|  |
| --- |
|  |
| Assignment 1- Design and Implement a Database  Unit Code: CIS017-1 |
| |  |  |  | | --- | --- | --- | | [Author name] | StudentID: | Computer System Structure | |

Contents

[Introduction 1](#_Toc57216891)

[Scenario 1](#_Toc57216892)

[Tasks: 2](#_Toc57216893)

[1) Entity Relationship Model 2](#_Toc57216894)

[2) Normalization 3](#_Toc57216895)

[3) Physical Table Design (Including Data Dictionary) 4](#_Toc57216896)

[Part A- Skeleton Tables: 4](#_Toc57216897)

[Part B – Data Dictionary: 4](#_Toc57216898)

[4) Query Design 7](#_Toc57216899)

[Part A: Queries and Justification 7](#_Toc57216900)

[Part B – SQL Queries 7](#_Toc57216901)

[5) Implementation of the Database 9](#_Toc57216902)

[Part A – Creating tables 9](#_Toc57216903)

[Part B: Insert Data 10](#_Toc57216904)

[Part C: SQL Queries 11](#_Toc57216905)

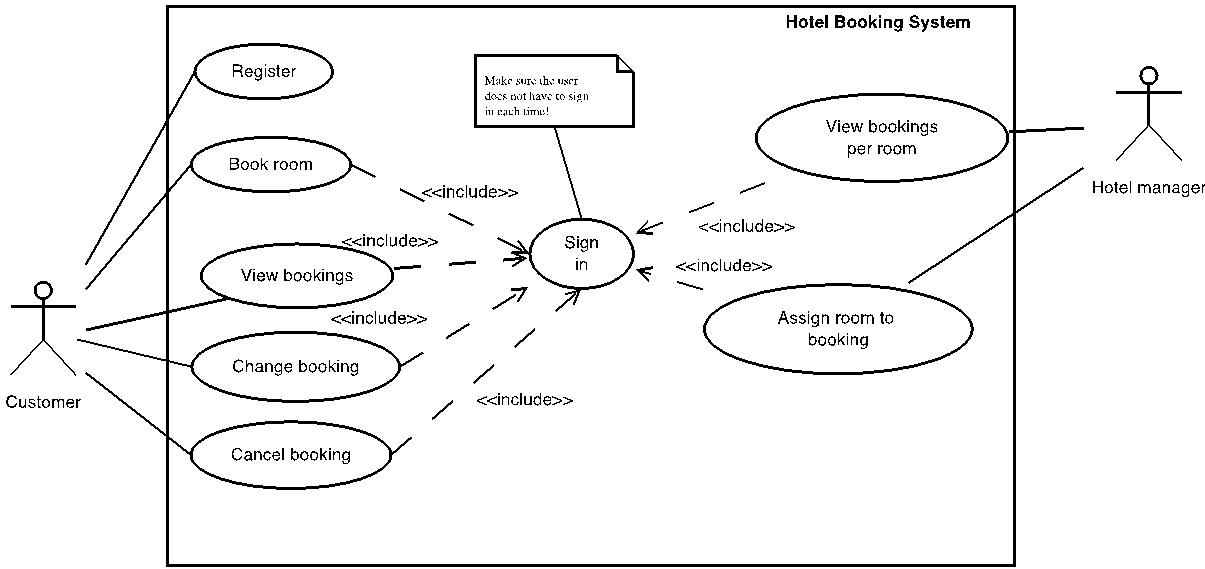
[Discussion and reflection 13](#_Toc57216906)

# **Introduction**

This report highlights the design and implementation of a relational database for the given scenario. The tasks undertaken in this assignment were: Entity-Relationship Model, Normalization, Physical Table Design (including Data Dictionary), Query Design and Implementation of the database.

## **Scenario**

A hotel needs a relational database for its registered customers and hotel managers. Once a customer is registered, they can, book a room, view bookings and change or cancel bookings if needed. The hotel managers should also be able to see existing bookings per room, assign rooms to each booking. Both registered customers and hotel managers must sign into the relational database before any operation can be performed. The system checks if the requested room is available for the time of stay.

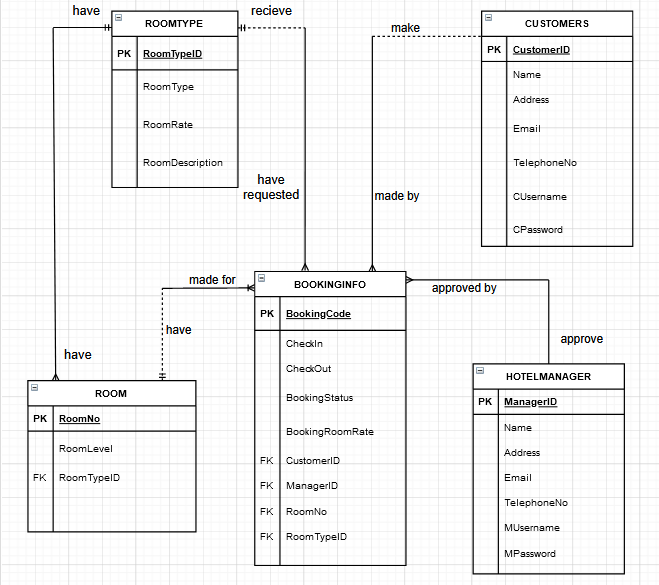


**Figure:** Hotel Booking System Use Case Diagram

# Tasks:

## Entity Relationship Model

Assumptions that were made for the entity relationship model are that customers can register but not make a booking to the hotel, thus it is optional for customers to make a booking. Second assumption is all hotel managers are required to approve all bookings and all bookings made by the customers will be approved by the hotel manager. The BookingRoomRate attribute is the price of the room at the time of booking because it is assumed that the price at booking can be less than usual depending on whether it was booked during a sale. Thirdly, bookings made for a room are mandatory but not all rooms will be booked at some point in time because customers will not prefer that room, thus making it optional. Fourth assumption is that when customers are booking, they must request one room type but not all room types may receive a booking because not all rooms are booked. Lastly, it is assumed that a room must have only one room type at a time and a room type can have many rooms thus making them both mandatory.



**Figure 1:** Entity Relationship Model for Hotel Booking System.

To create a relational database for the hotel booking system, the following entities were created: CUSTOMRS, BOOKINGINFO, HOTELMANAGER, ROOM and ROOMTYPE.

The relationship between CUSTOMER and BOOKINGINFO is a CUSTOMER may make many BOOKINGINFO and a BOOKINGINFO must be made by one CUSTOMER. Due to this relationship, CustomerID becomes a foreign key in the BOOKINGINFO entity and this is seen in figure 1.

From the case study we know that booking can only book one room thus the relationship is a BOOKINGINFO must be made for one ROOM and a ROOM may have many BOOKINGINFO. Consequently, RoomNo becomes a foreign key in the BOOKINGINFO entity.

The relationship between HOTELMANAGER and BOOKINGINFO is a HOTELMANAGER must approve of many BOOKINGINFO and each BOOKINGINFO must be approved by one HOTELMANAGER. Therefore, ManagerID becomes a foreign key in the BOOKINGINFO entity.

ROOMTYPE relationship with BOOKINGINFO is that each BOOKINGINFO must have requested one ROOMTYPE and a ROOMTYPE may receive many BOOKINGINFO. Therefore, RoomType becomes a foreign key in the BOOKINGINFO entity. ROOMTYPE also has a relationship with ROOM. The relationship is each ROOMTYPE must have many ROOM and a ROOM must have one ROOMTYPE, thus making RoomType the foreign key in the ROOM entity.

## Normalization

|  |  |  |  |
| --- | --- | --- | --- |
| **UNF** | **1NF** | **2NF** | **3NF** |
| **CustomerID** | **CustomerID** | **CustomerID** | **CustomerID** |
| CName | CName | CName | CName |
| CAddress | CAddress | CAddress | CAddress |
| CEmail | CEmail | CEmail | CEmail |
| CTelephoneNo | CTelephoneNo | CTelephoneNo | CTelephoneNo |
| CUsername | CUsername | CUsername | CUsername |
| CPassword | CPassword | CPassword | CPassword |
| BookingCode |  |  |  |
| CheckIn | **BookingCode** | **BookingCode** | **BookingCode** |
| CheckOut | CheckIn | CheckIn | CheckIn |
| BookingStatus | CheckOut | CheckOut | CheckOut |
| BookingRoomRate | BookingStatus | BookingStatus | BookingStatus |
| RoomNo | BookingRoomRate | BookingRoomRate | BookingRoomRate |
| RoomLevel | RoomNo | RoomNo | CustomerID\* |
| RoomTypeID | RoomLevel | RoomLevel | ManagerID\* |
| RoomType | RoomTypeID | RoomTypeID | RoomNo\* |
| RoomRate | RoomType | RoomType | RoomTypeID\* |
| RoomDescription | RoomRate | RoomRate |  |
| ManagerID | RoomDescription | RoomDescription | **ManagerID** |
| MName | ManagerID | ManagerID | MName |
| MAddress | MName | MName | MAddress |
| MEmail | MAddress | MAddress | MEmail |
| MTelephoneNo | MEmail | MEmail | MTelephoneNo |
| MUsername | MTelephoneNo | MTelephoneNo | MUsername |
| MPassword | MUsername | MUsername | MPassword |
|  | MPassword | MPassword |  |
|  | CustomerID\* | CustomerID\* | **RoomNo** |
|  |  |  | RoomLevel |
|  |  |  | RoomTypeID\* |
|  |  |  |  |
|  |  |  | **RoomTypeID** |
|  |  |  | RoomType |
|  |  |  | RoomRate |
|  |  |  | RoomDescription |

