Hotel Management System

-- Technical Report --

BUAN 6320.002 – Database Foundations for Business Analytics

Spring 2025

Group 7

Ankita Jadhav
Nghi Ngo
Mohib Siddiqui
Yujuan Xu
Judy Yang

Naveen Jindal School of Management, UT Dallas

Table of Contents

Introduction

The goal of this project is to design and implement a relational database system that effectively models and supports core operations of a medium-sized hotel management system. This includes managing customer reservations, payments, room assignments, employee information, and departmental structure. Utilizing a structured database design approach, our team developed a schema that ensures integrity, enforces business rules, and supports future scalability.

In the beginning, we conducted a thorough analysis of the entities typically involved in hotel operations, identifying their attributes, relationships and dependencies. We ensured that each entity followed the principles of normalization, eliminating redundancy and anomalies, and supporting efficient data retrieval. Based on our analysis, we created a set of SQL Data Definition Language (DDL) scripts to define the database schema, including tables, primary and foreign key constraints, sequences, and triggers.

Afterwards, we used Data Manipulation Language (DML) to populate the database with realistic data for customers, reservations, rooms, employees, departments, and payments. This populated data enables the execution of various SQL queries to demonstrate the functionality of the hotel management system, ranging from basic selects to advanced joins, aggregations, subqueries, views, and transaction controls using ROLLBACK.

In this report, we document our database design process, schema implementation, and the SQL operations used to manage and query the data, effectively forming the foundation of a functional and a well-normalized hotel management system.

Assumptions and Constraints

Newly added Assumptions and Constraints:

- We initially stated that ReceiptID and ResID are to be unique string characters. Instead of a mix of characters, we changed it to only be digits (still in string format) for sequence and triggers.
- Originally, we defined that DepartmentID value increments by 100 in a sequence. However, for format purpose, we changed to increment of 50 (e.g., one department is 100, another department is 150, etc).

Same Assumptions and Constraints from Project 1:

- Only guest rooms are considered. Staff room, gym room, etc are not considered for RoomType.
    - RoomTypes: Single (2 people), Double (4 people), and Family (6 people)

- Customer already has an account with the Hotel.
- Rooms are reserved after the full payment is paid.
- Payment is done through credit card or debit card. Only one payment method is allowed per transaction (reservation). This means that a customer cannot pay part by credit card and part by cash.
- The real-life scenario is based off of a medium scale hotel, with around 100 rooms and around 40 employees.
- 1 Customer can book multiple rooms in 1 reservation. A customer can make multiple separate reservations, booking multiple rooms.
- A customer can make more than 1 payment because they can make multiple separate reservations.
- Employment status is only Full-time or Part-time.
- The hotel has one phone number and the extension number will direct callers into the right department, format integer 1 2 3 4 5 etc (e.g., 1 [phone number] is directed to department 100, 2 [phone number] is directed to department 200, etc)

Project Step # 1

Entity and Attribute Description

- **Entity Name: ROOM**
  Description: Records of different types of guest room and the room's characteristics.
  Attributes
  - *RoomID (Primary Key)*: a unique 3-digit number that identifies each room.
  - *ResID (Foreign Key):* show if a specific room is associated with a reservation, can allow null as if that room is not booked. Referenced ResID in Reservation entity.
  - *RoomType:* the layout of a specific room, whether it is single-bed, double-bed, or family.
  - *RoomView:* where the room's balcony is facing and what the scenery is. This is a single valued attribute that only states what the main scenery is.
  - *MaxOccupancy (Transitive Dependency):* a maximum number of persons a room can accommodate, determined by RoomType.
  - *Floor:* which floor each room is located.

