Tristan Garner

Databases

Mr. Ivers

4/11/23

Hotel Reservation Database

**Database Design Overview:**

A diagram of a business

Description automatically generated with medium confidence

**Summary:** This is a small database created to represent what a hotel may use to support their reservation services. It contains tables representing Customers, Rooms, Bookings, Payments, Services, and a junction table named Booking\_Services to allow the hotel to store key information that is useful when running a hotel and reserving rooms. The following sections of this report will detail key information in utilizing this database properly and efficiently:

* Create Commands for each table.
* Views for easy data visualization.
* Example Commands for common use-case scenarios.
* Example Data used to mimic real world data.
* Indexes for each table for faster use-cases.
* Creation of Stored Procedures, Functions, Transactions, Triggers to further demonstrate the use case of the database in real-world scenarios.
* User Creation and Permissions.
* Overall Database Design Diagram

**Table Creation Commands:**

CREATE TABLE Customers (

CustomerID INT AUTO\_INCREMENT,

FirstName VARCHAR(20),

LastName VARCHAR(20),

Email VARCHAR(40),

PhoneNumber VARCHAR(10),

Address TEXT,

PRIMARY KEY (CustomerID)

);

CREATE TABLE Rooms (

RoomID INT AUTO\_INCREMENT,

RoomNumber VARCHAR(10),

RoomType VARCHAR(20),

PricePerNight DECIMAL(10,2),

MaxOccupancy INT,

CurrentStatus VARCHAR(20),

PRIMARY KEY (RoomID)

);

CREATE TABLE Bookings (

BookingID INT AUTO\_INCREMENT,

CustomerID INT,

RoomID INT,

CheckInDate DATE,

CheckOutDate DATE,

NumberOfGuests INT,

BookingStatus VARCHAR(20),

PRIMARY KEY (BookingID),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID)

);

CREATE TABLE Payments (

PaymentID INT AUTO\_INCREMENT,

BookingID INT,

PaymentDate DATE,

PaymentMethod VARCHAR(20),

AmountPaid DECIMAL(10,2),

PRIMARY KEY (PaymentID),

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)

);

CREATE TABLE Services (

ServiceID INT AUTO\_INCREMENT,

ServiceName VARCHAR(50),

Description TEXT,

Price DECIMAL(10,2),

PRIMARY KEY (ServiceID)

);

CREATE TABLE Booking\_Services (

BookingServiceID INT AUTO\_INCREMENT,

BookingID INT,

ServiceID INT,

Quantity INT,

ServiceDate DATE,

PRIMARY KEY (BookingServiceID),

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID),

FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID)

);

**Two View Creations:**

1. Current Bookings View - This view is used to see what rooms are currently booked in the hotel and further information about the customer and rooms.

CREATE VIEW `CurrentBookingsView` AS

SELECT

Bookings.BookingID,

CONCAT(Customers.FirstName, ' ', Customers.LastName) AS CustomerName,

Customers.Email,

Rooms.RoomNumber,

Rooms.RoomType,

Bookings.CheckInDate,

Bookings.CheckOutDate,

Bookings.BookingStatus

FROM Bookings

JOIN Customers ON Bookings.CustomerID = Customers.CustomerID

JOIN Rooms ON Bookings.RoomID = Rooms.RoomID

WHERE Bookings.BookingStatus in ('Confirmed', 'Checked-In');

1. This second view is to see the current available rooms within the hotel and more detailed information about the room such as occupancy, price and so on.

CREATE VIEW `AvailableRoomsView` AS

SELECT

Rooms.RoomID,

Rooms.RoomNumber,

Rooms.RoomType,

Rooms.PricePerNight,

Rooms.MaxOccupancy,

Rooms.CurrentStatus

FROM Rooms

WHERE NOT EXISTS (

SELECT \*

FROM Bookings

WHERE Bookings.RoomID = Rooms.RoomID AND Bookings.BookingStatus IN ('Confirmed', 'Checked-In')

) AND Rooms.CurrentStatus = 'Available';

**Example Commands for Each Table:**

**\*Customers Table\***

* **SELECT COMMANDS**
  + **Join Command:**

SELECT Customers.FirstName, Customers.LastName, Bookings.CheckInDate

FROM Customers

JOIN Bookings ON Customers.CustomerID = Bookings.CustomerID

WHERE Bookings.CheckInDate >= CURDATE();

