DATABASE SYSTEM AIRBNB PROJECT

TABLE OF CONTENTS

Data Requirements for the System

Services offered by Airbnb

Airbnb is an online marketplace and hospitality service which is accessible via its website and mobile app. Members can use the service to book or offer homestays, or restaurants.

Using the services provided by Airbnb, people can rent out their properties or spare rooms to guests and also restaurant owners can list their restaurant names and services they provide. Airbnb takes a commission for every booking from hosts/restaurant owners.

The entire AirBnb system is subdivided into few functional subsystems.

User Handling Subsystem

Everyone has to create an Airbnb account with a password and an email id. This email id should be unique for each user.

Now each user can register as guest, customer or both. System generates a unique userid for each user to maintain that user in the database. Each user is asked to provide following information to the system.

- First Name
- Last Name
- Gender
- Date of Birth
- Phone Number

Additionally customers of Airbnb services can provide details about themselves mentioning what they are exactly looking for. Similarly, hosts can also provide details about themselves.

A customer earns reward points based on the amount they spent on Airbnb platform. A host can also earn bonus amounts, based on the ratings their properties and services receive. These rewards and bonuses are redeemable in form of cash or Airbnb credit.

Property Handling Subsystem

Two types of properties can be listed in Airbnb system, houses and restaurants. System generates a unique id for every property. For each property, the host is asked to provide following information,

- Type of the property
- Address
- City
- State
- Country
- Zip
- Available Capacity
- Price

- Special offer

For houses, price is the price per night. For restaurants, price is a fee for reserving seats in advance. Appropriate taxes are exclusive of shown prices. Each type of properties can have multiple amenities.

A house can list following amenities,

- Air-Conditioning
- WiFi
- Private Bathroom
- Bed
- Kitchen
- Gym
- Pool
- Parking
- Washer
- Cleaner

A restaurant can list following amenities and additional details,

- Cuisine
- Menu
- Bar
- Total Seats
- Available Seats

User can search a property by selecting one or many of these details.

Bank and Card Handling Subsystem

A customer pays using a credit/debit card and the money is transferred to the bank account of the host. Airbnb collects a flat 3% commission for each successful transaction. A customer can save details of up to 3 cards and a host can save details of up to 3 bank accounts. One of the card or bank account has to be set primary, but that can be changed at any time.

To save a card, following information are needed,

- Card number
- Card type
- Card holders name
- Expiry date

It is not mandatory to save card information. A user can choose to enter card information at the time of a transaction and then not save it.

For bank details, following information are needed,

- Bank account number
- Routing number

- Account holders name

A host must save at least one bank account details.

Booking Order Handling Subsystem

Booking order handing subsystem asks for following information from the customer,

- Property to be booked
- Reservation date
- Guest count
- Reservation date
- Moving out date

This subsystem also fetches the order date, property price to complete the transaction. Once a transaction is successful a unique order id is generated. Host can provide the check in and check out time of the guest into the system.

Now every booking is cancellable by the customer. If a booking is cancelled 30 days before the reservation date, full money is refunded. Cancellation with in 2 days of reservation date or after reservation date is non-refundable. Otherwise 50% of the booking amount is refunded. Host can not cancel a booking once done, unless any severe condition occurs, which is subject to approval from Airbnb and in that case Airbnb refunds the total amount to the customer. The transaction is marked as cancelled and the cancellation date and a unique refund id is stored. Customer can choose to add the refund amount as reward points or get refunded in the card account that was used for payment.

Payment Handling Subsystem

Payment handling subsystem is an internal subsystem and not exposed to user. It fetches following information and executes a payment instruction,

- Order number
- Customer card details
- Host bank account details
- Payment type
- Amount
- Payment date

