

Hotel Management

Database Management System
Final Project

Alessandro Profenna
Tolaz Hewa
Ryan Randive

Table of Contents

Introduction	1
Entity-Relationship Diagram	2
Database Schema with Normalization & Functional Dependencies	3
Simple Database Queries (SQL & Relational Algebra)	11
Advanced Database Queries (SQL & Relational Algebra)	17
UNIX Shell Implementation	19
Java GUI – Hotel Booking Manager	22
Conclusion	49

Introduction

Description

This Hotel Management Database System is responsible for maintaining all relevant information within the structure of a hotel business. It stores data related to guests, rooms, amenities, employees, bookings, inventory, suppliers, departments, and promotions. This system is able to perform common database functions, including inserting, deleting, and updating new information. The primary end users of this system are front desk clerks and hotel managers. The general goal of this database management system is to increase the efficiency and reliability of daily hotel operations.

Basic Functions

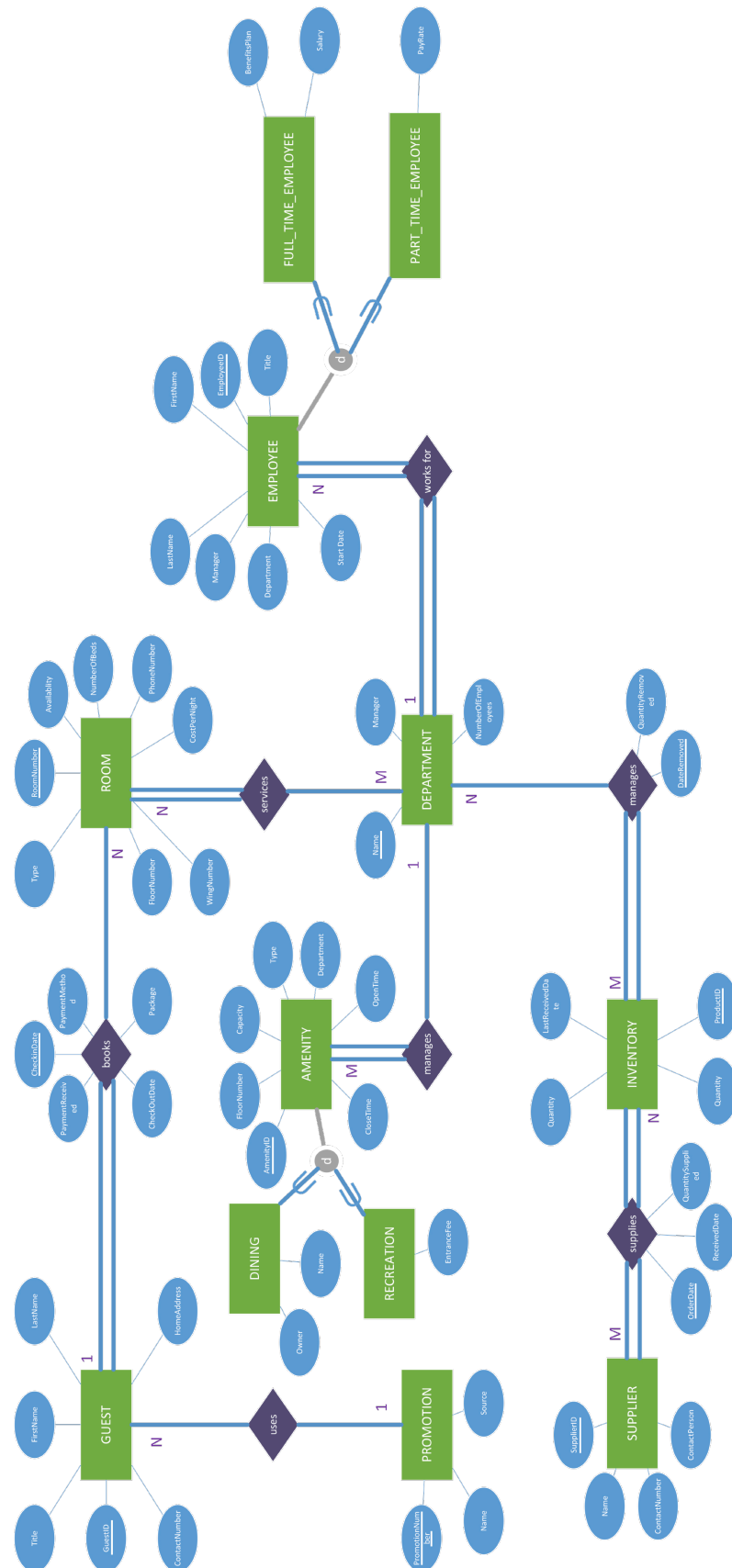
Some of the general potential functions of this DBMS are outlined below:

Functions	Description
InsertGuest	Add an entry of a guest's information when they first check into the hotel
ViewGuestInfo	Request/display guest information
ManageInventory	Update, change, and monitor inventory for the different rooms and amenities in the hotel
ManageEmployees	Update, change, and monitor information about the employees working for the hotel
SearchRooms	Search all available rooms based on specific criteria
InsertBooking	Insert a new booking based on room and guest information

The following is a list of entities in this DBMS and their relationships:

Entities	Relationships
<ul style="list-style-type: none">• Guest• Employee<ul style="list-style-type: none">• Full-time• Part-time• Room• Amenity<ul style="list-style-type: none">• Dining• Recreation• Inventory• Supplier• Promotion• Department	<ul style="list-style-type: none">• Department(1) manages Amenity(M)• Employee(N) works for Department(1)• Department(1) manages Inventory(M)• Department(M) services Room(N)• Supplier (M) supplies Inventory (N)• Guest (1) books Room (N)• Guest (N) uses Promotion(1)

Entity-Relationship Diagram



GUEST Table

```
CREATE TABLE guest(  
  GuestID          INTEGER PRIMARY KEY,  
  Title            VARCHAR2(4),  
  FirstName        VARCHAR(25) NOT NULL,  
  LastName         VARCHAR(25),  
  ContactNumber    VARCHAR(12) NOT NULL,  
  HomeAddress      VARCHAR(48)  
);
```

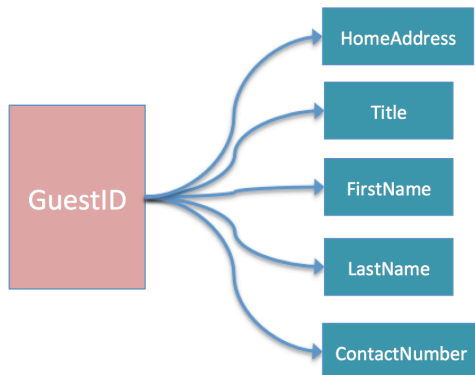
GuestID -> Title, FirstName, LastName, ContactNumber, HomeAddress

This table is in 1NF because all values are atomic.

This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key, GuestID.

This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, GuestID.

This table is in BCNF because all attributes are dependent on the primary (candidate) key GuestID.



ROOM Table

```
CREATE TABLE room(  
  RoomNumber       INTEGER PRIMARY KEY,  
  Type             VARCHAR2(15) NOT NULL ,  
  Availability      VARCHAR(1) CONSTRAINT a_check_yn CHECK (Availability IN ('Y','N')),  
  NumberOfBeds     INTEGER,  
  PhoneNumber       VARCHAR2(15),  
  CostPerNight     INTEGER,  
  WingNumber       INTEGER,  
  FloorNumber      INTEGER  
);
```

RoomNumber -> Type, Availability, NumberOfBeds, PhoneNumber, CostPerNight, WingNumber, FloorNumber

PhoneNumber -> RoomNumber

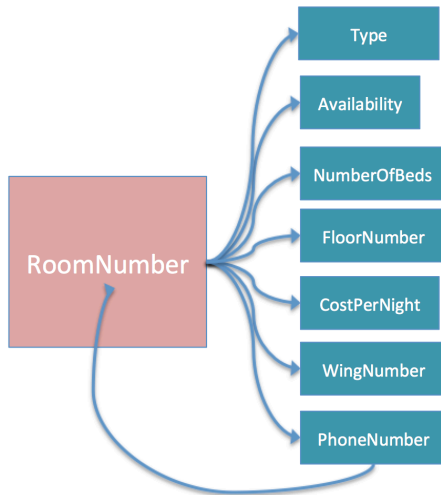
This table is in 1NF because all values are atomic.

This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key, RoomNumber.

This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, RoomNumber.

RoomNumber is dependent on PhoneNumber, but since RoomNumber is not a non-candidate key attribute, 3NF still holds.

This table is in BCNF because all attributes are dependent on the primary key RoomNumber. Also, PhoneNumber is a candidate key so BCNF still holds.



AMENITY Table

```

CREATE TABLE amenity(
  AmenityID          INTEGER PRIMARY KEY,
  FloorNumber        INTEGER NOT NULL,
  Capacity           INTEGER NOT NULL,
  Type               VARCHAR2(20),
  Department         VARCHAR2(20),
  OpenTime           FLOAT,
  CloseTime          FLOAT
);
  
```

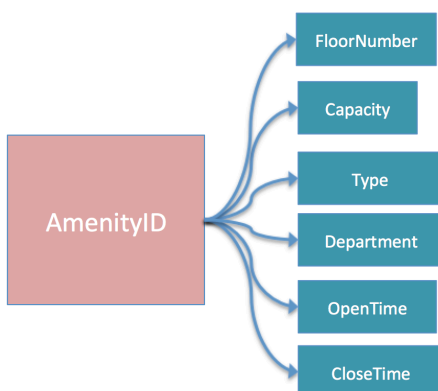
AmenityID -> FloorNumber, Capacity, Type, Department, OpenTime, CloseTime

This table is 1NF because all values are atomic.

This table is 2NF because all non-key attributes are fully functionally dependent on the primary key, AmenityID.

This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, AmenityID.

This table is in BCNF because all attributes are dependent on the primary key AmenityID.