- **Entity Name: RESERVATION**
  Description: Information of each booking.
  Attributes
  - *ResID (Primary Key):* a unique 12-character string that identifies a reservation.
  - *AccountID (Foreign Key):* show the associated account that made the reservation. Cannot be null because a reservation needs to be associated with a customer. Referenced AccountID in Customer entity.
  - *RoomQuantity:* total number of room(s) a reservation has.
  - *PartySize:* total of guests the customer declares on the reservation.
  - *CheckIn:* beginning date of the stay, format MM/DD/YYYY
  - *CheckOut:* last date of the stay, format MM/DD/YYYY

- **Entity Name: CUSTOMER**
  Description: The information of the customer who books with their sign-in account and is the point of contact.
  Attributes
  - *AccountID (Primary Key)*: a unique 8-character string that is assigned for a customer account.
  - *ResID (Foreign Key):* records of the reservation(s) the customer has, referenced ResID in Reservation entity. Can allow null such that the customer does not have any reservation.
  - *ReceiptID (Foreign Key):* the proof of payment that the customer has for their reservation(s), referenced ReceiptID in Payment entity. Can allow null such that the customer did not make any payment.
  - *LastName:* customer's last name.
  - *FirstName:* customer's first name.
  - *Email:* customer's email associated with the account.
  - *PhoneNumber:* customer's contact number.

- **Entity Name: PAYMENT**
  Description: Record of payment for each reservation to access and troubleshoot any issue.

  Attributes
  - *ReceiptID (Primary Key):* a random 10-character string that generated after a reservation is fully paid.
  - *ResID (Foreign Key):* indicate the reservation that is linked to a receipt, referenced ResID in Reservation entity. Cannot be null because a reservation is required for there to be a payment.
  - *PaymentMethod:* show what type of card that makes the purchase and its last four-digits.
  - *TotalAmount:* the total amount paid in USD, including tax with 2 decimal places.
  - *DatePaid:* the date the purchase was made, format MM/DD/YYYY.
  - *Notes:* any additional notes or comments regarding customer's special requests or adjustments, stored in string.

- **Entity Name: EMPLOYEE**
  Description: Information of current employees.

  Attributes
  - *EmployeeID (Primary Key):* a 6-character string, begins with the first initial of their last name, the first initial of their first name, and four random numbers.
  - *LastName:* employee's last name
  - *FirstName:* employee's first name
  - *DepartmentID (Foreign Key):* indicate which department the employee belongs to. References DepartmentID in Department Entity. Cannot be null because an employee has to be part of a department.
  - *Title:* the official title of the employee, such as Front Desk, Room Service, Shift Manager, General Manager, etc.
  - *EmploymentType:* indicate if an employee is Full-time or Part-Time.
  - *PhoneNumber:* employee's personal phone number.
  - *Email:* employee's personal email.

- **Entity Name: DEPARTMENT**
  Description: The information of the department where each employee belongs to.

  Attributes
  - *DepartmentID (Primary Key)*: a unique 3-digit number that identifies the department.
  - *DepartmentName:* Name of the department.
  - *DepartmentHead (Foreign Key):* employee's ID who is the department's head. Can not be null because each department must have one and only one department head.
    References EmployeeID from Employee entity.
  - *Capacity:* maximum number of employees within a department
  - *DepartmentEmail:* email address of the department.
  - *PhoneExtension:* extension line of the department with the hotel's phone number being the main number.

Relationship and Cardinality Description

Relationship Reserves (Between RESERVATION and ROOM)
Description: “reserves” between RESERVATION and ROOM
Cardinality: 1:M between RESERVATION and ROOM
Business Rules:

- A RESERVATION may reserve more than one rooms
- A ROOM must be reserved only by one RESERVATION

Relationship Confirms (Between CUSTOMER and RESERVATION)
Description: “confirms” between CUSTOMER and RESERVATION
Cardinality: 1:M between CUSTOMER and RESERVATION
Business Rules:

- A CUSTOMER can confirm more than one RESERVATIONS
- A RESERVATION must be confirmed by one and only one CUSTOMER

Relationship Makes (Between CUSTOMER and PAYMENT)
Description: “makes” between CUSTOMER and PAYMENT
Cardinality: 1:M between CUSTOMER and PAYMENT
Business Rules:

- A CUSTOMER can make more than one PAYMENT
- A PAYMENT can only be made by one CUSTOMER

Relationship Employs (Between DEPARTMENT and EMPLOYEE)
Description: “employs” between DEPARTMENT and EMPLOYEE
Cardinality: 1:M between DEPARTMENT and EMPLOYEE
Business Rules:

- An EMPLOYEE must be employed by only one DEPARTMENT
- A DEPARTMENT can have one or many EMPLOYEES

Assumptions and Special Considerations (Step 1)

- Only guest rooms are considered. Staff room, gym room, etc are not considered for RoomType.
    - RoomTypes: Single (2 people), Double (4 people), and Family (6 people)
- Customer already has an account with the Hotel.
- Rooms are reserved after the full payment is paid.
- Payment is done through credit card or debit card. Only one payment method is allowed per transaction (reservation). This means that a customer cannot pay part by credit card and part by cash.
- The real-life scenario is based off of a medium scale hotel, with around 100 rooms and around 40 employees.

