ASHRAY

Hotel Reservations

Everything you are looking for in hotels.

Ashray Hotel Reservation System

Overview:

The hospitality industry in the United States is a vital sector that contributes significantly to the economy. With millions of tourists and business travelers seeking accommodation options across the country, there is a growing demand for efficient hotel reservation systems that offer convenience, reliability, and value-added services. **Ashray** aims to meet these needs by offering a comprehensive platform for hotel bookings, serving as a marketplace for both hotels and customers. Introducing a loyalty program with royalty points will encourage customers to make repeat bookings, building long-term relationships and boosting customer loyalty. Additionally, providing hotels with an inventory management system will streamline their operations and enhance overall efficiency

Objectives:

The objectives of the Ashray project are as follows:

- Developing a user-friendly hotel reservation system accessible to both customers and hotels, catering to the market's diverse needs.
- Establishing a marketplace where hotels can list their properties, manage bookings, and update availability in real-time, providing customers with a wide range of accommodation options.
- Implement a loyalty program that rewards customers for booking on our platform, encouraging repeat bookings and increasing customer retention.
- Providing an inventory management system to hotels, enabling them to manage their resources efficiently, optimize room availability, and improve overall operational efficiency.
- Enhancing customer satisfaction and loyalty through personalized services, seamless booking experiences, and value-added benefits.

Scope:

The scope of the Ashray project includes the following key components:

- Design and development of a web-based platform for hotel reservations, marketplace integration, loyalty program implementation, and inventory management.
- Creation of user interfaces for both customers and hotels, ensuring ease of use, functionality, and accessibility across devices.
- Integration of a marketplace interface allowing hotels to list their properties, manage bookings, and update availability in real-time.
- Development of a loyalty program framework, including point accrual, redemption options, and personalized rewards for customers.
- Provision of an inventory management system accessible to hotels through our platform, enabling efficient resource management and optimization.

- Implementing robust security measures to safeguard user data and ensure compliance with relevant regulations and industry standards.

Deliverables:

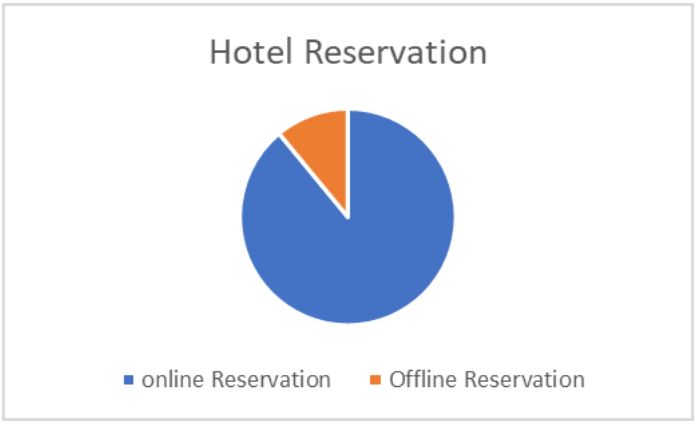
A fully functional hotel reservation system with marketplace integration, enabling seamless booking experiences for both customers and hotels.
Implementation of a loyalty program framework, including the development of reward tiers, point accrual mechanisms, and redemption options.
Deployment of an inventory management system accessible to hotels through our platform, facilitating efficient resource allocation and optimization.
User manuals, documentation, and training materials for system usage, administration, and support.
Marketing collateral, promotional materials, and communication strategies to promote the platform and incentivize customer engagement.

Marketing:

The following are the 3 P's of Marketing (Price, Place and Promotion) strategy:

Ashray utilizes a transparent pricing structure to ensure fairness and encourage participation from both hosts and guests. Hosts are charged a 3% service fee, while guests typically pay around 10% of the booking subtotal, along with 5% on royalty points. This approach balances affordability for customers with profitability for the platform, contributing to its success and sustainability.

In terms of its presence, Ashray primarily operates online, leveraging digital platforms to tap into the vast hospitality market. However, the platform also explores offline partnerships with hotels to extend its reach. This multichannel approach allows Ashray to cater to various customer preferences and maximize its market penetration.



For promotion, Ashray adopts a comprehensive marketing strategy to increase brand awareness and drive user adoption. Through targeted digital campaigns, promotional activities, and strategic communication, Ashray effectively communicates its value proposition to customers and hotel partners. This approach establishes Ashray as a trusted choice in the competitive hospitality landscape, positioning it for long-term success and growth.

Stakeholders:

The stakeholders involved in the Ashray project include:

- *Project Sponsor*: Provides funding, resources, and support for the project.
- *Project Manager and Team*: Oversees project execution, manages resources, and ensures timely delivery of project objectives.
- *Development Team*: Responsible for designing, developing, and testing the platform, marketplace, loyalty program, and inventory management system.
- *Marketing Team*: Develops promotional strategies, marketing campaigns, and communication plans to promote the platform and drive customer engagement.
- *Customer Service Team*: Provides support, assistance, and troubleshooting for customers and hotels using the platform.
- *Hotel Partners:* Collaborate with the project team to list their properties, manage bookings, and utilize the inventory management system.
- *Customers*: Utilize the platform to search, compare, and book accommodations, as well as participate in the loyalty program to earn rewards and benefits.

Risks and Mitigation:

The Ashray project faces several risks that may impact its successful execution. These risks include:

- *Technical Challenges*: Potential issues with platform development, integration, and testing may delay project timelines. To mitigate this risk, the project team will conduct regular reviews, testing, and collaboration to address technical issues promptly.
- *Adoption Rate*: Low user adoption and engagement may hinder the success of the platform and loyalty program. To mitigate this risk, the marketing team will implement comprehensive marketing strategies, promotional campaigns, and user education initiatives to increase awareness and incentivize customer participation.
- *Competitor Response*: Intense competition from existing players in the market may affect the platform's ability to attract customers and hotels. To mitigate this risk, the project team will continuously monitor competitor activities, analyze market trends, and adapt strategies to differentiate the platform and maintain a competitive edge.

