

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.

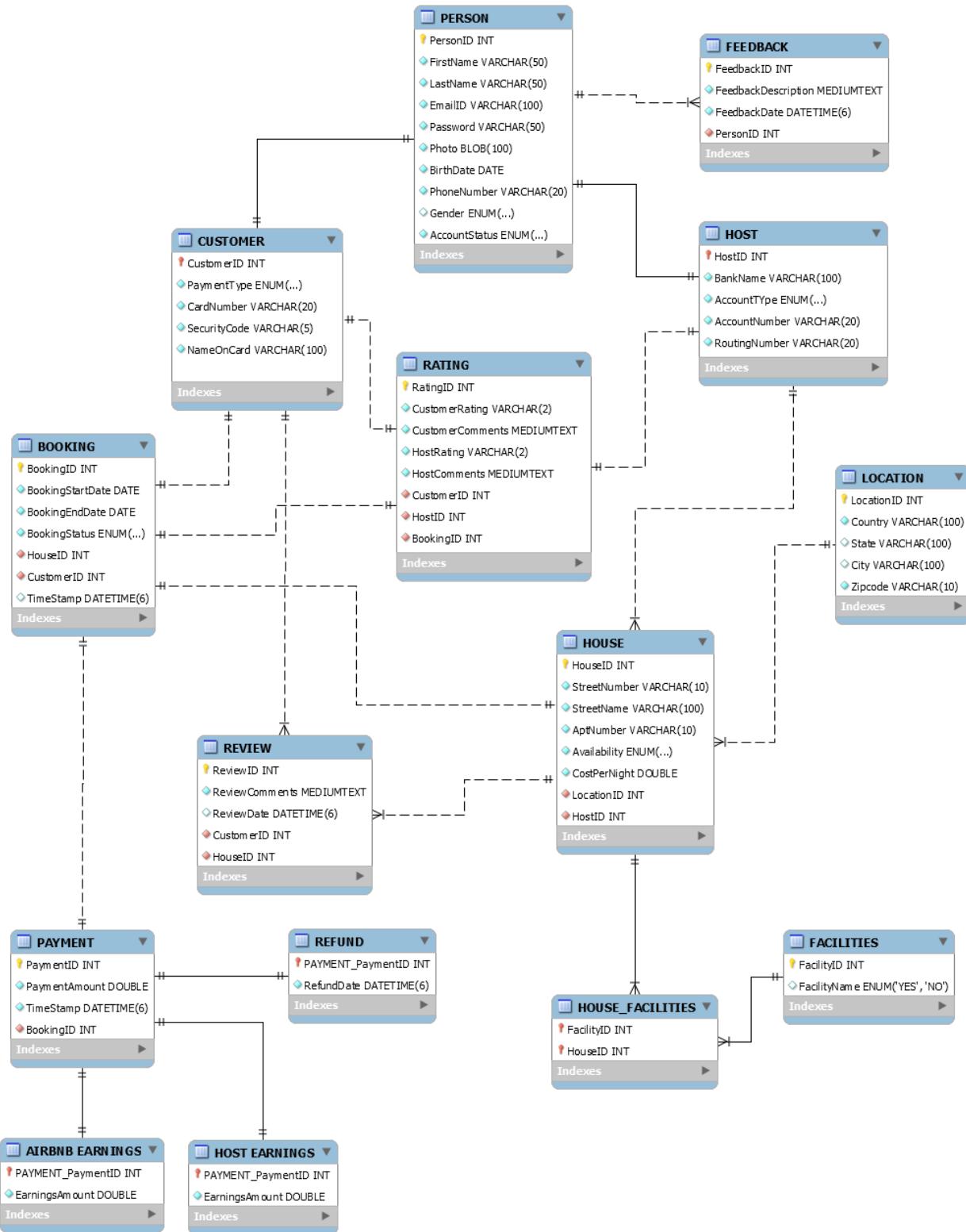
As an Enterprise :

- 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.

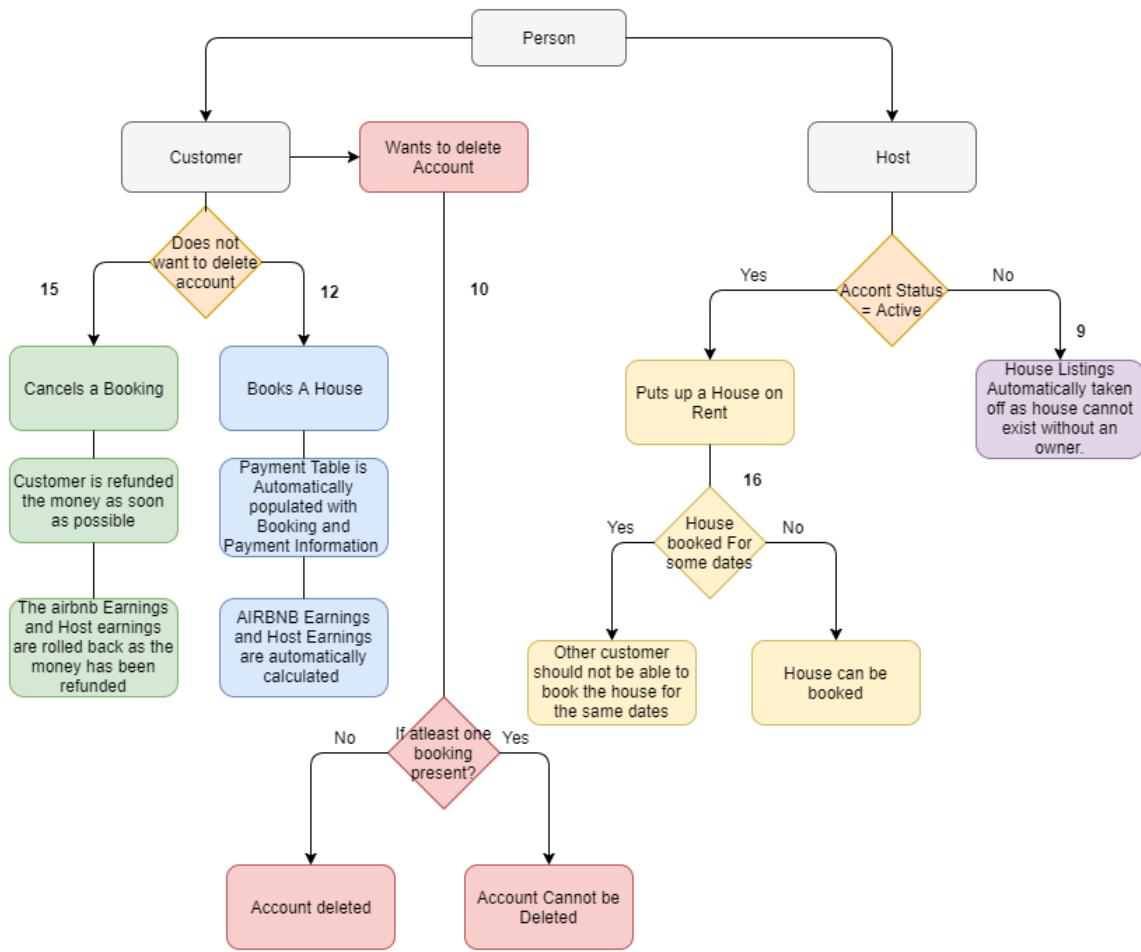
Other Business Rules :