DINING Table

```
CREATE TABLE dining(  
    AmenityID  
    Name  
    Owner  
);  
  
INTEGER REFERENCES amenity(AmenityID),  
VARCHAR2(25) NOT NULL,  
VARCHAR2(25),
```

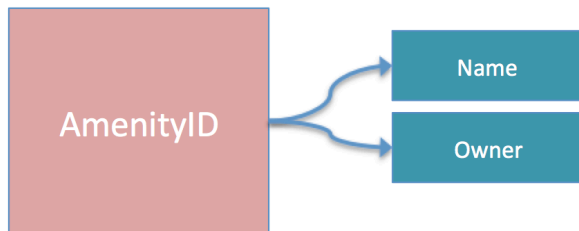
AmenityID -> Name, Owner

This table is 1NF because all values are atomic.

This table is 2NF because all non-key attributes are fully functionally dependent on the primary key, AmenityID.

This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, AmenityID.

This table is in BCNF because all attributes are dependent on the primary key AmenityID.



RECREATION Table

```
CREATE TABLE recreation(  
    AmenityID  
    EntranceFee  
);  
  
INTEGER REFERENCES amenity(AmenityID)  
INTEGER NOT NULL,
```

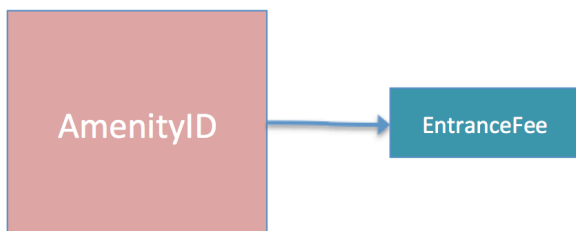
AmenityID -> EntranceFee

This table is 1NF because all values are atomic.

This table is 2NF because all non-key attributes are fully functionally dependent on the primary key, AmenityID.

This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, AmenityID.

This table is in BCNF because all attributes are dependent on the primary key AmenityID.



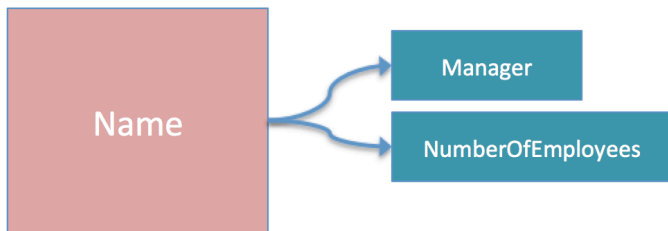
DEPARTMENT Table

```
CREATE TABLE department(  
    Name  
    Manager  
    NumberOfEmployees  
);  
  
VARCHAR2(30) PRIMARY KEY,  
VARCHAR2(30),  
INTEGER
```

Name -> Manager, NumberOfEmployees

This table is in 1NF because all values are atomic.

This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key, Name.
 This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, Name.
 This table is in BCNF because all attributes are dependent on the primary key Name.



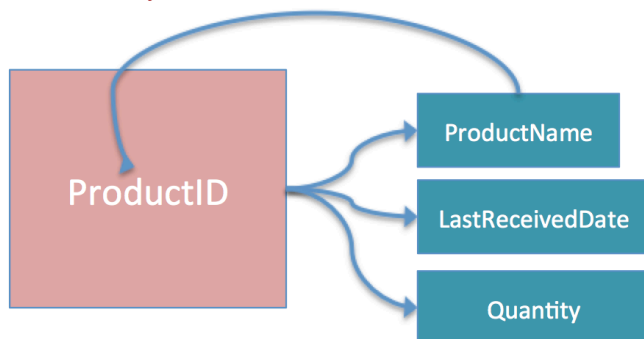
INVENTORY Table

```

CREATE TABLE inventory(
  ProductID          INTEGER PRIMARY KEY,
  ProductName        VARCHAR(30),
  LastReceivedDate   DATE,
  Quantity           INTEGER
);
  
```

ProductID -> ProductName, LastReceivedDate, Quantity
 ProductName -> ProductID

This table is in 1NF because all values are atomic.
 This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key, ProductID.
 This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, ProductID.
 ProductID is dependent on ProductName, but since ProductID is not a non-candidate key attribute, 3NF still holds.
 This table is in BCNF because all attributes are dependent on the primary key ProductID. Also, ProductName is a candidate key so BCNF still holds.



SUPPLIER Table

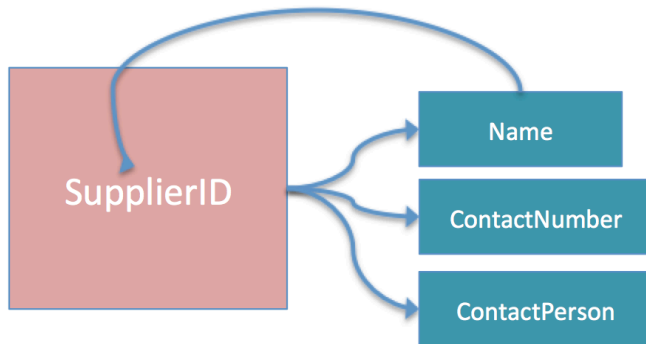
```

CREATE TABLE supplier(
  SupplierID         INTEGER PRIMARY KEY,
  Name               VARCHAR2(20),
  ContactNumber      VARCHAR2(15),
  ContactPerson      VARCHAR2(20)
);
  
```

SupplierID -> Name, Contact Number, ContactPerson
 Name -> SupplierID

This table is 1NF because all values are atomic.
 This table is 2NF because all non-key attributes are fully functionally dependent on the primary key, SupplierID.
 This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, SupplierID.

SupplierID is dependent on Name, but since SupplierID is not a non-candidate key attribute, 3NF still holds. This table is in BCNF because all attributes are dependent on the primary key SupplierID. Also, Name is a candidate key so BCNF still holds.



PROMOTION Table

```
CREATE TABLE promotion(
    PromotionNumber
    Name
    Source
);
```

INTEGER PRIMARY KEY,
VARCHAR2(20),
VARCHAR2(20)

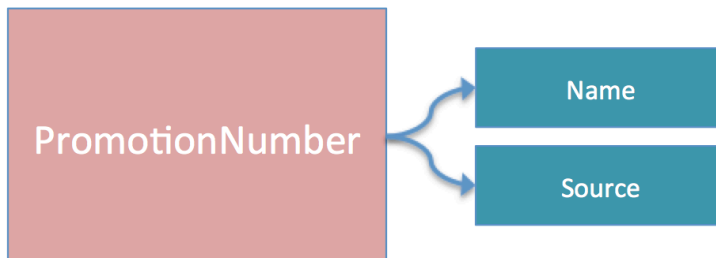
PromotionNumber -> Name, Source

This table is in 1NF because all values are atomic.

This table is in 2NF because all non-key attributes are fully functionally dependent on the primary key, PromotionNumber.

This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, PromotionNumber.

This table is in BCNF because all attributes are dependent on the primary key PromotionNumber.



FULL-TIME EMPLOYEE Table

```
CREATE TABLE full_time_employee(
    EmployeeID
    Title
    FirstName
    LastName
    Manager
    Department
    StartDate
    BenefitsPlan
    Salary
);
```

INTEGER PRIMARY KEY,
VARCHAR2(25),
VARCHAR2(25),
VARCHAR(25),
VARCHAR(25),
VARCHAR(25) REFERENCES department(Name),
DATE
VARCHAR2(20),
INTEGER,

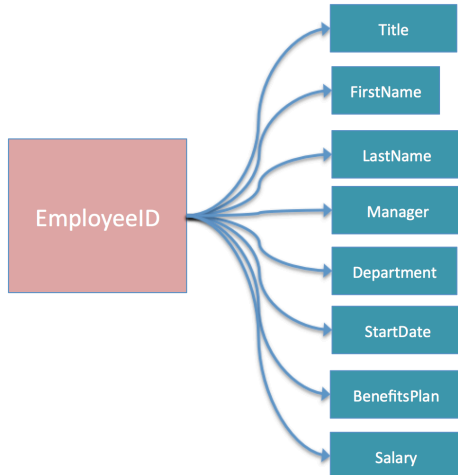
EmployeeID -> Title, FirstName, LastName, Manager, Department, StartDate, BenefitsPlan, Salary

This table is 1NF because all values are atomic.

This table is 2NF because all non-key attributes are fully functionally dependent on the primary key, EmployeeID.

This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, EmployeeID.

This table is in BCNF because all attributes are dependent on the primary key EmployeeID.



PART-TIME EMPLOYEE Table

```
CREATE TABLE part_time_employee(  
    EmployeeID          INTEGER PRIMARY KEY,  
    Title               VARCHAR2(25),  
    FirstName           VARCHAR2(25),  
    LastName            VARCHAR(25),  
    Manager             VARCHAR(25),  
    Department          VARCHAR(25) REFERENCES department(Name),  
    StartDate           DATE,  
    PayRate             INTEGER,  
);
```

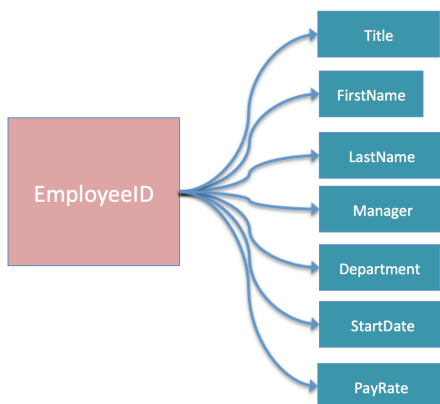
EmployeeID -> Title, FirstName, LastName, Manager, Department, StartDate, PayRate

This table is 1NF because all values are atomic.

This table is 2NF because all non-key attributes are fully functionally dependent on the primary key, EmployeeID.

This table is in 3NF because all non-key attributes are non-transitively dependent on the primary key, EmployeeID.

This table is in BCNF because all attributes are dependent on the primary key EmployeeID.



