CPS510 - A10
Fall 2024

Jannis Saini - #501168140
Rose Pagano - #501165663
Nitya Malik - #501203793

Section 3
Instructor: Dr. Abdolreza Abhari
TA: Kiana Kheiri

11/26/2024

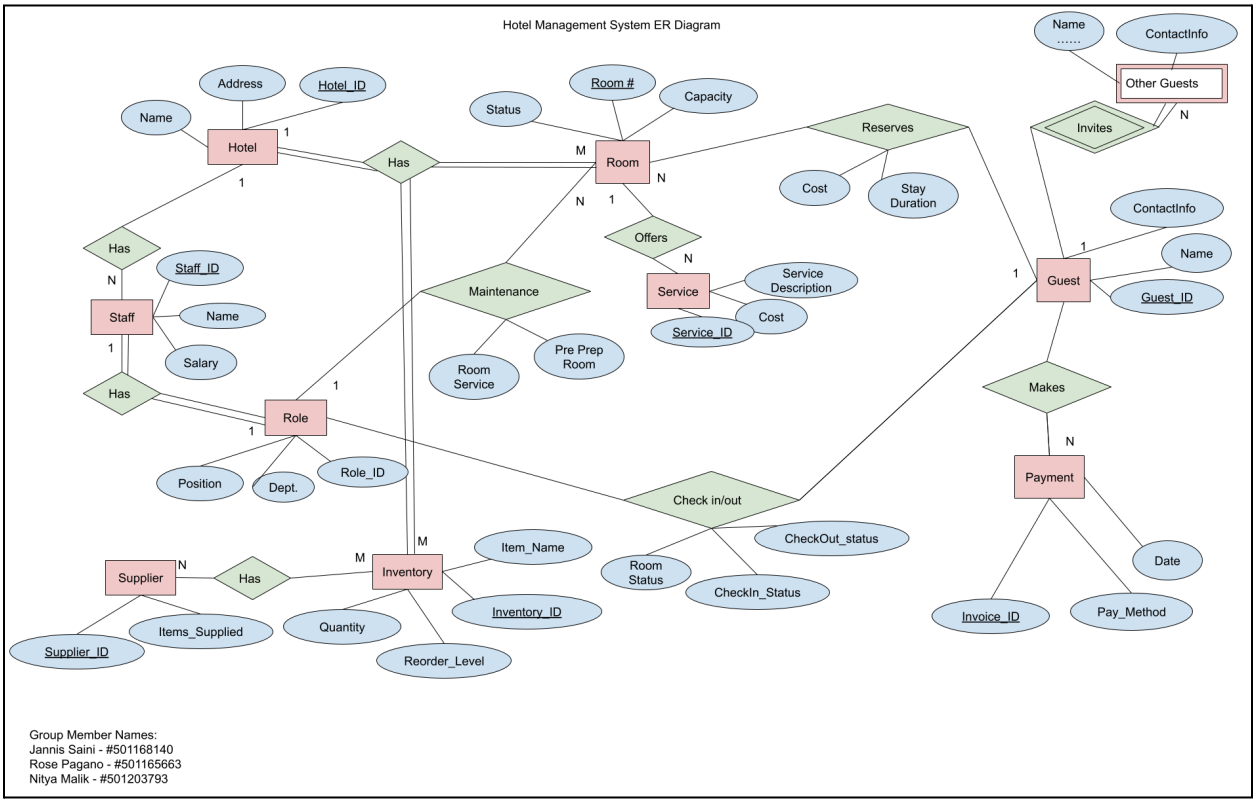
TABLE OF CONTENTS

INTRODUCTION

In the dynamic world of hospitality, efficient management is the cornerstone of success. Our innovative project aims to transform the way hotels operate by introducing a database system designed to streamline processes and help hotels have an easier time managing their operations. The following sections showcase the steps taken to create our database that concluded in creating a user friendly web application using Python as our language.

SCHEMA DESIGN

Below the initial schema design for our database can be found.



CREATE TABLE CODE

The final code to create all tables can be found under the *Appendix*.

INSERT STATEMENTS

All insert statements can be found under the *Appendix*.

SIMPLE QUERIES

Below some of the simple queries created for our database can be found.

<table>
<tr><th>Code</th><th>Output</th></tr>
<tr><td>SELECT DISTINCT Supplier_ID, Items_Supplied FROM Supplier;</td><td>

Supplier_ID	Items_Supplied
123	Towels
367	Toilet Paper
582	Duvets
281	Toiletries

</td></tr>
<tr><td>SELECT * FROM Supplier ORDER BY Order_No;</td><td>

Supplier_ID	Items_Supplied	Hotel_ID	Order_No
582	Duvets	17	1837
582	Duvets	26	1938
281	Toiletries	26	2478
123	Towels	26	2718
123	Towels	17	3926
367	Toilet Paper	17	4718
281	Toiletries	17	4892
367	Toilet Paper	26	7163

</td></tr>
<tr><td>SELECT * FROM Supplier WHERE Supplier_ID = 367;</td><td>

Supplier_ID	Items_Supplied	Hotel_ID	Order_No
367	Toilet Paper	26	7163
367	Toilet Paper	17	4718

</td></tr>
<tr><td>SELECT * FROM Staff GROUP BY Staff_Name;</td><td>

Staff_ID	Staff_Name	Salary	Hotel_ID
13	Dhara Patel	23	26
18	Manpreet Atwal	24	17
36	Priyanshi Patel	21	26
19	Swastik Saini	21	39
27	Takdeer Kaur	22	17
16	Veenita Gajju	24	5

</td></tr>
<tr><td>SELECT * FROM Hotel_Service WHERE service_id NOT IN (SELECT room_no FROM room WHERE r_capacity < 4);</td><td>

Service_ID	Service_Description	Ser_Cost
3	Drinks/Snacks in Room	10
5	Spa Services	100
8	Lunch Room Service	60

</td></tr>
<tr><td>SELECT * FROM Guest WHERE ContactInfo LIKE '(647)%';</td><td>

Guest_ID	G_Name	ContactInfo
2	Rose Pagano	(647) 649-8888
3	Nitya Malik	(647) 829-3721
6	Frank Ackton	(647) 244-1899
7	Emily Carter	(647) 555-6789

</td></tr>
<tr><td>SELECT * FROM Staff WHERE Salary < 23;</td><td>

Staff_ID	Staff_Name	Salary	Hotel_ID
27	Takdeer Kaur	22	17
19	Swastik Saini	21	39
36	Priyanshi Patel	21	26

</td></tr>
</table>

SELECT * FROM Guest WHERE ContactInfo = '(416) 123-4567';	

Guest_ID	G_Name	ContactInfo
1	Jannis Saini	(416) 123-4567

ADVANCED QUERIES

The images below showcases examples of some advanced queries created during this project.

Tables	Code
JOIN Staff + Hotel + Staff_Role	SELECT Staff_Name, R_Position FROM Staff s, Hotel h, Staff_Role sr WHERE h.H_Name = 'Moonlight' AND s.Hotel_ID = h.Hotel_ID AND s.Staff_ID = sr.Staff_ID;
Output: Staff_Name | R_Position Takdeer Kaur | Porter Manpreet Atwal | Kitchen staff	
JOIN Supplier + Hotel	SELECT Supplier_ID, Items_Supplied FROM Supplier s, Hotel h WHERE h.H_Name = 'Sunshine' AND s.Hotel_ID = h.Hotel_ID;
Output: Supplier_ID | Items_Supplied 123 | Towels 367 | Toilet Paper 582 | Duvets 281 | Toiletries	
JOIN Room + Reservation + Guest	SELECT Reservation_ID, G_Name FROM Room r, Reservation rID, Guest g WHERE r.Room_No = 5 AND rID.Room_No = r.Room_No AND rID.Reservation_ID = g.Guest_ID;
Output: Reservation_ID | G_Name 5 | Brock Hampton	
JOIN Hotel + Room	SELECT Room_No, R_Capacity FROM Room r, Hotel h

	WHERE h.H_name = 'DayNight' AND h.Hotel_ID = r.Hotel_ID;

