Project Report: HotelSphere

An Integrated SQL Database Solution for Hotel Chain Management

1. Problem Statement

Modern hotel chains operate in a highly competitive and complex environment. Managing a multi-location hotel business involves overseeing a wide array of interconnected operations, including guest reservations, customer relationships, human resources, payroll, procurement, and inventory management. Many hotel chains suffer from data fragmentation, where critical information is stored in disparate, non-integrated systems. This leads to several significant Challenges:

- **Data Redundancy and Inconsistency**: The same information (e.g., customer details, supplier information) is often stored in multiple places, leading to inconsistencies and making updates cumbersome.

- **Operational Inefficiency**: Lack of a centralized system makes it difficult to get a holistic view of the business. Simple tasks, like tracking a guest's history across different hotel branches or managing inventory levels chain-wide, become complex and time-consuming.
- **Impaired Decision-Making:** Without a unified data source, management cannot perform
  effective business analytics. It is challenging to identify key trends like room occupancy rates, determine top-performing customers, or spot gaps in the supply chain.

- **Scalability Issues:** As the hotel chain grows, decentralized systems become increasingly
  difficult and expensive to manage, hindering scalability.

The HotelSphere project aims to solve these problems by designing and implementing a centralized, end-to-end, enterprise-grade SQL database solution. This system will integrate Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), and Supply Chain Management (SCM) modules into a single, cohesive, and normalized relational database. The goal is to ensure data integrity, eliminate redundancy, streamline operations, and provide powerful business intelligence capabilities to support strategic decision-making.

Entity Information & Assumptions

Entity Information by Module

The entities (tables) in your schema can be classified into the three specified modules as follows:

Customer Relationship Management (CRM) Module

This module focuses on the customer-facing aspects of the hotel.

- **branch**: Stores information about each hotel location.
- **branch_phone_numbers**: Stores the multiple phone numbers for each branch.
- **room_type**: Defines the different types of rooms available and their standard price.
- **room**: Represents an individual room within a branch.
- **room_amenities**: Lists the various amenities available in a specific room.
- **customer**: Contains details about the hotel's guests.
- **customer_phone_numbers**: Stores the multiple phone numbers for each customer.
- **loyalty_program**: Manages the loyalty status and points for customers.
- **booking**: Records the details of a room reservation made by a customer.
- **booking_special_requests**: Stores any special requests associated with a booking.
- **payment**: Tracks payments made for bookings.
- **feedback**: Stores customer feedback and ratings for their stay.

Enterprise Resource Planning (ERP) Module

This module deals with internal resources, primarily the employees and their roles/tasks.

- **department**: Lists the different departments within the hotel (e.g., Housekeeping, Front Desk).
- **employee**: A master table for all individuals employed by the hotel.
- **manager**: A specialization of an employee, indicating they have a managerial role.
- **staff**: A specialization of an employee, representing non-managerial staff.
- **general_manager**: A specialization of a manager, specifically for managing a branch.
- **warehouse_manager**: A specialization of a manager, specifically for managing a warehouse.
- **task**: Represents tasks assigned to staff members.

Supply Chain Management (SCM) Module

This module handles the procurement and management of goods and inventory.

- **supplier**: Stores information about companies that supply products to the hotel.
- **supplier_contact_persons**: Stores the multiple contact people for each supplier.

- **product**: Contains details about the items the hotel purchases (e.g., food items, toiletries).
- **purchase_order**: Represents an order placed with a supplier.
- **purchase_order_item**: The junction table detailing which products and quantities are in each purchase order.
- **warehouse**: Stores information about the hotel's storage facilities.
- **inventory**: Tracks the quantity of each product available in a specific warehouse.

Assumptions in the ER Model

The provided schema is based on several underlying business rules and assumptions:

1. **Organizational Structure**:
    - An employee can only belong to **one** department and be assigned to **one** branch at a time.
    - There is a clear employee hierarchy (specialization). An `employee` can be either a `manager` or a `staff`member, but not both. Furthermore, a `manager` can be specialized as a `general_manager` or a `warehouse_manager`.
    - Each `branch` is managed by exactly one `general_manager`.
    - Each `warehouse` is managed by exactly one `warehouse_manager`.
2. **Room and Booking**:
    - A room's price is determined by its **type** (e.g., 'Standard', 'Deluxe', 'Suite'), not by the individual room itself. This is a key assumption for normalization (3NF).
    - A single booking pertains to **one** customer and **one** room for a specified duration. Group bookings (one customer booking multiple rooms) would require separate entries in the `booking` table.
3. **Customer and Supplier Data**:
    - Customers, branches, and suppliers can have **multiple** phone numbers or contact persons, which necessitates separate tables in a normalized design (1NF).
    - A `customer` can have at most **one** `loyalty_program` account, creating a one-to-one relationship.
4. **Inventory and Procurement**:
    - A `purchase_order` is associated with exactly **one** supplier. To order from multiple suppliers, multiple purchase orders must be created.
    - A product's stock is tracked per `warehouse`. The same product can exist in multiple warehouses, and each instance will have its own `inventory` record.
5. **Derived Attributes**:
    - Certain values are intentionally not stored because they can be calculated on the fly. These include:

    - `duration_of_stay` (calculated from `check_out_date` - `check_in_date`).
    - `years_of_service` (calculated from `CURRENT_DATE` - `hire_date`).
    - `total_amount` of a purchase order (calculated by summing items from `purchase_order_item`).
    - `stock_value` in inventory (calculated from `quantity_on_hand` * `product.price`).

Relationships Information

This section describes the connections between the entities, based on the foreign keys and business logic implied by the normalized schema.