- 1 Customer can book multiple rooms in 1 reservation. A customer can make multiple separate reservations, booking multiple rooms.
- A customer can make more than 1 payment because they can make multiple separate reservations.
- Employment status is only Full-time or Part-time.
- DepartmentID value increments by 100 for different departments (e.g., one department is 100, another department is 200, etc).
- The hotel has one phone number and the extension number will direct callers into the right department, format integer 1 2 3 4 5 etc (e.g., 1 [phone number] is directed to department 100, 2 [phone number] is directed to department 200, etc)

Normalization Steps

1NF:

Objectives:

1. All Atributes contain atomic values.
2. Each record is unique and identified by a primary key.
3. No multivalued or repeating groups.

Room Table:

1. MaxOccupancy depends on RoomType, a Transitive Dependency.
2. All attributes in other tables are atomic, and each record is uniquely identified by their respective primary keys.

Room							
RoomID	ResID(fk)	RoomType	RoomView	MaxOccupancy	Floor		
Reservation							
ResID	AccountID(fk)	RoomQuantity	PartySize	CheckIn	CheckOut		
Customer							
AccountID	ResID(fk)	ReceiptID(fk)	LastName	FirstName	Email	PhoneNumber	
Payment							
ReceiptID	ResID(fk)	PaymentMethod	TotalAmount	DatePaid	Notes		
Employee							
EmployeeID	LastName	FirstName	DepartmentID(fk)	Title	EmploymentType	PhoneNumber	Email
Department							
DepartmentID	DepartmentName	DepartmentHead(fk)	Capacity	DepartmentEmail	PhoneExtension		

2NF:

Objective:

1. Meets the requirements of 1NF
2. All non-key attributes are fully functionally dependent on the primary key
3. No partial dependencies exist Identifying Partial Dependencies

No partial dependencies exist in Room, Reservation, Customer, Payment, Employee, and Department table, so the structure remains unchanged in 2NF.

There is a transitive dependency: MaxOccupancy depends on RoomType.

Room							
RoomID	ResID(fk)	RoomType	RoomView	MaxOccupancy	Floor		
Reservation							
ResID	AccountID(fk)	RoomQuantity	PartySize	CheckIn	CheckOut		
Customer							
AccountID	ResID(fk)	ReceiptID(fk)	LastName	FirstName	Email	PhoneNumber	
Payment							
ReceiptID	ResID(fk)	PaymentMethod	TotalAmount	DatePaid	Notes		
Employee							
EmployeeID	LastName	FirstName	DepartmentID(fk)	Title	EmploymentType	PhoneNumber	Email
Department							
DepartmentID	DepartmentName	DepartmentHead(fk)	Capacity	DepartmentEmail	PhoneExtension		

3NF:

Objectives:

1. Eliminate transitive dependencies.
2. Non-key attributes must depend only on the primary key.

Eliminating Transitive Dependency:

Transitive dependencies (like MaxOccupancy → RoomType → RoomID) have been resolved by creating the RoomType_Details table

Proper Referential Integrity:

FKs link logically related tables (e.g., DepartmentHead in DEPARTMENT references EmployeeID from EMPLOYEE)

Room							
RoomID	ResID(fk)	RoomType(fk)	RoomView	Floor			
RoomType							
RoomType	MaxOccupancy						
Reservation							
ResID	AccountID(fk)	RoomQuantity	PartySize	CheckIn	CheckOut		
Customer							
AccountID	ResID(fk)	ReceiptID(fk)	LastName	FirstName	Email	PhoneNumber	
Payment							
ReceiptID	ResID(fk)	PaymentMethod	TotalAmount	DatePaid	Notes		
Employee							
EmployeeID	LastName	FirstName	DepartmentID(fk)	Title	EmploymentType	PhoneNumber	Email
Department							
DepartmentID	DepartmentName	DepartmentHead(fk)	Capacity	DepartmentEmail	PhoneExtension		

