ROLE BASED HOTEL MANAGEMENT SYSTEM

InnMATE
Version: 1.0 | **Date**: 13th November 2024

Project Overview

InnMATE is a **Role-Based Hotel Management System** designed with a strong emphasis on database management, focusing on efficient use of MySQL for data storage and retrieval. The system supports multiple user roles, including Admin, Property Owners, and Regular Users, each with specific access and functionalities.

The core of the system relies on a **relational MySQL database**, which is structured to handle complex relationships between hotel properties, user profiles, bookings, and payments. The database is optimized for performance and scalability, leveraging **normalized tables** to ensure data integrity and minimize redundancy.

Authors:

- **Aniruddh Suresh Gutikar**

  (*5th Semester, Computer Science and Engineering, PES University*)

- **Ashlesha T**
  (*5th Semester, Computer Science and Engineering, PES University*)

Technologies Used

- **Frontend**: Next.js, TypeScript
- **Backend**: MySQL
- **ORM**: Prisma (with raw queries)
- **Authentication**: Kinde
- **Image Upload**: Uploadcare
- **Validation**: Zod

Setup and Installation

To set up and install **InnMATE**, follow the instructions in the **README** file available in the GitHub repository.

GitHub: https://github.com/ASHLESHA05/Inn-mate

Contact Information

For further inquiries, please contact:

- Aniruddh Suresh Gutikar: [aniruddhguttikar@gmail.com]
- Ashlesha T: [ashleshat5@example.com]

Table of Contents

        - User Retrieval (#28, #29, #30)
        - User Updates (#32)
        - User Deletion (#33)
        - User Authentication (#34)
    - D. Favorites and Listings
        - Favorites Management (#7-13)
        - Listings Management (#14-18)
    - E. Reviews and Amenities
        - Review Management (#26, #27)
        - Amenities Retrieval (#1)
    - F. Email Notifications
        - Booking Confirmation Emails (#36)
        - Cancellation Notifications (#35)

4. **Stored Procedures**
    - 4.1 Add Property Procedure
        - Explanation and usage.
5. **Custom Functions**
    - 5.1 match_destination Function
        - Function logic and application.
6. **Triggers**
    - 6.1 Delete Log Trigger
        - Implementation of deletion logging for auditing.
    - 6.2 Duplicate Property Prevention Trigger
        - Ensuring unique property entries.
7. **Event Scheduling**
    - 7.1 Booking Status Update Event
        - Automated updates for booking status changes.
8. **Conclusion**
    - Summary of Database Design and Key Functionalities
    - Overview of integration with the front-end and back-end systems
    - Future improvements and scaling considerations, including additional features and optimizations.

Introduction

Project Overview: InnMate - A Role-Based Hotel Management System

InnMate is a role-based hotel management web application designed to facilitate property and booking management in a user-friendly, streamlined environment. With InnMate, property owners can create and manage property listings and view all bookings associated with their properties. Users can explore available properties and make reservations for specific dates. The system also incorporates a role-based access control structure, providing appropriate permissions to different types of users.

Key Functionalities:

1. Property Management: Property owners can create, update, and delete their property listings. They can also view and manage bookings for each property they own.
2. Booking System: Users can view properties based on available dates and make reservations. Booking details are securely stored, and users can manage or view their reservations.
3. Role-Based Access Control:
    - Admin: Has access to delete properties and manage bookings across the platform.
    - Property Owners: Can manage their own listings and bookings, but are restricted from accessing or altering data belonging to others.
    - Guests: Can browse properties and make bookings but have limited access to management features.

Front-End and Back-End Technologies

- **Front-End**: The InnMate website is built with Next.js and TypeScript, offering a modern, fast, and responsive user interface.
- **Validation**: Zod is used for data validation, ensuring all user inputs meet predefined requirements and improving data integrity.
- **Authentication**: Kinde handles user authentication, managing secure sign-ups and logins.
- **Image Management**: UploadCare is used for property image uploads, providing high-quality image storage and delivery.

Database and Querying

- **Database**: MySQL is used for managing all data related to properties, bookings, users, and roles.
- **Direct Querying with Prisma**: Although Prisma is utilized for schema management, InnMate does not use Prisma's ORM methods. Instead, database operations are handled with prisma.$queryRaw, which allows for raw SQL querying, providing precise control over the data operations and enabling customized SQL queries for optimized performance.

Purpose and Scope of Documentation

This documentation is designed to guide developers and administrators through the database structure, raw SQL queries, stored procedures, and event scheduling that support the InnMate platform. Each section covers:

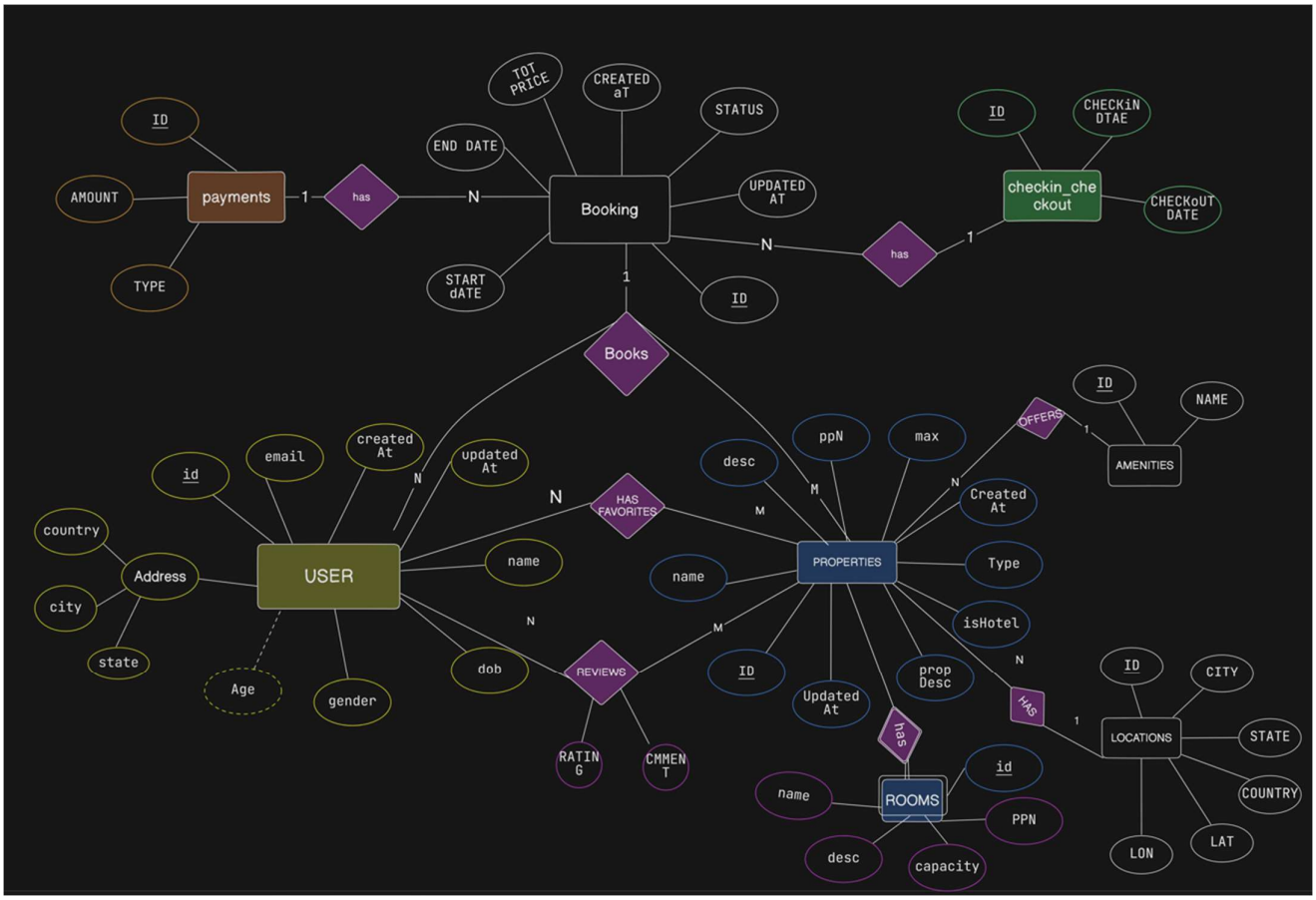
- **Database Schema**: Comprehensive details of table structures, data types, relationships, and integrity constraints.
- **SQL Queries:** Raw SQL queries for property, booking, and user management, allowing for flexible and efficient data retrieval and manipulation.
- **Stored Procedures, Functions, and Triggers:** Enhancements to the database functionality to support efficient data operations and business rules.
- **Event Scheduling**: Automated processes for handling updates and maintaining up-to-date booking information.

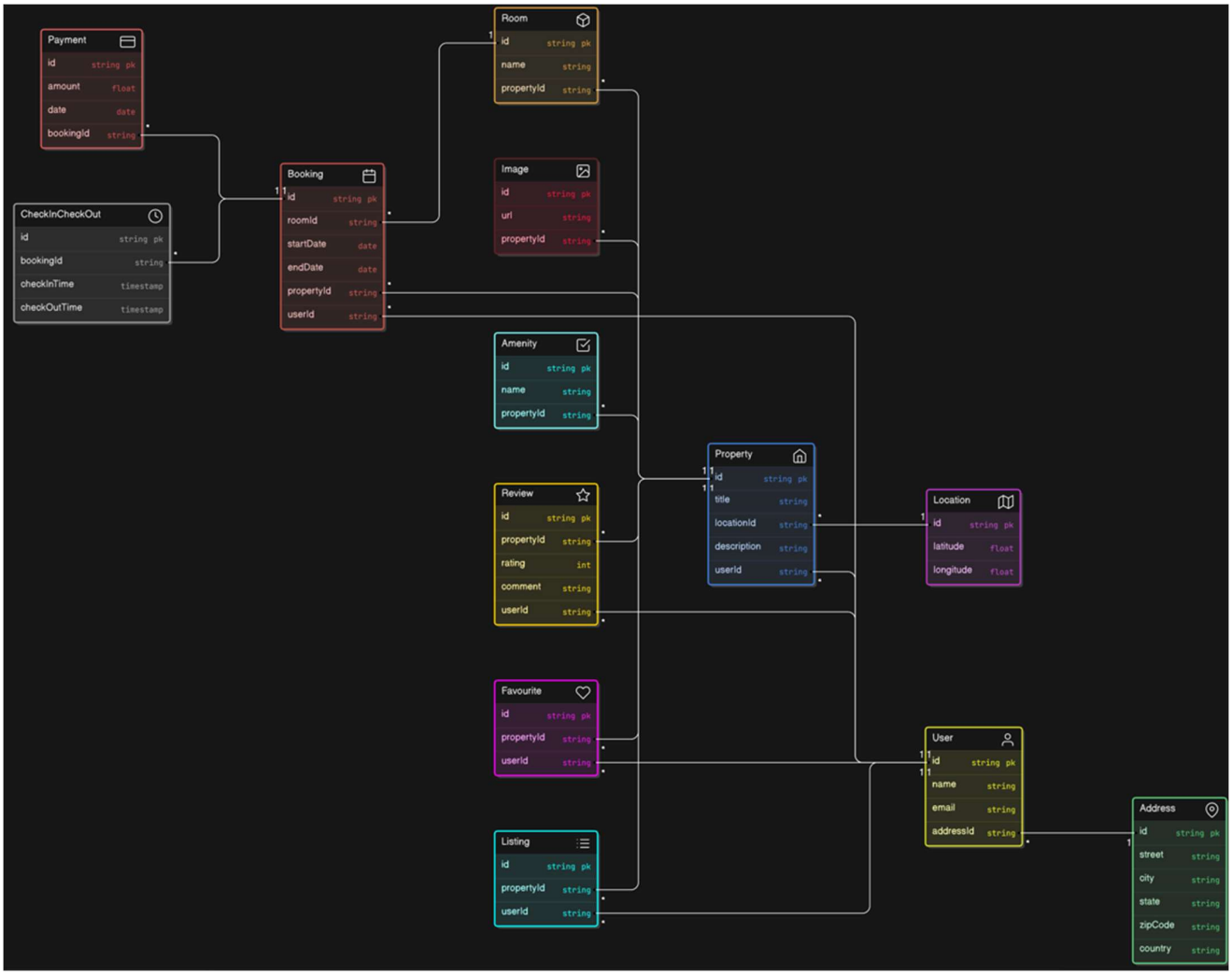
Database Design: Entity-Relationship Diagram (ERD) and Normalization:

The InnMate database is designed in Third Normal Form (3NF) to ensure data consistency, avoid redundancy, and support efficient querying and data manipulation. Below are the main components:

ER Diagram Overview

The ER Diagram provides a visualization of the key entities in the database and their relationships:





Database Normalization: 3NF

The InnMate database is structured in Third Normal Form (3NF), ensuring:

1. First Normal Form (1NF): All tables contain only atomic values, with no repeating groups.
2. Second Normal Form (2NF): All non-key attributes are fully dependent on the primary key.
3. Third Normal Form (3NF): All attributes are directly dependent on the primary key and are free from transitive dependencies.

2.1 TABLE CREATION AND FOREIGN KEY:

-- Create USER Table

```
CREATE TABLE `User` (
    `id` VARCHAR(191) NOT NULL,
    `kindeId` VARCHAR(191) NOT NULL,
    `email` VARCHAR(191) NOT NULL,
    `createdAt` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `updatedAt` DATETIME(3) NOT NULL,
    `name` VARCHAR(191) NOT NULL,
    `dob` DATETIME(3) NOT NULL,
    `gender` ENUM('MALE', 'FEMALE', 'OTHERS') NOT NULL,
    `image` VARCHAR(191) NULL,
    `addressId` VARCHAR(191) NOT NULL,

    UNIQUE INDEX `User_kindeId_key`(`kindeId`),
    UNIQUE INDEX `User_email_key`(`email`),
    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

-- Create Address Table

```
CREATE TABLE `Address` (
    `id` VARCHAR(191) NOT NULL,
    `city` VARCHAR(191) NOT NULL,
    `state` VARCHAR(191) NOT NULL,
    `country` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

-- Create Property Table

```
CREATE TABLE `Property` (
```

```
    `id` VARCHAR(191) NOT NULL,
    `name` VARCHAR(191) NOT NULL,
    `description` VARCHAR(191) NOT NULL,
    `pricePerNight` DOUBLE NOT NULL,
    `maxGuests` INTEGER NOT NULL,
    `createdAt` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `updatedAt` DATETIME(3) NOT NULL,
    `propertyType` ENUM('Hotel', 'Home', 'Resort', 'Farmhouse', 'Beachhouse', 'Cottage', 'Apartment') NOT NULL,
    `isHotel` BOOLEAN NOT NULL DEFAULT false,
    `isDeleted` BOOLEAN NOT NULL DEFAULT false,
    `RoomType` VARCHAR(191) NOT NULL DEFAULT '',
    `locationId` VARCHAR(191) NOT NULL,
    `userId` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

-- Create Room Table

```
CREATE TABLE `Room` (
    `id` VARCHAR(191) NOT NULL,
    `name` VARCHAR(191) NOT NULL,
    `description` VARCHAR(191) NOT NULL,
    `capacity` INTEGER NOT NULL,
    `pricePerNight` DOUBLE NOT NULL,
    `propertyId` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