BOOKS Table

```
CREATE TABLE books(  
    GuestID  
    RoomNumber  
    CheckInDate  
    CheckOutDate  
    PaymentMethod  
    PaymentReceived  
    PRIMARY KEY  
);  
  
INTEGER REFERENCES guest(GuestID),  
INTEGER REFERENCES room(RoomNumber),  
DATE NOT NULL,  
DATE NOT NULL,  
VARCHAR2(15),  
VARCHAR(1) CONSTRAINT pr_check_yn CHECK (PaymentReceived IN ('Y','N')),  
(RoomNumber, CheckInDate)
```

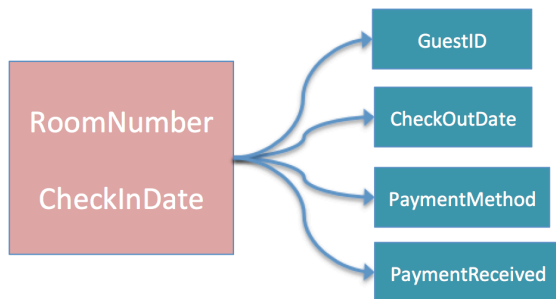
RoomNumber, CheckInDate -> CheckOutDate, PaymentMethod, PaymentReceived, GuestID

This table is in 1NF because all values are atomic.

This table is in 2NF because all non-key attributes are fully functionally dependent on the composite primary key, consisting of RoomNumber, and CheckInDate.

This table is in 3NF because all non-key attributes are non-transitively dependent on the composite primary key, consisting of RoomNumber, and CheckInDate.

This table is in BCNF because all attributes are dependent on the primary key (composite key) consisting of RoomNumber and CheckInDate.



SERVICES Table

```
CREATE TABLE services(  
    DepartmentName  
    RoomNumber  
    PRIMARY KEY  
);  
  
Varchar(25) REFERENCES department(Name),  
INTEGER REFERENCES room(RoomNumber),  
(DepartmentName, RoomNumber)
```

This table has no functional dependencies.

MANAGES Table

```
CREATE TABLE manages(  
    DepartmentName  
    ProductID  
    QuantityRemoved  
    DateRemoved  
    PRIMARY KEY  
);  
  
VARCHAR(25) REFERENCES department(Name),  
INTEGER REFERENCES inventory(ProductID),  
INTEGER,  
DATE,  
(DepartmentName, ProductID, DateRemoved)
```

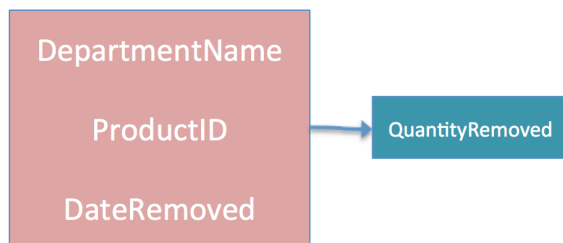
DepartmentName, ProductID, DateRemoved -> QuantityRemoved

This table is in 1NF because all values are atomic.

This table is in 2NF because all non-key attributes are fully functionally dependent on the composite primary key, consisting of DepartmentName, ProductID, and DateRemoved.

This table is in 3NF because all non-key attributes are non-transitively dependent on the composite primary key, consisting of DepartmentName, ProductID, and DateRemoved.

This table is in BCNF because all attributes are dependent on the primary key (composite key) consisting of DepartmentName, ProductID and DateRemoved.



SUPPLIES Table

```
CREATE TABLE supplies(  
    SupplierID          INTEGER REFERENCES supplier(SupplierID),  
    ProductID           INTEGER REFERENCES inventory(ProductID),  
    QuantitySupplied    INTEGER,  
    OrderDate           DATE,  
    ReceivedDate        DATE,  
    PRIMARY KEY         (SupplierID, ProductID, OrderDate)  
);
```

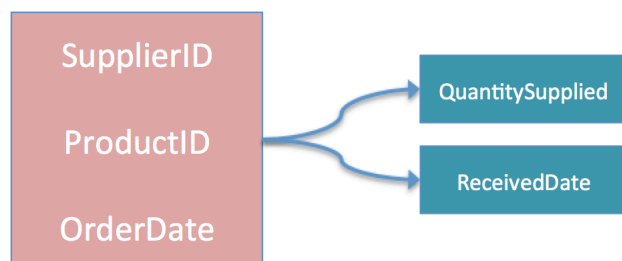
SupplierID, ProductID, OrderDate -> QuantitySupplied, ReceivedDate

This table is in 1NF because all values are atomic.

This table is in 2NF because all non-key attributes are fully functionally dependent on the composite primary key, consisting of SupplierID, ProductID, and OrderDate.

This table is in 3NF because all non-key attributes are non-transitively dependent on the composite primary key, consisting of SupplierID, ProductID, and OrderDate.

This table is in BCNF because all attributes are dependent on the primary key (composite key) consisting of SupplierID, ProductID and OrderDate.



Simple Database Queries (SQL & Relational Algebra)

SQL:

```
SELECT *
  FROM guest
 WHERE Title = 'Mr.' ;
```

Relational Algebra:

$\sigma_{\text{title} = \text{'Mr.'}}(\text{guest})$

```
SQL> SELECT * FROM guest WHERE Title = 'Mr.' ;

  GUESTID TITL FIRSTNAME          LASTNAME          CONTACTNUMBE
-----
HOMEADDRESS
-----
      2 Mr.  John          Tory          555-353-5555
33 Dundas St.
      4 Mr.  Alessandro    Profenna          555-555-5555
8 Dundas St.
```

SQL:

```
SELECT RoomNumber, availability
  FROM room
 WHERE NumberOfBeds = 2;
```

Relational Algebra:

$\pi_{\text{RoomNumber, Availability}} \sigma_{\text{NumberOfBeds} = 2}(\text{room})$

```
SQL> SELECT RoomNumber, availability FROM room WHERE NumberOfBeds = 2;

ROOMNUMBER A
-----
      220 N
      453 Y
      832 Y
```

SQL:

```
SELECT RoomNumber AS Budget_Rooms, CostPerNight
  FROM room
 WHERE CostPerNight <= 200
```

ORDER BY CostPerNight ASC;

Relational Algebra:

$\tau_{\text{CostPerNight asc}} \rho_{\text{Budget_Rooms}} \leftarrow \text{RoomNumber} \pi_{\text{RoomNumber, CostPerNight}} \sigma_{\text{CostPerNight} \leq 200}(\text{room})$

```
SQL> SELECT RoomNumber AS Budget_Rooms, CostPerNight FROM room WHERE CostPerNight <= 200 ORDER BY CostPerNight ASC;
```

BUDGET_ROOMS	COSTPERNIGHT
832	80
110	120
220	165

SQL:

```
CREATE VIEW open_late_dining AS
(SELECT AmenityID
FROM amenity
WHERE CloseTime > 21.0
AND type = 'Restaurant');
```

```
SQL> CREATE VIEW open_late_dining AS (SELECT AmenityID FROM amenity WHERE CloseTime > 21.0 AND type = 'Restaurant');
View created.

SQL> SELECT * FROM open_late_dining
2 ;
```

AMENITYID
2
3

SQL:

```
CREATE VIEW large_department AS
(SELECT *
FROM department
WHERE NumberOfEmployees >= 20) ;
```

```
SQL> CREATE VIEW large_department AS (SELECT * FROM department WHERE NumberOfEmployees >= 20) ;
View created.

SQL> SELECT * FROM large_department;
```

NAME	MANAGER	NUMBEROFEMPLOYEES
Recreational	Tolaz Hewa	30
FoodServices	Regina George	22

```
SQL> █
```

SQL:

```
SELECT *  
    FROM inventory  
    WHERE Quantity <= 50;
```

Relational Algebra:

$\sigma_{\text{Quantity} \leq 50}(\text{inventory})$

```
SQL> SELECT * FROM inventory WHERE Quantity <= 50;  
  
PRODUCTID PRODUCTNAME                LASTRECEI  QUANTITY  
-----  
          38 towel_large                19-SEP-16         27  
  
SQL> █
```

SQL:

```
SELECT *  
    FROM supplier  
    WHERE Name = 'CourtesyProducts' ;
```

Relational Algebra:

$\sigma_{\text{Name} = \text{'CourtesyProducts'}}(\text{supplier})$

```
SQL> SELECT * FROM supplier WHERE Name = 'CourtesyProducts' ;  
  
SUPPLIERID NAME                CONTACTNUMBER  CONTACTPERSON  
-----  
          31 CourtesyProducts    525-353-5555  Gregor Clegane  
  
SQL> █
```

SQL:

```
SELECT *  
    FROM promotion  
    WHERE Name <> 'FreeNight';
```

Relational Algebra:

$\sigma_{\text{Name} \neq \text{'FreeNight'}}(\text{Promotion})$

```

SQL>
SQL> SELECT * FROM promotion WHERE Name <> 'FreeNight';

PROMOTIONNUMBER NAME                SOURCE
-----
37 50%OFF                expedia.com/promo

SQL>

```

SQL:

```

SELECT *
FROM employee
WHERE Department = 'Kitchen' OR Department = 'FoodServices';

```

Relational Algebra:

σ Department = 'Kitchen' or Department = 'FoodServices' (employee)

```

SQL>
SQL> SELECT * FROM employee WHERE Department = 'Kitchen' OR Department = 'FoodServices';

EMPLOYEEID TITLE                FIRSTNAME
-----
LASTNAME                MANAGER                DEPARTMENT
-----
STARTDATE
-----
335 Lead Chef                Arya
Stark                    No One                FoodServices
22-SEP-16
SQL>

```

SQL:

```

SELECT DISTINCT DepartmentName AS Departments_Servicing_Floor8
FROM services
WHERE RoomNumber >= 800
AND RoomNumber < 900;

```

Relational Algebra:

ρ Departments_Servicing_Floor8 \leftarrow DepartmentName π DepartmentName σ RoomNumber \geq 800 and RoomNumber < 900 (services)


```
SQL>
SQL> SELECT DISTINCT DepartmentName AS Departments_Servicing_Floor8 FROM services WHERE RoomNumber >= 800 AND RoomNumber < 900;

DEPARTMENTS_SERVICING_FLO
-----
Recreational
```

SQL:

```
SELECT RoomNumber, CheckInDate, PaymentReceived
FROM books
WHERE PaymentReceived = 'N';
```

Relational Algebra:

Π RoomNumber, CheckInDate, PaymentReceived σ PaymentReceived = 'N' (books)

```
SQL> SELECT RoomNumber, CheckInDate, PaymentReceived FROM books WHERE PaymentReceived = 'N';

ROOMNUMBER CHECKINDATE PAYMENTRECEIVED
-----
220 15-SEP-16 N

SQL>
```

SQL:

```
SELECT EmployeeID, 'works for: ', DepartmentName
FROM works_for;
```

Relational Algebra:

Π EmployeeID, 'works for: ', DepartmentName (works_for)

```
SQL>
SQL> SELECT EmployeeID, 'works for: ', DepartmentName FROM works_for;

EMPLOYEEID 'WORKSFOR:' DEPARTMENTNAME
-----
223 works for: Recreational

SQL>
```

SQL:

```
SELECT QuantityRemoved, ProductID, DateRemoved
FROM manages
WHERE QuantityRemoved > 0;
```

Relational Algebra:

Π QuantityRemoved, ProductID, DateRemoved σ QuantityRemoved > 0 (manages)

```
SQL> SELECT QuantityRemoved, ProductID, DateRemoved FROM manages WHERE QuantityRemoved > 0;
```

QUANTITYREMOVED	PRODUCTID	DATEREMOV
5	38	15-OCT-16

SQL:

```
SELECT *
  FROM supplies
 WHERE QUANTITYSUPPLIED = 40;
```

Relational Algebra:

σ QUANTITYSUPPLIED = 40 (supplies)

```
SQL> SELECT * FROM supplies WHERE QUANTITYSUPPLIED = 40;
```

SUPPLIERID	PRODUCTID	QUANTITYSUPPLIED	ORDERDATE	RECEIVEDD
31	32	40	15-OCT-16	17-OCT-16

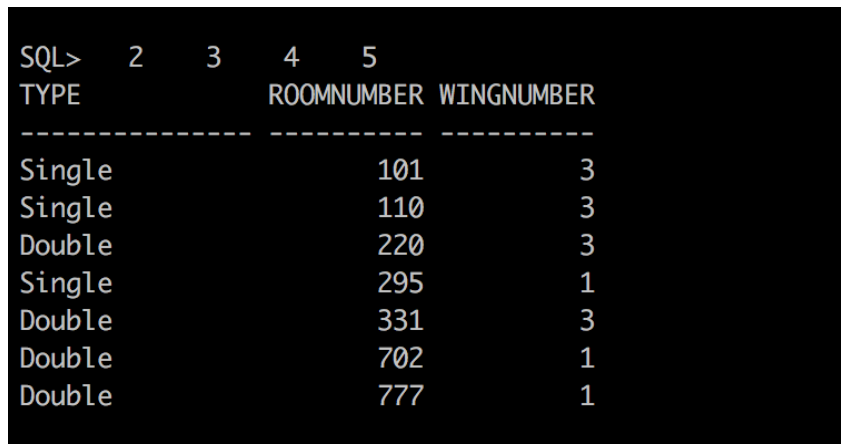
```
SQL> █
```

SQL:

```
SELECT Type, RoomNumber
FROM room
WHERE WingNumber <> 2
GROUP BY WingNumber
ORDER BY RoomNumber;
```

Relational Algebra:

Π Type, RoomNumber (σ WingNumber \neq 2 (room))



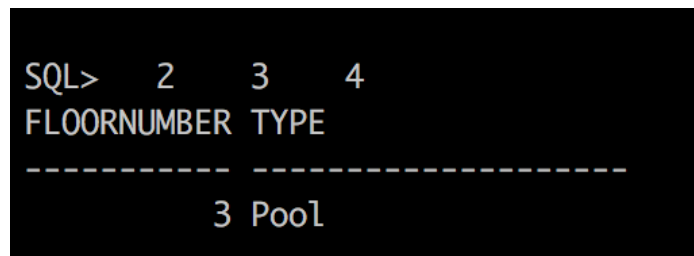
```
SQL> 2 3 4 5
TYPE          ROOMNUMBER WINGNUMBER
-----
Single        101         3
Single        110         3
Double        220         3
Single        295         1
Double        331         3
Double        702         1
Double        777         1
```

SQL:

```
SELECT FloorNumber, Type
FROM room, amenity
WHERE Availability = 'Y'
AND room.FloorNumber = amenity.FloorNumber;
```

Relational Algebra:

Π FloorNumber, Type (σ Availability = 'Y' AND room.FloorNumber = amenity.FloorNumber (room \bowtie amenity))



```
SQL> 2 3 4
FLOORNUMBER TYPE
-----
3 Pool
```

SQL:

```
SELECT FirstName, LastName, Title
FROM full_time_employee
WHERE Department = 'FoodServices'
MINUS
(SELECT * FROM full_time_employee x WHERE x.Title = 'Cook');
```

Relational Algebra:

π FirstName, LastName, Title (σ Department = 'FoodServices' - (σ x.title = 'Cook' (full_time_employee)) (full_time_employee))

```
SQL> 2 3 4 5
```

FIRSTNAME	LASTNAME	TITLE
Anakin	Skywalker	Bartender
Guy	Fieri	Waiter

SQL:

```
SELECT 'Average salary for full-time employees is: ', AVG(Salary)
FROM full_time_employee;
```

Relational Algebra:

π AVG(salary) (full_time_employee)

```
SQL> 2
```

'AVERAGESALARYFORFULL-TIMEEMPLOYEESIS: '	AVG(SALARY)
Average salary for full-time employees is:	85250

SQL:

```
SELECT DISTINCT amenity.AmenityID, Name, Capacity
FROM amenity, dining
WHERE amenity.AmenityID = dining.AmenityID
ORDER BY Capacity;
```

Relational Algebra:

π amenity.AmenityID, Name, Capacity (σ amenity.AmenityID = dining.AmenityID (amenity \bowtie dining))

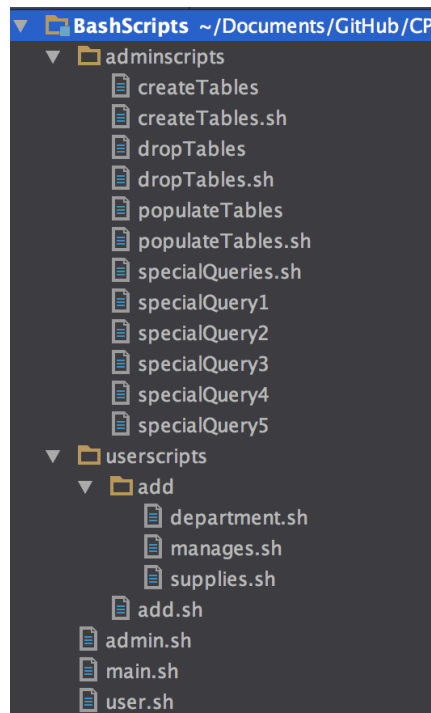
```
SQL> 2 3 4
```

AMENITYID	NAME	CAPACITY
4	CoffeeLife	20
2	Beefcakes	60
3	TasteOfItaly	120

UNIX Shell Implementation

In our UNIX Shell implementation, we provide users and administrators the ability to create, drop, and populate tables in our Hotel Management Database. These scripts also include some specific queries for the database.

The structure of scripts is as follows:



Below is the code for main.sh, admin.sh, and user.sh:

main.sh

```
#!/bin/bash

echo $'\n\nWelcome to the Hotel Database Management System, how can I help you?\n\n'
echo '1: User'
echo '2: Administrator'
echo 'q: Quit'

read input

while [ "$input" != "q" ] && [ "$input" != "Q" ];
do
    if [ "$input" = "1" ]; then
        echo 'You have chosen option 1'
        ./user.sh
    elif [ "$input" = "2" ]; then
        echo 'You have chosen option 2'
        ./admin.sh
    else
        echo 'Invalid Input'
    fi
done
```

```

fi
echo $'\n'
echo '1: User'
echo '2: Administrator'
echo 'q: Quit'

read input
done

```

admin.sh

```

#!/bin/bash

echo $'\n'
echo '1: Create Tables'
echo '2: Populate Tables'
echo '3: Drop Tables'
echo '4: Special Queries'
echo 'q: Quit'

read input

while [ "$input" != "q" ] && [ "$input" != "Q" ];
do
    if [ "$input" = "1" ]; then
        echo 'You have chosen option 1'
        ./adminscripts/createTables.sh
    elif [ "$input" = "2" ]; then
        echo 'You have chosen option 2'
        ./adminscripts/populateTables.sh
    elif [ "$input" = "3" ]; then
        echo 'You have chosen option 3'
        ./adminscripts/dropTables.sh
    elif [ "$input" = "4" ]; then
        echo 'You have chosen option 4'
        ./adminscripts/specialQueries.sh
    else
        echo 'Invalid Input'
    fi

    echo $'\n'
    echo '1: Create Tables'
    echo '2: Populate Tables'
    echo '3: Drop Tables'
    echo '4: Special Queries'
    echo 'q: Quit'

    read input
done

```

user.sh

```

#!/bin/bash

echo $'\n\n'
echo '1: Add'
echo '2: Update'