COMPLEX QUERIES:

1. **Number of Booked Rooms for Each Date and Hotel:**

The number of BOOKED rooms in each hotel for each date between May 1, 2024, and May 10, 2024, you can modify the query to group by date and hotel and count the available rooms.

```
SELECT ra.Date, h.Hotel_ID, h.Hotel_Name, COUNT(ra.Room_ID) AS BOOKED_ROOM
FROM Hotel h
INNER JOIN Room r ON h.Hotel_ID = r.Hotel_ID
LEFT JOIN RoomAvailability ra ON r.Room_ID = ra.Room_ID
WHERE ra.Date BETWEEN '2024-05-01' AND '2024-05-10'
AND ra.Available = 0
GROUP BY ra.Date, h.Hotel_ID, h.Hotel_Name
ORDER BY ra.Date, h.Hotel_ID;
```

Output:

Date	Hotel_ID	Hotel_Name	BOOKED_ROOM
5/1/24	7	Country Manor	1
5/1/24	40	Lakeside Paradise Inn	1
5/1/24	41	Seaside Serenity Resort	1
5/1/24	51	Riverside Retreat Resort	2
5/1/24	60	Lakeside Luxury Inn	1
5/1/24	68	Riverfront Romance Resort	1
5/1/24	72	Mountain Majesty Manor	1
5/1/24	83	Beachfront Bliss Inn	1
5/2/24	7	Country Manor	1
5/2/24	41	Seaside Serenity Resort	1

Description:

This query retrieves the count of booked rooms for each hotel on each date between May 1, 2024, and May 10, 2024.
It joins the Hotel, Room, and RoomAvailability tables to get the necessary information.
The RoomAvailability table is used to check whether a room is available on a particular date (ra.Available = 0 indicates it's booked).
Results are grouped by date, hotel ID, and hotel name to count the number of booked rooms for each combination.

2. **Check for Overbooked Rooms:**

```
SELECT DISTINCT b1.Booking_ID AS Booking1_ID, b2.Booking_ID AS Booking2_ID,
    b1.Room_ID, b1.Checkin_Date AS Booking1_Checkin, b1.Checkout_Date AS Booking1_Checkout,
    b2.Checkin_Date AS Booking2_Checkin, b2.Checkout_Date AS Booking2_Checkout
FROM Booking b1
JOIN Booking b2 ON b1.Room_ID = b2.Room_ID AND b1.Booking_ID < b2.Booking_ID
WHERE b1.Checkin_Date BETWEEN b2.Checkin_Date AND b2.Checkout_Date
  OR b1.Checkout_Date BETWEEN b2.Checkin_Date AND b2.Checkout_Date;
```

Output:

Booking1_ID	Booking2_ID	Room_ID	Booking1_Checkin	Booking1_Checkout	Booking2_Checkin	Booking2_Checkout

Description:

This query identifies overlapping bookings for the same room on the same date, which could indicate overbooking.

It joins the Booking table with itself to compare different bookings (b1 and b2 aliases are used).

The WHERE clause checks if the check-in or check-out date of one booking falls within the check-in and check-out dates of another booking.

The query returns pairs of bookings (Booking1_ID and Booking2_ID) that overlap for the same room on the same date.

3. **Booking Date-wise Revenue:**

```
Select DATE_FORMAT(b.created_at, '%Y-%m') as booking_month,
sum(b.booking_amount - pp.Amount) as profit , sum(b.booking_amount) as revenue,
sum(case when pp.type = "payable" then pp.Amount end ) as hotel_payouts,
sum(case when pp.type = "royality payable" then pp.Amount end ) as
royality_payable_cost

from booking b
left join Payment_Payable pp on pp.booking_id = b.booking_id
group by DATE_FORMAT(b.created_at, '%Y-%m');
```

Output:

booking_month	profit	revenue	hotel_payouts	royality_payable_cost
2024-04	747563.25	1423930	640768.5	35598.25

Description:

Calculates the revenue, profit, and payouts on a monthly basis.

Joins the Booking table with the Payment_Payable table to get payment information.

Revenue, profit, and payouts are aggregated on a monthly basis using the DATE_FORMAT function to extract the month from the booking creation date.

Hotel-wise Revenue and Payout:

Similar to the previous query but aggregates revenue and payouts on a monthly basis for each hotel.

Joins the Booking table with the Hotel table to get hotel information.

4. **Customer-wise Monthly Revenue:**

```
Select Date_format(b.checkin_date, '%Y-%m') as accrual_month, h.Cust_ID,
h.Cust_Name,
 sum(b.booking_amount - pp.Amount) as profit , sum(b.booking_amount) as revenue,
sum(case when pp.type = "payable" then pp.Amount end ) as hotel_payouts,
sum(case when pp.type = "royality payable" then pp.Amount end ) as
royality_payable_cost
 from booking b
 left join Customer h on b.Cust_ID = h.Cust_ID
 left join Payment_Payable pp on pp.booking_id = b.booking_id
 group by DATE_FORMAT(b.checkin_date, '%Y-%m'), h.Cust_ID, h.Cust_Name;
```

Output:

accrual_month	Cust_ID	Cust_Name	profit	revenue	hotel_payouts	royality_payable_cost
2024-10	163	Jonathan Martinez	105	200	90	5
2025-10	206	Jessica Mitchell	735	1400	630	35
2026-08	90	Julie Gonzalez	178.5	340	153	8.5
2024-02	55	Mark Perez	105	200	90	5
2025-07	28	Deborah Hernandez	420	800	360	20
2026-08	142	Michael Lee	210	400	180	10
2024-08	322	Michael Miller	283.5	540	243	13.5
2025-02	212	Emily Moore	299.25	570	256.5	14.25
2024-09	399	Jessica Lee	945	1800	810	45
2026-03	364	Michael Hernandez	472.5	900	405	22.5
2026-08	65	Timothy Coleman	1260	2400	1080	60
2025-05	113	Kevin Barnes	315	600	270	15

Description

This SQL query retrieves monthly revenue-related information on a customer-by-customer basis. It calculates the profit, revenue, hotel payouts, and royalty payable costs for each customer, aggregated monthly. This information helps in understanding the financial performance of each customer over time.

5. **Calculates revenue, profit, and payouts on a monthly basis for each customer.**

```
select sd1.check_month, count(sd1.Cust_ID), sum( sd1.no_of_booking) as
no_of_booking,
 sum(case when sd1.lead_1 > 0 then 1 else 0 end) as person_n_1,
sum(case when sd1.lead_2 > 0 then 1 else 0 end) as person_n_2,
sum(case when sd1.lead_3 > 0 then 1 else 0 end) as person_n_3,
sum(case when sd1.lead_4 > 0 then 1 else 0 end) as person_n_4,
sum(case when sd1.lead_5 > 0 then 1 else 0 end)as person_n_5,
sum(case when sd1.lead_6 > 0 then 1 else 0 end)as person_n_6
from
(
select sd.check_month,sd.Cust_ID,sd.Cust_Name,sd.no_of_booking,
lag(sd.no_of_booking, 1, 0) OVER(PARTITION BY sd.Cust_ID ORDER BY
sd.check_month) as lead_1,
 lag(sd.no_of_booking, 2, 0) OVER(PARTITION BY sd.Cust_ID ORDER BY
sd.check_month) as lead_2,
 lag(sd.no_of_booking, 3, 0) OVER(PARTITION BY sd.Cust_ID ORDER BY
sd.check_month) as lead_3,
 lag(sd.no_of_booking, 4, 0) OVER(PARTITION BY sd.Cust_ID ORDER BY
sd.check_month) as lead_4,
 lag(sd.no_of_booking, 5, 0) OVER(PARTITION BY sd.Cust_ID ORDER BY
sd.check_month) as lead_5,
 lag(sd.no_of_booking, 6, 0) OVER(PARTITION BY sd.Cust_ID ORDER BY
sd.check_month) as lead_6
from
(
Select DATE_FORMAT(b.checkin_date, '%Y-%m') as check_month,
b.Cust_ID,c.Cust_Name, count(Distinct b.Booking_ID) as no_of_booking
from booking as b
left join customer as c on c.Cust_ID = b.Cust_ID

group by DATE_FORMAT(b.checkin_date, '%Y-%m') , b.Cust_ID,c.Cust_Name

 )sd
 ) sd1
 group by sd1.check_month;
```