Data Definition Language (DDL)

```
/* Project - DDL */

DROP SCHEMA IF EXISTS hotel_schema CASCADE;
CREATE SCHEMA hotel_schema;

SET search_path TO hotel_schema;

DROP TABLE IF EXISTS EMPLOYEE CASCADE;
DROP TABLE IF EXISTS DEPARTMENT CASCADE;
DROP TABLE IF EXISTS CUSTOMER CASCADE;
DROP TABLE IF EXISTS RESERVATION CASCADE;
DROP TABLE IF EXISTS ROOM CASCADE;
DROP TABLE IF EXISTS PAYMENT CASCADE;

-- ==================================
-- CREATE TABLES
-- ==================================

CREATE TABLE EMPLOYEE (
    EmployeeID CHAR(6) PRIMARY KEY,
    LastName VARCHAR(50) NOT NULL,
    FirstName VARCHAR(50) NOT NULL,
    DepartmentID CHAR(3),
    Title VARCHAR(50) NOT NULL,
    EmploymentType VARCHAR(10) NOT NULL,
    PhoneNumber VARCHAR(15),
    Email VARCHAR(100)
);

CREATE TABLE DEPARTMENT (
    DepartmentID CHAR(3) PRIMARY KEY,
    DepartmentName VARCHAR(100) NOT NULL,
    DepartmentHead CHAR(6),
    Capacity INT NOT NULL,
    DepartmentEmail VARCHAR(100),
    PhoneExtension INT,
    FOREIGN KEY (DepartmentHead) REFERENCES EMPLOYEE(EmployeeID)
);
```

```
ALTER TABLE EMPLOYEE
ADD FOREIGN KEY (DepartmentID) REFERENCES DEPARTMENT(DepartmentID);

CREATE TABLE CUSTOMER (
    AccountID CHAR(8) PRIMARY KEY,
    LastName VARCHAR(50) NOT NULL,
    FirstName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    PhoneNumber VARCHAR(15)
);

CREATE TABLE RESERVATION (
    ResID CHAR(12) PRIMARY KEY,
    AccountID CHAR(8) NOT NULL,
    RoomQuantity INT NOT NULL,
    PartySize INT NOT NULL,
    CheckIn DATE NOT NULL,
    CheckOut DATE NOT NULL,
    FOREIGN KEY (AccountID) REFERENCES CUSTOMER(AccountID)
);

CREATE TABLE ROOM (
    RoomID CHAR(3) PRIMARY KEY,
    ResID CHAR(12),
    RoomType VARCHAR(20) NOT NULL,
    RoomView VARCHAR(50),
    MaxOccupancy INT,
    Floor INT NOT NULL,
    FOREIGN KEY (ResID) REFERENCES RESERVATION(ResID)
);

CREATE TABLE PAYMENT (
    ReceiptID CHAR(10) PRIMARY KEY,
    ResID CHAR(12) NOT NULL,
    PaymentMethod VARCHAR(50) NOT NULL,
    TotalAmount DECIMAL(10,2) NOT NULL,
    DatePaid DATE NOT NULL,
    Notes TEXT,
    FOREIGN KEY (ResID) REFERENCES RESERVATION(ResID)
);

-- ==================================
-- SEQUENCES
-- ==================================
```

```
DROP SEQUENCE IF EXISTS res_id_seq CASCADE;
DROP SEQUENCE IF EXISTS receipt_id_seq CASCADE;
DROP SEQUENCE IF EXISTS department_id_seq CASCADE;

-- RESID: 12-digit padded string
CREATE SEQUENCE res_id_seq
  START WITH 100000000001
  INCREMENT BY 1;


-- RECEIPTID: 10-digit padded string
CREATE SEQUENCE receipt_id_seq
  START WITH 3000000001
  INCREMENT BY 1;


--DEPARTMENTID: 3-digit padded string
CREATE SEQUENCE department_id_seq
    START 100
    INCREMENT 50;


-- ===================================
-- TRIGGER FUNCTIONS
-- ===================================

-- RESERVATION
CREATE OR REPLACE FUNCTION assign_res_id()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.ResID IS NULL THEN
    NEW.ResID := LPAD(nextval('res_id_seq')::text, 12, '0');
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


-- PAYMENT
CREATE OR REPLACE FUNCTION assign_receipt_id()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.ReceiptID IS NULL THEN
    NEW.ReceiptID := LPAD(nextval('receipt_id_seq')::text, 10, '0');
  END IF;
```

```
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


--DEPARTMENT
CREATE OR REPLACE FUNCTION assign_department_id()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.DepartmentID IS NULL THEN
    NEW.DepartmentID := LPAD(nextval('department_id_seq')::text, 3,
'0');
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


--MaxOccupancy
CREATE OR REPLACE FUNCTION set_max_occupancy()
RETURNS TRIGGER AS $$
BEGIN
    IF LOWER(NEW.RoomType) = 'king' THEN
        NEW.MaxOccupancy := 2;
    ELSIF LOWER(NEW.RoomType) = 'double' THEN
        NEW.MaxOccupancy := 4;
    ELSIF LOWER(NEW.RoomType) = 'family' THEN
        NEW.MaxOccupancy := 6;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


-- ==================================
-- TRIGGERS
-- ==================================

-- RESERVATION
CREATE TRIGGER trg_assign_res_id
BEFORE INSERT ON RESERVATION
FOR EACH ROW
EXECUTE FUNCTION assign_res_id();
```

```
-- PAYMENT
CREATE TRIGGER trg_assign_receipt_id
BEFORE INSERT ON PAYMENT
FOR EACH ROW
EXECUTE FUNCTION assign_receipt_id();

--DEPARTMENT
CREATE TRIGGER trg_assign_department_id
BEFORE INSERT ON DEPARTMENT
FOR EACH ROW
EXECUTE FUNCTION assign_department_id();

--MaxOccupancy
CREATE TRIGGER insert_max_occupancy
BEFORE INSERT ON ROOM
FOR EACH ROW
EXECUTE FUNCTION set_max_occupancy();
```