Output:

Room_No	R_Capacity
3	4

VIEW + JOIN Hotel + Staff_Role + Staff	CREATE VIEW Hotel_Dept_Staff AS SELECT H_Name, Dept, Staff_Name FROM Hotel h, Staff_Role sr, Staff s WHERE sr.Dept = 'Security department' AND s.Staff_ID = sr.Staff_ID AND s.Hotel_ID = h.Hotel_ID;

Output: SELECT * FROM Hotel_Dept_Staff;

H_Name	Dept	Staff_Name
Sunshine	Security department	Priyanshi Patel
DayNight	Security department	Swastik Saini

VIEW	CREATE VIEW Hotel_Services AS SELECT H.H_Name, HS.Service_Description, HS.Ser_Cost FROM Hotel H, Hotel_Service HS WHERE HS.Service_ID = H.Hotel_ID;

Output: SELECT * from Hotel_Services;

H_Name	Service_Description	Ser_Cost
Luxury Hotel	Spa Services	100

VIEW	CREATE VIEW Room_Occupancy AS SELECT H.H_Name, R.Room_No, G.G_Name FROM Room R JOIN Reservation RES ON R.Room_No = RES.Room_No JOIN Guest G ON RES.Guest_ID = G.Guest_ID JOIN Hotel H ON H.Hotel_ID = R.Hotel_ID WHERE RES.Status = 'RESERVED';

Output: – counts number of rooms sorted by Hotel

```
SELECT H.H_Name, COUNT(R.Room_No) AS Total_Rooms
FROM Hotel H
JOIN Room R ON H.Hotel_ID = R.Hotel_ID
GROUP BY H.H_Name
ORDER BY Total_Rooms DESC;
```

H_Name	Total_Rooms
Sunshine	3
Moonlight	2
Luxury Hotel	2
DayNight	1

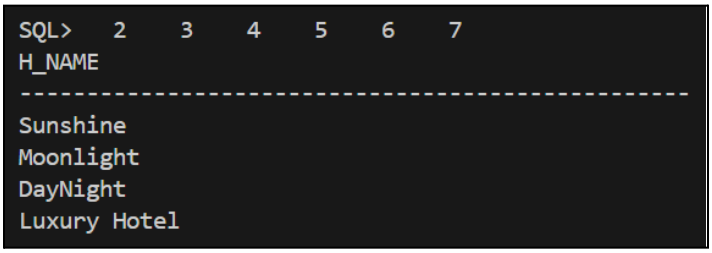
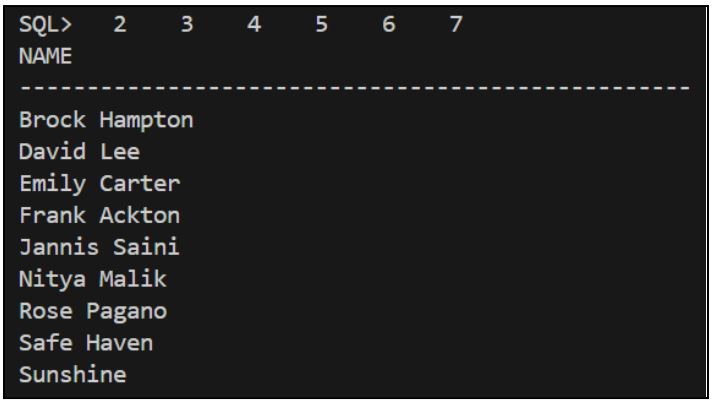
VIEW	CREATE VIEW Porter_Hotel_Assignments AS SELECT H.Hotel_ID, S.Staff_Name FROM Hotel H, Staff S, Staff_Role SR WHERE SR.R_Position = 'Porter' AND S.Staff_ID = SR.Staff_ID AND S.Hotel_ID = H.Hotel_ID;

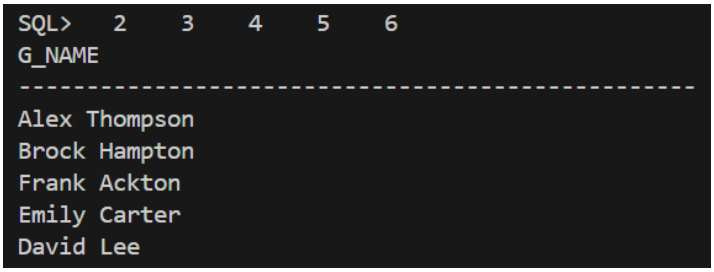
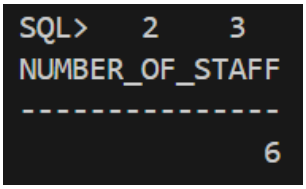
Output: SELECT * from Porter_Hotel_Assignments;

Hotel_ID	Staff_Name
17	Takdeer Kaur

UNIX OUTPUT

The unix code created during our lab can be found under *Appendix*. Below are some examples of advanced queries that were created under the unix environment.

Purpose	Output
Returns list of hotels that have at least one unoccupied room	
Return a list of hotels containing a letter ‘s’ UNION with guests containing letter ‘a’	

Display Guests that are in Rooms by themselves	
Counts the total staff the company employs	
Returns hotels where no. of rooms > 2	

FUNCTIONAL DEPENDENCIES

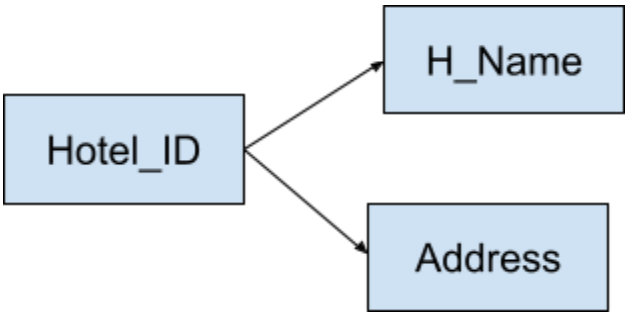
The table below showcases the functional dependencies the tables of our database follows.

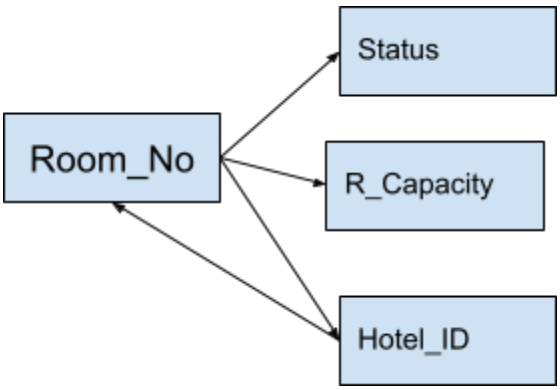
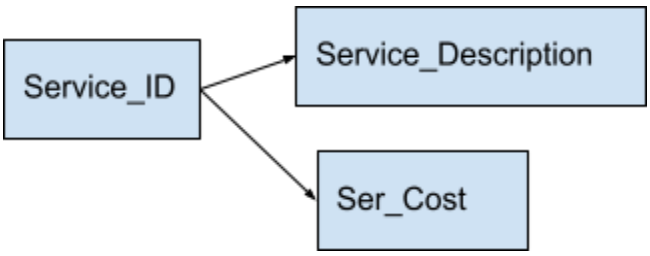
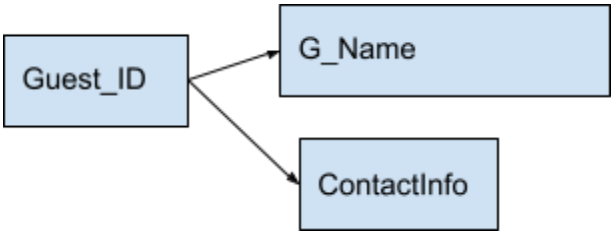
Table ()	Functional Dependency
HOTEL (Hotel_ID, H_Name, Address)	Hotel_ID → H_Name, Address
ROOM (Room_No, Status, R_Capacity, Hotel_ID)	Room_No → Status, R_Capacity, Hotel_ID Hotel_ID → Room_No
HOTEL_SERVICE (Service_ID, Service_Description, Ser_Cost)	Service_ID → Service_Description, Ser_Cost
GUEST (Guest_ID, G_Name, ContactInfo)	Guest_ID → G_Name, ContactInfo
STAFF (Staff_ID, Staff_Name, Salary, Hotel_ID)	Staff_ID → Staff_Name, Salary, Hotel_ID Staff_ID → Hotel_ID
STAFF_ROLE (Role_ID, R_Position, Dept, Staff_ID)	Role_ID → R_Position, Dept, Staff_ID Staff_ID → Role_ID
SUPPLIER (Order_No, Supplier_ID, Items_Supplied, Hotel_ID)	Order_No → Supplier_ID, Items_Supplied, Hotel_ID Hotel_ID → Order_No
INVENTORY (Inventory_ID, Hotel_ID, Item_Name, Quantity, Reorder_Level, Order_No	Inventory_ID → Hotel_ID, Item_Name, Quantity, Reorder_Level, Order_No