Output:

check_month	count(sd1.Cust_ID)	no_of_booking	person_n_1	person_n_2	person_n_3	person_n_4
2025-07	48	53	40	26	23	9
2025-03	57	60	43	25	16	6
2026-06	48	52	47	41	30	23
2024-03	62	66	12	0	0	0
2024-08	50	51	36	15	3	0
2024-11	44	47	34	20	2	1
2025-04	49	49	42	33	21	9
2025-06	47	50	42	30	14	7
2026-03	37	39	37	32	24	15
2024-02	50	55	6	0	0	0
2025-05	43	46	37	26	14	10

Description:

Joins the Booking table with the Customer table to get customer information.

Booking Waterfall Month-over-Month:

Analyzes the month-over-month trend in bookings for each customer.

Uses window functions to calculate the previous months' booking counts (lead_1, lead_2, etc.) for each customer.

6. **Average Order Value and Booking Days:**

```
Select DATE_FORMAT(b.checkin_date, '%Y-%m') as check_month,
    count(Distinct b.Booking_ID) as no_of_booking,
avg(b.booking_amount) as avg_booking_amount,
avg(datediff(b.Checkout_Date,b.Checkin_Date)) as avg_stay_no_of_days
from booking as b
group by DATE_FORMAT(b.checkin_date, '%Y-%m');
```

Output:

check_month	no_of_booking	avg_booking_amount	avg_stay_no_of_days
2024-01	51	432.745098	4.1373
2024-02	55	413.272727	3.8545
2024-03	66	448.106061	4.2727
2024-04	69	446.666667	4.087
2024-05	52	401.442308	3.8846
2024-06	53	411.981132	3.7736
2024-07	56	414.107143	3.9286
2024-08	51	432.352941	4.0392
2024-09	47	469.893617	4.383
2024-10	52	419.134615	3.9038
2024-11	47	427.340426	4.0426

Description:

Calculates the average booking amount and the average number of days stayed per booking on a monthly basis.

Aggregates booking data by month and calculates average values.

7. **Hotel-wise Average Order and Booking Days:**

```
Select DATE_FORMAT(b.checkin_date, '%Y-%m') as check_month, h.Hotel_Name,
    count(Distinct b.Booking_ID) as no_of_booking,
avg(b.booking_amount) as avg_booking_amount,
avg(datediff(b.Checkout_Date,b.Checkin_Date)) as avg_stay_no_of_days
from booking as b
left join hotel as h on h.Hotel_ID = b.Hotel_ID
group by DATE_FORMAT(b.checkin_date, '%Y-%m'), h.Hotel_Name;
```

Output:

check_month	Hotel_Name	no_of_booking	avg_booking_amount	avg_stay_no_of_days
2024-01	Beachfront Beauty Resort	2	635	6.5
2024-01	Beachfront Bliss Resort	1	270	3
2024-01	Beachside Resort	2	475	3.5
2024-01	City Center Charm Suites	2	205	2
2024-01	City Center Comfort Inn	3	533.333333	4.3333
2024-01	City Center Courtyard	2	400	4
2024-01	City Center Suites	4	345	3.25
2024-01	City Lights Hotel	1	600	6
2024-01	Cityscape Suites	1	150	1
2024-01	Country Club Charm Inn	1	425	5
2024-01	Country Club Resort	1	500	5
2024-01	Country Manor	1	450	3

Description:

Similar to the previous query but broken down by hotels.

Employee Information:

Selects all employee information from the Employee_Info table

8. **Changes in Booking:**

```
SELECT
  b.Booking_ID,
  b.Cust_ID,
  b.Room_ID,
  b.Hotel_ID,
  b.Checkin_Date,
  b.Checkout_Date
FROM
  Booking b
JOIN
  State_Transition st ON b.Booking_ID = st.Booking_ID
WHERE
  st.Status IN ('Cancelled', 'Date Change');
```

Output:

Booking_ID	Cust_ID	Room_ID	Hotel_ID	Checkin_Date	Checkout_Date

Description:

Identifies bookings that have been canceled or had date changes.
Joins the Booking table with the Booking_Changes table to get information about booking status changes.

G. **Salary Amount Monthly:**

```
SELECT
  ei.Emp_ID,
  ei.Emp_First_Name,
  ei.Emp_Last_Name,
  es.Joining_date,
  DATE_FORMAT(es.Joining_date, '%Y-%m') AS Start_Month,
  es.Salary / 12 AS Monthly_Salary,
  DATE_FORMAT(es.Joining_date, '%Y-%m') AS Salary_Month,
  SUM(es.Salary / 12) OVER (ORDER BY es.Joining_date ASC) AS Cumulative_Salary
FROM
  Employee_Info ei
JOIN
  Employees_Salary es ON ei.Emp_ID = es.Emp_ID;
```

Output:

Emp_ID	Emp_First_Name	Emp_Last_Name	Joining_date	Start_Month	Monthly_Salary	Salary_Month	Cumulative_Salary
35	Michael	Jones	1/8/24	2024-01	4166.666667	2024-01	4166.666667
19	William	Rodriguez	1/17/24	2024-01	6250	2024-01	10416.66667
34	Amelia	Brown	1/18/24	2024-01	6250	2024-01	16666.66667
53	Michael	Jones	1/20/24	2024-01	4583.333333	2024-01	21250
44	Isabella	Perez	2/3/24	2024-02	7083.333333	2024-02	28333.33333
40	Olivia	Lopez	2/12/24	2024-02	5416.666667	2024-02	33750
6	Olivia	Garcia	5/4/24	2024-05	7083.333333	2024-05	40833.33333
46	Sophia	Jones	5/7/24	2024-05	4583.333333	2024-05	45416.66667
27	James	Brown	5/30/24	2024-05	6250	2024-05	51666.66667
1	John	Doe	6/4/24	2024-06	4583.333333	2024-06	56250

Description:

Calculates the monthly salary for each employee and the cumulative salary over time. Joins the Employee_Info table with the Employees_Salary table to get salary information and calculates monthly and cumulative salaries.

