Ari Butterfield - ASB180007
James Boyer - JSB170130
Dr. Nurcan Yuruk and Dr. Timothy Farage
CS 6360

Airbnb Database Design

Defining Entities and Relationship

Overview

Airbnb is an online platform for connecting hosts to users searching for a rental property. Properties hold an array of information for consumers to view, and all users store a variety of personal and financial information for transactional purposes. Users can become hosts by listing a property for rent. Users can rent and save properties, and hosts can have properties, tax forms, and payout methods, in addition to the general user information.

Entities/Components

- User - Stores information for anyone with a registered account on Airbnb. Includes all the account information and preferences.
    - Account Info
        - Legal Name
        - Gender
        - DOB
        - Email
        - Phone Number
        - Government ID (Key)
        - Address
        - Emergency Contact
        - Password
        - Default Language
        - Translation On?
        - Default Currency
        - Timezone
        - Number of Referrals (Derived)
    - Notification Preferences
        - Inspiration and offers
            - Email (On / Off)
            - SMS (On / Off)
        - Trip planning
            - Email (On / Off)
            - SMS (On / Off)
        - News and programs
            - Email (On / Off)
            - SMS (On / Off)
        - Feedback
            - Email (On / Off)
            - SMS (On / Off)
        - Travel regulations
            - Email (On / Off)
            - SMS (On / Off)
        - Account activity
            - Email (On / Off)
            - SMS (On / Off)
        - Guest policies
            - Email (On / Off)
            - SMS (On / Off)
        - Reminders
            - Email (On / Off)
            - SMS (On / Off)

        - Messages
            - Email (On / Off)
            - SMS (On / Off)
    - Privacy and Sharing Information
        - Include Profile on Search Engines (On / Off)
        - Use First Name and Photo (On / Off)
- Payout Method - Stores financial information for the Airbnb hosts to receive payouts
    - Account Number (Key)
    - Routing Number (Key)
    - Account Holder Name
    - Checking or Savings
- Payment Method - Stores financial information for the Airbnb users to rent properties
    - Card Number (Key)
    - CVV (Key)
    - Expiration Date
    - Country/Region
    - Zip Code
- Tax From - Stores host tax information
    - Country
    - City
    - State
    - Zip Code
    - Address
    - Full Name
    - US Tax ID Number (Key)
    - Business Name (Key)
    - Tax Classification
- Host (Subclass of User) - Stores information for users with the ability to rent out properties
    - # Reviews
    - Identity verified (Y or N)
    - Superhost (Y or N)
    - Response Rate
    - Allows Contributions
    - Simplified or Split-fee Pricing
- Financial Receipt - Stores information of past renting transactions
    - Service Fee
    - Taxes
    - Receipt ID
- Property - Stores all relevant information regarding the property
    - Location
        - Zip Code
        - Street
        - City
        - State / Province / Region

            - Country
    - Property Info
        - Property Type
        - Max. Guest #
        - # Bedrooms
        - # Beds
        - # Bathrooms
        - Pictures
        - Amenities (multi-valued)
    - Rent Pricing
        - Cost per night
        - Cleaning fee
        - Service fee
    - Additional Info
        - Getting there (multi-valued)
        - Price & availability (multi-valued)
        - House rules (multi-valued)
        - Health & safety (multi-valued)
        - Cancellation policy (multi-valued)
    - Open Dates
        - Time Period (multi-valued)
            - Start Date
                - Day
                - Month
                - Year
            - End Date
                - Day
                - Month
                - Year