- 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 :



Workflow :



List Of Tables :

mydb	
Tables	
▶	airbnb earnings
▶	booking
▶	booking_dummy
▶	create_person_table_logs
▶	customer
▶	facilities
▶	feedback
▶	host
▶	host earnings
▶	house
▶	house_facilities
▶	location
▶	message
▶	payment
▶	person
▶	person_table_update_logs
▶	rating
▶	refund
▶	review

List of Stored Procedures And Views :

mydb	
Views	
▶	view_max_customer_booking
▶	view_max_houses_host
▶	view_max_houses_location
▶	view_negative_feedback
▶	view_positive_feedback
Stored Procedures	
▶	facility_selection
▶	host_earnings_particular_host
▶	insert_into_dummy
▶	p1
▶	sp_active_person
▶	sp_bookingcost_in_booking
▶	total_airbnb_earnings

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)

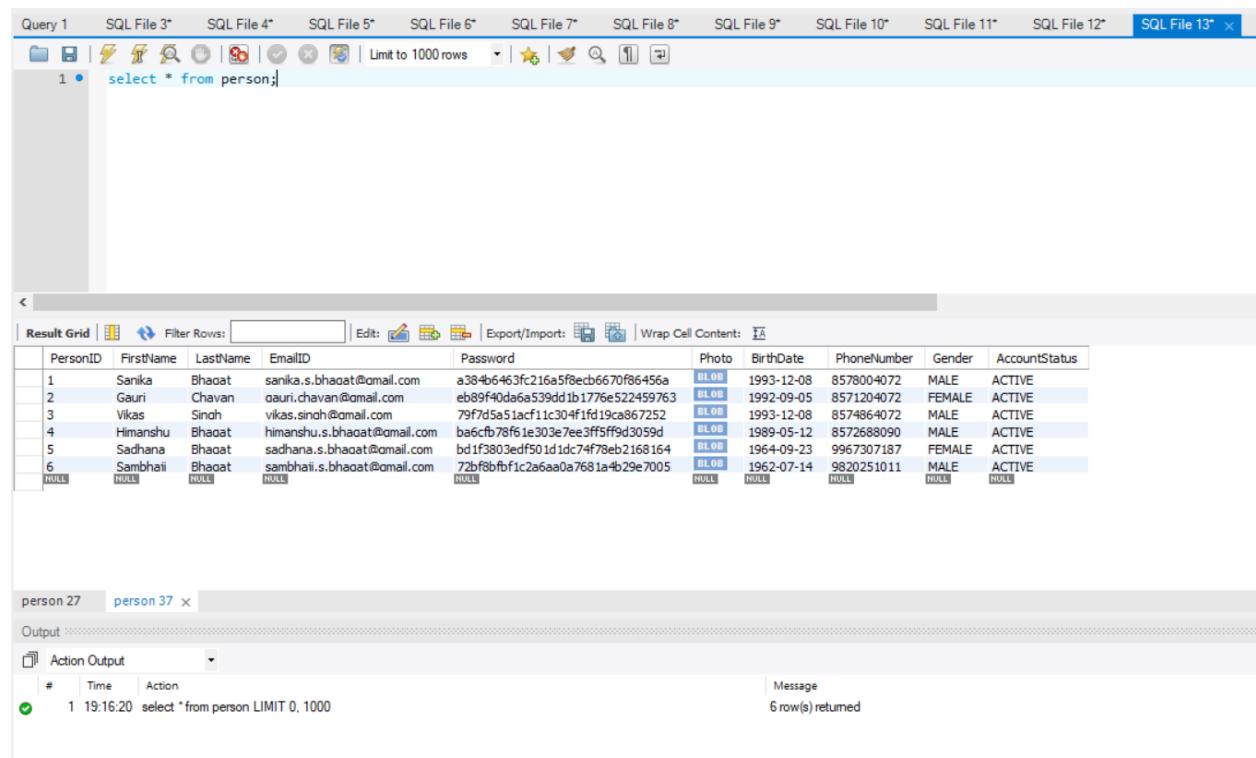
Scenarios :

Scenario 1 :