-- Create Image Table

```
CREATE TABLE `Image` (
    `id` VARCHAR(191) NOT NULL,
    `link` VARCHAR(191) NOT NULL,
    `propertyId` VARCHAR(191) NOT NULL,
```

```
    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- Create Booking Table
CREATE TABLE `Booking` (
    `id` VARCHAR(191) NOT NULL,
    `totalPrice` DOUBLE NOT NULL,
    `createdAt` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `updatedAt` DATETIME(3) NOT NULL,
    `userId` VARCHAR(191) NOT NULL,
    `propertyId` VARCHAR(191) NOT NULL,
    `roomId` VARCHAR(191) NULL,
    `status` ENUM('CONFIRMED', 'ACTIVE', 'COMPLETED') NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- Create Review Table
CREATE TABLE `Review` (
    `id` VARCHAR(191) NOT NULL,
    `rating` INTEGER NOT NULL,
    `comment` VARCHAR(191) NOT NULL,
    `createdAt` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `userId` VARCHAR(191) NOT NULL,
    `propertyId` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- Create Favourite Table
CREATE TABLE `Favourite` (
    `id` VARCHAR(191) NOT NULL,
    `userId` VARCHAR(191) NOT NULL,
```

```
    `propertyId` VARCHAR(191) NOT NULL,

    UNIQUE INDEX `Favourite_userId_propertyId_key`(`userId`, `propertyId`),
    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- Create Listing Table
CREATE TABLE `Listing` (
    `id` VARCHAR(191) NOT NULL,
    `createdAt` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `updatedAt` DATETIME(3) NOT NULL,
    `availabilityStart` DATETIME(3) NOT NULL,
    `availabilityEnd` DATETIME(3) NOT NULL,
    `userId` VARCHAR(191) NOT NULL,
    `propertyId` VARCHAR(191) NOT NULL,

    UNIQUE INDEX `Listing_userId_propertyId_key`(`userId`, `propertyId`),
    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;


-- Create Amenity Table
CREATE TABLE `Amenity` (
    `id` VARCHAR(191) NOT NULL,
    `name` ENUM('WIFI', 'PARKING', 'AIR_CONDITIONING', 'COFFEE', 'PARK', 'POOL', 'GYM', 'KITCHEN', 'TV', 'LAUNDRY',
'PET_FRIENDLY') NOT NULL,
    `propertyId` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- Create Location Table
CREATE TABLE `Location` (
    `id` VARCHAR(191) NOT NULL,
```

```
    `city` VARCHAR(191) NOT NULL,
    `state` VARCHAR(191) NOT NULL,
    `country` VARCHAR(191) NOT NULL,
    `latitude` DOUBLE NULL,
    `longitude` DOUBLE NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

-- Create Payment Table

```
CREATE TABLE `Payment` (
    `id` VARCHAR(191) NOT NULL,
    `amount` DOUBLE NOT NULL,
    `paymentDate` DATETIME(3) NOT NULL,
    `bookingId` VARCHAR(191) NOT NULL,

    UNIQUE INDEX `Payment_bookingId_key`(`bookingId`),
    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

-- Create checkincheckout Table

```
CREATE TABLE `CheckInCheckOut` (
    `id` VARCHAR(191) NOT NULL,
    `checkInDate` DATETIME(3) NOT NULL,
    `checkOutDate` DATETIME(3) NOT NULL,
    `bookingId` VARCHAR(191) NOT NULL,

    UNIQUE INDEX `CheckInCheckOut_bookingId_key`(`bookingId`),
    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

-- AddForeignKey

```
ALTER TABLE `User` ADD CONSTRAINT `User_addressId_fkey` FOREIGN KEY (`addressId`) REFERENCES `Address`(`id`)
ON DELETE RESTRICT ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Property` ADD CONSTRAINT `Property_locationId_fkey` FOREIGN KEY (`locationId`) REFERENCES `Location`(`id`) ON DELETE RESTRICT ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Property` ADD CONSTRAINT `Property_userId_fkey` FOREIGN KEY (`userId`) REFERENCES `User`(`id`) ON DELETE RESTRICT ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Room` ADD CONSTRAINT `Room_propertyId_fkey` FOREIGN KEY (`propertyId`) REFERENCES `Property`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Image` ADD CONSTRAINT `Image_propertyId_fkey` FOREIGN KEY (`propertyId`) REFERENCES `Property`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Booking` ADD CONSTRAINT `Booking_userId_fkey` FOREIGN KEY (`userId`) REFERENCES `User`(`id`) ON DELETE RESTRICT ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Booking` ADD CONSTRAINT `Booking_propertyId_fkey` FOREIGN KEY (`propertyId`) REFERENCES `Property`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Booking` ADD CONSTRAINT `Booking_roomId_fkey` FOREIGN KEY (`roomId`) REFERENCES `Room`(`id`) ON DELETE SET NULL ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Review` ADD CONSTRAINT `Review_userId_fkey` FOREIGN KEY (`userId`) REFERENCES `User`(`id`) ON DELETE RESTRICT ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Review` ADD CONSTRAINT `Review_propertyId_fkey` FOREIGN KEY (`propertyId`) REFERENCES `Property`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Favourite` ADD CONSTRAINT `Favourite_userId_fkey` FOREIGN KEY (`userId`) REFERENCES `User`(`id`) ON DELETE RESTRICT ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Favourite` ADD CONSTRAINT `Favourite_propertyId_fkey` FOREIGN KEY (`propertyId`) REFERENCES `Property`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Listing` ADD CONSTRAINT `Listing_userId_fkey` FOREIGN KEY (`userId`) REFERENCES `User`(`id`) ON DELETE RESTRICT ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Listing` ADD CONSTRAINT `Listing_propertyId_fkey` FOREIGN KEY (`propertyId`) REFERENCES `Property`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Amenity` ADD CONSTRAINT `Amenity_propertyId_fkey` FOREIGN KEY (`propertyId`) REFERENCES `Property`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `Payment` ADD CONSTRAINT `Payment_bookingId_fkey` FOREIGN KEY (`bookingId`) REFERENCES `Booking`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- AddForeignKey

```
ALTER TABLE `CheckInCheckOut` ADD CONSTRAINT `CheckInCheckOut_bookingId_fkey` FOREIGN KEY (`bookingId`) REFERENCES `Booking`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

FOREIGN KEY RELATIONSHIP TABLE:

Foreign Key Constraint Name	Source Table	Source Column	Target Table	Target Column	On Delete Action	On Update Action
User_addressId_fkey	User	addressId	Address	id	RESTRICT	CASCADE
Property_locationId_fkey	Property	locationId	Location	id	RESTRICT	CASCADE
Property_userId_fkey	Property	userId	User	id	RESTRICT	CASCADE
Room_propertyId_fkey	Room	propertyId	Property	id	CASCADE	CASCADE
Image_propertyId_fkey	Image	propertyId	Property	id	CASCADE	CASCADE
Booking_userId_fkey	Booking	userId	User	id	RESTRICT	CASCADE
Booking_propertyId_fkey	Booking	propertyId	Property	id	CASCADE	CASCADE
Booking_roomId_fkey	Booking	roomId	Room	id	SET NULL	CASCADE
Review_userId_fkey	Review	userId	User	id	RESTRICT	CASCADE
Review_propertyId_fkey	Review	propertyId	Property	id	CASCADE	CASCADE
Favourite_userId_fkey	Favourite	userId	User	id	RESTRICT	CASCADE
Favourite_propertyId_fkey	Favourite	propertyId	Property	id	CASCADE	CASCADE
Listing_userId_fkey	Listing	userId	User	id	RESTRICT	CASCADE
Listing_propertyId_fkey	Listing	propertyId	Property	id	CASCADE	CASCADE
Amenity_propertyId_fkey	Amenity	propertyId	Property	id	CASCADE	CASCADE
Payment_bookingId_fkey	Payment	bookingId	Booking	id	CASCADE	CASCADE
CheckInCheckOut_bookingId_fkey	CheckInCheckOut	bookingId	Booking	id	CASCADE	CASCADE

SQL QUERIES:

1 . getAllAmenitiesForProperty

Purpose: Retrieves all amenities for a given property.

SQL Query:

```
SELECT * FROM amenity AS a WHERE a.propertyId = ${propertyId}
```

Parameters:

- propertyId (string): The property ID to filter amenities.

Return Value:

- Promise<TAmenity[] | null>: Returns a validated array of amenities or null if an error occurs.

2**. getBookingDetailsByPropertyId**

Purpose: Fetches booking details for a given propertyId, including property and check-in/check-out information.

```
SELECT
  b.*,
  p.name, p.description, p.Pricepernight, p.maxGuests,
  p.propertytype, p.isHotel, p.isDeleted, p.RoomType,
  p.locationId,
  c.checkInDate, c.checkOutDate,
  b.totalPrice, b.userId
FROM booking AS b
JOIN (SELECT * FROM property WHERE id = ${propertyId} AND isDeleted = false) AS p ON b.propertyId = p.id
JOIN checkIncheckOut AS c ON b.id = c.bookingId
WHERE b.status IN ('ACTIVE', 'CONFIRMED')
```

3. createBooking

Purpose: Inserts a new booking into the booking table, with payment and optional check-in/check-out information.

SQL Queries:

1. **Insert Booking:**

```
INSERT INTO booking (id, totalPrice, createdAt, updatedAt, userId, propertyId, status)
VALUES (${bookingid}, ${validatedBooking.totalPrice}, ${new Date()}, ${new Date()}, ${validatedBooking.userId},
${validatedBooking.propertyId}, ${validatedBooking.status})
```

2. **Insert Payment:**

```
INSERT INTO payment (id, amount, paymentDate, bookingId)
VALUES (${cuid()}, ${validatedBooking.totalPrice}, ${new Date()}, ${bookingid})
```

3. **Insert Check-in/Check-out (if provided):**

```
INSERT INTO checkIncheckOut (id, checkInDate, checkOutDate, bookingId)
VALUES (${cuid()}, ${validatedBooking.checkInOut.checkInDate}, ${validatedBooking.checkIn
```

4. getAllBookingsForProperty

Purpose: Retrieves all bookings for a specific property, along with check-in/check-out dates.

SQL Query:

```
SELECT b.*, c.checkInDate, c.checkOutDate
FROM booking AS b
JOIN checkIncheckOut AS c ON b.id = c.bookingId
WHERE b.propertyId = ${propertyId}
```

5. DeleteBookingsbyIds

Purpose: Deletes bookings by given booking IDs if they are associated with non-deleted properties.

```
Copy code
DELETE FROM booking
WHERE id IN (
  SELECT id FROM booking WHERE id IN (${Prisma.join(bookingIds)})
  AND propertyId IN (SELECT id FROM property WHERE isDeleted = false)
)
```

6. getAllBookedProperties

Purpose: Retrieves all booked properties for a user identified by kindeId, including property and check-in/check-out details.

```
SELECT
  b.*, p.userId AS propUser,
  p.name, p.description, p.Pricepernight, p.maxGuests,
  p.propertytype, p.isHotel, p.isDeleted, p.RoomType,
  p.locationId,
  c.checkInDate, c.checkOutDate,
  b.totalPrice, b.userId
FROM booking AS b
JOIN property AS p ON b.propertyId = p.id
JOIN checkIncheckOut AS c ON b.id = c.bookingId
WHERE b.status IN ('ACTIVE', 'CONFIRMED')
  AND b.userId = ${user.id}
```

```
AND p.id IN (SELECT propertyId FROM booking WHERE status IN ('ACTIVE', 'CONFIRMED'))
```

7. Insert Query in `addliked` Function

```
INSERT INTO favourite (id, userId, propertyId)
VALUES (${id}, ${validatedFavorites.userId}, ${validatedFavorites.propertyId});
```

8. Selection after Insertion in `addliked` Function

```
SELECT * FROM favourite
WHERE userId = ${validatedFavorites.userId} AND propertyId = ${validatedFavorites.propertyId}
ORDER BY id DESC LIMIT 1;
```

9. Selection Query in `deleteLiked` Function

```
SELECT * FROM favourite
WHERE userId = ${validatedFavorites.userId} AND propertyId = ${validatedFavorites.propertyId};
```

10. Deletion Query in `deleteLiked` Function

```
DELETE FROM favourite
WHERE userId = ${validatedFavorites.userId} AND propertyId = ${validatedFavorites.propertyId};
```

11. Check Favorite Query in `getIsFavorite` Function

