
Zealand
Academy of Technologies and Business

Databases

Mandatory assignment
Software Design

Carsten Lydeking

cal002@edu.zealand.dk

Teacher:	M. Lynggaard Krarup
Report Deadline:	03/05/24
Handed-in:	13/04/24
Faculty:	Computer Science

Cover: Generated image of binary using DALL-E
Style: ZBC template – created by Carsten Lydeking

Zealand
Academy of Technologies and Business

Contents

Nomenclature

0.1. Abbreviations and definitions

The following abbreviations and definitions are used throughout the report:

Abb.	Definition
DB	Database: A structured collection of data stored electronically.
RDB	Relational Database: A type of DB that stores and provides access to data points that are related to one another.
DBMS	Database Management System: Software that handles the storage, retrieval, and updating of data in a database.
RDBMS	Relational DBMS: A database management system based on the relational model introduced by E.F. Codd.
SQL	Structured Query Language: A programming language used to manage and manipulate relational databases.
CRUD	Create, Read, Update, Delete The four basic operations of persistent storage: adding, retrieving, modifying, and removing data.
GUI	Graphical User Interface A user interface that allows users to interact with electronic devices using graphical icons and visual indicators.
PK	Primary Key:A unique identifier for each record in a database table.
FK	Foreign Key: A field in a database table that links to the primary key of another table.
NML	Normalization: In relation to RDB design, the process of organizing the columns (attributes) and tables (relations) to minimize data redundancy.
1NF	First Normal Form: A stage of NML where each table has atomic (i.e. indivisible) values and each record needs to be unique.
2NF	Second Normal Form: A stage of NML where it meets all the requirements of the 1NF and does not have partial dependency.

3NF	Third Normal Form: A stage of NML where it meets all the requirements of the 2NF and has no transitive functional dependencies.
ERD	Entity Relationship Diagram: A graphical representation of entities and their relationships to each other, typically used in database design.
UML	Unified Modeling Language: A standardized modeling language used to specify, visualize, construct, and document the artifacts of software systems.
DDL	Data Definition Language: A subset of SQL used to define database structures, such as tables, schemas, and databases.
DML	Data Manipulation Language: A subset of SQL used for adding (inserting), deleting, and modifying (updating) data in a database.
DCL	Data Control Language: A subset of SQL used to control access to data in a database.
TCL	Transaction Control Language: A subset of SQL used to manage the changes made by DML statements.
XML	Extensible Markup Language: A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
JSON	JavaScript Object Notation: A lightweight data-interchange format that is easy for humans to read and write, and for machines to parse and generate.

1 Introduction

1.1. The Report

This report is written in LaTeXin VS Code with the LaTeXWorkshop extension. A template has also been prepared for use by other students at Zealand. It is expected that this will be made available on GitHub at a later date. The book [**connolly2023database**] has been used as a reference for the database design and normalization throughout the project. As a strong opposer of *Danglish*, I have chosen to write in english, as it is the base language for the applied software scripts, programming languages and terminology.

1.2. Used Tools

- LaTeX- For writing the report
- Visual Studio Code v. 1.60.2 - With LaTeXWorkshop extension for editing
- Visual Studio 2022 Enterprise Edition v. 17.8.0 - For application development
- GitHub - For version control
- Microsoft SQL Server Management Studio v. 19.3 - For database development
- Microsoft SQL Server 2022 - For hosting the database

1.3. Purpose

This report has been prepared as part of the curriculum for Software Design at Zealand. The report deals with a case where a database is to be developed for a fictitious Hotel chain. In addition, an application is to be developed that can access the database. Furthermore an application is to be developed that can access the database.

1.4. The assigmnet

Has been copied verbatim from Moodle and is as follows:

1.4.1. Afleveringsformat

Følgende uploades i WiseFlow:

- En rapport på PDF format
- En reference til jeres kode uploades sammen med rapporten enten som ZIP-fil eller med en reference til GitHub i rapporten.

1.4.2. Opgaven:

1. Lav en database model (Hint: beskriv attributes, relationer, multiplicitet, primary keys (PK) og foreign keys (FK)).
2. Redegør for, at dine tabeller er på 3je Normalform.
3. Lav et SQL script over, hvorledes du kan oprette de nødvendige tabeller.
4. Lav din database, så den indeholder tabellerne.
5. Indsæt nogle passende værdier.
6. Lav en Windows Form Applikation (eller console applikation), så den kan lave CRUD på faciliteter.

Besvarelsen skal indeholde de nødvendige oplysninger, så jeres lærer kan afprøve jeres applikation.

2
Database

2.1. Database Model

The logic behind the database is to keep track of reservations, customers, hotels, rooms and facilities. The database is designed to handle reservations of rooms at hotels, and to keep track of customer information and what facilities the hotels offer.

This results in the database consisting of the following tables:

- Hotels
- Customers
- Rooms
- Reservations
- Facilities
- HotelFacilities

2.1.1. Relations and Multiplicity

- A Hotel can have many Rooms (1 to Many)
- A Customer can have many Reservations (1 to Many)
- A Room can be part of one Reservation at a time (1 to Many)
- Facilities are shared among Hotels through HotelFacilities (Many to Many)

2.1.2. 3NF State

All tables are in 3NF because:

1. They are in 1NF as all attributes are *atomic* - no groups are repeated or contain more than one value. This can be seen, for example, by HotelFacilities being a relationship table between Hotel and Facility.
2. They are in 2NF as there is no *partial dependency* - Non-key columns depend on the (whole) primary key. This can be seen, for example, by Hotel having a relationship to Facility instead of containing Facility as an attribute. Also there is no composite key in any table.
3. There are no *transitive dependencies* - Non-key columns depend only on the primary key. It is hard to prove, that something is not, but it is a result of the 1NF and 2NF states.

3

Database Diagrams

3.1. Models and relationships

The diagrams should be simple enough to be self-explanatory. The DMD is created first and based on the many-to-many relationship between hotels and facilities, it gives rise to an index table (HotelFacilities). In the DMD, HotelFacilities is not considered as an independent entity but rather as a way to manage the relationship between Hotel and Facilities. The ERD is then created from the DMD and shows the relationships between the tables, their PK, FK, and data types.

3.1.1. Domain Model Diagram

See figure **??**. This DMD only contains entities, multiplicities, and relationships. This provides input on how the database should be designed based on the individual tables and their relationships to each other.

