CPS 510 - Final Report
Section 8
Team 14

Hotel Database Management System

Professor
Soheila Bashardoust Tajali

Team
Japneet Dhaliwal
Busola Elumeze
Faizan Noor

Type	**Member and Percent**
A1 - Description/Basic Functions	Japneet Dhaliwal - 50% Busola Elumeze - 25% Faizan Noor - 25%
A2 - Entity Diagram & Constraints	Japneet Dhaliwal - 100%
A3, A6, A7, A8 - Database Normalization	Japneet Dhaliwal - 85% Busola Elumeze - 10% Faizan Noor - 5%
A4 - Simple/Advanced Queries	Japneet Dhaliwal - 100%
A5 - Unix Shell Implementation	Japneet Dhaliwal - 100%
A9 - Java Implementation	Japneet Dhaliwal - 100%
A10 - Final Report	Japneet Dhaliwal - 100%

Description

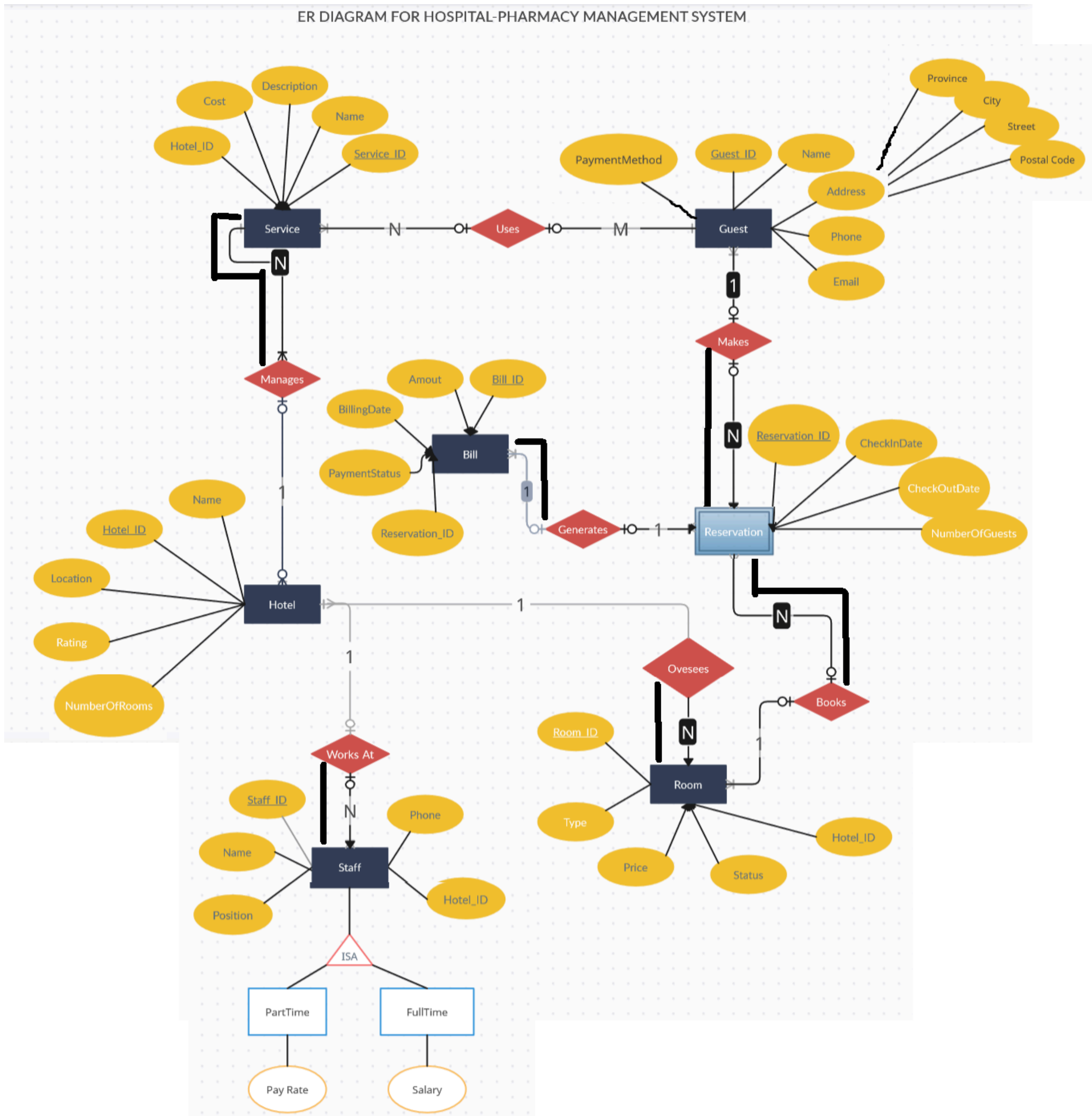
The Hotel Management Database Management System is designed as the core platform for organizing and handling the essential data required to run a hotel efficiently. It carefully stores and manages information related to guests, reservations, room availability, billing, staff, and services. The system ensures that day-to-day tasks such as guest check-ins, room bookings, invoicing, payments, staff scheduling, and service management are seamlessly handled. The complete system's end users will primarily be hotel staff interacting with the system regularly to meet guest needs and manage hotel resources. The primary objective of this database management system is to enhance the operational efficiency and accuracy of hotel processes, delivering a smooth experience for both guests and staff.

