# DATABASE DESIGN FOR AIRBNB

CS 6360.002
FINAL PROJECT

TEAM: AIRBNB-1

Akash P Akki

APA190001

Rahul Shivakumar

RXS180137

Theja Shree Kunam

TXK190012

# Table of Contents

# Requirements

## Introduction

The system facilitates booking a stay and listing a property for stays. The user of Airbnb can create an account and browse for properties available at a particular location of choice. Additional choices like type of stay, number of people that can be accommodated can be selected while browsing. User can also look for amenities and facilities available, previous user reviews for a particular stay and based on this he can judge the convenience. Details like price and availability are also mentioned. A property owner can use Airbnb to list his property for stays. He can mention all the details that the user needs to know. For any further questions about the property, the user and owner are facilitated with a messaging service to get in contact with each other. Restaurants that are nearby a property i.e. belong to a particular zip code are listed in a separate tab. Information about the restaurant like name, cuisine, rating and description are recorded in the database.

## Main Components in the system:

### Property

Each listing is stored in the database as a property with details like name, description, ID, number of bedrooms, number of baths. Property can belong to one of the types – house, apartment, service apartment, villa, guest suite, cottage, resort, guesthouse, hostel, bungalow. Availability of the property is maintained in the database for the users to refer and book accordingly. Properties marked as 'instant book' have the convenience of booking them immediately without having to be reviewed by the lister/owner.

### Booking

Every time a user makes a booking of a property, an entry is made in this table with corresponding user's ID and property ID. Details like check-in date, checkout date, booking ID are also tracked. Each booking is associated with a status which can take values to be reviewed or booked. In case of properties marked as 'instant book', the status field gets updated to booked as soon as user completes the transaction. After tax amount for the booking is calculated and updated in the table. Tax varies on the type of property – for example, luxurious properties like Villa has more tax compared to a Cottage.

### Amenities

Each property can have amenities and facilities like kitchen, Wi-Fi, microwave, refrigerator, washing machine, dryer, gym, pool. Mentioning these explicitly helps user understand more about the property and decide if it fits his needs. These attributes are simple Yes/No options which the Owner has to opt while listing his property.

## Payment

Once the transaction is made, the details of the total amount paid (we calculate this after deducting the discount, if any from the amount after tax), booking ID, user ID is stored in the database. Each payment is associated with a payment ID which helps in uniquely identifying each entry. It is important to keep a track of all these details as in case of discrepancies or if any refund needs to be processed, this can be referred. Discrepancies can include user cancelling the booking, user logging complaints about property and expecting a refund etc. Information about any promos/ discounts applied on the transaction are taken note of.

## User

User details like name, email, address are collected during creation of account. User account details like username and password are stored in the database. User ID is the unique entry using which each user is identified.

## Calendar

The date slots during which each property is available is recorded. Whenever a property is listed, start date and end date are noted. This information needs to be stored in order to avoid any clashes. A property can be listed multiple times with different starting and ending date.

## Address

Each property's address details i.e. the street, city and zip code are recorded. This is further used to retrieve the restaurants nearby the property. Each restaurant address details like street, city, zip code is noted. Multiple restaurants around a specific area can be searched based on a zip code.

