MOVIE TICKET BOOKING SYSTEM

SANIYA D (15/11/2025)

MOVIE TICKET BOOKING SYSTEM

Description:

This project is designed to manage and automate the process of booking movie tickets using SQL.
It stores and handles data related to movies, customers, bookings, and payments efficiently.
The system allows users to view available movies, make bookings, and track payments in an organized way.
It demonstrates the use of SQL concepts like **DDL, DML, DQL, Joins, Grouping, Constraints, and Subqueries** in a real-world scenario.

Objectives:

- Track movie bookings per customer
- Calculate total earnings per movie
- Manage payments and receipts
- Generate reports for management

Tables in the Database:

1. Movies**:** Stores details of all movies such as title, type, director, duration, and language.
2. Customers**:** Contains customer information like name, email, and contact number.
3. Bookings**:** Records all movie ticket bookings made by customers.
4. Payments**:** Stores payment details related to each booking, including method and amount.

QUERIES:

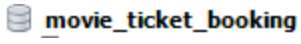
Create Database

Question: How to create a new database for the project?

Input:

```
# creating database
Create database Movie_Ticket_Booking;
```

Output:



Insight: By creating this database, I've built the base for my entire project.

Create Table

Question: Why did we create different tables in the Movie Ticket Booking database?

Input:

```
-- Table 1: Movies - stores details of each movie
    CREATE TABLE Movies (
    Movie_id INT PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    Movie_Type VARCHAR(50) NOT NULL,
    Duration_min INT CHECK(duration_min > 0), Release_date DATE );

-- Table 2: Customers - stores details of people who book tickets
    CREATE TABLE Customers ( Customer_id INT PRIMARY KEY,
    Customer_name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) unique NOT NULL,
    Contact BIGINT UNIQUE);

-- Table 3: Bookings - stores ticket booking information
    CREATE TABLE Bookings (
    Booking_id INT PRIMARY KEY,
    Movie_id INT NOT NULL,
    Customer_id INT NOT NULL,
    Seats INT NOT NULL CHECK(seats > 0),
    Booking_date DATE DEFAULT (CURRENT_DATE),
    total_amount DECIMAL(10,2) CHECK(total_amount >= 0),
    FOREIGN KEY (movie_id) REFERENCES Movies(movie_id),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id));

-- Table 4: Payments - stores details of payments
    CREATE TABLE Payments (
    Payment_id INT PRIMARY KEY,
    Booking_id INT NOT NULL,
    Payment_method VARCHAR(50) NOT NULL,
    Payment_date DATE DEFAULT (CURRENT_DATE),
    Amount DECIMAL(10,2) CHECK (Amount >= 0),
    FOREIGN KEY (Booking_id) REFERENCES Bookings(Booking_id));
```

Insight: Creating these tables gave proper structure to my project. Each table stores a different type of data — movies, users, their bookings, and payments. It feels like dividing the whole system into clear parts so everything stays organized and easy to manage later.

Alter Table Movies (Add Column)

Question: Why did I add new columns “Director” and “Language” to the Movies table?

Input:

```
ALTER TABLE Movies ADD COLUMN  Director VARCHAR(100) NOT NULL AFTER Movie_Type;
ALTER TABLE Movies ADD COLUMN Language VARCHAR(100) NOT NULL;
```

Insight: Adding these columns made my movie data more detailed and real.

Now, every movie record can show who directed it and in which language it was released.

Inserting Values in Tables:

Question: Why did I insert data into all the tables?

