TERM PROJECT: AİRC&C DATABASE SYSTEM

PREPARED BY

STUDENT ID	NAME- SURNAME
528241005	AYSAN PAKMANESH
528241003	SELİN BOZKURT
528241041	SAMET ERKEK

SUBMISSION DATE:

CRN : ...15176...

COURSE : DATABASE MANAGEMENT AND BİG DATA

CONTENT

1. SYSTEM ANALYSIS

INTRODUCTION:

AirC&C is a **Property Rental Management System** that facilitates property rentals by connecting hosts and guests. It ensures streamlined processes for listing, booking, communication, and reviews, while also maintaining data validation.

Main Idea: A platform to make short-term property rentals.

How It Works:

1. Hosts create property listings with detailed information, photos, and pricing.
2. Guests browse properties, interact with Hosts, book them, and leave reviews.
3. Admins ensure content moderation and system integrity.

Benefits:

1. Enhances trust via validated bookings and reviews.
2. Provides analytics for better decision-making.
3. Automates management tasks, reducing errors and manual work.
4. Enabling direct interaction between users.

Similar Global/Local Systems:

Airbnb (Global): A leading property rental platform offering similar features like property listings, bookings, and reviews.

Emlakjet (Local - Turkey): A property marketplace catering to rental and sale properties, focusing more on long-term rentals and sales.

Booking.com (Global): Primarily focuses on vacation rentals and hotel bookings.

Sahibinden (Local - Turkey): Features property rentals with broader classified ads.

Stakeholders of the System:

1. Hosts: Property owners who list and manage their rentals.
2. Guests: Users who book properties and leave reviews.
3. System Administrators (Editors): Oversee operations, approve listings, and maintain data integrity by handling disputes.

KEY FUNCTİONALİTİES OF THE SYSTEM:

1. **User Management**
   - Registration/login for hosts and guests.
   - Profile setup with account types and personal information.

2. **Property Listings**
   - Hosts can create, edit, and manage property listings.
   - Add photos, descriptions, amenities, and pricing.

3. **Bookings**
   - Guests can search, book, and pay for properties.
   - Real-time booking status updates (**Pending**, **Confirmed**, **Cancelled**).

4. **Reviews & Ratings**
   - Guests can rate properties post-stay.
   - System can find biased or invalid reviews automatically.

5. **Messaging**
   - Direct communication between hosts and guests for queries and coordination.

6. **Data Validation**
   - Ensures booking, listing, and review accuracy (e.g., no overlapping bookings, invalid dates).

7. **Analytics & Insights**
   - Identifies popular property types, high-earning hosts, and guest activity trends.

8. **Editor Tools**
   - Moderates content, resolves disputes, and monitors system health.

2. DATA REQUIREMENTS

1. User Management

- **Data Needs**:
  - Store user details: Name, email, phone number, password, profile picture, account type (Host/Guest), and registration date.
  - Ensure email uniqueness and secure passwords.
  - Enable different role-specific functionalities (Host, Guest, Editor).
- **Operations**:
  - Register and authenticate users.
  - Update user profiles (e.g., uploading profile pictures).
  - Retrieve user details for system interactions.

---

2. Property Listings

- **Data Needs**:
  - Maintain property information: Title, description, location, amenities, property type, price per night, maximum guests, and listing date.
  - Associate properties with hosts and include images with upload details (URL, primary image flag).
- **Operations**:
  - Add, edit, and retrieve property details.
  - Display property information with images for guest searches.

---

3. Booking Management

- **Data Needs**:
    - Track booking details: Guest ID, property ID, check-in and check-out dates, total price, and booking status (*Pending*, *Confirmed*, *Cancelled*).
    - Validate booking dates against property availability and listing dates.
    - Ensure booking accuracy (e.g., no overlaps).
- **Operations**:
    - Insert and manage bookings.
    - Retrieve booking statuses and calculate durations for billing.

---

4. Messaging

- **Data Needs**:
    - Store message content: Sender ID, receiver ID, message body, and timestamp.
    - Facilitate communication between hosts and guests before or after booking.
- **Operations**:
    - Insert new messages.
    - Retrieve conversation history for reference.

---

5. Reviews and Ratings

- **Data Needs**:
    - Maintain review details: Reviewer ID, property ID, rating, comments, and review date.

  - Validate reviews to ensure they are tied to confirmed bookings.
- **Operations**:
  - Store and retrieve property reviews.
  - Detect invalid or biased reviews using system logic.

---

6. Admin Moderation

- **Data Needs**:
  - Track property listing approvals.
  - Record disputes and monitor system metrics.
- **Operations**:
  - Approve or reject property listings.
  - Flag invalid bookings or reviews.

---

7. Analytics

- **Data Needs**:
  - Aggregate data for insights: Popular properties, active users, host earnings, guest spending, and booking trends.
  - Identify patterns based on booking counts and review scores.
- **Operations**:
  - Generate reports for decision-making.

  - Retrieve analytical data for stakeholders.

