Room Rental Marketplace Database System





Part of a
Advanced Database – CMP7214

Student ID: 23206188
Student Name: Dinesh Thapa

Student ID: 22227184
Student Name: Arun Rajput

BIRMINGHAM CITY University

Table of Contents

Table of Figures

1. Introduction

A “room rental marketplace” is an online platform in which individuals and property owners can list and rent out rooms, apartments, or other types of accommodations to travelers, students, or anyone who is looking for a room for a short or long duration.

This is the ultimate solution for property owners who are looking for better tenants and guests who are seeking for better room for living.

Some Examples of Room Rental Marketplace Systems in the UK are Rightmove, SpareRoom, OpenRent, and RoomGo.

1.1 Domain Description

Our project is to create a “Complete Database System for Room Rental Marketplace” that keeps all the information safe and organized. This system is designed to store and manage the data of rental ecosystem like information of property owners, potential tenants, property lists like rooms and flats, user’s details, messages, notifications and many more.

This database system helps us to manage our marketplace platform’s data and make sure it works properly. This complete database system ensures the secure and seamless interactions, secure transactions, proper data management to make property management effective and easy.

2. Database Analysis

In our Rental Marketplace Database, it should store and manage the different information like Property details, Users information, Reviews, Messages, Notifications, Support Tickets, Subscriptions and Booking Request. We have to maintain the relationship between each information by implementing Database Relationships feature like Join based on Primary Keys and Foreign Keys. We have to maintain the integrity and efficiency of a database by using the right datatype and their constraints.

Our Database System must keep the information properly such as the process of listing a room or flat on the system, sending request for a room booking by guest, chats between listing owner and guest, subscription information and package details

of premium features, payment integration and details, reviews and ratings by users for listing owners, customer support for all the users in the system and notification system to notify them.

2.1 Business Rules

Our Business Rules includes following components: -

1. User Registration and Authentication
   a. Users must provide accurate information during registration.
   b. Each user should have a unique username and email address.
   c. Users must set a secure password and can reset it if forgotten.
   d. Only registered and authenticated users can access the platform.

2. User Profiles
   a. Users can create and manage their profiles, including personal information, contact details, payment options. This information helps to identify the users of the system properly.

3. User Roles
   a. All users in a system can create a listing for the property. Property can be his/her own (I own the property) or living there in the property (I am living in the property) or Agent (I am listing on a landlord's behalf).
   b. User cannot be able to send request to his own property listed on the platform.
   c. There are two main user roles:
      i. Admin (System Admin)
      ii. Users (Property Owner/Guest)

4. Property Listings
   a. Property Owners or individuals can create detailed listings for the rooms/flats/apartments or accommodation they want to rent.
   b. This listing typically includes property descriptions, property photos, location, pricing, availability, and any property rules or requirements.

5. Search and Filters
   a. Guests can search for available properties based on criteria such as location, price range, property type, number of guests, and specific amenities.

7. Reviews and Ratings

a. Both property/listing owner and guest/individual/tenant can give reviews and rating to each other. This feedback system establishes trust and transparency within the community.

b. The system should have a mechanism to report and moderate inappropriate content or reviews.

8. Messaging and Communication

   a. Messaging feature enables property owners to communicate directly with guests to discuss booking details, ask questions, and clarify expectations.

9. Property/Listing Owner Identity Verification

   a. To enhance trust and safety, platforms often offer property owner/listing owner identity verification, background checks, or references.

10. Customer Support

a. The platform should offer customers support to assist users with issues, concerns or disputes. Users can create issue tickets on the platform.

11. Notifications

   a. Users should receive timely notifications, such as should notify property owner when guest send request, notify guest when property owner responds/reply to his/her message, reminders, messages through notifications.

   b. Users should receive subscription confirmation, subscription reminders notifications too.

12. Subscription Types

   a. Gold Subscription

   Gold Subscription is a free subscription. Users need to register on the system to access this service.

   b. Diamond Subscription

      a. Diamond Subscription is a paid subscription. Users need to pay on the system to access this service.

         There will be 3 payment options for diamond subscription:

         1. 15 Days Subscription: 20 Pound
         2. 30 Days Subscription: 35 Pound
         3. 1 Year Subscription: 200 Pound

13. Gold Subscription (Free Subscription)

a. A user can list up to only 2 listings or advertisements per month.
b. A user can view all the listings but can only send requests to those which are 7+ days old.
c. You may see some sort of third-party ads on the platform.

14. Diamond Subscription (Paid Subscription)
    a. A Diamond Subscribed user will have diamond symbol after their name on the profile. It increases trustworthiness and transparency in the system.
    b. A user can list unlimited listings or advertisements. Ads that are listed by diamond subscribers will get bold visibility in the platform. It is called Bold Listing. Bold Listing will be visible to people with ease.
    a. A user can access all the instant/newly posted ads say, early bird access to the room listing.
    b. No third-party ads on the platform.
    c. Your bold listing will get on 3x as visits as compared to normal listing.

15. Payment Plan
    a. A user can add payment details on the platform after choosing an appropriate subscription and pay securely through online payment gateway system like Visa Card/Master Card or PayPal or Google Pay.

2.2 Business Constraints

Based on the above business rules, we can create a set of database business constraints to ensure data accuracy, security, and proper functionality of the room rental marketplace platform.

1. **User Registration and Authentication:**
    - Uniqueness Constraint: Enforce uniqueness for usernames and email addresses to prevent multiple users from registering with the same information.

2. **User Profiles:**
    - Completeness Constraint: Ensure that all user profiles contain complete and accurate information, including personal details, contact information, and payment options.

3. **User Roles:**
    - Validity Constraint: Users must be provided a valid user role from the system user roles options: "Admin," "User".

- User-Property Relationship Constraint: Users cannot send booking requests or interact with properties they own. The system must prevent a user from booking their own property.

4. **Property Listings:**
   - Completeness Constraint: Property listings must contain complete and accurate information, including property descriptions, photos, location, pricing, availability, and any rules or requirements.

5. **Search and Filters:**
   - Search Validity Constraint: Property search and filtering criteria should be limited to valid options, such as location, price range, property type, and amenities.

6. **Booking:**
   - Request Validity Constraint: Users can only request book properties, not other types of interactions (e.g., sending messages).

7. **Reviews and Ratings:**
   - Feedback Validity Constraint: The system should validate reviews and ratings, ensuring they come from legitimate users and adhere to content guidelines.
   - Moderation Constraint: Implement a mechanism to allow users to report and moderate inappropriate content, reviews, or ratings.

8. **Messaging and Communication:**
   - Secure Communication Constraint: Ensure secure messaging and communication channels to protect user privacy and data.

9. **User Identity Verification:**
   - Verification Constraint: If identity verification and background checks are implemented, ensure that verified information is securely stored and linked to the respective users.

10. **Customer Support**
    - Issue Tracking Constraint: Users can create issue tickets for customer support, and the database should store these tickets for tracking and resolution.

11. **Notifications:**

- Notification Delivery Constraint: Implement a notification system that delivers timely notifications to users based on specific events and interactions.

12.Subscription Types:

- Subscription Constraint: Users must automatically be assigned with "Gold Subscription" (free) or "Diamond Subscription" (paid) if they choose this.

13.Gold Subscription:

- Listing Limit Constraint: Users with "Gold Subscription" can list up to a maximum of 2 listings per month.
- Request Constraint: "Gold Subscription" users can only send requests to property listings that are at least 7 days old.

14.Diamond Subscription:

- Bold Listing Constraint: "Diamond Subscription" users can create bold listings for higher visibility, and the database should manage this visibility.
- Early Access Constraint: "Diamond Subscription" users have early access to newly posted ads.
- Third-Party Ads Constraint: Ensure that no third-party ads are displayed on the platform for "Diamond Subscription" users.

15.Payment Plan:

- Payment Details Constraint: Users must provide valid payment details, and the database should securely handle payment processing through the chosen online payment gateway systems.