CRM Module Relationships

- **Branch - Room**: One-to-Many (1:M). A `branch` can have many `rooms`, but each `room` belongs to exactly one `branch`.
- **Room_Type - Room**: One-to-Many (1:M). A `room_type` (like 'Deluxe') can be applied to many `rooms`, but each `room` has only one `room_type`.
- **Customer - Booking**: One-to-Many (1:M). A `customer` can make multiple `bookings`, but each `booking` is associated with a single `customer`.
- **Room - Booking**: One-to-Many (1:M). A `room` can be part of many `bookings` over time, but a specific `booking` is for only one `room`.
- **Customer - Loyalty_Program**: One-to-One (1:1). Each `customer` can have at most one `loyalty_program`account, enforced by the `UNIQUE` constraint on `customer_id`.
- **Booking - Payment**: One-to-Many (1:M). A single `booking` can have multiple `payments` (e.g., a deposit and a final payment).
- **Booking - Feedback**: One-to-One (1:1). While the schema technically allows one booking to have multiple feedback entries, the business logic implies that a single stay (booking) would receive a single feedback entry.
- **Multivalued Attribute Relationships** (1:M):
    - **Branch - Branch_Phone_Numbers**: A `branch` can have many phone numbers.
    - **Room - Room_Amenities**: A `room` can have many amenities.
    - **Customer - Customer_Phone_Numbers**: A `customer` can have many phone numbers.
    - **Booking - Booking_Special_Requests**: A `booking` can have many special requests.

ERP Module Relationships

- **Department - Employee**: One-to-Many (1:M). A `department` has many `employees`, but an `employee`works in only one `department`.
- **Branch - Employee**: One-to-Many (1:M). A `branch` employs many `employees`, but an `employee` is assigned to a single `branch`.
- **Staff - Task**: One-to-Many (1:M). A `staff` member can be assigned many `tasks`, but each `task` is assigned to exactly one `staff` member.
- **General_Manager - Branch**: One-to-One (1:1). Each `branch` is managed by exactly one `general_manager`.
- **ISA (Specialization) Relationships** (1:1): These relationships represent hierarchical types.
    - **Employee** is a generalization of **Manager** and **Staff**.
    - **Manager** is a generalization of **General_Manager** and **Warehouse_Manager**.

SCM Module Relationships

- **Supplier - Purchase_Order**: One-to-Many (1:M). A `supplier` can receive many `purchase_orders`, but each `purchase_order` is from a single `supplier`.
- **Purchase_Order - Product**: Many-to-Many (M:N). A `purchase_order` can contain many `products`, and a `product` can appear on many `purchase_orders`. This relationship is implemented via the **`purchase_order_item`**junction table.
- **Warehouse - Inventory**: One-to-Many (1:M). A `warehouse` holds inventory for many different `products`.
- **Product - Inventory**: One-to-Many (1:M). A `product` can be stored in the `inventory` of multiple `warehouses`.
- **Warehouse_Manager - Warehouse**: One-to-One (1:1). Each `warehouse` is managed by exactly one `warehouse_manager`.
- **Multivalued Attribute Relationship** (1:M):
    - **Supplier - Supplier_Contact_Persons**: A `supplier` can have many contact persons.

ER Schema (Before & After Normalization)

Here is a textual representation of the database schema. The notation is as follows:

`TableName(PrimaryKey, Attribute1, {MultivaluedAttribute}, *ForeignKey*, (DerivedAttribute))` Primary Keys are the first attribute(s) listed in each entity. Attributes in `{}` are multivalued attributes that were improperly stored as delimited strings. Attributes in `()` are derived attributes that are not stored in the database.

ER Schema BEFORE Normalization

This version contains multivalued attributes and a transitive dependency, violating 1NF and 3NF.

CRM Module

- `branch(branch_id, location, {phone_numbers}, *manager_id*)`
- `room(room_id, room_number, type, price, {amenities}, *branch_id*)`
- `customer(customer_id, name, email, address, {phone_numbers})`
- `loyalty_program(program_id, tier_level, points, *customer_id*)`
- `booking(booking_id, check_in_date, check_out_date, {special_requests}, *customer_id*, *room_id*, (duration_of_stay))`
- `payment(payment_id, amount, payment_date, payment_method, *booking_id*)`
- `feedback(feedback_id, rating, comments, *booking_id*)`

ERP Module

- `department(department_id, name)`
- `employee(employee_id, name, hire_date, salary, *branch_id*, *department_id*, (years_of_service))`
- `manager(*employee_id*)`
- `staff(*employee_id*)`
- `general_manager(*employee_id*)`
- `warehouse_manager(*employee_id*)`
- `task(task_id, description, due_date, status, *assigned_to_staff_id*)`

SCM Module

- `supplier(supplier_id, name, contact_info, {contact_persons})`
- `product(product_id, name, price, {allergens})`

- purchase_order(order_id, order_date, status, *supplier_id*, (total_amount))
- purchase_order_item(*order_id*, *product_id*, quantity)
- warehouse(warehouse_id, location, *branch_id*, *manager_id*)
- inventory(inventory_id, quantity_on_hand, *warehouse_id*, *product_id*, (stock_value))

ER Schema AFTER Normalization

This version is in 3NF. Multivalued attributes have been moved to separate tables, and the transitive dependency has been resolved.