```
SELECT * FROM favourite
WHERE userId = ${userId} AND propertyId = ${propertyId};
```

---

12. Check Single Favorite Query in `getIsfavourite` Function

```
SELECT * FROM favourite
WHERE userId = ${userId} AND propertyId = ${propertyId}
LIMIT 1;
```

13. Retrieve All Favorites Query in `getAllfavourite` Function

```
SELECT * FROM favourite
WHERE userId = ${user_id.id};
```

14. Check for Existing Listing in `createListing` Function

```
SELECT * FROM listing WHERE userId = ${listingValues.userId} AND propertyId = ${listingValues.propertyId};
```

---

15. Insert New Listing in `createListing` Function

```
INSERT INTO listing (id, availabilityStart, availabilityEnd, updatedAt, userId, propertyId)
VALUES (${newListingId}, ${validatedListing.availabilityStart}, ${validatedListing.availabilityEnd}, ${new Date()}, ${validatedListing.userId}, ${validatedListing.propertyId});
```

---

16. Retrieve Newly Created Listing in `createListing` Function

```
SELECT * FROM listing as l WHERE l.id = ${newListingId};
```

---

17. Retrieve Unique Listing with Availability in `getListing` Function

Using Prisma's ORM syntax to retrieve listings where `availabilityEnd` is greater than or equal to the current date:

```
js
await prisma.listing.findUnique({
  where: {
    userId_propertyId: {
      userId,
      propertyId
    },
    availabilityEnd: {
      gte: new Date(),
    },
  }
});
```

18. Retrieve Location by `locationId` in `getLocationById` Function

```
SELECT * FROM location WHERE id = ${locationId};
```

---

19. **`getAllListedProperties` - Fetch All Listed Properties**

Description: Retrieves up to 20 listed properties, joining the `listing` and `property` tables. Validates the result with a Zod schema.

SQL Query:

```
SELECT p.*, l.availabilityStart, l.availabilityEnd
FROM listing AS l
JOIN property AS p ON p.id = l.propertyId
LIMIT 20;
```

---

20. **`getFilteredListings` - Fetch Properties Based on Filter Criteria**

Description: Retrieves properties based on dynamic filters like `destination`, `checkIn`, `checkOut`, and `type`. Uses `match_destination` (assumed to be a function for text-based filtering) and validates results.

SQL Query:

```
SELECT p.*, l.availabilityStart, l.availabilityEnd
FROM listing AS l
JOIN property AS p ON p.id = l.propertyId
WHERE 1=1
  AND match_destination(p.name, p.description, p.city, p.state, p.country, {destination})
  AND l.availabilityStart <= {checkIn}
  AND l.availabilityEnd >= {checkOut}
  AND p.propertyType = {propertyTypeValue}
LIMIT 20;
```

---

21. **`getAllPropertiesByUserId` - Fetch Properties by User ID**

Description: Retrieves properties associated with a specific `userId`. Optional relationship fields can be included in the query if needed.

SQL Query:

```
SELECT *
FROM property
WHERE userId = {userId};
```

---

22. **`getPropertyById` - Fetch Property by ID**

Description: Retrieves a single property using its `propertyId`.

SQL Query:

```
SELECT *
FROM property
WHERE id = {propertyId};
```

---

23. **`addProperty` - Add a New Property**

Description: Adds a new property entry, including details like location and amenities, and then validates the data. Utilizes a stored procedure `AddProperty`.

SQL Query (Procedure Call):

```
CALL AddProperty(
 {propertyId},
 {user.id},
 {validatedProperty.name},
 {validatedProperty.description},
```

```
  {false},
  {validatedProperty.pricePerNight},
  {validatedProperty.maxGuests},
  {new Date()},
  {validatedLocation.country},
  {validatedLocation.state},
  {validatedLocation.city},
  {validatedProperty.propertyType},
  {validatedProperty.RoomType ?? 'N/A'},
  {validatedProperty.propertyType === 'Hotel'},
  {validatedImagesJson},
  {validatedAmenitiesJson},
  {imageId},
  {amenityId},
  {locationId}
);
```

---

24. **`updateProperty` - Update Existing Property**

Description: Updates property details based on the provided `propertyId` and new data. Also deletes old images and inserts new ones if necessary.

SQL Queries:

- **Update Property**:

```
  UPDATE property
  SET
     name = {validatedProperty.name},
     description = {validatedProperty.description},
     pricePerNight = {validatedProperty.pricePerNight},
```

```
    maxGuests = {validatedProperty.maxGuests},
    PropertyType = {validatedProperty.propertyType},
    UpdatedAt = {new Date()},
    locationId = {location.id},
    isHotel = {isHotel},
    RoomType = {isHotel ? validatedProperty.RoomType : 'N/A'}
  WHERE id = {propertyId};
```

- **Delete Old Images**:

```
  DELETE FROM image WHERE propertyId = {propertyId};
```

- **Insert New Images**:

```
  INSERT INTO Image (id, propertyId, link) VALUES ({cuid()}, {propertyId}, {image.link});
```

- **Fetch Updated Property**:

```
  SELECT
    p.*,
    l.city,
    l.country,
    l.state,
    GROUP_CONCAT(i.link) AS image_links
  FROM property p
    JOIN location l ON p.locationId = l.id
    LEFT JOIN image i ON i.propertyId = p.id
  WHERE p.id = {propertyId}
  GROUP BY p.id, l.city, l.country, l.state;
```

---

25. **`Delete_UploadCare` - Delete an Image from Uploadcare**

Description: Deletes an image from Uploadcare based on its URL using the `deleteFile` function from the Uploadcare API.

No SQL queries are used in this function as it interacts with an external API (Uploadcare).

---

26. **Fetch All Reviews by Property ID:**