Relationships between Entities

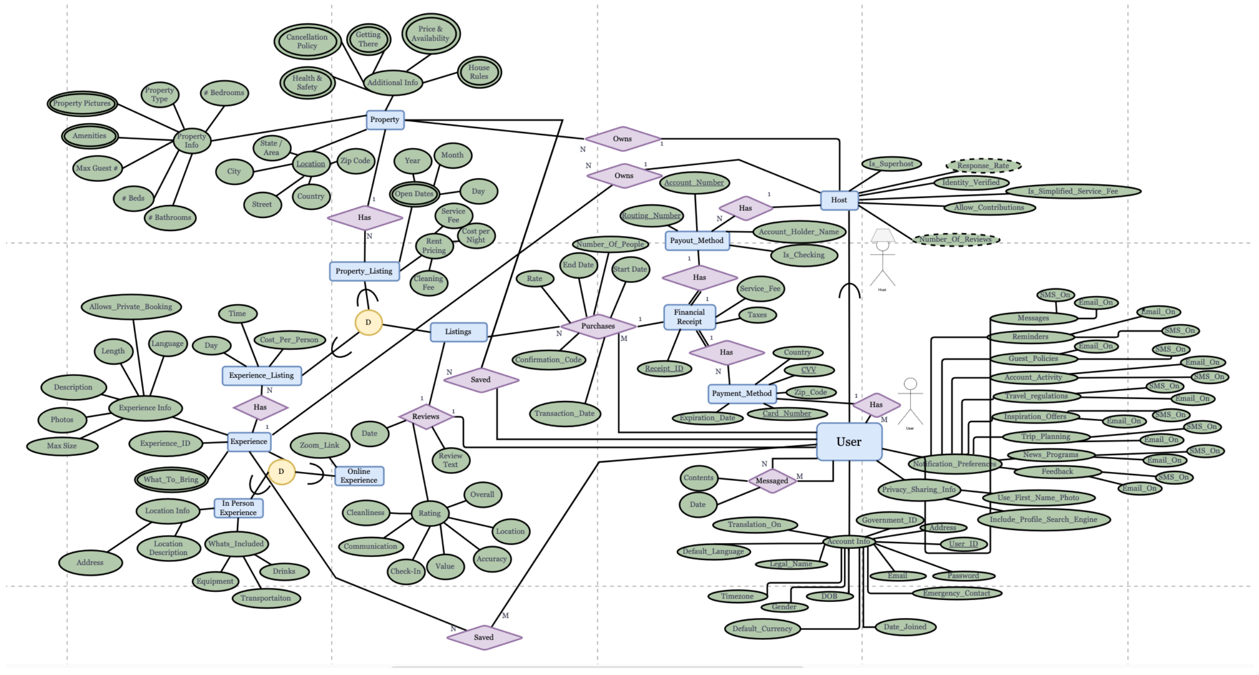
- User **Rents** Property recorded by Financial Receipt
    - Start Date
    - End Date
    - Rate
    - Transaction
    - Ratings / Reviews
- User **Saves** Property (Many to Many)
- User **Has** Payment Method (One to Many)
- User **Messages** User (Many to Many)
    - Contents
    - Date of Message

- Host **Owns** Property (One to Many)
- Host **Has** Tax Form (One to One)
- Host **Has** Payout Method (One to Many)
- Financial Receipt **Has** Payout Method (One to One)
- Financial Receipt **Has** Payment Method (One to Many)