---

8. Data Validation

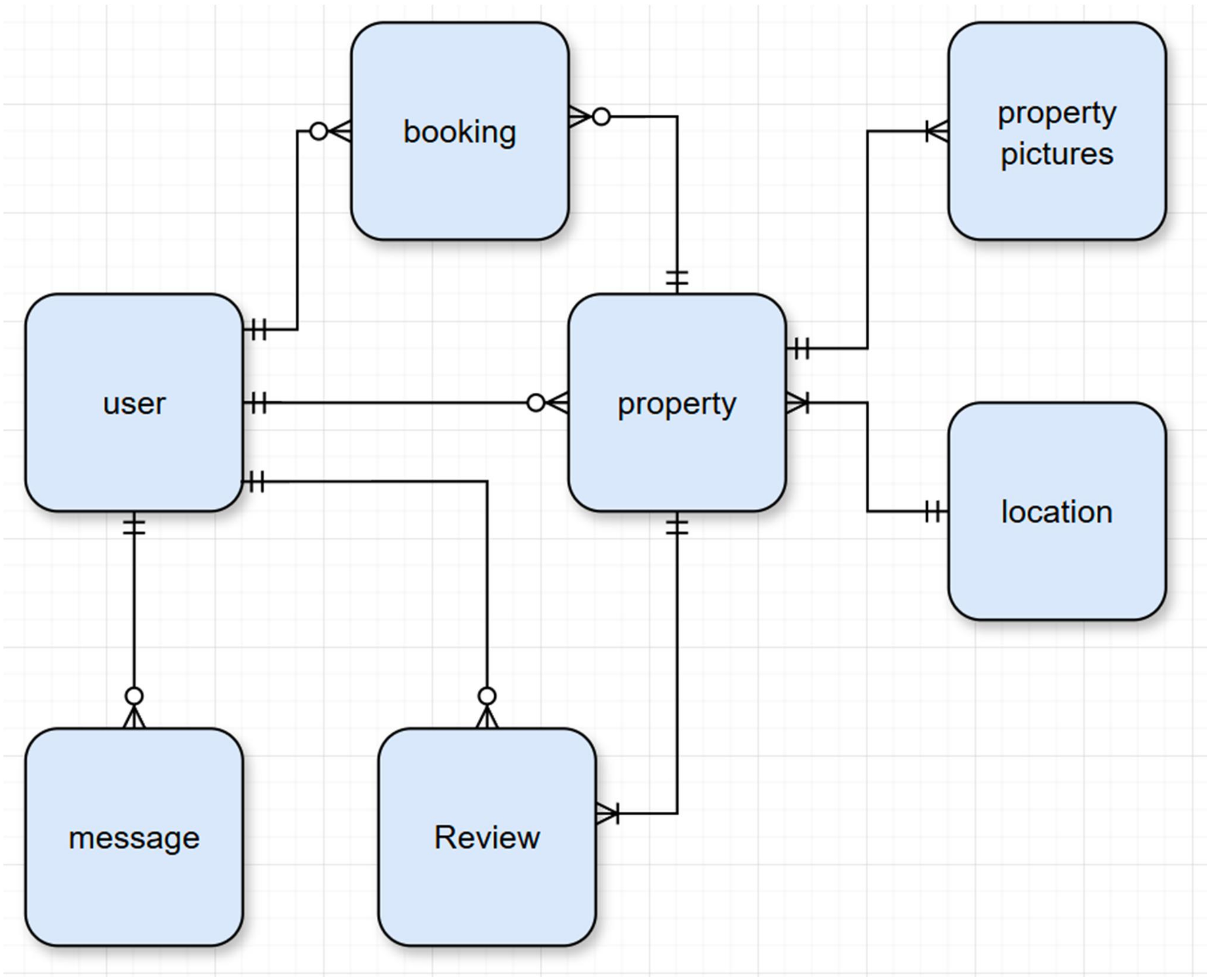
- **Data Needs**:
  - Check booking overlaps, invalid review dates, and property listing accuracy.
  - Ensure alignment between property images and listing dates or host registration dates.
- **Operations**:
  - Validate data integrity during insertion or updates.
  - Find and correct faulty records.

3. ENTITY RELATINSHIP DIAGRAMS

Entities in the Database:

1. **User**
    - Represents the users of the system, including both hosts and guests.
2. **Property**
    - Represents the properties listed by hosts for booking.
3. **Location**
    - Represents the geographic location of a property.
4. **Country**
    - Represents the countries associated with property locations.
5. **Booking**
    - Represents the bookings made by guests for properties.
6. **Review**
    - Represents the reviews left by users for properties they have booked.
7. **Message**
    - Represents the messages exchanged between users (e.g., guest and host).
8. **PropertyPictures**
    - Represents the pictures associated with property listings.

- **Conceptual ERD:**



- **Relationships between Entities:**

1. User ↔ Property

- **Relationship Type**: One-to-Many
- **Explanation**: A single user (host) can list multiple properties.
- **Optionality**:
  - A user **may have no properties** (e.g., a user who signed up as a guest only).
  - A property **must belong to exactly one user** (every property needs a host).

---

2. User ↔ Booking

- **Relationship Type**: One-to-Many
- **Explanation**: A single user (guest) can make multiple bookings.
- **Optionality**:
  - A user **may have no bookings** (e.g., a new user who hasn't booked yet).
  - A booking **must belong to exactly one user** (every booking is associated with a specific guest).

---

3. User ↔ Message

- **Relationship Type**: One-to-Many
- **Explanation**: A single user can send or receive multiple messages.
- **Optionality**:
  - A user **may not have sent or received any messages** (e.g., a user who hasn’t initiated communication).
  - A message **must have exactly one sender and one receiver** (a message is always sent by one user and received by one user).

---

4. User ↔ Review

- **Relationship Type**: One-to-Many
- **Explanation**: A single user can write multiple reviews.
- **Optionality**:
  - A user **may not have written any reviews** (e.g., a guest who hasn’t stayed anywhere yet).
  - A review **must be written by exactly one user** (each review is tied to one user).

---

5. Property ↔ Booking

- **Relationship Type**: One-to-Many

- **Explanation**: A single property can have multiple bookings.
- **Optionality**:
    - A property **may have no bookings** (e.g., a newly listed property with no reservations yet).
    - A booking **must belong to exactly one property** (each booking is for a specific property).

---

6. Property ↔ Review

- **Relationship Type**: One-to-Many
- **Explanation**: A single property can have multiple reviews.
- **Optionality**:
    - A property **may not have any reviews** (e.g., a newly listed property with no guest stays yet).
    - A review **must be about exactly one property** (each review pertains to a specific property).

---

7. Property ↔ Property Pictures

- **Relationship Type**: One-to-Many
- **Explanation**: A single property can have multiple associated pictures.
- **Optionality**:
    - A property **may have no pictures** (e.g., a property that hasn't uploaded any images yet).
    - A picture **must belong to exactly one property** (each picture is associated with a specific property).