Input:

```
# Inserting values in table 1: MOVIES
    INSERT INTO Movies VALUES
(1, "Pathaan" , "Action" , "Siddharth Anand" , 146 , '2023-01-25' , "Hindi" ),
(2, "KGF: Chapter 2" , "Action" , "Prashanth Neel" , 168 , '2022-04-14' , "Kannada" ),
(3, "3 Idiots" , "Drama" , "Rajkumar Hirani" , 170 , '2009-12-25' , "Hindi" ),
(4, "Bahubali: The Beginning" , "Fantasy" , "S. S. Rajamouli" , 159 , '2015-07-10' , "Telugu" ),
(5, "Bahubali: The Conclusion" , "Fantasy" , "S. S. Rajamouli" , 171 , '2017-04-28' , "Telugu" ),
(6, "Dangal" , "Biography" , "Nitesh Tiwari" , 161 , '2016-12-23' , "Hindi" ),
(7, "Chhichhore" ,  "Drama" , "Nitesh Tiwari" , 143 , '2019-09-06' , "Hindi" ),
(8, "Tanhaji" , "Historical" , "Om Raut" , 135 , '2020-01-10' , "Hindi" ),
(9, "Interstellar" , "Sci-Fi" , "Christopher Nolan" , 169 , '2014-11-07' , "English" ),
(10, "Inception" ,  "Sci-Fi" , "Christopher Nolan" , 148 , '2010-07-16' , "English" ),
(11, "The Dark Knight" , "Action" , "Christopher Nolan" , 152 , '2008-07-18' , "English" ),
(12, "RRR" , "Action" , "S. S. Rajamouli" , 187 , '2022-03-25' , "Telugu" ),
(13, "Pushpa: The Rise" , "Action" , "Bandreddi Sukumar" , 179 , '2021-12-17' , "Telugu" ),
(14, "Jawaan" , "Action" , "Atlee Kumar" , 169 , '2023-09-07' , "Hindi" ),
(15, "Zindagi Na Milegi Dobara" , "Adventure" , "Zoya Akhtar" , 155 , '2011-07-15' , "Hindi" ),
(16, "Barfi!" , "Romance" , "Anurag Basu" , 151 , '2012-09-14' , "Hindi" ),
(17, "Dil Bechara" , "Romance" , "Mukesh Chhabra" , 101 , '2020-07-24' , "Hindi" ),
(18, "Drishyam 2" , "Thriller" , "Abhishek Pathak" , 140 , '2022-11-18' , "Hindi" ),
(19, "Vikram Vedha" , "Thriller" , "Pushkar-Gayathri" , 147 , '2022-09-30' , "Hindi" ),
(20, "Kantara" , "Drama" , "Rishab Shetty" , 147 , '2022-09-30' , "Kannada" ),
(21, "Super 30" , "Biography" , "Vikas Bahl" , 155 , '2019-07-12' , "Hindi" ),
(22, "Avatar: The Way of Water" , "Sci-Fi" , "James Cameron" , 192 , '2022-12-16' , "English" ),
(23, "Titanic" , "Romance" , "James Cameron" , 195 , '1997-12-19' , "English" ),
(24, "Andhadhun" , "Thriller" , "Sriram Raghavan" , 139 , '2018-10-05' , "Hindi" ),
(25, "Dunki" , "Comedy" , "Rajkumar Hirani" , 160 , '2024-12-22' , "Hindi" );
```