Basic Functions

The following list includes some of the general potential functions of this DBMS:

Functions	Description
InsertGuest	Register a new guests information when they check into Hotel
ViewGuestInfo	Request/display guest information
ViewAvailability	Check the availability of rooms based on specific criteria (type, price, status)
ManageStaff	Modify and monitor information about employees working in the Hotel
GenerateBill	Generates billing information for reservation (invoice)
AddStaff	Adds new staff members to the system
ManageService	Modify and monitor information about services operating in the Hotel

Entity Diagram:



Entity	Attributes
Guest (Strong Entity)	• Guest_ID (Primary Key) • Name • Address (Composite Attribute) • Phone • Email • PaymentMethod

Room (Strong Entity)	• Room_ID (Primary Key) • Type • Price • Status • Hotel_ID (Foreign Key)
Hotel (Strong Entity)	• Hotel_ID (Primary Key) • Name • Location • Rating • NumberOfRooms
Reservation (Weak Entity)	• Reservation_ID (Primary Key) • CheckInDate • CheckOutDate • NumberOfGuests • Guest_ID (Foreign Key) • Room_ID (Foreign Key)
Service (Strong Entity)	• Service_ID (Primary Key) • Name • Description • Cost • Hotel_ID (Foreign Key)
Bill (Strong Entity)	• Bill_ID (Primary Key) • Amount • BillingDate • PaymentStatus • Reservation_ID (Foreign Key)
Staff (Strong Entity)	• Staff_ID (Primary Key) • Name • Position • Phone • Hotel_ID (Foreign Key) • Disjoint Specialization (Part-time or Full-time)

Relationships & Constraints (Mapping/Participation):

Relationship	Constraint
Guest Makes Reservation	● **Guest (1) ---< Reservation (Many)** ● **Mapping Constraint**: One-to-Many (A guest can make many reservations, but each reservation is made by one guest) ● **Participation Constraint**: ○ **Total for Reservation:** Every reservation must be associated with a guest. ○ **Partial for Guest:** Not all guests need to have made a reservation.
Reservation Books Room	● **Reservation (Many) ---< Room (One)** ● **Mapping Constraint**: Many-to-One (A room can be booked in many reservations over time, but each reservation books only one room) ● **Participation Constraint**: ○ **Total for Reservation:** Every reservation must book a room. ○ **Partial for Room:** Not all rooms need to be associated with reservations.
Reservation Generates Bill	● **Reservation (1) ---< Bill (1)** ● **Mapping Constraint**: One-to-One (Each reservation generates one bill, and each bill is associated with one reservation) ● **Participation Constraint**: ○ **Total for Bill:** Every bill must be associated with a reservation. ○ **Partial for Reservation:** Not every reservation will generate a bill immediately.
Hotel Oversees Room	● **Hotel (1) ---< Room (Many)**