```

```

echo '3: Delete'
echo '4: Query'
echo 'q: Quit'

read input

while [ "$input" != "q" ] && [ "$input" != "Q" ];
do
    if [ "$input" = "1" ]; then
        echo 'You have chosen option 1'
        ./userscripts/add.sh
    elif [ "$input" = "2" ]; then
        echo 'You have chosen option 2'
        ./userscripts/update.sh
    elif [ "$input" = "3" ]; then
        echo 'You have chosen option 3'
        ./userscripts/delete.sh
    elif [ "$input" = "4" ]; then
        echo 'You have chosen option 4'
        ./userscripts/query.sh
    else
        echo 'Invalid Input'
    fi

    echo $'\n1: Add'
    echo '2: Update'
    echo '3: Delete'
    echo '4: Query'
    echo 'q: Quit'

    read input
done

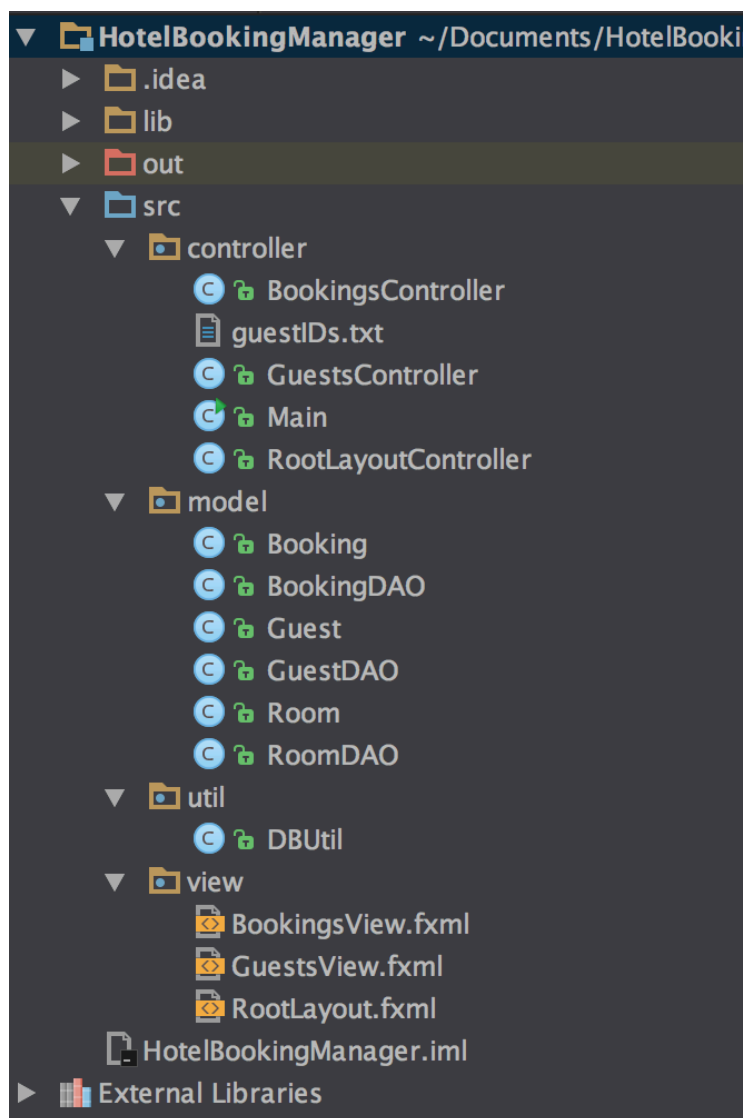
```

Java GUI – Hotel Booking Manager

In our Java implementation, we designed a GUI using JavaFX that would be typically used by a hotel concierge. It is representative of hotel room booking software.

This booking manager greets users with a main page that includes views for two tables. The user is able to view all available rooms in the hotel, and even search for rooms based on specific criteria. He or she is also able to book a room by inputting guest and booking information. Doing so adds entries to the guest table and books table in the database. Via the dropdown menu, the hotel guest table is also viewable in a separate window.

The structure of program uses the DAO Design Pattern to perform operations. In DAO pattern, domain (business) logic does not directly communicate with the database. It communicates with DAO layer and DAO layer handles database operations and sends the results to the business layer.



[illegible]

DBUtil.java

```
package util;
import com.sun.rowset.CachedRowSetImpl;
import java.sql.*;

import java.util.Properties;

public class DBUtil {
    //Declare JDBC Driver
    private static final String JDBC_DRIVER = "oracle.jdbc.driver.OracleDriver";

    //Connection
    private static Connection conn = null;

    public static String host = "jdbc:oracle:thin:@141.117.57.159:1521:orcl";
    public static String uName = "thewa";
    //Password is hidden here. Only the real code has the actual password.
    public static String uPass = "*****";

    //Connect to DB
    public static void dbConnect() throws SQLException, ClassNotFoundException {
        //Setting Oracle JDBC Driver
        try {
            Class.forName(JDBC_DRIVER);
        } catch (ClassNotFoundException e) {
            System.out.println("Where is your Oracle JDBC Driver?");
            e.printStackTrace();
            throw e;
        }

        System.out.println("Oracle JDBC Driver Registered!");

        //Establish the Oracle Connection using Connection String
        try {
            conn = DriverManager.getConnection(host, uName, uPass);
        } catch (SQLException e) {
            System.out.println("Connection Failed! Check output console" + e);
            e.printStackTrace();
            throw e;
        }
    }

    //Close Connection
    public static void dbDisconnect() throws SQLException {
        try {
            if (conn != null && !conn.isClosed()) {
                conn.close();
            }
        } catch (Exception e){
            throw e;
        }
    }

    //DB Execute Query Operation
    public static ResultSet dbExecuteQuery(String queryStmt) throws SQLException,
    ClassNotFoundException {
        //Declare statement, resultSet and CachedResultSet as null
        Statement stmt = null;
        ResultSet resultSet = null;
    }
}
```

```

CachedRowSetImpl crs = null;
try {
    //Connect to DB (Establish Oracle Connection)
    dbConnect();
    System.out.println("Select statement: " + queryStmt + "\n");

    //Create statement
    stmt = conn.createStatement();

    //Execute select (query) operation
    resultSet = stmt.executeQuery(queryStmt);

    //CachedRowSet Implementation
    //In order to prevent "java.sql.SQLRecoverableException: Closed Connection: next"
error    //We are using CachedRowSet
    crs = new CachedRowSetImpl();
    crs.populate(resultSet);
} catch (SQLException e) {
    System.out.println("Problem occurred at executeQuery operation : " + e);
    throw e;
} finally {
    if (resultSet != null) {
        //Close resultSet
        resultSet.close();
    }
    if (stmt != null) {
        //Close Statement
        stmt.close();
    }
    //Close connection
    dbDisconnect();
}
//Return CachedRowSet
return crs;
}

//DB Execute Update (For Update/Insert/Delete) Operation
public static void dbExecuteUpdate(String sqlStmt) throws SQLException,
ClassNotFoundException {
    //Declare statement as null
    Statement stmt = null;
    try {
        //Connect to DB (Establish Oracle Connection)
        dbConnect();
        //Create Statement
        stmt = conn.createStatement();
        //Run executeUpdate operation with given sql statement
        stmt.executeUpdate(sqlStmt);
    } catch (SQLException e) {
        System.out.println("Problem occurred at executeUpdate operation : " + e);
        throw e;
    } finally {
        if (stmt != null) {
            //Close statement
            stmt.close();
        }
        //Close connection
        dbDisconnect();
    }
}
}

```

```
}
```

Main.java

```
package controller;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

import java.io.IOException;

//controller.Main class which extends from Application Class
public class Main extends Application {

    private Stage primaryStage;

    private BorderPane rootLayout;

    @Override
    public void start(Stage primaryStage) {
        //1) Declare a primary stage (Everything will be on this stage)
        this.primaryStage = primaryStage;

        //Optional: Set a title for primary stage
        this.primaryStage.setTitle("Hotel Booking Manager");

        //2) Initialize RootLayout
        initRootLayout();

        //3) Display the BookingOperations View
        showBookingsView();
    }

    //Initializes the root layout.
    public void initRootLayout() {
        try {
            //First, load root layout from RootLayout.fxml
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(Main.class.getResource("../view/RootLayout.fxml"));
            rootLayout = (BorderPane) loader.load();

            //Second, show the scene containing the root layout.
            Scene scene = new Scene(rootLayout); //We are sending rootLayout to the Scene.
            primaryStage.setScene(scene); //Set the scene in primary stage.

            //Give the controller access to the main.
            RootLayoutController controller = loader.getController();
            controller.setMain(this);

            //Third, show the primary stage
            primaryStage.show(); //Display the primary stage
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //Shows the bookings view inside the root layout.
}
```

```

public void showBookingsView() {
    try {
        //First, load BookingsView from BookingsView.fxml
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("../view/BookingsView.fxml"));
        AnchorPane bookingsOperationsView = (AnchorPane) loader.load();

        // Set Bookings Operations view into the center of root layout.
        rootLayout.setCenter(bookingsOperationsView);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

//Shows the guest view inside the root layout.
public void showGuestsView() {
    try {
        //First, load BookingsView from BookingsView.fxml
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("../view/GuestsView.fxml"));
        AnchorPane GuestsOperationsView = (AnchorPane) loader.load();

        // Set Bookings Operations view into the center of root layout.
        rootLayout.setCenter(GuestsOperationsView);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void main(String[] args) {
    launch(args);
}
}

```

RootLayoutController.java

```

package controller;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.*;

public class RootLayoutController {

    //Reference to the main application
    private Main main;

    //Is called by the main application to give a reference back to itself.
    public void setMain (Main main) {
        this.main = main;
    }

    public void showGuestsView() {
        main.showGuestsView();
    }

    public void showBookingsView() {
        main.showBookingsView();
    }

    //Exit the program
    public void handleExit(ActionEvent actionEvent) {
        System.exit(0);
    }
}

```

```
}
```

```
}
```

GuestsController.java

```
package controller;

import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.paint.Color;
import model.*;

import java.io.*;
import java.sql.Date;
import java.sql.SQLException;

public class GuestsController {

    @FXML
    private Label roomErrorText;
    @FXML
    private Label bookingErrorText;

    @FXML
    private TableView<Guest> guestsTable;
    @FXML
    private TableColumn<Guest, Integer> guestIDColumn;
    @FXML
    private TableColumn<Guest, String> titleColumn;
    @FXML
    private TableColumn<Guest, String> firstNameColumn;
    @FXML
    private TableColumn<Guest, String> lastNameColumn;
    @FXML
    private TableColumn<Guest, String> contactNumberColumn;
    @FXML
    private TableColumn<Guest, String> homeAddressColumn;

    //View all Guests
    @FXML
    private void viewGuests(ActionEvent actionEvent) throws SQLException,
    ClassNotFoundException {

        try {
            //Get all Rooms information
            ObservableList<Guest> guestData = GuestDAO.viewGuests();
            //Populate Rooms on TableView
            populateGuests(guestData);
        } catch (SQLException e){
            roomErrorText.setTextFill(Color.web("#dd0000"));
            roomErrorText.setText("An error occurred. Please try again.");
        }
    }
}
```

```

        System.out.println("Error occurred while getting Guest information from DB.\n" +
e);
        throw e;
    }
}

//Initializing the controller class.
//This method is automatically called after the fxml file has been loaded.
@FXML
private void initialize () {

    guestIDColumn.setCellValueFactory(cellData ->
cellData.getValue().guestIDProperty().asObject());
    titleColumn.setCellValueFactory(cellData -> cellData.getValue().titleProperty());
    firstNameColumn.setCellValueFactory(cellData ->
cellData.getValue().firstNameProperty());
    lastNameColumn.setCellValueFactory(cellData ->
cellData.getValue().lastNameProperty());
    contactNumberColumn.setCellValueFactory(cellData ->
cellData.getValue().contactNumberProperty());
    homeAddressColumn.setCellValueFactory(cellData ->
cellData.getValue().homeAddressProperty());
}

//Populate Guests for TableView
@FXML
private void populateGuests (ObservableList<Guest> guestData) throws
ClassNotFoundException {
    //Set items to the Guest Table
    guestsTable.setItems(guestData);
}
}