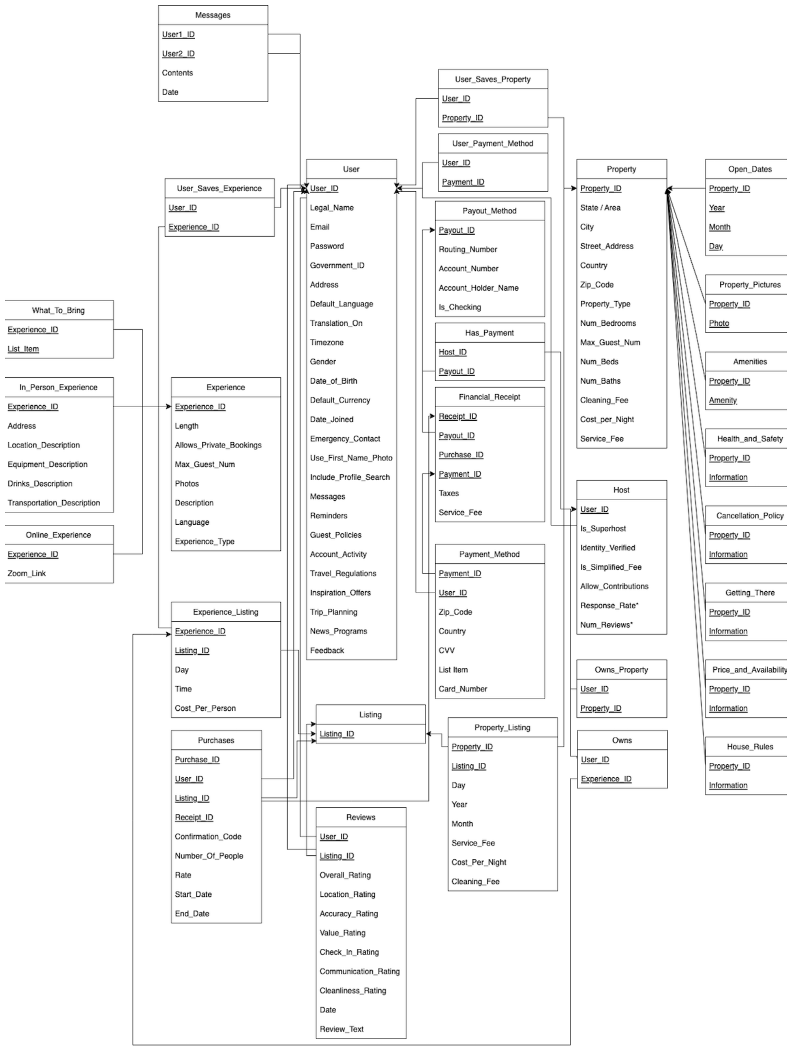
Enhanced ER Elements

- Host **Is Subclass** of User

Designing ER Diagram



Designing Relational Model



Updating Model to 3NF

Discuss database normalization rules on your tables. Show the functional dependencies that violate 1st, 2nd and 3rd normal forms. Normalize your table(s) into 3NF.

1NF

In creating our relational schema, we already took the effort to remove any multi-valued attributes. The key in every relation determines every attribute implying that our relational schema is in at least 1NF.

2NF

We noticed a few violations of 2NF:

- We observed that for our Purchases relation that the Receipt_ID was unnecessary, and listed as a key attribute. This violated 2NF as non-prime attributes were not dependent on Receipt_ID, and therefore were not dependent on the whole key. We removed it and created a dependency from the Purchases_ID attribute in Receipt to the Purchases relation.
- We also realized that listing Payout_ID, Purchase_ID, and Payment_ID as key attributes also violates 2NF in Financial Receipt, as neither Taxes nor Service_Fee are both dependent on any of them.
- We also realized that Property_ID should not be a key for Property_Listing, as none of the other attributes are derived from Property_ID. Listing_ID is the only necessary key attribute.

Additional edits:

- We also removed the Listing relation as it was redundant and unnecessary, as Property_Listing already holds a unique Listing_ID.
- We removed User_ID as a key for Payment_Method since User_ID does not uniquely determine the attributes of the Payment_Method relation.

Otherwise, all other attributes are dependent on the whole key of the relation for all other relations. With the aforementioned changes, the relational schema is in 3NF.

3NF

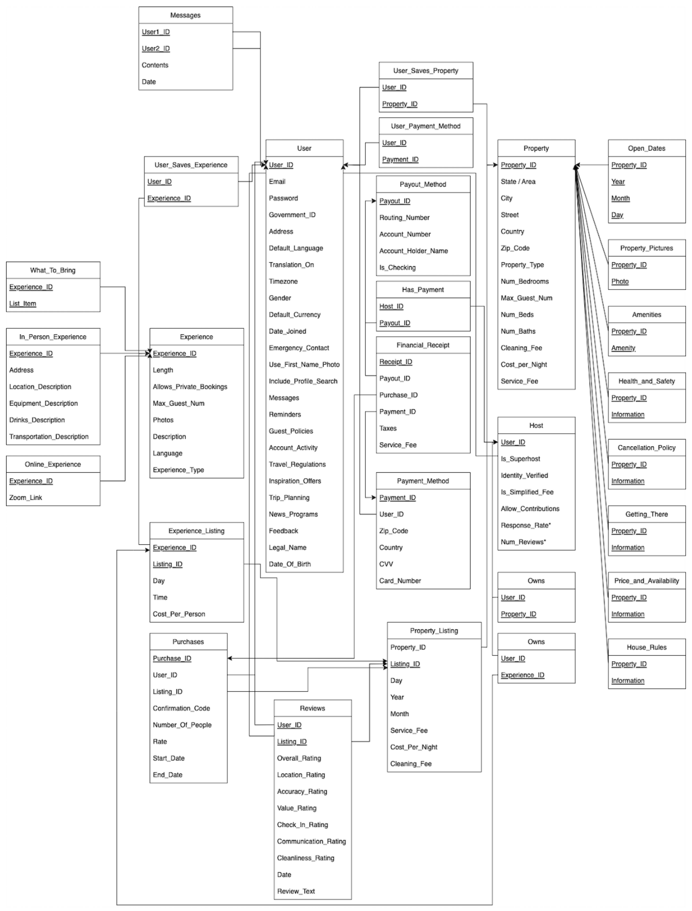
There were two cases we thought might violate 3NF but we found both were consistent with 3NF in the end.