	● **Mapping Constraint**: One-to-Many (A hotel can have many rooms, but each room belongs to only one hotel) ● **Participation Constraint**: ○ **Total for Room:** Every room must belong to a hotel. ○ **Partial for Hotel:** A hotel may or may not have rooms available.
Hotel Manages Service	● **Hotel (One) ---< Service (Many)** ● **Mapping Constraint**: One-to-Many (A hotel can offer many services, but each service belongs to only one hotel) ● **Participation Constraint**: ○ **Total for Service:** Every service must be associated with a hotel. ○ **Partial for Hotel:** Not all hotels need to offer services.
Guest Uses Service	● **Guest (Many) >---< Service (Many)** ● **Mapping Constraint**: Many-to-Many (A guest can use many services, and each service can be used by many guests) ● **Participation Constraint**: ○ **Partial for Guest:** A guest may or may not use services. ○ **Partial for Service:** A service may or may not be used by guests.
Staff Works At Hotel	● **Hotel (1) ---< Staff (Many)** ● **Mapping Constraint**: Many-to-One (A hotel can employ many staff, but each staff member works for one hotel) ● **Participation Constraint**: ○ **Total for Hotel:** Every staff member must work at a hotel. ○ **Partial for Staff:** Not all hotels need to have staff immediately.

Database Schema & Normalization:

1. **Hotel Table**
    - **Schema**: **Hotel**(<u>Hotel_ID</u>, Name, Location, Rating, NumberOfRooms)
- **Primary Key:** Hotel_ID
- **Functional Dependencies:**
    - {Hotel_ID} → {Name, Location, Rating, NumberOfRooms}
- **Normalization**:
    - This table is in **1NF** because all values are atomic.
    - It is in **2NF** because there are no partial dependencies; all non-key attributes (**`Name`, `Location`, `Rating`, `NumberOfRooms`**) are fully functionally dependent on the primary key, **`Hotel_ID`**.
    - It is in **3NF** as there are no transitive dependencies because all non-key attributes depend directly on the primary key, **`Hotel_ID`**.
- **BCNF**:
    - The only functional dependency is {Hotel_ID} → {Name, Location, Rating, NumberOfRooms}.
    - The determinant **Hotel_ID** is the primary key, and it uniquely identifies each hotel record, so this table satisfies BCNF.

2. **Guest Table**
    - **Schema**: **Guest** (<u>Guest_ID</u>, Name, Address_ID, PhoneNumber, Email, PaymentMethod)
- **Primary Key:** Guest_ID
- **Functional Dependencies:**
    - {Guest_ID} → {Name, Address_ID, PhoneNumber, Email, PaymentMethod}
- **Normalization**:
    - This table is in **1NF** because all values are atomic.
    - It is in **2NF** because there are no partial dependencies; all non-key attributes (**`Name`, `Street`, `PostalCode`, `City`, `Province`, `PhoneNumber`, `Email`, `PaymentMethod`**) are fully functionally dependent on the primary key, **`Guest_ID`**.
    - It is in **3NF** as there are no transitive dependencies because all non-key attributes depend directly on the primary key, **`Guest_ID`**.
- **BCNF**:
    - The only functional dependency is {Guest_ID} → {Name, Address_ID, PhoneNumber, Email, PaymentMethod}.
    - The determinant **Guest_ID** is the primary key, and it uniquely identifies each guest record, so this table satisfies BCNF.

3. **Address Table**
    - **Schema**: **Address**(<u>Address_ID</u>, Street, PostalCode, City, Province)
- **Primary Key**: Address_ID
- **Functional Dependencies**:

    - {Address_ID} → {Street, PostalCode, City, Province}
- **Normalization**:
    - This table is in **1NF** because all attributes are atomic.
    - It is in **2NF** because there are no partial dependencies; all non-key attributes (Street, PostalCode, City, Province) are fully functionally dependent on the primary key, **Address_ID**.
    - It is in **3NF** as there are no transitive dependencies because all non-key attributes depend directly on the primary key, **Address_ID**.
- **BCNF**:
    - The only functional dependency is {Address_ID} → {Street, PostalCode, City, Province}.
    - The determinant **Address_ID** is the primary key, and it uniquely identifies each address record, so this table satisfies BCNF.