## Physical Table Design (Including Data Dictionary)

### Part A- Skeleton Tables:

CUSTOMER (CustomerID, Name, Address, Email, TelephoneNo, Username, Password,)

BOOKINGINFO (BookingCode, CheckIn, CheckOut, BookingStatus, BookingRoomRate, CustomerID\*, ManagerID\*, RoomNo\*, RoomTypeID\*)

HOTELMANAGER (ManagerID, Name, Address, Email, TelephoneNo, Username, Password)

ROOM (RoomNo, RoomLevel, RoomTypeID\*)

ROOMTYPE (RoomTypeID, RoomType, RoomRate, RoomDescription)

### Part B – Data Dictionary:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CUSTOMERS | | | | | | | |
| Description: Customer details | | | | | | | |
| Field Name | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| CustomerID | integer | 10 | PK | No |  |  | Autoincremented Uniquely identifies every customer |
| Name | varchar (100) | 100 |  | No |  |  | Name of customer |
| Address | varchar (100) | 100 |  | No |  |  | Address of customer |
| Email | varchar (100) | 100 |  | No |  | Must be email format containing an @ and a ‘.’  Regex expression used | Email of customer |
| TelephoneNo | Integer | 15 |  | No |  | The format is without “-“and spaces.  Example: 18685689087 | Customer’s telephone number |
| CUsername | varchar (30) | 30 |  | No |  |  | Customer’s username to sign-in |
| CPassword | varchar(30) | 30 |  | No |  |  | Customer’s password to sign-in |

Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | CustomerID | No |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| BOOKINGINFO | | | | | | | |
| Description: Customer’s booking information | | | | | | | |
| Field Name | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| BookingCode  *(Primary)* | varchar(10) | 10 | PK | No |  |  | Autoincremented Unique code for customer’s booking |
| CheckIn | date | 10 |  | No |  |  | The date the customer is suppose to arrive to the hotel |
| CheckOut | date | 10 |  | No |  |  | The date the customer will leave the hotel |
| BookingStatus | varchar (10) | 10 |  | No |  | Must be email format containing an @ and a ‘.’  Regex expression used | States whether this booking was Confirmed, Cancelled or Pending. |
| BookingRoomRate | real |  |  | No |  |  | The cost of the room during booking |
| ManagerID  *(Foreign)* | integer |  | FK | Yes |  |  | Autoincremented Unique code for hotel managers |
| RoomNo  *(Foreign)* | integer |  | FK | Yes |  |  | Autoincremented Unique code for the room number |
| CustomerID  *(Foreign)* | integer |  | FK | No |  |  | Autoincremented Unique code for the customer |
| RoomTypeID  (*Foreign)* | varchar (10) | 10 | FK | No |  |  | Autoincremented Unique code for the room type. |

Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | BookingCode | No |
| FOREIGN | BTREE | Yes | ManagerID | No |
| FOREIGN | BTREE | Yes | RoomNo | No |
| FOREIGN | BTREE | Yes | CustomerID | No |
| FOREIGN | BTREE | Yes | RoomTypeID | No |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| HOTELMANAGER | | | | | | | |
| Description: Hotel Manager’s detail | | | | | | | |
| Field Name | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| ManagerID  *(Primary)* | integer | 10 | PK | No |  |  | Autoincremented Uniquely identifies every hotel manager |
| Name | varchar (100) | 100 |  | No |  |  | Name of hotel manager |
| Address | varchar (100) | 100 |  | No |  |  | Address of hotel manager |
| Email | varchar (100) | 100 |  | No |  | Must be email format containing an @ and a ‘.’  Regex expression used | Email of hotel manager |
| TelephoneNo | varchar(15) | 15 |  | No |  | The format is without ” –“ and spaces.  Example: 18685689087 | Hotel manager’s telephone number |
| MUsername | varchar (30) | 30 |  | No |  |  | Manager’s username to sign-in |
| MPassword | Varchar(30) | 30 |  | No |  |  | Manager’s password to sign-in |

Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | ManagerID | No |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ROOM | | | | | | | |
| Description: The rooms in the hotel | | | | | | | |
| Field Name | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| RoomNo  *(Primary)* | integer | 10 | PK | No |  |  | Autoincremented Uniquely identifies every room |
| RoomLevel | varchar (5) | 100 |  | No |  |  | The hotel floor level where the room is located |
| RoomTypeID  *(Foreign)* | varchar (5) | 100 | FK | No |  |  | Autoincremented Uniquely identifies every room type |

Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | RoomNo | No |
| FOREIGN | BTREE | Yes | RoomType | No |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ROOMTYPE | | | | | | | |
| Description: The types of rooms in the hotel | | | | | | | |
| Field Name | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| RoomTypeID  *(Primary)* | varchar(10) | 10 | PK | No |  |  | Autoincremented Uniquely identifies every room type |
| RoomType | varchar (10) | 10 |  | No |  |  | The type of room the customer can request. Either single or double. Unique |
| RoomRate | real | 10 |  | No |  |  | The cost of the room for the check-in and check-out dates |
| RoomDescription | varchar (200) | 200 |  | Yes |  |  | A description of the room. |

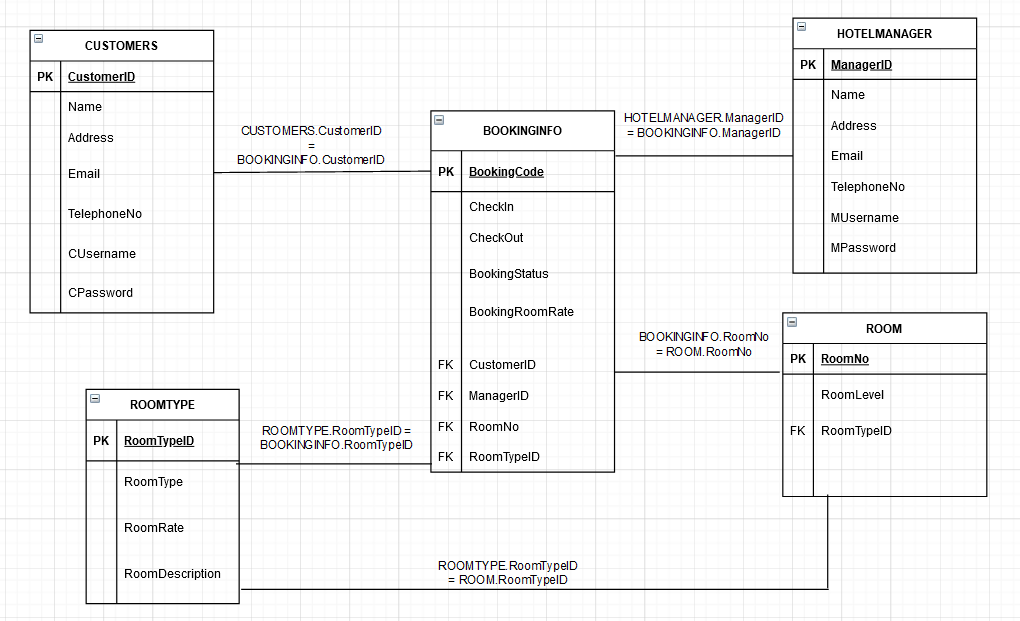
Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | RoomTypeID | No |

## Query Design

### Part A: Queries and Justification

|  |  |  |
| --- | --- | --- |
| # | Query | Justification |
| 1 | List of bookings that are pending. | Hotel managers are required to assign rooms to customers based on their booking information thus this query will allow hotel managers to find all bookings that are pending and require a room. |
| 2 | A list of Bookings that have been cancelled. | Hotel managers need to view all bookings that have been cancelled so that the rooms can be rebooked by other customers. |
| 3 | List Hotel managers that assigned confirmed rooms to customers. | The hotel can use this query to see which the hotel manager assigned the most customers can make them the employee of the month. |
| 4 | The number of rooms per room type that has confirmed booking | The hotel would want to know how much rooms for each room type is being booking as part of capacity planning and to see which room type receives the most bookings. |
| 5 | The top customer based on booking room rate. | The hotel can reward their top customer as a promotion pitch. |
| 6 | Uses a customer’s name to find their booking information. | A customer may forget their room number and ask the front desk for this information so the query will allow hotel managers to find a customer’s booking information by using their name. |

****

**Figure 2:** Shows the join conditions for each entity table

### Part B – SQL Queries

1. **List of bookings that are pending.**

**SELECT** c.CustomerID, c.Name, b.BookingCode, b.CheckIn, b.CheckOut, b.BookingStatus

**FROM** CUSTOMERS c, BOOKINGINFO b

**WHERE** c.CustomerID = b.CustomerID

**AND** BookingStatus = 'Pending'

**AND** RoomNo **IS NULL**

**AND** ManagerID **IS NULL**

This query allows hotel managers to see bookings that are pending because they are required to assign rooms to customers.