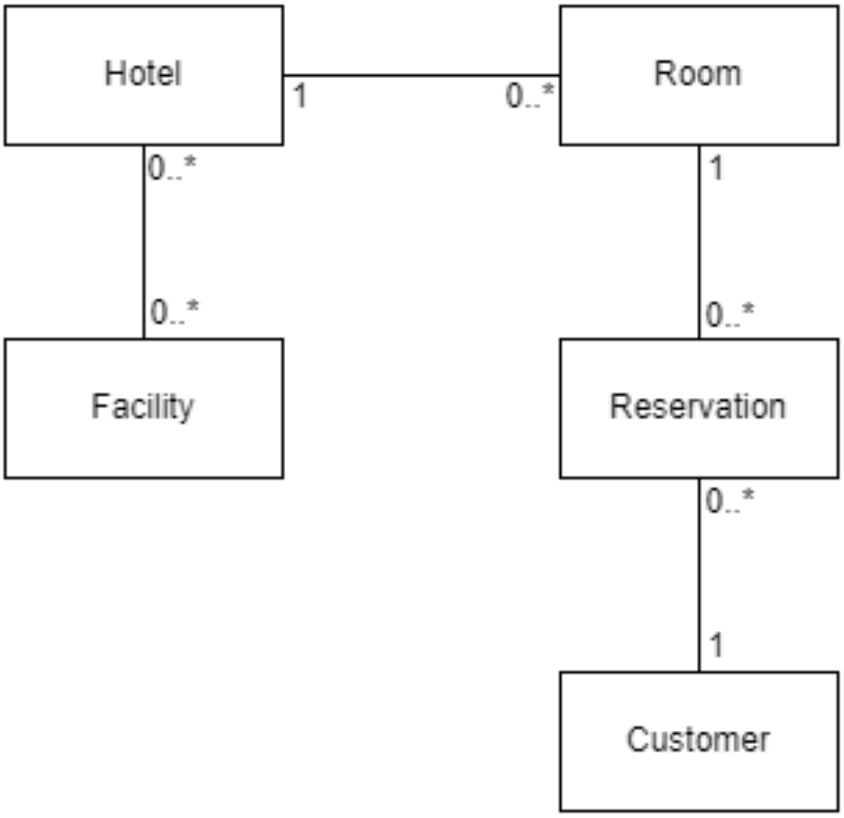

Figure 3.1: Domain Model Diagram

Figure 3.2: Entity Relationship Diagram

3.1.2. Entity-Relationship Diagram

See figure **??**. The ERD is generated from the tables in the database and shows the relationships between the tables, their PK, FK, and data types.

4

WinForm Application

4.1. Considerations regarding the application

See **??** for screenshots of the application.

4.1.1. Scope

Based on the task description and my interpretation of the intention behind the task, the application is the tertiary priority - implying that the database is the primary focus, and diagrams and arguments for documentation are secondary. Therefore, I have narrowed down the task to an application that can perform CRUD operations on the "Facilities" table in the database. This leads me to the following functionality:

- Connection to my local database
- Displaying the "Facilities" table
- Adding a record to "Facilities" using DB autoincrement
- Updating the Name and Description of a record in "Facilities"
- Deleting a record from "Facilities"

4.1.2. Out of scope

This leads me to the conclusion that I will not implement features such as adding a Facility to a Hotel through HotelFacilities. This is because it would require more work on the UI side, rather than on and in the database work.

4.1.3. However

I have added a DataGridView for the "ViewHotelFacilities", which is a view that I have created in the database. The script can be found in **??**. This is because I wanted to see how the relationship between the Hotel and the Facilities would be displayed in a DataGridView.

4.1.4. Design

The application will be a simple WinForm application with a DataGridView for displaying the "Facilities" table, and a few buttons for the CRUD operations. Classic WinForm controls will be used, and the application will be designed to be as simple as possible. The reason is the same as the one for the scope of the application: the database is the primary focus, and the application is secondary.

4.2. Implementation

A new WinForm application was created in Visual Studio 2022 using .NET 8.0. The application is named HotelFacilityApp. It has been uploaded to GitHub, and can be found at `https://github.com/Cal-ly/SWD-DB-Assigment.git` along with the rest of the project. Be aware, that the application is using a local database, and the connection string in the code (or app.config file) will need to be changed to match your local database.

4.2.1. Connection to the database

The connection to the database is established in the ConnectForm.cs, which is the first form that is shown when the application is started. See **??** for a screenshot of the form. From there the user will connect to the default database, requested to supply their own string or get an error message - **??** shows the form with a successful connection. The connection in the other "slave" forms (CreateForm and UpdateForm) is handled via the *Using* statement, which ensures that the connection is closed when the form is closed.

4.2.2. Displaying the "Facilities" table

The ConnectForm.cs contains a DataGridView that is populated with the "Facilities" table from the database. See **??** for a screenshot of the form. This is simply done using a SQL query:

```
1 SELECT * FROM Facilities
```

with the C# side handled with SqlCommand and SqlDataReader.

4.2.3. Adding a record to "Facilities" using DB autoincrement

The CreateForm.cs contains a TextBox for the Name and a TextBox for the Description. See **??** for a screenshot of the form. When the user clicks the "Add Facility" button, the data is inserted into the "Facilities" table using a SQL query:

```
1 INSERT INTO Facilities (Name, Description) VALUES (@FacilityName, @FacilityDescription)
```

See **??** and **??** for a screenshot of the form with the data inserted.

4.2.4. Updating the Name and Description of a record in "Facilities"

The ConnectForm.cs contains a DataGridView that is populated with the "Facilities" table from the database. When the user clicks a rowheader in the DataGridView, the Name and Description of the selected row is displayed in TextBoxes. See **??** for a screenshot of the form. When the user clicks the "Update" button, the data is updated in the "Facilities" table using a SQL query:

```
1 UPDATE Facilities SET Name = @FacilityName, Description = @FacilityDescription WHERE Id = @FacilityId
```

See **??** and **??** for a screenshot of the form with the data updated.

4.2.5. Deleting a record from "Facilities"

The ConnectForm.cs contains a DataGridView that is populated with the "Facilities" table from the database. When the user clicks a rowheader in the DataGridView, the Name and Description of the selected row is displayed in TextBoxes. See **??** and **??** for a screenshot of the form. When the user clicks the "Delete" button, the data is deleted from the "Facilities" table using a SQL query:

```
1 DELETE FROM Facilities WHERE Id = @FacilityId
```

4.2.6. Alternativ to Add Facility

In the UpdateForm.cs, there is also an *Add as New* button. This is a simple way to either add a new or clone it, while stille giving it a new ID.