---

8. Property ↔ Location

- **Relationship Type**: Many-to-One
- **Explanation**: Multiple properties can share the same location.
- **Optionality**:

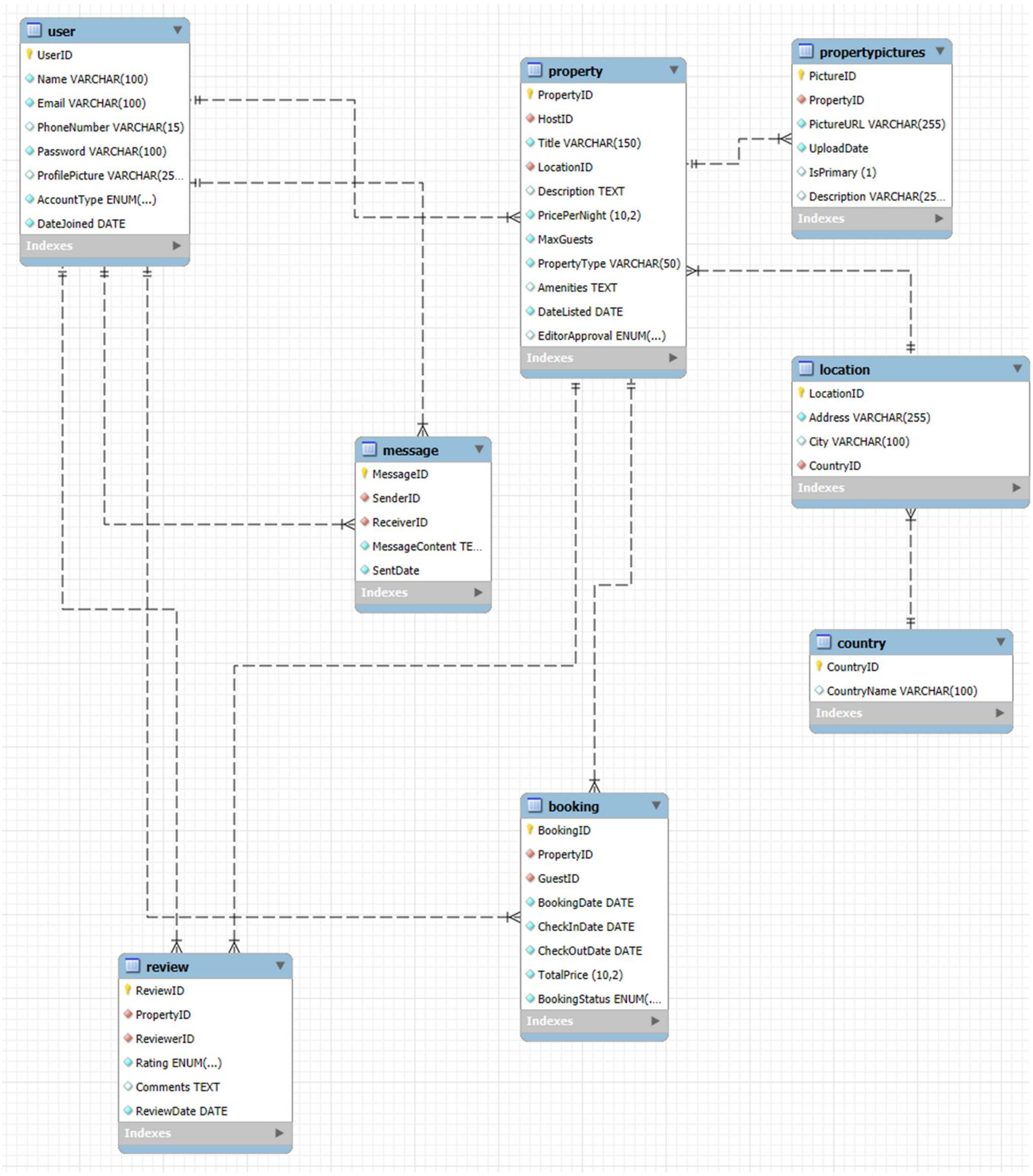
   - A property **must belong to exactly one location** (a property cannot exist without a location).
   - A location **may not have any properties** (e.g., a location that hasn't been assigned any properties yet).

---

9. Location ↔ Country

- **Relationship Type**: Many-to-One
- **Explanation**: Multiple locations can belong to the same country.
- **Optionality**:
   - A location **must belong to exactly one country** (every location must be within a country).
   - A country **may not have any locations** (e.g., a country not yet included in the system).

- **Logical ERD:**


user
UserID
Name VARCHAR(100)
Email VARCHAR(100)
PhoneNumber VARCHAR(15)
Password VARCHAR(100)
ProfilePicture VARCHAR(25...
AccountType ENUM(...)
DateJoined DATE
Indexes
property
PropertyID
HostID
Title VARCHAR(150)
LocationID
Description TEXT
PricePerNight (10,2)
MaxGuests
PropertyType VARCHAR(50)
Amenities TEXT
DateListed DATE
EditorApproval ENUM(...)
Indexes
propertypictures
PictureID
PropertyID
PictureURL VARCHAR(255)
UploadDate
IsPrimary (1)
Description VARCHAR(25...
Indexes
location
LocationID
Address VARCHAR(255)
City VARCHAR(100)
CountryID
Indexes
message
MessageID
SenderID
ReceiverID
MessageContent TE...
SentDate
Indexes
country
CountryID
CountryName VARCHAR(100)
Indexes
booking
BookingID
PropertyID
GuestID
BookingDate DATE
CheckInDate DATE
CheckOutDate DATE
TotalPrice (10,2)
BookingStatus ENUM(....
Indexes
review
ReviewID
PropertyID
ReviewerID
Rating ENUM(...)
Comments TEXT
ReviewDate DATE
Indexes

4. TABLES OF THE SYSTEM

1. User

The `User` table stores details about all users, whether they are guests or hosts.