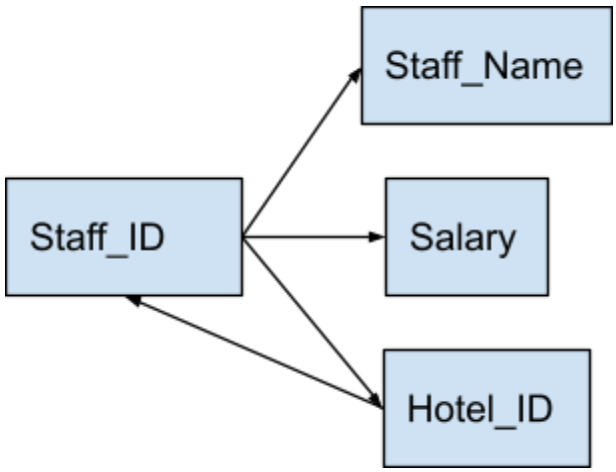
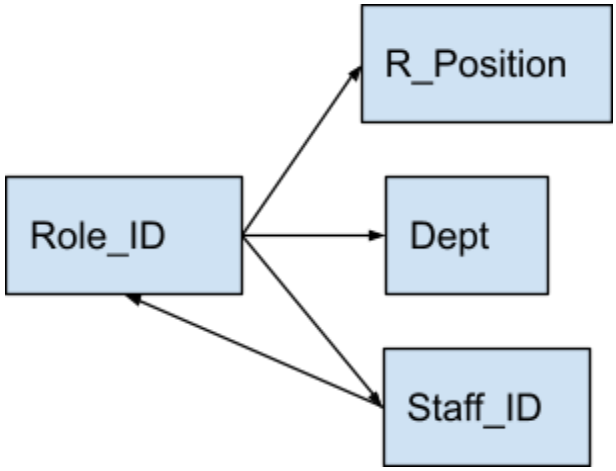
	Hotel_ID → Inventory_ID
OTHER_GUESTS (Guest_ID, OG_Name, ContactInfo)	Guest_ID → OG_Name, ContactInfo
PAYMENT (Invoice_ID, Payment_Date, Pay_Method, Guest_ID)	Payment → Payment_Date, Pay_Method, Guest_ID. Guest_ID → Payment
RESERVATION (Reservation_ID, Hotel_ID, Guest_ID, Room_No, Status, Stay_Cost, Start_Date, End_Date)	Reservation_ID → Hotel_ID, Guest_ID, Room_No, Status, Stay_Cost, Start_Date, End_Date. Hotel_ID → Reservation_ID Guest_ID → Reservation_ID Room_No → Reservation_ID
CHECK_IN_OUT (Reservation_ID, Status, Check_in_out)	Reservation_ID → Status, Check_in_out

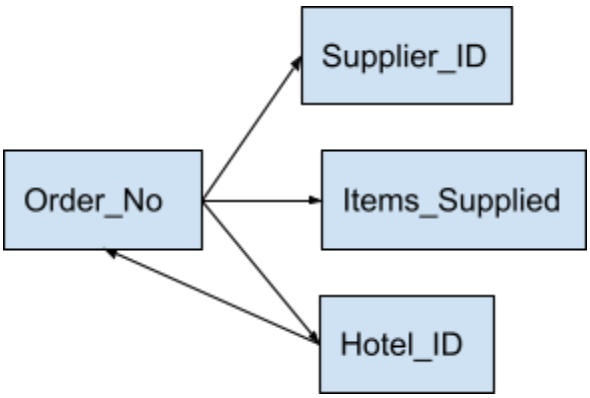
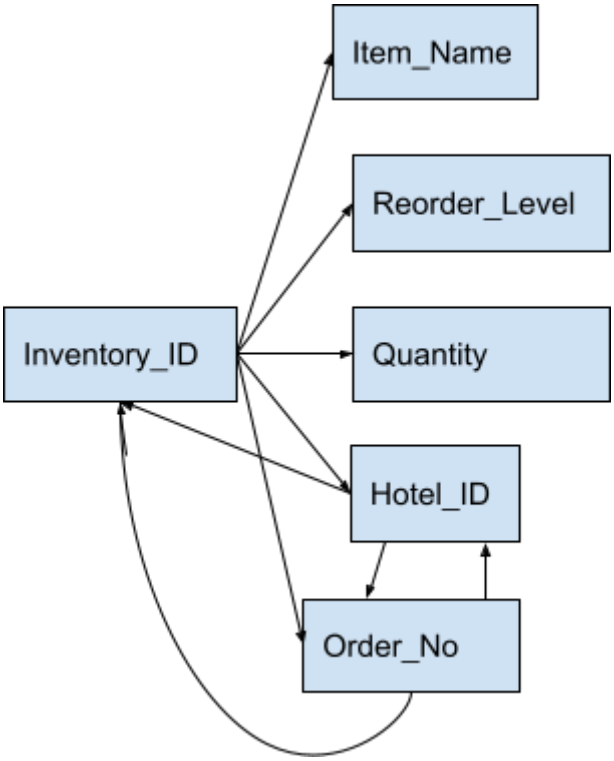
3NF/BCNF

After deliberating and breaking down our database tables the table below showcases which of our tables hold for 3NF and BCNF. Below the changes made to our tables to meet 3NF and BCNF can be found.

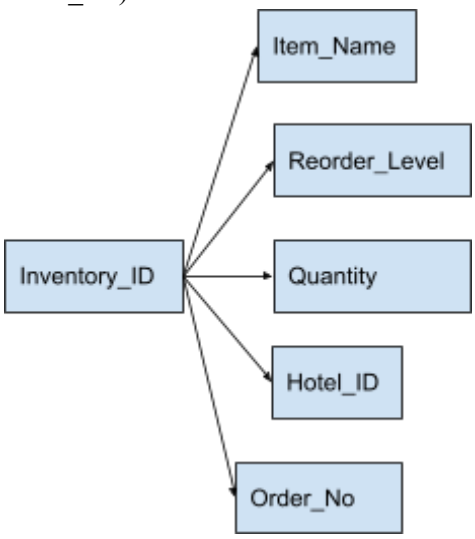
Table ()	3NF/BCNF Normal Form Met?
HOTEL	 3NF: Yes, because all non-key attributes are non-transitively dependent on the primary key. BCNF: Yes, every nontrivial, left irreducible FD has a candidate key as its determinant. Here, Hotel_ID is the candidate key that determines all attributes.

ROOM	 3NF: Yes, because all non-key attributes are non-transitively dependent on the primary key. BCNF: Yes, This table is BCNF because every nontrivial, left irreducible FD has a candidate key as its determinant. Here, Room_No obtains all other attributes, making Room_no the candidate key. Additionally, for the 2nd functional dependency, Hotel_ID is a candidate key so BCNF still holds.
HOTEL_SERVICE	 3NF: Yes, because all non-key attributes are non-transitively dependent on the primary key. BCNF: Yes, every nontrivial, left irreducible FD has a candidate key as its determinant. Here, Servicel_ID is the candidate key that determines all attributes.
GUEST	 3NF: Yes, because all non-key attributes are non-transitively dependent on the primary key. BCNF: Yes, every nontrivial, left irreducible FD has a candidate key as its determinant. Here, Guest_ID is the candidate key that determines all attributes.