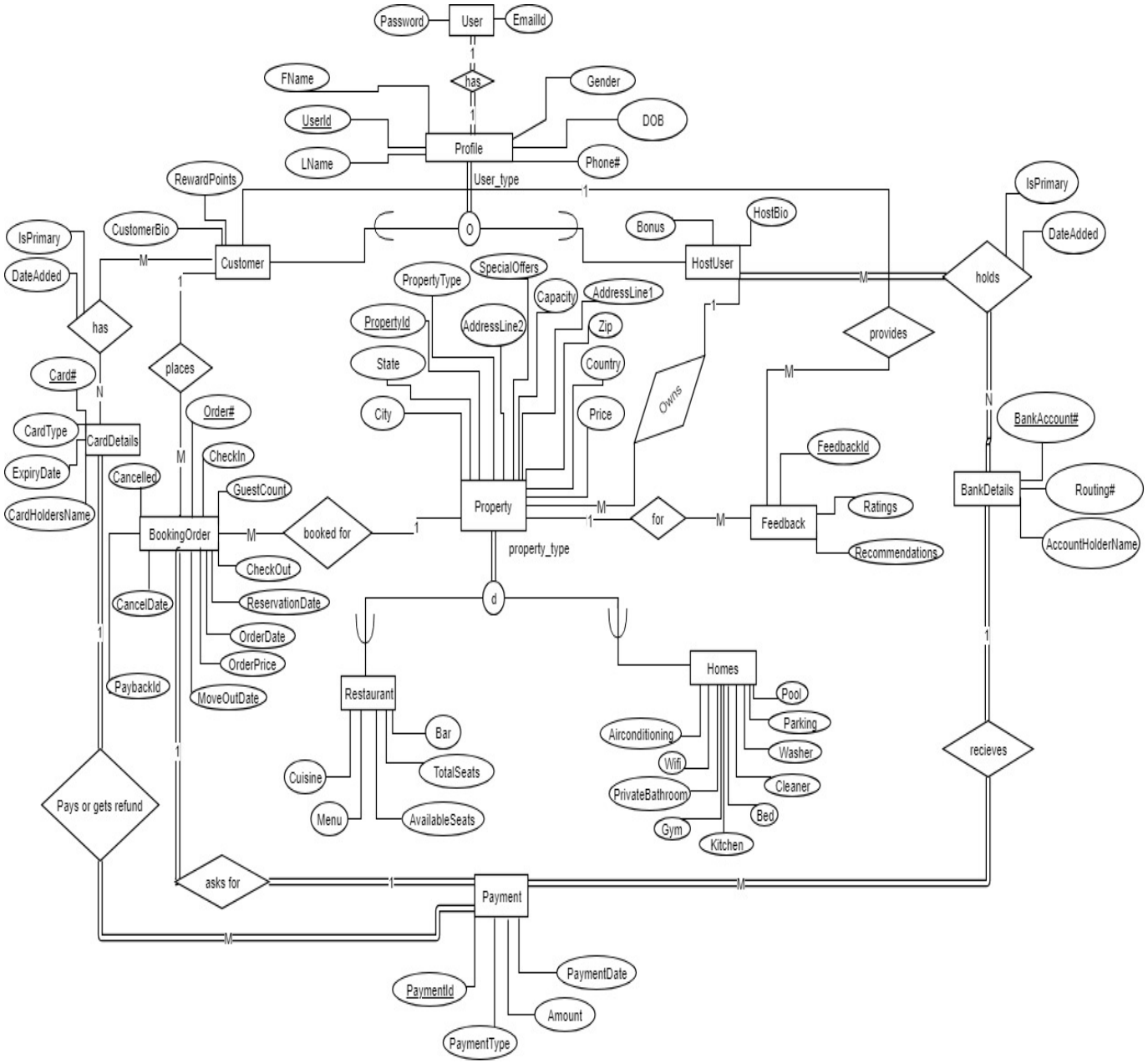
For each successful payment operation one payment id is generated. If a payment operation fails, a customer is asked to pay again and if not the payment made with in 10 minutes, booking order gets cancelled.

The amount customer pays, goes to the host bank account after deduction of standard AirBnb commission and payment to host is only made after the customer has used Airbnb services.

Feedback handling subsystem:

Customers can provide a feedback for the services they use. A customer can rate and recommend a property. Airbnb lists only those properties that have average rating of 2 or more out of 5. This constraint is relaxed in case a property has no feedback. For each feedback one unique feedback is generated.

Modelling the Data Requirements using ER DIAGRAM



Relationships Between Entities

a> 1 to 1 relations:
- 1> The relation between User and profile is 1:1.
- 2> The relation between payment and BookingOrder is 1:1.

b> 1 to many relations:
- 1> The relation between customer and BookingOrder is 1 to many.
- 2> The relation between customer and feedback is one to many.
- 3> The relation between host and property is 1 to many.
- 4> The relation between card and payment is 1 to many.
- 5> The relation between BankDetails and payment is one to many.
- 6> The relation between property and BookingOrder is one to many.
- 7> The relation between property and feedback is one to many.

c> Many to many relations:
- 1> The relation between Customer and Card is many to many.
- 2> The relation between host and BankDetails is many to many.