4. **Room Table**
    - **Schema**: **Room**(Room_ID, RoomType, Price, AvailabilityStatus, Hotel_ID)
- **Primary Key:** Room_ID
- **Functional Dependencies:**
    - {Room_ID} → {Hotel_ID, RoomType, Price, AvailabilityStatus}
- **Normalization**:
    - This table is in **1NF** because all values are atomic.
    - It is in **2NF** because there are no partial dependencies; all non-key attributes (**`Hotel_ID`**, **`RoomType`**, **`Price`**, **`AvailabilityStatus`**) are fully functionally dependent on the primary key, **`Room_ID`**.
    - It is in **3NF** as there are no transitive dependencies because all non-key attributes depend directly on the primary key, **Room_ID**.
- **BCNF**:
    - The only functional dependency is {Room_ID} → {Hotel_ID, RoomType, Price, AvailabilityStatus}.
    - The determinant **Room_ID** is the primary key, and it uniquely identifies each room record, so this table satisfies BCNF.

5. **Reservation Table**
    - **Schema**: **Reservation**(Reservation_ID, Guest_ID, CheckInDate, CheckOutDate, NumberOfGuests, Room_ID)
- **Primary Key:** Reservation_ID, Guest_ID
- **Functional Dependencies:**
    - {Reservation_ID, Guest_ID} → {Room_ID, CheckInDate, CheckOutDate, NumberOfGuests}
- **Normalization**:
    - This table is in **1NF** because all values are atomic.
    - It is in **2NF** because there are no partial dependencies; all non-key attributes (**`Room_ID`**, **`CheckInDate`**, **`CheckOutDate`**, **`NumberOfGuests`**) are fully

functionally dependent on the composite primary key, consisting of **Reservation_ID** and **Guest_ID**.
    - It is in **3NF** as there are no transitive dependencies because all non-key attributes depend directly on the composite primary key, consisting of **Reservation_ID** and **Guest_ID**.
- **BCNF**:
    - The only functional dependency is {Reservation_ID, Guest_ID} → {Room_ID, CheckInDate, CheckOutDate, NumberOfGuests}.
    - The determinant **{Reservation_ID, Guest_ID}** is the primary key, and it uniquely identifies each reservation record, so this table satisfies BCNF.

6. **Bill Table**
    - **Schema**: **Bill**(Bill_ID, Amount, PaymentStatus, BillingDate, Reservation_ID)
- **Primary Key:** Bill_ID
- **Functional Dependencies:**
    - {Bill_ID} → {Reservation_ID, Amount, PaymentStatus, BillingDate}
- **Normalization**:
    - This table is in **1NF** because all values are atomic.
    - It is in **2NF** because there are no partial dependencies; all non-key attributes (**Reservation_ID**, **Amount**, **PaymentStatus**, **BillingDate**) are fully functionally dependent on the primary key, **Bill_ID**.
    - It is in **3NF** as there are no transitive dependencies because all non-key attributes depend directly on the primary key, **Bill_ID**.
- **BCNF**:
    - The only functional dependency is {Bill_ID} → {Reservation_ID, Amount, PaymentStatus, BillingDate}.
    - The determinant **Bill_ID** is the primary key, and it uniquely identifies each bill record, so this table satisfies BCNF.

7. **Staff Table**
    - **Schema**: **Staff**(Staff_ID, Name, Position, PhoneNumber, Hotel_ID)
- **Primary Key:** Staff_ID
- **Functional Dependencies:**
    - {Staff_ID} → {Hotel_ID, Name, Position, PhoneNumber}
- **Normalization**:
    - This table is in **1NF** because all values are atomic.
    - It is in **2NF** because there are no partial dependencies; all non-key attributes (**Hotel_ID**, **Name**, **Position**, **PhoneNumber**) are fully functionally dependent on the primary key, **Staff_ID**.
    - It is in **3NF** as there are no transitive dependencies because all non-key attributes depend directly on the primary key, **Staff_ID**.
- **BCNF**:

- ○ The only functional dependency is {Staff_ID} → {Hotel_ID, Name, Position, PhoneNumber}.
- ○ The determinant **Staff_ID** is the primary key, and it uniquely identifies each staff record, so this table satisfies BCNF.