10. hotel wise revenue and payout montly based on actual accrual

```
 Select Date_format(b.checkin_date, '%Y-%m') as accrual_month, h.Hotel_Name,
sum(b.booking_amount - pp.Amount) as profit , sum(b.booking_amount) as revenue,
sum(case when pp.type = "payable" then  pp.Amount end ) as hotel_payouts,
sum(case when pp.type = "royality payable" then  pp.Amount end ) as royality_payable_cost
  from booking  b
 left join hotel h on b.Hotel_ID = h.Hotel_ID
 left join Payment_Payable pp on pp.booking_id = b.booking_id
 group by DATE_FORMAT(b.checkin_date, '%Y-%m'), h.Hotel_Name;
```

Output:

accrual_month	Hotel_Name	profit	revenue	hotel_payouts	royality_payable_cost
2024-10	Urban Oasis	420	800	360	20
2025-10	City Center Courtyard	735	1400	630	35
2026-08	Seaside Sanctuary Lodge	987	1880	846	47
2024-02	Valley View Hotel	735	1400	630	35
2025-07	Downtown Delight Hotel	420	800	360	20
2026-08	Mountain Magic Inn	210	400	180	10
2024-08	City Center Suites	567	1080	486	27
2025-02	Beachfront Bliss Inn	1685.25	3210	1444.5	80.25
2024-09	Mountain View Resort	1890	3600	1620	90
2026-03	Country Manor	472.5	900	405	22.5
2026-08	Tropical Paradise Inn	735	1400	630	35
2025-05	Oceanfront Oasis Resort	315	600	270	15

Description

This SQL query computes the monthly revenue, profit, hotel payouts, and royalty payable costs for each hotel based on actual accrual. It selects data from the booking table and joins it with the hotel and payment payable tables. By grouping the data by month and hotel name, the query summarizes the financial metrics for each hotel on a monthly basis. This provides valuable insights into the financial performance of each hotel over time.

PROCEDURES:

1. **UpdateBooking:**

Parameters:

p_BookingID: The ID of the booking to be updated.

p_CheckinDate: The new check-in date for the booking.

p_CheckoutDate: The new check-out date for the booking.

p_BookingAmount: The new booking amount for the booking.

p_CurrentActive: A boolean parameter indicating whether the booking is currently active.

It updates the **'Checkin_Date'**, **'Checkout_Date'**, **'Booking_amount'**, **'Current_Active**', and **Updated_At** fields in the '**Booking'** table for the specified booking ID.

Code:

```
Select * from booking

-- update booking procedure

DELIMITER //

CREATE PROCEDURE UpdateBooking(

  IN p_BookingID INT,

  IN p_CheckinDate DATE,

  IN p_CheckoutDate DATE,

  IN p_BookingAmount DECIMAL(10, 2),

  IN p_CurrentActive BOOLEAN )

BEGIN

  UPDATE Booking

  SET Checkin_Date = p_CheckinDate,

    Checkout_Date = p_CheckoutDate,

    Booking_amount = p_BookingAmount,

    Current_Active = p_CurrentActive,

    Updated_At = CURRENT_TIMESTAMP

  WHERE Booking_ID = p_BookingID;

END //

DELIMITER ;
```

Output:

Booking_ID	Cust_ID	Room_ID	Hotel_ID	Checkin_Date	Checkout_Date	Created_At	Updated_At	Booking_amount	Settled_Amount	Current_Active
1	163	41	9	10/30/24	10/31/24	4/30/24 17:21	4/30/24 17:21	100	0	1
2	206	172	35	10/5/25	10/12/25	4/30/24 17:21	4/30/24 17:21	700	0	1
3	90	422	85	8/17/26	8/19/26	4/30/24 17:21	4/30/24 17:21	170	0	1
4	55	57	12	2/8/24	2/9/24	4/30/24 17:21	4/30/24 17:21	100	0	1
5	28	141	29	7/14/25	7/18/25	4/30/24 17:21	4/30/24 17:21	400	0	1
6	142	238	48	8/16/26	8/18/26	4/30/24 17:21	4/30/24 17:21	200	0	1
7	322	368	74	8/6/24	8/9/24	4/30/24 17:21	4/30/24 17:21	270	0	1
8	212	262	53	2/14/25	2/17/25	4/30/24 17:21	4/30/24 17:21	285	0	1
9	399	40	8	9/22/24	9/28/24	4/30/24 17:21	4/30/24 17:21	900	0	1

Description:
This procedure is designed to update the details of a booking in the Booking table.

2. RetrieveBookingDetails:
Parameters:

p_BookingID: The ID of the booking to retrieve details for.

It selects all columns (*) from the '**Booking '** table where the **'Booking_ID'** matches the provided ID.

Code:

```
DELIMITER //

CREATE PROCEDURE RetrieveBookingDetails(

  IN p_BookingID INT

)
```

```
BEGIN
  SELECT *
  FROM Booking
  WHERE Booking_ID = p_BookingID;
END //

DELIMITER ;
```

CALL RetrieveBookingDetails(12);

Output:

Booking_ID	Cust_ID	Room_ID	Hotel_ID	Checkin_Date	Checkout_Date	Created_At	Updated_At	Booking_amount	Settled_Amount	Current_Active
12	113	182	37	5/19/25	5/22/25	4/30/24 17:21	4/30/24 17:21	300	0	1

Description:

This procedure retrieves the details of a booking from the Booking table based on the provided booking ID.

3. GetBookedDaysForHotel:
Parameters:

p_HotelID: The ID of the hotel to check bookings for.

p_StartDate: The start date of the period to check bookings for.

p_EndDate: The end date of the period to check bookings for.

It counts the distinct dates where rooms are marked as unavailable **(Available = 0)** in the '**RoomAvailability'** table for the specified hotel and date range.

Code:

```
DELIMITER //
```

```
CREATE PROCEDURE GetBookedDaysForHotel(
  IN p_HotelID INT,
  IN p_StartDate DATE,
  IN p_EndDate DATE,
  OUT p_TotalBookedDays INT
)
BEGIN
  DECLARE totalBookedDays INT;

  SELECT COUNT(DISTINCT Date) INTO totalBookedDays
  FROM RoomAvailability
  WHERE Hotel_ID = p_HotelID
  AND Date BETWEEN p_StartDate AND p_EndDate
  AND Available = 0;

  SET p_TotalBookedDays = totalBookedDays;
END //

DELIMITER ;



CALL GetBookedDaysForHotel(1, '2024-05-01', '2024-05-10', @p_TotalBookedDays);
SELECT @p_TotalBookedDays;
SET @p_TotalBookedDays = 5;

CALL GetBookedDaysForHotel(74, '2024-08-05', '2024-08-30', @p_TotalBookedDays);
SELECT @p_TotalBookedDays;
SET @p_TotalBookedDays = 5;
```

Output:

@p_TotalBookedDays
4

Description: This procedure calculates the total number of booked days for a specific hotel within a given date range.