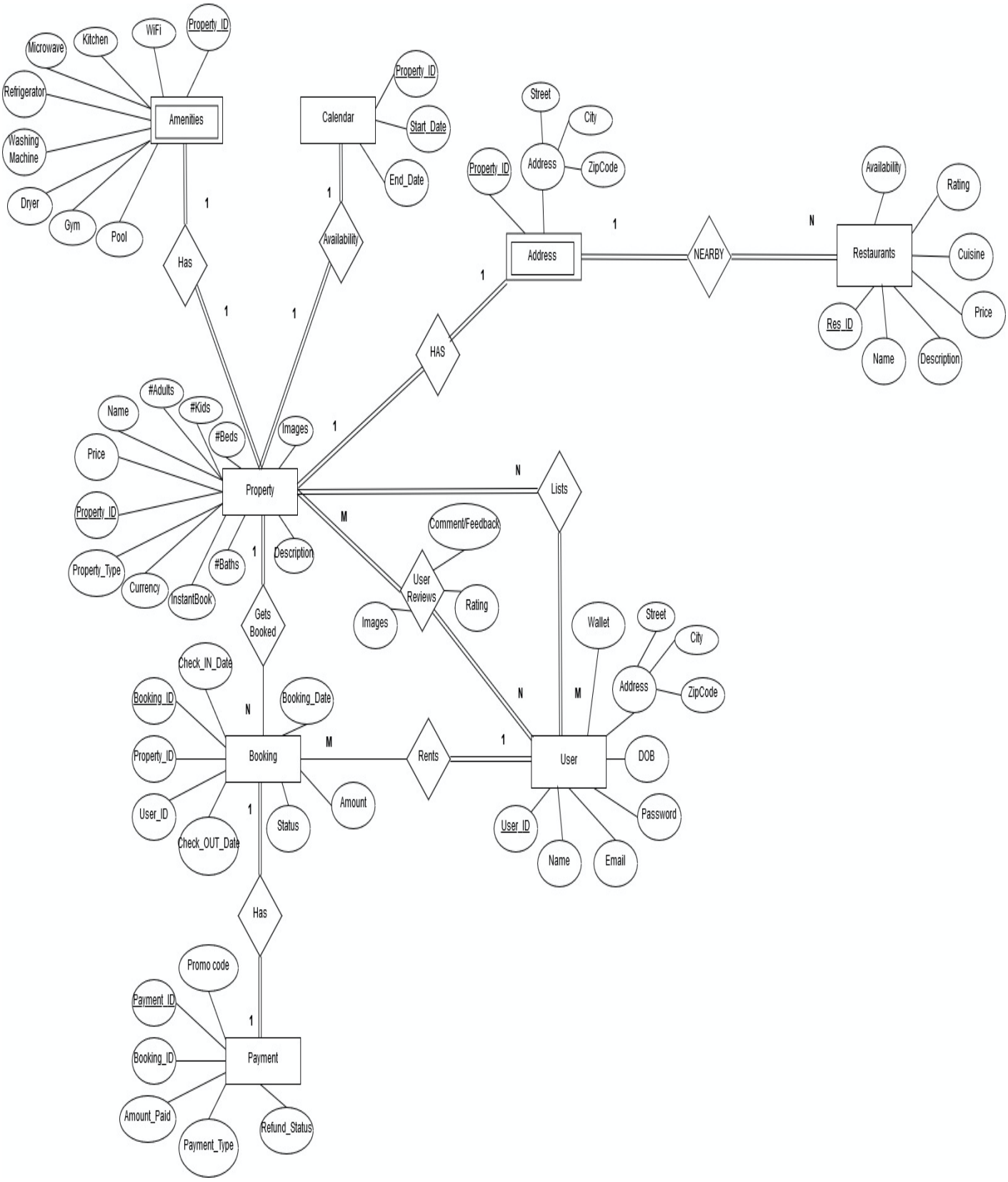
## Restaurant

The restaurants that are closer to the property are stored. Each restaurant is related to the property entity. The details about restaurant like name, description, price, cuisine, availability and rating are stored. Each restaurant is uniquely identified by restaurant ID.

## ASSUMPTIONS:

1. We assume that the images uploaded by owner or the user is compressed and of type varchar.
2. We assume that each user uploads only one image as part of his review.

# Modeling of Requirements as ER-Diagram



The requirements can be summarized/ derived from ERD as –

Entities and Attributes:

1. Property entity has attributes Property_ID, Property_Type, Price, Name, #Adults, #Kids, #Beds, #Baths, Currency, InstantBook.
2. Amenities entity has attributes Property_ID, Wi-Fi, Kitchen, Microwave, Refrigerator, Washing machine, dryer, gym, pool. The values for these attributes are yes or no.
3. Calendar entity has attributes Property_ID, Start_Date, End_Date.
4. Booking entity has attributes Booking_ID, Property_ID, User_ID, Check_in_Date, Check_out_Date, Status, Booking_Date, amount.
5. Payment entity has attributes Payment_ID, Booking_ID, amount_paid Payment_Type, Refund_Status, Promo_code.
6. User entity has attributes User_ID, name, email, DOB, password, address-street, city, zipcode, wallet.
7. Address entity has attributes Property_ID, composite attribute address with simple attributes Street, City, ZipCode.
8. Restaurant entity has attributes Restaurant_ID, Property_ID, name, description, price, cuisine, availability, rating, propertyID that it is closest to.