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

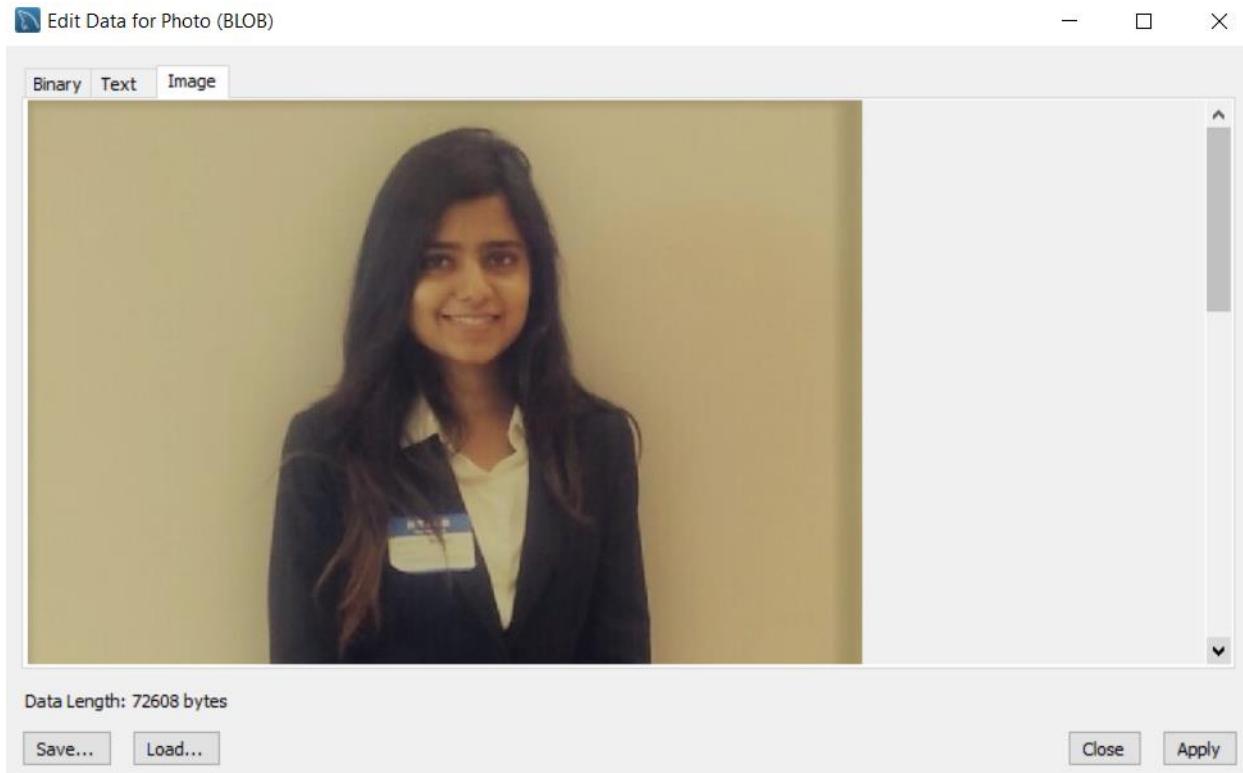
Query :

Select * from person;



The screenshot shows the SSMS interface with the following details:

- Query Editor:** The query `select * from person;` is entered in the top pane.
- Result Grid:** The bottom pane displays the results of the query as a grid. The columns are: PersonID, FirstName, LastName, EmailID, Password, Photo, BirthDate, PhoneNumber, Gender, and AccountStatus.
- Data:** The grid contains 6 rows of data, each representing a person record. The 'Password' column shows encrypted values like `a384b6463fc216a5f8ecb6670f86456a` and `eb89f40da6a539dd1b1776e522459763`. The 'Photo' column shows binary data labeled as 'BLOB'. The 'BirthDate' column shows dates like '1993-12-08' and '1992-09-05'. The 'Gender' column shows 'MALE' and 'FEMALE'. The 'AccountStatus' column shows 'ACTIVE'.
- Output Window:** Below the result grid, the output window shows the command: `1 19:16:20 select * from person LIMIT 0, 1000` and the message: `6 row(s) returned`.



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));
```

The screenshot shows a SQL query editor interface with the following details:

- Query Bar:** Shows the current query number (Query 1) and a list of other open queries: 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*, and SQL File 13*.
- Toolbar:** Includes standard icons for file operations (New, Open, Save, Print, etc.) and search functions.
- Text Area:** Displays the SQL query:

```

1 • select firstname as 'Person First Name', lastName as 'Person Last Name', emailID as 'Person EMail ID', PhoneNumber as 'Person Phone Number' from person
2 where personID in (select customerID from customer where customerID in (select hostID from host where customerID=hostID));
3

```
- Result Grid:** A table showing the results of the query:

Person First Name	Person Last Name	Person EMail ID	Person Phone Number
Sanika	Bhaat	sanika.s.bhaat@gmail.com	8578004072
Gauri	Chavan	gauri.chavan@gmail.com	8571204072
Vikas	Sindh	vikas.sindh@gmail.com	8574864072
- Output Panel:** Labeled "person 3" and "Output". It shows the action history:

#	Time	Action	Message
1	18:03:58	select firstname as 'Person First Name', lastName as 'Person Last Name', emailID as 'Person EMail ID', PhoneNu...	3 row(s) returned

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;

```

The screenshot shows a SQL query window with the following content:

```

1
2 • create view view_positive_feedback
3
4 as
5 select * from feedback
6 where feedbackDescription like '%good%' or feedbackDescription like '%best%';
7
8 • select * from view_positive_feedback;
9

```

Below the query window is a Result Grid displaying the results of the executed query:

FeedbackID	FeedbackDescription	FeedbackDate	PersonID
1	AirBNB is good	2017-12-03 19:14:02.000000	1
4	AirBNB provides good services	2017-12-03 20:28:54.000000	3
5	AirBNB is good	2017-12-03 20:28:54.000000	1

At the bottom, the Output pane shows the command run and the message:

Action Output

#	Time	Action
1	18:21:48	select * from view_positive_feedback LIMIT 0, 1000

Message

3 row(s) returned

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;

```

The screenshot shows a SQL query editor interface with multiple tabs at the top labeled 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*, and SQL File 13*. Below the tabs is a toolbar with various icons. The main area contains the following SQL code:

```

1 create view view_negative_feedback
2 as
3 select * from feedback
4 where feedbackDescription like '%bad%' or feedbackDescription like '%worse%' or feedbackDescription like '%worst%';
5
6
7 select * from view_negative_feedback;

```

Below the code is a Result Grid showing the output of the query:

FeedbackID	FeedbackDescription	FeedbackDate	PersonID
6	AirBNB is bad	2017-12-11 18:17:19.000000	6
7	AirBNB service is worse than many other websites	2017-12-11 18:18:34.000000	5

At the bottom, there is an Output window titled "view_negative_feedback7 x" showing the action history:

#	Time	Action
1	18:24:53	select * from view_negative_feedback LIMIT 0, 1000

Message: 2 row(s) returned.

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;

```

The screenshot shows a SQL query editor interface with the following details:

- Query Editor:** The top bar has tabs for "Query 1" and "SQL File 3*" through "SQL File 13*". Below the tabs is a toolbar with icons for file operations, search, and export.
- Query Text:**

```

1
2 • SELECT feedback.personID, firstName,lastName, COUNT(*) as 'Number of Feedbacks' FROM feedback
3     inner join person
4     on person.personID=feedback.personID
5     GROUP BY personID
6     ORDER BY 'Number of Feedbacks' DESC;
7