(Note: We are assuming, one user can have multiple cards/bank accounts and also one card/bank account can be used by multiple users).

Mapping ER DIAGRAM to RELATIONAL SCHEMA

1. AirbnbUser

Since user and profile are two entities in 1:1 relations and in full participation at both side, we merge these two table to create one new AirbnbUser use table.

UserID	Password	Fname	Lname	Gender	DOB	Phone#	EmailId

Primary key - UserID

2. Customer

This table is constructed using rule 8.a for table construction in generalization/specialization scenario. Customer is a specialized user profile.

UserID	CustomerBio	RewardPoints

Primary key – UserID

Foreign key – FOREIGN KEY (UserID) REFERENCES AirbnbUser (UserID)

3. HostUser

This table is constructed using rule 8.a for table construction in generalization/specialization scenario. Host is a specialized user profile.

UserID	Bonus	HostBio

Primary key – UserID

Foreign key – FOREIGN KEY (UserID) REFERENCES AirbnbUser (UserID)

4. CardDetails

This table is for the CardDetails entity.

Card#	CardType	CardHoldersName	ExpiryDate

Primary Key – Card#

5. CustomerCardDetails

This table is created to represent the many to many relation between CardDetails and Customer.

Card#	CustomerID	IsPrimary	DateAdded

Foreign key – FOREIGN KEY (CustomerID) REFERENCES Customer (UserID)

Foreign key - FOREIGN KEY (Card#) REFERENCES CardDetails (Card#)

Primary key – Card#, CustomerID

6. BankDetails

This table is for the BankDetails entity.

BankAccount#	Routing#	AccountHoldersName

Primary Key – BankAccount#

7. HostAccountDetails

This table is created to represent the many to many relation between BankDetails and Host.

BankAccount#	HostID	IsPrimary	DateAdded

Foreign key – FOREIGN KEY (HostID) REFERENCES HostUser (UserID)

Foreign key - FOREIGN KEY (BankAccount#) REFERENCES BankDetails (BankAccount#)

Primary key – BankAccount#, HostID

8. Property

This table is for the property entity.

PropertyID	Property Type	HostID	Capacity	Address Line1	Address Line2	City	State	Country	Zip	Price	Special offer

Foreign key – FOREIGN KEY (HostID) REFERENCES HostUser (UserID)

Primary Key - PropertyId

9. Homes

This table is constructed using rule 8.a for table construction in generalization/specialization scenario. The entity homes is a special type of property.

Property ID	Air Conditioning	WiFi	Private BathRoom	Bed	Kitchen	Gym	Pool	Parking	Washer	Cleaner

Primary Key – PropertyID

Foreign key – FOREIGN KEY (PropertyID) REFERENCES Property (PropertyID)

10. Restaurants

This table is constructed using rule 8.a for table construction in generalization/specialization scenario. The entity restaurants is a special type of property.

Property ID	Cuisine	Menu	Bar	TotalSeats	Available seats

Primary Key – PropertyID

Foreign key – FOREIGN KEY (PropertyID) REFERENCES Property (PropertyID)

11. FeedBack

This table is for the FeedBack entity.

FeedbackID	CustomerUserid	PropertyId	Ratings	Recommendation

Primary key – FeedbackId

Foreign key – FOREIGN KEY (CustomerUserid) REFERENCES Customer (UserID)

Foreign key – FOREIGN KEY (PropertyId) REFERENCES Property (PropertyID)

12. BookingOrder

This table is for the BookingOrder entity.

Order#	Property ID	Customer ID	Reservation Date	Guest Count	Order Date	Order Price	Check in	Check out	Move out date	Cancelled	CancelDate	Payback Id

Primary key – Order#

Foreign key – FOREIGN KEY (PropertyID) REFERENCES Property (PropertyID)

Foreign key – FOREIGN KEY (Customerid) REFERENCES Customer (UserID)

13. Payment

This table is for the Payment entity.