Data Manipulation Language (DML) & Query Source Code

```
set search_path to hotel_schema;

/* Project - DML */

-- Insert into CUSTOMER table
INSERT INTO CUSTOMER (AccountID, LastName, FirstName, Email,
PhoneNumber)
VALUES
('AUEE9347', 'Anderson', 'Sam', 'samoldsame@gmail.com', '489-123-
4567'),
('F904WFHS', 'Nguyen', 'Lilia', 'lilnguyen031@yahoo.com', '289-987-
6543'),
('SRG9WKND', 'Williams', 'Erik', 'eriknotme@icloud.com', '378-246-
8012'),
('SVN93W02', 'Brown', 'Lucas', 'lucas.brownie@gmail.com', '912-135-
7910'),
('JROWG903', 'Davis', 'Patricia', 'patdavis98@hotmail.com', '426-864-
2098'),
('NAEGP923', 'Miller', 'Sabrina', 'sabrinaqueen@yahoo.com', '917-753-
1986'),
```

```
('VR84320F', 'Rodriguez', 'Linda', 'lindarod445@icloud.com', '714-420-
8521'),
('PVB843SK', 'Garcia', 'Jenna', 'jenna.garcia117@gmail.com', '512-319-
7410'),
('SP294ASP', 'Monroe', 'Maria', 'yummymm96@gmail.com', '853-648-
3109'),
('AQF39HSQ', 'Martinez', 'Christopher', 'chri.smart@icloud.com', '786-
537-0897'),
('PQ2WSG81', 'Toups', 'Angela', 'angela.lotus00@gmail.com', '479-026-
9785'),
('EOD024WS', 'Patel', 'Kevin', 'kevinpatel219@yahoo.com', '389-915-
6374');


-- Insert into RESERVATION table
INSERT INTO RESERVATION (AccountID, RoomQuantity, PartySize, CheckIn,
CheckOut)
VALUES
('AUEE9347', 1, 3, '2025-06-15', '2025-06-20'),
('F904WFHS', 1, 2, '2025-07-05', '2025-07-08'),
('SRG9WKND', 2, 4, '2025-10-01', '2025-10-03'),
('SVN93W02', 1, 2, '2025-06-05', '2025-06-07'),
('NAEGP923', 2, 4, '2025-08-20', '2025-08-25'),
('VR84320F', 1, 2, '2025-06-12', '2025-06-18'),
('SP294ASP', 2, 6, '2025-07-01', '2025-07-07'),
('AQF39HSQ', 1, 4, '2025-08-18', '2025-08-22'),
('PQ2WSG81', 1, 3, '2025-09-02', '2025-09-05'),
('EOD024WS', 1, 2, '2025-10-10', '2025-10-14');


-- Insert into ROOM table
INSERT INTO ROOM (RoomID,ResID, RoomType, RoomView, Floor)
VALUES
('101', NULL , 'Family', 'Ocean View', 1),
('102', '100000000001','Double', 'City View', 1),
('103', '100000000004','King', 'Ocean View', 1),
('104', NULL,'Double', 'Pool View', 1),
('105', '100000000002','King', 'City View', 1),
('201', NULL,'Family', 'Ocean View', 2),
('202', '100000000009','Double', 'City View', 2),
('203', '100000000003','King', 'Ocean View', 2),
('204', '100000000003','Double', 'Pool View', 2),
('205', NULL, 'King', 'City View', 2),
('301', NULL, 'Family', 'Ocean View', 3),
('302', '100000000005','Double', 'City View', 3),
```

```
('303', '100000000005','King', 'Ocean View', 3),
('304', '100000000006','Double', 'Pool View', 3),
('305', NULL, 'King', 'City View', 3),
('401', '100000000007','Family', 'Ocean View', 4),
('402', '100000000007','Double', 'City View', 4),
('403', NULL, 'King', 'Ocean View', 4),
('404', '100000000008','Double', 'Pool View', 4),
('405', '100000000010','King', 'City View', 4);


-- Insert into PAYMENT table
INSERT INTO PAYMENT (ResID, PaymentMethod, TotalAmount, DatePaid,
Notes)
VALUES
('100000000001', 'Credit Card', 250.00, '2025-04-01', 'Paid in full'),