First, we observed that the Government_ID attribute uniquely determines the Legal_Name and Date_Of_Birth for the User relation. However, Government_ID is a candidate key for the User table as it uniquely determines all other values of the User relation. Hence, Legal_Name and Date_Of_Birth are not transitively dependent on the key through a non-prime attribute.

Second, we observed for property that City, Street_Address, and Zip_Code can uniquely identify a State / Area and the Country. However, we also realized that City, Street_Address, and Zip_Code uniquely identify a property, so they form a candidate key. As these attributes are prime, dependencies on these attributes are allowed in 3NF. On a global scale, city might not uniquely identify a state, and city and state might not uniquely identify a country, therefore there is no point in taking a risk by assuming any such dependencies. Therefore, the Property relation is in 3NF and no further action is necessary.

After analyzing these two cases, we determined that both potential violations were dependencies on prime attributes. However, no relations possess transitive dependencies on non-prime attributes. Therefore, the relational schema is in 3NF.

3NF Normalized Relational Model



SQL Database Creation

```
CREATE TABLE User(
     User_ID INT,
     Email VARCHAR(30) NOT NULL,
     Password VARCHAR(30) NOT NULL,
     Government_ID INT,
     Address VARCHAR(50) NOT NULL,
     Default_Language(15) DEFAULT ‘English’,
     Translation_On BOOL DEFAULT False,
     Timezone VARCHAR(3),
     Gender VARCHAR(10),
     Default_Currency VARCHAR(10),
     Date_Joined DATE,
     Emergency_Contact VARCHAR(30),
     Use_First_Name_Photo BOOL DEFAULT False,
     Include_Profile_Search BOOL DEFAULT False,
     Messages BOOL DEFAULT True,
     Reminders BOOL DEFAULT True,
     Guest_Policies BOOL DEFAULT True,
     Account_Activity BOOL DEFAULT True,
     Travel_Regulations BOOL DEFAULT True,
     Inspiration_Offers BOOL DEFAULT True,
     Trip_Planning BOOL DEFAULT True,
     News_Programs BOOL DEFAULT True,
     Feedback BOOL DEFAULT True,
     PRIMARY KEY (User_ID)
);

CREATE TABLE Property (
     Property_ID INT,
     State_or_Area VARCHAR(20) NOT NULL,
     City VARCHAR(20) NOT NULL,
     Street_Address VARCHAR(50) NOT NULL,
     Country VARCHAR(20) NOT NULL,
     Property_Type VARCHAR(20) NOT NULL,
     Num_Bedrooms INT NOT NULL,
     Max_Guest_Num INT NOT NULL,
     Num_Beds INT NOT NULL,
     Num_Baths INT NOT NULL,
     Cleaning_Fee DECIMAL(10,2) NOT NULL,
     Cost_per_Night DECIMAL(10,2) NOT NULL,
     Service_Fee DECIMAL(10,2) NOT NULL,
     PRIMARY KEY (Property_ID)
);

CREATE TABLE Payout_Method (
     Payout_ID INT,
     Routing_Number INT NOT NULL,
     Account_Number INT NOT NULL,
     Account_Holder_Name VARCHAR(50) NOT NULL,
     Is_Checking BOOL NOT NULL,
     PRIMARY KEY (Payout_ID)
);

CREATE TABLE Experience (
     Experience_ID INT,
     Length VARCHAR(10) NOT NULL,
     Allows_Private_Bookings BOOL DEFAULT True,
     Max_Guest_Num INT NOT NULL,
     Photos BLOB,
     Description VARCHAR(1000) NOT NULL,
     Language VARCHAR(20) NOT NULL,
     Experience_Type VARCHAR(20) NOT NULL,