1. **Bookings that have been cancelled so that they can be rebooked.**

**SELECT**  BOOKINGINFO.BookingCode, BOOKINGINFO.CheckIn, BOOKINGINFO.CheckOut,

BOOKINGINFO.CustomerID, CUSTOMERS.Name

**FROM**  BOOKINGINFO, CUSTOMERS

**WHERE** BOOKINGINFO.CustomerID = CUSTOMERS.CustomerID

**AND**  BookingStatus = 'Cancelled'

This query allows hotel managers to see bookings that were cancelled so that the rooms can be rebooked by new customers.

1. **Which customers, hotel managers are assigned to for bookings**

**SELECT**  b.BookingCode, b.CustomerID, c.Name, b.ManagerID, hm.Name

**FROM** CUSTOMERS c, BOOKINGINFO b, HOTELMANAGER hm

**WHERE** c.CustomerID = b.CustomerID AND hm.ManagerID = b.ManagerID

**AND** BookingStatus = 'Confirmed'

**ORDER BY** b.BookingCode = DESC

Each hotel manager must assign a room to a customer thus this query lets hotel managers see what customers they assigned rooms to and the booking was approved. The booking code is also arranged in descending order.

1. **The number of rooms per room type.**

**SELECT**  RoomType, count(\*) as NumberofRooms

**FROM** ROOMTYPE, BOOKINGINFO

**WHERE**  BOOKINGINFO.RoomTypeID = ROOMTYPE.RoomTypeID

**AND**  BOOKINGINFO.BookingStatus = 'Confirmed'

**GROUP BY** RoomType

This query allows the hotel to see how much rooms are being booked by each room type to see which room type is being booked more.

1. **The top customer.**

**SELECT**  c.CustomerID, Name, **COUNT(\*), SUM(BookingRoomRate)**

**FROM**  CUSTOMERS c, BOOKINGINFO b

**WHERE** c.CustomerID = b.CustomerID

**AND**  BookingStatus <> 'Cancelled'

**GROUP BY** c.CustomerID, c.Name

**ORDER BY** **SUM**(BookingRoomRate) **DESC**

This query lets the hotel find the top customer by calculating the sum of each customer’s booking room rate. The sum of the customer’s booking room rate in arranged in descending order.

1. **Uses a customer’s name to find they booking information**

**SELECT**  CustomerID, BookingCode, CheckIn, CheckOut, BookingRoomRate, RoomNo

**FROM** BOOKINGINFO

**WHERE** BOOKINGINFO.CustomerID **IN**

(**SELECT** CUSTOMERS.CustomerID

**FROM** CUSTOMERS

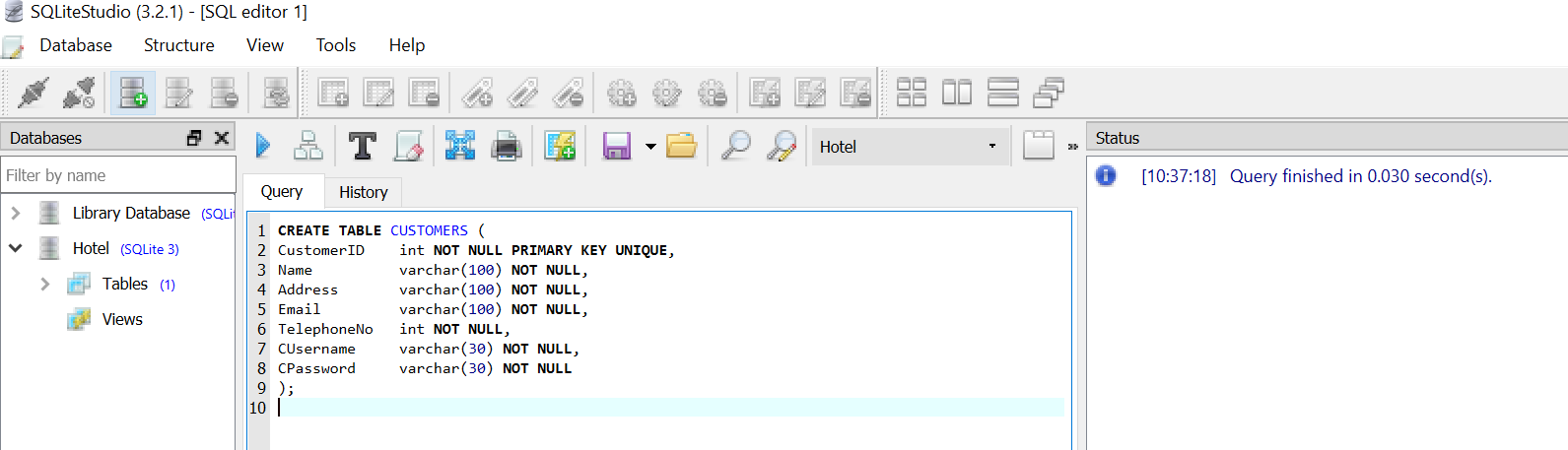
**WHERE** Name = 'James Bond')

**ORDER BY** BookingCode = DESC

This query finds the booking information of a customer when using a given name. James Bond is used as an example but any name can be used.