Relationships:

1. Each property can have a unique list of amenities. This is a 1:1 relationship.
2. The property's availability can be marked in a calendar. Each property maintains its own calendar. This is a 1:1 relationship.
3. Multiple bookings can be made on the same property during different dates. This is a 1:M relationship between property and booking entities.
4. A booking is associated with one payment receipt. This is 1:1 relationship.
5. Multiple owners can own a property and list it. Multiple properties can be owned and listed by a single owner. This a M: N relationship between User and Property.
6. Multiple users can rate the same property. A user can rate multiple properties. This is a M: N relationship between user and Property.
7. A user can make multiple bookings of different properties at different times, but a booking is associated with a single user. This is a 1: N relationship between user and booking.
8. The relationship relation user_reviews has attributes PropertyID, images, ratings, comments.
9. Each Property can have one address and each address corresponds to one property. This is a 1:1 relationship between address and property.

10. Multiple restaurants are associated with a zip code that is restaurants near a particular area. However, each restaurant has unique address. This is a 1:M relationship between address and restaurants.

## Functional Dependencies

a) Property

- Property_ID -> Name, Price, Description Images, Property_Type, Currency, InstantBook, #Baths, #Adults, #Kids, #Beds

b) User

- User_ID -> User_Name, Email, Password, DOB, Street, City, ZipCode, Wallet

c) Booking

- Booking_ID -> Property_ID, User_ID, Status, Booking_Date, Check_in_Date, Check_Out_Date, Status

d) Payment

- Payment_ID, Booking_ID ->Amount_Paid, Refund_Status, Promo_code, Payment_Type
- Payment_Type -> Promo_code

e) Amenities

- Property_ID -> Wifi, Kitchen, Microwave, Refrigerator, Dryer, Gym, Pool, washingmachine

f) Calendar

- Property_ID , Start_Date -> End_Date

g) User Reviews

- Property_ID, User_ID -> Images, Rating, Comment

h) Address

- Property_ID -> Street, City, ZipCode

i) Restaurant

- Restaurant_ID , Property_ID-> Name, Description, Price, Cuisine, Availability, Rating
- Restaurant_ID -> Name, Description, Price, Cuisine, Availability, Rating

## Functional Dependencies that violated Normalization Rules:

Property_ID -> Images

violates 1NF because images are a multi-valued attribute

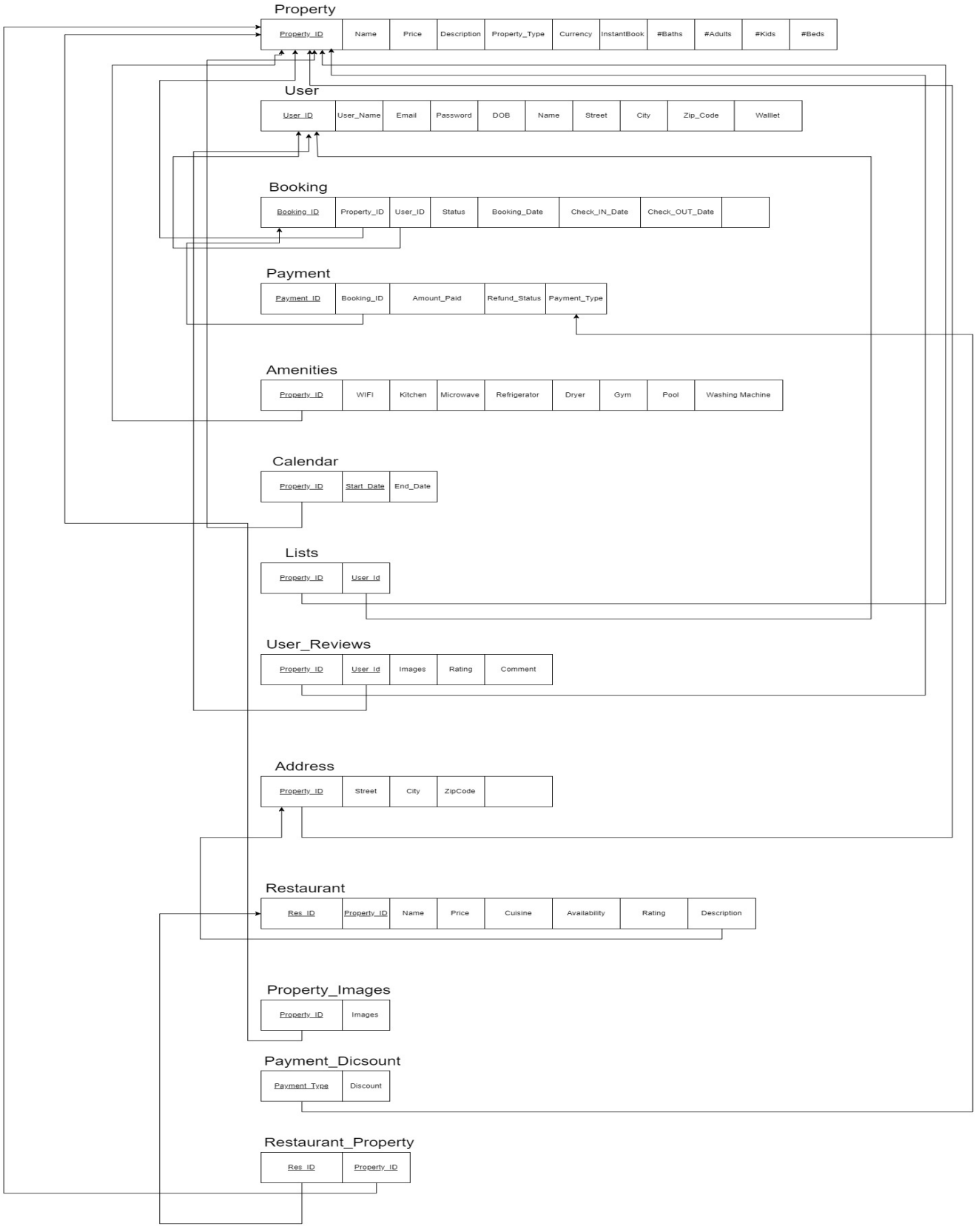
Restaurant_ID -> Name, Description, Price, Cuisine, Availability, Rating