- **ATTRİBUTES**:
  - **`UserID`**:
    - Data Type: `int`, auto-increment
    - Description: A unique identifier for each user. Using `int` ensures scalability as the user base grows.
    - Constraints: Primary Key (PK)
  - **`Name`**:
    - Data Type: `varchar(100)`
    - Description: Stores the user's full name, restricted to 100 characters.
    - Constraints: `NOT NULL` to ensure that every user must provide their name.
  - **`Email`**:
    - Data Type: `varchar(100)`
    - Description: Stores the user's email for unique identification and login.
    - Constraints: `UNIQUE`, `NOT NULL`
  - **`PhoneNumber`**:
    - Data Type: `varchar(15)`
    - Description: Allows users to provide an optional contact number.
    - Constraints: Optional (`NULL`)
  - **`Password`**:
    - Data Type: `varchar(100)`
    - Description: Stores encrypted passwords for user accounts.
    - Constraints: `NOT NULL`
  - **`ProfilePicture`**:
    - Data Type: `varchar(255)`
    - Description: Stores the file path or URL to the user's profile picture.
    - Constraints: Optional (`NULL`)
  - **`AccountType`**:
    - Data Type: `enum ('Guest', 'Host')`
    - Description: Distinguishes user roles for different functionalities.
    - Constraints: Default value is 'Guest'.
  - **`DateJoined`**:
    - Data Type: `DATE`
    - Description: Tracks the date of user registration.
    - Constraints: `NOT NULL`

---

2. Property

The `Property` table stores details about listings created by hosts.

- **ATTRİBUTES**:
  - `PropertyID`:
    - Data Type: `int`, auto-increment
    - Description: A unique identifier for each property.
    - Constraints: Primary Key (PK)
  - `HostID`:
    - Data Type: `int`
    - Description: Links the property to its owner (host).
    - Constraints: Foreign Key (FK) referencing `User.UserID`.
  - `Title`:
    - Data Type: `varchar(150)`
    - Description: A concise, descriptive name for the property.
    - Constraints: `NOT NULL`
  - `LocationID`:
    - Data Type: `int`
    - Description: Associates the property with a specific location.
    - Constraints: Foreign Key (FK) referencing `Location.LocationID`.
  - `Description`:
    - Data Type: `text`
    - Description: Provides additional details about the property.
    - Constraints: Optional (`NULL`)
  - `PricePerNight`:
    - Data Type: `decimal(10, 2)`
    - Description: Specifies the nightly rental cost.
    - Constraints: `NOT NULL`
  - `MaxGuests`:
    - Data Type: `int`
    - Description: Specifies the maximum number of guests allowed.
    - Constraints: `NOT NULL`
  - `PropertyType`:
    - Data Type: `varchar(50)`
    - Description: Categorizes the property (e.g., Apartment, House).
    - Constraints: `NOT NULL`
  - `Amenities`:
    - Data Type: `text`
    - Description: Lists available facilities (e.g., Wi-Fi, pool).
    - Constraints: Optional (`NULL`)
  - `DateListed`:
    - Data Type: `DATE`
    - Description: Indicates when the property was listed.
    - Constraints: `NOT NULL`
  - `EditorApproval`:
    - Data Type: `enum ('Pending', 'Listed')`
    - Description: Ensures only approved listings are visible.
    - Constraints: Default value is 'Pending'.

3. Booking

The `Booking` table manages reservation details.

- **ATTRİBUTES**:
  - **`BookingID`**:
    - Data Type: `int`, auto-increment
    - Description: Unique identifier for each booking.
    - Constraints: Primary Key (PK)
  - **`PropertyID`**:
    - Data Type: `int`
    - Description: Links the booking to a property.
    - Constraints: Foreign Key (FK) referencing `Property.PropertyID`.
  - **`GuestID`**:
    - Data Type: `int`
    - Description: Identifies the guest who made the booking.
    - Constraints: Foreign Key (FK) referencing `User.UserID`.
  - **`CheckInDate`**:
    - Data Type: `DATE`
    - Description: Start date of the stay.
    - Constraints: `NOT NULL`
  - **`CheckOutDate`**:
    - Data Type: `DATE`
    - Description: End date of the stay.
    - Constraints: `NOT NULL`
  - **`TotalPrice`**:
    - Data Type: `decimal(10, 2)`
    - Description: Total cost based on duration and price per night.
    - Constraints: `NOT NULL`
  - **`BookingStatus`**:
    - Data Type: `enum ('Pending', 'Confirmed', 'Cancelled')`
    - Description: Tracks the lifecycle of the booking.
    - Constraints: Default value is 'Pending'.

---

4. Review

The `Review` table stores guest feedback about their stays.

- **ATTRİBUTES**:
  - **`ReviewID`**:
    - Data Type: `int`, auto-increment
    - Description: A unique identifier for each review.
    - Constraints: Primary Key (PK)
  - **`PropertyID`**:
    - Data Type: `int`

-
  -
    - Description: Links the review to the property it is about.
    - Constraints: Foreign Key (FK) referencing `Property.PropertyID`.
  - **`ReviewerID`**:
    - Data Type: `int`
    - Description: Identifies the user who wrote the review.
    - Constraints: Foreign Key (FK) referencing `User.UserID`.
  - **`Rating`**:
    - Data Type: `int`
    - Description: Numeric score given to the property (e.g., 1-5).
    - Constraints: `NOT NULL`
  - **`Comments`**:
    - Data Type: `text`
    - Description: Written feedback provided by the reviewer.
    - Constraints: Optional (`NULL`)
  - **`ReviewDate`**:
    - Data Type: `DATE`
    - Description: The date the review was submitted.
    - Constraints: `NOT NULL`

---

5. Message

The `Message` table tracks conversations between users.