* + **Aggregate Function:**

SELECT COUNT(\*) AS NumberOfCustomers

FROM Customers;

* + **Group-By/Having Command:**

SELECT COUNT(CustomerID) AS ReservationsCount, CustomerID

FROM Bookings

GROUP BY CustomerID

HAVING COUNT(CustomerID) > 1;

* + **Order-By Command:**

SELECT FirstName, LastName, Email

FROM Customers

ORDER BY LastName ASC;

* + **Like Command:**

SELECT \*

FROM Customers

WHERE Email LIKE '%gmail.com';

* **Insert Command:**

INSERT INTO Customers (FirstName, LastName, Email, PhoneNumber, Address)

VALUES ('John', 'Doe', 'john.doe@gmail.com', '1234567890', '123 Main St');

* **Two Update Commands:**

UPDATE Customers

SET Email = 'john.d.updated@gmail.com'

WHERE CustomerID = 1;

UPDATE Customers

SET Address = '456 Elm St'

WHERE LastName = 'Doe';

* **Delete Command:**

DELETE FROM Customers

WHERE CustomerID = 1;

* **Alter Command:**

ALTER TABLE Customers

ADD COLUMN MembershipStatus VARCHAR(20) DEFAULT 'Standard';

**\*Bookings Table\***

* **SELECT COMMANDS**
  + **Join Command:**

SELECT Bookings.BookingID, Rooms.RoomType, Customers.FirstName, Customers.LastName

FROM Bookings

JOIN Rooms ON Bookings.RoomID = Rooms.RoomID

JOIN Customers ON Bookings.CustomerID = Customers.CustomerID

WHERE Bookings.CheckOutDate < CURDATE();

* + **Aggregate Function:**

SELECT RoomID, COUNT(\*) AS TotalBookings

FROM Bookings;

* + **Group-By/Having Command:**

SELECT RoomID, COUNT(\*) AS BookingCount

FROM Bookings

GROUP BY CustomerID

HAVING COUNT(\*) > 5;

* + **Order-By Command:**

SELECT BookingID, CheckInDate, CheckOutDate

FROM Bookings

ORDER BY CheckInDate DESC;

* + **Like Command:**

SELECT \*

FROM Bookings

WHERE BookingStatus LIKE 'Confirmed';

* **Insert Command:**

INSERT INTO Bookings (CustomerID, RoomID, CheckInDate, CheckOutDate, NumberOfGuests, BookingStatus)

VALUES (1, 101, '2024-04-01', '2024-04-05', 2, 'Confirmed');

* **Two Update Commands:**

UPDATE Bookings

SET NumberOfGuests = 3

WHERE BookingID = 1;

UPDATE Bookings

SET BookingStatus = 'Cancelled'

WHERE BookingID = 2;

* **Delete Command:**

DELETE FROM Bookings

WHERE BookingID = 3;

* **Alter Command:**

ALTER TABLE Bookings

ADD COLUMN SpecialRequests TEXT;

* **Single Subquery Example for Database:** Finds Rooms that have never been booked.

SELECT Rooms.RoomID, Rooms.RoomType

FROM Rooms

WHERE Rooms.RoomID NOT IN (

SELECT RoomID

FROM Bookings);

**\*Rooms Table\***

* **SELECT COMMANDS**
  + **Join Command:**

SELECT Rooms.RoomNumber, Rooms.RoomType, Bookings.CheckInDate, Bookings.CheckOutDate

FROM Rooms

JOIN Bookings ON Rooms.RoomID = Bookings.RoomID

WHERE Rooms.CurrentStatus = 'Booked';

* + **Aggregate Function:**

SELECT RoomType, COUNT(\*) AS NumberOfRooms

FROM Rooms

GROUP BY RoomType;

* + **Group-By/Having Command:**

SELECT RoomType, AVG(PricePerNight) AS AveragePrice

FROM Rooms

GROUP BY RoomType

HAVING AVG(PricePerNight) > 100;

* + **Order-By Command:**

SELECT RoomNumber, PricePerNight

FROM Rooms

ORDER BY PricePerNight DESC;

* + **Like Command:**

SELECT \*

FROM Rooms

WHERE RoomType LIKE '%Double%';

* **Insert Command:**

INSERT INTO Rooms (RoomNumber, RoomType, PricePerNight, MaxOccupancy, CurrentStatus)

VALUES ('101', 'Single', 80.00, 2, 'Available');