```
- Result Grid:** Below the query text is a table titled "Result Grid" with the following data:

	personID	firstName	lastName	Number of Feedbacks
1	Sanika	Bhaat	3	
2	Gauri	Chavan	1	
3	Vikas	Sindh	1	
5	Sadhana	Bhaat	1	
6	Sambhaii	Bhaat	1	
- Output:** A section titled "Output" shows the execution log:

Action Output			
#	Time	Action	Message
1	18:37:08	SELECT feedback.personID, firstName,lastName, COUNT(*) as 'Number of Feedbacks' FROM feedback inner j...	5 row(s) returned

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;

```

The screenshot shows a SQL Server Management Studio interface. The top bar has tabs for 'Query 1' through 'SQL File 13*'. Below the tabs is a toolbar with icons for file operations, search, and other utilities. The main area contains a code editor with the following SQL script:

```

1
2 •  create view view_max_houses_location
3   as
4   SELECT house.LocationID, country,state, city, COUNT(*) as 'Number of Houses' FROM house
5   inner join location
6   on house.locationID=location.locationID
7   GROUP BY locationID
8   ORDER BY 'Maximum Houses';
9
10 |
11 •  select * from view_max_houses_location;
12
13

```

Below the code editor is a 'Result Grid' table with the following data:

	LocationID	country	state	city	Number of Houses
110	US	New York	Huntington Station	1	
111	US	Arizona	Phoenix	3	
112	US	Massachusetts	Boston	2	

At the bottom of the interface is an 'Output' window titled 'view_max_houses_location 25 x'.

#	Time	Action	Message
1	18:53:06	create view view_max_houses_location as SELECT house.LocationID, country,state, city, COUNT(*) as 'Numb...	0 row(s) affected
2	18:53:14	select * from view_max_houses_location LIMIT 0, 1000	3 row(s) returned

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;

```

The screenshot shows the SQL Server Management Studio interface. In the top tab bar, 'Query 1' is selected, followed by tabs for various SQL files. Below the tabs is a toolbar with icons for file operations, search, and refresh.

The main area contains the following SQL code:

```

1
2 •  create view view_max_houses_host
3   as
4   SELECT hostID, FirstName,lastName, COUNT(*) as NumberofHouses FROM house
5   inner join person
6   on house.hostID=person.personID
7   GROUP BY hostID
8   ORDER BY NumberofHouses desc
9   limit 1;
10
11
12 •  select * from view_max_houses_host;
13

```

Below the code is a 'Result Grid' showing the output of the query:

hostID	FirstName	lastName	NumberofHouses
2	Gauri	Chavan	3

At the bottom, the 'Output' window shows the execution log for 'person 27' and 'view_max_houses_host 35':

#	Time	Action	Message
1	19:11:18	create view view_max_houses_host as SELECT hostID, FirstName,lastName, COUNT(*) as NumberofHouses F...	0 row(s) affected
2	19:11:26	select * from view_max_houses_host LIMIT 0, 1000	1 row(s) returned

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;

```

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

3
4 • create view view_max_customer_booking
5 as
6 SELECT PersonID, FirstName, lastName, count(*) as NumberofBookings FROM Booking
7 inner join Person
8 on Booking.CustomerID=person.personID
9 GROUP BY CustomerID
10 ORDER BY NumberofBookings desc
11 ;
12
13
14 • select * from view_max_customer_booking
15 |

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

PersonID	FirstName	lastName	NumberofBookings
1	Sanika	Bhaat	2
4	Himanshu	Bhaat	2
2	Gauri	Chavan	1
3	Vikas	Sindh	1

view_max_customer_booking 12 x

Output ::::::::::::::::::::

Action Output

#	Time	Action	Message
148	17:54:53	select *from booking LIMIT 0, 1000	6 row(s) returned
149	17:57:31	create view view_max_customer_booking as SELECT PersonID, FirstName, lastName, count(*) as NumberofB...	0 row(s) affected
150	17:57:40	select *from view_max_customer_booking LIMIT 0, 1000	4 row(s) returned
151	17:58:04	select *from view_max_customer_booking LIMIT 0, 1000	4 row(s) returned

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* ×

1 2 • 3 4 |

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 Canyon	54	AVAILABLE	75	111	2
4	23	Arizona Grand	2	AVAILABLE	120	111	3
5	27	Cave Creek	1024	AVAILABLE	20	111	2
6	75	Citiview	612	AVAILABLE	83	112	2
HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

person 27 person 37 house 39 ×

Output :::::

Action Output

#	Time	Action
1	19:30:05	select * from house LIMIT 0, 1000

Message
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* ×

1 2 • |

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

HostID	BankName	AccountType	AccountNumber	RoutingNumber
1	Santander	CHECKING	36552587167	75908686
2	BOFA	CHECKING	02812679024	38051212
3	Citi	CHECKING	97487333861	65388827
HULL	HULL	HULL	HULL	HULL

person 27 person 37 host 40 ×

Output :::::

Action Output

#	Time	Action
1	19:30:05	select * from house LIMIT 0, 1000
2	19:31:01	select * from host LIMIT 0, 1000

Message
6 row(s) returned

Message
3 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*** ×

1
2
3 • update person set accountStatus = 'inactive' where personId=3;
4
5 • 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	Bhaoot	sanika.s.bhaoot@gmail.com	a384b6463fc216a5f8e6b6670f86456a	BLOB	1993-12-08	8576004072	MALE	ACTIVE
2	Gauri	Chavan	gauri.chavan@gmail.com	eb89f40da6a539dd1b1776e522459763	BLOB	1992-09-05	8571204072	FEMALE	ACTIVE
3	Vikas	Sindh	vikas.sindh@gmail.com	79f7d5a51acf1c304f1fd19ca867252	BLOB	1993-12-08	8574864072	MALE	INACTIVE
4	Himanshu	Bhaoot	himanshu.s.bhaoot@gmail.com	ba6cfb78f61e303e7ee3ff5ff93059d	BLOB	1989-05-12	8572688090	MALE	ACTIVE
5	Sadhana	Bhaoot	sadhana.s.bhaoot@gmail.com	bd1f3803edf501d1dc74f78eb2168164	BLOB	1964-09-23	9967307187	FEMALE	ACTIVE
6	Sambhali	Bhaoot	sambhali.s.bhaoot@gmail.com	72bf8fbf1c2a6aa0a7681a4b29e7005	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

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*** ×

1
2
3 • update person set accountStatus = 'inactive' where personId=3;
4
5 • select * from person;
6
7 • 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	Cityview	612	AVAILABLE	83	112	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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* SQL File

1
2 • select * from booking;
3
4 |

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

	BookingID	BookingStartDate	BookingEndDate	BookingStatus	HouseID	CustomerID	finalCost	timestamp
1	1	2017-12-08	2017-12-12	OPEN	1	1	-200	NULL
2	2	2017-12-08	2017-12-12	OPEN	2	3	-400	NULL
3	3	2017-12-08	2017-12-12	OPEN	3	1	-300	2017-12-10 14:24:15
4	4	2017-12-08	2017-12-12	OPEN	1	1	-200	2017-12-10 14:40:42
	HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

booking 4 x

Output ::

Action Output

#	Time	Action
1	20:16:16	select * from booking LIMIT 0, 1000

Message
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*