violates 2NF because Restaurant_ID is part of primary key {Restaurant_ID, Property_ID}

Payment_Type -> Promo_Code

violates 3NF because Payment_Type is not a super key and Promo_Code is not a candidate key.

# Mapping of ERD in Relational Schema-Final Schema after normalization



Property

- Primary Key: Property_ID

User

- Primary Key: User_ID

Booking

- Primary Key: Booking_ID
- Foreign Key: PropertyID References Property (Property_ID) and UserID References User (User_ID)

Payment

- Primary Key: Payment_ID
- Foreign Key: Booking_ID References Booking (Booking_ID)

Amenities

- Primary Key: Property_ID
- Foreign Key: Property_ID References Property (Property_ID)

Calendar

- Primary Key: Property_ID, Start_Date
- Foreign Key: Property_ID References Property (Property_ID)

Lists

- Primary Key: Property_ID, User_ID
- Foreign Key: Property_ID references Property (Property_ID) and User_ID references User (User_ID)

User_Reviews

- Primary Key: Property_ID, User_ID
- Foreign Key: Property_ID references Property (Property_ID) and User_ID references User (User_ID)

Address

- Primary Key: Property_ID
- Foreign Key: Property_ID references Property (Property_ID)

Restaurant

- Primary Key: Restaurant_ID, Property_ID
- Foreign Key: Property_ID references Property (Property_ID)

Property_Images

- Primary Key: Property_ID
- Foreign Key: Property_ID references Property (Property_ID)

Payment_discount

- Primary Key: Payment_type

Restaurant_Property

- Primary Key: Property_ID, Restaurant_ID
- Foreign Key: Property_ID references Property (Property_ID)
- Foreign Key: Restaurant_ID references Restaurant(Restaurant_ID)