CRM Module

- branch(branch_id, location, *manager_id*)
- branch_phone_numbers(*branch_id*, phone_number)
- room_type(type_name, price)
- room(room_id, room_number, *branch_id*, *room_type_name*)
- room_amenities(*room_id*, amenity)
- customer(customer_id, name, email, address)
- customer_phone_numbers(*customer_id*, phone_number)
- loyalty_program(program_id, tier_level, points, *customer_id*)
- booking(booking_id, check_in_date, check_out_date, *customer_id*, *room_id*)
- booking_special_requests(*booking_id*, request)
- payment(payment_id, amount, payment_date, payment_method, *booking_id*)
- feedback(feedback_id, rating, comments, *booking_id*)

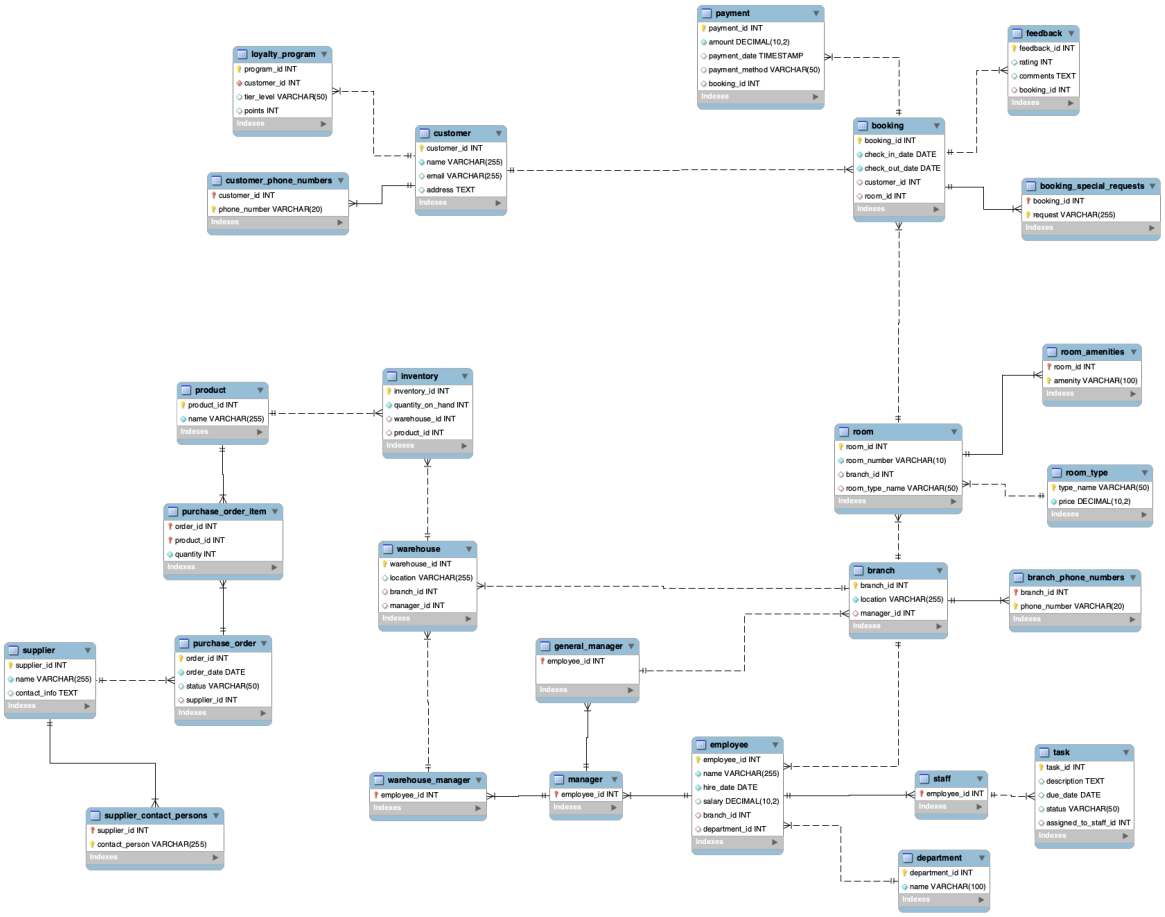
ERP Module

- department(department_id, name)
- employee(employee_id, name, hire_date, salary, *branch_id*, *department_id*)
- manager(*employee_id*)
- staff(*employee_id*)
- general_manager(*employee_id*)
- warehouse_manager(*employee_id*)

- task(task_id, description, due_date, status, *assigned_to_staff_id*)

SCM Module

- supplier(supplier_id, name, contact_info)
- supplier_contact_persons(*supplier_id*, contact_person)
- product(product_id, name, price)
- product_allergens(*product_id*, allergen)
- purchase_order(order_id, order_date, status, *supplier_id*)
- purchase_order_item(*order_id*, *product_id*, quantity)
- warehouse(warehouse_id, location, *branch_id*, *manager_id*)
- inventory(inventory_id, quantity_on_hand, *warehouse_id*, *product_id*)