```
SELECT * FROM review WHERE propertyId = ${propertyId}
```

27. **Add a New Review:**

```
INSERT INTO review(id, rating, comment, userId, propertyId, createdAt)
VALUES(${cuid()}, ${data.rating}, ${data.comment}, ${data.userId}, ${data.propertyId}, ${data.createdAt})
```

---

28. `getUserById`

- **Use**: Retrieves user details, including city, state, and country, by user ID.
- **SQL Query**:

```
SELECT u.*, city, state, country
FROM User as u
JOIN address as p ON u.addressId = p.id
WHERE u.id = ${id_}
```

---

29. `getUserByKindeId`

- **Use**: Fetches user details by `kindeId`.

- **SQL Query**:

```
SELECT u.*, city, state, country
FROM User as u
JOIN address as p ON u.addressId = p.id
WHERE kindeId = ${kindeId}
```

---

30. `isAuthenticatedUserInDb`

- **Use**: Checks if an authenticated user is present in the database by ID.

- **SQL Query**:

```
SELECT u.*, city, state, country
FROM User as u
JOIN address as ad ON u.addressId = ad.id
WHERE u.id = ${id}
```

---

31. `createUser`

- **Use**: Creates a new user in the database and returns the created user data.

- **SQL Queries**:

  - **Insert into `Address`:**

```
INSERT INTO Address (id, city, state, country)
VALUES (${addressId}, ${address.city}, ${address.state}, ${address.country})
```

  - **Retrieve address:**

```
SELECT * FROM Address WHERE id = ${addressId}
```

- **Insert into `User`:**

```
INSERT INTO User (id, name, email, dob, gender, UpdatedAt, kindeId, addressId)
VALUES (${userId}, ${name}, ${email}, ${dob}, ${gender}, ${new Date()}, ${kindeId}, ${addressId})
```

- **Retrieve user:**

```
SELECT u.*, city, state, country FROM User as u
JOIN address as ad ON u.addressId = ad.id
```

---

32. `updateUser`

- **Use**: Updates an existing user's information and address in the database.

- **SQL Queries**:

- **Retrieve address ID:**

```
SELECT addressId FROM User WHERE id = ${id}
```

- **Insert or update `Address`:**

```
INSERT INTO Address (id, city, state, country)
VALUES (${AddId}, ${address.city}, ${address.state}, ${address.country})
```

or

```
UPDATE Address
SET city = ${address.city}, state = ${address.state}, country = ${address.country}
WHERE id = ${AddId}
```

- **Update `User`:**

```
UPDATE User as u
SET name = ${name}, email = ${email}, dob = ${dob}, gender = ${gender}, addressId = ${AddId}
```

```
WHERE id = ${id}
```

- Retrieve updated user:

```
SELECT u.*, city, state, country FROM User as u
JOIN address as ad ON u.addressId = ad.id
WHERE u.id = ${id}
```

---

33. `deleteUser`

- **Use**: Deletes a user and their associated address in the database.

- **SQL Queries**:

  - **Delete from `Address`:**

```
DELETE FROM Address WHERE id = ${addressId}
```

  - Delete from `User` and return deleted user:

```
DELETE FROM User WHERE id = ${id} RETURNING *
```

---

34. `mapKindeUserToUser`

- **Use**: Maps Kinde user data to the local user data schema, checks if the user exists, and returns the user or new mapped data.

- **SQL Queries**:

  - **Check if Kinde user exists:**

```
SELECT u.*, city, state, country FROM User as u
JOIN address as ad ON u.addressId = ad.id
WHERE u.kindeId = ${user.id}
```

---

35. `SendMailToUsers`

- **Use**: Sends an email to users whose bookings have been canceled.

- **SQL Queries**:

  - **Retrieve user by ID:**

```
SELECT u.*, city, state, country FROM User as u
JOIN address as ad ON u.addressId = ad.id
WHERE u.id = ${userId}
```

---

36. `BookingCnfMail`

- **Use**: Sends a booking confirmation email to a user, using the booking and transaction details.

- **SQL Queries**:

  - Retrieve user by ID:

```
SELECT u.*, city, state, country FROM User as u
JOIN address as ad ON u.addressId = ad.id
WHERE u.id = ${userId}
```

PROCEDURES:

1 . Add-property-procedure:

```
DELIMITER //

CREATE PROCEDURE AddProperty(
  IN propertyId VARCHAR(191),
```

```
 IN userId VARCHAR(191),
 IN name VARCHAR(191),
 IN description VARCHAR(191),
 IN isDeleted BOOLEAN,
 IN pricePerNight DECIMAL(10, 2),
 IN maxGuests INT,
 IN updatedAt DATETIME,
 IN propertyCountry VARCHAR(191),
 IN propertyState VARCHAR(191),
 IN propertyCity VARCHAR(191),
 IN propertyType VARCHAR(191),
 IN roomType VARCHAR(191),
 IN isHotel BOOLEAN,
 IN images JSON,
 IN amenities JSON,
 IN imageId VARCHAR(191),
 IN AmenityId VARCHAR(191),
 IN inputLocationId VARCHAR(191)  -- Renamed from newLocationId
)
BEGIN
 DECLARE locationId VARCHAR(191);

 -- Check if location exists, else insert and set locationId
 SELECT id INTO locationId
 FROM location
 WHERE country COLLATE utf8mb4_unicode_ci = propertyCountry COLLATE utf8mb4_unicode_ci
   AND state COLLATE utf8mb4_unicode_ci = propertyState COLLATE utf8mb4_unicode_ci
   AND city COLLATE utf8mb4_unicode_ci = propertyCity COLLATE utf8mb4_unicode_ci;