2.3 List of Entity/Attributes

Entity 1: Users

- Attributes: ***user_id***, *username, email, password, firstname, lastname, phone, address, user_type, created_at, updated_at, deleted_at*

Entity 2: UserProfiles

- Attributes: ***profile_id***, *user_id, dob, gender, photo, occupation, about_me, lang_pref, smoking_pref, notify_enabled, last_login_at, account_status, id_verify_status, created_at, updated_at, deleted_at*

Entity 3: Properties

- Attributes: ***property_id***, *user_id, title, des, location, price, availability, rules, property_type, created_at, updated_at, deleted_at*

Entity 4: Reviews

- Attributes: ***review_id***, *user_id, rating, comment, created_at, updated_at, deleted_at*

Entity 5: Messages

- Attributes: ***message_id***, *sender_user_id, receiver_user_id, message, created_at, updated_at, deleted_at*

Entity 6: Notifications

- Attributes: ***notification_id***, *user_id, content, created_at, updated_at, deleted_at*

Entity 7: subs_type

- Attributes: ***subs_id***, *name, no_of_days, fee, created_at, updated_at, deleted_at*

Entity 8: IssueTickets

- Attributes: ***ticket_id***, *user_id, content, status, created_at, updated_at, deleted_at*

Entity 9: PaymentDetails

- Attributes: ***payment_details_id***, *user_id, payment_option_id, card_number, expiry_date, cvv, created_at, updated_at, deleted_at*

Entity 10: SubsTransactions

- Attributes: ***transaction_id,*** *user_id, subs_type_id, payment_details_id, start_date, end_date, created_at, updated_at, deleted_at*

2.4 Simple Relationships

Users 1 <has> 1 [UserProfiles]

Users 1 <owns> M [Properties]

Users 1 <writes> M [Reviews]

Users M <sends> M [Messages]

Users M <raises> M [IssueTickets]

Users 1 <has> M [PaymentDetails]

Users M <performs> M [SubsTransactions]

UserProfiles M <sends> M [Messages]

UserProfiles M <receives> M [Notifications]

UserProfiles M <raises> M [IssueTickets]

2.5 Connectivity, Cardinalities and Participation

A USER has a minimum of __1__ USERPROFILE

A USER has a maximum of __1__ USERPROFILE

Reverse:

A USERPROFILE is associated with a minimum of __1__ USER

A USERPROFILE is associated with a maximum of __1__ USER

A USER owns a minimum of __0__ PROPERTY

A USER owns a maximum of __M__ PROPERTIES

Reverse:

A PROPERTY is owned by a minimum of __1__ USER

A PROPERTY is owned by a maximum of __1__ USER

A USER writes a minimum of __0__ REVIEW

A USER writes a maximum of __M__ REVIEWS

Reverse:

A REVIEW is written by a minimum of __1__ USER

A REVIEW is written by a maximum of __1__ USER

A USER sends a minimum of __0__ MESSAGE

A USER sends a maximum of __M__ MESSAGES

A USER receives a minimum of __0__ MESSAGE

A USER receives a maximum of __M__ MESSAGES

Reverse:

A MESSAGE is sent by a minimum of __1__ USER

A MESSAGE is sent by a maximum of __1__ USER

A MESSAGE is received by a minimum of __1__ USER

A MESSAGE is received by a maximum of __1__ USER

A USER receives a minimum of __0__ NOTIFICATION

A USER receives a maximum of __M__ NOTIFICATIONS

Reverse:

A NOTIFICATION is received by a minimum of __1__ USER

A NOTIFICATION is received by a maximum of __1__ USER

A USER raises a minimum of __0__ ISSUETICKET

A USER raises a maximum of __M__ ISSUETICKETS

Reverse:

An ISSUETICKET is raised by a minimum of __1__ USER

An ISSUETICKET is raised by a maximum of __1__ USER

A USER has a minimum of __0__ PAYMENTDETAIL

A USER has a maximum of __M__ PAYMENTDETAILS

Reverse:

A PAYMENTDETAIL is associated with a minimum of __1__ USER

A PAYMENTDETAIL is associated with a maximum of __1__ USER

A USER performs a minimum of __0__ SUBSTRANSACTION

A USER performs a maximum of __M__ SUBSTRANSACTIONS

Reverse:

A SUBSTRANSACTION is performed by a minimum of __1__ USER

A SUBSTRANSACTION is performed by a maximum of __1__ USER

A PROPERTY has a minimum of __0__ REVIEW

A PROPERTY has a maximum of __M__ REVIEWS

Reverse:

A REVIEW is associated with a minimum of __1__ PROPERTY

A REVIEW is associated with a maximum of __1__ PROPERTY

2.6 Normalization

Users Table Normalization

user_id, *username, email, password, firstname, lastname, phone, address, user_type, created_at, updated_at, deleted_at*

First Normal Form:

1NF ensures that each column contains atomic values.

In our case, the table is already in 1NF since each cell contains a single volume and there are no repeating groups.

Second Normal Form:

2NF requires 1NF and ensures that all columns are fully dependent on the primary key.

In our case, “Users” table seems to meet the condition of 2NF since there are no partial dependency in the primary key.

Third Normal Form:

3NF requires meeting 2NF and ensures that no non-primary key column is transitively dependent on the primary key.

In our case, “Users” table meet the requirement of 3NF too.

Fourth Normal Form:

4NF deals with multi-valued dependencies.

In our case, there seems to be no any multi-valued dependencies. Thus, it meets the condition of 4NF.

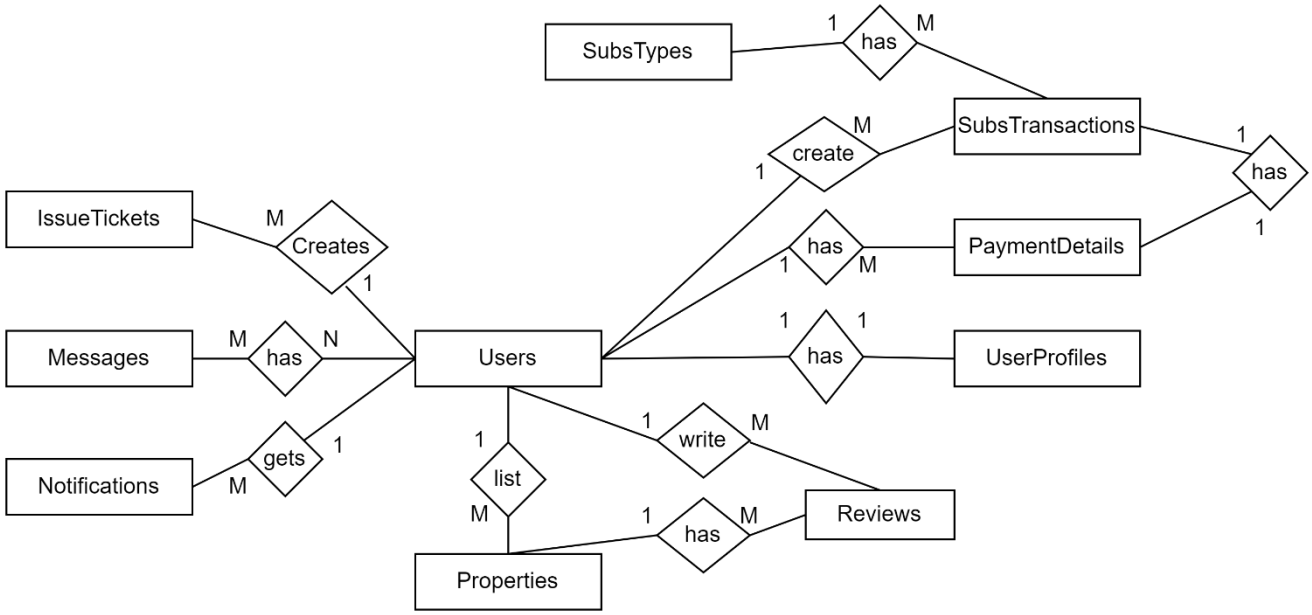
Fifth Normal Form:

5NF deals with cases where a table has joined dependencies.

In our case, our table is already in 5NF.

Our all 10 tables are already in normalized form. So, Normalization analysis is not necessary at this time.