PaymentID	Order#	Card#	BankAccount#	Payment Type	Amount	PaymentDate

Primary Key – PaymentID

Foreign key - FOREIGN KEY (Order#) REFERENCES BookingOrder(Order#)

Foreign key - FOREIGN KEY (Card#) REFERENCES CardDetails (Card#)

Foreign key - FOREIGN KEY (BankAccount#) REFERENCES BankDetails (BankAccount#)

FUNCTIONAL DEPENDENCIES and NORMALIZATION Process

1> AirBnbUser: This table has two candidate keys UserId and EmailId, The following functional dependencies exist in this table. UserId is chosen as primary key.

UserId-> Password, Fname, Lname, Gender, DOB, Phone#, EmailId
EmailId -> Password, Fname, Lname, Gender, DOB, Phone#, UserId

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

2> Customer: UserID is the primary key for this table. The following functional dependencies exist.

UserID-> CustomerBio, RewardPoints

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

3> HostUser: UserID is the primary key for this table. The following functional dependencies exist.

UserID-> Bonus, HostBio

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

4> CardDetails: Card# is the primary key for this table. The following functional dependencies exist.

Card#-> CardType, CardHoldersName, ExpiryDate

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

5> CustomerCardDetails: The combination of Card# and CustomerID is the primary key for this table. The following functional dependencies exist.

Card#, CustomerID -> IsPrimary, DateAdded

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

6> BankDetails: BankAccount# is the primary key for this table. The following functional dependencies exist.

BankAccount#-> Routing#, AccountHoldersName

7> HostAccountDetails: The combination of BankAccount# and HostID is the primary key for this table. The following functional dependencies exist.

BankAccount#, HostID-> IsPrimary, DateAdded

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

8> Property: PropertyID is the primary key for this table. The following functional dependencies exist.

PropertyID-> PropertyType, HostID, Capacity, AddressLine1, AddressLine2, City, State, Country, Zip, Price, Special Offer

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

9> Homes: PropertyID is the primary key for this table. The following functional dependencies exist.

PropertyID-> AirConditioning, Wifi, PrivateBathRoom, Bed, Kitchen, Gym, Pool, Parking, Washer, Cleaner

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

10> Restaurants: PropertyID is the primary key for this table. The following functional dependencies exist.

PropertyID-> Cuisine, Menu, Bar, TotalSeats, Availableseats

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

11> FeedBack: FeedBackId is the primary key for this table. The following functional dependencies exist.

FeedBackId-> CustomerUserid, PropertyId, Ratings, Recommendation

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

12> BookingOrder: Order# is the primary key for this table. The following functional dependencies exist.

Order#-> PropertyId, CustomerId, ReservationDate, GuestCount, OrderDate, OrderPrice, CheckIn, CheckOut, MoveOutDate, Cancelled, CancelDate, PaybackId

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

13> Payment: PaymentId is the primary key for this table. The following functional dependencies exist.

PaymentId -> Order#, Card#, BankAccount#, PaymentType, Amount, PaymentDate

These Fds satisfy 1NF, 2NF and 3NF. So this table is already in 3NF.