5 Conclusion

The purpose of this assignment was to design a database for a hotel management systemm, with an application that could do CRUD operations on the Facilities table. While this has been an academic excersise, the focus has been on the process of designing and implementing a database, and not on the application (UI and UX) itself.

The report has details this process, diving into the thoughts behind designing the database, including the requirements, the design, and the implementation.

Furthermore, the report entails the process of creating a database schema and the process of implementing the database schema in a SQL database. This was enhanced upon with a WinForms application that could interact with the database.

The database has been designed and implemented according to the requirements, and the application has been tested to ensure that it can perform the CRUD operations on the Facilities table. Overall, the assignment will be regarded as a success, and the process has been a learning experience.

A

Example

A.1. Creating the Database

A.1.1. Create Database Script

This can be used for recreating the database from scratch.

```
-- Hotels
CREATE TABLE Hotels (
    HotelID INT IDENTITY (1,1) PRIMARY KEY,
    Name NVARCHAR(100),
    Address NVARCHAR(100),
        City NVARCHAR(50),
        PostalCode VARCHAR(10),
    Rating INT
);

-- Customers
CREATE TABLE Customers (
    CustomerID INT IDENTITY (1,1) PRIMARY KEY,
    Name NVARCHAR(100),
    Email VARCHAR(100),
    Phone VARCHAR(15)
);

-- Rooms
CREATE TABLE Rooms (
    RoomID INT IDENTITY (1,1) PRIMARY KEY,
    HotelID INT,
    RoomNumber VARCHAR(10),
    Type VARCHAR(50),
    Price DECIMAL(10, 2),
    FOREIGN KEY (HotelID) REFERENCES Hotels(HotelID)
);

-- Reservations
CREATE TABLE Reservations (
    ReservationID INT IDENTITY (1,1) PRIMARY KEY,
    CustomerID INT,
    RoomID INT,
    CheckInDate DATE,
    CheckOutDate DATE,
    Status VARCHAR(50),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID)
);

```

```
-- Facilities
CREATE TABLE Facilities (
    FacilityID INT IDENTITY (1,1) PRIMARY KEY,
    Name VARCHAR(100),
    Description TEXT
);

-- HotelFacilities
CREATE TABLE HotelFacilities (
    HotelFacilityID INT IDENTITY (1,1) PRIMARY KEY,
    HotelID INT,
    FacilityID INT,
    FOREIGN KEY (HotelID) REFERENCES Hotels(HotelID),
    FOREIGN KEY (FacilityID) REFERENCES Facilities(FacilityID)
);
```

A.1.2. Mock Data

This section contains the SQL code for inserting mock data into the tables. Note the use of reseeding the ID of all tables to 1, so that the mock data can be inserted without having to worry about the ID. This of course is only intended for mock data, and should not be used in a production environment. The mock data is in Danish, because the UTF-8 encoding took me a little while to troubleshoot, so I am keeping the æ, ø and å.

```
-- Delete all records
DELETE FROM Reservations;
DELETE FROM HotelFacilities;
DELETE FROM Rooms;
DELETE FROM Facilities;
DELETE FROM Customers;
DELETE FROM Hotels;

-- This resets all Identity to 0, so the next entry is 1
DBCC CHECKIDENT ('Hotels', RESEED, 0);
DBCC CHECKIDENT ('Customers', RESEED, 0);
DBCC CHECKIDENT ('Rooms', RESEED, 0);
DBCC CHECKIDENT ('Reservations', RESEED, 0);
DBCC CHECKIDENT ('Facilities', RESEED, 0);
DBCC CHECKIDENT ('HotelFacilities', RESEED, 0);

-- Insert mock data into Hotels
INSERT INTO Hotels (Name, Address, City, PostalCode, Rating) VALUES
('Scandic Hotel', 'Ved Ringen 2', 'Roskilde', '4000', 5),
('Bjergflugt', 'Højlandet Alle 34', 'Bjergby', '3400', 4),
('Byhotel', 'Centrum Boulevard 56', 'Storby', '4500', 4);

-- Insert mock data into Customers
INSERT INTO Customers (Name, Email, Phone) VALUES
('Jens Hansen', 'jens.hansen@example.com', '10203040'),
('Mette Nielsen', 'mette.nielsen@example.com', '40506070'),
('Lars Andersen', 'lars.andersen@example.com', '70809010'),
('Lars Larsen', 'lars.larsen@mail.com', '10405060'),
('Laila Larsen', 'laila.larsen@mail.com', '40302010');

```

```sql
-- Insert mock data into Rooms
INSERT INTO Rooms (HotelID, RoomNumber, Type, Price) VALUES
(1, '101', 'Enkelt', 750.00),
(1, '102', 'Dobbelt', 1125.00),
(1, '103', 'Suite', 1875.00),
(2, '101', 'Enkelt', 750.00),
(2, '102', 'Dobbelt', 1125.00),
(2, '103', 'Suite', 1875.00),
(3, '101', 'Enkelt', 750.00),
(3, '102', 'Dobbelt', 1125.00),
(3, '103', 'Suite', 1875.00);

-- Insert mock data into Reservations
INSERT INTO Reservations (CustomerID, RoomID, CheckInDate, CheckOutDate, Status) VALUES
(1, 1, '2024-04-01', '2024-04-05', 'Bekræftet'),
(2, 2, '2024-04-10', '2024-04-12', 'Bekræftet'),
(3, 3, '2024-05-01', '2024-05-03', 'Bekræftet'),
(4, 5, '2024-05-24', '2024-05-26', 'Bekræftet'),
(5, 9, '2024-05-24', '2024-05-26', 'Bekræftet');

-- Insert mock data into Facilities
INSERT INTO Facilities (Name, Description) VALUES
('Svømmepøl', 'Udendørs pool med liggestole og bar service'),
('Spa', 'Fuld-service spa der tilbyder massage, ansigtsbehandlinger, og kropsbehandlinger'),
('Fitnessrum', '24/7 adgang til træningsudstyr og frie vægte'),
('Børnebasin', 'Udendørs børnepool med legetøj'),
('Bar', 'All inclusive bar med stort udvalg'),
('Wellness Area', 'Med spa og haram');

-- Insert mock data into HotelFacilities
INSERT INTO HotelFacilities (HotelID, FacilityID) VALUES
(1, 1),
(1, 2),
(2, 3),
(2, 4),
(3, 5),
(3, 6);
```