EER Diagram

Creation of table (DDL Scripts)

```
-- ====================================================================

-- ========================= CRM MODULE ============================

-- ====================================================================


CREATE TABLE branch (

    branch_id INT PRIMARY KEY AUTO_INCREMENT,

    location VARCHAR(255) NOT NULL,

    manager_id INT,

    phone_numbers TEXT -- NON-NORMALIZED: Stored as a comma-separated string.

);


CREATE TABLE room (

    room_id INT PRIMARY KEY AUTO_INCREMENT,

    room_number VARCHAR(10) NOT NULL,

    type VARCHAR(50) NOT NULL,

    price DECIMAL(10, 2) NOT NULL,

    branch_id INT,

    amenities TEXT, -- NON-NORMALIZED: Stored as a comma-separated string (e.g., 'Wi-Fi,TV,AC').

    FOREIGN KEY (branch_id) REFERENCES branch(branch_id)

);


CREATE TABLE customer (

    customer_id INT PRIMARY KEY AUTO_INCREMENT,

    name VARCHAR(255) NOT NULL,

    email VARCHAR(255) UNIQUE,

    address TEXT,

    phone_numbers TEXT -- NON-NORMALIZED: Stored as a comma-separated string.

);
```

```
CREATE TABLE loyalty_program (
    program_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT UNIQUE NOT NULL,
    tier_level VARCHAR(50) DEFAULT 'Bronze',
    points INT DEFAULT 0,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON DELETE CASCADE
);

CREATE TABLE booking (
    booking_id INT PRIMARY KEY AUTO_INCREMENT,
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    customer_id INT,
    room_id INT,
    special_requests TEXT, -- NON-NORMALIZED: Stored as a delimited string.
    -- Derived Attribute: duration_of_stay would be calculated as (check_out_date - check_in_date) in a query.
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (room_id) REFERENCES room(room_id)
);

CREATE TABLE payment (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    amount DECIMAL(10, 2) NOT NULL,
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    payment_method VARCHAR(50),
    booking_id INT,
    FOREIGN KEY (booking_id) REFERENCES booking(booking_id)
);

CREATE TABLE feedback (
```

```
    feedback_id INT PRIMARY KEY AUTO_INCREMENT,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    comments TEXT,
    booking_id INT,
    FOREIGN KEY (booking_id) REFERENCES booking(booking_id)
);

-- ================================================================
-- ========================= ERP MODULE ===========================
-- ================================================================

CREATE TABLE department (
    department_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE employee (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    hire_date DATE NOT NULL,
    salary DECIMAL(10, 2),
    branch_id INT,
    department_id INT,
    -- Derived Attribute: years_of_service would be calculated as (CURRENT_DATE - hire_date) in a query.
    FOREIGN KEY (branch_id) REFERENCES branch(branch_id),
    FOREIGN KEY (department_id) REFERENCES department(department_id)
);

CREATE TABLE manager (
    employee_id INT PRIMARY KEY,
    FOREIGN KEY (employee_id) REFERENCES employee(employee_id) ON DELETE CASCADE
```

```
);

CREATE TABLE staff (
    employee_id INT PRIMARY KEY,
    FOREIGN KEY (employee_id) REFERENCES employee(employee_id) ON DELETE CASCADE
);

CREATE TABLE general_manager (
    employee_id INT PRIMARY KEY,
    FOREIGN KEY (employee_id) REFERENCES manager(employee_id) ON DELETE CASCADE
);

CREATE TABLE warehouse_manager (
    employee_id INT PRIMARY KEY,
    FOREIGN KEY (employee_id) REFERENCES manager(employee_id) ON DELETE CASCADE
);

CREATE TABLE task (
    task_id INT PRIMARY KEY AUTO_INCREMENT,
    description TEXT,
    due_date DATE,
    status VARCHAR(50) DEFAULT 'Pending',
    assigned_to_staff_id INT,
    FOREIGN KEY (assigned_to_staff_id) REFERENCES staff(employee_id)
);

-- ================================================================

-- ========================= SCM MODULE ============================

-- ================================================================

CREATE TABLE supplier (
```

```
    supplier_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    contact_info TEXT,
    contact_persons TEXT -- NON-NORMALIZED: Stored as a comma-separated string.
);

CREATE TABLE product (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    allergens TEXT -- NON-NORMALIZED: Stored as a comma-separated string.
);

CREATE TABLE purchase_order (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    order_date DATE NOT NULL,
    status VARCHAR(50) DEFAULT 'Placed',
    supplier_id INT,
    -- Derived Attribute: total_amount would be calculated by summing the (quantity * price) from purchase_order_items.
    FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id)
);

CREATE TABLE purchase_order_item (
    order_id INT,
    product_id INT,
    quantity INT NOT NULL,
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES purchase_order(order_id),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);
```

```
CREATE TABLE warehouse (
    warehouse_id INT PRIMARY KEY AUTO_INCREMENT,
    location VARCHAR(255),
    branch_id INT,
    manager_id INT,
    FOREIGN KEY (branch_id) REFERENCES branch(branch_id),
    FOREIGN KEY (manager_id) REFERENCES warehouse_manager(employee_id)
);

CREATE TABLE inventory (
    inventory_id INT PRIMARY KEY AUTO_INCREMENT,
    quantity_on_hand INT NOT NULL,
    warehouse_id INT,
    product_id INT,
    -- Derived Attribute: stock_value would be calculated as (quantity_on_hand * product.price) in a query.
    UNIQUE (warehouse_id, product_id),
    FOREIGN KEY (warehouse_id) REFERENCES warehouse(warehouse_id),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);

-- ================================================================
-- ===================== ADD FINAL CONSTRAINTS ====================
-- ================================================================

ALTER TABLE branch
ADD CONSTRAINT fk_branch_manager
FOREIGN KEY (manager_id) REFERENCES general_manager(employee_id);
```

```
-- == NORMALISATION IS DONE == --




-- ====================================================================

-- ========================= CRM MODULE ============================

-- ====================================================================


CREATE TABLE branch (

    branch_id INT PRIMARY KEY AUTO_INCREMENT,

    location VARCHAR(255) NOT NULL,

    manager_id INT -- This will be a foreign key to the general_manager table

);


-- NORMALIZED: Table for multivalued attribute 'phone_numbers'