8. **FullTime_Staff Table**
   - **Schema**: **FullTime_Staff**(Staff_ID, Salary)
- **Primary Key:** Staff_ID
   - **Functional Dependencies:**
   - {Staff_ID} → {Salary}
- **Normalization**:
   - This table is in **1NF** because all values are atomic.
   - It is in **2NF** because the non-key attribute (`Salary`) is fully functionally dependent on the primary key, `Staff_ID`.
   - It is in **3NF** because no transitive dependencies exist.
- **BCNF**:
   - The only functional dependency is {Staff_ID} → {Salary}.
   - The determinant **Staff_ID** is the primary key, and it uniquely identifies each staff record, so this table satisfies BCNF.

9. **PartTime_Staff Table**
   - **Schema**: **PartTime_Staff**(Staff_ID, HourlyRate)
- **Primary Key:** Staff_ID
- **Functional Dependencies:**
   - {Staff_ID} → {HourlyRate}
- **Normalization**:
   - This table is in **1NF** because all values are atomic.
   - It is in **2NF** because the non-key attribute (`HourlyRate`) is fully functionally dependent on the primary key, `Staff_ID`.
   - It is in **3NF** because no transitive dependencies exist.
- **BCNF**:
   - The only functional dependency is {Staff_ID} → {HourlyRate}.
   - The determinant **Staff_ID** is the primary key, and it uniquely identifies each staff record, so this table satisfies BCNF.

10. **Service Table**
   - **Schema**: **Service**(Service_ID, Name, Price, Description, Hotel_ID)
- **Primary Key:** Service_ID
- **Functional Dependencies:**
   - {Service_ID} → {Hotel_ID, Name, Price, Description}
- **Normalization**:
   - This table is in **1NF** because all values are atomic.

    - It is in **2NF** because there are no partial dependencies; all non-key attributes (**`Hotel_ID`**, **`Name`**, **`Price`**, **`Description`**) are fully functionally dependent on the primary key, **`Service_ID`**.
    - It is in **3NF** as there are no transitive dependencies because all non-key attributes depend directly on the primary key, **Service_ID**.
- **BCNF**:
    - The only functional dependency is {Service_ID} → {Hotel_ID, Name, Price, Description}.
    - The determinant **Service_ID** is the primary key, and it uniquely identifies each service record, so this table satisfies BCNF.

11. Uses Table

    - **Schema**: **Uses**(<u>Guest_ID, Service_ID</u>)
- **Primary Key:** (Guest_ID, Service_ID)
- **Functional Dependencies:**
    - {Guest_ID, Service_ID} → { }
        - No Functional dependencies
- **Normalization**:
    - This table has no functional dependencies

Simple Database Queries (SQL & Relational Algebra)

1. Guest

SELECT Guest_ID, Name, PhoneNumber, Email
FROM Guest
WHERE PaymentMethod = ‘Card’
ORDER BY Name;

GUEST_ID	NAME	PHONENUMBER	EMAIL
2	Jane Smith	905-555-5678	jane.smith@gmail.com
4	Sarah Lee	905-555-9999	sarah.lee@gmail.com

Relational Algebra: π_Guest_ID, Name, PhoneNumber, Email (σ_PaymentMethod='Card' (Guest))

2. Address

SELECT DISTINCT City
FROM Address
ORDER BY City;

Relational Algebra: π_City (Address)

CITY
Brampton
Mississauga
Toronto

3. Room

SELECT Room_ID, RoomType, Price
FROM Room
WHERE AvailabilityStatus = 'Available'
ORDER BY Price DESC;

Relational Algebra: τ_Price DESC (π_Room_ID, RoomType, Price (σ_AvailabilityStatus = 'Available' (Room)))