```

BookingsController.java

```

package controller;

import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.paint.Color;
import model.Room;
import model.RoomDAO;
import model.GuestDAO;
import model.Booking;
import model.BookingDAO;

import java.io.*;
import java.sql.Date;
import java.sql.SQLException;

public class BookingsController {

    @FXML

```

```

private TextField roomTypeText;
@FXML
private TextField numberOfBedsText;
@FXML
private TextField maxNightlyRateText;
@FXML
private TextField floorNumberText;
@FXML
private TextField roomNumberText;
@FXML
private TextField checkInDateText;
@FXML
private TextField checkOutDateText;
@FXML
private TextField packageText;
@FXML
private TextField guestFirstNameText;
@FXML
private TextField guestLastNameText;
@FXML
private TextField guestTitleText;
@FXML
private TextField guestContactNumberText;
@FXML
private TextField guestHomeAddressText;

@FXML
private Label roomErrorText;
@FXML
private Label bookingErrorText;

@FXML
private TableView availableRoomsTable;
@FXML
private TableColumn<Room, Integer> roomNumberColumn;
@FXML
private TableColumn<Room, String> roomTypeColumn;
@FXML
private TableColumn<Room, String> roomAvailabilityColumn;
@FXML
private TableColumn<Room, Integer> roomNumberOfBedsColumn;
@FXML
private TableColumn<Room, String> roomPhoneNumberColumn;
@FXML
private TableColumn<Room, Integer> roomNightlyRateColumn;
@FXML
private TableColumn<Room, Integer> roomWingNumberColumn;
@FXML
private TableColumn<Room, Integer> roomFloorNumberColumn;

@FXML
private TableView bookingsTable;
@FXML
private TableColumn<Booking, Integer> booksGuestIDColumn;
@FXML
private TableColumn<Booking, Integer> booksRoomNumberColumn;
@FXML
private TableColumn<Booking, Date> booksCheckInDateColumn;
@FXML

```



```

private TableColumn<Booking, Date> booksCheckOutDateColumn;
@FXML
private TableColumn<Booking, String> booksPackageColumn;
@FXML
private TableColumn<Booking, String> booksPaymentMethodColumn;
@FXML
private TableColumn<Booking, String> booksPaymentReceivedColumn;

//Search all Rooms
@FXML
private void searchRooms(ActionEvent actionEvent) throws SQLException,
ClassNotFoundException {

    String type = null;
    int numOfBeds = 0;
    int maxRate = 0;
    int floorNum = 0;

    if(!roomTypeText.getText().equals(""))
        type = roomTypeText.getText();
    if(!numberOfBedsText.getText().equals(""))
        numOfBeds = Integer.parseInt(numberOfBedsText.getText());
    if(!maxNightlyRateText.getText().equals(""))
        maxRate = Integer.parseInt(maxNightlyRateText.getText());
    if(!floorNumberText.getText().equals(""))
        floorNum = Integer.parseInt(floorNumberText.getText());

    try {
        //Get all Rooms information
        ObservableList<Room> roomData = RoomDAO.searchRooms(type, numOfBeds, maxRate,
floorNum);
        //Populate Rooms on TableView
        populateRooms(roomData);
    } catch (SQLException e){
        roomErrorText.setText("Invalid search. Change input and try again.");
        System.out.println("Error occurred while getting Room information from DB.\n" +
e);
        throw e;
    }

    //View all Rooms
    @FXML
    private void viewRooms(ActionEvent actionEvent) throws SQLException,
ClassNotFoundException {

        try {
            //Get all Rooms information
            ObservableList<Room> roomData = RoomDAO.viewRooms();
            //Populate Rooms on TableView
            populateRooms(roomData);
        } catch (SQLException e){
            roomErrorText.setTextFill(Color.web("#dd0000"));
            roomErrorText.setText("An error occurred. Please try again.");
            System.out.println("Error occurred while getting Room information from DB.\n" +
e);
            throw e;
        }

        //Search all Bookings

```

```

@FXML
private void searchBookings(ActionEvent actionEvent) throws SQLException,
ClassNotFoundException {

    try {
        //Get all Rooms information
        ObservableList<Booking> bookingData = BookingDAO.searchBookings();
        //Populate Rooms on TableView
        populateBookings(bookingData);
    } catch (SQLException e){
        bookingErrorText.setTextFill(Color.web("#dd0000"));
        bookingErrorText.setText("An error occurred. Please try again.");
        System.out.println("Error occurred while getting Booking information from DB.\n"
+ e);
        throw e;
    }

    //Initializing the controller class.
    //This method is automatically called after the fxml file has been loaded.
    @FXML
    private void initialize () {

        roomNumberColumn.setCellValueFactory(cellData ->
cellData.getValue().roomNumberProperty().asObject());
        roomTypeColumn.setCellValueFactory(cellData -> cellData.getValue().typeProperty());
        roomAvailabilityColumn.setCellValueFactory(cellData ->
cellData.getValue().availabilityProperty());
        roomNumberOfBedsColumn.setCellValueFactory(cellData ->
cellData.getValue().numberOfBedsProperty().asObject());
        roomPhoneNumberColumn.setCellValueFactory(cellData ->
cellData.getValue().phoneNumberProperty());
        roomNightlyRateColumn.setCellValueFactory(cellData ->
cellData.getValue().nightlyRateProperty().asObject());
        roomWingNumberColumn.setCellValueFactory(cellData ->
cellData.getValue().wingNumberProperty().asObject());
        roomFloorNumberColumn.setCellValueFactory(cellData ->
cellData.getValue().floorNumberProperty().asObject());

        booksGuestIDColumn.setCellValueFactory(cellData ->
cellData.getValue().guestIDProperty().asObject());
        booksRoomNumberColumn.setCellValueFactory(cellData ->
cellData.getValue().roomNumberProperty().asObject());
        booksCheckInDateColumn.setCellValueFactory(cellData ->
cellData.getValue().checkInDateProperty());
        booksCheckOutDateColumn.setCellValueFactory(cellData ->
cellData.getValue().checkOutDateProperty());
        booksPackageColumn.setCellValueFactory(cellData ->
cellData.getValue().packageUsedProperty());
        booksPaymentMethodColumn.setCellValueFactory(cellData ->
cellData.getValue().paymentMethodProperty());
        booksPaymentReceivedColumn.setCellValueFactory(cellData ->
cellData.getValue().paymentReceivedProperty());

        roomTypeText.textProperty().addListener(new ChangeListener<String>() {
            @Override
            public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
                if (newValue.length() > 10) {

```

```

        roomTypeText.setText(newValue.replaceAll(newValue,
newValue.substring(0,10)));
    }
});

numberOfBedsText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (!newValue.matches("\\d*")) {
            numberOfBedsText.setText(newValue.replaceAll("[^\\d]", ""));
        }
        if (newValue.length() > 1) {
            numberOfBedsText.setText(newValue.replaceAll(newValue,
newValue.substring(0,1)));
        }
    }
});

maxNightlyRateText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (!newValue.matches("\\d*")) {
            maxNightlyRateText.setText(newValue.replaceAll("[^\\d]", ""));
        }
        if (newValue.length() > 6) {
            maxNightlyRateText.setText(newValue.replaceAll(newValue,
newValue.substring(0,6)));
        }
    }
});

floorNumberText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (!newValue.matches("\\d*")) {
            floorNumberText.setText(newValue.replaceAll("[^\\d]", ""));
        }
        if (newValue.length() > 3) {
            floorNumberText.setText(newValue.replaceAll(newValue,
newValue.substring(0,3)));
        }
    }
});

roomNumberText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (!newValue.matches("\\d*")) {
            roomNumberText.setText(newValue.replaceAll("[^\\d]", ""));
        }
        if (newValue.length() > 4) {
            roomTypeText.setText(newValue.replaceAll(newValue,
newValue.substring(0,4)));
        }
    }
});

```

```

    }
});

checkInDateText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (newValue.length() > 10) {
            checkInDateText.setText(newValue.replaceAll(newValue,
newValue.substring(0,10)));
        }
    }
});

checkOutDateText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (newValue.length() > 10) {
            checkOutDateText.setText(newValue.replaceAll(newValue,
newValue.substring(0,10)));
        }
    }
});

packageText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (newValue.length() > 10) {
            packageText.setText(newValue.replaceAll(newValue,
newValue.substring(0,10)));
        }
    }
});

guestFirstNameText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (newValue.length() > 20) {
            guestFirstNameText.setText(newValue.replaceAll(newValue,
newValue.substring(0,20)));
        }
    }
});

guestLastNameText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
        if (newValue.length() > 20) {
            guestLastNameText.setText(newValue.replaceAll(newValue,
newValue.substring(0,20)));
        }
    }
});

guestTitleText.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable, String

```

```

oldValue, String newValue) {
    if (newValue.length() > 4) {
        guestTitleText.setText(newValue.replaceAll(newValue,
newValue.substring(0, 4)));
    }
});

    guestContactNumberText.textProperty().addListener(new ChangeListener<String>() {
        @Override
        public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
            if (newValue.length() > 12) {
                guestContactNumberText.setText(newValue.replaceAll(newValue,
newValue.substring(0,12)));
            }
        }
    });

    guestHomeAddressText.textProperty().addListener(new ChangeListener<String>() {
        @Override
        public void changed(ObservableValue<? extends String> observable, String
oldValue, String newValue) {
            if (newValue.length() > 20) {
                guestHomeAddressText.setText(newValue.replaceAll(newValue,
newValue.substring(0,20)));
            }
        }
    });
}

//Populate Rooms for TableView
@FXML
private void populateRooms (ObservableList<Room> roomData) throws ClassNotFoundException
{
    //Set items to the Room Table
    availableRoomsTable.setItems(roomData);
}

//Populate Bookings for TableView
@FXML
private void populateBookings (ObservableList<Booking> bookingData) throws
ClassNotFoundException {
    //Set items to the Room Table
    bookingsTable.setItems(bookingData);
}

//Insert a Booking to the DB
@FXML
private void insertBooking (ActionEvent actionEvent) throws SQLException,
ClassNotFoundException {

    //Used to keep track of Guest ID sequence values
    // The name of the file to open.
    String fileName = "src/controller/guestIDs.txt";
    int guestID = 0;

    // This will reference one line at a time
    String line = null;

    try {