## Implementation of the Database

### Part A – Creating tables

****

**Figure1:**Creation of CUSTOMER table

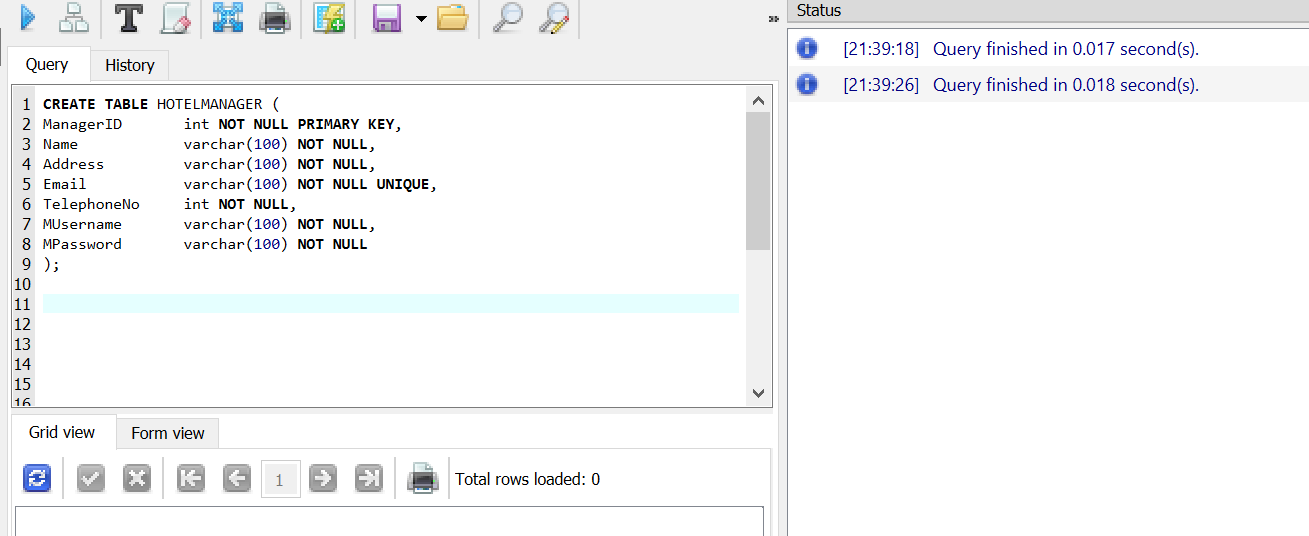


Figure 2: Creation of HOTELMANAGER TABLE

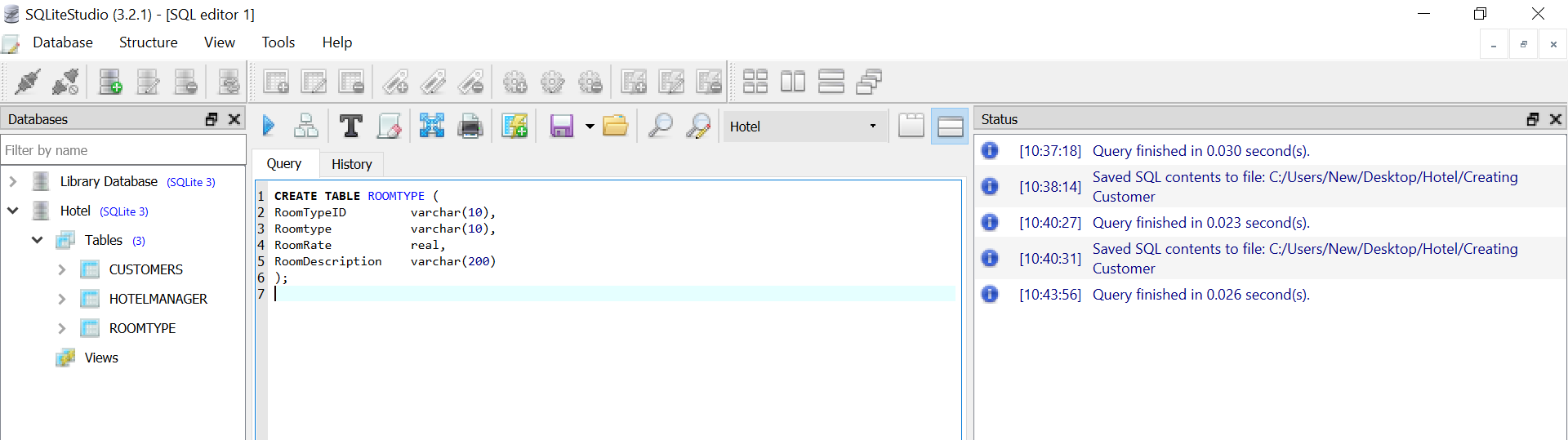


Figure 3: Creation of ROOMTYPE table

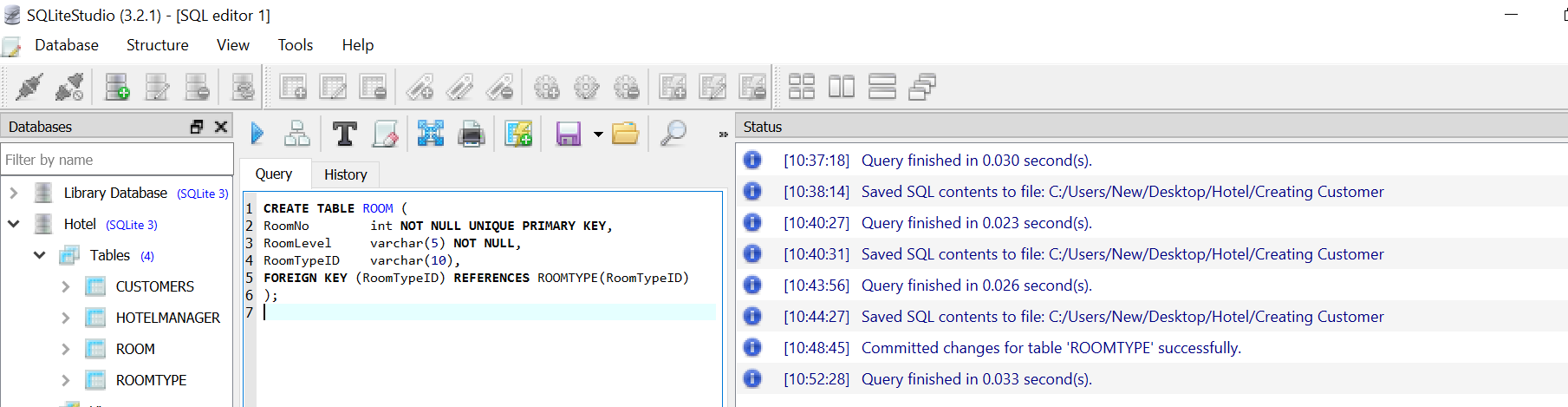
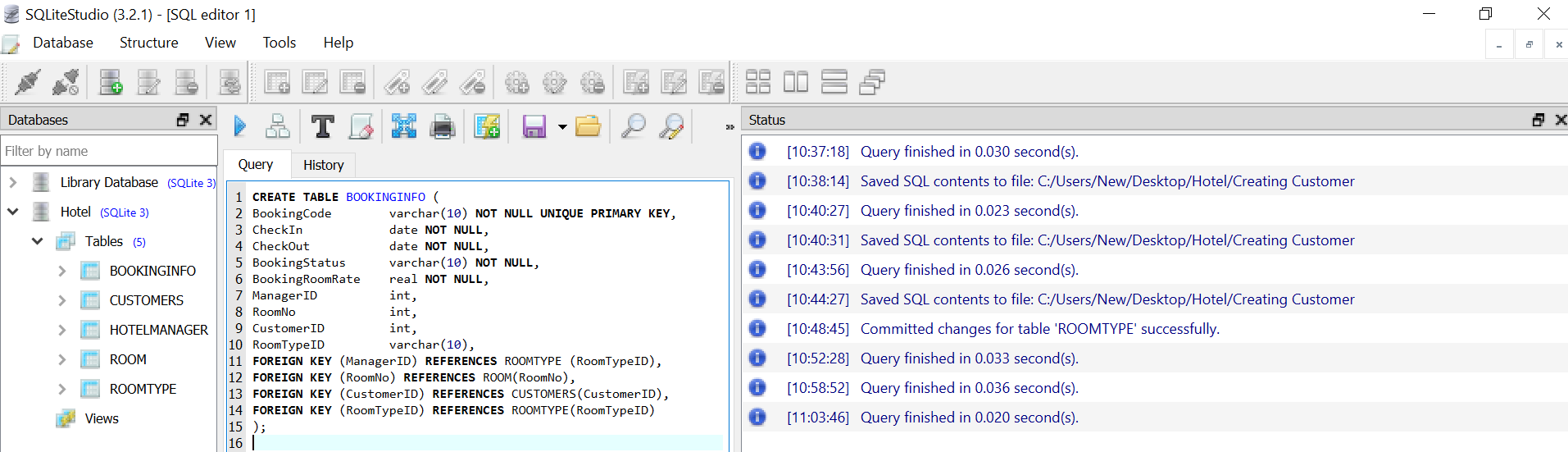
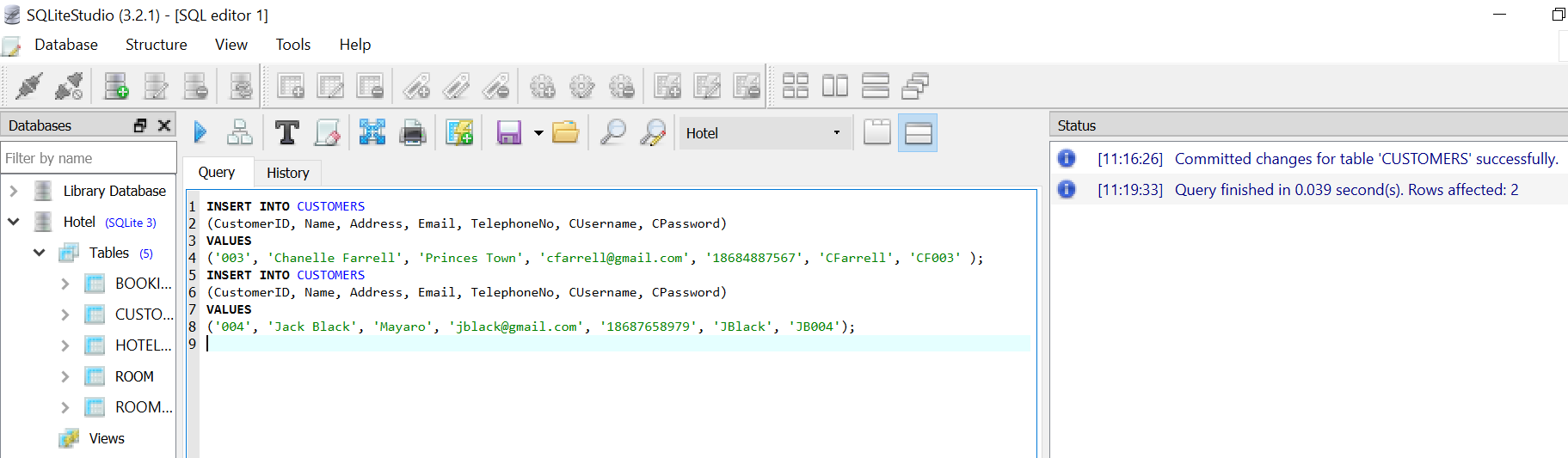