     PRIMARY KEY (Experience_ID)
);

CREATE TABLE Owns_Property (
     User_ID INT FOREIGN KEY REFERENCES User(User_ID),
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     PRIMARY KEY (User_ID, Property_ID)
     ON DELETE CASCADE
);

CREATE TABLE Owns_Experience (
     User_ID INT FOREIGN KEY REFERENCES User(User_ID),
     Experience_ID INT FOREIGN KEY REFERENCES
Experience(Experience_ID)
     ON DELETE CASCADE,
     PRIMARY KEY (User_ID, Experience_ID)
     ON DELETE CASCADE
);

CREATE TABLE Host (
     User_ID INT FOREIGN KEY REFERENCES User(User_ID)
     ON DELETE CASCADE,
     Is_Superhost BOOL DEFAULT False,
     Identity_Verified BOOL DEFAULT False,
     Is_Simplified_Fee BOOL DEFAULT True,
     Allow_Contributions Bool DEFAULT True,
     Response_Rate DECIMAL(2,2),
     Num_Reviews INT DEFAULT 0,
     PRIMARY KEY (User_ID)
);

CREATE TABLE Open_Dates (
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     Day DATE NOT NULL,
     PRIMARY_KEY (Property_ID, Day)
);

CREATE TABLE Property_Pictures (
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     Photo BLOB NOT NULL,
     PRIMARY_KEY (Property_ID, Photo)
);

CREATE TABLE Amenities (
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     Amenity VARCHAR(50) NOT NULL,
     PRIMARY_KEY (Property_ID, Amenity)
);

CREATE TABLE Health_and_Safety (
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     Information VARCHAR(500) NOT NULL,
     PRIMARY_KEY (Property_ID, Information)
);

CREATE TABLE Cancellation_Policy (
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     Information VARCHAR(500) NOT NULL,
     PRIMARY_KEY (Property_ID, Information)
);

CREATE TABLE Price_and_Availability (
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     Information VARCHAR(500) NOT NULL,
     PRIMARY_KEY (Property_ID, Information)
);

CREATE TABLE House_Rules (
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     Information VARCHAR(500) NOT NULL,
     PRIMARY_KEY (Property_ID, Information)
);

CREATE TABLE Property_Listing (
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     Listing_ID INT,
     Date DATE NOT NULL,
     Service_Fee DECIMAL(10,2) NOT NULL,
     Cost_Per_Night DECIMAL(10,2) NOT NULL,
     Cleaning_Fee DECIMAL(10,2) NOT NULL,
     PRIMARY KEY (Listing_ID)
);

CREATE TABLE Payment_Method (
     Payment_ID INT,
     User_ID INT FOREIGN KEY REFERENCES User(User_ID),
     ON DELETE CASCADE,
     Zip_Code VARCHAR(50) NOT NULL,
     Country VARCHAR(100) NOT NULL,
     CVV VARCHAR(10) NOT NULL,
     Card_Number INT(20) NOT NULL,
     PRIMARY KEY (Payment_ID)
);

CREATE TABLE Has_Payment (
     Host_ID INT FOREIGN KEY REFERENCES Host(User_ID)
     ON DELETE CASCADE,
     Payout_ID INT FOREIGN KEY REFERENCES
Payout_Method(Payout_ID)
     ON DELETE CASCADE,
     PRIMARY KEY (Host_ID, Payout_ID)
);

CREATE TABLE User_Payment_Method (
     User_ID INT FOREIGN KEY REFERENCES User(User_ID)
     ON DELETE CASCADE,
     Payment_ID INT FOREIGN KEY REFERENCES
Payment_Method(Payment_ID)
     ON DELETE CASCADE,
     PRIMARY KEY (User_ID, Payment_ID)
);

CREATE TABLE User_Saves_Property (
     User_ID INT FOREIGN KEY REFERENCES User(User_ID)