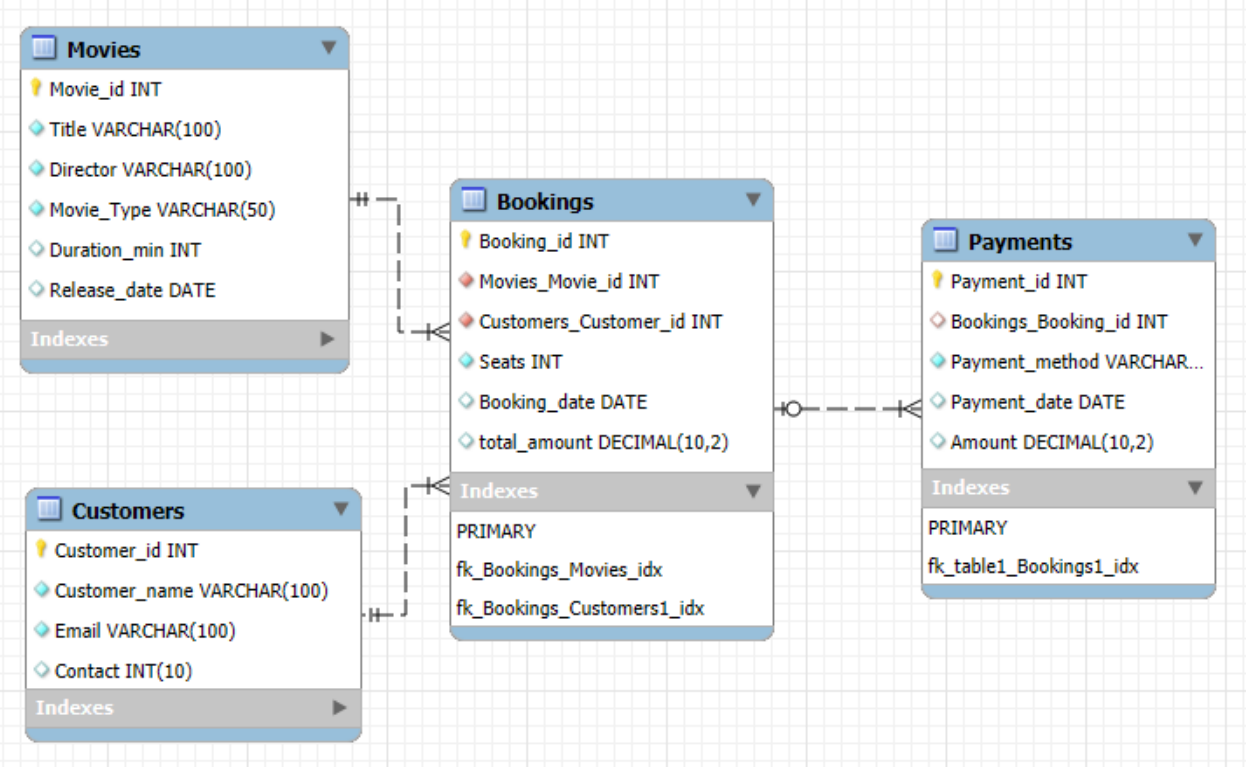
```
# Inserting values in table 2: CUSTOMERS
    INSERT INTO Customers VALUES
(101, "Asha Patil" , "asha.patil@gmail.com" ,9876543210),
(102, "Rohit Sharma" , "rohit.sharma@gmail.com" ,9823456789),
(103, "Sneha Kulkarni" , "sneha.kulkarni@yahoo.com" ,9890011223),
(104, "Kumar Joshi" , "kumar.joshi@gmail.com" ,9765432109),
(105, "Priya Deshmukh" , "priya.deshmukh@gmail.com" ,9832145678),
(106, "Rahul Pawar" , "rahul.pawar@yahoo.com" ,9977885544 ),
(107, "Meera Shinde" , "meera.shinde@gmail.com" ,9812345670),
(108, "Sahil Patankar" , "sahil.patankar@gmail.com" ,9922334455),
(109, "Neha Jadhav" , "neha.jadhav@gmail.com" ,9845123001),
(110, "Ajay Kale" , "ajay.kale@gmail.com" ,9988776655),
(111, "Rutuja More" , "rutuja.more@gmail.com" ,9876001122),
(112, "Aniket Gawande" , "aniket.gawande@gmail.com" ,9898456321),
(113, "Pooja Naik" , "pooja.naik@yahoo.com" ,9933445566),
(114, "Vaibhav Patil" , "vaibhav.patil@gmail.com" ,9822103456),
(115, "Komal Chavan" , "komal.chavan@gmail.com",9944556677),
(116, "Sagar Desai" , "sagar.desai@gmail.com" ,9966332211),
(117, "Riya Nair" , "riya.nair@gmail.com" ,9811789023),
(118, "Harshad Khot" , "harshad.khot@gmail.com" ,9823004455),
(119, "Tanvi Kulkarni" , "tanvi.kulkarni@gmail.com" ,9899001122),
(120, "Tejas Mane" , "tejas.mane@gmail.com" ,9900112233);

# Inserting values in table 3: BOOKINGS
    INSERT INTO Bookings VALUES
(1001, 1, 101, 2, "2025-10-01", 400.00),
(1002, 1, 103, 1, "2025-10-02", 200.00),
(1003, 2, 104, 3, "2025-10-03", 750.00),
(1004, 3, 105, 2, "2025-10-03", 300.00),
(1005, 4, 106, 4, "2025-10-04", 800.00),
(1006, 5, 107, 5, "2025-10-04", 1000.00),
(1007, 6, 108, 2, "2025-10-05", 500.00),
(1008, 7, 109, 3, "2025-10-05", 600.00),
(1009, 8, 110, 2, "2025-10-06", 400.00),
(1010, 9, 111, 1, "2025-10-06", 250.00),
(1011, 10, 112, 2, "2025-10-07", 500.00),
(1012, 11, 113, 4, "2025-10-07", 800.00),
(1013, 12, 114, 3, "2025-10-08", 900.00),
(1014, 13, 115, 2, "2025-10-08", 500.00),
(1015, 14, 116, 5, "2025-10-09", 1250.00),
(1016, 15, 117, 1, "2025-10-09", 200.00),
(1017, 16, 118, 3, "2025-10-10", 600.00),
(1018, 17, 119, 2, "2025-10-10", 400.00),
(1019, 18, 120, 4, "2025-10-11", 1000.00),
(1020, 19, 101, 2, "2025-10-11", 500.00),
(1021, 20, 102, 3, "2025-10-12", 750.00),
(1022, 21, 103, 2, "2025-10-12", 400.00),
(1023, 22, 104, 1, "2025-10-13", 300.00),
(1024, 23, 105, 3, "2025-10-13", 750.00),
(1025, 24, 106, 2, "2025-10-14", 400.00),
```

```
# Inserting values in table 4: PAYMENTS
    INSERT INTO Payments VALUES
(201, 1001, "UPI", "2025-10-01", 400.00),
(202, 1002, "Cash", "2025-10-02", 200.00),
(203, 1003, "Credit Card", "2025-10-03", 750.00),
(204, 1004, "Debit Card", "2025-10-03", 300.00),
(205, 1005, "UPI", "2025-10-04", 800.00),
(206, 1006, "Net Banking", "2025-10-04", 1000.00),
(207, 1007, "UPI", "2025-10-05", 500.00),
(208, 1008, "Cash", "2025-10-05", 600.00),
(209, 1009, "Credit Card", "2025-10-06", 400.00),
(210, 1010, "Debit Card", "2025-10-06", 250.00),
(211, 1011, "UPI", "2025-10-07", 500.00),
(212, 1012, "Cash", "2025-10-07", 800.00),
(213, 1013, "Credit Card", "2025-10-08", 900.00),
(214, 1014, "UPI", "2025-10-08", 500.00),
(215, 1015, "Wallet", "2025-10-09", 1250.00),
(216, 1016, "Cash", "2025-10-09", 200.00),
(217, 1017, "Credit Card", "2025-10-10", 600.00),
(218, 1018, "UPI", "2025-10-10", 400.00),
(219, 1019, "Debit Card", "2025-10-11", 1000.00),
(220, 1020, ",UPI", "2025-10-11", 500.00),
(221, 1021, "Net Banking", "2025-10-12", 750.00),
(222, 1022, "Cash", "2025-10-12", 400.00),
(223, 1023, "Credit Card", "2025-10-13", 300.00),
(224, 1024, "UPI", "2025-10-13", 750.00),
(225, 1025, "Debit Card", "2025-10-14", 400.00),
```