('100000000002','Credit Card', 150.00, '2025-04-02', 'Paid in cash at
counter'),
('100000000003','Credit Card', 300.00, '2025-04-03', NULL),
('100000000004','Debit Card', 220.00, '2025-04-04', 'Paid via debit'),
('100000000005','Credit Card', 275.00, '2025-04-05', 'Online
transfer'),
('100000000006','Credit Card', 190.00, '2025-04-06', NULL),
('100000000007','Debit Card', 310.00, '2025-04-07', 'Discount
applied'),
('100000000008','Debit Card', 450.00, '2025-04-08', NULL),
('100000000009','Debit Card', 400.00, '2025-04-09', 'Late fee
included'),
('100000000010','Credit Card', 180.00, '2025-04-10', 'Paid same day');


-- Insert into DEPARTMENT table
INSERT INTO DEPARTMENT (DepartmentName, DepartmentHead, Capacity,
DepartmentEmail, PhoneExtension)
VALUES
('Front Desk', NULL, 10, 'frontdesk@hotel.com', 1),
('Housekeeping', NULL, 25, 'housekeeping@hotel.com', 2),
('Food and Beverage', NULL, 15, 'fnb@hotel.com', 3),
('Maintenance', NULL, 8, 'maintenance@hotel.com', 4),
('Security', NULL, 6, 'security@hotel.com', 5),
('Accounting', NULL, 5, 'accounting@hotel.com', 6),
('Human Resources', NULL, 4, 'hr@hotel.com', 7),
('Sales and Marketing', NULL, 6, 'sales@hotel.com', 8),
('IT Support', NULL, 3, 'it@hotel.com', 9),
('Events and Banquets', NULL, 5, 'events@hotel.com', 10);
```

```
-- Insert into EMPLOYEE table
INSERT INTO EMPLOYEE (EmployeeID, LastName, FirstName, DepartmentID,
Title, EmploymentType, PhoneNumber, Email)
VALUES
('SJ2743', 'Smith', 'John', 450, 'Sales Manager', 'Full-Time', '123-
456-7890', 'john.smith@email.com'),
('DJ1987', 'Doe', 'Jane', 400, 'HR Specialist', 'Full-Time', '987-654-
3210', 'jane.doe@email.com'),
('WA5629', 'Williams', 'Alex', 500, 'IT Technician', 'Part-Time',
'456-789-0123', 'alex.williams@email.com'),
('GM4832', 'Garcia', 'Maria', 150, 'Housekeeping Supervisor', 'Full-
Time', '789-654-3210', 'maria.garcia@email.com'),
('JM1305', 'Johnson', 'Michael', 100, 'Front Desk Receptionist',
'Full-Time', '321-987-6540', 'michael.johnson@email.com'),
('LS7461', 'Lee', 'Sophia', 200, 'Chef', 'Full-Time', '654-321-7890',
'sophia.lee@email.com'),
('BD9114', 'Brown', 'David', 300, 'Security Officer', 'Part-Time',
'987-321-4560', 'david.brown@email.com'),
('KE8046', 'Kim', 'Emily', 250, 'Maintenance Technician', 'Full-Time',
'321-654-9870', 'emily.kim@email.com'),
('AC3372', 'Anderson', 'Chris', 550, 'Marketing Coordinator', 'Full-
Time', '789-123-6540', 'chris.anderson@email.com'),
('MI6790', 'Martinez', 'Isabella', 350, 'Finance Analyst', 'Full-
Time', '654-987-3210', 'isabella.martinez@email.com');


-- Assign employees as DepartmentHeads
UPDATE DEPARTMENT SET DepartmentHead = 'JM1305' WHERE DepartmentName =
'Front Desk';
UPDATE DEPARTMENT SET DepartmentHead = 'GM4832' WHERE DepartmentName =
'Housekeeping';
UPDATE DEPARTMENT SET DepartmentHead = 'LS7461' WHERE DepartmentName =
'Food and Beverage';
UPDATE DEPARTMENT SET DepartmentHead = 'KE8046' WHERE DepartmentName =
'Maintenance';