CREATE TABLE branch_phone_numbers (

    branch_id INT,

    phone_number VARCHAR(20) NOT NULL,

    PRIMARY KEY (branch_id, phone_number),

    FOREIGN KEY (branch_id) REFERENCES branch(branch_id) ON DELETE CASCADE

);


-- NORMALIZED: New table to satisfy 3NF by removing transitive dependency from room table

CREATE TABLE room_type (

    type_name VARCHAR(50) PRIMARY KEY,

    price DECIMAL(10, 2) NOT NULL

);


CREATE TABLE room (

    room_id INT PRIMARY KEY AUTO_INCREMENT,
```

```
    room_number VARCHAR(10) NOT NULL,
    branch_id INT,
    room_type_name VARCHAR(50), -- Foreign key to the new room_type table
    FOREIGN KEY (branch_id) REFERENCES branch(branch_id),
    FOREIGN KEY (room_type_name) REFERENCES room_type(type_name)
);

-- NORMALIZED: Table for multivalued attribute 'amenities'
CREATE TABLE room_amenities (
    room_id INT,
    amenity VARCHAR(100) NOT NULL,
    PRIMARY KEY (room_id, amenity),
    FOREIGN KEY (room_id) REFERENCES room(room_id) ON DELETE CASCADE
);

CREATE TABLE customer (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE,
    address TEXT
);

-- NORMALIZED: Table for multivalued attribute 'phone_numbers'
CREATE TABLE customer_phone_numbers (
    customer_id INT,
    phone_number VARCHAR(20) NOT NULL,
    PRIMARY KEY (customer_id, phone_number),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON DELETE CASCADE
);

CREATE TABLE loyalty_program (
```

```
    program_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT UNIQUE NOT NULL,
    tier_level VARCHAR(50) DEFAULT 'Bronze',
    points INT DEFAULT 0,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON DELETE CASCADE
);

CREATE TABLE booking (
    booking_id INT PRIMARY KEY AUTO_INCREMENT,
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    customer_id INT,
    room_id INT,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (room_id) REFERENCES room(room_id)
);

-- NORMALIZED: Table for multivalued attribute 'special_requests'
CREATE TABLE booking_special_requests (
    booking_id INT,
    request VARCHAR(255) NOT NULL,
    PRIMARY KEY (booking_id, request),
    FOREIGN KEY (booking_id) REFERENCES booking(booking_id) ON DELETE CASCADE
);

CREATE TABLE payment (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    amount DECIMAL(10, 2) NOT NULL,
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    payment_method VARCHAR(50),
    booking_id INT,
```

```
    FOREIGN KEY (booking_id) REFERENCES booking(booking_id)
);

CREATE TABLE feedback (
    feedback_id INT PRIMARY KEY AUTO_INCREMENT,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    comments TEXT,
    booking_id INT,
    FOREIGN KEY (booking_id) REFERENCES booking(booking_id)
);

-- ================================================================
-- ========================= ERP MODULE ===========================
-- ================================================================

CREATE TABLE department (
    department_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE employee (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    hire_date DATE NOT NULL,
    salary DECIMAL(10, 2),
    branch_id INT,
    department_id INT,
    FOREIGN KEY (branch_id) REFERENCES branch(branch_id),
    FOREIGN KEY (department_id) REFERENCES department(department_id)
);
```

```
CREATE TABLE manager (
    employee_id INT PRIMARY KEY,
    FOREIGN KEY (employee_id) REFERENCES employee(employee_id) ON DELETE CASCADE
);

CREATE TABLE staff (
    employee_id INT PRIMARY KEY,
    FOREIGN KEY (employee_id) REFERENCES employee(employee_id) ON DELETE CASCADE
);

CREATE TABLE general_manager (
    employee_id INT PRIMARY KEY,
    FOREIGN KEY (employee_id) REFERENCES manager(employee_id) ON DELETE CASCADE
);

CREATE TABLE warehouse_manager (
    employee_id INT PRIMARY KEY,
    FOREIGN KEY (employee_id) REFERENCES manager(employee_id) ON DELETE CASCADE
);

CREATE TABLE task (
    task_id INT PRIMARY KEY AUTO_INCREMENT,
    description TEXT,
    due_date DATE,
    status VARCHAR(50) DEFAULT 'Pending',
    assigned_to_staff_id INT,
    FOREIGN KEY (assigned_to_staff_id) REFERENCES staff(employee_id)
);

-- ================================================================
-- ======================== SCM MODULE ============================
```

```
-- ================================================================


CREATE TABLE supplier (
    supplier_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    contact_info TEXT
);


-- NORMALIZED: Table for multivalued attribute 'contact_persons'
CREATE TABLE supplier_contact_persons (
    supplier_id INT,
    contact_person VARCHAR(255) NOT NULL,
    PRIMARY KEY (supplier_id, contact_person),
    FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE CASCADE
);


CREATE TABLE product (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL
);


-- NORMALIZED: Table for multivalued attribute 'allergens'
CREATE TABLE product_allergens (
    product_id INT,
    allergen VARCHAR(100) NOT NULL,
    PRIMARY KEY (product_id, allergen),
    FOREIGN KEY (product_id) REFERENCES product(product_id) ON DELETE CASCADE
);


CREATE TABLE purchase_order (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
```

```
    order_date DATE NOT NULL,
    status VARCHAR(50) DEFAULT 'Placed',
    supplier_id INT,
    FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id)
);

CREATE TABLE purchase_order_item (
    order_id INT,
    product_id INT,
    quantity INT NOT NULL,
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES purchase_order(order_id),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);

CREATE TABLE warehouse (
    warehouse_id INT PRIMARY KEY AUTO_INCREMENT,
    location VARCHAR(255),
    branch_id INT,
    manager_id INT,
    FOREIGN KEY (branch_id) REFERENCES branch(branch_id),
    FOREIGN KEY (manager_id) REFERENCES warehouse_manager(employee_id)
);

CREATE TABLE inventory (
    inventory_id INT PRIMARY KEY AUTO_INCREMENT,
    quantity_on_hand INT NOT NULL,
    warehouse_id INT,
    product_id INT,
    UNIQUE (warehouse_id, product_id),
    FOREIGN KEY (warehouse_id) REFERENCES warehouse(warehouse_id),
```

```
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);

-- ================================================================
-- ===================== ADDING FINAL CONSTRAINTS ====================
-- ================================================================

ALTER TABLE branch
ADD CONSTRAINT fk_branch_manager
FOREIGN KEY (manager_id) REFERENCES general_manager(employee_id);
```