* **Two Update Commands:**

UPDATE Rooms

SET CurrentStatus = 'Maintenance'

WHERE RoomID = 1;

UPDATE Rooms

SET PricePerNight = PricePerNight \* 1.1

WHERE RoomType = 'Double';

* **Delete Command:**

DELETE FROM Rooms

WHERE RoomID = 1;

* **Alter Command:**

ALTER TABLE Rooms

ADD COLUMN Amenities TEXT;

**\*Payments Table\***

* **SELECT COMMANDS**
  + **Join Command:**

SELECT Payments.PaymentID, Payments.AmountPaid, Customers.FirstName, Customers.LastName

FROM Payments

JOIN Bookings ON Payments.BookingID = Bookings.BookingID

JOIN Customers ON Bookings.CustomerID = Customers.CustomerID;

* + **Aggregate Function:**

SELECT SUM(AmountPaid) AS TotalRevenue

FROM Payments;

* + **Group-By/Having Command:**

SELECT BookingID, COUNT(\*) AS NumberOfPayments

FROM Payments

GROUP BY BookingID

HAVING COUNT(\*) > 1;

* + **Order-By Command:**

SELECT PaymentID, PaymentDate, AmountPaid

FROM Payments

ORDER BY PaymentDate;

* + **Like Command:**

SELECT \*

FROM Payments

WHERE PaymentMethod LIKE 'Credit%';

* **Insert Command:**

INSERT INTO Payments (BookingID, PaymentDate, PaymentMethod, AmountPaid)

VALUES (1, CURDATE(), 'Credit Card', 200.00);

* **Two Update Commands:**

UPDATE Payments

SET AmountPaid = 250.00

WHERE PaymentID = 1;

UPDATE Payments

SET PaymentMethod = 'Debit Card'

WHERE PaymentID = 2;

* **Delete Command:**

DELETE FROM Payments

WHERE PaymentID = 3;

* **Alter Command:**

ALTER TABLE Payments

ADD COLUMN ConfirmationNumber VARCHAR(20);

**\*Services Table\***

* **SELECT COMMANDS**
  + **Join Command:**

SELECT Services.ServiceName, Services.Description, Booking\_Services.Quantity

FROM Services

JOIN Booking\_Services ON Services.ServiceID = Booking\_Services.ServiceID

WHERE Services.Price > 50;

* + **Aggregate Function:**

SELECT COUNT(\*) AS TotalServices

FROM Services;

* + **Group-By/Having Command:**

SELECT ServiceName, COUNT(\*) AS TimesRequested

FROM Services

JOIN Booking\_Services ON Services.ServiceID = Booking\_Services.ServiceID

GROUP BY ServiceName

HAVING COUNT(\*) > 10;

* + **Order-By Command:**

SELECT ServiceName, Price

FROM Services

ORDER BY Price DESC;

* + **Like Command:**

SELECT \*

FROM Services

WHERE Description LIKE '%massage%';

* **Insert Command:**

INSERT INTO Services (ServiceName, Description, Price)

VALUES ('Gym', 'Access to the full size gym and equipment', 15.00);

* **Two Update Commands:**

UPDATE Services

SET Price = 100.00

WHERE ServiceID = 1;

UPDATE Services

SET Description = 'Deep Tissue Massage'

WHERE ServiceName = 'Massage';

* **Delete Command:**

DELETE FROM Services

WHERE ServiceID = 1;

* **Alter Command:**

ALTER TABLE Services

ADD COLUMN Available BOOLEAN DEFAULT TRUE;

**\*Booking\_Services Table\***

* **SELECT COMMANDS**
  + **Join Command:**

SELECT Booking\_Services.BookingID, Services.ServiceName, Booking\_Services.Quantity

FROM Booking\_Services

JOIN Services ON Booking\_Services.ServiceID = Services.ServiceID;

* + **Aggregate Function:**

SELECT ServiceID, SUM(Quantity) AS TotalQuantityOrdered

FROM Booking\_Services

GROUP BY ServiceID;

* + **Group-By/Having Command:**

SELECT BookingID, COUNT(\*) AS NumberOfServicesBooked

FROM Booking\_Services

GROUP BY BookingID

HAVING COUNT(\*) > 1;

* + **Order-By Command:**

SELECT BookingID, ServiceID, Quantity

FROM Booking\_Services

ORDER BY Quantity DESC;