     ON DELETE CASCADE,
     Property_ID INT FOREIGN KEY REFERENCES
Property(Property_ID)
     ON DELETE CASCADE,
     PRIMARY KEY (User_ID, Property_ID)
);

CREATE TABLE Messages (
     User1_ID INT FOREIGN KEY REFERENCES User(User_ID)
     ON DELETE SET NULL,
     User2_ID INT FOREIGN KEY REFERENCES User(User_ID)
     ON DELETE SET NULL,
     Contents VARCHAR(1000) NOT NULL,
     Date DATE,
     PRIMARY KEY (User1_ID, User2_ID)
);
```

```
CREATE TABLE User_Saves_Experience (
     User_ID INT FOREIGN KEY REFERENCES User(User_ID)
     ON DELETE CASCADE,
     Experience_ID INT FOREIGN KEY REFERENCES
Experience(Experience_ID)
     ON DELETE CASCADE,
     PRIMARY KEY (User_ID, Experience_ID)
);

CREATE TABLE What_To_Bring (
     Experience_ID INT FOREIGN KEY REFERENCES
Experience(Experience_ID)
     ON DELETE CASCADE,
     List_Item VARCHAR(3000),
     PRIMARY KEY (Experience_ID, List_Item)
);

CREATE TABLE In_Person_Experience (
     Experience_ID INT FOREIGN KEY REFERENCES
Experience(Experience_ID)
     ON DELETE CASCADE,
     Address VARCHAR(50) NOT NULL,
     Location_Description VARCHAR(1000),
     Equipment_Description VARCHAR(1000),
     Drinks_Description VARCHAR(1000),
     Transportation_Description VARCHAR(1000),
     PRIMARY KEY (Experience_ID)
);

CREATE TABLE Online_Experience (
     Experience_ID INT FOREIGN KEY REFERENCES
Experience(Experience_ID)
     ON DELETE CASCADE,
     Zoom_Link VARCHAR(100) NOT NULL,
     PRIMARY KEY (Experience_ID)
);

CREATE TABLE Experience_Listing (
     Experience_ID INT FOREIGN KEY REFERENCES
Experience(Experience_ID)
```

```
     ON DELETE CASCADE,
     Listing_ID INT FOREIGN KEY REFERENCES
Property_Listing(Listing_ID)
     ON DELETE CASCADE,
     Day DATE,
     Time TIME,
     Cost_Per_Person DECIMAL(10,2),
     PRIMARY KEY (Experience_ID, Listing_ID)
);

CREATE TABLE Purchases (
     Purchase_ID INT,
     User_ID INT FOREIGN KEY REFERENCES User(User_ID)
     ON DELETE SET NULL,
     Listing_ID INT FOREIGN KEY REFERENCES
Property_Listing(Listing_ID)
     ON DELETE SET NULL,
     Confirmation_Code INT,
     Number_Of_People INT,
     Rate DECIMAL(10,2),
     Start_Date DATE,
     End_Date DATE,
     PRIMARY KEY(Purchase_ID)
);

CREATE TABLE Reviews (
     User_ID INT FOREIGN KEY REFERENCES User(User_ID)
     ON DELETE CASCADE,
     Listing_ID INT FOREIGN KEY REFERENCES
Property_Listing(Listing_ID)
     ON DELETE CASCADE,
     Overall_Rating DECIMAL(3,2),
     Location_Rating DECIMAL(3,2),
     Accuracy_Rating DECIMAL(3,2),
     Value_Rating DECIMAL(3,2),
     Check_In_Rating DECIMAL(3,2),
     Communication_Rating DECIMAL(3,2),
     Cleanliness_Rating DECIMAL(3,2),
     Date DATE,
     Review_Text VARCHAR(1000),
     PRIMARY KEY(User_ID,Listing_ID)
```

```
);

CREATE TABLE Financial_Receipt (
     Receipt_ID INT,
     Payout_ID INT FOREIGN KEY REFERENCES
Payout_Method(Payout_ID)
     ON DELETE SET NULL,
     Purchase_ID INT FOREIGN KEY REFERENCES
Purchases(Purchase_ID)
     ON DELETE SET NULL,
     Payment_ID INT FOREIGN KEY REFERENCES
Payment_Method(Payment_ID)
     ON DELETE SET NULL,
     Taxes DECIMAL(10,2) NOT NULL,
     Service_Fee DECIMAL(10,2) NOT NULL,
     PRIMARY KEY (Receipt_ID)
);
```