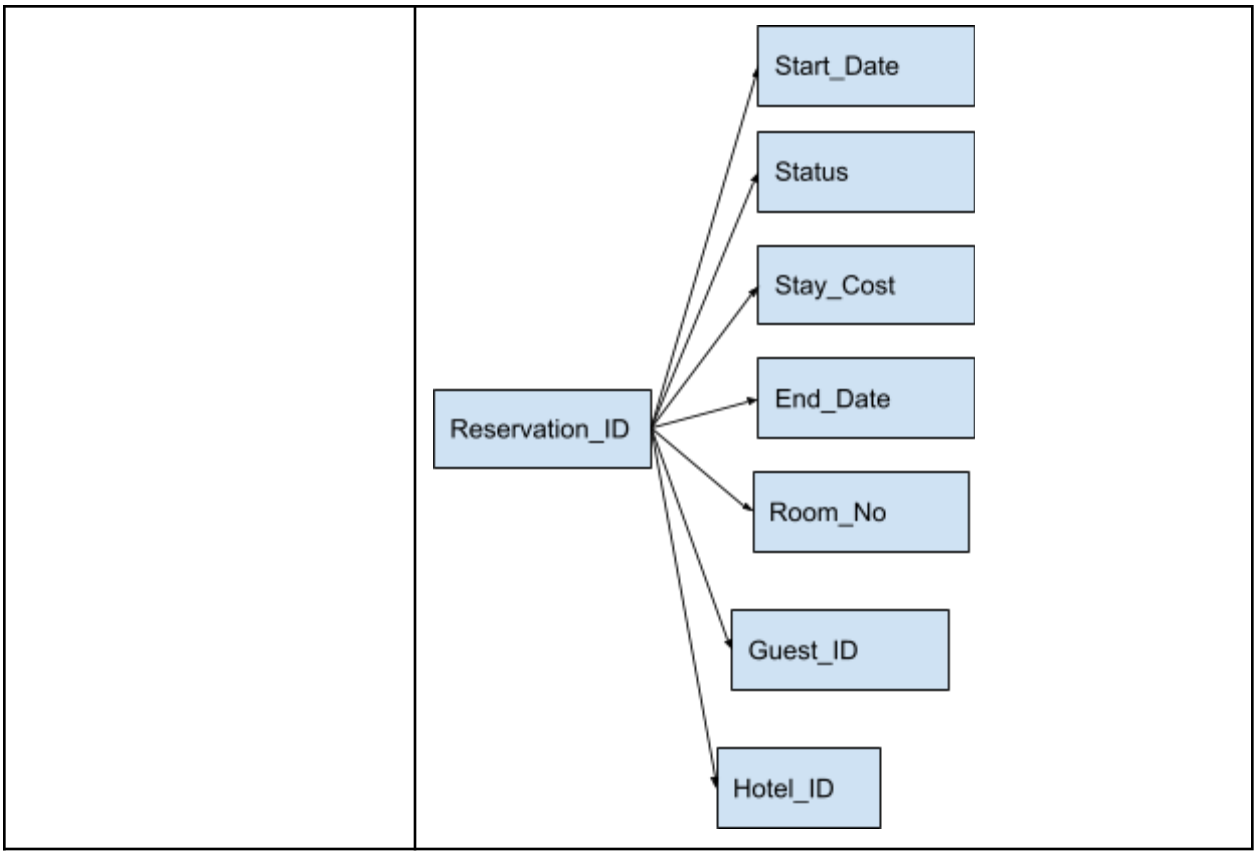
STAFF	 3NF: Yes, because all non-key attributes are non-transitively dependent on the primary key. BCNF: Yes, This table is BCNF because every nontrivial, left irreducible FD has a candidate key as its determinant. Here, Staff_ID obtains all other attributes, making Staff_ID the candidate key. Additionally, for the 2nd functional dependency, Hotel_ID is a candidate key so BCNF still holds.
STAFF_ROLE	 3NF: Yes, because all non-key attributes are non-transitively dependent on the primary key. BCNF: Yes, This table is BCNF because every nontrivial, left irreducible FD has a candidate key as its determinant. Here, Role_ID obtains all other attributes, making Role_ID the candidate key. Additionally, for the 2nd functional dependency, Staff_ID is a candidate key so BCNF still holds.

SUPPLIER	 3NF: Yes, because all non-key attributes are non-transitively dependent on the primary key. BCNF: Yes, This table is BCNF because every nontrivial, left irreducible FD has a candidate key as its determinant. Here, Order_No obtains all other attributes, making Order_No the candidate key. Additionally, for the 2nd functional dependency, Hotel_ID is a candidate key so BCNF still holds.
INVENTORY	 3NF: No, rearranged to 3NF below using an algorithm. Some non-key attributes are transitively dependent on each other as well as the primary key. BCNF: No, this table is not in BCNF because while the left irreducible FD has a candidate key as its determinant, not all keys are only dependent on the candidate key. Order_No and Hotel_ID break the relationship by

Decomposing Tables that do not meet BCNF criteria using Bernstein's Algorithm

Table ()	
INVENTORY	Inventory_ID → Hotel_ID, Item_Name, Quantity, Reorder_Level, Order_No Hotel_ID → Inventory_ID, Order_No Order_No→ Hotel_ID Step 2 Inventory_ID → Item_Name Inventory_ID → Quantity Inventory_ID → Reorder_Level Inventory_ID → Hotel_ID Order_No → Hotel_ID Inventory_ID→ Order_No Hotel_ID → Order_No Hotel_ID → Inventory_ID Step 3 3 candidate keys: - Inventory_ID - Order_No - Hotel_ID Step 4: We get 3 relations: R1(Inventory_ID, Hotel_ID, Item_Name, Quantity, Reorder_Level, Order_No) R2(Hotel_ID, Inventory_ID, Order_No) R3(Order_No, Hotel_ID) R2&R3 are subsets of R1, Therefore final schema is: R1(Inventory_ID, Hotel_ID, Item_Name, Quantity, Reorder_Level, Order_No) 

RESERVATION	Reservation_ID → Hotel_ID, Guest_ID, Room_No, Status, Stay_Cost, Start_Date, End_Date. Hotel_ID → Reservation_ID Guest_ID → Reservation_ID Room_No → Reservation_ID Hotel_ID→Room_No Room_No → Hotel_ID Step 2 Reservation_ID → Hotel_ID Reservation_ID → Guest_ID Reservation_ID → Room_No Reservation_ID → Status Reservation_ID → Stay_Cost Reservation_ID → Start_Date Reservation_ID → End_Date Hotel_ID → Reservation_ID Guest_ID → Reservation_ID Room_No → Reservation_ID Hotel_ID→ Room_No Room_No → Hotel_ID Step 3: 4 candidate keys: - Reservation_ID - Hotel_ID - Guest_ID - Room_No Step 4 We get 4 relations: R1(Reservation_ID, Hotel_ID, Guest_ID, Room_No, Status, Stay_Cost, Start_Date, End_Date) R2(Hotel_ID, Reservation_ID, Room_No) R3(Guest_ID, Reservation_ID) R4(Room_No, Reservation_ID, Hotel_ID) R2/R3/R4 are subsets of R1, Therefore final schema is: R1(Reservation_ID, Hotel_ID, Guest_ID, Room_No, Status, Stay_Cost, Start_Date, End_Date)



PYTHON UI DEMO

Below some snapshots of our web application using python can be found to showcase how we made a user friendly interface.

Introduction and GUI walkthrough.
During the course of the project, we decided to develop our GUI using python and utilizing its various libraries for executing the project with maximum efficiency. We used 2 programs namely : “main.py” and “app.py”. The programs utilize “tkinter” and “customTkinter” libraries for their respective GUI designs. Furthermore, we used “cx_oracle” and “oracledb” for database connectivity with our school’s database. While “main.py” serves as the comprehensive tool for administrators (us), “app.py” uses a streamlined solution for database setup and its operations.