Insertion of values (DML Scripts)

```
-- ================================================================
-- ===================== POPULATE DATABASE ========================
-- ================================================================

-- The insertion process is ordered to handle dependencies (foreign keys).
-- We will start with tables that do not have foreign keys, or whose foreign keys can be temporarily NULL.

-- ================================================================
-- ================== ERP MODULE DATA INSERTION ===================
-- ================================================================
-- We begin with the ERP module to establish branches, departments, and employees,
-- which are referenced by other modules.

-- 1. Insert Departments
INSERT INTO `department` (`name`) VALUES
('Management'),
('Front Desk'),
('Housekeeping'),
```

('Maintenance'),

('Kitchen');

-- 2. Insert Branches (with manager_id as NULL initially to avoid circular dependency)

-- We will update the manager_id after creating the managers.

INSERT INTO `branch` (`location`, `manager_id`) VALUES

('Hanamkonda Central', NULL),

('Hyderabad Hitech', NULL),

('Warangal Downtown', NULL);

-- 3. Insert Employees

-- We need to create employees before they can be assigned roles like manager or staff.

-- Employee IDs 1, 2, 3 will be managers. IDs 4, 5, 6, 7 will be staff.

INSERT INTO `employee` (`name`, `hire_date`, `salary`, `branch_id`, `department_id`) VALUES

('Alice Johnson', '2020-01-15', 90000.00, 1, 1), -- General Manager for Branch 1

('Bob Williams', '2021-03-22', 85000.00, 2, 1), -- General Manager for Branch 2

('Charlie Brown', '2019-07-10', 75000.00, 1, 1), -- Warehouse Manager for Branch 1

('David Smith', '2022-08-01', 45000.00, 1, 2), -- Staff, Front Desk

('Eve Davis', '2023-01-20', 48000.00, 2, 2), -- Staff, Front Desk

('Frank Miller', '2021-11-05', 38000.00, 1, 3), -- Staff, Housekeeping

('Grace Wilson', '2022-05-12', 40000.00, 1, 5); -- Staff, Kitchen

-- 4. Specialize Employees into Roles (Manager, Staff)

-- A. Insert into manager table

INSERT INTO `manager` (`employee_id`) VALUES

(1), (2), (3);

-- B. Insert into staff table

INSERT INTO `staff` (`employee_id`) VALUES

(4), (5), (6), (7);

```
-- 5. Further Specialize Managers into General/Warehouse Managers

-- A. Insert into general_manager

INSERT INTO `general_manager` (`employee_id`) VALUES

(1), (2);


-- B. Insert into warehouse_manager

INSERT INTO `warehouse_manager` (`employee_id`) VALUES

(3);


-- 6. UPDATE Branches with Manager IDs

-- Now that the general managers exist, we can link them to their branches.

UPDATE `branch` SET `manager_id` = 1 WHERE `branch_id` = 1;

UPDATE `branch` SET `manager_id` = 2 WHERE `branch_id` = 2;


-- 7. Insert Tasks for Staff

INSERT INTO `task` (`description`, `due_date`, `status`, `assigned_to_staff_id`) VALUES

('Prepare welcome kits for VIP guests', '2025-10-10', 'Pending', 4),

('Conduct inventory check of minibar items in rooms 101-110', '2025-10-08', 'In Progress', 6),

('Service AC unit in Room 205', '2025-10-09', 'Pending', NULL), -- Unassigned task

('Follow up on customer feedback from booking #1', '2025-10-07', 'Completed', 5);


-- ================================================================

-- ================== SCM MODULE DATA INSERTION ==================

-- ================================================================


-- 1. Insert Suppliers

INSERT INTO `supplier` (`name`, `contact_info`) VALUES

('Global Linen Supply', 'sales@globallinen.com'),

('Fresh Foods Inc.', 'orders@freshfoods.com'),

('Hospitality Essentials', 'contact@hospitalityessentials.com');
```

```
-- 2. Insert Supplier Contact Persons

INSERT INTO `supplier_contact_persons` (`supplier_id`, `contact_person`) VALUES

(1, 'Sarah Connor'),

(1, 'John Reese'),

(2, 'Mike Ervin'),

(3, 'Laura Palmer');


-- 3. Insert Products

INSERT INTO `product` (`name`) VALUES

('King Size Bed Sheets'),

('Bath Towels'),

('Shampoo (1oz bottle)'),

('Coffee Beans (1kg bag)'),

('Sanitizer (5L can)');


-- 4. Insert Product Allergens

INSERT INTO `product_allergens` (`product_id`, `allergen`) VALUES

(4, 'Caffeine');


-- 5. Insert Warehouses (linking them to branches and warehouse managers)

INSERT INTO `warehouse` (`location`, `branch_id`, `manager_id`) VALUES

('Hanamkonda Central Storage Unit A', 1, 3),

('Hyderabad Hitech Basement Storage', 2, NULL);


-- 6. Insert Inventory (stocking products into warehouses)

INSERT INTO `inventory` (`quantity_on_hand`, `warehouse_id`, `product_id`) VALUES

(500, 1, 1), -- 500 bed sheets in warehouse 1

(1000, 1, 2),-- 1000 towels in warehouse 1

(2000, 1, 3),-- 2000 shampoo bottles in warehouse 1

(100, 1, 4), -- 100kg coffee beans in warehouse 1

(200, 2, 5); -- 200 sanitizer cans in warehouse 2
```

```
-- 7. Insert Purchase Orders
INSERT INTO `purchase_order` (`order_date`, `status`, `supplier_id`) VALUES
('2025-09-20', 'Delivered', 1),
('2025-10-01', 'Placed', 2),
('2025-10-03', 'Shipped', 3);

-- 8. Insert Purchase Order Items
INSERT INTO `purchase_order_item` (`order_id`, `product_id`, `quantity`) VALUES
(1, 1, 100), -- 100 King Size Bed Sheets on order 1
(1, 2, 200), -- 200 Bath Towels on order 1
(2, 4, 50),  -- 50kg Coffee Beans on order 2
(3, 3, 1000),-- 1000 Shampoo bottles on order 3
(3, 5, 50);  -- 50 Sanitizer cans on order 3

-- ================================================================
-- ================== CRM MODULE DATA INSERTION ===================
-- ================================================================

-- 1. Insert Branch Phone Numbers
INSERT INTO `branch_phone_numbers` (`branch_id`, `phone_number`) VALUES
(1, '+918702000111'),
(1, '+918702000112'),
(2, '+91404000222'),
(3, '+918702000333');

-- 2. Insert Room Types
INSERT INTO `room_type` (`type_name`, `price`) VALUES
('Standard', 3500.00),
('Deluxe', 5500.00),
('Suite', 9500.00);
```

```
-- 3. Insert Rooms (linking them to branches and room types)
INSERT INTO `room` (`room_number`, `branch_id`, `room_type_name`) VALUES
('101', 1, 'Standard'),
('102', 1, 'Standard'),
('201', 1, 'Deluxe'),
('301', 1, 'Suite'),
('101', 2, 'Standard'),
('201', 2, 'Deluxe');

-- 4. Insert Room Amenities
INSERT INTO `room_amenities` (`room_id`, `amenity`) VALUES
(1, 'Wi-Fi'), (1, 'TV'),
(2, 'Wi-Fi'), (2, 'TV'),
(3, 'Wi-Fi'), (3, 'TV'), (3, 'AC'), (3, 'Minibar'),
(4, 'Wi-Fi'), (4, 'TV'), (4, 'AC'), (4, 'Minibar'), (4, 'Jacuzzi'),
(5, 'Wi-Fi'), (5, 'TV'),
(6, 'Wi-Fi'), (6, 'TV'), (6, 'AC');

-- 5. Insert Customers
INSERT INTO `customer` (`name`, `email`, `address`) VALUES
('John Doe', 'john.doe@email.com', '123 Main St, Hanamkonda'),
('Jane Smith', 'jane.smith@email.com', '456 Oak Ave, Hyderabad'),
('Peter Jones', 'peter.jones@email.com', '789 Pine Ln, Warangal');

-- 6. Insert Customer Phone Numbers
INSERT INTO `customer_phone_numbers` (`customer_id`, `phone_number`) VALUES
(1, '+919876543210'),
(2, '+919988776655'),
(2, '+919988776644');
```

```
-- 7. Insert Loyalty Program Data

INSERT INTO `loyalty_program` (`customer_id`, `tier_level`, `points`) VALUES

(1, 'Gold', 1250),

(2, 'Silver', 600);

-- Customer 3 is not yet in the loyalty program.


-- 8. Insert Bookings

INSERT INTO `booking` (`check_in_date`, `check_out_date`, `customer_id`, `room_id`) VALUES

('2025-09-25', '2025-09-28', 1, 3), -- John Doe in a Deluxe room at branch 1

('2025-10-05', '2025-10-08', 2, 6), -- Jane Smith in a Deluxe room at branch 2

('2025-10-12', '2025-10-15', 1, 4); -- John Doe in a Suite room at branch 1


-- 9. Insert Booking Special Requests

INSERT INTO `booking_special_requests` (`booking_id`, `request`) VALUES

(1, 'Extra pillows'),

(1, 'Late check-out if possible'),

(3, 'Champagne on arrival');


-- 10. Insert Payments

-- Note: Payment amounts might not exactly match room price * nights due to taxes, discounts, etc.

INSERT INTO `payment` (`amount`, `payment_date`, `payment_method`, `booking_id`) VALUES

(17500.00, '2025-09-28 11:00:00', 'Credit Card', 1),

(18200.50, '2025-10-08 09:30:00', 'Debit Card', 2),

(30150.00, '2025-10-12 14:00:00', 'Online Transfer', 3);


-- 11. Insert Feedback

INSERT INTO `feedback` (`rating`, `comments`, `booking_id`) VALUES

(5, 'Excellent stay! The room was clean and the staff was very friendly. The special request for extra pillows was handled promptly.', 1),

(4, 'Good hotel, convenient location. The AC in the room was a bit noisy but overall a pleasant experience.', 2);
```