A.1.3. View Hotel Facilities

The following is the SQL code for creating a view that shows the relationship between the Hotel and the Facilities.

```sql
CREATE VIEW HotelsWithFacilities AS
SELECT
    h.HotelID,
    h.Name AS HotelName,
    f.FacilityID,
    f.Name AS FacilityName,
    f.Description
FROM
    Hotels h
JOIN
    HotelFacilities hf ON h.HotelID = hf.HotelID
JOIN
    Facilities f ON hf.FacilityID = f.FacilityID;
```

```
14
15 SELECT * FROM HotelsWithFacilities;
```

B

Screen Shots from the Application

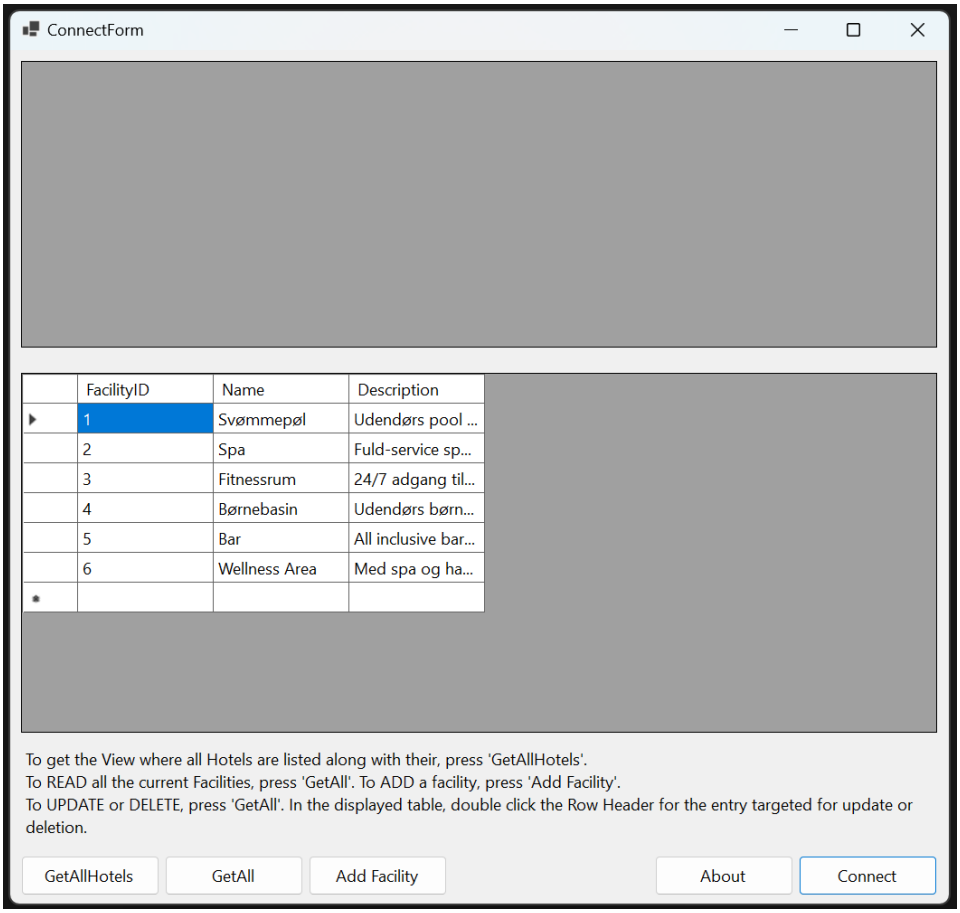

Figure B.1: The main screen of the application.

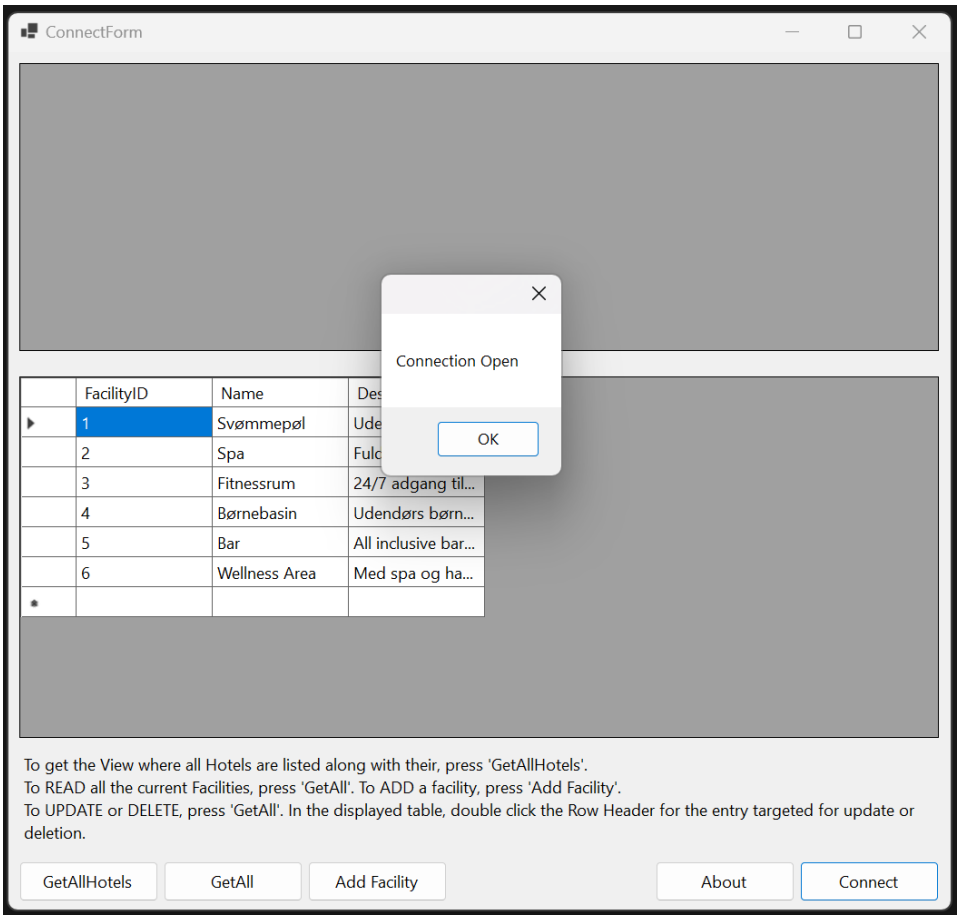

Figure B.2: Connection open.