PL/SQL Triggers and Procedures

PL/SQL: Define two relevant stored procedures and two triggers (they should have a meaningful application in real-world cases).

Procedures

1) Inserts a new purchase entry into the Purchases table.

```
CREATE OR REPLACE PROCEDURE New_Purchase
     (User_ID IN INT,
      Listing_ID IN INT,
      Number_Of_People IN INT,
      Start_Date IN DATE,
      End_Date IN DATE)

DECLARE
```

```
    Purchase_ID INT;
    Cleaning_Fee DEC(10,2);
    Service_Fee DEC(10,2);
    Rate DEC(10,2);
    DateDiff INT;
    Confirmation_Code INT;

BEGIN

    SELECT (COUNT(Purchases_ID) FROM Purchases) + 1
    INTO Purchase_ID;

    SELECT DATEDIFF(day, Start_Date, End_Date)
    INTO DateDiff;

    SELECT Property_Listing.Service_Fee,
    Property_Listing.Cleaning_Fee
    FROM Property_Listing
    WHERE Property_Listing.Listing_ID=New_Purchase.Listing_ID
    INTO New_Purchase.Service_Fee, New_Purchase.Cleaning_Fee

    Rate := (DateDiff * Cost_Per_Night) + Cleaning_Fee +
    Service_Fee;

    SELECT ROUND(RAND()*(1000000000000),0)
    INTO Confirmation_Code;

    INSERT INTO Purchases
         (Purchase_ID,
         User_ID,
         Listing_ID,
         Confirmation_Code,
         Number_Of_People,
         Rate,
         Start_Date,
         End_Date)
    VALUES
         (Purchase_ID,
         User_ID,
         Listing_ID,
         Confirmation_Code,
          Number_Of_People,
          Rate,
          Start_Date,
          End_Date)

END;
```

2) Inserts a new property into the Property table.

```
CREATE OR REPLACE PROCEDURE New_Property (
    (State_or_Area IN VARCHAR(20),
     City IN VARCHAR(20),
     Street_Address IN VARCHAR(50),
     Country IN VARCHAR(20),
     Property_Type IN VARCHAR(20),
     Num_Bedrooms IN INT,
     Max_Guest_Num IN INT,
     Num_Beds IN INT,
     Num_Baths IN INT,
     Cleaning_Fee IN DECIMAL(10,2),
     Cost_per_Night IN DECIMAL(10,2),
     Service_Fee IN DECIMAL(10,2))

DECLARE
     Property_ID INT;

BEGIN

     SELECT (COUNT(Property_ID)+1) INTO Property_ID
     FROM Property;

INSERT INTO Property
     (Property_ID,
     State_or_Area,
     City,
     Street_Address,
     Country,
     Poperty_Type,
     Num_Bedrooms,
     Max_Guest,
     Num_Beds,
     Num_Baths,
     Cleaning_Fee,
     Cost_per_Night,
     Service_Fee)
VALUES
     (Property_ID,
     State_or_Area,
     City,
     Street_Address,
     Country,
     Poperty_Type,
     Num_Bedrooms,
     Max_Guest,
     Num_Beds,
     Num_Baths,
     Cleaning_Fee,
     Cost_per_Night,
     Service_Fee)

END;
```

Triggers

1) Update average reviews upon new review for a given property.

We use the following formula for updating the averages:

New_Mean = (Old_Mean * Old_Number_Of_Points + New_Value) / (Old_Number_Of_Points + 1)

This allows us to update the average without having to add up all the reviews every single time.

To accomplish this trigger, we assume the existence of the following table with foregin key Property_ID pointing to the Property entity.