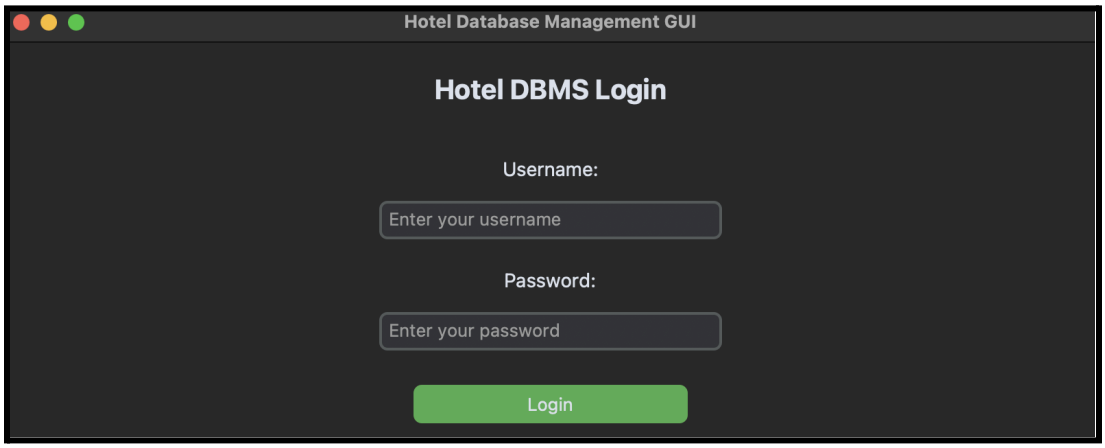
Installation
In order to run the script, we need to install Python (Ver. 3.12.4) and the libraries used in the making of our project. The libraries can be installed using the command “pip install ‘name of the library’”. Next step to make sure that the project is properly initialized upon running, is to install ‘Oracle Instant Client’ and make sure the directories of installation are the same as where the project is initially stored. After that, to ensure we are connected to the school’s database, we need to make sure we have an active VPN connection (Tunnelblick for macOS) and an active terminal connection with the “moon” servers.

Libraries
Installation of these libraries is necessary to run the GUI :

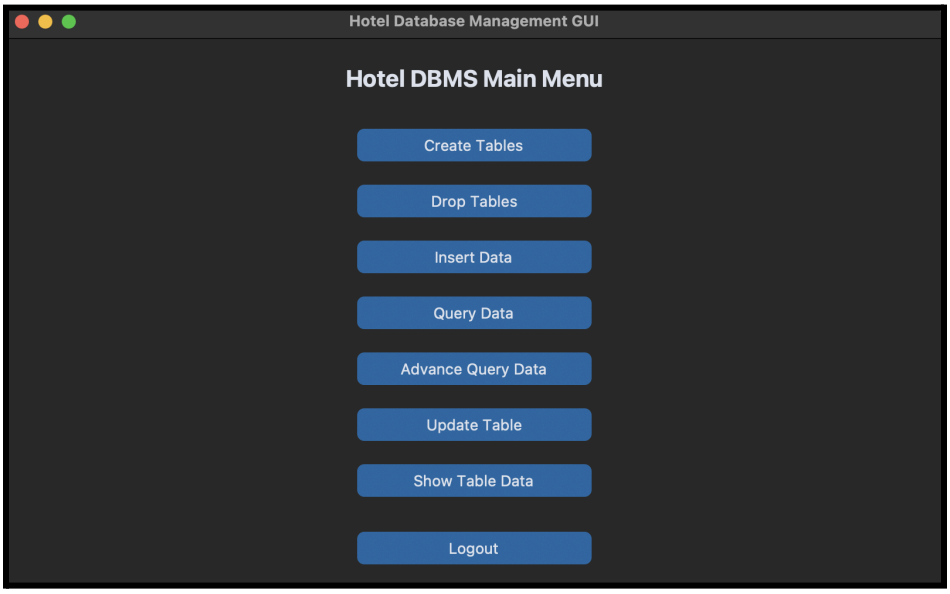
- Cx_oracle library
- Customtkinter
- Tkinter
- Pandas
- Oracledb

Executing scripts:

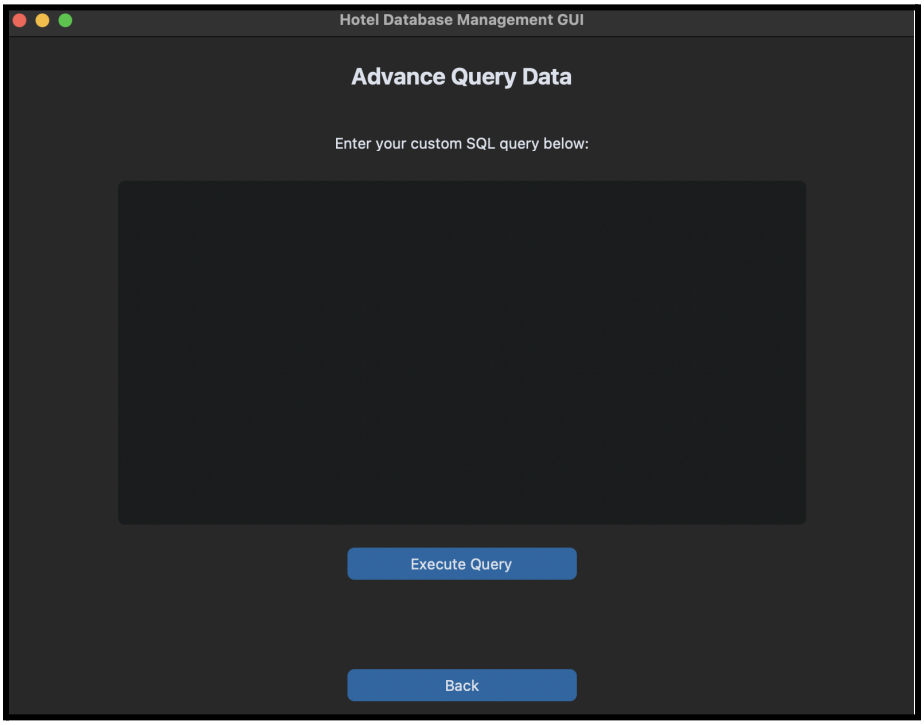
1. In order to interact with the GUI, we will execute “app.py” first to offer a base for “main.py” to execute the GUI. After running “main.py”, we are greeted with this screen which asks for login details. Users have to enter their username and password for their respective oracle logins.