Figure 4: Creation of ROOM table

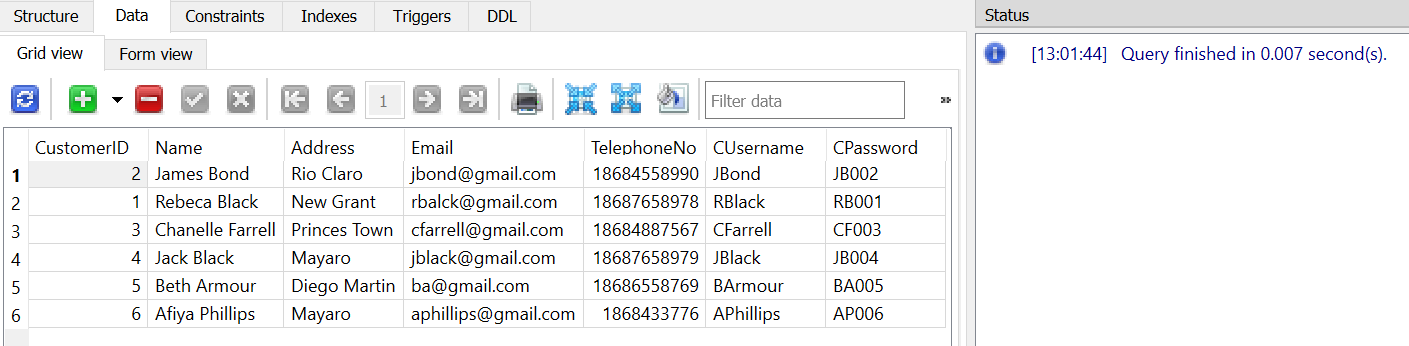


**Figure 5:** Creation of BOOKINGINFO table

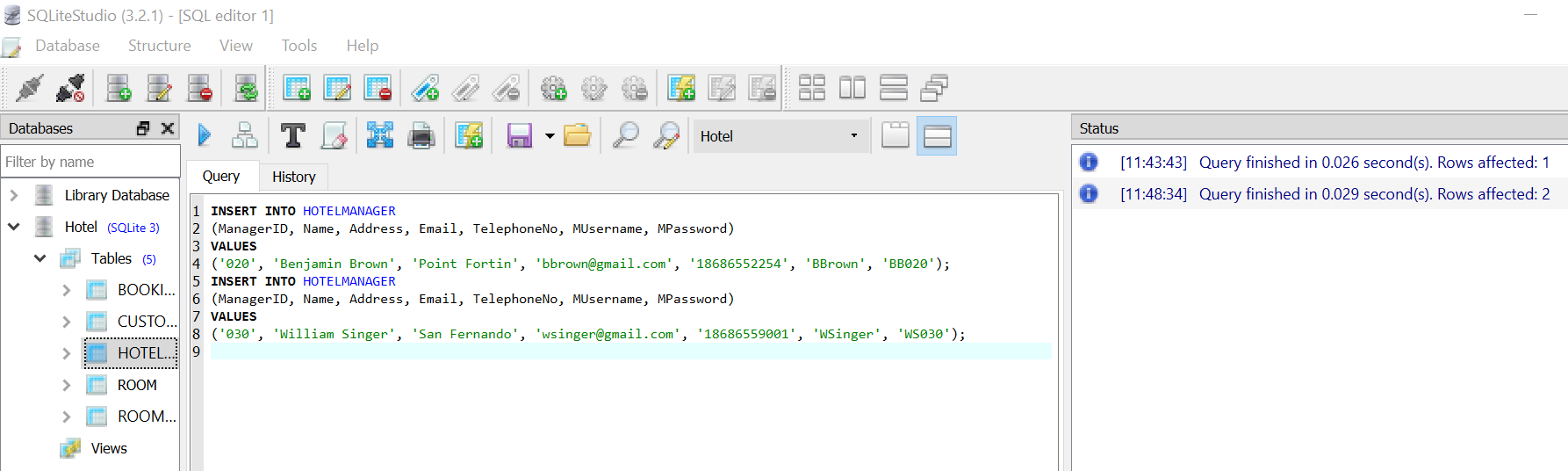
### Part B: Insert Data



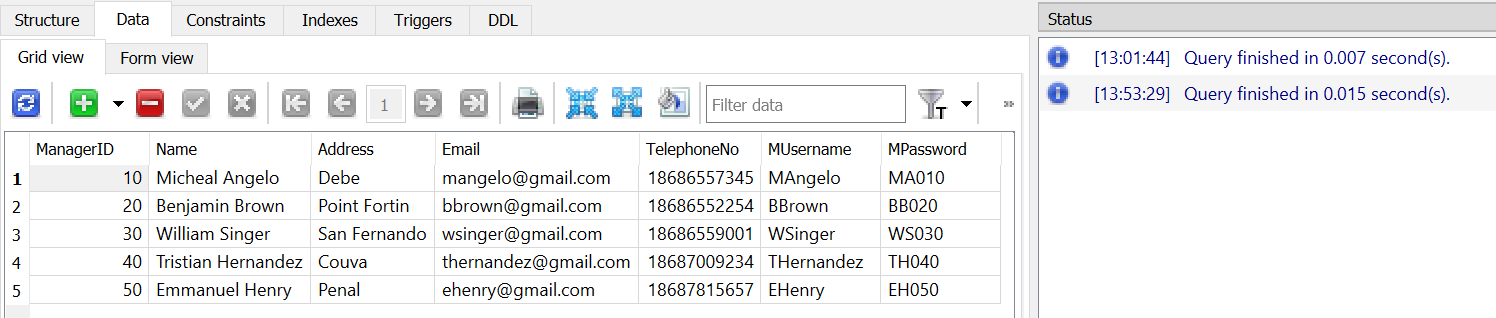
**Figure 1A:** Inserting two rows of data into the CUSTOMER table

****

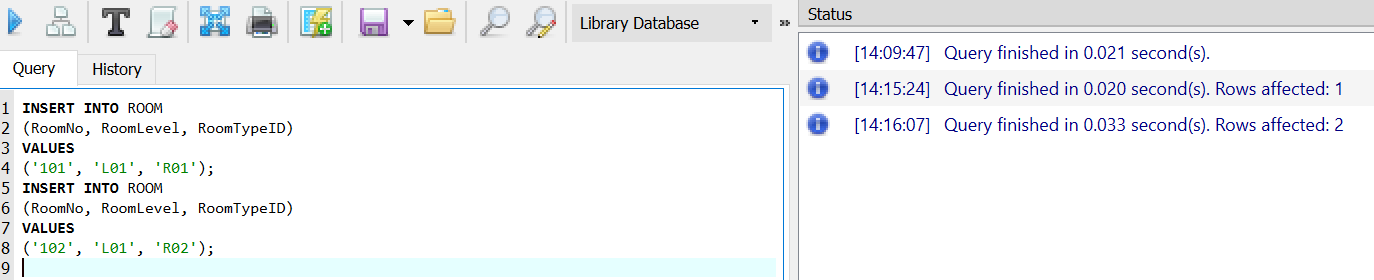
**Figure 1B:** The data in CUSTOMER table



**Figure 2A:** Inserting two rows of data into the HOTELMANAGER table.



**Figure 2B**: The data in the HOTELMANAGER table



**Figure 3A:** Inserting two rows of data into the ROOM table

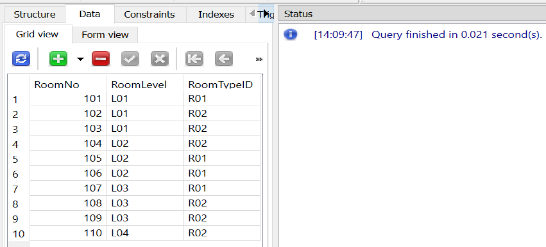
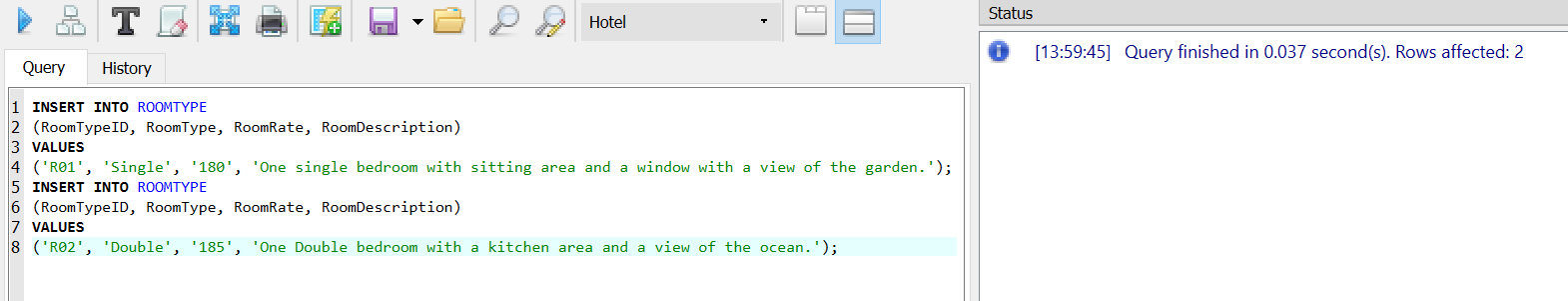
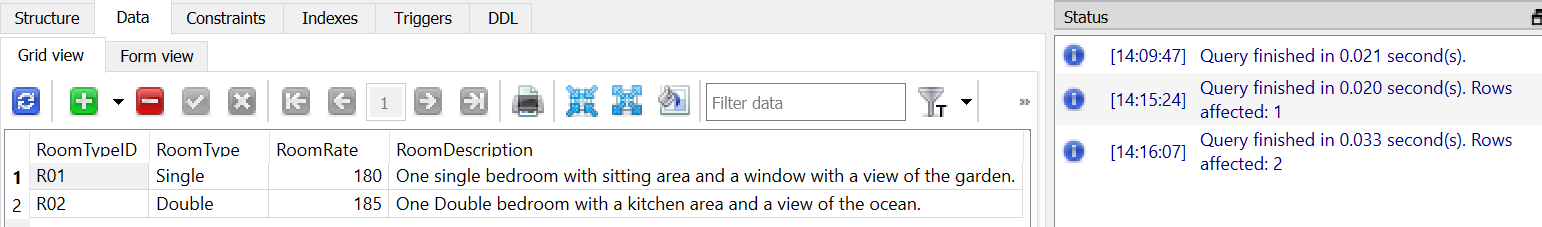