```
1 •  create table message
2   (
3     messagename varchar(200)
4   );
5
6
7
8 delimiter //
9 •  create trigger delete_customer_without_booking
10 after update
11 on person
12 for each row
13 begin
14   if new.accountStatus= 'inactive'
15   then
16     if (new.personID in (Select customerID from booking))
17     then
18       insert into message values ('Cannot delete customer');
19     end if;
20     else
21       delete from booking where booking.customerID=new.personID;
22     end if;
23   end
24   //
25
26
27 L
```

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* SQL File 13*

1
2
3 • update person set accountStatus = 'inactive' where personId=3;
4
5 L

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* SQL File 13*

1
2
3 • select messagename as 'Action' from message;
4
5
6 |

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

```

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 1
1 •  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 Bhaat sanika.s.bhaat@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 Bhaat himanshu.s.bhaat@gmail.com ba6cfb78f61e303e7ee3ff5ff9d3059d BLOB 1989-05-12 8572688090 MALE ACTIVE
5 Sadhana Bhaat sadhana.s.bhaat@gmail.com bd1f3803edf501d1dc74f78eb2168164 BLOB 1964-09-23 9967307187 FEMALE ACTIVE
6 Sambhali Bhaat sambhali.s.bhaat@gmail.com 72bf8fbfb1c2a6aa0a7681a4b29e7005 BLOB 1962-07-14 9820251011 MALE ACTIVE
NULL NULL

```

person 2 x

Output

Action Output

#	Time	Action
1	20:37:53	select * from person LIMIT 0, 1000

Message
6 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');
```

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*** X

File Edit View Insert Tools Help

1
2
3 • update person set accountStatus = 'inactive' where personId=3;
4
5 • 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	Bhaat	sanika.s.bhaat@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	Sindh	vikas.sindh@gmail.com	79f7d5a51acf1c304ffd19ca867252	BLOB	1993-12-08	8574864072	MALE	INACTIVE
4	Himanshu	Bhaat	himanshu.s.bhaat@gmail.com	ba6cfb78f61e303e7ee3ff5ff9d3059d	BLOB	1989-05-12	8572688090	MALE	ACTIVE
5	Sadhana	Bhaat	sadhana.s.bhaat@gmail.com	bd1f3803edf501d1dc74f78eb2168164	BLOB	1964-09-23	9967307187	FEMALE	ACTIVE
6	Sambhai	Bhaat	sambhai.s.bhaat@gmail.com	72bf8fbff1c2a6aa0a7681a4b29e7005	BLOB	1962-07-14	9820251011	MALE	ACTIVE
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

person 27 person 37 **person 41** X

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

File Edit View Insert Tools Help

1 • call sp_active_person('vikas.singh@gmail.com');
2 • 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	Bhaat	sanika.s.bhaat@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	Sindh	vikas.sindh@gmail.com	79f7d5a51acf1c304ffd19ca867252	BLOB	1993-12-08	8574864072	MALE	ACTIVE
4	Himanshu	Bhaat	himanshu.s.bhaat@gmail.com	ba6cfb78f61e303e7ee3ff5ff9d3059d	BLOB	1989-05-12	8572688090	MALE	ACTIVE
5	Sadhana	Bhaat	sadhana.s.bhaat@gmail.com	bd1f3803edf501d1dc74f78eb2168164	BLOB	1964-09-23	9967307187	FEMALE	ACTIVE
6	Sambhai	Bhaat	sambhai.s.bhaat@gmail.com	72bf8fbff1c2a6aa0a7681a4b29e7005	BLOB	1962-07-14	9820251011	MALE	ACTIVE
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

person 9 X

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' where email = ? end	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 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

```

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* X

```

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 x

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* ×

```

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

```

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
NULL	NULL	NULL	NULL

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

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

```

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

```

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

PAYOUT_PaymentID	EarningsAmount
1	140
2	140
3	280
4	105
5	116.19999999999999
6	84
HULL	HULL

airbnb earnings 21 X

Action Output

#	Time	Action	Message
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
36	03:09:25	select * from 'airbnb earnings' 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);

```

The screenshot shows the MySQL Workbench interface. In the top tab bar, 'Query 1' is selected, followed by tabs for 'create table*', 'payment', 'person_table_update_logs', 'refund', 'rating - Table', 'rating', 'SQL File 4*', 'SQL File 5*', and 'SQL File 6*'. The main area contains a code editor with the following SQL script:

```

1
2 delimiter //
3 • create procedure host_earnings_particular_host(in hostID1 int)
4 begin
5 select sum(earningsamount) as 'Total Earnings By This Host' from `host earnings`
6 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)));
7 end
8 //
9
10
11 • call host_earnings_particular_host(1);

```

Below the code editor is a results grid titled 'Result Grid' with the following data:

Total Earnings By This Host
240

On the right side of the interface, there is a sidebar with icons for 'Result Grid', 'Form Editor', and 'Field Types'.

At the bottom left, there is a 'Result 22' tab and an 'Output' section containing a table of action logs:

#	Time	Action	Message	Duration / Fetch
35	03:08:05	select * from payment LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
36	03:09:25	select * from `airbnb earnings` LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
37	03:17:33	create procedure host_earnings_particular_host(in hostID1 int) begin select sum(earningsamount) as 'Total Ear... 0 row(s) affected		0.000 sec
38	03:17:53	call host_earnings_particular_host(1)	1 row(s) returned	0.000 sec / 0.000 sec

A 'Read Only' status indicator is visible next to the output table.

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();
```