4. **GetCustomerBookingDetails:**
Parameters:

p_CustID: The ID of the customer to calculate bookings for.

p_StartDate: The start date of the period to calculate bookings for.

p_EndDate: The end date of the period to calculate bookings for.

It counts the number of bookings and calculates the total booking amount from the ’ **Booking ‘**table where the **‘Cust_ID’** matches the provided customer ID and the ‘**Checkin_Date’** is within the specified date range.
CALL GetCustomerBookingDetails(123, '2022-01-01', '2022-01-31', @total_bookings, @total_amount);
SELECT @total_bookings, @total_amount;

@total_bookings	@total_amount
0	NULL

Description: This procedure calculates the number of bookings made by a customer and the total booking amount within a specified time period.

5. **UpdateEmployeeDetails:**
Parameters:

p_EmpID: The ID of the employee whose details are to be updated.

p_NewPhoneNumber: The new phone number for the employee.

p_NewAddress: The new address for the employee.

It updates the **Emp_Phone_Number** and **Employee_Address** fields in the **Employee_Info table** for the specified employee ID.

Code:

```
DELIMITER //


CREATE PROCEDURE UpdateEmployeeDetails(
  IN p_EmpID INT,
  IN p_NewPhoneNumber VARCHAR(20),
  IN p_NewAddress VARCHAR(100)
)
BEGIN
  UPDATE Employee_Info
  SET Emp_Phone_Number = p_NewPhoneNumber,
    Employee_Address = p_NewAddress
  WHERE Emp_ID = p_EmpID;
END //
CALL UpdateEmployeeDetails(4,'555-456-7890', '101 Pine St’);
```

Description: This procedure updates the phone number and address of an employee in the Employee_Info table.

6. UpdateCustomerEmail:
Parameters:

p_CustomerID: The ID of the customer whose email address is to be updated.

p_NewEmail: The new email address for the customer.

It updates the '**Cust_Email_Id'** field in the '**Customer'** table for the specified customer ID.

```
DELIMITER //


CREATE PROCEDURE UpdateCustomerEmail(
    IN p_CustomerID INT,
    IN p_NewEmail VARCHAR(100)
)
BEGIN
    UPDATE Customer
    SET Cust_Email_Id = p_NewEmail
    WHERE Cust_ID = p_CustomerID;
END //
 DELIMITER ;
 CALL UpdateCustomerEmail(123, 'new_email@example.com');
```

Description: This procedure updates the email address of a customer in the Customer table.

7. UpdateCustomerPhoneNumber:
Parameters:

p_CustomerID: The ID of the customer whose phone number is to be updated.

p_NewPhoneNumber: The new phone number for the customer.

It updates the '**Cust_Phone_No'** field in the '**Customer'** table for the specified customer ID.

```
DELIMITER //
 CREATE PROCEDURE UpdateCustomerPhoneNumber(
   IN p_CustomerID INT,
   IN p_NewPhoneNumber VARCHAR(20)
)
BEGIN
   UPDATE Customer
   SET Cust_Phone_No = p_NewPhoneNumber
   WHERE Cust_ID = p_CustomerID;
END //
 DELIMITER ;
CALL UpdateCustomerPhoneNumber(123, '123-456-7890');
```

Description: This procedure updates the phone number of a customer in the Customer table.

8. MarkEmployeeInactive:
Parameters:

p_EmpID INT: This parameter specifies the ID of the employee who is to be marked as inactive. It's an integer value representing the unique identifier of the employee.

The purpose of this stored procedure is to facilitate the deactivation of an employee by updating the **'current_active'** field in the **'Employee_Info'** table. Once the procedure is executed with the ID of the employee to be marked as inactive, it will set the **current_active** flag to **0,** indicating that the employee is no longer active in the system.

```
DELIMITER //


CREATE PROCEDURE MarkEmployeeInactive(
  IN p_EmpID INT
)
BEGIN
  UPDATE Employee_Info
  SET current_active = 0
  WHERE Emp_ID = p_EmpID;
END //


DELIMITER ;
```

Description: This is the name of the stored procedure, indicating its purpose to mark an employee as inactive.

TRIGGERS

Trigger 1: populate_royalty_points

Event: This trigger fires after an insertion (AFTER INSERT) operation on the Booking table.

Purpose: The trigger calculates royalty points based on the booking amount and inserts a record into the Royalty_Points table.

Logic: It calculates the points as 5% of the booking amount (NEW.Booking_amount * 0.05).

Inserts a record into the Royalty_Points table with the booking ID, customer ID, calculated points, and a default value of 0 for the settled amount.

```
DELIMITER //


CREATE TRIGGER populate_royalty_points
AFTER INSERT ON Booking
FOR EACH ROW
BEGIN
  DECLARE points DECIMAL(10, 2);


  -- Calculate the points based on the booking amount
  SET points = NEW.Booking_amount * 0.05; --  5 point for every $100 spent


  -- Insert the points into the Royalty_Points table
  INSERT INTO Royalty_Points (Booking_ID, Cust_ID, Points, Settled_Amount)
  VALUES (NEW.Booking_ID, NEW.Cust_ID, points, 0);
END;
//


DELIMITER
```

Trigger 2: populate_payment_recivables

Event: This trigger also fires after an insertion (AFTER INSERT) operation on the Booking table.

Purpose: It inserts a record into the Payment_Recivables table for the newly inserted booking, indicating an amount receivable from the customer.

Logic: Inserts a record into the Payment_Recivables table with the booking ID, customer ID, booking amount (NEW.Booking_amount), and 'receivable' type.

```
DELIMITER //


CREATE TRIGGER populate_payment_recivables

AFTER INSERT ON Booking

FOR EACH ROW

BEGIN

  -- Insert a record into Payment_Recivables for the newly inserted booking

  INSERT INTO Payment_Recivables (Booking_ID, Cust_ID, Amount, Settled_Amount, Type)

  VALUES (NEW.Booking_ID, NEW.Cust_ID, NEW.Booking_amount, 0, 'receivable');

END;

//


DELIMITER ;
```

Trigger 3: populate_payment_payable

Event: Triggered after an insertion (AFTER INSERT) operation on the Booking table.

Purpose: It calculates the payable amount for the hotel and inserts a record into the Payment_Payable table.

Logic:

Calculates the payable amount as 90% of the booking amount (NEW.Booking_amount * 0.90).