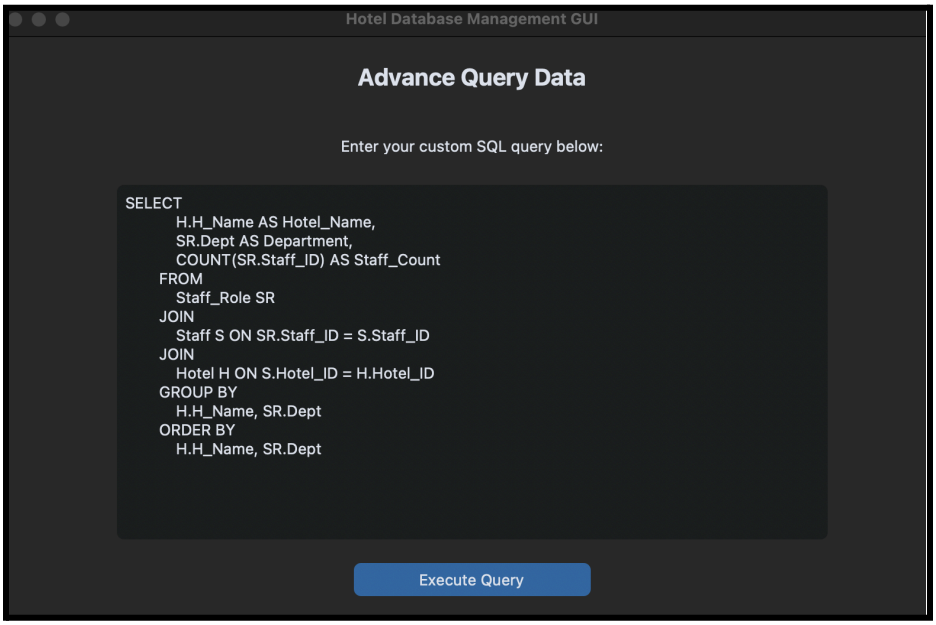


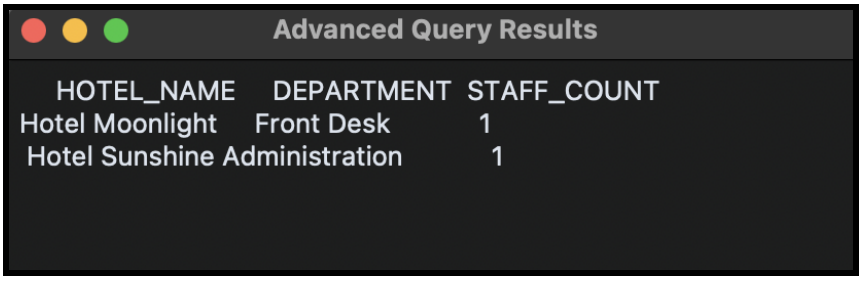
2. After entering your respective login details, we proceed to CRUD operations and additional database functionalities. The available options include creating and dropping tables, inserting data directly into the tables through the user interface using simple SQL statements, querying existing data, executing advanced queries, and displaying table data. We incorporated dropdown menus for a neater look. This intuitive layout ensures that users can navigate and perform database management tasks efficiently.



3. The rest of the GUI is designed to be intuitive and straightforward, allowing users to navigate through various database management operations. To showcase the advanced capabilities of our application, we will demonstrate the functionality by running a custom advanced query, highlighting the system.

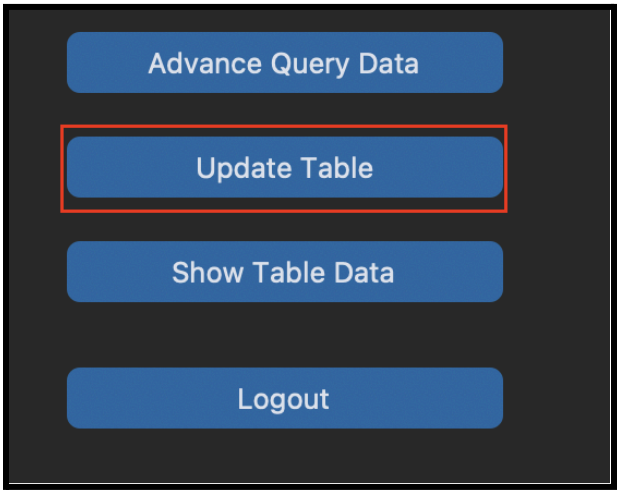


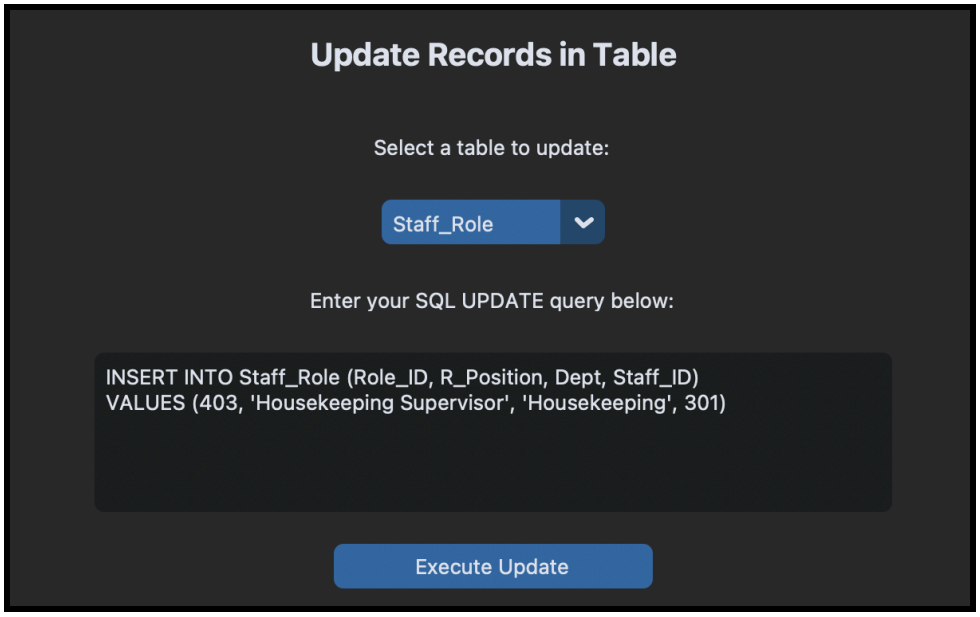




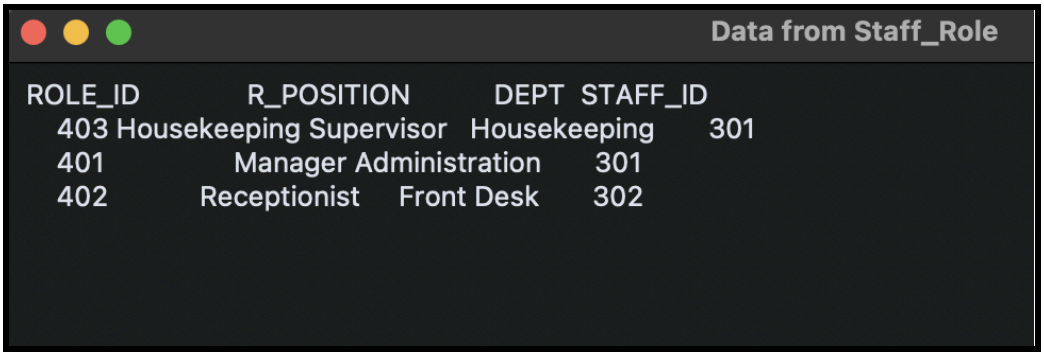
4. This query retrieves the number of staff members in each hotel department. While the system and the GUI is capable of handling large datasets, we have kept our tables moderately populated due to time constraints. Despite this, the functionality remains the same.

5. We will also be testing one of our functionalities which is “Update Table”. This allows us to modify existing tables in the database using simple SQL statements, Below are the results demonstrating this feature.





6. In order to verify that the table has been successfully updated, we can use the "Show table Data" functionality. This feature allows us to view the current state of the table and confirm the changes have been made.



7. One of the challenges we faced during the project was ensuring proper alignment of the resultant tabular data within the GUI. Although completely functional, the presentation could be further refined for improved clarity and better aesthetics. Given more time, we would have focused on making the layout even more polished.

RELATIONAL ALGEBRA

Below are the queries and the corresponding relational algebra.