SQL Queries for Table creation and addition of constraints

```
CREATE TABLE AirbnbUser (
  UserId VARCHAR(20) NOT NULL,
  Password VARCHAR(20) NOT NULL,
  FName VARCHAR(10) NOT NULL,
  LName VARCHAR(10) NOT NULL,
 Gender CHAR(1),
 DOB DATE,
Phone# VARCHAR(15),
EmailId VARCHAR(20) UNIQUE,
 PRIMARY KEY (UserId)
);

CREATE TABLE Customer (
  UserId VARCHAR(20) NOT NULL,
  CustomerBio  VARCHAR(200),
  RewardPoints INT,
PRIMARY KEY(UserId),
```

```
FOREIGN KEY(UserId) references Airbnbuser(UserID)
);

CREATE TABLE Hostuser(
  UserId VARCHAR(20) NOT NULL,
  HostBio  VARCHAR(200),
  Bonus INT,
PRIMARY KEY(UserId),
FOREIGN KEY(UserId) references Airbnbuser(UserID) on delete cascade
);

CREATE TABLE CardDetails (
  Card# VARCHAR(20) NOT NULL,
  CardType VARCHAR(10) NOT NULL,
  CardHoldersName VARCHAR(30) NOT NULL,
  ExpiryDate DATE NOT NULL,
  PRIMARY KEY(Card#)
);

CREATE TABLE CustomerCardDetails (
Card# VARCHAR(20) NOT NULL,
CustomerId VARCHAR(20) NOT NULL,
IsPrimary  CHAR,
DateAdded DATE NOT NULL,
PRIMARY KEY(Card#,CustomerId),
FOREIGN KEY(CustomerId) REFERENCES Customer(UserId) on delete cascade,
FOREIGN KEY(Card#) REFERENCES CardDetails(Card#) on delete cascade
);

CREATE TABLE BankDetails (
BankAccount# VARCHAR(20) NOT NULL,
```

```
Routing# VARCHAR(20) NOT NULL,
AccountHoldersname VARCHAR(20),
PRIMARY KEY(BankAccount#)
);

CREATE TABLE HostAccountDetails (
BankAccount# VARCHAR(20) NOT NULL,
HostID VARCHAR(20) NOT NULL,
AccountHoldersname VARCHAR(20),
IsPrimary  CHAR,
DateAdded DATE NOT NULL,
PRIMARY KEY(BankAccount#,HostID),
FOREIGN KEY(HostId) REFERENCES Hostuser(UserId) on delete cascade,
FOREIGN KEY(BankAccount#) REFERENCES BankDetails(BankAccount#) on delete
cascade
);

CREATE TABLE Property (
  PropertyId INT NOT NULL,
  PropertyType VARCHAR(15) NOT NULL,
  HostId VARCHAR(20) NOT NULL,
  Capacity INT NOT NULL,
  AddressLine1  VARCHAR(15),
  AddressLine2 VARCHAR(15),
  City VARCHAR(10),
  State VARCHAR(10),
  Country VARCHAR(10),
  Zip VARCHAR(5),
  Price INT,
  SpecialOffer VARCHAR(50),
 PRIMARY KEY(PropertyId),
 FOREIGN KEY(HostId) REFERENCES HostUser(UserId) on delete cascade
```

```
);

CREATE TABLE Homes (
PropertyId INT NOT NULL,
AirConditioning  VARCHAR(10),
WiFi VARCHAR(10),
PrivateBathroom VARCHAR(10),
Bed VARCHAR(10),
Kitchen VARCHAR(10),
Gym VARCHAR(10),
Pool VARCHAR(10),
Parking VARCHAR(10),
Washer VARCHAR(10),
Cleaner VARCHAR(10),
PRIMARY KEY (PropertyId),
FOREIGN KEY (PropertyID) REFERENCES Property (PropertyId)
on delete cascade
);

CREATE TABLE Restaurant (
PropertyId INT NOT NULL,
Cuisine VARCHAR(15),
Menu VARCHAR(15),
Bar VARCHAR(15),
TotalSeats INT,
AvailableSeats INT,
PRIMARY KEY(PropertyId),
FOREIGN KEY(PropertyId) REFERENCES Property(PropertyId)
on delete cascade
);
```

```
CREATE TABLE Feedback (
 FeedbackId INT NOT NULL,
 CustomerUserId VARCHAR(20),
 PropertyId INT NOT NULL,
 Ratings INT  NOT NULL,
 Recommendation VARCHAR(20),
 PRIMARY KEY (FeedbackId),
 FOREIGN KEY (CustomerUserId) REFERENCES Customer (UserId) on delete cascade,
 FOREIGN KEY (PropertyId) REFERENCES Property(PropertyId)
 on delete cascade
 );

CREATE TABLE BookingOrder (
  Order# INT NOT NULL,
  PropertyId INT NOT NULL,
  CustomerId VARCHAR(20) NOT NULL,
  ReservationDate DATE,
  GuestCount INT NOT NULL,
  OrderDate CHAR(10),
  OrderPrice INT,
  CheckIn CHAR(10),
  CheckOut CHAR(10),
  MoveOutDate DATE,
  Cancelled CHAR,
  CancelDate Date,
  PaybackId VARCHAR(20),
  PRIMARY KEY(Order#),
  FOREIGN KEY(PropertyId) REFERENCES Property(PropertyId)
  on delete cascade,
  FOREIGN KEY(CustomerId) REFERENCES Customer(UserId) on delete cascade
);
```

```
CREATE TABLE Payment (
  PaymentId INT NOT NULL,
  Order# INT NOT NULL,
  Card# VARCHAR(20) NOT NULL,
  BankAccount# VARCHAR(20),
  PaymentType VARCHAR(10),
  Amount INT,
  PaymentDate DATE,
  PRIMARY KEY(PaymentId),
  FOREIGN KEY(Order#) REFERENCES BookingOrder(Order#) on delete cascade,
  FOREIGN KEY(Card#) REFERENCES CardDetails(Card#) on delete cascade,
  FOREIGN KEY(BankAccount#) REFERENCES BankDetails(BankAccount#) on delete
cascade
);
```