Inserts a record into the Payment_Payable table with the booking ID, hotel ID, customer ID, payable amount, and 'payable' type.

```
DELIMITER //


CREATE TRIGGER populate_payment_payable
AFTER INSERT ON Booking
FOR EACH ROW
BEGIN


   DECLARE points DECIMAL(10, 2);


   -- Calculate the points based on the booking amount
   SET points = NEW.Booking_amount * 0.90;


   -- Insert a record into Payment_Payable for the newly inserted booking
   INSERT INTO Payment_Payable (Booking_ID, Hotel_ID,Cust_ID,  Amount, Settled_Amount, type)
   VALUES (NEW.Booking_ID, NEW.Hotel_ID, NUll, points, 0 , "payable");
END;
//


DELIMITER ;
```

Trigger 4: **populate_payment_to_customer**

Event: Fired after an insertion (AFTER INSERT) operation on the Royalty_Points table.

Purpose: It inserts a record into the Payment_Payable table indicating royalty payment payable to the customer.

Logic:

Inserts a record into the Payment_Payable table with the booking ID, customer ID, calculated points (NEW.Points), and 'royalty payable' type.

```
DELIMITER //


CREATE TRIGGER populate_payment_to_customer

AFTER INSERT ON Royalty_Points

FOR EACH ROW

BEGIN



  -- Insert a record into Payment_Payable for the newly inserted Royalty_Points entry

  INSERT INTO Payment_Payable (Booking_ID, Hotel_ID, Cust_ID, Amount, Settled_Amount, type )

  VALUES (NEW.Booking_ID, NULL,NEW.Cust_ID, NEW.Points,0, "royality payable");

END;

//


DELIMITER ;
```

Trigger 5: **BeforeBookingInsert**

Event: Triggered before an insertion (BEFORE INSERT) operation on the Booking table.

Purpose: It checks if the room is available for the specified check-in and check-out dates.

Logic: Counts the number of rows in the RoomAvailability table where the room is not available for the given dates.

If any such rows exist, it raises an error indicating that the room is not available.

```
DELIMITER //

CREATE TRIGGER BeforeBookingInsert

BEFORE INSERT ON Booking

FOR EACH ROW

BEGIN

  DECLARE available_count INT;


  -- Calculate the number of rows where the room is not available for the given check-in and check-out dates

  SELECT COUNT(*)

  INTO available_count

  FROM RoomAvailability

  WHERE Room_ID = NEW.Room_ID

    AND Hotel_ID = NEW.Hotel_ID

    AND Date BETWEEN NEW.Checkin_Date AND NEW.Checkout_Date

    AND Available = 0;

  -- If available_count is greater than 0, then raise an error

  IF available_count > 0 THEN

    SIGNAL SQLSTATE '45000'

    SET MESSAGE_TEXT = 'Room is not available for the selected dates';

  END IF;

END //

DELIMITER ;
```

Trigger 6: BeforeBookingInsert_update_booking

Event: Triggered after an insertion (AFTER INSERT) operation on the Booking table.

Purpose: Updates the RoomAvailability table to mark the booked rooms as unavailable for each date within the booking period.

Logic:

Uses a loop to update the RoomAvailability table for each date between the check-in and check-out dates of the booking.

```
DELIMITER //


CREATE TRIGGER BeforeBookingInsert_update_booking

After INSERT ON Booking

FOR EACH ROW

BEGIN

  DECLARE d_date DATE;

  -- Set current date to check-in date

  SET d_date = NEW.Checkin_Date;

  -- Loop through each date from check-in to check-out

  WHILE d_date <= NEW.Checkout_Date DO

    -- Update the RoomAvailability table to set "available" to 0 for the given room and hotel on each date

    UPDATE RoomAvailability

    SET Available = 0

    WHERE Room_ID = NEW.Room_ID

      AND Hotel_ID = NEW.Hotel_ID

      AND Date = d_date;

-- Move to the next date

    SET d_date = DATE_ADD(d_date, INTERVAL 1 DAY);

  END WHILE;

END //

DELIMITER ;
```

Trigger 7 : InsertPaymentAfterPayableInsert

Event: Triggered after an insertion (AFTER INSERT) operation on the Payment_Payable table.

Purpose: It inserts a record into the Payment table for royalty payments payable to customers.

Logic:

Checks if the inserted record in Payment_Payable is of type 'royalty payable' and inserts a corresponding record into the Payment table.

```
DELIMITER //


CREATE TRIGGER InsertPaymentAfterPayableInsert

AFTER INSERT ON Payment_Payable

FOR EACH ROW

BEGIN

  IF NEW.Type = 'royalty payable' THEN

    INSERT INTO Payment (Cust_ID, Amount, Payment_Date, Payment_Method)

    VALUES (NEW.Cust_ID, NEW.Amount, CURDATE(), 'Royalty Payment');

  END IF;

END//


DELIMITER ;
```

Trigger 8: AfterBookingChangesInsert

Event: The trigger AfterBookingChangesInsert fires after an insertion operation (INSERT) on the Booking_Changes table..

Purpose: The purpose of this trigger is to maintain a record of state transitions in the State_Transition table whenever there's a change or update in the booking details. It ensures that the State_Transition table accurately reflects the history of changes made to bookings, including date changes and cancellations, by capturing the initial booking state before any modifications are made.

Logic: Retrieve Initial Booking Information: The trigger fetches the initial check-in and check-out dates (Checkin_Date and Checkout_Date) from the Booking table corresponding to the Booking_ID of the newly inserted record in the Booking_Changes table.

Determine Booking Status:

If there's a change in booking dates (NEW.Change_Booking_Dates is not NULL), it sets the bookingStatus variable to 'Date change'.

If there's no change in booking dates, it assumes the booking is cancelled and sets bookingStatus to 'Cancelled'.

Update Booking Table:

If there's a change in booking dates:

It updates the Checkin_Date to the new check-in date (NEW.Change_Booking_Dates).

It calculates the new Checkout_Date by adding the difference between the initial check-out date (initialCheckoutDate) and the initial check-in date (initialCheckinDate) to the new check-in date.

If the booking is cancelled:

It sets the Current_Active field to 0 for the booking, marking it as inactive.

Insert into State_Transition:

After determining the booking status and updating the booking table accordingly, it inserts a new record into the State_Transition table.