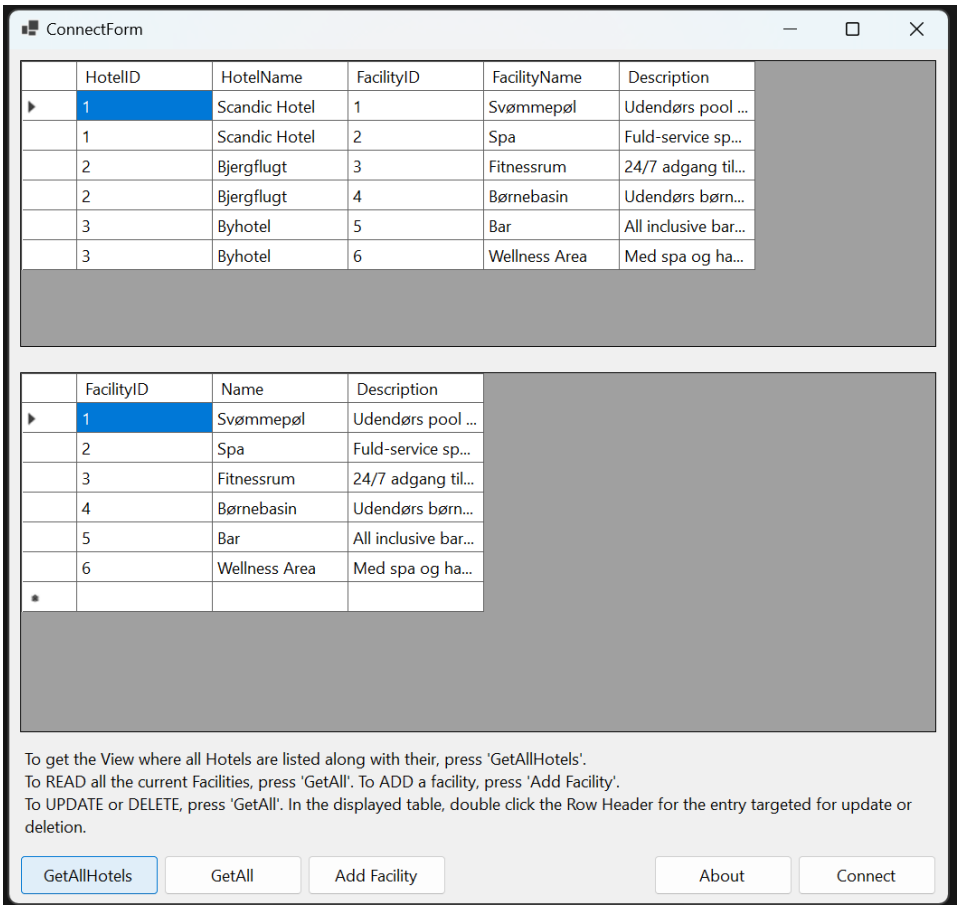

Figure B.3: Get all hotels and their facilities.

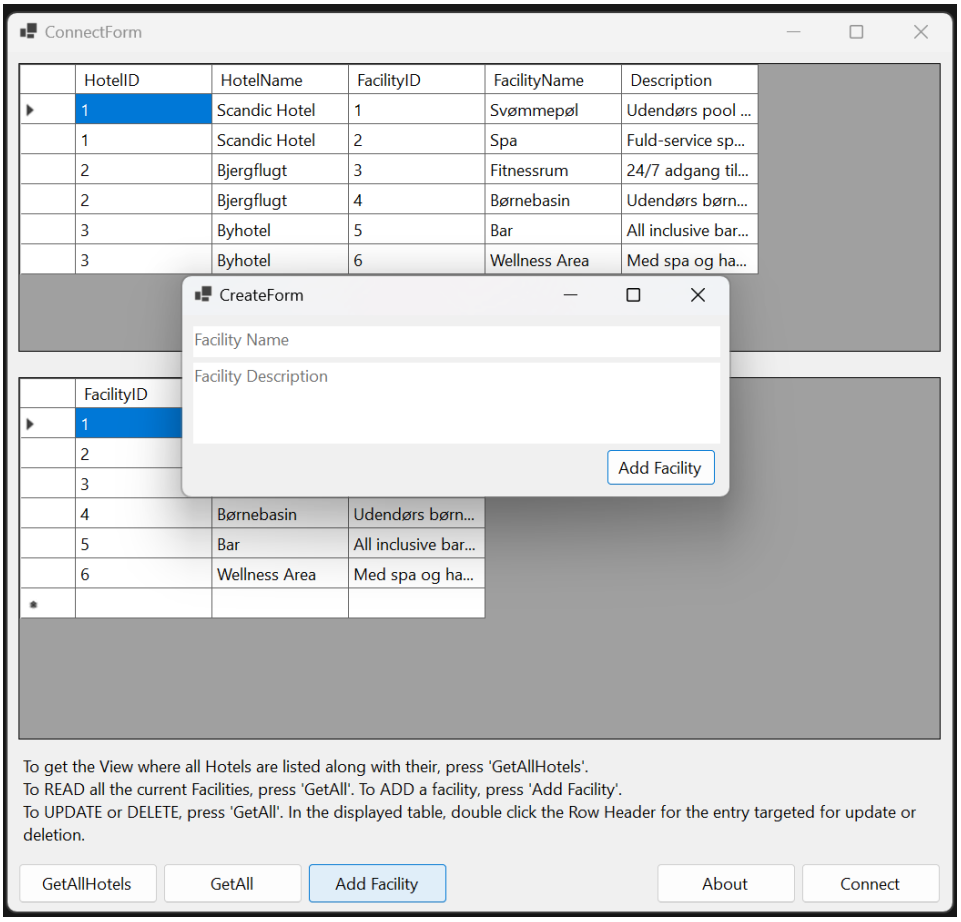

Figure B.4: Add a new facility.