- **ATTRİBUTES**:
  - **`MessageID`**:
    - Data Type: `int`, auto-increment
    - Description: Unique identifier for each message.
    - Constraints: Primary Key (PK)
  - **`SenderID`**:
    - Data Type: `int`
    - Description: Identifies the user who sent the message.
    - Constraints: Foreign Key (FK) referencing `User.UserID`.
  - **`ReceiverID`**:
    - Data Type: `int`
    - Description: Identifies the user who received the message.
    - Constraints: Foreign Key (FK) referencing `User.UserID`.
  - **`MessageContent`**:
    - Data Type: `text`
    - Description: The body of the message.
    - Constraints: `NOT NULL`
  - **`SentDate`**:
    - Data Type: `DATETIME`
    - Description: Timestamp for when the message was sent.
    - Constraints: `NOT NULL`

6. PropertyPictures

The `PropertyPictures` table stores images related to properties.

- **ATTRİBUTES**:
  - **`PictureID`**:
    - Data Type: `int`, auto-increment
    - Description: Unique identifier for each picture.
    - Constraints: Primary Key (PK)
  - **`PropertyID`**:
    - Data Type: `int`
    - Description: Links the picture to its associated property.
    - Constraints: Foreign Key (FK) referencing `Property.PropertyID`.
  - **`PictureURL`**:
    - Data Type: `varchar(255)`
    - Description: Path or URL for the image file.
    - Constraints: `NOT NULL`
  - **`UploadDate`**:
    - Data Type: `DATE`
    - Description: Date when the picture was uploaded.
    - Constraints: `NOT NULL`
  - **`IsPrimary`**:
    - Data Type: `boolean`
    - Description: Indicates if the picture is the main display image.
    - Constraints: Default value is `false`.
  - **`Description`**:
    - Data Type: `text`
    - Description: Optional textual description of the image.
    - Constraints: Optional (`NULL`)

7. Location

The `Location` table stores address information for properties.

- **ATTRİBUTES**:
  - **`LocationID`**:
    - Data Type: `int`, auto-increment
    - Description: Unique identifier for each location.
    - Constraints: Primary Key (PK)
  - **`Address`**:
    - Data Type: `varchar(255)`
    - Description: Detailed street address of the property.
    - Constraints: `NOT NULL`

- **`City`**:
  - Data Type: `varchar(100)`
  - Description: The city where the property is located.
  - Constraints: `NOT NULL`
- **`CountryID`**:
  - Data Type: `int`
  - Description: Links the location to its country.
  - Constraints: Foreign Key (FK) referencing `Country.CountryID`.

---

8. Country

The `Country` table stores information about countries for properties and users.

- **ATTRİBUTES**:
  - **`CountryID`**:
    - Data Type: `int`, auto-increment
    - Description: Unique identifier for each country.
    - Constraints: Primary Key (PK)
  - **`CountryName`**:
    - Data Type: `varchar(100)`
    - Description: The name of the country.
    - Constraints: `NOT NULL`, `UNIQUE`

5. VIEWS OF THE SYSTEMS

==

View Name: InvalidBillingData

Description: This view identifies inconsistencies in booking costs and payments.

Purpose:

- Detects bookings where the calculated total price does not match the actual cost based on:

- The number of nights stayed.

- The property's nightly price.

- Highlights bookings with potential monetary transaction issues.

- Assists in flagging and resolving issues with hosts and guests.

- Actions:

- Bookings with `BookingStatus = 'Pending'` should be reviewed by the administration.

- Inconsistent bookings should ideally be flagged as "Cancelled."

==

```
CREATE OR REPLACE VIEW InvalidBillingData AS

SELECT

    b.BookingID,

    b.CheckInDate,

    b.CheckOutDate,

    b.TotalPrice,

    p.PricePerNight,
    DATEDIFF(b.CheckOutDate, b.CheckInDate) AS DurationInNights,

    p.PricePerNight * DATEDIFF(b.CheckOutDate, b.CheckInDate) AS RealPrice,

    b.BookingStatus

FROM Booking b

JOIN Property p ON b.PropertyID = p.PropertyID

WHERE b.TotalPrice != (p.PricePerNight * DATEDIFF(b.CheckOutDate, b.CheckInDate));
```

===

View Name: InvalidDateTimeData

Description: This view identifies invalid date-time data in the database.

Purpose:

- Detects bookings where dates are inconsistent with expected rules:

- Check-in or check-out dates are earlier than the property's listing date.

- Booking dates are earlier than the guest's account creation date.

- Booking or stay dates are logically incorrect (e.g., check-in is after check-out).

- Helps maintain data integrity and flag faulty inputs for correction.

```
-- ===========================================================


CREATE OR REPLACE VIEW InvalidDateTimeData AS

SELECT

    u.UserID,

    b.BookingID,

    b.CheckInDate,
    b.CheckOutDate,
    b.BookingDate,
    p.DateListed,
    u.DateJoined
FROM Booking b
JOIN Property p ON b.PropertyID = p.PropertyID
JOIN User u ON b.GuestID = u.UserID
WHERE b.CheckInDate < p.DateListed
   OR b.CheckOutDate < p.DateListed
   OR b.CheckInDate < u.DateJoined
   OR b.CheckOutDate < u.DateJoined
   OR b.BookingDate < u.DateJoined
   OR b.BookingDate < p.DateListed
   OR b.CheckInDate >= b.CheckOutDate
   OR b.BookingDate >= b.CheckInDate;
```

==

View Name: PropertyPerformanceMetrics

Description:

- This view provides a comprehensive analysis of property performance

- by aggregating data related to bookings and reviews. It calculates

- key performance metrics to aid in decision-making for property

- management, marketing, and pricing strategies.

Purpose:

- To evaluate property profitability and performance.
- To provide actionable insights into revenue, occupancy, and guest engagement.
- To categorize properties based on average revenue (Premium, Standard, Economy).

- Key Operations Performed:

- Aggregates booking data to calculate:
  * Total revenue (sum of confirmed booking prices).
  * Total number of bookings (confirmed only).
  * Average revenue per guest (total revenue divided by bookings).
  * Occupancy rate percentage (based on the number of booked nights and the total period).
- Aggregates review data to calculate:
  * Total number of reviews per property.
- Merges booking and review data with the property table to ensure all properties are included, even if they have no bookings or reviews.
- Prevents data duplication by using subqueries to separate booking and review aggregations.
- Includes logic to categorize properties based on average revenue:
  * Premium: Average revenue > 3000.
  * Standard: Average revenue between 1000 and 3000.
  * Economy: Average revenue < 1000.