The screenshot shows the MySQL Workbench interface with several tabs at the top: Query 1, create table*, payment, person_table_update_logs, refund, rating - Table, rating, SQL File 4*, SQL File 5*, and SQL File 6*. Below the tabs is a toolbar with various icons. The main area contains a code editor with the following SQL script:

```

1
2
3 delimiter //
4 • create procedure total_airbnb_earnings()
5 select sum(EarningsAmount) as 'Total AirBNB Earnings' from `airbnb earnings`
6 end
7 //
8
9
10
11
12
13 •     total_airbnb_earnings();
14

```

The cursor is positioned at the end of the procedure definition. Below the code editor is a result grid showing the output of the procedure:

Total AirBNB Earnings
865.2

At the bottom of the interface, there is a "Result 25" tab with an "Output" section containing a table titled "Action Output". The table has columns for #, Time, Action, and Message. The data is as follows:

#	Time	Action	Message
1	03:32:38	call total_airbnb_earnings()	1 row(s) returned
2	03:33:29	drop procedure total_airbnb_earnings	0 row(s) affected
3	03:33:58	create procedure total_airbnb_earnings() select sum(EarningsAmount) as 'Total AirBNB Earnings'from `airbnb ea...`	0 row(s) affected
4	03:34:03	call total_airbnb_earnings()	1 row(s) returned

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;
```

The screenshot shows the MySQL Workbench interface with several tabs at the top: Query 1, create table*, payment, person_table_update_logs, refund, rating - Table, rating, SQL File 4*, SQL File 5*, and SQL File 6*. Below the tabs is a toolbar with various icons.

In the main query editor area, the code for a stored procedure is displayed:

```

17 after insert
18 on refund
19 for each row
20 begin
21 delete from `airbnb earnings` where Payment_paymentID=new.Payment_paymentID;
22 delete from `host earnings` where Payment_paymentID=new.Payment_paymentID;
23 end;
//
```

Below the stored procedure code, a select statement is shown:

```
29 • select * from booking
```

At the bottom of the query editor, there is a result grid table with the following columns: BookingID, BookingStartDate, BookingEndDate, BookingStatus, HouseholdID, CustomerID, FinalCost, and TimeStamp. The data returned is:

BookingID	BookingStartDate	BookingEndDate	BookingStatus	HouseholdID	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	OPEN	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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Below the result grid, there is an output window titled "booking 28" which shows the execution details:

Action Output

#	Time	Action
1	03:44:31	select * from booking LIMIT 0, 1000

Message: 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*

```
1 • select * from booking;
2
3
4 • update booking set bookingStatus='cancelled' where bookingID=3;
5
6
7 •     *      refund;
8
9 • 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 x

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

```

1 select * from booking;
2
3
4 update booking set bookingStatus='cancelled' where bookingID=3;
5
6
7 select * from refund;
8
9 *      "airbnb earnings";

```

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

PAYOUT_PaymentID	EarningsAmount
1	140
2	140
4	105
5	116.1999999999999
6	84
NULL	NULL

airbnb earnings 5 x

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

```

1 select * from booking;
2
3
4 update booking set bookingStatus='cancelled' where bookingID=3;
5
6
7 select * from refund;
8
9 select * from `airbnb earnings`;
10
11 select * from `host earnings`;

```

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

PAYOUT_PaymentID	EarningsAmount
1	60
2	60
4	45
5	49.8
6	36
NULL	NULL

host earnings 6 x

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*

```

1
2 • select * from booking;
3
4

```

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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

booking 8 x

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*

```

1 • select * from booking
2
3

```

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	NULL	2017-12-12 13:05:22.000000
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

booking 10 x

Output

Action Output

#	Time	Action	Message
1	13:09:27	select * from booking LIMIT 0, 1000	7 row(s) returned

The screenshot shows the MySQL Workbench interface. At the top, there are tabs for various SQL files. Below the tabs is a toolbar with icons for file operations, search, and zoom. A status bar at the bottom indicates "Limit to 1000 rows".

Query Editor:

```

1 • select * from booking;
2
3 • call p1();
4
5 • select * from booking;
    
```

Result Grid:

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
HULL	HULL	HULL	HULL	HULL	HULL	HULL	HULL

Output:

```

booking 11 ×
Output :-----
Action Output
# | Time | Action | Message
1 13:09:27 select *from booking LIMIT 0, 1000 7 row(s) returned
2 13:10:03 call p1() 1 row(s) affected
3 13:10:08 select *from booking LIMIT 0, 1000 6 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');
    
```

The screenshot shows the MySQL Workbench interface. In the top tab bar, 'Query 1' is selected, along with tabs for 'create table*', 'payment', 'person_table_update_logs', 'refund', 'rating - Table', 'rating', 'SQL File 4*', 'SQL File 5*', and 'SQL File 6*'. Below the tabs is a toolbar with icons for file operations, search, and refresh.

```

1 delimiter //
2 create procedure facility_selection(in facility1 varchar(20))
3 begin
4 select * from house
5 where houseID in (select houseID from house_facilities where facilityID = all(select facilityID from facilities where facilityName=facility1));
6 end //
7
8 call facility_selection('workspace');

```

The results grid displays two rows of data:

HouseID	StreetNumber	StreetName	AptNumber	Availability	CostPerNight	LocationID	HostID
2	10	Mav Street	8	AVAILABLE	100	110	1
5	27	Cave Creek	1024	AVAILABLE	20	111	2

The 'Result 37' pane shows the execution history:

#	Time	Action	Message	Durat
42	01:15:54	select * from facilities LIMIT 0, 1000	7 row(s) returned	0.000
43	01:16:04	select facilityID from facilities where facilityName='wifi' and facilityName='workspace' LIMIT 0, 1000	0 row(s) returned	0.000
44	01:18:02	create procedure facility_selection(in facility1 varchar(20)) begin select * from house where houseID in (select ...)	0 row(s) affected	0.016
45	01:18:39	call facility_selection('workspace')	2 row(s) returned	0.015

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* SQL I

1
2
3 • * 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 enjoy 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
HULL	HULL	HULL	HULL	HULL

review 51 x

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

```

1
2
3
4 • select houseID, count(houseID) as 'count' from review
5   group by HouseID
6   order by count desc;
7