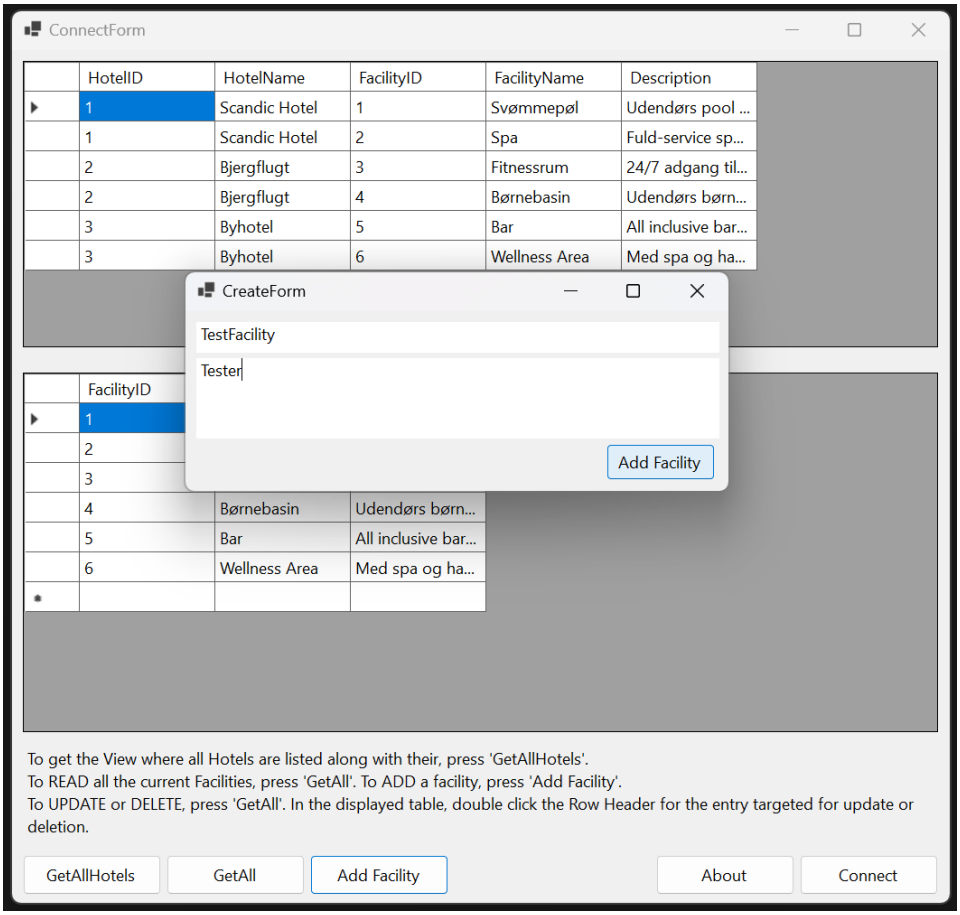

Figure B.5: Adding facility.

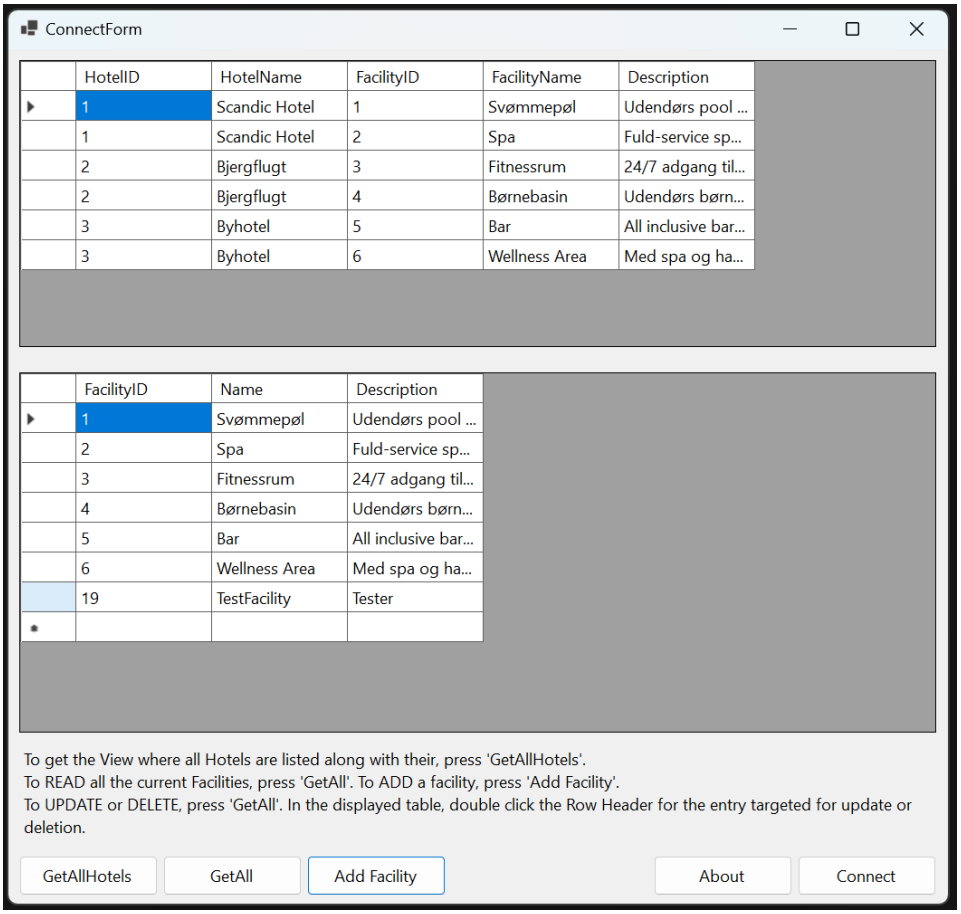

Figure B.6: Facility added.