Benefits:

- Helps hosts and administrators identify high-performing and underperforming properties.
- Provides insights to optimize pricing, marketing, and guest experience.
- Ensures data accuracy by avoiding duplication and calculating metrics independently.

==

```
CREATE OR REPLACE VIEW PropertyPerformance AS
SELECT
  p.PropertyID,
  p.Title AS PropertyTitle,
  COALESCE(b.TotalRevenue, 0) AS TotalRevenue,
  COALESCE(b.TotalBookings, 0) AS TotalBookings,
  COALESCE(ROUND(b.TotalRevenue / NULLIF(b.TotalBookings, 0), 2), 0) AS AvgRevenuePerGuest,
  COALESCE(r.TotalReviews, 0) AS TotalReviews,
  COALESCE(ROUND((b.TotalNights / NULLIF(DATEDIFF(CURDATE(), MIN(b.FirstBookingDate)), 0)) * 100, 2), 0) AS
OccupancyRatePercentage,
  CASE
    WHEN COALESCE(ROUND(b.TotalRevenue / NULLIF(b.TotalBookings, 0), 2), 0) > 3000 THEN 'Premium'
    WHEN COALESCE(ROUND(b.TotalRevenue / NULLIF(b.TotalBookings, 0), 2), 0) BETWEEN 1000 AND 3000 THEN
'Standard'
    ELSE 'Economy'
  END AS PropertyTAG
FROM Property p
```

```
LEFT JOIN (
    SELECT
        b.PropertyID,
        SUM(b.TotalPrice) AS TotalRevenue,
        COUNT(b.BookingID) AS TotalBookings,
        SUM(DATEDIFF(b.CheckOutDate, b.CheckInDate)) AS TotalNights,
        MIN(b.BookingDate) AS FirstBookingDate
    FROM Booking b
    WHERE b.BookingStatus = 'Confirmed'
    GROUP BY b.PropertyID
) b ON p.PropertyID = b.PropertyID
LEFT JOIN (
    SELECT
        r.PropertyID,
        COUNT(r.ReviewID) AS TotalReviews
    FROM Review r
    GROUP BY r.PropertyID
) r ON p.PropertyID = r.PropertyID
GROUP BY p.PropertyID, p.Title, b.TotalRevenue, b.TotalBookings, r.TotalReviews, b.TotalNights, b.FirstBookingDate
ORDER BY TotalRevenue DESC;
-- ============================================================
```

-- View Name: UserSpending_FrequencyAnalysis

-- Description:

-- This view provides a detailed analysis of user spending and booking behavior.

It calculates key metrics related to monetary value, frequency of bookings,

and recency of user activity to assist in understanding customer segmentation

and behavior.

Purpose:

- To identify high-value, loyal, and active users based on their booking data.
- To help with user segmentation for marketing, loyalty programs, and retention efforts.
- To provide actionable insights into user activity trends.

Key Operations Performed:

Aggregates booking data to calculate:

* Total spending (`TotalSpending`): Sum of confirmed booking prices for each user.
* Total bookings (`TotalBookings`): Count of confirmed bookings.
* Average spending per booking (`AvgSpendingPerBooking`): Total spending divided by total bookings.
* Last booking date (`LastBookingDate`): Most recent booking for recency analysis.
* Booking frequency (`BookingFrequency`): Average time between consecutive bookings.

Categorizes users into the following groups:

* **RecencyLabel**: Based on the last booking date.

- `Recent`: Booked in the last 30 days.

- `Active`: Booked in the last 90 days.

- `Inactive`: No bookings in the last 90 days.

* **FrequencyLabel**: Based on the total number of bookings.

- `Loyal`: 10 or more bookings.

- `Frequent`: Between 5 and 9 bookings.

- `Rare`: Fewer than 5 bookings.

* **MonetaryLabel**: Based on total spending.

- `High Value`: Spending of 5000 or more.

- `Medium Value`: Spending between 1000 and 4999.

- `Low Value`: Spending under 1000.

- Ensures data accuracy and consistency by:

* Aggregating confirmed bookings in a subquery to avoid duplication.

* Using safe division (`NULLIF`) to prevent errors.

Benefits:

- Enables targeted marketing and customer engagement strategies.

- Provides data for loyalty programs and retention analysis.

- Identifies high-value customers for personalized offers and promotions.

==

```
CREATE OR REPLACE VIEW UserSpending_FrequencyAnalysis AS
SELECT
    u.UserID,
    u.Name AS UserName,
    COALESCE(b.TotalSpending, 0) AS TotalSpending,
    COALESCE(ROUND(b.TotalSpending / NULLIF(b.TotalBookings, 0), 2), 0) AS AvgSpendingPerBooking,
    b.LastBookingDate,
    COALESCE(b.TotalBookings, 0) AS TotalBookings,
    CASE
        WHEN b.TotalBookings > 1 THEN
            (SELECT ROUND(AVG(DATEDIFF(b2.BookingDate, b1.BookingDate)), 2)
             FROM Booking b1
             INNER JOIN Booking b2 ON b1.GuestID = b2.GuestID
             AND b2.BookingDate > b1.BookingDate
             WHERE b1.GuestID = u.UserID)
        ELSE NULL
    END AS BookingFrequency,
    CASE
        WHEN b.LastBookingDate >= DATE_SUB(CURDATE(), INTERVAL 30 DAY) THEN 'Recent'
        WHEN b.LastBookingDate >= DATE_SUB(CURDATE(), INTERVAL 90 DAY) THEN 'Active'
        ELSE 'Inactive'
    END AS RecencyLabel,
    CASE
```

```
        WHEN b.TotalBookings >= 10 THEN 'Loyal'

        WHEN b.TotalBookings BETWEEN 5 AND 9 THEN 'Frequent'

        ELSE 'Rare'

    END AS FrequencyLabel,

    CASE

        WHEN b.TotalSpending >= 5000 THEN 'High Value'

        WHEN b.TotalSpending BETWEEN 1000 AND 4999 THEN 'Medium Value'

        ELSE 'Low Value'

    END AS MonetaryLabel

FROM User u

LEFT JOIN (

    SELECT

        GuestID,

        SUM(TotalPrice) AS TotalSpending,

        COUNT(BookingID) AS TotalBookings,

        MAX(BookingDate) AS LastBookingDate

    FROM Booking

    WHERE BookingStatus = 'Confirmed'

    GROUP BY GuestID

) b ON u.UserID = b.GuestID

ORDER BY TotalSpending DESC;
```

==

View Name: BookingTimeAnalysis

Description:

This view provides a monthly analysis of bookings, revenue, and property trends.