Simple Queries	
Query	**Relational Algebra**
SELECT DISTINCT	$\Pi_{Supplier\_ID,\, Items\_Supplied}\,(\sigma(Supplier))$

Supplier_ID, Items_Supplied FROM Supplier;	
SELECT * FROM Supplier ORDER BY Order_No;	$\tau\ Order\_No\ \ \sigma\ (Supplier)$
SELECT * FROM Supplier WHERE Supplier_ID = 367;	$\sigma_{Supplier\_ID}\ =\ '367'\ (Supplier)$
SELECT * FROM Staff GROUP BY Staff_Name;	$\sigma\ (Staff)$
SELECT * FROM Hotel_Service WHERE service_id NOT IN (SELECT room_no FROM room WHERE r_capacity < 4);	$\sigma_{Service\_ID}\ \neq\ (\sigma_{room\_no}\ >\ 4\ (Room))\ (Hotel\_Service)$
SELECT * FROM Guest WHERE ContactInfo LIKE '(647)%';	$\sigma_{ContactInfo}\ LIKE\ '(647)\%'\ (Guest)$
SELECT * FROM Staff WHERE Salary < 23;	$\sigma_{Salary}\ <\ 23\ (Staff)$
SELECT * FROM Guest WHERE ContactInfo = '(416) 123-4567';	$\sigma_{ContactInfo}\ =\ '(416)\ 123\ -\ 4567'\ (Guest)$

Advanced Queries		
Table	**Query**	**Relational Algebra**

JOIN Staff + Hotel + Staff_Role	SELECT Staff_Name, R_Position FROM Staff s, Hotel h, Staff_Role sr WHERE h.H_Name = 'Moonlight' AND s.Hotel_ID = h.Hotel_ID AND s.Staff_ID = sr.Staff_ID;	$\Pi_{Staff\_Name,R\_Position}(\sigma_{H\_Name} =$ $'Moonlight'(Staff \bowtie_{Staff.Hotel\_ID} =$ $Hotel.Hotel\_ID$ $Hotel \bowtie_{Staff.Staff\_ID=Staff\_Role.Staff\_ID}$ $Staff\_Role))$
JOIN Supplier + Hotel	SELECT Supplier_ID, Items_Supplied FROM Supplier s, Hotel h WHERE h.H_Name = 'Sunshine' AND s.Hotel_ID = h.Hotel_ID;	$\Pi_{Supplier\_ID,Items\_Supplied}(\sigma_{H\_Name= 'Sunshine'}$ $(Supplier \bowtie_{Supplier.Hotel\_ID=Hotel.Hotel\_ID}$ $Hotel))$
JOIN Room + Reservation + Guest	SELECT Reservation_ID, G_Name FROM Room r, Reservation rID, Guest g WHERE r.Room_No = 5 AND rID.Room_No = r.Room_No AND rID.Reservation_ID = g.Guest_ID;	$\Pi_{Reservation\_ID,\ G\_Name}$ $(\sigma_{r.Room\_No = 5\ AND\ (rID.Room\_No = r.Room\_No)\ AND}$ $rID.Reservation\_ID = g.Guest\_ID)$ (Room ⋈ Guest ⋈ Reservation)
JOIN Hotel + Room	SELECT Room_No, R_Capacity FROM Room r, Hotel h WHERE h.H_name = 'DayNight' AND h.Hotel_ID = r.Hotel_ID;	$\Pi_{Room\_No,R\_Capacity}(\sigma_{H\_Name = 'DayNight'}$ $(Room \bowtie_{Room.Hotel\_ID=Hotel.Hotel\_ID} Hotel))$
VIEW + JOIN Hotel + Staff_Role + Staff	CREATE VIEW Hotel_Dept_Staff AS SELECT H_Name, Dept, Staff_Name FROM Hotel h, Staff_Role sr, Staff s WHERE sr.Dept = 'Security department' AND s.Staff_ID = sr.Staff_ID AND s.Hotel_ID = h.Hotel_ID;	$\Pi_{H\_Name,Dept,Staff\_Name}(\sigma_{Dept = 'Securitydepartment'}$ $(Hotel \bowtie_{Hotel.Hotel\_ID = Staff.Hotel\_ID}$ $Staff \bowtie_{Staff.Staff\_ID=Staff\_Role.Staff\_ID}$ $Staff\_Role))$
VIEW Hotel_services	CREATE VIEW Hotel_Services AS SELECT H.H_Name, HS.Service_Description, HS.Ser_Cost FROM Hotel H, Hotel_Service HS WHERE HS.Service_ID = H.Hotel_ID;	$\Pi_{H\_Name,Service\_Description,Ser\_Cost}$ ($Hotel$ $\bowtie_{Hotel.Hotel\_ID=Hotel\_Service.Service\_ID}$ $Hotel\_Service)$
Room_Occupany	SELECT H.H_Name, COUNT(R.Room_No) AS Total_Rooms FROM Hotel H JOIN Room R ON H.Hotel_ID = R.Hotel_ID GROUP BY H.H_Name ORDER BY Total_Rooms DESC;	$\tau\ COUNT(Room\_No)\ (\Pi_{H\_Name}(\sigma(Hotel)) \cap$ $\Pi_{H\_Name}(\sigma(Room))$

VIEW Porter_Hotel _Assignments	CREATE VIEW Porter_Hotel_Assignments AS SELECT H.Hotel_ID, S.Staff_Name FROM Hotel H, Staff S, Staff_Role SR WHERE SR.R_Position = 'Porter' AND S.Staff_ID = SR.Staff_ID AND S.Hotel_ID = H.Hotel_ID;	$\Pi_{Hotel\_ID,Staff\_Name}(\sigma_{R\_Position = 'Porter'}(Hotel \bowtie_{Hotel.Hotel\_ID=Staff.Hotel\_ID} Staff \bowtie_{Staff.Staff\_ID = Staff\_Role.Staff\_ID} Staff\_Role))$

CONCLUSION

The development of this Hotel Database Management System, from conceptualization to implementation, provided a comprehensive understanding of key principles and methodologies in database systems. The approach used during its creation provided a clear understanding of the steps needed to design and implement a fully functional database.The knowledge and technical skills acquired throughout this project are highly relevant to practical applications, demonstrating preparedness for future work in database design and management.

APPENDIX

Github link to our web application:
https://github.com/asce2619/Hotel-Database-Management-System/tree/main