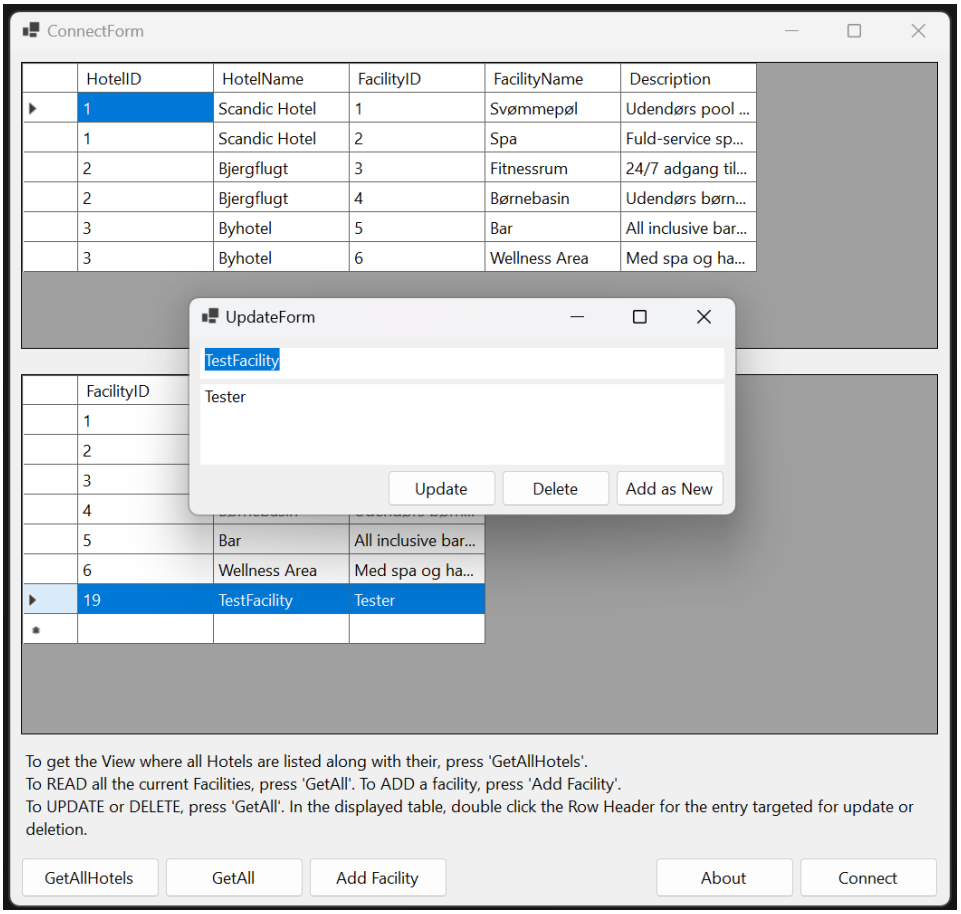

Figure B.7: Update a facility.

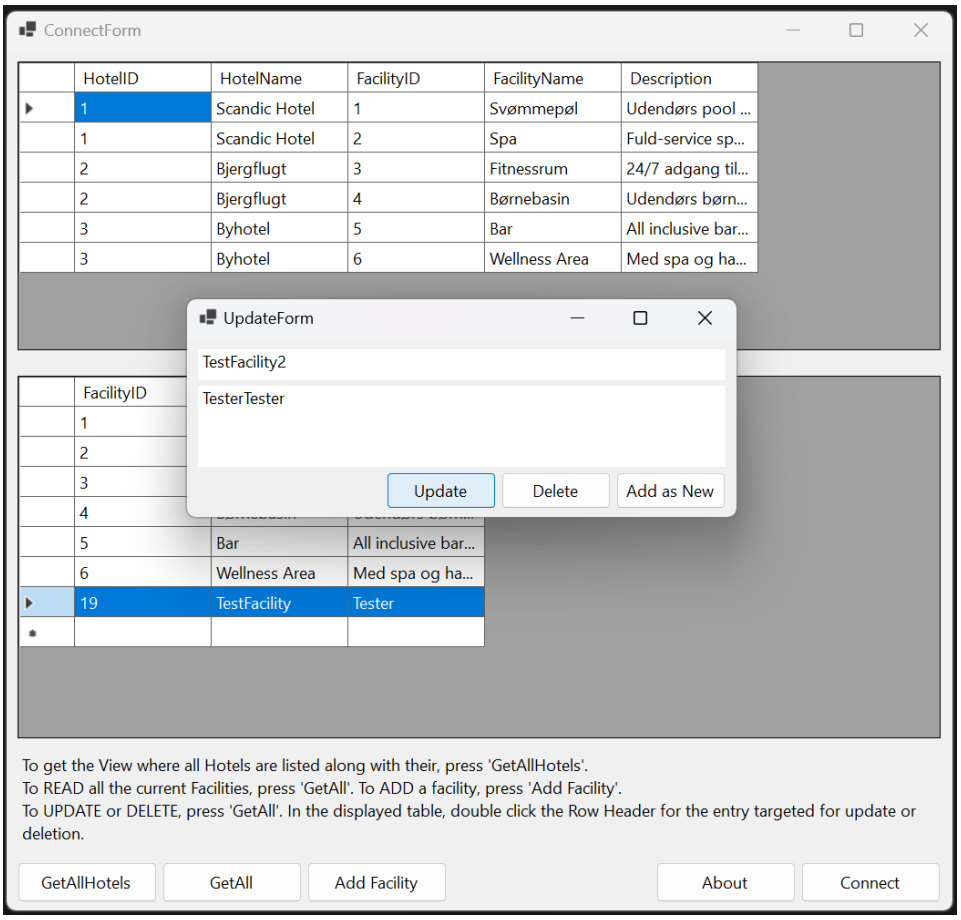

Figure B.8: Updating facility.

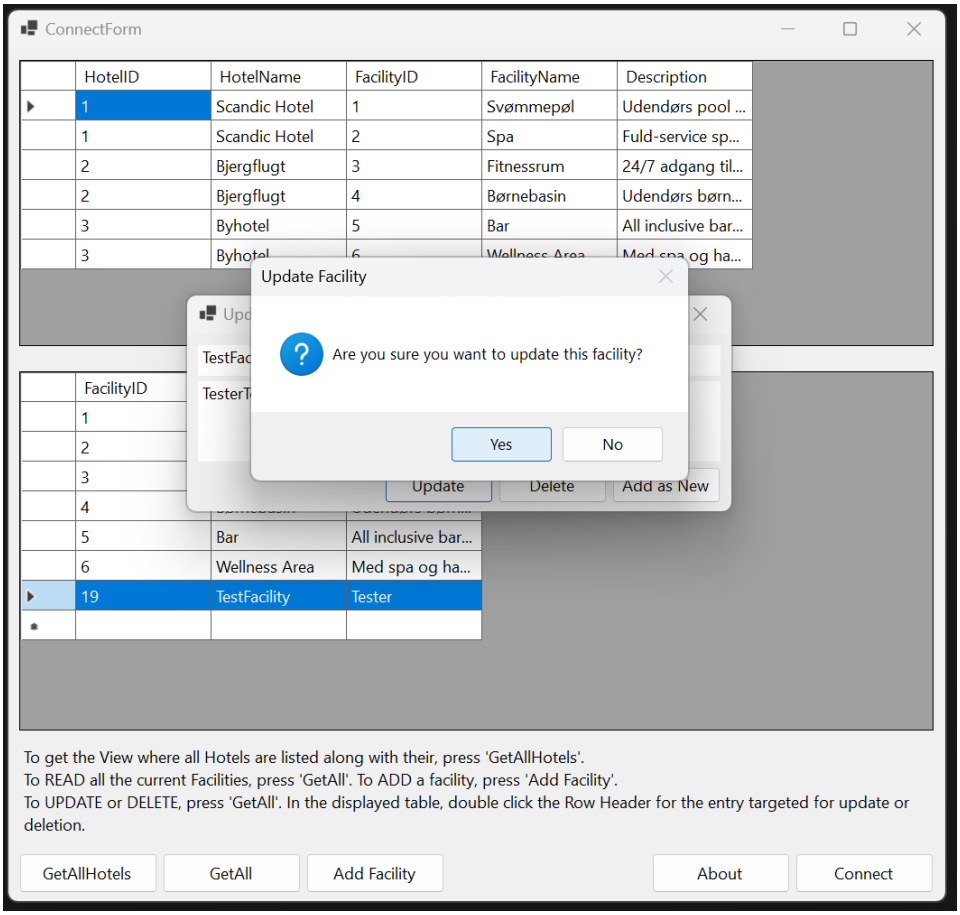

Figure B.9: Facility updated.