UPDATE DEPARTMENT SET DepartmentHead = 'BD9114' WHERE DepartmentName =
'Security';
UPDATE DEPARTMENT SET DepartmentHead = 'MI6790' WHERE DepartmentName =
'Accounting';
UPDATE DEPARTMENT SET DepartmentHead = 'DJ1987' WHERE DepartmentName =
'Human Resources';
UPDATE DEPARTMENT SET DepartmentHead = 'SJ2743' WHERE DepartmentName =
'Sales and Marketing';
```

```
UPDATE DEPARTMENT SET DepartmentHead = 'WA5629' WHERE DepartmentName =
'IT Support';
UPDATE DEPARTMENT SET DepartmentHead = 'AC3372' WHERE DepartmentName =
'Events and Banquets';

/* 14 SQL Queries */

-- Q1: Select all columns and all rows from one table
SELECT * from employee;

-- Q2: Select five columns and all rows from one table
SELECT roomid, resid, roomtype, roomview, maxoccupancy
FROM room;

-- Q3: Select all columns from all rows from one view
CREATE VIEW Room_Description AS
SELECT r.roomview, r.floor, r.roomtype
FROM room r
WHERE r.maxoccupancy >= 4;

SELECT * from Room_Description;

DROP VIEW Room_Description;

-- Q4: Using a join on 2 tables, select all columns and all rows from
the tables without the use of a Cartesian product
SELECT *
FROM reservation re
LEFT OUTER JOIN room r
ON re.resid=r.resid;

-- Q5: Select and order data retrieved from one table
SELECT * from payment
ORDER BY totalamount;

-- Q6: Using a join on 3 tables, select 5 columns from the 3 tables.
  --Use syntax that would limit the output to 3 rows
SELECT p.receiptid, re.resid, re.accountid, r.roomid, r.roomtype
FROM payment p
LEFT OUTER JOIN reservation re
ON p.resid=re.resid
LEFT OUTER JOIN room r
ON re.resid=r.resid
LIMIT 3;
```

```
-- Q7: Select distinct rows using joins on 3 tables
-- Customers who have notes added to their reservation
SELECT DISTINCT *
FROM customer c
LEFT JOIN reservation re
ON c.accountid = re.accountid
LEFT JOIN payment p
ON re.resid = p.resid
WHERE p.notes IS NOT NULL AND p.notes != '';

-- Q8: Use GROUP BY and HAVING in a select statement using one or more
tables
--Filter the information of departments with fewer than 5 employees
--Display the department id, department name, number of employees
SELECT d.departmentid,
               d.departmentname,
               COUNT(e.employeeid)
FROM department d

LEFT OUTER JOIN employee e
ON d.departmentid=e.departmentid

GROUP BY d.departmentid, d.departmentname
HAVING COUNT(e.employeeid)<5;

-- Q9: Use IN clause to select data from one or more tables
-- Find rooms that have reservations
SELECT * FROM room r
WHERE r.resid IN
  (SELECT re.resid FROM customer c
  LEFT JOIN reservation re
  ON c.accountid = re.accountid);

-- Q10: Select length of one column from one table (use LENGTH
function)
SELECT LENGTH(firstname)
FROM employee;

-- Q11: Delete one record from one table.
  -- Use select statements to demonstrate the table contents before
and after the DELETE statement
  -- Make sure you use ROLLBACK afterwards so that the data will not
be physically removed
-- Begin the transaction
```

```
BEGIN;

SELECT * FROM customer ORDER BY accountid;