PL/SQL – Triggers

Trigger 1- Increase property booking price when the available capacity becomes < 4

This trigger doubles the booking price whenever a booking order reduces the available seats of a property to less than 4.

```
CREATE OR REPLACE TRIGGER INCREASE_PROPERTY_BOOKING_PRICE
AFTER INSERT ON BookingOrder
FOR EACH ROW
ENABLE
BEGIN
UPDATE Property
SET Capacity = Capacity-:new.GuestCount
WHERE PropertyId = :new.PropertyId;
UPDATE Property
SET Price = Price*2
WHERE PropertyId = :new.PropertyId
AND Capacity < 4;
END;
```

Output:

Before execution,

SELECT * FROM Property;

PROPERTYID	PROPERTYTYPE	HOSTID	CAPACITY	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	COUNTRY	ZIP	PRICE	SPECIALOFFER
1	house	1	10	abc	xyz	a	b	c	123	100	qwerty
2	house	1	10	abc	xyz	a	b	c	123	100	qwerty
3	house	2	10	abc	xyz	a	b	c	123	100	qwerty
4	house	2	10	abc	xyz	a	b	c	123	100	qwerty
5	house	3	10	abc	xyz	a	b	c	123	100	qwerty
6	house	3	10	abc	xyz	a	b	c	123	100	qwerty

After execution,

INSERT INTO bookingOrder VALUES(9,3,'5',TRUNC(SYSDATE), 7, '2019/30/04', 2500, '13:00', '10:00', TRUNC(SYSDATE+2),'N','','');

INSERT INTO bookingOrder VALUES(10,2,'5',TRUNC(SYSDATE), 5, '2019/30/04', 2500, '13:00', '10:00', TRUNC(SYSDATE+2),'N','','');

SELECT * FROM Property;

PROPERTYID	PROPERTYTYPE	HOSTID	CAPACITY	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	COUNTRY	ZIP	PRICE	SPECIALOFFER
1	house	1	10	abc	xyz	a	b	c	123	100	qwerty
2	house	1	5	abc	xyz	a	b	c	123	100	qwerty
3	house	2	3	abc	xyz	a	b	c	123	200	qwerty
4	house	2	10	abc	xyz	a	b	c	123	100	qwerty
5	house	3	10	abc	xyz	a	b	c	123	100	qwerty
6	house	3	10	abc	xyz	a	b	c	123	100	qwerty

Since the available capacity of propertyID 3 is 3, the price has doubled. But for propertyID 2, available capacity is 5 and price for this did not change.

Trigger 2- Award bonus to hosts when they get good feedback

After receiving 10 good ratings (4 or more), whenever a host receives good ratings(4 or more), that host is awarded 5 bonus point for each such rating.

```
CREATE OR REPLACE TRIGGER AWARD_BONUS_TO_HOST
BEFORE INSERT ON Feedback
FOR EACH ROW
ENABLE
DECLARE
feedback_count INT;
temp_count INT;
property_host_id HostUser.UserId%TYPE;
property_entry Property%ROWTYPE;
CURSOR property_details IS
SELECT * FROM Property
WHERE HostId IN
(SELECT HostID FROM Property WHERE PropertyId = :new.PropertyID);
BEGIN
IF (:new.Ratings > 3) THEN
   feedback_count := 0;
   OPEN property_details;
   LOOP
      FETCH property_details INTO property_entry;
      EXIT WHEN (property_details%NOTFOUND);
      property_host_id := property_entry.HostId;
      SELECT COUNT(*) INTO temp_count FROM Feedback Where PropertyId =
property_entry.PropertyId;
      feedback_count := feedback_count + temp_count;
   END LOOP;
   IF(feedback_count > 9) THEN
      UPDATE Hostuser SET Bonus = Bonus+5 Where UserId = property_host_id;
   END IF;
   CLOSE property_details;
END IF;
END;
```

Output: Before execution,
Select * from HostUser;

USERID	HOSTBIO	BONUS
1	abc	0
2	abc	0
3	abc	0
4	abc	0

Select * from Feedback;

FEEDBACKID	CUSTOMERUSERID	PROPERTYID	RATINGS	RECOMMENDATION
1	5	1	4	abc
16	5	1	4	abc
6	5	1	4	abc
7	5	1	4	abc
8	5	1	4	abc
9	5	1	4	abc
10	5	1	4	abc
11	5	1	4	abc
12	5	1	4	abc
15	5	1	4	abc

Insert 11th good review.
insert into feedback values(100,'5',1,4,'abc');
Select * from HostUser;

USERID	HOSTBIO	BONUS
1	abc	5
2	abc	0
3	abc	0
4	abc	0

The bonus has been added to userID 1 in HostUser table.