It aggregates key metrics like the number of bookings, total revenue, average

Revenue per booking, and the count of unique properties booked for each year and month.

Purpose:

- To identify booking trends and peak periods for better resource allocation.
- To analyze revenue patterns across different months and years.
- To track property diversity in terms of unique properties booked.

Key Operations Performed:

- Aggregates booking data for confirmed bookings only.
- Calculates:
  * Total bookings (`TotalBookings`): Count of confirmed bookings per month.
  * Total revenue (`TotalRevenue`): Sum of total prices for all confirmed bookings.
  * Average revenue per booking (`AvgRevenuePerBooking`): Average price per booking.
  * Unique properties booked (`UniquePropertiesBooked`): Count of distinct properties booked.
- Groups data by year and month for monthly-level insights.
- Orders results by year and month in descending order for easy review of recent trends.

Benefits:

- Provides clear visibility into the busiest booking months and years.

- Offers insights into revenue generation trends for better financial planning.

- Tracks property utilization by analyzing unique properties booked each month.

==

```
CREATE OR REPLACE VIEW BookingTimeAnalysis AS
SELECT
  YEAR(b.CheckInDate) AS BookingYear,
  MONTH(b.CheckInDate) AS BookingMonth,
  COUNT(b.BookingID) AS TotalBookings,
  SUM(b.TotalPrice) AS TotalRevenue,
  ROUND(AVG(b.TotalPrice), 2) AS AvgRevenuePerBooking,
  COUNT(DISTINCT b.PropertyID) AS UniquePropertiesBooked
FROM Booking b
WHERE b.BookingStatus = 'Confirmed'
GROUP BY YEAR(b.CheckInDate), MONTH(b.CheckInDate)
ORDER BY BookingYear DESC, BookingMonth DESC;
```

==

View Name: PopularPropertyTypes

Description:

This view provides an analysis of the most popular property types based on bookings, revenue, ratings, and reviews. It offers insights into the performance of each property type.

Purpose:

- To identify the property types with the highest number of bookings.
- To analyze revenue trends across different property types.
- To measure customer satisfaction through average ratings and review counts.

Key Operations Performed:

- Aggregates data from bookings and reviews to calculate:
  * Total bookings (`TotalBookings`): Count of confirmed bookings for each property type.
  * Total revenue (`TotalRevenue`): Sum of total prices for confirmed bookings.
  * Average rating (`AverageRating`): Average of all ratings for each property type.
  * Total reviews (`TotalReviews`): Count of unique reviews for each property type.
- Uses `COALESCE` to handle NULL values for ratings and revenue.
- Groups data by property type and orders it by total bookings, revenue, and average ratings.

Benefits:

- Helps identify high-performing property types for marketing and pricing strategies.
- Tracks customer satisfaction through ratings and reviews.
- Enables better resource allocation for properties in high-demand categories.

==

```
CREATE OR REPLACE VIEW PopularPropertyTypes AS

SELECT

    p.PropertyType,
    COUNT(b.BookingID) AS TotalBookings,

    COALESCE(SUM(b.TotalPrice), 0) AS TotalRevenue,

    COALESCE(AVG(r.Rating), 0) AS AverageRating,

    COUNT(DISTINCT r.ReviewID) AS TotalReviews

FROM Property p

LEFT JOIN Booking b ON p.PropertyID = b.PropertyID AND b.BookingStatus = 'Confirmed'

LEFT JOIN Review r ON p.PropertyID = r.PropertyID

GROUP BY p.PropertyType

ORDER BY TotalBookings DESC, TotalRevenue DESC, AverageRating DESC;
```

==

View Name: HostEarningsAnalysis

Description:

This view provides an analysis of host performance based on earnings, bookings, and property management. It identifies top-performing hosts and tracks their activity status.

Purpose:

- To identify hosts generating the highest revenue and reward high-performing hosts.

- To analyze host performance based on property count and booking frequency and provide insights into host activity.

- To track host activity status (active or inactive) based on recent bookings, And help in identifying inactive hosts for re-engagement campaigns.

Key Operations Performed:

- Aggregates data from the `Property` and `Booking` tables to calculate:

  * Total properties managed (`TotalProperties`).

  * Total bookings made (`TotalBookings`).

  * Total revenue generated (`TotalRevenue`).

  * Average revenue per booking (`AvgRevenuePerBooking`).

  * Most recent booking date (`LastBookingDate`).

  * Activity status (`ActivityStatus`): Active if the host has bookings in the last 30 days.

- Groups data by host (user) and ranks them by total revenue.

- Ensures data consistency by using `COALESCE` for `NULL` handling.

