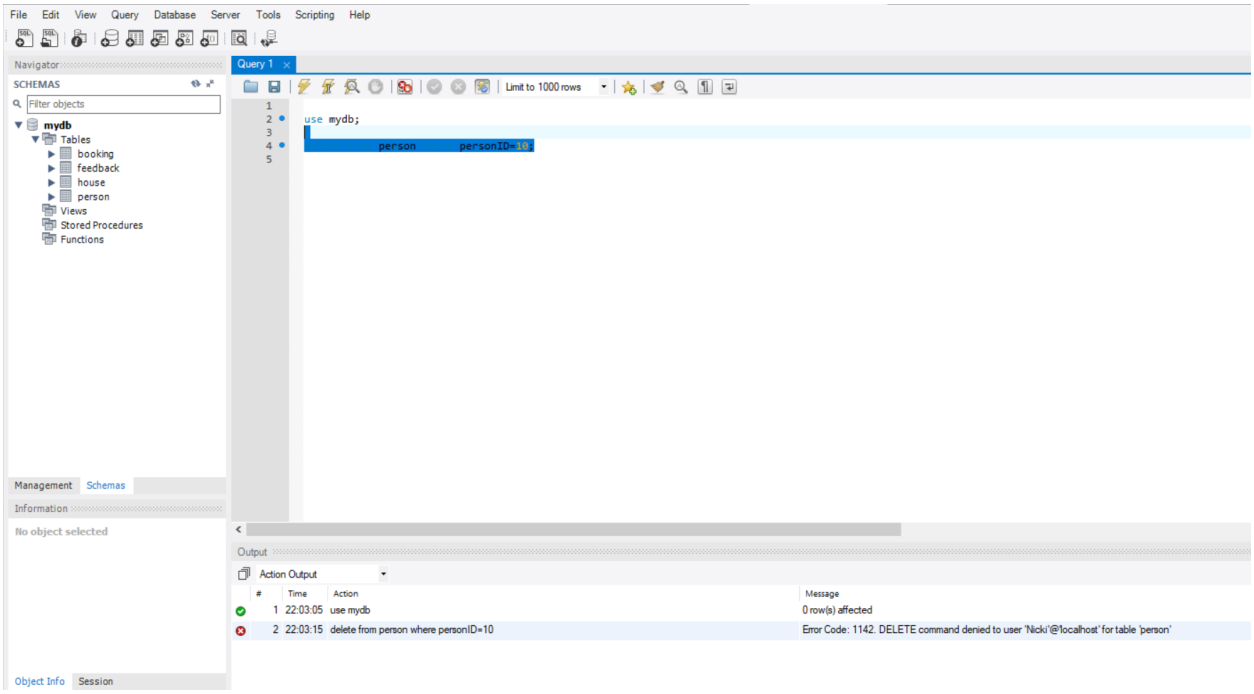


User Sophia can view the tables or the columns that they have been granted privilege on and can perform actions depending upon the privilege that they have been granted.

User Privileges and Column Privileges :

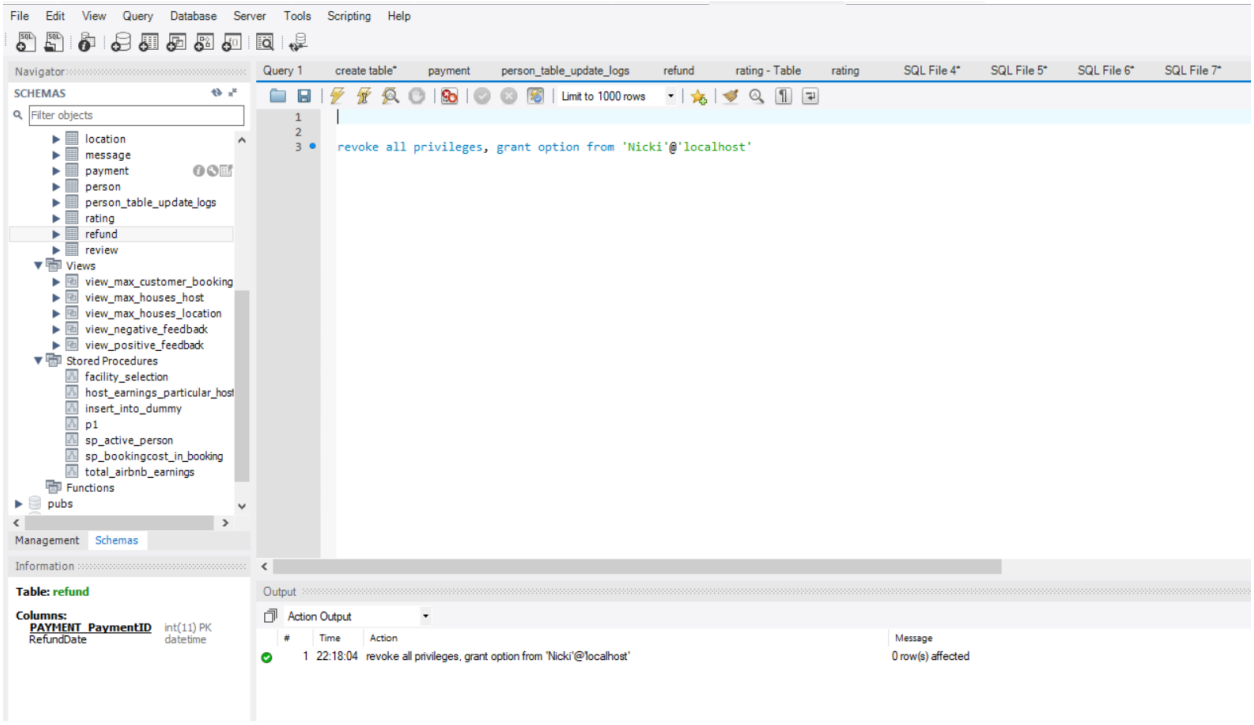


If a user does not certain privilege, the permission is denied.



Privileges of Nicki are Revoked.

Revoke Privileges :



After Nicki again logs in, she cannot see any tables.

Login After Revoke Privileges :