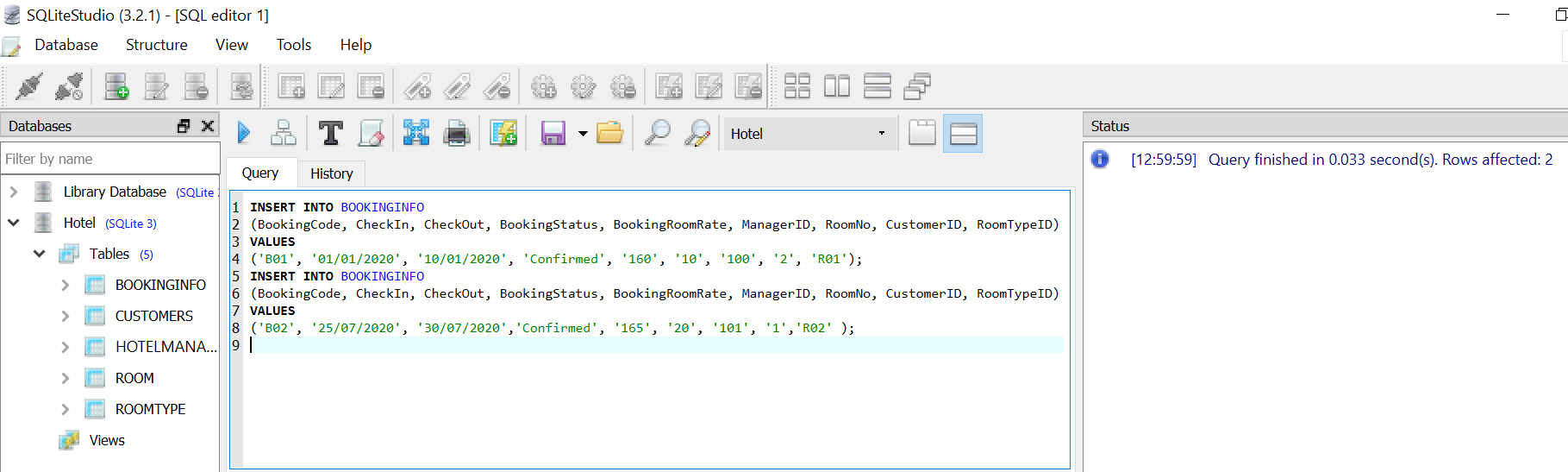
Figure 3B: The data in ROOM table.



**Figure 4A:** Inserting two rows of data into the ROOMTYPE table



**Figure 4B:** The data in ROOMTYPE table



**Figure 5A:** Inserting two rows of data into the BOOKINGINFO table

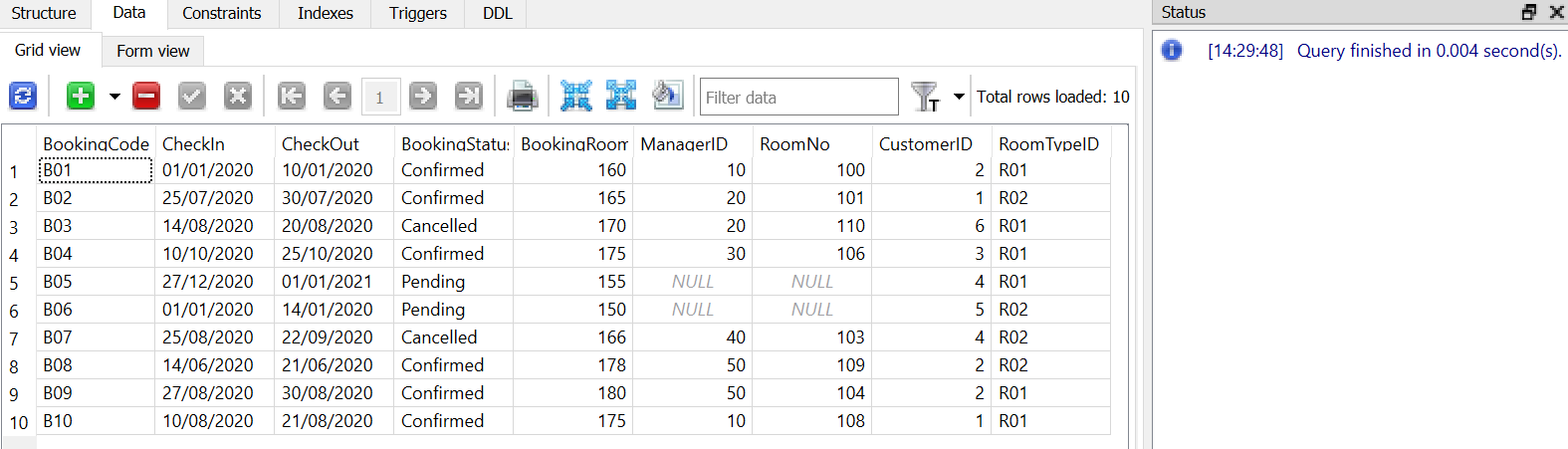
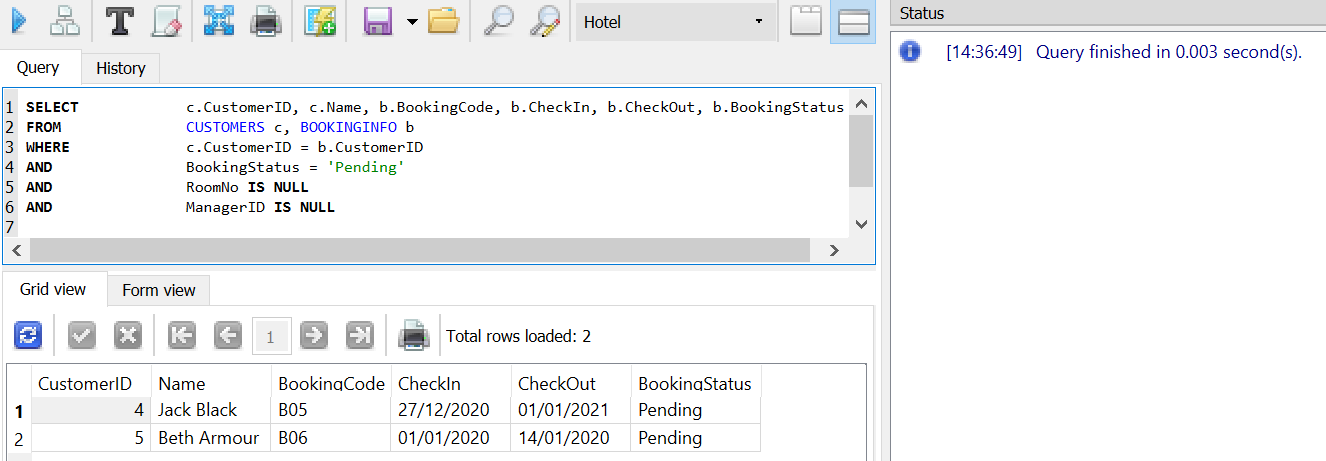


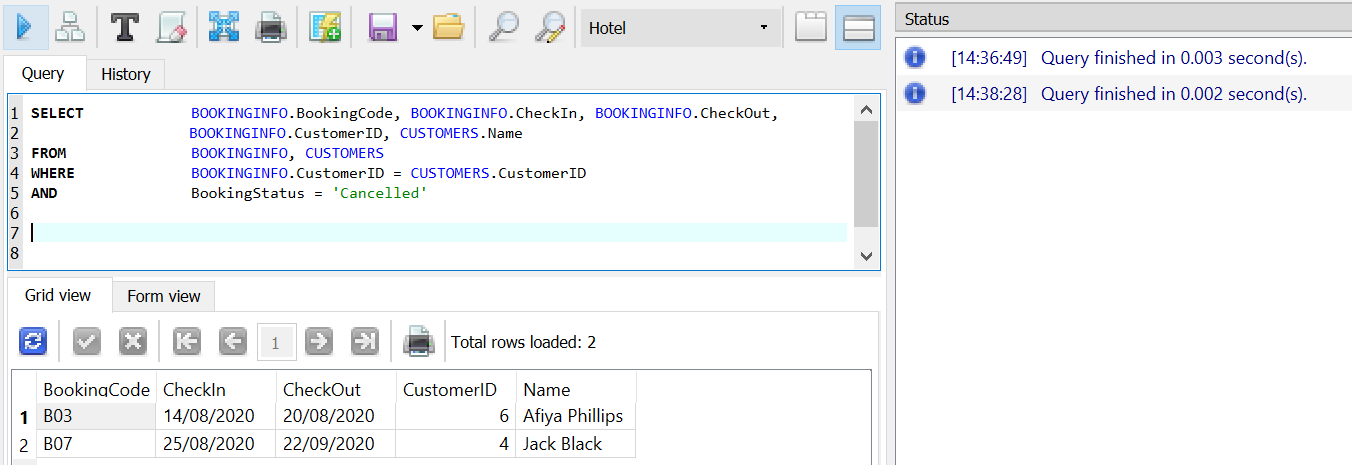
Figure 5B: The data in the BOOKINGINFO table