```

```

// FileReader reads text files in the default encoding.
FileReader fileReader =
    new FileReader(fileName);

// Always wrap FileReader in BufferedReader.
BufferedReader bufferedReader =
    new BufferedReader(fileReader);

while((line = bufferedReader.readLine()) != null) {
    guestID = Integer.parseInt(line) + 1;
}

// Always close files.
bufferedReader.close();
}
catch(FileNotFoundException ex) {
    System.out.println(
        "Unable to open file '" +
        fileName + "'");
}
catch(IOException ex) {
    System.out.println(
        "Error reading file '"
        + fileName + "'");
    // Or we could just do this:
    // ex.printStackTrace();
}
try{

    //true = append file
    FileWriter fileWriter = new FileWriter("src/controller/guestIDs.txt",true);
    BufferedWriter bufferWriter = new BufferedWriter(fileWriter);
    bufferWriter.write(guestID + "\n");
    bufferWriter.close();

}catch(IOException e){
    e.printStackTrace();
}

try {
    GuestDAO.insertGuest(guestID, guestTitleText.getText(),
    guestFirstNameText.getText(),

    guestLastNameText.getText(),guestContactNumberText.getText(),
    guestHomeAddressText.getText());
    //resultArea.setText("Guest inserted! \n");
} catch (SQLException e) {
    bookingErrorText.setTextFill(Color.web("#dd0000"));
    bookingErrorText.setText("Invalid search. Change input and try again.");
    throw e;
}

try {
    BookingDAO.insertBooking(guestID, Integer.parseInt(roomNumberText.getText()),
    checkInDateText.getText(),
    checkOutDateText.getText(), packageText.getText());
    bookingErrorText.setTextFill(Color.web("#000000"));
    bookingErrorText.setText("Booking Confirmed and Entered!");
    roomNumberText.setText("");
}

```

```

        checkInDateText.setText("");
        checkOutDateText.setText("");
        packageText.setText("");
        guestFirstNameText.setText("");
        guestLastNameText.setText("");
        guestContactNumberText.setText("");
        guestTitleText.setText("");
        guestHomeAddressText.setText("");

        try {
            //Get all Rooms information
            ObservableList<Booking> bookingData = BookingDAO.searchBookings();
            //Populate Rooms on TableView
            populateBookings(bookingData);
        } catch (SQLException e){
            bookingErrorText.setTextFill(Color.web("#dd0000"));
            bookingErrorText.setText("An error occurred. Please try again.");
            System.out.println("Error occurred while getting Booking information from
DB.\n" + e);
            throw e;
        }

        } catch (RuntimeException e) {
            GuestDAO.deleteGuest(guestID);
            bookingErrorText.setTextFill(Color.web("#dd0000"));
            bookingErrorText.setText("Invalid search. Change input and try again.");
            throw e;
        }
    }
}

```

Booking.java

```

package model;

import javafx.beans.property.*;
import java.sql.Date;

public class Booking {
    //Declare Booking Table Columns
    private IntegerProperty guestID;
    private IntegerProperty roomNumber;
    private SimpleObjectProperty<Date> checkInDate;
    private SimpleObjectProperty<Date> checkOutDate;
    private StringProperty packageUsed;
    private StringProperty paymentMethod;
    private StringProperty paymentReceived;

    //Constructor
    public Booking() {
        this.guestID = new SimpleIntegerProperty();
        this.roomNumber = new SimpleIntegerProperty();
        this.checkInDate = new SimpleObjectProperty<>();
        this.checkOutDate = new SimpleObjectProperty<>();
        this.packageUsed = new SimpleStringProperty();
        this.paymentMethod = new SimpleStringProperty();
    }
}

```

```

        this.paymentReceived = new SimpleStringProperty();
    }

    //guestID
    public int getGuestID() {
        return guestID.get();
    }

    public void setGuestID(int guestID){
        this.guestID.set(guestID);
    }

    public IntegerProperty guestIDProperty(){
        return guestID;
    }

    //roomNumber
    public int getRoomNumber() {
        return roomNumber.get();
    }

    public void setRoomNumber(int roomNumber){
        this.roomNumber.set(roomNumber);
    }

    public IntegerProperty roomNumberProperty(){
        return roomNumber;
    }

    //checkInDate
    public Object getCheckInDate(){
        return checkInDate.get();
    }

    public void setCheckInDate(Date checkInDate){
        this.checkInDate.set(checkInDate);
    }

    public SimpleObjectProperty<Date> checkInDateProperty(){
        return checkInDate;
    }

    //checkOutDate
    public Object getCheckOutDate(){
        return checkOutDate.get();
    }

    public void setCheckOutDate(Date checkOutDate){ this.checkOutDate.set(checkOutDate);}

    public SimpleObjectProperty<Date> checkOutDateProperty(){
        return checkOutDate;
    }

    //packageUsed
    public String getpackageUsed () { return packageUsed.get(); }

    public void setPackageUsed(String packageUsed){
        this.packageUsed.set(packageUsed);
    }

    public StringProperty packageUsedProperty() {

```



```

        return packageUsed;
    }

    //paymentMethod
    public String getPaymentMethod () { return paymentMethod.get(); }

    public void setPaymentMethod(String paymentMethod){
        this.paymentMethod.set(paymentMethod);
    }

    public StringProperty paymentMethodProperty() {
        return paymentMethod;
    }

    //paymentReceived
    public String getPaymentReceived () { return paymentReceived.get(); }

    public void setPaymentReceived(String paymentReceived){
        this.paymentReceived.set(paymentReceived);
    }

    public StringProperty paymentReceivedProperty() {
        return paymentReceived;
    }
}

```

BookingDAO.java

```

package model;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import util.DBUtil;
import java.sql.ResultSet;
import java.sql.SQLException;

public class BookingDAO {

    //*****
    //SELECT Bookings
    //*****
    public static ObservableList<Booking> searchBookings () throws SQLException,
    ClassNotFoundException {
        //Declare a SELECT statement
        String selectStmt = "SELECT * FROM books" ;

        //Execute SELECT statement
        try {
            //Get ResultSet from dbExecuteQuery method
            ResultSet rsEmps = DBUtil.dbExecuteQuery(selectStmt);

            //Send ResultSet to the getBookingList method and get booking object
            ObservableList<Booking> bookingList = getBookingList(rsEmps);

            //Return room object
            return bookingList;
        } catch (SQLException e) {
            System.out.println("SQL select operation has been failed: " + e);
        }
    }
}

```

```

        //Return exception
        throw e;
    }
}

//Select * from booking operation
private static ObservableList<Booking> getBookingList(ResultSet rs) throws SQLException,
ClassNotFoundException {
    //Declare a observable List which comprises of Booking objects
    ObservableList<Booking> bookingList = FXCollections.observableArrayList();

    while (rs.next()) {
        Booking booking = new Booking();
        booking.setRoomNumber(rs.getInt("RoomNumber"));
        booking.setGuestID(rs.getInt("GuestID"));
        booking.setCheckInDate(rs.getDate("checkInDate"));
        booking.setCheckOutDate(rs.getDate("checkOutDate"));
        booking.setPackageUsed(rs.getString("Package"));
        booking.setPaymentMethod(rs.getString("PaymentMethod"));
        booking.setPaymentReceived(rs.getString("PaymentReceived"));

        //Add booking to the ObservableList
        bookingList.add(booking);
    }
    //return bookingList (ObservableList of bookings)
    return bookingList;
}

//*****
//INSERT a Booking
//*****
public static void insertBooking (int guestID, int roomNum, String checkInDate, String
checkOutDate,
                                String packageUsed) throws SQLException,
ClassNotFoundException {
    //Declare an UPDATE statement
    String updateStmt =
        "INSERT INTO books\n" +
        "(GuestID, RoomNumber, CheckInDate, CheckOutDate, Package,
PaymentMethod, PaymentReceived)\n" +
        "VALUES\n" +
        "("+guestID+", "+roomNum+", TO_DATE('"+checkInDate+" 15:00:00', " +
        "'yy/mm/dd hh24:mi:ss'), TO_DATE('"+checkOutDate+" 11:00:00',
'yy/mm/dd hh24:mi:ss')," +
        " '"+packageUsed+"', 'Visa', 'N')";

    //Execute UPDATE operation
    try {
        DBUtil.dbExecuteUpdate(updateStmt);
    } catch (SQLException e) {
        System.out.print("Error occurred while INSERT Operation: " + e);
        throw e;
    }
}
}

```

Guest.java

```
package model;
```

```

import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Guest {
    //Declare Guest Table Columns
    private IntegerProperty guestID;
    private StringProperty title;
    private StringProperty firstName;
    private StringProperty lastName;
    private StringProperty contactNumber;
    private StringProperty homeAddress;

    //Constructor
    public Guest() {
        this.guestID = new SimpleIntegerProperty();
        this.title = new SimpleStringProperty();
        this.firstName = new SimpleStringProperty();
        this.lastName = new SimpleStringProperty();
        this.contactNumber = new SimpleStringProperty();
        this.homeAddress = new SimpleStringProperty();
    }

    //guestID
    public int getGuestID() {
        return guestID.get();
    }

    public void setGuestID(int guestID){
        this.guestID.set(guestID);
    }

    public IntegerProperty guestIDProperty(){
        return guestID;
    }

    //title
    public String getTitle () {
        return title.get();
    }

    public void setTitle(String title){
        this.title.set(title);
    }

    public StringProperty titleProperty() { return title; }

    //firstName
    public String getFirstName () {
        return firstName.get();
    }

    public void setFirstName(String firstName){
        this.firstName.set(firstName);
    }

    public StringProperty firstNameProperty() {
        return firstName;
    }
}