==

```
CREATE OR REPLACE VIEW HostEarningsAnalysis AS

SELECT

  u.UserID AS HostID,

  u.Name AS HostName,

  COUNT(DISTINCT p.PropertyID) AS TotalProperties,

  COUNT(b.BookingID) AS TotalBookings,

  COALESCE(SUM(b.TotalPrice), 0) AS TotalRevenue,

  ROUND(COALESCE(SUM(b.TotalPrice), 0) / NULLIF(COUNT(b.BookingID), 0), 2) AS AvgRevenuePerBooking,

  MAX(b.CheckOutDate) AS LastBookingDate,

  CASE

    WHEN MAX(b.CheckOutDate) >= DATE_SUB(CURDATE(), INTERVAL 30 DAY) THEN 'Active'

    ELSE 'Inactive'
    END AS ActivityStatus
FROM User u
LEFT JOIN Property p ON u.UserID = p.HostID
LEFT JOIN Booking b ON p.PropertyID = b.PropertyID AND b.BookingStatus = 'Confirmed'
WHERE u.AccountType = 'Host'
GROUP BY u.UserID, u.Name
ORDER BY TotalRevenue DESC;
```

==

View Name: ReviewIntegrityCheck

Description:

This view identifies potentially fraudulent or suspicious reviews by analyzing booking and user activity data. It flags reviews based on booking status, timing, and user review frequency to help detect scam or biased reviews.

Purpose:

- To detect and flag suspicious reviews, such as:
    * Reviews submitted without a corresponding booking.
    * Reviews for canceled bookings.
    * Reviews submitted before the check-in date.
    * Reviews submitted long after the stay ended.
    * Users with unusually high review activity, indicating possible fake reviews.
- To enhance review integrity by identifying patterns and potential misuse.

Key Operations Performed:

- Validates the existence of a booking for each review:

  * Flags reviews without any booking (`Review Without Booking`).

  * Flags reviews for canceled bookings (`Review on Canceled Booking`).

- Analyzes the timing of reviews:

  * Flags reviews submitted before the stay started (`Review Before Stay`).

  * Flags reviews submitted more than 230 days after the stay ended (`Unusually Late Review`).

- Tracks user review activity:

  * Counts the total number of reviews per user to flag excessive activity (`Excessive Reviews by User`).

- Categorizes reviews with a `ReviewFlag` for easier identification and moderation.

  * `Valid`: if No red flags detected.

```
-- ==================================================================

CREATE OR REPLACE VIEW ReviewIntegrityCheck AS

SELECT

    r.ReviewID,

    r.PropertyID,

    r.ReviewerID,

    r.Rating,

    r.Comments,

    r.ReviewDate,

    b.CheckInDate,
    b.CheckOutDate,

    b.BookingStatus,

    u.DateJoined,

    (SELECT COUNT(*) FROM Review r2 WHERE r2.ReviewerID = r.ReviewerID) AS TotalReviewsByUser,

    CASE

        WHEN b.BookingID IS NULL THEN 'Review Without Booking'

        WHEN b.BookingStatus = 'Cancelled' THEN 'Review on Canceled Booking'

        WHEN r.ReviewDate < b.CheckInDate THEN 'Review Before Stay'

        WHEN DATEDIFF(r.ReviewDate, b.CheckOutDate) > 230 THEN 'Unusually Late Review'

        WHEN (SELECT COUNT(*) FROM Review r2 WHERE r2.ReviewerID = r.ReviewerID) > 10 THEN 'Excessive Reviews by
User'

        ELSE 'Valid'

    END AS ReviewFlag

FROM Review r

LEFT JOIN Booking b

    ON r.PropertyID = b.PropertyID

    AND r.ReviewerID = b.GuestID

LEFT JOIN User u

    ON r.ReviewerID = u.UserID

ORDER BY r.ReviewerID, r.ReviewDate;
```

==

View Name: BookingOverlapCheck

Description:

This view identifies overlapping bookings, analyzing two main scenarios:

1. Property Overlap: Two bookings for the same property with overlapping dates.

2. Guest Overlap: A guest having overlapping bookings across different properties.

Purpose:

- To detect and highlight scheduling conflicts:

  * Double bookings for the same property.

  * Guests booking multiple properties simultaneously.

- To help administrators resolve booking conflicts before they impact operations and improve guest experience and property management.

Key Operations Performed:

- **Property Overlap Detection**:

  * Compares bookings for the same property.

  * Checks if their check-in and check-out dates overlap.

- **Guest Overlap Detection**:

  * Compares bookings for the same guest across different properties.

  * Checks if their check-in and check-out dates overlap.

- Combines results for both scenarios into a single output using `UNION ALL`.

Notes:

- The view does not filter by booking status, so it includes all statuses (e.g., Pending, Confirmed, Cancelled).

- The `OverlapType` column distinguishes between property overlaps and guest overlaps.

-- ==

```
CREATE OR REPLACE VIEW BookingOverlapCheck AS

SELECT

    b1.BookingID AS Booking1ID,

    b2.BookingID AS Booking2ID,

    b1.PropertyID AS PropertyID1,

    b2.PropertyID AS PropertyID2,

    b1.GuestID AS GuestID,

    b1.CheckInDate AS Booking1CheckIn,

    b1.CheckOutDate AS Booking1CheckOut,

    b2.CheckInDate AS Booking2CheckIn,

    b2.CheckOutDate AS Booking2CheckOut,

    'Property Overlap' AS OverlapType

FROM Booking b1

JOIN Booking b2

    ON b1.PropertyID = b2.PropertyID

    AND b1.BookingID != b2.BookingID

    AND b1.CheckInDate < b2.CheckOutDate

    AND b1.CheckOutDate > b2.CheckInDate


UNION ALL


SELECT
```

```
    b1.BookingID AS Booking1ID,

    b2.BookingID AS Booking2ID,

    b1.PropertyID AS PropertyID1,

    b2.PropertyID AS PropertyID2,

    b1.GuestID AS GuestID,

    b1.CheckInDate AS Booking1CheckIn,

    b1.CheckOutDate AS Booking1CheckOut,

    b2.CheckInDate AS Booking2CheckIn,

    b2.CheckOutDate AS Booking2CheckOut,

    'Guest Overlap' AS OverlapType

FROM Booking b1

JOIN Booking b2

    ON b1.GuestID = b2.GuestID

    AND b1.PropertyID != b2.PropertyID

    AND b1.BookingID != b2.BookingID

    AND b1.CheckInDate < b2.CheckOutDate

    AND b1.CheckOutDate > b2.CheckInDate;
```