Rating
Property_ID
Overall_Rating
Accuracy_Rating
Value_Rating
Check_In_Rating
Communication_Rating
Number_Of_Reviews

```
CREATE OR REPLACE TRIGGER Update_Reviews_After_Insert
AFTER INSERT ON Reviews
DECLARE

     Property_ID INT;
     ANY_ROWS_FOUND INT;

     Old_Overall_Rating DECIMAL(3,2);
     Old_Accuracy_Rating DECIMAL(3,2);
     Old_Value_Rating DECIMAL(3,2);
     Old_Check_In_Rating DECIMAL(3,2);
     Old_Communication_Rating DECIMAL(3,2);
     Old_Review_Count INT;

     New_Overall_Rating DECIMAL(3,2);
     New_Accuracy_Rating DECIMAL(3,2);
     New_Value_Rating DECIMAL(3,2);
     New_Check_In_Rating DECIMAL(3,2);
     New_Communication_Rating DECIMAL(3,2);
     New_Review_Count INT;

BEGIN

     SELECT Property_ID INTO Property_ID
     FROM Property_Listing as PL
     WHERE PL.Listing_ID = :NEW.Listing_ID
```

```
-- Checks to See if Property Has Reviews

SELECT COUNT(*)
INTO ANY_ROWS_FOUND
FROM Rating
WHERE Rating.Property_ID = Property_ID

-- If no previous reviews, just use the new review.

IF ANY_ROWS_FOUND = 0 THEN

     INSERT INTO Rating
          (Properity_ID,
          Overall_Rating,
          Accuracy_Rating,
          Value_Rating,
          Check_In_Rating,
          Communication_Rating,
          Number_Of_Reviews)
     VALUES
          (Property_ID,
          :NEW.Overall_Rating,
          :NEW.Accuracy_Rating,
          :NEW.Value_Rating,
          :NEW.Check_In_Rating,
          :NEW.Communication_Rating,
          :NEW.Number_Of_Reviews,
          1);

-- If previous reviews, update using new review.

ELSE

     -- Select old averages

     SELECT
          Old_Overall_Rating := R.Overall_Rating,
          Old_Accuracy_Rating := R.
             Accuracy_Rating,
          Old_Value_Rating := R.Value_Rating
          Old_Check_In_Rating :=
             R.Check_In_Rating,
          Old_Communication_Rating :=
             R.Communication_Rating,
          Old_Review_Count := R.Review_Count
     FROM Rating AS R
     WHERE R.Property_ID = Property_ID;


     -- Compute New Average Reviews.

     New_Review_Count := Old_Review_Count + 1;

     New_Overall_Rating := (Old_Overall_Rating *
     Old_Review_Count + :NEW.Overall_Rating) /
     New_Review_Count;

     New_Accuracy_Rating := (Old_Accuracy_Rating *
     Old_Review_Count + :NEW.Accuracy_Rating) /
     New_Review_Count;

     New_Value_Rating := (Old_Value_Rating *
     Old_Review_Count + :NEW.Value_Rating) /
     New_Review_Count;

     New_Check_In_Rating := (Old_Check_In_Rating *
     Old_Review_Count + :NEW.Check_In_Rating) /
     New_Review_Count;

     New_Communication_Rating :=
     (Old_Communication_Rating * Old_Review_Count +
     :NEW.Communication_Rating) / New_Review_Count;


     -- Insert New Average Reviews.


     INSERT INTO Rating
          (Properity_ID,
          Overall_Rating
          Accuracy_Rating
```

```
               Value_Rating
               Check_In_Rating
               Communication_Rating
               Number_Of_Reviews)
          VALUES
               (Property_ID,
               New_Overall_Rating,
               New_Accuracy_Rating,
               New_Value_Rating,
               New_Check_In_Rating,
               New_Communication_Rating,
               New_Number_Of_Reviews,
               New_Review_Count);

     END IF;

END;
```

2) Update response rate for a given Host after a new message is sent

For this trigger, I assumed the existence of the following table, slightly modified from the Assignment 4 submission:

Messages
Message_ID
User1_ID
User2_ID
Contents
Date

```
CREATE OR REPLACE TRIGGER Update_Host_Response_Rate
AFTER INSERT ON Messages


DECLARE
     DECIMAL(2,2) R_Rate;
     INT User1ID;
     INT User2ID;
     DATE HDate;
     INT Timely_Responses;
     INT Untimely_Responses;

BEGIN

     SELECT User1_ID, User2_ID, Time, Date INTO User1ID,
     User2ID,HDate
     FROM Messages
     WHERE DISTINCT Message_ID  = :NEW.Message_ID

     DECLARE ReceivedMessages TABLE
     SELECT ROW_NUMBER() OVER(ORDER BY Date) AS id, User1ID,
     User2ID, Date FROM Messages, Host
     WHERE Messages.User2_ID=User2ID AND DATEADD(month,1,Date) <
     GETDATE() AND Messages.User2_ID IN (SELECT UserID FROM
     Host);

          -- Creates table of inquires received by the Host in
the past month

     Timely_Responses :=0;
     Untimely_Responses :=0;
     FOR i IN 0..COUNT(ReceivedMessages)
     LOOP
          IF EXISTS
          SELECT Message_ID FROM Messages, ReceivedMessages
          WHERE (id=i AND User_1ID=User2ID AND User_2ID=User1ID
AND Messages.Date>ReceivedMessages.Date AND
Messages.Date<DATEADD(day,1,ReceivedMessages.Date)
```

```
          THEN Timely_Responses := (Timely_Responses+1);

          ELSE Untimely_Responses := (Untimely_Responses+1);

          END IF;

     END LOOP;
          -- For all host messages received in the past month,
          -- increments timely or untimely response,
          -- based on response time


     R_Rate := 100 * (Timely_Responses /
     (Timely_Responses+Untimely_Responses)

          -- Calculates the host's response rate

     UPDATE Host
     SET Response_Rate=R_Rate
     WHERE Host.User_ID=User2ID

          -- Updates the host's response rate

END;
```