```

```

//lastName
public String getLastName () { return lastName.get(); }

public void setLastName(String lastName){
    this.lastName.set(lastName);
}

public StringProperty lastNameProperty() {
    return lastName;
}

//contactNumber
public String getContactNumber() {
    return contactNumber.get();
}

public void setContactNumber(String contactNumber){
this.contactNumber.set(contactNumber); }

public StringProperty contactNumberProperty(){
    return contactNumber;
}

//homeAddress
public String getHomeAddress () {
    return homeAddress.get();
}

public void setHomeAddress(String homeAddress){
    this.homeAddress.set(homeAddress);
}

public StringProperty homeAddressProperty() { return homeAddress; }
}

```

GuestDAO.java

```

package model;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import util.DBUtil;
import java.io.*;

import java.sql.ResultSet;
import java.sql.SQLException;

public class GuestDAO {

    //Select * from guest operation
    private static ObservableList<Guest> getGuestList(ResultSet rs) throws SQLException,
ClassNotFoundException {
        //Declare a observable List which comprises of Guest objects
        ObservableList<Guest> guestList = FXCollections.observableArrayList();

        while (rs.next()) {
            Guest guest = new Guest();

```

```

        guest.setGuestID(rs.getInt("GuestID"));
        guest.setTitle(rs.getString("Title"));
        guest.setFirstName(rs.getString("FirstName"));
        guest.setLastName(rs.getString("LastName"));
        guest.setContactNumber(rs.getString("ContactNumber"));
        guest.setHomeAddress(rs.getString("HomeAddress"));
        //Add guests to the ObservableList
        guestList.add(guest);
    }
    //return guestList (ObservableList of guests)
    return guestList;
}

//*****
//SELECT all Guests
//*****
public static ObservableList<Guest> viewGuests () throws SQLException,
ClassNotFoundException {
    //Declare a SELECT statement
    String selectStmt = "SELECT * FROM guest";

    //Execute SELECT statement
    try {
        //Get ResultSet from dbExecuteQuery method
        ResultSet rsEmps = DBUtil.dbExecuteQuery(selectStmt);

        //Send ResultSet to the getRoomList method and get room object
        ObservableList<Guest> guestList = getGuestList(rsEmps);

        //Return room object
        return guestList;
    } catch (SQLException e) {
        System.out.println("SQL select operation has been failed: " + e);
        //Return exception
        throw e;
    }
}

//*****
//INSERT a Guest
//*****
public static void insertGuest (int guestID, String title, String firstName,
                                String lastName, String contactNum, String homeAddr)
                                throws SQLException, ClassNotFoundException {

    //Declare an UPDATE statement
    String updateStmt =
        "INSERT INTO guest\n" +
        "(GuestID, Title, FirstName, LastName, ContactNumber, HomeAddress)\n"
+
        "VALUES\n" +
        "("+guestID+", '"+title+"', '"+firstName+"', '"+lastName+"',
        '"+contactNum+"', '"+homeAddr+"')";

    //Execute INSERT operation
    try {
        DBUtil.dbExecuteUpdate(updateStmt);
    } catch (SQLException e) {
        System.out.print("Error occurred while INSERT Operation: " + e);
        throw e;
    }
}

```

```

    }

    //*****
    //Delete a Guest
    //*****
    public static void deleteGuest (int guestID) throws SQLException, ClassNotFoundException
    {
        //Declare an UPDATE statement
        String updateStmt =
            "DELETE FROM guest WHERE GuestID = "+guestID ;

        //Execute INSERT operation
        try {
            DBUtil.dbExecuteUpdate(updateStmt);
        } catch (SQLException e) {
            System.out.print("Error occurred while DELETE Operation: " + e);
            throw e;
        }
    }
}

```

Room.java

```

package model;

import javafx.beans.property.*;
import java.sql.Date;

public class Room {
    //Declare Room Table Columns
    private IntegerProperty roomNumber;
    private StringProperty type;
    private StringProperty availability;
    private IntegerProperty numberOfBeds;
    private StringProperty phoneNumber;
    private IntegerProperty nightlyRate;
    private IntegerProperty wingNumber;
    private IntegerProperty floorNumber;

    //Constructor
    public Room() {
        this.roomNumber = new SimpleIntegerProperty();
        this.type = new SimpleStringProperty();
        this.availability = new SimpleStringProperty();
        this.numberOfBeds = new SimpleIntegerProperty();
        this.phoneNumber = new SimpleStringProperty();
        this.nightlyRate = new SimpleIntegerProperty();
        this.wingNumber = new SimpleIntegerProperty();
        this.floorNumber = new SimpleIntegerProperty();
    }

    //roomNumber
    public int getroomNumber() {
        return roomNumber.get();
    }

    public void setRoomNumber(int roomNumber){
        this.roomNumber.set(roomNumber);
    }
}

```

```

public IntegerProperty roomNumberProperty(){
    return roomNumber;
}

//type
public String getType () {
    return type.get();
}

public void setType(String type){
    this.type.set(type);
}

public StringProperty typeProperty() { return type; }

//availability
public String getAvailability () {
    return availability.get();
}

public void setAvailability(String availability){
    this.availability.set(availability);
}

public StringProperty availabilityProperty() {
    return availability;
}

//numberOfBeds
public int getNumberOfBeds () {
    return numberOfBeds.get();
}

public void setnumberOfBeds(int numberOfBeds){
    this.numberOfBeds.set(numberOfBeds);
}

public IntegerProperty numberOfBedsProperty() {
    return numberOfBeds;
}

//phoneNumber
public String getphoneNumber() {
    return phoneNumber.get();
}

public void setphoneNumber(String phoneNumber){ this.phoneNumber.set(phoneNumber); }

public StringProperty phoneNumberProperty(){
    return phoneNumber;
}

//nightlyRate
public int getNightlyRate () {
    return nightlyRate.get();
}

public void setNightlyRate(int nightlyRate){
    this.nightlyRate.set(nightlyRate);
}

```

```

    }

    public IntegerProperty nightlyRateProperty() {
        return nightlyRate;
    }

    //wingNumber
    public int getWingNumber () {
        return wingNumber.get();
    }

    public void setWingNumber(int wingNumber){
        this.wingNumber.set(wingNumber);
    }

    public IntegerProperty wingNumberProperty() {
        return wingNumber;
    }

    //floorNumber
    public int getFloorNumber () {
        return floorNumber.get();
    }

    public void setFloorNumber(int floorNumber){
        this.floorNumber.set(floorNumber);
    }

    public IntegerProperty floorNumberProperty() { return floorNumber; }
}

```

RoomDAO.java

```

package model;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import util.DBUtil;

import java.sql.ResultSet;
import java.sql.SQLException;

public class RoomDAO {

    //*****
    //SELECT available Rooms with search criteria
    //*****
    public static ObservableList<Room> searchRooms (String type, int numOfBeds, int maxRate,
    int floorNum)
        throws SQLException,
        ClassNotFoundException {
        //Declare a SELECT statement
        String availabilityY = "'Y'";
        String selectStmt = "SELECT * FROM room WHERE Availability =" + availabilityY;

        if(type != null)
            selectStmt += " AND Type = '" + type + "'";
        if(numOfBeds != 0)
            selectStmt += " AND NumberOfBeds = " + numOfBeds;
    }
}

```



```

        if(maxRate != 0)
            selectStmt += " AND CostPerNight <= " + maxRate;
        if(floorNum != 0)
            selectStmt += " AND floorNumber = " + floorNum;

        //Execute SELECT statement
        try {
            //Get ResultSet from dbExecuteQuery method
            ResultSet rsEmps = DBUtil.dbExecuteQuery(selectStmt);

            //Send ResultSet to the getRoomList method and get room object
            ObservableList<Room> roomList = getRoomList(rsEmps);

            //Return room object
            return roomList;
        } catch (SQLException e) {
            System.out.println("SQL select operation has been failed: " + e);
            //Return exception
            throw e;
        }
    }

    /*******
    //SELECT all available Rooms
    /*******
    public static ObservableList<Room> viewRooms () throws SQLException,
    ClassNotFoundException {
        //Declare a SELECT statement
        String availabilityY = "'Y'";
        String selectStmt = "SELECT * FROM room WHERE Availability =" + availabilityY;

        //Execute SELECT statement
        try {
            //Get ResultSet from dbExecuteQuery method
            ResultSet rsEmps = DBUtil.dbExecuteQuery(selectStmt);

            //Send ResultSet to the getRoomList method and get room object
            ObservableList<Room> roomList = getRoomList(rsEmps);

            //Return room object
            return roomList;
        } catch (SQLException e) {
            System.out.println("SQL select operation has been failed: " + e);
            //Return exception
            throw e;
        }
    }

    //Select * from room operation
    private static ObservableList<Room> getRoomList(ResultSet rs) throws SQLException,
    ClassNotFoundException {
        //Declare a observable List which comprises of Room objects
        ObservableList<Room> roomList = FXCollections.observableArrayList();

        while (rs.next()) {
            Room room = new Room();
            room.setRoomNumber(rs.getInt("RoomNumber"));
            room.setType(rs.getString("Type"));
            room.setAvailability(rs.getString("Availability"));
            room.setnumberOfBeds(rs.getInt("NumberOfBeds"));
        }
    }

```

```

        room.setphoneNumber(rs.getString("PhoneNumber"));
        room.setNightlyRate(rs.getInt("CostPerNight"));
        room.setWingNumber(rs.getInt("WingNumber"));
        room.setFloorNumber(rs.getInt("FloorNumber"));
        //Add room to the ObservableList
        roomList.add(room);
    }
    //return roomList (ObservableList of rooms)
    return roomList;
}
}

```

Conclusion

Working on this Hotel Database Management System has provided us with a solid foundation in all aspects of database design and implementation. Previously, we had not realized how important it was to represent data in a clear and concise way. With the theories we learned regarding entity-relationship diagrams, relational schema design, functional dependencies, normalization, etc., we were able to turn various pieces of data into a useful and accessible database.

On the technical side, we became accustomed to using SQL and the services provided by Oracle. Through this, we learned what it was like to create tables, drop tables, insert data, and query information. Making a GUI using Java also familiarized us with how a front-end interface connects and interacts with a back-end database.

This project also exercised our skills in teamwork, project management, and software development.

Overall, it was a pleasure working on this hotel management database. It truly allowed us to use the knowledge and skills acquired in this course.