3. Database Design: ER Diagram



3.1 Entities and Relationship

Figure 1: Entitles and Relationships

3.2 ER Diagram

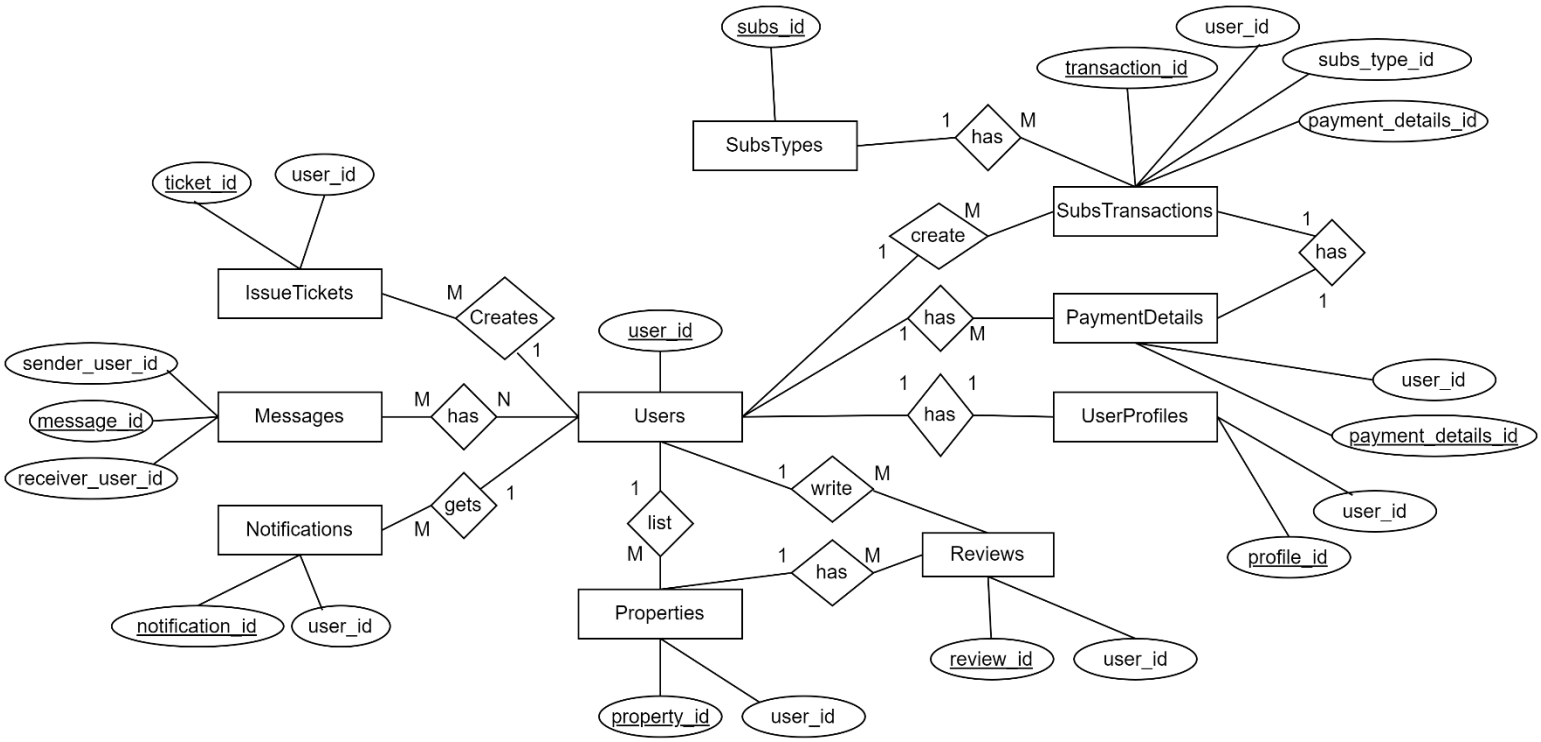


Figure 2: ER Diagram

In this ER Diagram, we have only included primary and foreign attributes because adding other attributes makes it more complex to read. As we have already provided information of all other attributes in the above lists of entities and attributes.

In the above diagram, Primary keys are underlined and all other remaining attributes are foreign keys.

4. Database Implementation – Table Creation

The following codes will create tables in the MySQL database management system.

Done By Arun Rajput - 22227184

4.1 Users

```
CREATE TABLE Users (

    user_id INT AUTO_INCREMENT PRIMARY KEY,

    username VARCHAR(50) UNIQUE,

    email VARCHAR(100) UNIQUE,

    password VARCHAR(255),

    firstname VARCHAR(50),

    lastname VARCHAR(50),

    phone VARCHAR(20),

    address TEXT,

    user_type ENUM('Admin', 'User'),

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    deleted_at TIMESTAMP DEFAULT NULL

);
```

4.2 UserProfiles

```
CREATE TABLE UserProfiles (

    profile_id INT AUTO_INCREMENT PRIMARY KEY,

    user_id INT,

    dob DATE,

    gender ENUM('male', 'female', 'other'),

    photo VARCHAR(255),

    occupation VARCHAR(100),

    about_me TEXT,

    lang_pref VARCHAR(50),
```

```
    smoking_pref VARCHAR(50),
    notify_enabled BOOLEAN,
    last_login_at TIMESTAMP,
    account_status ENUM('active', 'suspended', 'banned'),
    id_verify_status ENUM('Pending', 'Verified', 'Rejected'),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    deleted_at TIMESTAMP DEFAULT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

4.3 Properties

```
CREATE TABLE Properties (
    property_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    title VARCHAR(255),
    des LONGTEXT,
    location TEXT,
    price DOUBLE,
    availability INT,
    rules TEXT,
    property_type INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    deleted_at TIMESTAMP DEFAULT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

4.4 Reviews

```
CREATE TABLE Reviews (
```

```
    review_id INT AUTO_INCREMENT PRIMARY KEY,

    user_id INT,

    rating INT,

    comment TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    deleted_at TIMESTAMP DEFAULT NULL,

    FOREIGN KEY (user_id) REFERENCES Users(user_id)

);
```

4.5 Messages

```
CREATE TABLE Messages (

    message_id INT AUTO_INCREMENT PRIMARY KEY,

    sender_user_id INT,

    receiver_user_id INT,

    message TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    deleted_at TIMESTAMP DEFAULT NULL,

    FOREIGN KEY (sender_user_id) REFERENCES Users(user_id),

    FOREIGN KEY (receiver_user_id) REFERENCES Users(user_id)

);
```

Done By Dinesh Thapa - 23206188

4.6 Notifications

```
CREATE TABLE Notifications (

    notification_id INT AUTO_INCREMENT PRIMARY KEY,

    user_id INT,

    content TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    deleted_at TIMESTAMP DEFAULT NULL,

    FOREIGN KEY (user_id) REFERENCES Users(user_id)

);
```

4.7 subs_type

```
CREATE TABLE Subs_type (

    subs_id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(255),

    no_of_days INT,

    fee DOUBLE,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    deleted_at TIMESTAMP DEFAULT NULL

);
```

4.8 IssueTickets

```
CREATE TABLE IssueTickets (

    ticket_id INT AUTO_INCREMENT PRIMARY KEY,

    user_id INT,

    content VARCHAR(255),

    status INT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    deleted_at TIMESTAMP DEFAULT NULL,

    FOREIGN KEY (user_id) REFERENCES Users(user_id)

);
```

4.9 PaymentDetails

```
CREATE TABLE PaymentDetails (

    payment_details_id INT PRIMARY KEY,
```

```
    user_id INT,
    payment_option_id INT,
    card_number VARCHAR(20),
    expiry_date DATE,
    cvv VARCHAR(4),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    deleted_at TIMESTAMP DEFAULT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

4.10 SubsTransactions

```
CREATE TABLE SubsTransactions (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    subs_type_id INT,
    payment_details_id INT,
    start_date DATETIME,
    end_date DATETIME,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    deleted_at TIMESTAMP DEFAULT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (subs_type_id) REFERENCES Subs_types(subs_id),
    FOREIGN KEY (payment_details_id) REFERENCES PaymentDetails(payment_id)
);
```

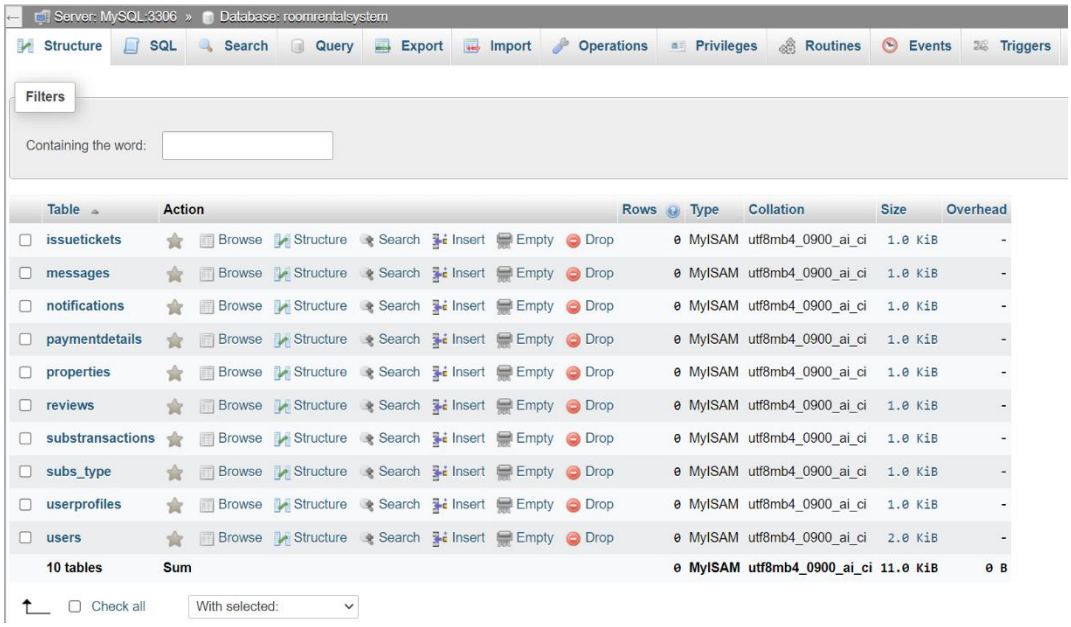


Figure 3: Results after all Tables Creation is done

5. Database Implementation

5.1 Data Insertion - Getting Data into Tables

<u>Done By Arun Rajput - 22227184</u>

a. Inserting Demo Data into Users Table