## SQL STATEMENTS

```
CREATE TABLE PROPERTY
(
        PROPERTY_ID     NUMBER PRIMARY KEY,
        NAME            VARCHAR (30),
        PRICE           NUMBER,
        PROPERTY_TYPE   VARCHAR (15),
        CURRENCY        NUMBER,
        INSTANTBOOK     BOOLEAN,
        #BATHS          NUMBER,
        #ADULTS         NUMBER,
        #KIDS           NUMBER,
        #BEDS           NUMBER,
        DESCRIPTION     VARCHAR (100)
)
CREATE TABLE USER
(
        USER_ID         NUMBER PRIMARY KEY,
        USERNAME        VARCHAR (30),
        EMAIL           VARCHAR (30),
        PASSWORD        VARCHAR (30),
        DOB             DATE,
        WALLET          NUMBER,
        STREET          VARCHAR (50),
        CITY            VARCHAR (50),
        ZIPCODE         NUMBER
)
CREATE TABLE BOOKING
(
        BOOKING_ID      NUMBER PRIMARY KEY,
        PROPID          NUMBER,
        USERID          NUMBER,
        BOOKING_DATE    DATE,
        CHECK_IN_DATE   DATE,
```

```
    CHECK_OUT_DATE   DATE,
    FOREIGN KEY(PROPID) REFERENCES PROPERTY(PROPERTY_ID) ON DELETE CASCADE
    FOREIGN KEY(USERID) REFERENCES USER(USER_ID) ON DELETE CASCADE
)


CREATE TABLE PAYMENT
(
    PAYMENT_ID        NUMBER PRIMARY KEY,
    BOOKINGID         NUMBER,
    AMOUNT_PAID       NUMBER,
    REFUND_STATUS     NUMBER,
    FOREIGN KEY(BOOKINGID) REFERENCES BOOKING(BOOKING_ID) ON DELETE
CASCADE
)
CREATE TABLE AMENITIES
(
    PROPID            NUMBER PRIMARY KEY,
    WIFI              BOOLEAN,
    KITCHEN           BOOLEAN,
    MICROWAVE         BOOLEAN,
    REFRIDGERATOR     BOOLEAN,
    DRYER             BOOLEAN,
    GYM               BOOLEAN,
    POOL              BOOLEAN,
    WASHINGMACHINE BOOLEAN
    FOREIGN KEY(PROPID) REFERENCES PROPERTY(PROPERTY_ID) ON DELETE CASCADE
)
CREATE TABLE CALENDAR
(
    PROPID            NUMBER PRIMARY KEY,
    START_DATE        DATE PRIMARY KEY,
    END_DATE          DATE,
    FOREIGN KEY(PROPID) REFERENCES PROPERTY(PROPERTY_ID) ON DELETE CASCADE
)
```

```
CREATE TABLE LISTS
(
        PROPID      NUMBER PRIMARY KEY,
        UID         NUMBER PRIMARY KEY,
        FOREIGN KEY(PROPID) REFERENCES PROPERTY(PROPERTY_ID) ON DELETE CASCADE
        FOREIGN KEY(UID) REFERENCES USER(USER_ID) ON DELETE CASCADE
)
CREATE TABLE USER_REVIEWS
(
        PROPID          NUMBER PRIMARY KEY,
        UID             NUMBER PRIMARY KEY,
        IMAGE           VARCHAR,
        RATING          NUMBER,
        COMMENT         VARCHAR (100),
        FOREIGN KEY(PROPID) REFERENCES PROPERTY(PROPERTY_ID) ON DELETE CASCADE
        FOREIGN KEY(UID) REFERENCES USER(USER_ID) ON DELETE CASCADE
)
CREATE TABLE ADDRESS
(
        PROPID      NUMBER PRIMARY KEY,
        STREET      VARCHAR (50),
        CITY        VARCHAR (50),
        ZIPCODE     NUMBER,
        FOREIGN KEY(PROPID) REFERENCES PROPERTY(PROPERTY_ID) ON DELETE CASCAD
)
CREATE TABLE RESTAURANT_PROPERTY
(
        RESTAURANT_ID           NUMBER PRIMARY KEY,
        PROPID                  NUMBER PRIMARY KEY,
        FOREIGN KEY(PROPID) REFERENCES PROPERTY(PROPERTY_ID) ON DELETE CASCAD
)
CREATE TABLE RESTAURANT
(
        RESTAURANT_ID   NUMBER PRIMARY KEY,
```

```
    NAME            VARCHAR (50),
    DESCRIPTION     VARCHAR (50),
    PRICE           NUMBER,
    CUISINE         VARCHAR (10),
    AVAILABILITY    VARCHAR (10),
    RATING          NUMBER,
    FOREIGN KEY(RESTAURANT_ID) REFERENCES RESTAURANT(RESTAURANT_ID) ON
DELETE CASCADE
)

CREATE TABLE PROPERTY_IMAGES

(

    PROPID      NUMBER PRIMARY KEY,
    IMAGES      VARCHAR (1000),
    FOREIGN KEY(PROPID) REFERENCES PROPERTY(PROPERTY_ID) ON DELETE CASCADE
)

CREATE TABLE PAYMENT_DISCOUNT

(

    PAYMENT_TYPE    VARCHAR (20) PRIMARY KEY,
    PROMO_CODE      NUMBER
)
```