ROOM_ID	ROOMTYPE	PRICE
103	Suite	200
101	Deluxe	150

4. **Hotel**

SELECT Name, Location, Rating, NumberOfRooms
FROM Hotel
WHERE NumberOfRooms > 100
ORDER BY Rating DESC;

Relational Algebra: τ_Rating DESC (π_Name, Location, Rating, NumberOfRooms (σ_NumberOfRooms > 100 (Hotel)))

NAME	LOCATION	RATING	NUMBEROFROOMS
Marriott Downtown	101 City Center Dr, Mississauga	5	500
Holiday Inn Richmond	789 Hotel Road, Toronto	4	700
Best Western Plus	202 Lakeshore Rd, Oakville	3	250

5. **Reservation**

SELECT Room_ID, COUNT(Reservation_ID) AS ReservationCount
FROM Reservation
GROUP BY Room_ID;

Relational Algebra: γ_Room_ID; COUNT(Reservation_ID → ReservationCount) (Reservation)

ROOM_ID	RESERVATIONCOUNT
102	1
101	1
103	1

6. **Bill**

SELECT Bill_ID, Amount
FROM Bill
WHERE PaymentStatus = 'Unpaid'
ORDER BY BillingDate DESC;

Relational Algebra: π_Bill_ID, Amount (σ_PaymentStatus='Unpaid' (Bill))

BILL_ID	AMOUNT
2002	480

7. **Staff**

SELECT Name, Position
FROM Staff
WHERE Hotel_ID = 1
ORDER BY Position;

Relational Algebra: π_Name, Position (σ_Hotel_ID=1 (Staff))

NAME	POSITION
Emily White	Manager
Paul Green	Receptionist

8. **Full-time Employee**

SELECT Staff_ID, Salary
FROM FullTime_Staff
ORDER BY Salary ASC;

Relational Algebra: τ_Salary ASC (π_Staff_ID, Salary (FullTime_Staff))

STAFF_ID	SALARY
3002	45000
3001	65000

9. **Part-time Employee**

SELECT Staff_ID, HourlyRate
FROM PartTime_Staff
ORDER BY HourlyRate DESC;

Relational Algebra: τ_HourlyRate DESC (π_Staff_ID, HourlyRate (PartTime_Staff))

STAFF_ID	HOURLYRATE
3003	16.5

10. **Service**

SELECT Name, Price
FROM Service
WHERE Hotel_ID = 1
ORDER BY Price DESC;

Relational Algebra: τ_Price DESC (π_Name, Price (σ_Hotel_ID=1 (Service)))

NAME	PRICE
Spa	100
Gym	50

11. Uses

SELECT Guest_ID, COUNT(Service_ID) AS ServiceCount
FROM Uses
GROUP BY Guest_ID
ORDER BY ServiceCount DESC;

Relational Algebra: τ_ServiceCount DESC (γ_Guest_ID; COUNT(Service_ID → ServiceCount) (Uses))

GUEST_ID	SERVICECOUNT
1	1
3	1
2	1

Advanced Database Queries (SQL & Relational Algebra)

Query1: Lists all guests who have booked rooms in hotels with a 5-star rating

```
SELECT g.Name, COUNT(r.Reservation_ID) AS ReservationCount
FROM Guest g, Reservation r, Room ro, Hotel h
WHERE g.Guest_ID = r.Guest_ID
AND r.Room_ID = ro.Room_ID
AND ro.Hotel_ID = h.Hotel_ID
AND h.Rating = 5
GROUP BY Guest.Name;
```

Relational Algebra: π_Name, COUNT(Reservation_ID → ReservationCount) (σ_Rating=5 (Guest ⋈ Reservation ⋈ Room ⋈ Hotel))

```
SQL>   2    3    4    5    6    7
NAME                                                    RESERVATIONCOUNT
------------------------------------------------------- ----------------
Sarah Lee                                                              1
```