Insight: After inserting the records, my database finally came alive with real data.

Now I can actually see how movies, customers, and their bookings connect with each other.

Entity Relationship [E-R] Diagram



Question: Updating the Movies table by changing the title and genre of the movie whose Movie_id is 3

Input:

```
UPDATE Movies
SET Title = 'Inception', Movie_Type = 'Sci-Fi'
WHERE Movie_id = 3;
```

Output:

Movie_id	Title	Movie_Type
1	Pathaan	Action
2	KGF: Chapter 2	Action
3	Inception	Sci-Fi
4	Bahubali: The Beginning	Fantasy
5	Bahubali: The Conclusion	Fantasy
6	Dangal	Biography
7	Chhichhore	Drama
8	Tanhaji	Historical

Insight: Updates wrong or old movie details and shows how to change specific fields correctly using a WHERE condition.

Question: Show all records from the Payments table.Input:

```
SELECT * FROM Payments;
```

Output:

Payment_id	Booking_id	Payment_method	Payment_date	Amount
201	1001	UPI	2025-10-01	400.00
202	1002	Cash	2025-10-02	200.00
203	1003	Credit Card	2025-10-03	750.00
204	1004	Debit Card	2025-10-03	300.00
205	1005	UPI	2025-10-04	800.00
206	1006	Net Banking	2025-10-04	1000.00
207	1007	UPI	2025-10-05	500.00
208	1008	Cash	2025-10-05	600.00
209	1009	Credit Card	2025-10-06	400.00

Insight: Viewing all payment records helps verify that every transaction is stored properly and no payment entries are missing.

Question: Calculate summary values (avg, sum, max, min) from total_amount.

Input:

```
SELECT 'Average Amount' AS Description, ROUND(AVG(total_amount), 2) AS Value FROM Bookings
UNION ALL
SELECT 'Total Amount', ROUND(SUM(total_amount), 2) FROM Bookings
UNION ALL
SELECT 'Largest Amount', ROUND(MAX(total_amount), 2) FROM Bookings
UNION ALL
SELECT 'Smallest Amount', ROUND(MIN(total_amount), 2) FROM Bookings;
```

Output:

Description	Value
Average Amount	626.56
Total Amount	20050.00
Largest Amount	1250.00
Smallest Amount	200.00

Insight: These values give a quick idea of how much money is coming in. They show the normal booking amount, the highest payment, and the smallest payment.