Testing Queries

Customer Relationship Management (CRM)

Query 1: Identify the top 5 most valuable customers

- **Use Case:** This query helps the marketing team identify the highest-spending customers over the last year. This information is valuable for targeting them with exclusive loyalty rewards and personalized marketing campaigns to enhance retention.
- **Code:**
  SQL
  ```
  SELECT
     c.customer_id,
     c.name,
     SUM(p.amount) AS total_spending
  FROM
     customer c
  JOIN
     booking b ON c.customer_id = b.customer_id
  JOIN
     payment p ON b.booking_id = p.booking_id
  WHERE
     p.payment_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
  GROUP BY
     c.customer_id, c.name
  ORDER BY
     total_spending DESC
  LIMIT 5;
  ```

Query 2: List all customers in the loyalty program

- **Use Case:** This query is useful for marketing and customer retention efforts by providing a complete list of customers enrolled in the loyalty program, along with their current tier and points balance.
- **Code:**
  SQL
  ```
  SELECT
      c.name AS customer_name,
      c.email,
      lp.tier_level,
      lp.points
  FROM
      customer c
  JOIN
      loyalty_program lp ON c.customer_id = lp.customer_id
  ORDER BY
      lp.points DESC;
  ```

Query 3: Identify rooms with specific amenities

- **Use Case:** This query allows front-desk staff to efficiently find available rooms that meet specific guest requests, such as the presence of 'Wi-Fi', a 'TV', or 'AC'. This improves customer service and streamlines the room allocation process.
- **Code:**
  SQL
  ```
  SELECT
      r.room_number,
      br.location,
      r.room_type_name
  FROM
      room r
  JOIN
      room_amenities ra ON r.room_id = ra.room_id
  JOIN
      branch br ON r.branch_id = br.branch_id
  WHERE
  ```

```
    ra.amenity = 'Wi-Fi' -- Can be changed to other amenities.
ORDER BY
    br.location, r.room_number;
```

Enterprise Resource Planning (ERP)