Query2: List Hotels with no part-time staff working

```
SELECT Hotel_ID, Name
FROM Hotel
MINUS
SELECT Hotel_ID, Name
FROM Hotel
WHERE Hotel_ID IN (
   SELECT Hotel_ID
   FROM Staff
   WHERE Staff_ID IN (SELECT Staff_ID FROM PartTime_Staff)
);
```

Relational Algebra: π_Hotel_ID, Name (Hotel) - π_Hotel_ID, Name (σ_Staff_ID ∈ π_Staff_ID (PartTime_Staff) (Staff ⋈ Hotel))

```
SQL>   2    3    4    5    6    7    8    9   10
  HOTEL_ID NAME
---------- --------------------------------------------------
         1 Holiday Inn Richmond
```

Query3: Lists full-time staff members whose salary is greater than the average salary of all full-time staff at their hotel.

```
SELECT
   'Name: ' || S.Name || ', Position: ' || S.Position || ', Salary: ' || F.Salary || ', Hotel: ' || H.Name AS
"High-Paying Full-Time Staff"
FROM
   Staff S, FullTime_Staff F, Hotel H
WHERE
   S.Staff_ID = F.Staff_ID
   AND S.Hotel_ID = H.Hotel_ID
   AND F.Salary > (
      SELECT AVG(FT.Salary)
      FROM FullTime_Staff FT
      WHERE FT.Staff_ID IN (
         SELECT Staff_ID
         FROM Staff
         WHERE Hotel_ID = S.Hotel_ID
      )
   );
```

Relational Algebra: π_Name, Position, Salary, Hotel_Name (σ_Salary > (γ_AVG(Salary) (FullTime_Staff)) (Staff ⋈ FullTime_Staff ⋈ Hotel))

```
SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
High-Paying Full-Time Staff
--------------------------------------------------------------------------------
Name: Emily White, Position: Manager, Salary: 65000, Hotel: Holiday Inn Richmond
```

Query4: Lists all Staff members, including their ID, Name, and Employment Type

```
SELECT Staff_ID, Name, 'Full-Time' AS EmploymentType
FROM Staff
WHERE Staff_ID IN (SELECT Staff_ID FROM FullTime_Staff)
UNION
SELECT Staff_ID, Name, 'Part-Time' AS EmploymentType
FROM Staff
WHERE Staff_ID IN (SELECT Staff_ID FROM PartTime_Staff);
```

Relational Algebra: (π_Staff_ID, Name, 'Full-Time' (σ_Staff_ID ∈ π_Staff_ID (FullTime_Staff) (Staff))) ∪
(π_Staff_ID, Name, 'Part-Time' (σ_Staff_ID ∈ π_Staff_ID (PartTime_Staff) (Staff)))

```
SQL>   2    3    4    5    6    7
  STAFF_ID NAME                                                 EMPLOYMEN
---------- ---------------------------------------------------- ---------
      3001 Emily White                                          Full-Time
      3002 Paul Green                                           Full-Time
      3003 Anna Brown                                           Part-Time
      3004 Tom Black                                            Full-Time
      3005 Rachel Adams                                         Part-Time
```

Query5: Lists name of all Hotels with available rooms

```
SELECT Name
FROM Hotel h
WHERE EXISTS (
  SELECT 1
  FROM Room r
  WHERE r.Hotel_ID = h.Hotel_ID
    AND r.AvailabilityStatus = 'Available'
);
```

Relational Algebra: π_Name (Hotel ⋈ (π_Hotel_ID (σ_AvailabilityStatus='Available' (Room))))

```
SQL>   2    3    4    5    6    7    8
NAME
--------------------------------------------------
Holiday Inn Richmond
Marriott Downtown
```

UNIX Shell Implementation

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

The list of scripts is as shown:

```
create_tables.sh   menu.sh               queries.sh  Query2  Query4
drop_tables.sh     populate_tables.sh    Query1      Query3  Query5
```

Our Hotel DBMS is executed through the main bash script, ‘menu.sh’, which displays the cases to create, drop, and populate tables, along with viewing the queries.