```
INSERT INTO Users (username, email, password, firstname, lastname, phone,
address, user_type)

VALUES

('john_doe', 'john@example.com', 'pass123', 'John', 'Doe', '1234567890', '123
Main St', 'User'),

('jane_smith', 'jane@example.com', 'pass456', 'Jane', 'Smith', '9876543210',
'456 Elm St', 'User'),

('admin1', 'admin1@example.com', 'adminpass', 'Admin', 'One', '1111111111',
'789 Oak St', 'Admin'),
```

```
('user3', 'user3@example.com', 'userpass', 'User', 'Three', '2222222222', '567
Pine St', 'User'),

('sam_jones', 'sam@example.com', 'sam123', 'Sam', 'Jones', '3333333333', '890
Cedar St', 'User'),

('sara_williams', 'sara@example.com', 'sara456', 'Sara', 'Williams',
'4444444444', '234 Birch St', 'User'),

('admin2', 'admin2@example.com', 'adminpass2', 'Admin', 'Two', '5555555555',
'678 Maple St', 'Admin'),

('user4', 'user4@example.com', 'userpass4', 'User', 'Four', '6666666666', '432
Walnut St', 'User'),

('alice_parker', 'alice@example.com', 'alice789', 'Alice', 'Parker',
'7777777777', '876 Cherry St', 'User'),

('admin3', 'admin3@example.com', 'adminpass3', 'Admin', 'Three', '8888888888',
'543 Oak St', 'Admin');
```

b. Inserting Demo Data into UserProfiles Table

```
INSERT INTO UserProfiles (user_id, dob, gender, photo, occupation, about_me,
lang_pref, smoking_pref, notify_enabled, last_login_at, account_status,
id_verify_status)

VALUES

(1, '1990-05-15', 'male', 'https://example.com/photo1.jpg', 'Engineer', 'I
love coding.', 'English', 'No', true, '2023-12-01 08:30:00', 'active',
'Verified'),

(2, '1985-08-22', 'female', 'https://example.com/photo2.jpg', 'Architect',
'Passionate about design.', 'French', 'No', true, '2023-12-02 10:15:00',
'active', 'Verified'),

(3, '1995-03-10', 'other', 'https://example.com/photo3.jpg', 'Artist',
'Expressing through art.', 'Spanish', 'Yes', true, '2023-12-03 12:45:00',
'active', 'Pending'),

(4, '1988-11-28', 'male', 'https://example.com/photo4.jpg', 'Writer', 'Words
create magic.', 'English', 'No', true, '2023-12-04 15:20:00', 'active',
'Verified'),

(5, '1992-07-17', 'female', 'https://example.com/photo5.jpg', 'Doctor',
'Healing lives.', 'English', 'No', true, '2023-12-05 17:55:00', 'active',
'Verified'),
```

```
(6, '1980-04-25', 'male', 'https://example.com/photo6.jpg', 'Chef', 'Cooking
is an art.', 'Italian', 'No', true, '2023-12-06 20:10:00', 'active',
'Verified'),

(7, '1998-09-03', 'female', 'https://example.com/photo7.jpg', 'Athlete',
'Determined to win.', 'English', 'No', true, '2023-12-07 22:30:00', 'active',
'Verified'),

(8, '1993-12-12', 'other', 'https://example.com/photo8.jpg', 'Musician',
'Creating melodies.', 'English', 'Yes', true, '2023-12-08 09:45:00', 'active',
'Verified'),

(9, '1983-06-30', 'male', 'https://example.com/photo9.jpg', 'Entrepreneur',
'Building dreams.', 'English', 'No', true, '2023-12-09 11:20:00', 'active',
'Verified'),

(10, '1996-01-05', 'female', 'https://example.com/photo10.jpg', 'Designer',
'Crafting innovation.', 'English', 'No', true, '2023-12-10 13:40:00',
'active', 'Verified');
```

c. Inserting Demo Data into Properties Table

```
INSERT INTO Properties (user_id, title, des, location, price, availability,
rules, property_type)

VALUES

(1, 'Cozy Apartment', 'A lovely apartment with a view', 'City Center',
1200.00, 1, 'No smoking, no pets allowed', 1),

(2, 'Modern Condo', 'Spacious condo in the heart of the city', 'Downtown',
1800.00, 1, 'No loud parties after 10 PM', 2),

(3, 'Luxury Villa', 'Beautiful villa with a pool', 'Suburbia', 2500.00, 1,
'Pets allowed with additional fee', 3),

(4, 'Rustic Cabin', 'Cozy cabin in the woods', 'Forest Area', 800.00, 1, 'No
smoking indoors', 4),

(5, 'Beach House', 'Amazing beachfront property', 'Seaside', 3000.00, 1,
'Beach access included', 5),

(6, 'Mountain Chalet', 'Charming chalet with mountain views', 'Mountain
Range', 1500.00, 1, 'Ski-in/ski-out', 6),

(7, 'Country Farmhouse', 'Quaint farmhouse in the countryside', 'Rural Area',
1000.00, 1, 'Farm experience available', 7),

(8, 'Urban Loft', 'Trendy loft in the city', 'Metropolis', 2000.00, 1, 'Close
to public transportation', 8),
```

```
(9, 'Historic Mansion', 'Elegant mansion with historical charm', 'Old Town',
3500.00, 1, 'Preservation rules apply', 9),

(10, 'Family Home', 'Spacious family-friendly home', 'Suburban Neighborhood',
1700.00, 1, 'Close to schools and parks', 10);
```

d. Inserting Demo Data into Reviews Table

```
INSERT INTO Reviews (user_id, rating, comment)

VALUES

(1, 5, 'Great service and excellent experience!'),

(2, 4, 'Very satisfied with the product.'),

(3, 3, 'Average service, could be better.'),

(4, 5, 'Highly recommended! Will come back again.'),

(5, 2, 'Disappointed with the quality.'),

(6, 4, 'Delicious food and friendly staff.'),

(7, 5, 'Fantastic performance!'),

(8, 3, 'Good service but needs improvement.'),

(9, 5, 'Amazing customer support!'),

(10, 4, 'Beautifully designed and well-made.');
```

e. Inserting Demo Data into Messages Table

```
INSERT INTO Messages (sender_user_id, receiver_user_id, message)

VALUES

(1, 2, 'Regarding the rental property, are you available for a viewing this
week?'),

(2, 1, 'Yes, I am available. Could we schedule it for Thursday afternoon?'),

(3, 4, 'I''m interested in renting the property. Could you provide more
details about the amenities?'),

(4, 3, 'Certainly! The property has a swimming pool, gym, and parking
space.'),

(5, 6, 'I''d like to discuss the rental terms before finalizing. Can we meet
to go over the contract?'),

(6, 5, 'Sure, let''s meet at the property tomorrow to discuss the terms.'),
```

```
(7, 8, 'Any updates on the rental availability?'),
(8, 7, 'Yes, the property is available for rent starting next month.'),
(9, 10, 'I got approved for the rental application!'),
(10, 9, 'That''s great news! Congratulations!');
```

Done By Dinesh Thapa - 23206188

f. Inserting Demo Data into Notifications Table

```
INSERT INTO Notifications (user_id, content)
VALUES
(1, 'New rental property available! Check it out.'),
(2, 'You have a rental inquiry. Respond soon.'),
(3, 'Reminder: Property viewing scheduled tomorrow at 10 AM.'),
(4, 'Your rental application status has been updated.'),
(5, 'New rental feature released! Explore now.'),
(6, 'Rental payment received for your property.'),
(7, 'Important update: Scheduled maintenance for rental system tonight.'),
(8, 'Your rental subscription expires soon. Renew now.'),
(9, 'Congratulations on renting your property!'),
(10, 'Welcome! Start renting on our platform!');
```

g. Inserting Demo Data into Subs_type Table