### Part C: SQL Queries

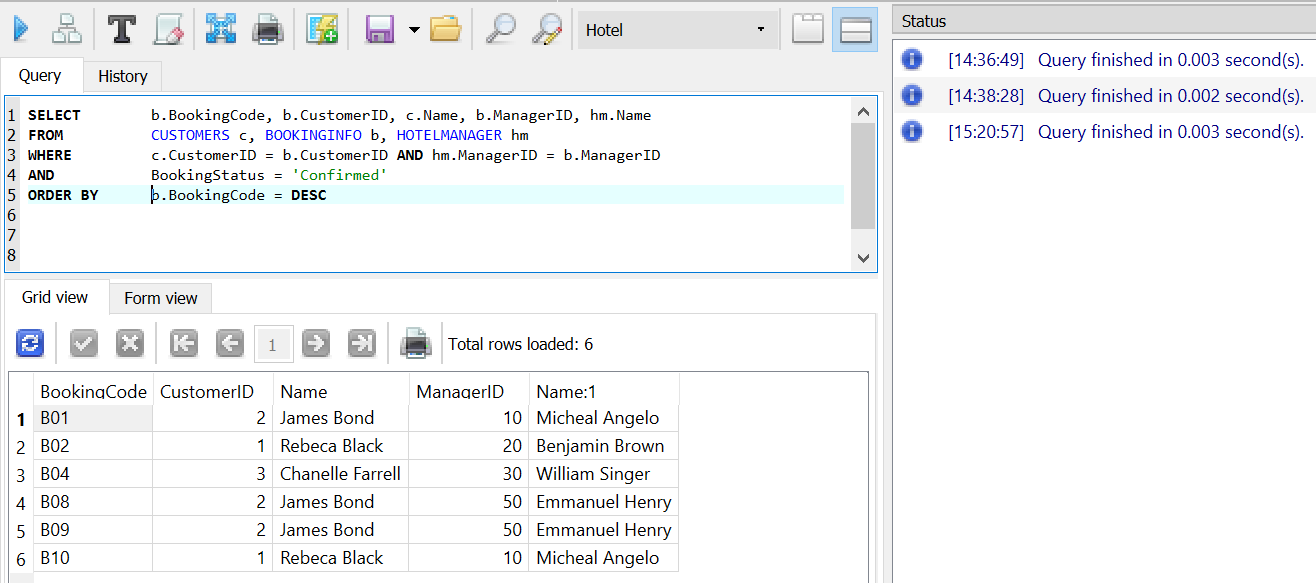
1. **List of bookings that are pending**



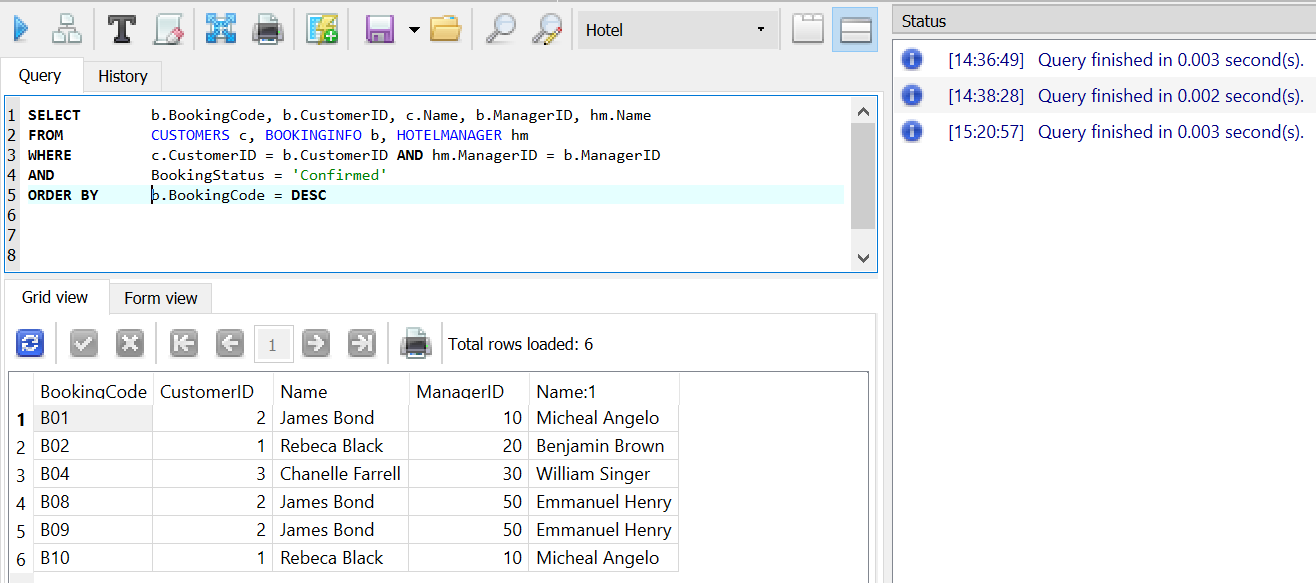
1. **Bookings that have been cancelled so that they can be rebooked**

****

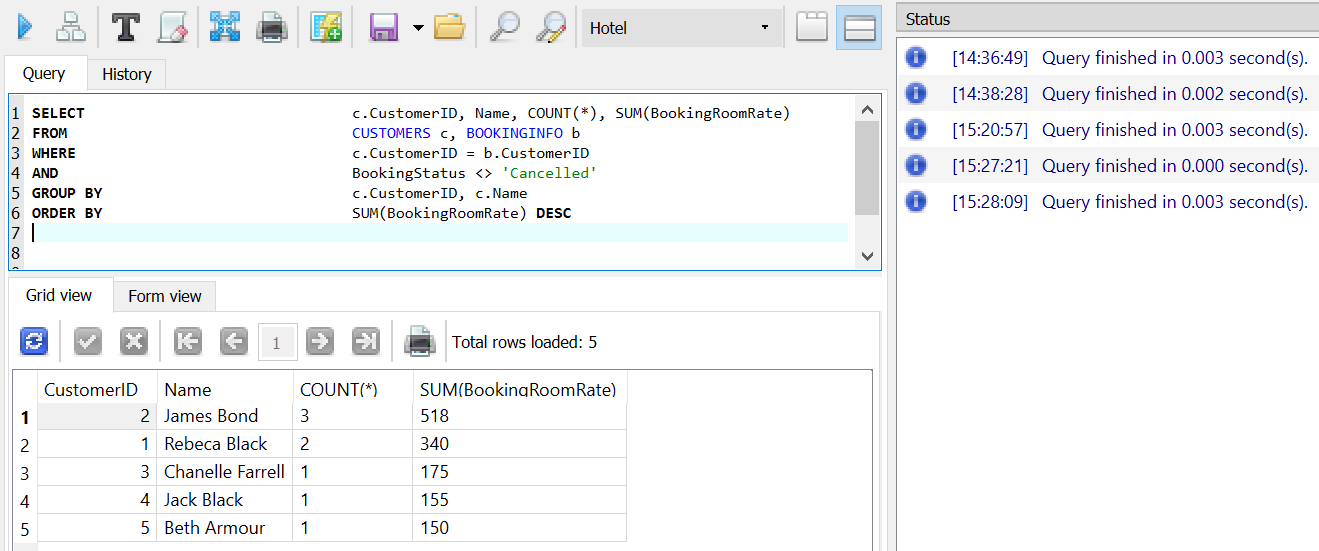
1. **Which customers, hotel managers are assigned to for bookings**



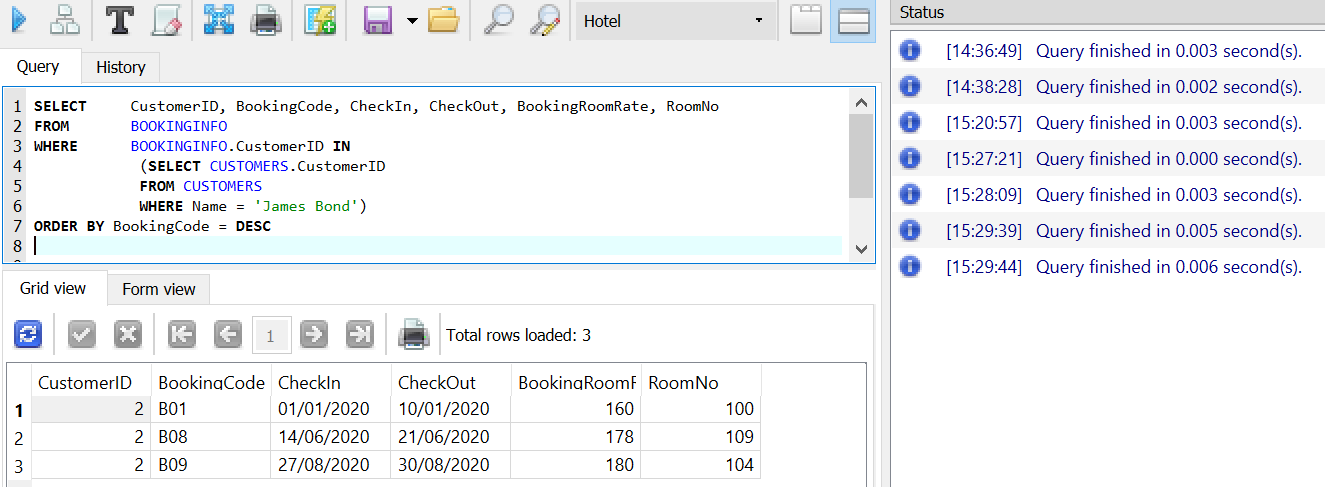
1. **The number of rooms per room type.**