Query 1: Generate a payroll list for all employees

- **Use Case:** Essential for the HR department, this query generates a list of all employees, their departments, and their salaries to facilitate accurate and efficient payroll processing.
- **Code:**
  SQL
  ```
  SELECT
      e.name AS employee_name,
      d.name AS department_name,
      e.salary
  FROM
      employee e
  JOIN
      department d ON e.department_id = d.department_id
  ORDER BY
      d.name, e.name;
  ```

Query 2: Calculate employee count and average salary per department

- **Use Case:** This query provides key metrics for HR to analyze workforce distribution and compensation structures. The results are useful for budgeting, strategic planning, and ensuring equitable pay across the organization.
- **Code:**
  SQL
  ```
  SELECT
  ```

```
    d.name AS department_name,
    COUNT(e.employee_id) AS number_of_employees,
    AVG(e.salary) AS average_salary
FROM
    department d
JOIN
    employee e ON d.department_id = e.department_id
GROUP BY
    d.name
ORDER BY
    number_of_employees DESC;
```

Query 3: List all currently pending tasks and assigned staff

- **Use Case:** This query helps managers track open tasks and monitor employee workloads. By providing a clear view of pending assignments and their due dates, it aids in project management and resource allocation.
- **Code:**
  SQL
  ```
  SELECT
      t.description,
      t.due_date,
      e.name AS assigned_to
  FROM
      task t
  JOIN
      staff s ON t.assigned_to_staff_id = s.employee_id
  JOIN
      employee e ON s.employee_id = e.employee_id
  WHERE
      t.status = 'Pending'
  ORDER BY
      t.due_date ASC;
  ```

Query 4: Find employees with more than 5 years of service

- **Use Case:** This query helps identify long-serving employees. This information is valuable for employee recognition programs, identifying candidates for succession planning, and acknowledging loyalty to the company.
- **Code:**
  SQL
  ```
  SELECT
    name,
    hire_date,
    TIMESTAMPDIFF(YEAR, hire_date, CURDATE()) AS years_of_service
  FROM
    employee
  WHERE
    TIMESTAMPDIFF(YEAR, hire_date, CURDATE()) > 5
  ORDER BY
    hire_date ASC;
  ```

Supply Chain Management (SCM)

Query 1: Show current inventory levels for all products

- **Use Case:** A critical query for inventory management, this provides a comprehensive overview of product stock levels across all warehouses. This is essential for making informed restocking and distribution decisions.
- **Code:**
  SQL
  ```
  SELECT
    w.location AS warehouse_location,
    p.name AS product_name,
    i.quantity_on_hand
  FROM
    inventory i
  JOIN
  ```

```
    warehouse w ON i.warehouse_id = w.warehouse_id
JOIN
    product p ON i.product_id = p.product_id
ORDER BY
    w.location, p.name;
```

Query 2: List all suppliers and their contact persons

- **Use Case:** This query generates a straightforward contact list for all suppliers. It is a fundamental tool for the procurement team to manage vendor relationships and streamline communication.
- **Code:**
  SQL
  ```
  SELECT
      s.name AS supplier_name,
      s.contact_info,
      scp.contact_person
  FROM
      supplier s
  JOIN
      supplier_contact_persons scp ON s.supplier_id = scp.supplier_id
  ORDER BY
      s.name, scp.contact_person;
  ```

Query 3: Identify products that need restocking

- **Use Case:** This query automates inventory control by identifying products whose quantities have fallen below a predefined threshold (e.g., 50 units). This helps prevent stockouts and ensures timely reordering.
- **Code:**
  SQL
  ```
  SELECT
      w.location AS warehouse_location,
      p.name AS product_name,
      i.quantity_on_hand
  FROM
  ```

```
    inventory i
JOIN
    warehouse w ON i.warehouse_id = w.warehouse_id
JOIN
    product p ON i.product_id = p.product_id
WHERE
    i.quantity_on_hand < 50  -- Threshold can be adjusted as needed.
ORDER BY
    i.quantity_on_hand ASC;
```

Conclusion

The HotelSphere project successfully addressed the critical need for a unified and efficient data management system for a hotel chain. By designing and implementing a centralized SQL database, we have created a single source of truth that integrates CRM, ERP, and SCM functionalities. The core of the solution is a robust relational model, normalized to BCNF, which guarantees data integrity, minimizes redundancy, and prevents modification anomalies that plague denormalized systems.

The structured approach, from problem statement analysis to entity identification, normalization, and final implementation with DDL/DML scripts, has resulted in a database that is both powerful and scalable. The system is capable of handling complex operations, from guest bookings and employee payroll to supply chain management and inventory control.

The development of analytical queries demonstrates the significant business value of this integrated approach. Management can now easily access key performance indicators, such as occupancy trends, top customer metrics, and supply chain status, enabling data-driven strategic decisions. In conclusion, HotelSphere provides a solid technological foundation that can streamline operations, enhance customer service, and support the future growth of the hotel Chain.

Outcome

The successful completion of the HotelSphere project has yielded the following key outcomes and deliverables:

- **A Fully Normalized Relational Database:** An enterprise-grade database schema normalized up to BCNF, ensuring high data integrity and minimal redundancy.
- **Integrated Management System:** A single, cohesive system that seamlessly combines modules for Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), and Supply Chain Management (SCM).
- **Complete DDL and DML Scripts:** A full set of CREATE TABLE scripts for easy deployment and INSERT scripts with sample data for testing and demonstration purposes.
- **Business Intelligence Capabilities:** A suite of powerful SQL queries designed to perform business analytics, providing insights into occupancy trends, top customers, payroll expenses, and supply chain efficiency.
- **Scalable Architecture:** A robust database architecture that can easily scale to accommodate an increasing number of hotels, customers, and transactions as the business grows.
- **Improved Operational Efficiency:** The centralized system eliminates data silos, automates data retrieval, and provides a holistic view of the business, significantly improving the efficiency of day-to-day operations.