```
INSERT INTO Subs_type (name, no_of_days, fee)
VALUES
('Gold', 0, 0.00), -- Gold Subscription is a free subscription.
('Diamond - 15 Days', 15, 20.00), -- 15 Days Subscription: 20 Pound
('Diamond - 30 Days', 30, 35.00), -- 30 Days Subscription: 35 Pound
('Diamond - 1 Year', 365, 200.00); -- 1 Year Subscription: 200 Pound
```

h. Inserting Demo Data into IssueTickets Table

```
INSERT INTO IssueTickets (user_id, content, status)
```

```
VALUES
(1, 'Having trouble logging in.', 1),
(2, 'Issue with the payment gateway.', 1),
(3, 'Request for additional features.', 2),
(4, 'Reporting a bug in the system.', 2),
(5, 'Query about subscription renewal.', 1),
(6, 'Request for account recovery.', 1),
(7, 'Complaint regarding service downtime.', 2),
(8, 'Suggestion for improving user interface.', 2),
(9, 'Question about product functionality.', 1),
(10, 'Seeking assistance with account settings.', 1);
```

i. Inserting Demo Data into PaymentDetails Table

```
INSERT INTO PaymentDetails
    (payment_details_id, user_id, payment_option_id, card_number, expiry_date,
cvv)
VALUES
    (1, 1, 1, '1234567890123456', '2024-12-31', '123'),
    (2, 2, 2, '9876543210987654', '2023-10-31', '456'),
    (3, 3, 1, '1111222233334444', '2025-08-31', '789'),
    (4, 4, 3, '5555666677778888', '2024-09-30', '321'),
    (5, 5, 2, '9999000011112222', '2023-07-31', '654'),
    (6, 6, 1, '4444333322221111', '2025-05-31', '987'),
    (7, 7, 2, '8888999977775555', '2024-03-31', '123'),
    (8, 8, 3, '6666555544443333', '2023-12-31', '456'),
    (9, 9, 1, '1212121212121212', '2025-11-30', '789'),
    (10, 10, 2, '1313131313131313', '2024-06-30', '321');
```

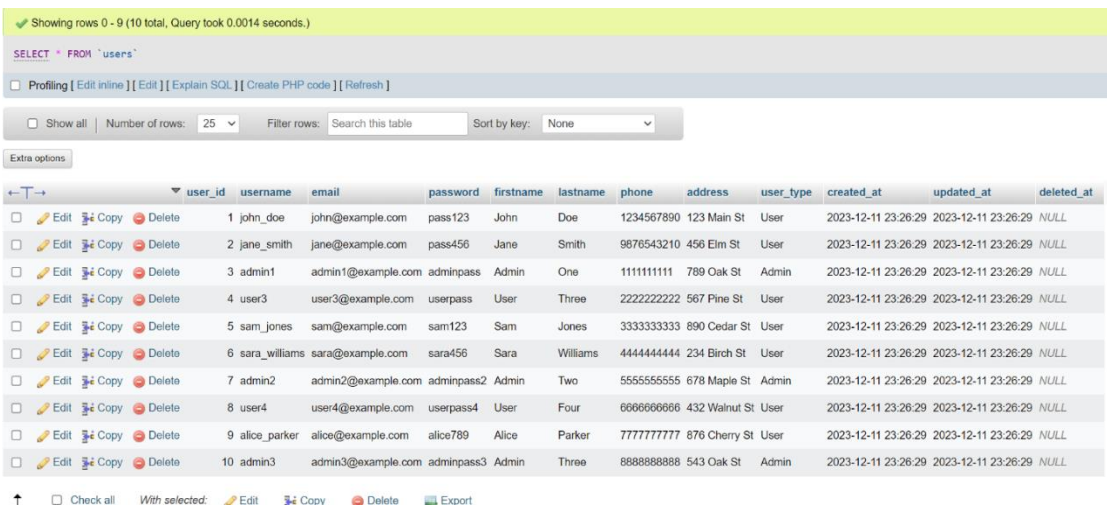
j. Inserting Demo Data into SubsTransactions Table

```
INSERT INTO SubsTransactions (user_id, subs_type_id, payment_details_id,
start_date, end_date)
VALUES
```

```
(1, 1, 1, '2023-12-11 12:00:00', '2024-01-10 12:00:00'),
(2, 2, 2, '2023-12-11 12:00:00', '2024-02-09 12:00:00'),
(3, 3, 3, '2023-12-11 12:00:00', '2024-03-11 12:00:00'),
(4, 4, 4, '2023-12-11 12:00:00', '2024-01-10 12:00:00'),
(5, 1, 5, '2023-12-11 12:00:00', '2024-03-11 12:00:00'),
(6, 2, 6, '2023-12-11 12:00:00', '2023-12-26 12:00:00'),
(7, 3, 7, '2023-12-11 12:00:00', '2024-01-25 12:00:00'),
(8, 4, 8, '2023-12-11 12:00:00', '2024-02-09 12:00:00'),
(9, 4, 9, '2023-12-11 12:00:00', '2024-03-11 12:00:00'),
(10, 4, 10, '2023-12-11 12:00:00', '2023-12-18 12:00:00');
```

After demo records are inserted into tables, let's see how it looks:

a. **Users Table**



Showing rows 0 - 9 (10 total, Query took 0.0014 seconds.)

SELECT * FROM `users`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

←T→	user_id	username	email	password	firstname	lastname	phone	address	user_type	created_at	updated_at	deleted_at
Edit Copy Delete	1	john_doe	john@example.com	pass123	John	Doe	1234567890	123 Main St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	2	jane_smith	jane@example.com	pass456	Jane	Smith	9876543210	456 Elm St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	3	admin1	admin1@example.com	adminpass	Admin	One	1111111111	789 Oak St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	4	user3	user3@example.com	userpass	User	Three	2222222222	567 Pine St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	5	sam_jones	sam@example.com	sam123	Sam	Jones	3333333333	890 Cedar St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	6	sara_williams	sara@example.com	sara456	Sara	Williams	4444444444	234 Birch St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	7	admin2	admin2@example.com	adminpass2	Admin	Two	5555555555	678 Maple St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	8	user4	user4@example.com	userpass4	User	Four	6666666666	432 Walnut St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	9	alice_parker	alice@example.com	alice789	Alice	Parker	7777777777	876 Cherry St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	10	admin3	admin3@example.com	adminpass3	Admin	Three	8888888888	543 Oak St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL

Check all With selected: Edit Copy Delete Export

Figure 4: Users Table Demo Data Insertion

b. **UsersProfiles Table**

Showing rows 0 - 9 (10 total, Query took 0.0017 seconds.)

```
SELECT * FROM `userprofiles`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

←T→	profile_id	user_id	dob	gender	photo	occupation	about_me	lang_pref	smoking_pref	notify_enabled	last_login_at	account_status	id_verify_s
Edit Copy Delete	1	1	1990-05-15	male	https://example.com/photo1.jpg	Engineer	I love coding.	English	No	1	2023-12-01 08:30:00	active	Verified
Edit Copy Delete	2	2	1985-08-22	female	https://example.com/photo2.jpg	Architect	Passionate about design.	French	No	1	2023-12-02 10:15:00	active	Verified
Edit Copy Delete	3	3	1995-03-10	other	https://example.com/photo3.jpg	Artist	Expressing through art.	Spanish	Yes	1	2023-12-03 12:45:00	active	Pending
Edit Copy Delete	4	4	1988-11-28	male	https://example.com/photo4.jpg	Writer	Words create magic.	English	No	1	2023-12-04 15:20:00	active	Verified
Edit Copy Delete	5	5	1992-07-17	female	https://example.com/photo5.jpg	Doctor	Healing lives.	English	No	1	2023-12-05 17:55:00	active	Verified
Edit Copy Delete	6	6	1980-04-25	male	https://example.com/photo6.jpg	Chef	Cooking is an art.	Italian	No	1	2023-12-06 20:10:00	active	Verified
Edit Copy Delete	7	7	1998-09-03	female	https://example.com/photo7.jpg	Athlete	Determined to win.	English	No	1	2023-12-07 22:30:00	active	Verified
Edit Copy Delete	8	8	1993-12-12	other	https://example.com/photo8.jpg	Musician	Creating melodies.	English	Yes	1	2023-12-08 09:45:00	active	Verified
Edit Copy Delete	9	9	1983-06-30	male	https://example.com/photo9.jpg	Entrepreneur	Building dreams.	English	No	1	2023-12-09 11:20:00	active	Verified
Edit Copy Delete	10	10	1996-01-05	female	https://example.com/photo10.jpg	Designer	Crafting innovation.	English	No	1	2023-12-10 13:40:00	active	Verified