 -- If location doesn't exist, use inputLocationId as new location id
 IF locationId IS NULL THEN
   INSERT INTO location (id, country, state, city)
   VALUES (inputLocationId, propertyCountry, propertyState, propertyCity);
   SET locationId = inputLocationId;
```

```
        END IF;

        -- Insert property
        INSERT INTO property (id, userId, name, description, isDeleted, pricePerNight, maxGuests, updatedAt,
locationId, propertyType, roomType, isHotel)
        VALUES (propertyId, userId, name, description, isDeleted, pricePerNight, maxGuests, updatedAt, locationId,
propertyType, roomType, isHotel);

        -- Insert images if provided
        IF JSON_LENGTH(images) IS NOT NULL AND JSON_LENGTH(images) > 0 THEN
          INSERT INTO image (id, propertyId, link)
          SELECT imageId, propertyId, link
          FROM JSON_TABLE(images, "$[*]" COLUMNS (link VARCHAR(191) PATH "$.link")) AS img
          WHERE link IS NOT NULL AND link <> '';  -- Ensures link is not NULL or empty
        END IF;

        -- Insert amenities if provided
        IF JSON_LENGTH(amenities) IS NOT NULL AND JSON_LENGTH(amenities) > 0 THEN
          INSERT INTO amenity (id, propertyId, name)
          SELECT AmenityId, propertyId, name
          FROM JSON_TABLE(amenities, "$[*]" COLUMNS (name VARCHAR(191) PATH "$.name")) AS amn
          WHERE name IN ('WIFI', 'PARKING', 'AIR_CONDITIONING', 'COFFEE', 'PARK', 'POOL', 'GYM', 'KITCHEN', 'TV',
'LAUNDRY', 'PET_FRIENDLY'); -- Validate amenity names
        END IF;
      END //

      DELIMITER ;
```

FUNCTIONS:

1. add-match-destination-function

```
-- This migration file adds the custom function to MySQL

DELIMITER $$

CREATE FUNCTION match_destination(
    prop_name VARCHAR(255),
    prop_description TEXT,
    prop_city VARCHAR(255),
    prop_state VARCHAR(255),
    prop_country VARCHAR(255),
    destination VARCHAR(255)
)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    RETURN
        LOWER(prop_name) LIKE CONCAT('%', LOWER(destination), '%') OR
        LOWER(prop_description) LIKE CONCAT('%', LOWER(destination), '%') OR
        LOWER(prop_city) LIKE CONCAT('%', LOWER(destination), '%') OR
        LOWER(prop_state) LIKE CONCAT('%', LOWER(destination), '%') OR
        LOWER(prop_country) LIKE CONCAT('%', LOWER(destination), '%');
END $$

DELIMITER ;
```

Triggers:

1.delete-log

```
-- Create a deletion_log table to store the log of deleted properties
CREATE TABLE IF NOT EXISTS deletion_log (
  id INT AUTO_INCREMENT PRIMARY KEY,
  propertyId VARCHAR(255) NOT NULL,
  deletedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);

-- Create the trigger to log deletions
DELIMITER $$

CREATE TRIGGER log_property_deletion
BEFORE DELETE ON property
FOR EACH ROW
BEGIN
  -- Log the deletion of the property
  INSERT INTO deletion_log (propertyId) VALUES (OLD.id);
END $$

DELIMITER ;
```

2. Duplicate property trigger

```
DELIMITER //
CREATE TRIGGER before_property_insert
BEFORE INSERT ON property
FOR EACH ROW
BEGIN
  DECLARE count INT;
  SELECT COUNT(*) INTO count
  FROM property
  WHERE userId = NEW.userId AND locationId = NEW.locationId;

  IF count > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Duplicate property for user at the same location.';
  END IF;
END //
DELIMITER ;
```

-- The trigger is designed to prevent a user from adding duplicate properties at the same location.

-- It ensures that a user cannot add more than one property at the same locationId in the property table.

Event Scheduling:

```
DELIMITER $$

CREATE EVENT update_booking_status
ON SCHEDULE EVERY 1 HOUR
DO
BEGIN
  -- Set bookings to 'ACTIVE' if the check-in date is today or earlier and check-out is in the future
  UPDATE booking
  SET status = 'ACTIVE'
  WHERE checkinDate <= CURDATE() AND checkoutDate > CURDATE();

  -- Set bookings to 'COMPLETED' if the check-out date is today or earlier
  UPDATE booking
  SET status = 'COMPLETED'
  WHERE checkoutDate <= CURDATE();
END $$

DELIMITER ;
```

CONCLUSION:

In conclusion, the Innmate role-based hotel management system represents a comprehensive solution for managing property listings, bookings, user interactions, and administrative controls within a secure, scalable environment. Designed using Next.js, TypeScript, and a robust backend with MySQL and Prisma (leveraging prisma.$queryRaw for raw SQL operations), the system provides high performance, streamlined database management, and precise data handling. User authentication is handled via Kinde, while Uploadcare manages image uploads, ensuring a smooth user experience.

The system architecture, designed around modular database tables, allows seamless interactions between entities such as User, Property, Booking, and Amenities. Additionally, a detailed ER diagram illustrates the relationships and dependencies, showcasing a schema optimized to Third Normal Form (3NF) for data integrity, minimized redundancy, and efficient querying.

With role-based access, Innmate effectively separates functionalities for users, property owners, and administrators, empowering users with control over their listings, bookings, and preferences, while allowing administrators the ability to maintain order and enforce policies through property and booking management. This documentation highlights the system's capabilities, technical implementation, and database structure, reflecting a solution engineered to enhance property management efficiency, user satisfaction, and overall system robustness.