* + **Like Command: Does Not Really Fit for this Table**

SELECT \*

FROM Booking\_Services

WHERE Description LIKE '%5%';

* **Insert Command:**

INSERT INTO Booking\_Services (BookingID, ServiceID, Quantity, ServiceDate)

VALUES (1, 2, 2, '2024-04-03');

* **Two Update Commands:**

UPDATE Booking\_Services

SET Quantity = 3

WHERE BookingServiceID = 1;

UPDATE Booking\_Services

SET ServiceDate = '2024-04-04'

WHERE BookingServiceID = 2;

* **Delete Command:**

DELETE FROM Booking\_Services

WHERE BookingServiceID = 3;

* **Alter Command:**

ALTER TABLE Booking\_Services

ADD COLUMN Time VARCHAR(20) DEFAULT '60 Min';

**Index Creation Per Table:**

* Customers Table:
  + CREATE INDEX idx\_customers\_name ON Customers(LastName, FirstName);
* Rooms Table:
  + CREATE INDEX idx\_rooms\_currentstatus ON Rooms(RoomNumber, CurrentStatus);
* Bookings Table:
  + CREATE INDEX idx\_bookings\_bookingstatus ON Bookings(BookingID, BookingStatus);
* Payments Table:
  + CREATE INDEX idx\_payments\_bookingid ON Payments(BookingID);
* Services Table:
  + CREATE INDEX idx\_services\_price ON Services(Price);
* Booking Services Table:
  + CREATE INDEX idx\_booking\_services\_bookingid ON Booking\_Services(BookingID);

**Stored Procedures (3):**

1. Procedure that adds new customers to the database:

DELIMITER //

CREATE PROCEDURE AddCustomer(IN firstName VARCHAR(255), IN lastName VARCHAR(255), IN email VARCHAR(255), IN phoneNumber VARCHAR(20), IN address TEXT)

BEGIN

INSERT INTO Customers (FirstName, LastName, Email, PhoneNumber, Address)

VALUES (firstName, lastName, email, phoneNumber, address);

END //

DELIMITER ;

\*EXAMPLE: CALL AddCustomer('Tristan', 'Garner', 'tg@gmail.com', '123-4567', '123 South St');

1. Procedure that allows users to update bookings status in database:  
   DELIMITER //

CREATE PROCEDURE UpdateBookingStatus(IN bookingID INT, IN newStatus VARCHAR(50))

BEGIN

UPDATE Bookings

SET BookingStatus = newStatus

WHERE BookingID = bookingID;

END //

DELIMITER ;

\*Example: CALL UpdateBookingStatus(101, 'Checked-In');

1. This procedure books a room with all of the necessary customer and room information instead of just updating it’s status like the previous procedure:

DELIMITER //

CREATE PROCEDURE BookARoom(IN custID INT, IN rmID INT, IN checkIn DATE, IN checkOut DATE, IN guests INT)

BEGIN

INSERT INTO Bookings (CustomerID, RoomID, CheckInDate, CheckOutDate, NumberOfGuests, BookingStatus)

VALUES (custID, rmID, checkIn, checkOut, guests, 'Confirmed');

UPDATE Rooms

SET CurrentStatus = 'Booked'

WHERE RoomID = rmID;

END //

DELIMITER ;

\*Example: CALL BookARoom(1, 102, '2024-04-20', '2024-04-25', 2);

**Functions (3):**

1. Function that basically acts as a calculator used for determining the total cost for a customers stay (Only for room and amount of nights, services not included.)

DELIMITER //

CREATE FUNCTION TotalStayCost(nights INT, pricePerNight DECIMAL(10,2))

RETURNS DECIMAL(10,2)

BEGIN

RETURN nights \* pricePerNight;

END //

DELIMITER ;

\*Example: SELECT TotalStayCost(5, 100.00);

1. The booking duration function, while trivial, allows employees to see how many days between two dates so that booking determinations can be made.

DELIMITER //

CREATE FUNCTION BookingDuration(checkIn DATE, checkOut DATE)

RETURNS INT

BEGIN

RETURN DATEDIFF(checkOut, checkIn);

END //

DELIMITER ;

\*Example: SELECT BookingDuration('2024-04-01', '2024-04-10');

1. The last function is a simple function that allows hotel employees to easily calculate a guests bill with a discount. While this could also just be done with a calculator, it would be easier to just use the system as an all in one and allow it to calculate. This is in tandem with the fact that other database entities could be used as well if needed later.