Create Table Code

```
-- Table: Hotel
CREATE TABLE Hotel (
    Hotel_ID NUMBER PRIMARY KEY,
    H_Name VARCHAR2(50) NOT NULL,
    Address VARCHAR2(100) NOT NULL
);

-- Table: Room
CREATE TABLE Room (
    Room_No NUMBER PRIMARY KEY,
    Status VARCHAR2(20) DEFAULT 'UNOCCUPIED',
    R_Capacity NUMBER NOT NULL,
    Hotel_ID NUMBER NOT NULL,
    FOREIGN KEY (Hotel_ID) REFERENCES Hotel(Hotel_ID)
);

-- Table: Hotel_Service
CREATE TABLE Hotel_Service (
    Service_ID NUMBER PRIMARY KEY,
    Service_Description VARCHAR2(100) NOT NULL,
    Ser_Cost NUMBER NOT NULL
);

-- Table: Guest
CREATE TABLE Guest (
```

```
    Guest_ID NUMBER PRIMARY KEY,
    G_Name VARCHAR2(50) NOT NULL,
    ContactInfo VARCHAR2(50)
);

-- Table: Staff
CREATE TABLE Staff (
    Staff_ID NUMBER PRIMARY KEY,
    Staff_Name VARCHAR2(50) NOT NULL,
    Salary NUMBER NOT NULL,
    Hotel_ID NUMBER NOT NULL,
    FOREIGN KEY (Hotel_ID) REFERENCES Hotel(Hotel_ID)
);

-- Table: Staff_Role
CREATE TABLE Staff_Role (
    Role_ID NUMBER PRIMARY KEY,
    R_Position VARCHAR2(50) NOT NULL,
    Dept VARCHAR2(50) NOT NULL,
    Staff_ID NUMBER NOT NULL,
    FOREIGN KEY (Staff_ID) REFERENCES Staff(Staff_ID)
);

-- Table: Supplier
CREATE TABLE Supplier (
    Supplier_ID NUMBER NOT NULL,
    Items_Supplied VARCHAR2(20),
    Hotel_ID NUMBER NOT NULL,
    Order_No NUMBER NOT NULL PRIMARY KEY,
    CONSTRAINT Hotel_Supplied FOREIGN KEY (Hotel_ID) REFERENCES Hotel(Hotel_ID)
);

-- Decomposed Table: Inventory_Details
CREATE TABLE Inventory_Details (
    Inventory_ID NUMBER PRIMARY KEY,
    Hotel_ID NUMBER NOT NULL,
    Item_Name VARCHAR2(50) NOT NULL,
    Quantity NUMBER NOT NULL,
    Reorder_Level VARCHAR2(20) CHECK (Reorder_Level IN ('Satisfactory', 'Refill Required'))
);

-- Decomposed Table: Order_Hotel_Mapping
CREATE TABLE Order_Hotel_Mapping (
    Order_No NUMBER PRIMARY KEY,
    Hotel_ID NUMBER NOT NULL,
    FOREIGN KEY (Hotel_ID) REFERENCES Hotel(Hotel_ID)
);

-- Table: Other_Guests
CREATE TABLE Other_Guests (
```

```
    OG_Name VARCHAR2(50),
    ContactInfo VARCHAR2(50),
    Guest_ID NUMBER NOT NULL,
    FOREIGN KEY (Guest_ID) REFERENCES Guest(Guest_ID)
);

-- Table: Payment
CREATE TABLE Payment (
    Invoice_ID NUMBER UNIQUE,
    Payment_Date DATE NOT NULL,
    Pay_Method VARCHAR2(20),
    Guest_ID NUMBER NOT NULL,
    FOREIGN KEY (Guest_ID) REFERENCES Guest(Guest_ID)
);

-- Decomposed Table: Reservation_Details
CREATE TABLE Reservation_Details (
    Reservation_ID NUMBER PRIMARY KEY,
    Hotel_ID NUMBER NOT NULL,
    Guest_ID NUMBER NOT NULL,
    Room_No NUMBER,
    Status VARCHAR2(20) DEFAULT 'RESERVED',
    Stay_Cost NUMBER NOT NULL,
    Start_Date DATE NOT NULL,
    End_Date DATE NOT NULL,
    FOREIGN KEY (Guest_ID) REFERENCES Guest(Guest_ID),
    FOREIGN KEY (Hotel_ID) REFERENCES Hotel(Hotel_ID),
    FOREIGN KEY (Room_No) REFERENCES Room(Room_No)
);

-- Decomposed Table: Room_Hotel_Mapping
CREATE TABLE Room_Hotel_Mapping (
    Room_No NUMBER PRIMARY KEY,
    Hotel_ID NUMBER NOT NULL,
    FOREIGN KEY (Hotel_ID) REFERENCES Hotel(Hotel_ID)
);

-- Decomposed Table: Guest_Reservation_Mapping
CREATE TABLE Guest_Reservation_Mapping (
    Guest_ID NUMBER NOT NULL,
    Reservation_ID NUMBER NOT NULL,
    PRIMARY KEY (Guest_ID, Reservation_ID),
    FOREIGN KEY (Guest_ID) REFERENCES Guest(Guest_ID),
    FOREIGN KEY (Reservation_ID) REFERENCES Reservation_Details(Reservation_ID)
);

-- Table: Check_in_out
CREATE TABLE Check_in_out (
    Reservation_ID NUMBER PRIMARY KEY,
    Status VARCHAR2(20) DEFAULT 'OCCUPIED',
```

```
    Check_in_out VARCHAR2(20) DEFAULT 'Checked-in',
    FOREIGN KEY (Reservation_ID) REFERENCES Reservation_Details(Reservation_ID)
);
```

Insert Statements

Table	Insert Statement
Guest	INSERT INTO Guest VALUES (1, 'John Doe', '123-456-7890'); INSERT INTO Guest VALUES (2, 'Jane Smith', '987-654-3210');
Hotel	INSERT INTO Hotel VALUES (1, 'Hotel Sunshine', '123 Main Street'); INSERT INTO Hotel VALUES (2, 'Hotel Moonlight', '456 Oak Avenue');
Room	INSERT INTO Room VALUES (101, 'UNOCCUPIED', 2, 1); INSERT INTO Room VALUES (102, 'UNOCCUPIED', 4, 2;
Hotel_Service	INSERT INTO Hotel_Service VALUES (201, 'Room Cleaning', 20); INSERT INTO Hotel_Service VALUES (202, 'Laundry', 15);
Staff	INSERT INTO Staff VALUES (301, 'Alice Johnson', 5000, 1); INSERT INTO Staff VALUES (302, 'Bob Brown', 4000, 2);
Staff_Role	INSERT INTO Staff_Role VALUES (401, 'Manager', 'Administration', 301); INSERT INTO Staff_Role VALUES (402, 'Receptionist', 'Front Desk', 302);
Supplier	INSERT INTO Supplier VALUES (501, 'Bedsheets', 1, 1001); INSERT INTO Supplier VALUES (502, 'Shampoo', 2, 1002);
Inventory	INSERT INTO Inventory_Details VALUES (601, 1, 'Bedsheets', 50, 'Satisfactory'); INSERT INTO Inventory_Details VALUES (602, 2, 'Shampoo', 30, 'Refill Required');
Other_Guests	INSERT INTO Other_Guests VALUES ('Charlie White', '555-123-4567', 1); INSERT INTO Other_Guests VALUES ('Daisy Green', '555-987-6543', 2);
Payment	INSERT INTO Payment VALUES (1101, TO_DATE('2024-01-01', 'YYYY-MM-DD'), 'Cash', 1); INSERT INTO Payment VALUES (1102, TO_DATE('2024-01-02', 'YYYY-MM-DD'), 'Card', 2);
Reservation	INSERT INTO Reservation_Details VALUES (1201, 1, 1, 101, 'RESERVED', 200, TO_DATE('2024-02-01', 'YYYY-MM-DD'),

	TO_DATE('2024-02-03', 'YYYY-MM-DD')); INSERT INTO Reservation_Details VALUES (1202, 2, 2, 102, 'RESERVED', 300, TO_DATE('2024-03-01', 'YYYY-MM-DD'), TO_DATE('2024-03-04', 'YYYY-MM-DD'));
Check_in_out	INSERT INTO Check_in_out VALUES (1201, 'OCCUPIED', 'Checked-in') INSERT INTO Check_in_out VALUES (1202, 'RESERVED', 'Checked-out');

Unix Code

MAIN FILE CODE:

```
#!/bin/sh
# Hotel Management System - Unix Shell Script
# This script will automate tasks for the ER model assignment

# Function: Main Menu
MainMenu() {
  while [ "$CHOICE" != "E" ]
  do
    clear
    echo "============================================================="
    echo "| Hotel Management System - Database Tool                   |"
    echo "============================================================="
    echo "| Main Menu - Select Operation:                             |"
    echo "| <CTRL-Z Anytime to Enter Interactive CMD Prompt>           |"
    echo "-------------------------------------------------------------"
    echo " 1) View ER Model Description"
    echo " 2) Drop Tables"
    echo " 3) Create Tables"
    echo " 4) Populate Tables"
    echo " 5) Query Tables"
    echo " E) Exit"
    echo "-------------------------------------------------------------"
    echo "Choose: "
    read CHOICE

    case "$CHOICE" in
      1)
        echo "ER Model Description:"
        echo "Entities: Guest, Hotel, Service, Staff, Role of Staff"
        echo "Relationships: Guest uses Service, Staff assigned to Role, Staff assigned to Hotel"
        Pause
```

```
      ;;
    2)
      bash drop_tables.sh  # SQL script for dropping tables
      Pause
      ;;
    3)
      bash create_tables.sh  # SQL script for creating tables
      Pause
      ;;
    4)
      bash populate_tables.sh  # SQL script for inserting data
      Pause
      ;;
    5)
      bash queries.sh  # SQL script for querying tables
      Pause
      ;;
    E)
      echo "Exiting..."
      exit
      ;;
    *)
      echo "Invalid option. Please choose again."
      ;;
  esac
 done
}

# Function: Pause after an operation
Pause() {
 echo "Press [Enter] key to continue..."
 read dummy
}

# Main Program: Start the program loop
ProgramStart() {
 while [ 1 ]
 do
   MainMenu
 done
}

# Start the program
```

ProgramStart

SUB FILES:
The following sub files all follow the same format as below with their respective create, insert, drop, and query statement.

The following subfiles where created:

1. create_tables.sh
2. populate_tables.sh
3. drop_tables.sh
4. queries.sh

The format is as below:

```
#!/bin/sh
# Hotel Management System - Unix Shell Script
# Drop Table Script

#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64
"username/password@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca
)
(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

…..
….
….

exit;
EOF
```