1. **The top customer.**



1. **Uses a customer’s name to find they booking information**



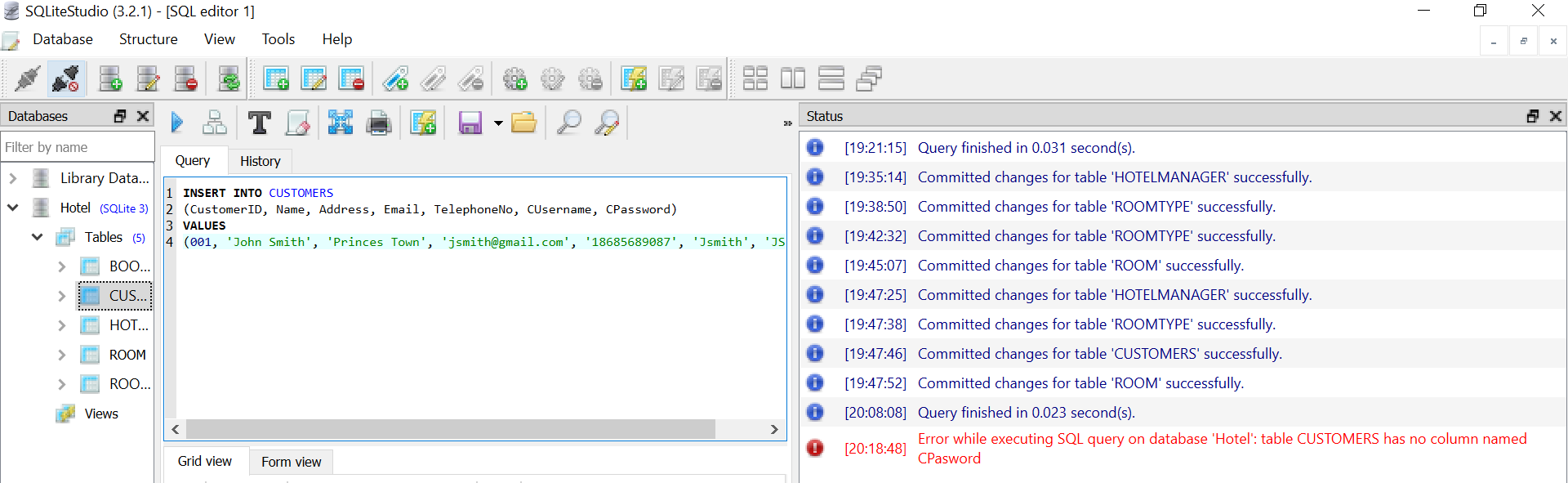
## Discussion and reflection

The implementation of the relationship database was successful. Each task was completed and lead to the success of the implementation of the database. From doing this assignment I learnt how the Entity-Relationship Model went hand in hand with Normalization. After completing the Normalization, I realized that the 3NF was the Entity relationship model.

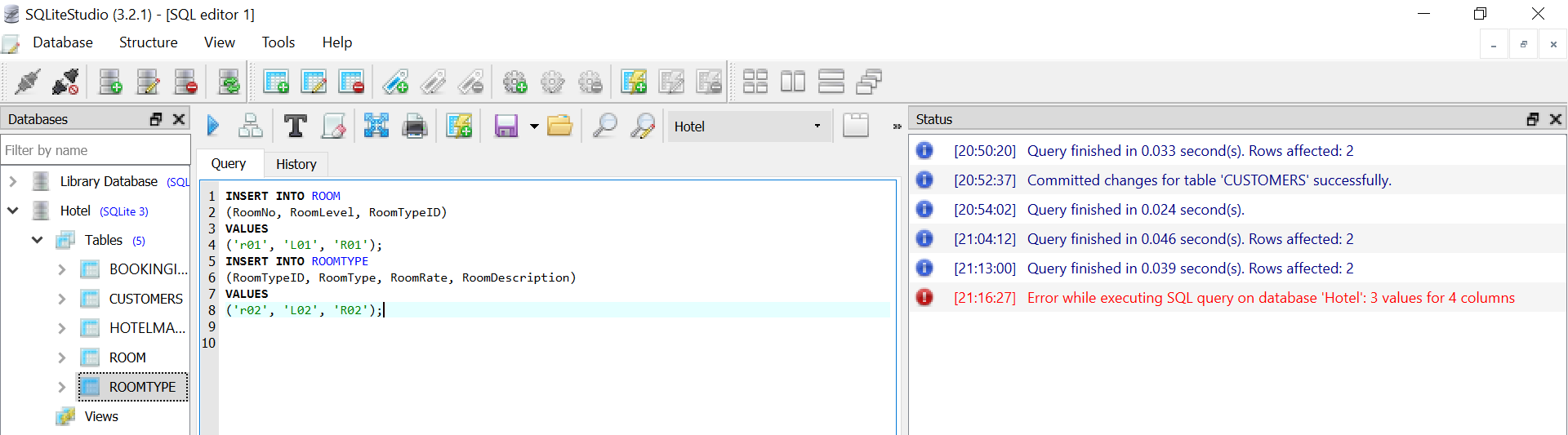
When implementing the database, the data dictionaries and SQL queries that were made for the tasks came in handy when writing the SQL in SQL lite as I just had to reference to my report. Unfortunately, syntax error still arose, some being misspelling of the column names, using the wrong joint condition when join tables, using the wrong data type and having foreign keys mismatch. When creating the tables I realized that BOOKINGINFO had to be created last because it contained foreign keys thus, they table had to be made before they could be foreign keys in the BOOKINGINFO table.

Fortunately, with a thorough debugging process and very late nights all errors were corrected, and the database and queries all ran smoothly. The implementation of the SQL queries was the best part of this assignment for me. It was amazing to see how the same query I drafted on paper can be used in the database to retrieve the data I was asking for. Overall is assignment made me learn a lot about the practical side of SQL and Computer Structure Systems and it was an enjoyable experience when the queries worked.

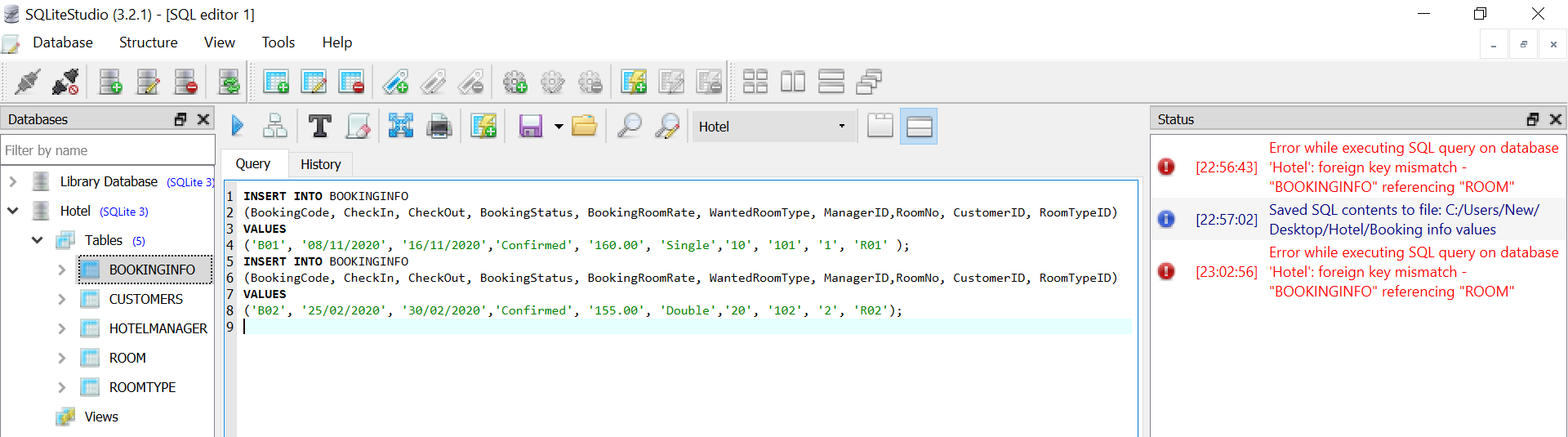
**Screenshots of some errors that occurred**:



**Figure 1:** CPassword was misspelt as CPasword



**Figure 2:** RoomRate was missing in the 2nd row of the SQL.



**Figure 3: Foreign Key mismatch.**