DELIMITER //

CREATE FUNCTION CalculateDiscount(totalAmount DECIMAL(10,2), discountRate DECIMAL(5,2))

RETURNS DECIMAL(10,2)

BEGIN

RETURN (totalAmount \* discountRate) / 100;

END //

DELIMITER ;

\*Example: SELECT CalculateDiscount(200.00, 15);

**Transactions (3):**

1. Transaction that inserts a new booking into the bookings table, and updates the rooms table’s CurrentStatus to ‘Booked’

START TRANSACTION;

INSERT INTO Bookings (CustomerID, RoomID, CheckInDate, CheckOutDate, NumberOfGuests, BookingStatus)

VALUES (1, 101, '2024-04-20', '2024-04-25', 2, 'Confirmed');

UPDATE Rooms

SET CurrentStatus = 'Booked'

WHERE RoomID = 101 AND CurrentStatus = 'Available';

IF ROW\_COUNT() = 0 THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Error: Room is not available.';

ELSE

COMMIT;

END IF;

1. Transaction that handles taking in payments for bookings and updates the rooms availability.

START TRANSACTION;

INSERT INTO Payments (BookingID, PaymentDate, PaymentMethod, AmountPaid)

VALUES (1, CURDATE(), 'Credit Card', 200.00);

UPDATE Bookings

SET BookingStatus = 'Paid'

WHERE BookingID = 1 AND BookingStatus != 'Paid';

IF ROW\_COUNT() = 0 THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Error: Booking already paid or does not exist.';

ELSE

COMMIT;

END IF;

1. This final transaction handles room cancellations by updating the room status back to available and marking the booking status to ‘cancelled;

START TRANSACTION;

UPDATE Bookings

SET BookingStatus = 'Canceled'

WHERE BookingID = 1 AND BookingStatus != 'Canceled';

IF ROW\_COUNT() = 0 THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Error: Booking already canceled or does not exist.';

ELSE

UPDATE Rooms

SET CurrentStatus = 'Available'

WHERE BookingID = 1

);

COMMIT;

END IF;

**Triggers (2):**

1. The first trigger example is one that will automatically update the ‘CurrentStatus’ of a room to ‘Booked’ when a new booking is inserted into the ‘Bookings’ table.

DELIMITER //

CREATE TRIGGER AfterBookingInsert

AFTER INSERT ON Bookings

FOR EACH ROW

BEGIN

IF NEW.BookingStatus = 'Confirmed' THEN

UPDATE Rooms

SET CurrentStatus = 'Booked'

WHERE RoomID = NEW.RoomID;

END IF;

END //

DELIMITER ;

1. The next trigger will update the ‘CurrentStatus’ of a room back to ‘Available’ when a booking’s status is updated to ‘Cancelled’.

DELIMITER //

CREATE TRIGGER AfterBookingUpdate

AFTER UPDATE ON Bookings

FOR EACH ROW

BEGIN

IF OLD.BookingStatus != 'Cancelled' AND NEW.BookingStatus = 'Cancelled' THEN

UPDATE Rooms

SET CurrentStatus = 'Available'

WHERE RoomID = NEW.RoomID;

END IF;

END //

DELIMITER ;

**Creating Users and Managing Permissions:**

1. Creating Users:
   * CREATE USER ‘TGarner’@’localhost’ IDENTIFIED BY ‘mypassword’;
   * CREATE USER ‘MMuhsen’@’localhost’ IDENTIFIED BY ‘mypassword’;
2. Granting Privileges to Views:
   * GRANT SELECT ON hotel\_reservation\_db.currentbookingsview TO 'TGarner'@'localhost', 'MMuhsen'@'localhost';
   * GRANT SELECT ON hotel\_reservation\_db.availableroomsview TO 'TGarner'@'localhost', 'MMuhsen'@'localhost';
3. Denying Permissions to Two Tables:
   * REVOKE ALL PRIVILEGES ON hotel\_reservation\_db.customers FROM 'TGarner'@'localhost', 'MMuhsen'@'localhost';
   * REVOKE ALL PRIVILEGES ON hotel\_reservation\_db.rooms FROM 'TGarner'@'localhost', 'MMuhsen'@'localhost';

**Hotel Reservation Database Final Diagram:**

A screenshot of a computer

Description automatically generated