This insertion includes the Booking_ID, old_checkin_date, and old_checkout_date from the initial booking details, along with the determined bookingStatus.

```
DELIMITER //
CREATE TRIGGER AfterBookingChangesInsert
AFTER INSERT ON Booking_Changes
FOR EACH ROW
BEGIN
  DECLARE bookingStatus VARCHAR(50);
  DECLARE initialCheckoutDate DATE;
  DECLARE initialCheckinDate DATE;

  -- Retrieve the initial Checkin_Date and Checkout_Date from the Booking table
  SELECT Checkin_Date, Checkout_Date INTO initialCheckinDate, initialCheckoutDate
  FROM Booking
  WHERE Booking_ID = NEW.Booking_ID;

  -- Determine the booking status based on the change
  IF NEW.Change_Booking_Dates IS NOT NULL THEN
    SET bookingStatus = 'Date change';
    -- Update Checkin_Date and Checkout_Date in Booking table
    UPDATE Booking
    SET Checkin_Date = NEW.Change_Booking_Dates,
      Checkout_Date = DATE_ADD(NEW.Change_Booking_Dates, INTERVAL
DATEDIFF(initialCheckoutDate, initialCheckinDate) DAY)
    WHERE Booking_ID = NEW.Booking_ID;
  ELSE
    SET bookingStatus = 'Cancelled';
    -- Mark Current_Active as 0 in Booking table
```

```
        UPDATE Booking
        SET Current_Active = 0
        WHERE Booking_ID = NEW.Booking_ID;
    END IF;

    -- Insert into State_Transition based on the type of change
    INSERT INTO State_Transition (Booking_ID, old_checkin_date, old_checkout_date,
Status)
    VALUES (NEW.Booking_ID, initialCheckinDate, initialCheckoutDate, bookingStatus);
END //

DELIMITER ;
```

FUNCTIONS

Function 1: Check rooms booked for a hotel on a particular date

```
DELIMITER //

CREATE FUNCTION GetTotalRoomsBooked(
  p_HotelID INT,
  p_Date DATE
)
RETURNS INT
READS SQL DATA
BEGIN
  DECLARE totalRoomsBooked INT;

  SELECT COUNT(*) INTO totalRoomsBooked
  FROM RoomAvailability
  WHERE Hotel_ID = p_HotelID
  AND Date = p_Date
  AND Available = 0;

  RETURN totalRoomsBooked;
END //

DELIMITER ;
```

Output:

```
SELECT  GetTotalRoomsBooked(53,  "2025-02-14");

# GetTotalRoomsBooked(53, "2025-02-14") 2
```

Description:

GetTotalRoomsBooked:

- This function calculates the total number of rooms booked for a specified hotel on a given date.
- Parameters:
  - **p_HotelID INT**: The ID of the hotel for which the rooms are being checked.
  - **p_Date DATE**: The specific date for which the rooms are being checked.
- Method: It counts the number of rows in the **RoomAvailability**table where the **Hotel_ID**matches the provided **p_HotelID**, the **Date**matches the provided **p_Date**, and the **Available**column is set to **0**(indicating that the room is not available/booked).

Function 2: Calculate Total Amount Spent by a Customer:

```
DELIMITER //

CREATE FUNCTION GetTotalAmountSpentByCustomer(
   p_CustomerID INT
)
RETURNS DECIMAL(10, 2)
READS SQL DATA
BEGIN
   DECLARE totalAmount DECIMAL(10, 2);

   SELECT SUM(Booking_amount) INTO totalAmount
   FROM Booking
   WHERE Cust_ID = p_CustomerID;

   RETURN totalAmount;
END //

DELIMITER ;
```

Output: SELECT GetTotalAmountSpentByCustomer(5);

```
# GetTotalAmountSpentByCustomer(5)
1795.00
```

Description:

GetTotalAmountSpentByCustomer:

- This function calculates the total amount spent by a customer based on their customer ID.
- Parameter:
  - **p_CustomerID INT**: The ID of the customer for whom the total amount spent is being calculated.
- Method: It sums up the **Booking_amount**from the **Booking**table where **Cust_ID**matches the provided **p_CustomerID**.

Function 3: Retrieve Employee's Position:

```
DELIMITER //

CREATE FUNCTION GetEmployeePosition(
  p_EmployeeID INT
)
RETURNS VARCHAR(50)
READS SQL DATA
BEGIN
  DECLARE employeePosition VARCHAR(50);

  SELECT Emp_Position INTO employeePosition
  FROM Employee_Info
  WHERE Emp_ID = p_EmployeeID;

  RETURN employeePosition;
END //

DELIMITER ;
```

Output: SELECT GetEmployeePosition(11);

```
# GetEmployeePosition(11)
Staff
```

Description:

GetEmployeePosition:

- This function retrieves the position of an employee based on their employee ID.
- Parameter:
  - **p_EmployeeID INT**: The ID of the employee for whom the position is being retrieved.
- Method: It retrieves the **Emp_Position**from the **Employee_Info**table where **Emp_ID** matches the provided **p_EmployeeID.**

Function 4: Calculate Total Occupancy Percentage of a Hotel:

```
  DELIMITER //

CREATE FUNCTION GetHotelOccupancyPercentage(
   p_HotelID INT
)
RETURNS DECIMAL(5, 2)
READS SQL DATA
BEGIN
   DECLARE totalRooms INT;
   DECLARE occupiedRooms INT;
   DECLARE occupancyPercentage DECIMAL(5, 2);

   SELECT COUNT(*) INTO totalRooms
   FROM Room
   WHERE Hotel_ID = p_HotelID;

   SELECT COUNT(*) INTO occupiedRooms
   FROM RoomAvailability
   WHERE Hotel_ID = p_HotelID
   AND Available = 0;

   IF totalRooms = 0 THEN
      SET occupancyPercentage = 0;
   ELSE
      SET occupancyPercentage = (occupiedRooms / totalRooms) * 100;
   END IF;

   RETURN occupancyPercentage;
END //

DELIMITER ;
```

Output: SELECT GetHotelOccupancyPercentage(14);

Discription:

GetHotelOccupancyPercentage:

- This function calculates the occupancy percentage of a hotel based on its hotel ID.
- Parameter:
  - **p_HotelID INT**: The ID of the hotel for which the occupancy percentage is being calculated.
- Method: It calculates the total number of rooms and occupied rooms for the hotel, then computes the occupancy percentage based on these values.

Fucntion 5: Calculate Average Room Price in a Hotel:

```
DELIMITER //

CREATE FUNCTION GetAverageRoomPriceInHotel(
   p_HotelID INT
)
RETURNS DECIMAL(10, 2)
READS SQL DATA
BEGIN
   DECLARE avgRoomPrice DECIMAL(10, 2);

   SELECT AVG(Room_Price) INTO avgRoomPrice
   FROM Room
   WHERE Hotel_ID = p_HotelID;

   RETURN avgRoomPrice;
END //

DELIMITER ;
```

Output: SELECT GetAverageRoomPriceInHotel(14);