Trigger 3- Give 10% discount when customer books a house for more than 7 days

This trigger gives 10% discount to a customer for a booking order made for more than 7 day.

```
CREATE OR REPLACE TRIGGER DISCOUNT_PRICE
Before INSERT OR UPDATE ON BookingOrder
FOR EACH ROW
ENABLE
BEGIN
IF ((:new.MoveOutDate - :new.ReservationDate) >7) THEN
   :new.OrderPrice := :new.OrderPrice *90/100;
END IF;
END;
```

Output:

Before execution,

SELECT * FROM bookingorder;

ORDER#	PROPERTYID	CUSTOMERID	RESERVATIONDATE	GUESTCOUNT	ORDERDATE	ORDERPRICE	CHECKIN	CHECKOUT	MOVEOUTDATE	CANCELLED	CANCELDATE	PAYBACKID
1	2	5	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-
2	2	5	30-APR-19	2	2019/30/04	2501	13:00	10:00	30-APR-19	N	-	-
3	5	6	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-
4	5	6	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-

After execution,

insert into bookingOrder values(7,3,'5',TRUNC(SYSDATE), 2, '2019/30/04', 2500, '13:00', '10:00', TRUNC(SYSDATE+8),'N','','');

insert into bookingOrder values(8,3,'5',TRUNC(SYSDATE), 2, '2019/30/04', 2500, '13:00', '10:00', TRUNC(SYSDATE+7),'N','','');

SELECT * FROM bookingorder;

ORDER#	PROPERTYID	CUSTOMERID	RESERVATIONDATE	GUESTCOUNT	ORDERDATE	ORDERPRICE	CHECKIN	CHECKOUT	MOVEOUTDATE	CANCELLED	CANCELDATE	PAYBACKID
1	2	5	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-
2	2	5	30-APR-19	2	2019/30/04	2501	13:00	10:00	30-APR-19	N	-	-
3	5	6	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-
4	5	6	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-
7	3	5	01-MAY-19	2	2019/30/04	2250	13:00	10:00	09-MAY-19	N	-	-
8	3	5	01-MAY-19	2	2019/30/04	2500	13:00	10:00	08-MAY-19	N	-	-

Order#7 has discounted order price 2250, but the original order price was 2500, but for ORDER#8 has the full order price 2500.

PL/SQL – Procedures

Procedure 1- Update Restaurant Seats

Given a property id and number of seats to book, this procedure updates the number of available seats. If number of seats to book is more than the number of available seats, a message is generated informing the user of it.

```
CREATE OR REPLACE PROCEDURE
update_restaurant_available_seat_count(restaurantId INT, seatsRequested INT) AS
CURSOR restaurants IS
SELECT PropertyId, AvailableSeats from Restaurant  WHERE PropertyId=restaurantId;
restaurant_entry restaurants%ROWTYPE;
BEGIN
   open restaurants;
   LOOP
      FETCH restaurants into restaurant_entry;
      IF restaurants%NOTFOUND THEN
         EXIT;
      END IF;
      IF (restaurant_entry.AvailableSeats<seatsRequested) THEN
         DBMS_OUTPUT.put_line('Not enough seats available to complete this
transaction.');
      Else
         UPDATE Restaurant set AvailableSeats = AvailableSeats-seatsRequested
            WHERE PropertyId = restaurantId;
      END IF;
   END LOOP;
   CLOSE restaurants;
END;
```

Arguments: restaurantId INT, seatsRequested INT

Output:
Before execution,

SELECT * FROM Restaurant ;

PROPERTYID	CUISINE	MENU	BAR	TOTALSEATS	AVAILABLESEATS
2	abc	xyz	Y	10	10

After execution:
EXECUTE update_restaurant_available_seat_count(2,8);

