Sure! Here's the database schema without the **ASP.NET Identity** tables, assuming you'll handle authentication and roles yourself or use a simplified version for the sake of this project.

**Database Tables (Without ASP.NET Identity)**

**1. Users Table**

This table stores information about both voyagers and admins. Since you're not using ASP.NET Identity, you'll need to manage authentication and roles manually.

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| Id | Guid | Primary Key, unique identifier for users |
| UserName | String | Username (unique) |
| Email | String | Email address (unique) |
| PasswordHash | String | Hashed password |
| FullName | String | Full name of the user |
| IsVoyager | Boolean | True if user is a voyager, false if admin |
| DateCreated | DateTime | Date the user was created |

**2. Roles Table**

This table stores roles (Admin, Voyager).

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| Id | Guid | Primary Key, unique identifier for roles |
| Name | String | Role name (e.g., "Admin", "Voyager") |
| Description | String | Role description |

**3. Facility Table**

This table stores information about the facilities available for booking.

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| Id | Int | Primary Key |
| Name | String | Name of the facility (e.g., Pool, Gym) |
| Description | String | Description of the facility |
| Fee | Decimal | Fee for booking the facility |
| AvailableSlots | Int | Number of available slots for booking |
| CreatedBy | Guid | FK to Users (Admin who created the facility) |

**4. Room Table**

This table stores information about rooms available for booking.

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| Id | Int | Primary Key |
| RoomNumber | String | Room number or identifier |
| Type | String | Room type (e.g., Single, Suite, etc.) |
| Fee | Decimal | Fee for booking the room |
| AvailableSlots | Int | Number of available rooms |
| CreatedBy | Guid | FK to Users (Admin who created the room) |

**5. Booking Table**

This table stores information about bookings made by voyagers for both facilities and rooms.

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| Id | Int | Primary Key |
| VoyagerId | Guid | FK to Users (Voyager who made the booking) |
| FacilityId | Int | FK to Facility (if booked) |
| RoomId | Int | FK to Room (if booked) |
| BookingDate | DateTime | Date the booking was made |
| StartTime | DateTime | Start time of the booking |
| EndTime | DateTime | End time of the booking |
| Status | String | Booking status (e.g., "Booked", "Cancelled") |

**6. Bill Table**

This table stores information about the bills generated for each voyager.

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| Id | Int | Primary Key |
| VoyagerId | Guid | FK to Users (Voyager who the bill belongs to) |
| Amount | Decimal | Total amount of the bill |
| Status | String | Payment status (e.g., "Paid", "Pending") |
| CreatedDate | DateTime | Date the bill was generated |
| BookingDetails | String | Details about the bookings (facility/room) |

**Relationship Overview:**

* **Users ↔ Bookings**: One-to-many (A voyager can have many bookings, but a booking belongs to one voyager).
* **Facilities ↔ Bookings**: One-to-many (A facility can have many bookings, but a booking refers to one facility).
* **Rooms ↔ Bookings**: One-to-many (A room can have many bookings, but a booking refers to one room).
* **Users ↔ Bills**: One-to-many (A voyager can have many bills, but a bill belongs to one voyager).

**Summary of the Key Points:**

1. **Users Table**: Stores user information (voyagers and admins).
2. **Roles Table**: Defines the roles (Admin, Voyager).
3. **Facility Table**: Stores details about facilities (e.g., Gym, Pool).
4. **Room Table**: Stores details about rooms available for booking.
5. **Booking Table**: Tracks bookings made by voyagers, including facility/room bookings.
6. **Bill Table**: Stores information about bills generated for voyagers.

**Managing Roles:**

To manage roles manually (since you’re not using ASP.NET Identity for this), you would:

1. Assign a role when creating a user.
2. For login, you can manually authenticate users by checking their password hash and assign them the role during login.
3. Based on the user's role, control access to views or controllers using logic in your controller or using role-based authorization like:

// Example of checking roles manually

if (user.IsVoyager)

{

// Voyager functionality

}

else

{

// Admin functionality

}

This approach gives you flexibility while still maintaining basic user-role management.

Let me know if you need any further clarification or help with the code implementation!