Figure 5: UsersProfiles Table Demo Data Insertion

c. PaymentDetails Table

```
SELECT * FROM `paymentdetails`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

←T→	payment_details_id	user_id	payment_option_id	card_number	expiry_date	cvv	created_at	updated_at	deleted_at
Edit Copy Delete	1	101	1	1234567890123456	2024-12-31	123	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	2	102	2	9876543210987654	2023-10-31	456	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	3	103	1	1111222233334444	2025-08-31	789	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	4	104	3	5555666677778888	2024-09-30	321	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	5	105	2	9999000011112222	2023-07-31	654	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	6	106	1	4444333322221111	2025-05-31	987	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	7	107	2	8888999977775555	2024-03-31	123	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	8	108	3	6666555544443333	2023-12-31	456	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	9	109	1	1212121212121212	2025-11-30	789	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL
Edit Copy Delete	10	110	2	1313131313131313	2024-06-30	321	2023-12-12 00:51:05	2023-12-12 00:51:05	NULL

Figure 6: PaymentDetails Table Demo Data Insertion

These above three screenshots show the demo data records that we have inserted into tables.

5.2 Constraints Tests

Done By Dinesh Thapa - 23206188

a. Check Constraints Test in Users Table

Attributes: ***user_id***, *username, email, password, firstname, lastname, phone, address, user_type, created_at, updated_at, deleted_at*

In users table, we have column named "user_type" which is created to handle two types of users in the system. In this column, we have declared user_type ENUM('Admin', 'User'). This means that this column can have only two values either Admin or User. But, when I try to insert other values than Admin or User, this is inserting that value. This is semantics error in our database design.
So, to handle this issue, we can implement Check Constraint while creating a table as follows:

```
CONSTRAINT chk_user_type CHECK (user_type IN ('Admin', 'User'))
```

This line of code will check and let us enter only the value that is either "Admin" or "User".
As I already have a table called users on my database, I am going to use ALTER TABLE command to perform the following operation:

```
ALTER TABLE Users

ADD CONSTRAINT chk_user_type CHECK (user_type IN ('Admin', 'User'));
```

After implementing this in “users” table, when we try to insert other value other than Admin or User, then it will show this error:

Figure 7: After Implementation of Check Constraint

<u>Done By Arun Rajput - 22227184</u>

b. Default Constraint Test in notifications table

Attributes: ***<u>notification_id</u>***, *user_id, content, created_at, updated_at, deleted_at*

Here, In notifications table, created_at columns keep the data of when this notification is created. But, to store this data automatically, we have used default constraint so that it stores current_timestamp (current date and time information) in that column automatically while inserting new record. This happens by default which is done by implementing default constraint. In simple terms, if we do not send any value to the table, it will give that default value in the column.

Let’s test this with simple SQL query.

```
INSERT INTO notifications (user_id, content) VALUES (2, “Your new property is here.”);
```

Here, in this INSERT command, we have not included value for the created_at column. But, when we run this command, that value will be automatically inserted.

☐	Edit	Copy	Delete	10	10	Welcome! Start renting on our platform!	2023-12-11 23:44:32	2023-12-11 23:44:32	NULL
☐	Edit	Copy	Delete	11	2	Your new property is here.	2023-12-12 06:14:25	2023-12-12 06:14:25	NULL

Check all *With selected:* Edit Copy Delete Export

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Figure 8: Default Constraint Check

Here, you can see that data is inserted but in the "created_at" column, timestamp value which stores date and time is inserted automatically.

6. Test SQL Queries

6.1 Select Queries (Data Retrieval Language)

<u>Done By Arun Rajput - 22227184</u>

Query 1: Provide Lists of Users in the Database.

```
SELECT * FROM Users;
```

Showing rows 0 - 10 (11 total, Query took 0.0006 seconds.)

```
SELECT * FROM Users;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

←T→	user_id	username	email	password	firstname	lastname	phone	address	user_type	created_at	updated_at	deleted_at
Edit Copy Delete	1	john_doe	john@example.com	pass123	John	Doe	1234567890	123 Main St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	2	jane_smith	jane@example.com	pass456	Jane	Smith	9876543210	456 Elm St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	3	admin1	admin1@example.com	adminpass	Admin	One	1111111111	789 Oak St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	4	user3	user3@example.com	userpass	User	Three	2222222222	567 Pine St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	5	sam_jones	sam@example.com	sam123	Sam	Jones	3333333333	890 Cedar St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	6	sara_williams	sara@example.com	sara456	Sara	Williams	4444444444	234 Birch St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	7	admin2	admin2@example.com	adminpass2	Admin	Two	5555555555	678 Maple St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	8	user4	user4@example.com	userpass4	User	Four	6666666666	432 Walnut St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	9	alice_parker	alice@example.com	alice789	Alice	Parker	7777777777	876 Cherry St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	10	admin3	admin3@example.com	adminpass3	Admin	Three	8888888888	543 Oak St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL

Figure 9: Lists of Users in the Database

Query 2: Filter all Users who are designated as 'Admin'.

```
SELECT * FROM Users WHERE user_type = 'Admin';
```

Showing rows 0 - 2 (3 total, Query took 0.0005 seconds.)

```
SELECT * FROM Users WHERE user_type = 'Admin';
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	user_id	username	email	password	firstname	lastname	phone	address	user_type	created_at	updated_at	deleted_at
Edit Copy Delete	3	admin1	admin1@example.com	adminpass	Admin	One	1111111111	789 Oak St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	7	admin2	admin2@example.com	adminpass2	Admin	Two	5555555555	678 Maple St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL
Edit Copy Delete	10	admin3	admin3@example.com	adminpass3	Admin	Three	8888888888	543 Oak St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL

Figure 10: Filter all Users who are designated as 'Admin'

Query 3: Fetch particular user based on their User ID.

```
SELECT * FROM Users WHERE user_id = 2;
```

Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.)

```
SELECT * FROM Users WHERE user_id = 2;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

	user_id	username	email	password	firstname	lastname	phone	address	user_type	created_at	updated_at	deleted_at
Edit Copy Delete	2	jane_smith	jane@example.com	pass456	Jane	Smith	9876543210	456 Elm St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL

Figure 11: Fetch particular user based on their User ID

Query 4: Show all the details of the users having ID verified on the system.

```
SELECT * FROM Users INNER JOIN UserProfiles ON users.user_id =
userprofiles.user_id WHERE id_verify_status = 'Verified';
```

Showing rows 0 - 8 (9 total, Query took 0.0008 seconds.)

```
SELECT * FROM Users INNER JOIN UserProfiles ON users.user_id = userprofiles.user_id WHERE id_verify_status = 'Verified';
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

user_id	username	email	password	firstname	lastname	phone	address	user_type	created_at	updated_at	deleted_at	profile_id	user_id	dob	gender	photo
1	john_doe	john@example.com	pass123	John	Doe	1234567890	123 Main St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	1	1	1990-05-15	male	https://example.com/photo1.
2	jane_smith	jane@example.com	pass456	Jane	Smith	9876543210	456 Elm St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	2	2	1985-08-22	female	https://example.com/photo2.
4	user3	user3@example.com	userpass	User	Three	2222222222	567 Pine St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	4	4	1988-11-28	male	https://example.com/photo4.
5	sam_jones	sam@example.com	sam123	Sam	Jones	3333333333	890 Cedar St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	5	5	1992-07-17	female	https://example.com/photo5.
6	sara_williams	sara@example.com	sara456	Sara	Williams	4444444444	234 Birch St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	6	6	1980-04-25	male	https://example.com/photo6.
7	admin2	admin2@example.com	adminpass2	Admin	Two	5555555555	678 Maple St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	7	7	1998-09-03	female	https://example.com/photo7.
8	user4	user4@example.com	userpass4	User	Four	6666666666	432 Walnut St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	8	8	1993-12-12	other	https://example.com/photo8.
9	alice_parker	alice@example.com	alice789	Alice	Parker	7777777777	876 Cherry St	User	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	9	9	1983-06-30	male	https://example.com/photo9.
10	admin3	admin3@example.com	adminpass3	Admin	Three	8888888888	543 Oak St	Admin	2023-12-11 23:26:29	2023-12-11 23:26:29	NULL	10	10	1996-01-05	female	https://example.com/photo1

Figure 12: Show all the details of the users having ID verified on the system

Query 5: Lists property title, location, price and user full name, phone who listed the property.

```
SELECT properties.title, properties.location, properties.price,
CONCAT(users.firstname, ' ', users.lastname) AS full_name, users.phone FROM
properties INNER JOIN users ON properties.user_id = users.user_id;
```

Showing rows 0 - 9 (10 total, Query took 0.0007 seconds.)