```

houseID	count
2	3
3	2
6	2
1	1
4	1
5	1

Result 52 x

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
3	20:09:46	select houseID, count(houseID) as 'count' from review group by HouseID order by count desc LIMIT 0, 1000	6 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;

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

```

1 • create view customer_not_written_review as
2   SELECT CUSTOMERID,FirstName,LastName FROM CUSTOMER AS C
3   inner join person as p
4   on c.customerID=p.personID
5   WHERE CUSTOMERID NOT IN (SELECT CUSTOMERID FROM REVIEW AS R);
6
7 • Select * from customer_not_written_review;

```

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

	CUSTOMERID	FirstName	LastName
5	Sadhana	Bhaoot	
7	Preshita	Chaudhari	

customer_not_written_review 64 x

Output

Action Output	#	Time	Action	Message
24	20:33:46		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);	2 row(s) returned
25	20:33:47		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);	2 row(s) returned
26	20:34:22		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);	0 row(s) affected
27	20:35:00		Select * from customer_not_written_review LIMIT 0, 1000	2 row(s) returned

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;

```

The screenshot shows a SQL Server Management Studio interface. In the top tab bar, 'Query 1' is selected, followed by tabs for 'create table*', 'payment', 'person_table_update_logs', 'refund', 'rating - Table', 'rating', 'SQL File 4*', 'SQL File 5*', and 'SQL File 6*'. The main pane displays the following SQL code:

```

1 • create view host_bad_ratings_by_customer as
2 select rating.hostID, firstName, lastName, customerComments from rating
3 inner join person
4 on rating.hostID=person.personID
5 where customerRating<3;
6
7 • select * from host_bad_ratings_by_customer;

```

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

	hostID	firstName	lastName	customerComments
1	Sanika	Bhadat	Verv Rude and Arroant	
2	Gauri	Chavan	Poor Treatment	
2	Gauri	Chavan	Hostile	

In the bottom pane, titled 'host_bad_ratings_by_customer...', the 'Output' section shows the following log entries:

Action	Time	Action	Message
35	20:47:20	select * from host_bad_ratings_by_customer LIMIT 0, 1000	3 row(s) returned
36	20:49:53	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;	0 row(s) affected
37	20:50:05	select * from customer_good_ratings_by_host LIMIT 0, 1000	3 row(s) returned
38	20:55:34	select * from host_bad_ratings_by_customer LIMIT 0, 1000	3 row(s) returned

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;

```

```

1
2 • create view customer_good_ratings_by_host as
3 select rating.customerID, firstName, lastName, hostComments from rating
4 inner join person
5 on rating.customerID=person.personID
6 where customerRating>2;
7
8 • select * from customer_good_ratings_by_host;

```

customerID	firstName	lastName	hostComments
1	Sanika	Bhaot	Nice Guests
3	Vikas	Singh	Decent People
4	Himanshu	Bhaot	Friendly Guests

customer_good_ratings_by_hos... x

Output

#	Time	Action	Message
34	20:46:57	select * from host_bad_ratings_by_customer LIMIT 0, 1000	3 row(s) returned
35	20:47:20	select * from host_bad_ratings_by_customer LIMIT 0, 1000	3 row(s) returned
36	20:49:53	create view customer_good_ratings_by_host as select rating.customerID, firstName, lastName, hostComments... 0 row(s) affected	0 row(s) affected
37	20:50:05	select * from customer_good_ratings_by_host LIMIT 0, 1000	3 row(s) returned

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*

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

```

1 • insert into person values ('Sanika','Bhagat', 'sanika.s.bhagat@gmail.com',md5('qwert'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1993/12/08', '8578004072', 'Male')
2 • insert into person values ('Gauri','Chavan', 'gauri.chavan@gmail.com',md5('zxcvb'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1992/09/05', '5186704072', 'Female')
3 • insert into person values ('Vikas','Singh', 'vikas.singh@gmail.com',md5('hgfrt'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1993/12/08', '8576534689', 'Male', 'Active')
4 • insert into person values ('Himanshu','Bhagat', 'himanshu.s.bhagat@gmail.com',md5('mbvssw'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1989/05/12', '8572688090', 'Male', 'Active')
5 • insert into person values ('Sadhana','Bhagat', 'sadhana.s.bhagat@gmail.com',md5('uhrc'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1964/09/23', '9967307187', 'Female')
6 • insert into person values ('Samonejai','Bhagat', 'samonejai.s.bhagat@gmail.com',md5('wsccb'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1962/07/14', '9820251011', 'Female')
7 • insert into person values ('Preshita','Chaudhari', 'preshita.chaudhari@gmail.com',md5('cbfsdfjh'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1993/11/26', '9969123456', 'Female')
8 • insert into person values ('Tannayee','Varpe', 'tannayee.varpe@gmail.com',md5('odhnuq'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1993/12/31', '9883304498', 'Female')
9 • insert into person values ('Mohit','Nangare', 'mohit.nangare@gmail.com',md5('cmsbd'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1993/10/23', '9324622277', 'Male')
10 • insert into person values ('Rituja','Gupte', 'rituja.gupte@gmail.com',md5('lksvts'), 'C:\Users\sanik\Desktop\NEU Subjects\DMDD\ER1.png', '1992/12/12', '9773424474', 'Female')
11 •
12 |
13 • SELECT * from create_person_table_logs

```

create_person_table_logs 3 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
148	01:05:22	insert into person values (8,'Tannayee','Varpe','tannayee.varpe@gmail.com',md5('odhnuq'))	C:\Users\sanik\Deskt... 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\Deskt... 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\Deskt... 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*

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

```

1 • LogID, PersonID, OldFirstName, NewFirstName,oldAccountStatus, NewAccountStatus      person_table_update_logs
2 •

```

person_table_update_logs 6 x

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 are 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 */
```

The screenshot shows the MySQL Workbench interface. In the top-left pane, there is a list of tables: 'table*', 'payment', 'person_table_update_logs', 'refund', 'rating - Table', 'rating', 'SQL File 4*', 'SQL File 5*', 'SQL File 6*', 'SQL File 7*', and 'SQL File 8*'. Below this is a SQL editor window containing the following code:

```
1  
2 • create user 'John'@'localhost' identified by 'pass1'; /* Managing Director */  
3 • create user 'Nicki'@'localhost' identified by 'pass2'; /* Marketing Department Employee */  
4 • create user 'Sophia'@'localhost' identified by 'pass3'; /* Finance Department Head */  
5 • create user 'Adam'@'localhost' identified by 'pass4'; /* Finance Department Employee */  
6  
7  
8 • grant all on mydb.* to 'John'@'localhost';  
9  
10  
11  
12  
13
```

In the bottom-right pane, there is an 'Output' tab showing the results of the executed commands:

Action	Output	Message
6	21:40:58 create user 'Nicki'@'localhost' identified by 'pass2'	0 row(s) affected
7	21:41:04 create user 'Sophia'@'localhost' identified by 'pass3'	0 row(s) affected
8	21:41:09 create user 'Adam'@'localhost' identified by 'pass4'	0 row(s) affected
9	21:41:54 grant all on mydb.* to 'John'@'localhost'	0 row(s) affected

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';
```

1 create table* payment person_table_update_logs refund rating - Table rating SQL File 4* SQL File 5* SQL File 6* SQL File 7*

1

2 • grant select on mydb.Booking to 'Adam'@'localhost';

3 • grant select on mydb.Refund to 'Adam'@'localhost';

4 • grant select on mydb.Payment to 'Adam'@'localhost';

5

6

7 • grant select on mydb.Booking to 'Sophia'@'localhost';

8 • grant select on mydb.Refund to 'Sophia'@'localhost';

9 • grant select on mydb.Payment to 'Sophia'@'localhost';

10 • grant select on mydb.`airbnb earnings` to 'Sophia'@'localhost';

11 • grant select on mydb.`host earnings` to 'Sophia'@'localhost';

12 • grant select (personID,firstName,lastName,emailID,phoneNumber) on mydb.Person to 'Sophia'@'localhost';

13

14

15

16 • grant select on mydb.Review to 'Nicki'@'localhost';

17 • grant select on mydb.Rating to 'Nicki'@'localhost';

18 • grant select on mydb.Feedback to 'Nicki'@'localhost';

19 • grant select (personID,firstName,lastName,emailID,phoneNumber) on mydb.Person to 'Nicki'@'localhost';

20

21

22

Output

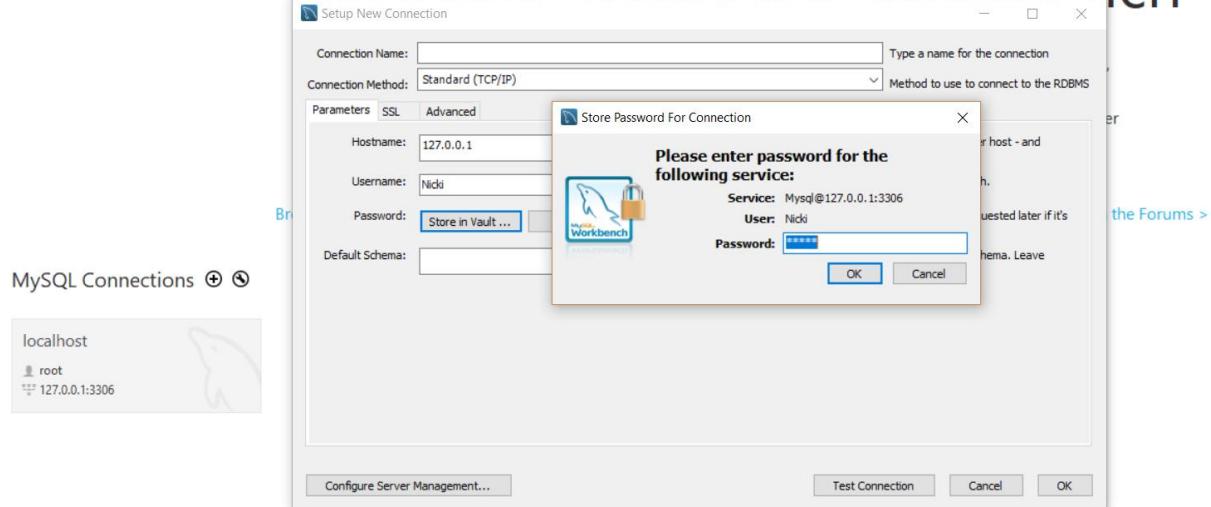
Action Output

#	Time	Action	Message
14	22:29:32	grant select on mydb.Review to 'Nicki'@'localhost'	0 row(s) affected
15	22:29:32	grant select on mydb.Rating to 'Nicki'@'localhost'	0 row(s) affected
16	22:29:32	grant select on mydb.Feedback to 'Nicki'@'localhost'	0 row(s) affected
17	22:29:32	grant select (personID,firstName,lastName,emailID,phoneNumber) on mydb.Person to 'Nicki'@'localhost'	0 row(s) affected

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

Login As a User :

Welcome to MySQL Workbench



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 :

The screenshot shows the MySQL Workbench Navigator pane. The 'Schemas' tree view shows a database named 'mydb' containing several tables: 'airbnb earnings', 'booking', 'host earnings', 'payment', 'person', 'refund', 'Views', 'Stored Procedures', and 'Functions'. Under each table, there are sub-folders for 'Columns', 'Indexes', 'Foreign Keys', and 'Triggers'. The 'payment' table is currently selected. The 'Management' tab is active at the bottom of the Navigator pane.

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

The screenshot shows the SSMS interface. In the top-left, the 'File' menu has 'File' selected. The 'Navigator' pane on the left shows the 'mydb' schema with tables like booking, feedback, house, and person. The 'Query 1' window contains the following SQL script:

```

1 use mydb;
2
3
4
5      person    personID=10;

```

The 'Output' pane at the bottom shows the execution log:

#	Time	Action	Message
1	22:03:05	use mydb	0 row(s) affected
2	22:03:15	delete from person where personID=10	Error Code: 1142. DELETE command denied to user 'Nicki'@localhost for table 'person'

Privileges of Nicki are Revoked.

Revoke Privileges :

The screenshot shows the SSMS interface. In the top-left, the 'File' menu has 'File' selected. The 'Navigator' pane on the left shows the 'refund' schema with various objects like tables, views, stored procedures, and functions. The 'Query 1' window contains the following SQL script:

```

1
2
3      revoke all privileges, grant option from 'Nicki'@localhost';

```

The 'Output' pane at the bottom shows the execution log:

#	Time	Action	Message
1	22:18:04	revoke all privileges, grant option from 'Nicki'@localhost'	0 row(s) affected

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

Login After Revoke Privileges :

