Got it! Since you're building a **Console App** using **.NET Framework** and **ADO.NET** for SQL interactions, I'll walk you through how to implement basic functionality for interacting with the SQL database based on the tables we discussed earlier.

**Step-by-Step Guide for Implementing ADO.NET with the SQL Tables**

**1. Setting Up the Console App**

* Open **Visual Studio**.
* Create a new **Console App (.NET Framework)** project.
* Add references to **System.Data.SqlClient** for ADO.NET operations.

**2. Connection String**

You will need a **connection string** to connect your application to the SQL database. Here’s how to add it:

* In your App.config, add the connection string for your SQL Server database:

<configuration>

<connectionStrings>

<add name="DefaultConnection"

connectionString="Server=your\_server\_name;Database=your\_database\_name;Integrated Security=True;"

providerName="System.Data.SqlClient" />

</connectionStrings>

</configuration>

Replace your\_server\_name and your\_database\_name with your actual SQL Server details.

**3. Creating a Database Helper Class**

Now, we will create a **DatabaseHelper** class to handle basic CRUD operations for each of the tables.

**Example of a Basic Helper Class to Connect to SQL:**

using System;

using System.Data;

using System.Data.SqlClient;

public class DatabaseHelper

{

private readonly string \_connectionString;

public DatabaseHelper(string connectionString)

{

\_connectionString = connectionString;

}

// Method to open SQL connection

private SqlConnection GetConnection()

{

return new SqlConnection(\_connectionString);

}

// Example method to execute a SQL command (for inserting data)

public int ExecuteCommand(string commandText, SqlParameter[] parameters = null)

{

using (var connection = GetConnection())

{

using (var command = new SqlCommand(commandText, connection))

{

if (parameters != null)

{

command.Parameters.AddRange(parameters);

}

connection.Open();

return command.ExecuteNonQuery(); // Returns the number of rows affected

}

}

}

// Example method to execute a SQL query (for retrieving data)

public DataTable ExecuteQuery(string query, SqlParameter[] parameters = null)

{

using (var connection = GetConnection())

{

using (var command = new SqlCommand(query, connection))

{

if (parameters != null)

{

command.Parameters.AddRange(parameters);

}

using (var adapter = new SqlDataAdapter(command))

{

var dataTable = new DataTable();

adapter.Fill(dataTable);

return dataTable;

}

}

}

}

}

This helper class provides methods for executing **non-query** commands (like INSERT, UPDATE, DELETE) and executing **queries** that return data.

**4. CRUD Operations for Each Table**

**Example: Inserting a New User (Voyager)**

Here’s an example of how to insert a new **voyager** user into the **Users** table.

public void RegisterUser(string userName, string email, string passwordHash, string fullName, string phoneNumber, bool isVoyager)

{

string query = "INSERT INTO Users (Id, UserName, Email, PasswordHash, FullName, PhoneNumber, IsVoyager, DateCreated) " +

"VALUES (@Id, @UserName, @Email, @PasswordHash, @FullName, @PhoneNumber, @IsVoyager, @DateCreated)";

var parameters = new[]

{

new SqlParameter("@Id", Guid.NewGuid()), // Generate a new GUID for the user

new SqlParameter("@UserName", userName),

new SqlParameter("@Email", email),

new SqlParameter("@PasswordHash", passwordHash),

new SqlParameter("@FullName", fullName),

new SqlParameter("@PhoneNumber", phoneNumber),

new SqlParameter("@IsVoyager", isVoyager),

new SqlParameter("@DateCreated", DateTime.Now)

};

var dbHelper = new DatabaseHelper("your\_connection\_string");

int result = dbHelper.ExecuteCommand(query, parameters);

Console.WriteLine($"{result} record(s) inserted.");

}

**Example: Retrieving User Information**

To retrieve user information (e.g., for login or viewing a user profile), you can use a query like this:

public DataTable GetUserById(Guid userId)

{

string query = "SELECT \* FROM Users WHERE Id = @UserId";

var parameters = new[]

{

new SqlParameter("@UserId", userId)

};

var dbHelper = new DatabaseHelper("your\_connection\_string");

return dbHelper.ExecuteQuery(query, parameters);

}

**Example: Updating a User’s Profile**

To update a user’s profile (e.g., update their phone number), you can use an UPDATE query like this:

public void UpdateUserProfile(Guid userId, string newPhoneNumber)

{

string query = "UPDATE Users SET PhoneNumber = @PhoneNumber WHERE Id = @UserId";

var parameters = new[]

{

new SqlParameter("@PhoneNumber", newPhoneNumber),

new SqlParameter("@UserId", userId)

};

var dbHelper = new DatabaseHelper("your\_connection\_string");

int result = dbHelper.ExecuteCommand(query, parameters);

Console.WriteLine($"{result} record(s) updated.");

}

**5. Example for Facility and Room Booking**

For booking facilities and rooms, you can follow similar steps to **insert** records into the **Booking** table, like this:

public void BookFacility(Guid voyagerId, int facilityId, DateTime startTime, DateTime endTime)

{

string query = "INSERT INTO Booking (VoyagerId, FacilityId, BookingDate, StartTime, EndTime, Status) " +

"VALUES (@VoyagerId, @FacilityId, @BookingDate, @StartTime, @EndTime, @Status)";

var parameters = new[]

{

new SqlParameter("@VoyagerId", voyagerId),

new SqlParameter("@FacilityId", facilityId),

new SqlParameter("@BookingDate", DateTime.Now),

new SqlParameter("@StartTime", startTime),

new SqlParameter("@EndTime", endTime),

new SqlParameter("@Status", "Booked")

};

var dbHelper = new DatabaseHelper("your\_connection\_string");

int result = dbHelper.ExecuteCommand(query, parameters);

Console.WriteLine($"{result} facility booked.");

}

**6. Canceling a Booking**

To cancel a booking, you can update the Status column in the **Booking** table:

public void CancelBooking(int bookingId)

{

string query = "UPDATE Booking SET Status = @Status WHERE Id = @BookingId";

var parameters = new[]

{

new SqlParameter("@Status", "Cancelled"),

new SqlParameter("@BookingId", bookingId)

};

var dbHelper = new DatabaseHelper("your\_connection\_string");

int result = dbHelper.ExecuteCommand(query, parameters);

Console.WriteLine($"{result} booking(s) cancelled.");

}

**Conclusion**

This basic approach using **ADO.NET** allows you to interact with your SQL database through the **DatabaseHelper** class. You can use these methods to **insert**, **update**, **delete**, and **select** data from the database for your **Users**, **Roles**, **Facilities**, **Rooms**, **Bookings**, and **Bills** tables.

With this structure in place, you can extend the functionality by adding more operations (like **role-based access control** or **viewing bookings**).

Let me know if you need more details or if you'd like to dive deeper into any part of the implementation!