```
SELECT properties.title, properties.location, properties.price, CONCAT(users.firstname, ' ', users.lastname) AS full_name, users.phone FROM properties INNER JOIN users ON properties.user_id =
users.user_id;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

title	location	price	full_name	phone
Cozy Apartment	City Center	1200	John Doe	1234567890
Modern Condo	Downtown	1800	Jane Smith	9876543210
Luxury Villa	Suburbia	2500	Admin One	1111111111
Rustic Cabin	Forest Area	800	User Three	2222222222
Beach House	Seaside	3000	Sam Jones	3333333333
Mountain Chalet	Mountain Range	1500	Sara Williams	4444444444
Country Farmhouse	Rural Area	1000	Admin Two	5555555555
Urban Loft	Metropolis	2000	User Four	6666666666
Historic Mansion	Old Town	3500	Alice Parker	7777777777
Family Home	Suburban Neighborhood	1700	Admin Three	8888888888

Figure 13: Lists property title, location, price and user full name, phone who listed the property

Query 6: Display reviews left by a specific user.

```
SELECT * FROM Reviews WHERE user_id = 3;
```

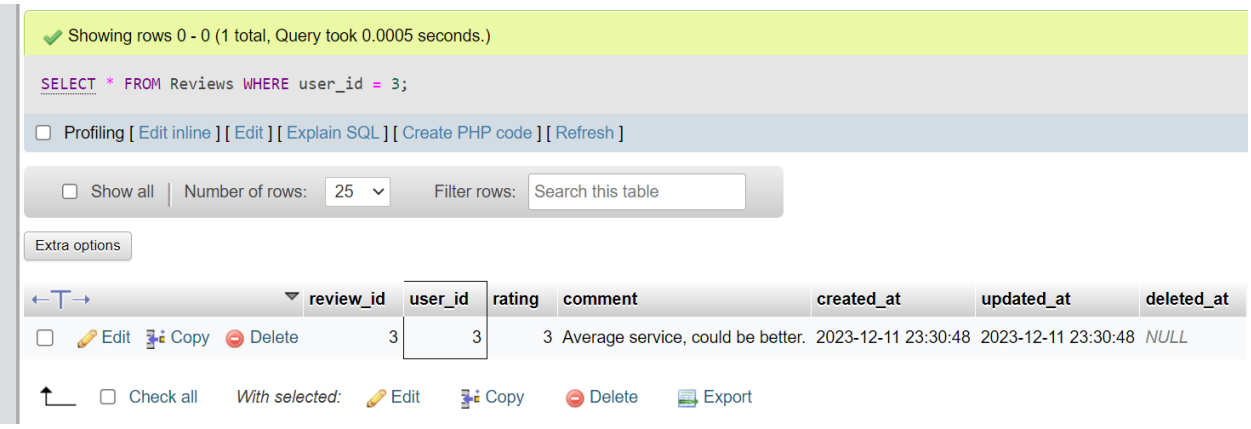


Figure 14: Display reviews left by a specific user

Done By Dinesh Thapa - 23206188

Query 7: Show all latest messages sent by specific user.

```
SELECT * FROM messages WHERE sender_user_id = 1 ORDER BY created_at DESC;
```

Figure 15: Show all latest messages sent by specific user

Query 8: Display all latest notifications associated with specific user.

```
SELECT * FROM notifications WHERE user_id = 1 ORDER BY created_at DESC;
```

Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.) [created_at: **2023-12-11 23:44:32... - 2023-12-11 23:44:32...**]

```
SELECT * FROM notifications WHERE user_id = 1 ORDER BY created_at DESC;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

←T→	notification_id	user_id	content	created_at	updated_at	deleted_at
Edit Copy Delete	1	1	New rental property available! Check it out.	2023-12-11 23:44:32	2023-12-11 23:44:32	NULL

Check all *With selected:* Edit Copy Delete Export

Figure 16: Display all latest notifications associated with specific user

Query 9: Lists subscriptions types with fees less than 25 Pound.

```
SELECT * FROM subs_type WHERE fee < 25;
```

Showing rows 0 - 1 (2 total, Query took 0.0004 seconds.)

```
SELECT * FROM subs_type WHERE fee < 25;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

←T→	subs_id	name	no_of_days	fee	created_at	updated_at	deleted_at
Edit Copy Delete	1	Gold	0	0	2023-12-11 23:50:12	2023-12-11 23:50:12	NULL
Edit Copy Delete	2	Diamond - 15 Days	15	20	2023-12-11 23:50:12	2023-12-11 23:50:12	NULL

Check all *With selected:* Edit Copy Delete Export

Figure 17: Lists subscriptions types with fees less than 25 Pound

Query 10: Display all the latest issue tickets associated with a particular user. Result should include user full name, issue ticket content and ticket created date.

```
SELECT CONCAT(users.firstname, ' ', users.lastname) AS full_name,
issuetickets.content, issuetickets.created_at FROM issuetickets INNER JOIN
users ON issuetickets.user_id = users.user_id WHERE users.user_id = 1;
```

Figure 18: Display all the latest issue tickets associated with a particular user

Query 11: Display full name with payment details including card number, expiry date and cvv of a specific user.

```
SELECT CONCAT(users.firstname, ' ', users.lastname) AS full_name,
paymentdetails.card_number, paymentdetails.expiry_date, paymentdetails.cvv
FROM paymentdetails INNER JOIN users ON paymentdetails.user_id = users.user_id
WHERE users.user_id = 6;
```

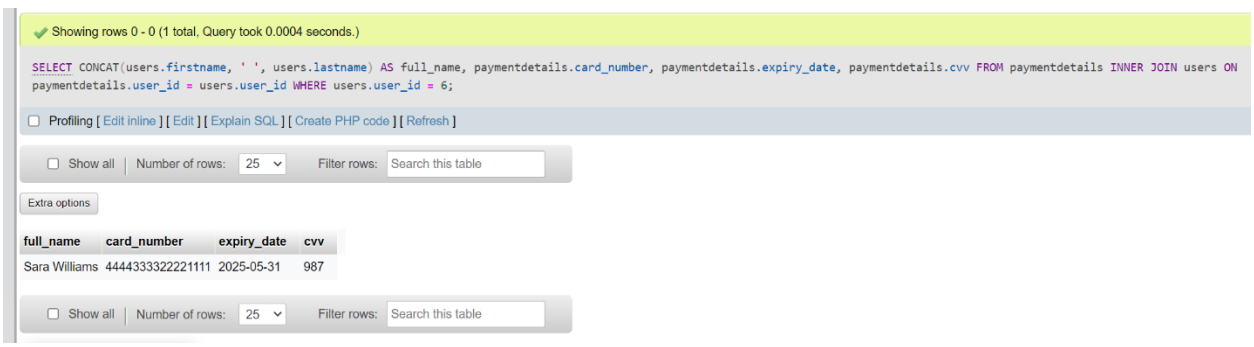


Figure 19: Display full name with payment details including card number, expiry date and cvv of a specific user

Query 12: Display all subscription transactions that are made in 2023. Show user full name, Subscription name, and Card number used for the transaction.

```
SELECT CONCAT(users.firstname, ' ', users.lastname) AS full_name,
subs_type.name, paymentdetails.card_number FROM substransactions

INNER JOIN users ON substransactions.user_id = users.user_id

INNER JOIN subs_type ON substransactions.subs_type_id = subs_type.subs_id
```

```
INNER JOIN paymentdetails ON substransactions.payment_details_id =
paymentdetails.payment_details_id;
```

Showing rows 0 - 9 (10 total, Query took 0.0007 seconds.)

```
SELECT CONCAT(users.firstname, ' ', users.lastname) AS full_name, subs_type.name, paymentdetails.card_number FROM substransactions INNER JOIN users ON substransactions.user_id = users.user_id INNER JOIN subs_type ON substransactions.subs_type_id = subs_type.subs_id INNER JOIN paymentdetails ON substransactions.payment_details_id = paymentdetails.payment_details_id;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

full_name	name	card_number
John Doe	Gold	1234567890123456
Sam Jones	Gold	9999000011112222
Jane Smith	Diamond - 15 Days	9876543210987654
Sara Williams	Diamond - 15 Days	4444333322221111
Admin One	Diamond - 30 Days	1111222233334444
Admin Two	Diamond - 30 Days	8888999977775555
User Three	Diamond - 1 Year	5555666677778888
User Four	Diamond - 1 Year	6666555544443333
Alice Parker	Diamond - 1 Year	1212121212121212
Admin Three	Diamond - 1 Year	1313131313131313

Figure 20: Display all subscription transactions that are made in 2023

Query 13: What is the total income generated by room rental system by selling subscriptions.