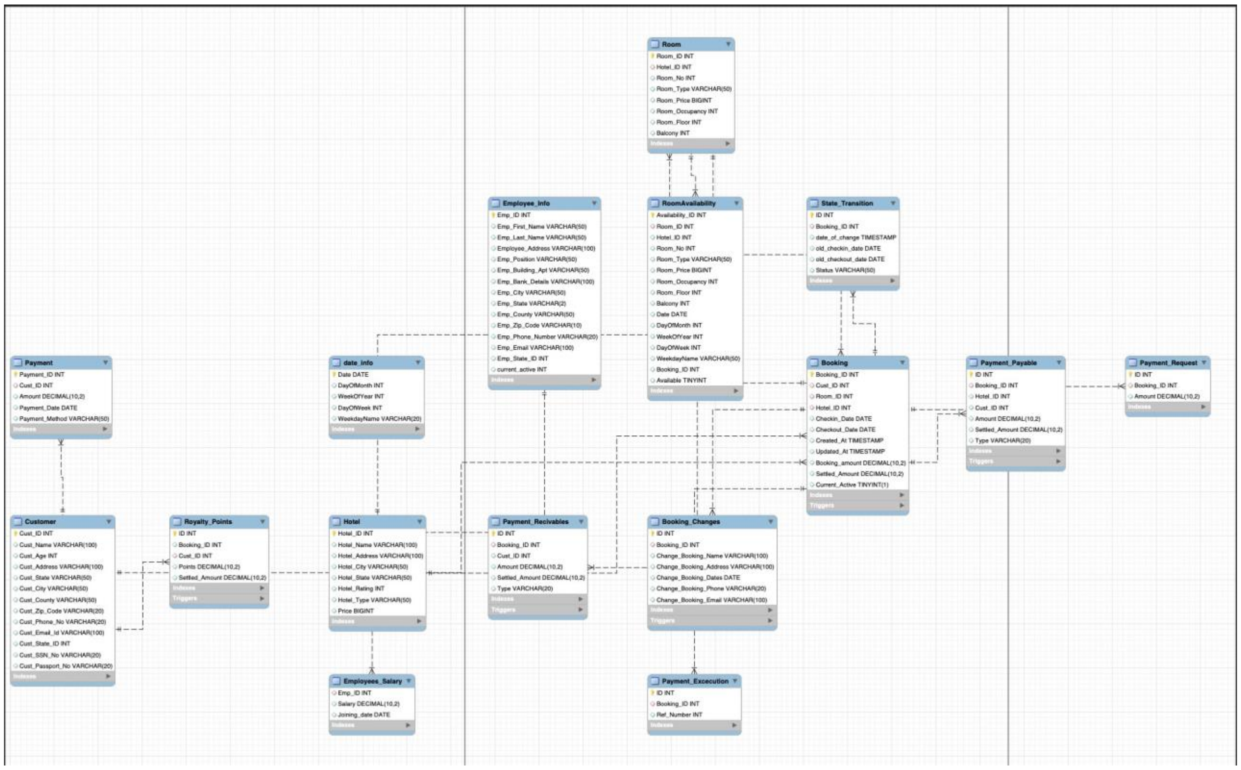
120.00

Description:

GetAverageRoomPriceInHotel:

- This function calculates the average room price in a hotel based on its hotel ID.
- Parameter:
- **p_HotelID INT**: The ID of the hotel for which the average room price is being calculated.
- Method: It calculates the average of the **Room_Price**column from the **Room**table where **Hotel_ID**matches the provided **p_HotelID**.

ERR Diagram


Room
Room_ID INT
Hotel_ID INT
Room_No INT
Room_Type VARCHAR(50)
Room_Price BIGINT
Room_Occupancy INT
Room_Floor INT
Balcony INT
Employee_Info
Emp_ID INT
Emp_First_Name VARCHAR(50)
Emp_Last_Name VARCHAR(50)
Employee_Address VARCHAR(100)
Emp_Position VARCHAR(50)
Emp_Building_Apt VARCHAR(50)
Emp_Bank_Details VARCHAR(100)
Emp_City VARCHAR(50)
Emp_State VARCHAR(2)
Emp_County VARCHAR(50)
Emp_Zip_Code VARCHAR(10)
Emp_Phone_Number VARCHAR(20)
Emp_Email VARCHAR(100)
Emp_State_ID INT
current_active INT
RoomAvailability
Availability_ID INT
Room_ID INT
Hotel_ID INT
Room_No INT
Room_Type VARCHAR(50)
Room_Price BIGINT
Room_Occupancy INT
Room_Floor INT
Balcony INT
Date DATE
DayOfMonth INT
WeekOfYear INT
DayOfWeek INT
WeekdayName VARCHAR(50)
Booking_ID INT
Available TINYINT
State_Transition
ID INT
Booking_ID INT
date_of_change TIMESTAMP
old_checkin_date DATE
old_checkout_date DATE
Status VARCHAR(50)
Payment
Payment_ID INT
Cust_ID INT
Amount DECIMAL(10,2)
Payment_Date DATE
Payment_Method VARCHAR(50)
date_info
Date DATE
DayOfMonth INT
WeekOfYear INT
DayOfWeek INT
WeekdayName VARCHAR(20)
Booking
Booking_ID INT
Cust_ID INT
Room_ID INT
Hotel_ID INT
Checkin_Date DATE
Checkout_Date DATE
Created_At TIMESTAMP
Updated_At TIMESTAMP
Booking_amount DECIMAL(10,2)
Settled_Amount DECIMAL(10,2)
Current_Active TINYINT(1)
Payment_Payable
ID INT
Booking_ID INT
Hotel_ID INT
Cust_ID INT
Amount DECIMAL(10,2)
Settled_Amount DECIMAL(10,2)
Type VARCHAR(20)
Payment_Request
ID INT
Booking_ID INT
Amount DECIMAL(10,2)
Customer
Cust_ID INT
Cust_Name VARCHAR(100)
Cust_Age INT
Cust_Address VARCHAR(100)
Cust_State VARCHAR(50)
Cust_City VARCHAR(50)
Cust_County VARCHAR(50)
Cust_Zip_Code VARCHAR(20)
Cust_Phone_No VARCHAR(20)
Cust_Email_Id VARCHAR(100)
Cust_State_ID INT
Cust_SSN_No VARCHAR(20)
Cust_Passport_No VARCHAR(20)
Royalty_Points
ID INT
Booking_ID INT
Cust_ID INT
Points DECIMAL(10,2)
Settled_Amount DECIMAL(10,2)
Hotel
Hotel_ID INT
Hotel_Name VARCHAR(100)
Hotel_Address VARCHAR(100)
Hotel_City VARCHAR(50)
Hotel_State VARCHAR(50)
Hotel_Rating INT
Hotel_Type VARCHAR(50)
Price BIGINT
Payment_Recivables
ID INT
Booking_ID INT
Cust_ID INT
Amount DECIMAL(10,2)
Settled_Amount DECIMAL(10,2)
Type VARCHAR(20)
Booking_Changes
ID INT
Booking_ID INT
Change_Booking_Name VARCHAR(100)
Change_Booking_Address VARCHAR(100)
Change_Booking_Dates DATE
Change_Booking_Phone VARCHAR(20)
Change_Booking_Email VARCHAR(100)
Employees_Salary
Emp_ID INT
Salary DECIMAL(10,2)
Joining_date DATE
Payment_Excecution
ID INT
Booking_ID INT
Ref_Number INT
Indexes
Triggers