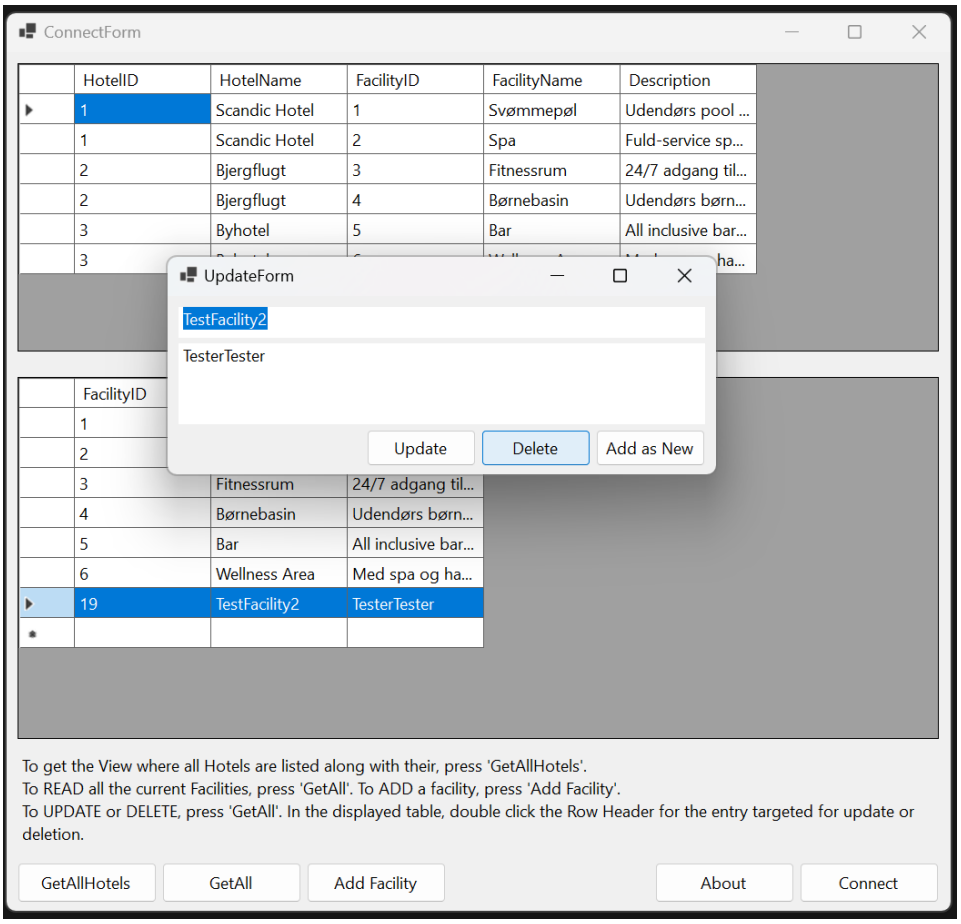

Figure B.10: Delete a facility.

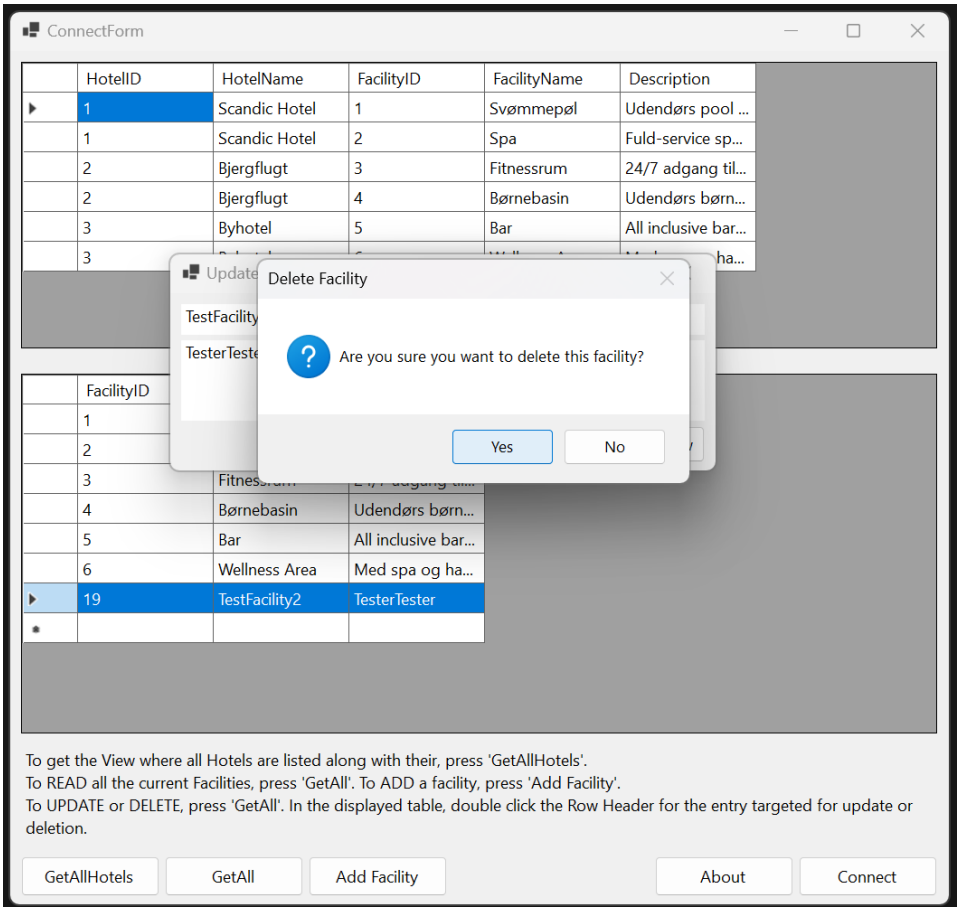

Figure B.11: Deleting facility.