```
SELECT sum(subs_type.fee) FROM substransactions INNER JOIN subs_type ON
substransactions.subs_type_id=subs_type.subs_id;
```

Showing rows 0 - 0 (1 total, Query took 0.0004 seconds.)

```
SELECT sum(subs_type.fee) FROM substransactions INNER JOIN subs_type ON substransactions.subs_type_id=subs_type.subs_id;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

sum(subs_type.fee)
910

Show all | Number of rows: 25 Filter rows: Search this table

Figure 21: What is the total income generated by room rental system by selling subscriptions

Question 14: Find the users who have subscribed to a “Diamond – 1 Year” Subscription type.

```
SELECT
    U.user_id,
    U.username,
    S.name AS subscription_type,
    PT.start_date,
    PT.end_date
FROM
    Users U
JOIN
    SubsTransactions PT ON U.user_id = PT.user_id
JOIN
    subs_type S ON PT.subs_type_id = S.subs_id
WHERE
    PT.subs_type_id = (
        SELECT subs_id
        FROM subs_type
        WHERE name = 'Diamond - 1 Year'
    )
    AND PT.payment_details_id IS NOT NULL;
```

Showing rows 0 - 3 (4 total, Query took 0.0020 seconds.)

```
SELECT U.user_id, U.username, S.name AS subscription_type, PT.start_date, PT.end_date FROM Users U JOIN SubsTransactions PT ON U.user_id = PT.user_id JOIN subs_type S ON PT.subs_type_id = S.subs_id
WHERE PT.subs_type_id = ( SELECT subs_id FROM subs_type WHERE name = 'Diamond - 1 Year' ) AND PT.payment_details_id IS NOT NULL;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

user_id	username	subscription_type	start_date	end_date
4	user3	Diamond - 1 Year	2023-12-11 12:00:00	2024-01-10 12:00:00
8	user4	Diamond - 1 Year	2023-12-11 12:00:00	2024-02-09 12:00:00
9	alice_parker	Diamond - 1 Year	2023-12-11 12:00:00	2024-03-11 12:00:00
10	admin3	Diamond - 1 Year	2023-12-11 12:00:00	2023-12-18 12:00:00

Show all | Number of rows: 25 Filter rows: Search this table

Figure 22: Find the users who have subscribed to a "Diamond – 1 Year" Subscription type

6.2 Database Optimization Techniques

Done By Arun Rajput - 22227184

Question 1: How can we improve query performance for retrieving user records based on their user types?

We will implement **Indexing Database Optimization Technique** to solve this mentioned problem.

a. Create Index on "user_type" column to enhance query performance when filtering by user types.

```
CREATE INDEX idx_user_type ON Users (user_type);
```

b. Testing Index Effectiveness.

```
-- Query using user_type column

SELECT * FROM Users WHERE user_type = 'Admin';


-- Query using user_type column with other conditions

SELECT * FROM Users WHERE user_type = 'User' AND created_at > '2023-01-01';
```

c. Analyze Query Execution

```
-- Analyze query using user_type column
EXPLAIN SELECT * FROM Users WHERE user_type = 'Admin';
```

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM Users WHERE user_type = 'Admin';
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

Extra options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Users	*NULL*	ref	idx_user_type	idx_user_type	2	const	3	100.00	Using index condition

Figure 23: Analyzing Query Using user_type column

```
-- Analyze query using user_type column with other conditions
EXPLAIN SELECT * FROM Users WHERE user_type = 'User' AND created_at >
'2023-01-01';
```

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM Users WHERE user_type = 'User' AND created_at > '2023-01-01';
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

Extra options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Users	*NULL*	ref	idx_user_type	idx_user_type	2	const	6	33.33	Using index condition; Using where

Query results operations

Figure 24: Analyze query using user_type column with other conditions

<u>Done By Dinesh Thapa - 23206188</u>

Question 2: How can we optimize the retrieval of user records based on their registration date (created_at)?

We will implement **Partitioning Database Optimization Technique** to solve this mentioned problem.

a. Implementing Range Partitioning

```
CREATE TABLE UsersPartition (

    user_id INT NOT NULL AUTO_INCREMENT,
```

```
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    password VARCHAR(100) NOT NULL,
    firstname VARCHAR(50),
    lastname VARCHAR(50),
    phone VARCHAR(20),
    address VARCHAR(255),
    user_type VARCHAR(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    deleted_at TIMESTAMP,
    PRIMARY KEY (user_id, created_at)
) ENGINE=InnoDB
PARTITION BY RANGE ( UNIX_TIMESTAMP(created_at) ) (
    PARTITION p120 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
    PARTITION p121 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
    PARTITION p112 VALUES LESS THAN MAXVALUE
);
```

b. Insert Sample Data

```
INSERT INTO userspartition
    SELECT * FROM Users;
```

c. Test the performance

```
    -- Query 1: Retrieve user data before 2022 from non-partitioned table
    SELECT * FROM Users WHERE YEAR(created_at) < 2024;

    -- Query 2: Retrieve user data before 2022 from partitioned table
    SELECT * FROM userspartition PARTITION (p112);
```

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM userspartition PARTITION (p112);
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

Extra options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	userspartition	p112	ALL	NULL	NULL	NULL	NULL	10	100.00	NULL

Query results operations

Print Copy to clipboard Create view

Note: #1003 /* select#1 */ select `roomrentalsystem`.`userspartition`.`user_id` AS `user_id`,`roomrentalsystem`.`userspartition`.`username` AS `username`,`roomrentalsystem`.`userspartition`.`email` AS `email`,`roomrentalsystem`.`userspartition`.`password` AS `password`,`roomrentalsystem`.`userspartition`.`firstname` AS `firstname`,`roomrentalsystem`.`userspartition`.`lastname` AS `lastname`,`roomrentalsystem`.`userspartition`.`phone` AS `phone`,`roomrentalsystem`.`userspartition`.`address` AS `address`,`roomrentalsystem`.`userspartition`.`user_type` AS `user_type`,`roomrentalsystem`.`userspartition`.`created_at` AS `created_at`,`roomrentalsystem`.`userspartition`.`updated_at` AS `updated_at`,`roomrentalsystem`.`userspartition`.`deleted_at` AS `deleted_at` from `roomrentalsystem`.`userspartition` PARTITION (`p112`)

Figure 25: Testing the Performance - Partitioning Database Optimization Technique

7. Conclusions

The “Room Rental Marketplace Database System” is a reliable solution designed to meet the dynamic needs of both property owners and prospective tenants. This complete database system includes all the functionalities necessary for efficient management of user profiles, property listings, reviews, messaging, subscriptions, and booking requests. This system guarantees data accuracy, integrity, and security by strictly following our mentioned business rules and constraints offering trustworthy and seamless database platform for interaction and engagement.

Using a relational database model that includes a range of entities such as Users, Profiles, Reviews, Messages, and more to streamline the database operations, arguments user experience, increases efficiency, strengthens stakeholders trust. The integration of user authentication, property listing mechanisms, review systems, and subscription functionalities enhances the overall reliability and usability of the platform. In addition, operational flow is ensured by the system’s ability to store, retrieve, and manage a variety of data features. This allows for instant access to database information while reducing errors and redundancies.

In conclusion, we have developed a efficient, secured and reliable database management system for the Rental Marketplace Management System.

8. References

Song, I.-Y., Evans, M. and Park, U. (1995). A Comparative Analysis of Entity-Relationship Diagrams 1. *Journal of Computer and Software Engineering*, [online] 3(4), pp.427–459. Available at: https://www.cin.ufpe.br/~in1008/aulas/A%20Comparative%20Analysis%20of%20Entity-Relationship%20Diagrams.pdf.

vpadmin (2023). *A Comprehensive Guide to Database Normalization with Examples*. [online] Visual Paradigm Guides. Available at: https://guides.visual-paradigm.com/a-comprehensive-guide-to-database-normalization-with-examples/.

Fergusson, K. (2018). *Entity Relationship Diagrams with draw.io*. [online] draw.io. Available at: https://drawio-app.com/blog/entity-relationship-diagrams-with-draw-io/.

Vertabelo Data Modeler. (2022). *A Detailed Guide to Database Schema Design*. [online] Available at: https://vertabelo.com/blog/database-design-guide/.

Microsoft (2022). *Database design basics*. [online] support.microsoft.com. Available at: https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5.

Intuji. (2023). *Database Design & Database Architecture: Why They're Crucial*. [online] Available at: https://intuji.com/database-design-database-architecture/.