# PL/SQL STATEMENTS

## Triggers

### 1. Payment Refund

This Trigger fires when the status of the booking is changed to ‘Cancelled’. We update the refund status and add the amount back to user’s wallet. This trigger lets the refunds to be processed without any hassles. Users have extra convenience if they know that they can cancel the booking anytime and get the refund.

```
CREATE or REPLACE TRIGGER PAYMENT_REFUND
AFTER UPDATE ON BOOKING
FOR EACH ROW
WHEN (NEW.BOOKING_STATUS ='CANCELLED')
BEGIN
        SELECT AMOUNT_PAID INTO REFUND_AMT, User_ID INTO UID
        FROM PAYMENT, BOOKING, USER WHERE BOOKING_ID = NEW.BOOKING_ID AND BOOKING.BOOKING_ID = PAYMENT.BOOKING_ID AND BOOKING.User_ID = User.User_ID;
                UPDATE USER SET WALLET = WALLET + REFUND_AMT WHERE User_ID = UID;
                UPDATE PAYMENT SET REFUND_STATUS = 'Y' WHERE BOOKING_ID = NEW.BOOKING_ID;
END;
```

### 2. Wallet Payment Check

This trigger fires when wallet balance is less than the amount to be paid to book a property. This trigger raises and error in that case and does not allow the user to proceed further. This trigger can act like a gentle reminder for the user to add more money in their wallet before booking a property. User can also choose another payment option and pay using it.

```
CREATE or REPLACE TRIGGER WALLET_PAY_CHECK

AFTER UPDATE ON PAYMENT

FOR EACH ROW

WHEN (NEW.PAYMENT_TYPE ='WALLET')

BEGIN

        SELECT AMOUNT_PAID INTO AMOUNT, User_ID INTO UID, WALLET INTO W_AMT,

        FROM PAYMENT, BOOKING, USER WHERE PAYMENT_TYPE = NEW.PAYMENT_TYPE AND BOOKING.BOOKING_ID = PAYMENT.BOOKING_ID AND BOOKING.User_ID = User.User_ID;

                IF W_AMT < AMOUNT THEN

                        raise_application_error(2000,'ERROR: INSUFFICIENT WALLET BALANCE! PLEASE CHOOSE A DIFFERENT PAYMENT METHOD.');

END;
```

## Stored Procedures

1. Calculate Tax

   This procedure calculates tax for property and adds it to the total money to be paid. The price of property is considered for calculating the tax amount and this is then added to the initial cost calculated by price per day multiplied by number of days for which property is booked. The tax amount depends on the property type.

```
create or replace PROCEDURE CALC_TAX_AMT

(PRICE IN PROPERTY.PRICE%TYPE

PROP_TYPE IN PROPERTY.PROPERTY_TYPE%TYPE,

AMT OUT BOOKING.AMOUNT%TYPE)  AS


BEGIN

        SELECT AMOUNT INTO AMT_WITH_TAX FROM BOOKING;
```

```
        IF PROP_TYPE = 'VILLA' OR PROP_TYPE = 'HOTEL_ROOM' THEN
                AMT_WITH_TAX := PRICE + (PRICE * 0.18);
        ELSE IF PROP_TYPE = 'SHARED_ROOM' OR PROP_TYPE = 'COTTAGE' THEN
                AMT_WITH_TAX := PRICE + (PRICE * 0.12);
        ELSE
                AMT_WITH_TAX := PRICE + (PRICE * 0.08);
        END IF;
    END IF;
    END;
```

2. Calculate Discount Amount

This procedure takes the amount calculated i.e. addition of tax and initial cost into consideration. This amount is used to calculate the discount amount. Discount amount also depends on the payment type used by the user. Users using card payment tend to get more discount than users paying using other payment methods. After discount is applied, the final amount due is calculated and updated.

```
create or replace PROCEDURE CALC_DISCOUNT_AMT
(UID IN USER.USER_ID%TYPE,
AMOUNT IN BOOKING.AMOUNT%TYPE,
DISC_AMT OUT PAYMENT.AMOUNT_PAID%TYPE)  AS

PAYTYPE PAYMENT.PAY_TYPE%TYPE;

BEGIN
        SELECT PAYMENT_TYPE INTO PAYTYPE FROM PAYMENT WHERE
BOOKING_ID= BOOKINGID;
        SELECT AMOUNT_PAID INTO DISC_AMT FROM PAYMENT;
```

```
        IF PAYTYPE = 'CardPayment' THEN
                DISC_AMT := AMOUNT - AMOUNT * 0.15;
        ELSE
                DISC_AMT := AMOUNT - AMOUNT * 0.10;
        END IF;
END IF;
END;
```