DELETE FROM customer
WHERE accountid = (
    SELECT accountid
    FROM customer
    WHERE accountid NOT IN (SELECT accountid FROM reservation)
    LIMIT 1
);

SELECT * FROM customer ORDER BY accountid;

ROLLBACK;

-- Q12: Update one record from one table.
-- Use select statements to demonstrate the table contents before and
after the UPDATE statement
-- Make sure you use ROLLBACK afterwards so that the data will not be
physically removed
-- Start a transaction and show contents before update
BEGIN;
SELECT * FROM room;

-- Update table
UPDATE room
SET maxoccupancy = 2
WHERE roomtype='Double';

-- Show contents after update
SELECT * FROM room;

-- Revert back to before update
ROLLBACK;
SELECT * FROM room;

-- Q13: Additional Advanced Queries
/* Identify customers based on total payments, and classify them by
spending amount.
Only include customer who has spent more than the average total
spending */
CREATE VIEW Advanced_Customer_Analytics AS
SELECT
    CONCAT(c.firstname, ' ', c.lastname) AS CustomerName,
```

```
    COUNT(DISTINCT r.resid) AS TotalReservations,
    MAX(r.checkin) AS LatestReservation,
    SUM(p.totalamount) AS TotalSpent,
    ROUND(SUM(p.totalamount) / NULLIF(COUNT(DISTINCT r.resid), 0), 2)
AS AvgPaymentPerReservation,
    CASE
        WHEN SUM(p.totalamount) >= 400 THEN 'Platinum'
        WHEN SUM(p.totalamount) >= 200 THEN 'Gold'
        WHEN SUM(p.totalamount) >= 100 THEN 'Silver'
        ELSE 'Bronze'
    END AS SpendingTier
FROM customer c
JOIN reservation r ON c.accountid = r.accountid
JOIN payment p ON r.resid = p.resid
WHERE c.accountid IN (
    SELECT cp.accountid
    FROM (
        SELECT
            c2.accountid,
            SUM(p2.totalamount) AS cust_total
        FROM customer c2
        JOIN reservation r2 ON c2.accountid = r2.accountid
        JOIN payment p2 ON r2.resid = p2.resid
        GROUP BY c2.accountid
    ) cp
    WHERE cp.cust_total > (
        SELECT AVG(total)
        FROM (
            SELECT
                SUM(p3.totalamount) AS total
            FROM customer c3
            JOIN reservation r3 ON c3.accountid = r3.accountid
            JOIN payment p3 ON r3.resid = p3.resid
            GROUP BY c3.accountid
        ) all_totals
    )
)

GROUP BY c.accountid, c.firstname, c.lastname;

SELECT * FROM Advanced_Customer_Analytics ORDER BY TotalSpent DESC
LIMIT 10;

DROP VIEW Advanced_Customer_Analytics;
```

```
-- Q14: Additional Advanced Queries
/* Identify long-stay customers who stayed more than 3 days.
Show account ID, full customer name, and length of stay. Sort results
by length of stay in descending order. */
SELECT c.accountid,
               CONCAT(c.firstname, ' ', c.lastname) AS "CustomerName",
               (r.checkout - r.checkin) AS "LengthOfStay"
FROM customer c

INNER JOIN reservation r
ON c.accountid = r.accountid

WHERE r.accountid IN
               (SELECT re.accountid FROM reservation re WHERE
(re.checkout - re.checkin) > 3)

ORDER BY (r.checkout - r.checkin) DESC;
```