This is the main screen displayed through menu.sh:

```
j27dhali@metis:~/cps510$ ./menu.sh


Welcome to the Hotel Database Management System! Type the corresponding digit to get said result!


1: Create Tables
2: Populate Tables
3: Drop Tables
4: Query Tables
q: Quit/Exit
```

Each command (1-4, q) references the other bash script files to execute the following command.

Below is the code for our **menu.sh**:

```
#!/bin/bash

echo $'\n\nWelcome to the Hotel Database Management System! Type the corresponding digit to get said result!\n\n'
echo '1: Create Tables'
echo '2: Populate Tables'
echo '3: Drop Tables'
echo '4: Query Tables'
echo 'q: Quit/Exit'

read input

while [ "$input" != "q" ] && [ "$input" != "Q" ];
do
    if [ "$input" = "1" ]; then
        echo 'You have chosen option 1'
        ./create_tables.sh
```

```
    elif [ "$input" = "2" ]; then
        echo 'You have chosen option 2'
        ./populate_tables.sh
    elif [ "$input" = "3" ]; then
        echo 'You have chosen option 3'
        ./drop_tables.sh
    elif [ "$input" = "4" ]; then
        echo 'You have chosen option 4'
        ./queries.sh
    else
        echo 'Invalid Input'
    fi

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

    read input
done
```

Java Implementation

For our Java implementation, we essentially designed a version of our Unix Implementation through Java.

This implementation runs through the terminal. It will greet the user in the main page and ask for input to do any of the following:

1. Create Tables
2. Drop Tables
3. Populate Tables
4. View Query 1
5. View Query 2
6. View Query 3
7. View Query 4
8. View Query 5
9. Exit

The user is able to create, drop, and populate tables. Additionally, they can view all of the five advanced queries described above. Below are screenshots of their outputs:

Case 1: Create Tables

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 1
Tables created successfully.
```

Case 2: Drop Tables

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 2
Tables dropped successfully.
```

Case 3: Populate Tables

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 3
Tables populated successfully.
```

Case 4: Query 1

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 4
Guests who booked rooms in 5-star hotels:
Guest Name: Sarah Lee, Reservations Count: 1
```

Case 5: Query 2

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 5
Hotels with no part-time staff:
Hotel ID: 1, Hotel Name: Holiday Inn Richmond
```

Case 6: Query 3

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 6
High-paying full-time staff members:
Name: Emily White, Position: Manager, Salary: 65000, Hotel: Holiday Inn Richmond
```

Case 7: Query 4

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 7
Staff Members with Employment Type:
Staff ID: 3001, Name: Emily White, Employment Type: Full-Time
Staff ID: 3002, Name: Paul Green, Employment Type: Full-Time
Staff ID: 3003, Name: Anna Brown, Employment Type: Part-Time
Staff ID: 3004, Name: Tom Black, Employment Type: Full-Time
Staff ID: 3005, Name: Rachel Adams, Employment Type: Part-Time
```

Case 8: Query 5

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 8
Hotels with available rooms:
Hotel Name: Holiday Inn Richmond
Hotel Name: Marriott Downtown
```

Case 9: Exit

```
--- Hotel Management System ---
1. Create Tables
2. Drop Tables
3. Populate Tables
4. List Guests with 5-Star Hotel Bookings
5. List Hotels with No Part-Time Staff
6. List High-Paying Full-Time Staff
7. List All Staff Members with Employment Type
8. List Hotels with Available Rooms
9. Exit
Enter your choice: 9
Exiting the system...
```

Conclusion

We have gained a strong foundation in all facets of database design and implementation due to our work on this Hotel Database Management System.

We were able to transform disparate data sets into a practical and easily accessible database by applying the theories we studied about entity-relationship diagrams, relational schema design, functional relationships, normalization, and more.

On the technological side, we grew used to utilizing SQL and Oracle's services. This gave us experience creating tables, dropping tables, inserting data, and running queries. Additionally, we familiarized ourselves on how to connect to our back-end, Oracle database, and make it interact with our front-end interface through the UNIX and Java implementations.

Our abilities in software development, project management, and teamwork were also tested by this project. Overall, this project allowed us to use the knowledge and skills we had learned throughout this course, and apply it directly to our Hotel DBMS.