PROPERTYID	CUISINE	MENU	BAR	TOTALSEATS	AVAILABLESEATS
2	abc	xyz	Y	10	2

EXECUTE update_restaurant_available_seat_count(2,8);

```
Statement processed.
Not enough seats available to complete this transaction.
```

Procedure 2 - Remove Properties With Bad Ratings

This procedure deletes properties with rating less than 2.

```
CREATE OR REPLACE PROCEDURE remove_bad_property AS
CURSOR badFeedbackProperty IS
SELECT PropertyId , avg(Ratings) as AverageRating from Feedback group by
PropertyId;
badProperty badFeedbackProperty%ROWTYPE;
BEGIN
   open badFeedbackProperty;
   LOOP
      FETCH badFeedbackProperty into badProperty;
      IF badFeedbackProperty%NOTFOUND THEN
         EXIT;
      END IF;
      IF (badProperty.AverageRating<2) THEN
         DELETE FROM Property where PropertyId= badProperty.PropertyId;
      END IF;
   END LOOP;
   CLOSE badFeedbackProperty;
END;
```

Arguments: None

Output:

Before executing the procedure,

SELECT * FROM Property;

PROPERTYID	PROPERTYTYPE	HOSTID	CAPACITY	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	COUNTRY	ZIP	PRICE	SPECIALOFFER
1	house	1	10	abc	xyz	a	b	c	123	100	qwerty
2	house	1	10	abc	xyz	a	b	c	123	100	qwerty
3	house	2	10	abc	xyz	a	b	c	123	100	qwerty
4	house	2	10	abc	xyz	a	b	c	123	100	qwerty
5	house	3	10	abc	xyz	a	b	c	123	100	qwerty
6	house	3	10	abc	xyz	a	b	c	123	100	qwerty

SELECT PropertyId , avg(Ratings) as AverageRating from Feedback group by PropertyId;

PROPERTYID	AVERAGERATING
6	5
1	1
2	2
4	1
5	4
3	3

After the execution of the procedure,

SELECT * FROM Property;

PROPERTYID	PROPERTYTYPE	HOSTID	CAPACITY	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	COUNTRY	ZIP	PRICE	SPECIALOFFER
2	house	1	10	abc	xyz	a	b	c	123	100	qwerty
3	house	2	10	abc	xyz	a	b	c	123	100	qwerty
5	house	3	10	abc	xyz	a	b	c	123	100	qwerty
6	house	3	10	abc	xyz	a	b	c	123	100	qwerty

Procedure 3 – Reward Points Update

This procedure provides reward points of 250 to the guests who have bookings worth more than 5000$.

```
CREATE OR REPLACE PROCEDURE update_reward_points AS

CURSOR heavyBuyers IS

SELECT CustomerId , SUM(OrderPrice) as TotalOrder from BookingOrder group by CustomerId;

heavyBuyer heavyBuyers%ROWTYPE;

BEGIN

   open heavyBuyers;

   LOOP

      FETCH heavyBuyers into heavyBuyer;

      IF heavyBuyers%NOTFOUND THEN

         EXIT;

      END IF;

      IF (heavyBuyer.TotalOrder>5000) THEN

         UPDATE Customer SET RewardPoints=RewardPoints+250 WHERE USERID=heavyBuyer.CustomerId;

      END IF;

   END LOOP;

   CLOSE heavyBuyers;

END;
```

Arguments: None

Output:

Before execution,

SELECT * FROM BookingOrder;

ORDER#	PROPERTYID	CUSTOMERID	RESERVATIONDATE	GUESTCOUNT	ORDERDATE	ORDERPRICE	CHECKIN	CHECKOUT	MOVEOUTDATE	CANCELLED	CANCELDATE	PAYBACKID
1	2	5	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-
2	2	5	30-APR-19	2	2019/30/04	2501	13:00	10:00	30-APR-19	N	-	-
3	5	6	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-
4	5	6	30-APR-19	2	2019/30/04	2500	13:00	10:00	30-APR-19	N	-	-

SELECT * FROM Customer;

USERID	CUSTOMERBIO	REWARDPOINTS
5	abc	0
6	abc	0
7	abc	0
8	abc	0

After execution,

USERID	CUSTOMERBIO	REWARDPOINTS
5	abc	250
6	abc	0
7	abc	0
8	abc	0

Reference

https://en.wikipedia.org/wiki/Airbnb