Question: Show bookings where total_amount is either 500 or 1000.

Input:

```
SELECT * FROM Bookings where total_amount in (500, 1000);
```

Output:

Booking_id	Movie_id	Customer_id	Seats	Booking_date	total_amount
1006	5	107	5	2025-10-04	1000.00
1007	6	108	2	2025-10-05	500.00
1011	10	112	2	2025-10-07	500.00
1014	13	115	2	2025-10-08	500.00
1019	18	120	4	2025-10-11	1000.00
1020	19	101	2	2025-10-11	500.00
1028	2	109	2	2025-10-15	500.00
1032	5	111	4	2025-10-16	1000.00

Insight: The query demonstrates filtering with the IN condition and helps identify frequently occurring ticket values.

Question: List movies with titles beginning with the letter D.

Input:

```
SELECT * FROM Movies where Title like 'D%';
```

Output:

Movie_id	Title	Movie_Type	Director	Duration_min	Release_date	Language
6	Dangal	Biography	Nitesh Tiwari	161	2016-12-23	Hindi
17	Dil Bechara	Romance	Mukesh Chhabra	101	2020-07-24	Hindi
18	Drishyam 2	Thriller	Abhishek Pathak	140	2022-11-18	Hindi
25	Dunki	Comedy	Rajkumar Hirani	160	2024-12-22	Hindi
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Insight: Pattern searching with LIKE helps validate text-based filtering and is useful for search features in real applications.

Question: Show payment count and total revenue for each payment method.

Input:

```
SELECT Payment_method, COUNT(*) AS payments_count, SUM(Amount) AS total_received
FROM Payments
GROUP BY Payment_method
ORDER BY total_received DESC;
```

Output:

Payment_method	payments_count	total_received
UPI	9	4550.00
Credit Card	6	3550.00
Debit Card	5	2750.00
Cash	6	2700.00
Wallet	2	2500.00
Net Banking	2	1750.00

Insight: Grouping by payment method reveals which payment options customers prefer and which generate more revenue.

Question: Find customers who booked more than once on the same date.

Input:

```
SELECT b.Customer_id, b.Booking_date, COUNT(*) AS count_same_day
FROM Bookings b
GROUP BY b.Customer_id, b.Booking_date
HAVING COUNT(*) > 1;
```

Output:

Customer_id	Booking_date	count_same_day
106	2025-10-14	2
111	2025-10-16	2

Insight: This helps detect repeated or multiple transactions. It may indicate bulk bookings or users who are very active.

Question: List each customer's bookings with a serial number.

Input:

```
SELECT Customer_name, Title, Booking_date,
ROW_NUMBER() OVER (PARTITION BY Customers.Customer_id ORDER BY Booking_date) AS Booking_No
FROM Customers JOIN Bookings
ON Customers.Customer_id = Bookings.Customer_id
JOIN Movies ON Bookings.Movie id = Movies.Movie id;
```

Output:

Customer_name	Title	Booking_date	Booking_No
Asha Patil	Pathaan	2025-10-01	1
Asha Patil	Vikram Vedha	2025-10-11	2
Rohit Sharma	Kantara	2025-10-12	1
Sneha Kulkarni	Pathaan	2025-10-02	1
Sneha Kulkarni	Super 30	2025-10-12	2
Kumar Joshi	KGF: Chapter 2	2025-10-03	1
Kumar Joshi	Avatar: The Way of Water	2025-10-13	2
Priya Deshmukh	3 Idiots	2025-10-03	1

Insight: ROW_NUMBER highlights customer activity patterns and helps trace booking history in sequence.

Question: Show booking ID with related movie, customer, and payment details.

Input:

```
SELECT b.Booking_id, m.Title, c.customer_name, p.Payment_method, p.Amount
FROM Bookings b
JOIN Movies m ON b.Movie_id = m.Movie_id
JOIN Customers c ON b.Customer_id = c.Customer_id
JOIN Payments p ON p.Booking_id = b.Booking_id
ORDER BY Booking_id;
```

Output:

Booking_id	Title	customer_name	Payment_method	Amount
1001	Pathaan	Asha Patil	UPI	400.00
1002	Pathaan	Sneha Kulkarni	Cash	200.00
1003	KGF: Chapter 2	Kumar Joshi	Credit Card	750.00
1004	3 Idiots	Priya Deshmukh	Debit Card	300.00
1005	Bahubali: The Beginning	Rahul Pawar	UPI	800.00
1006	Bahubali: The Conclusion	Meera Shinde	Net Banking	1000.00
1007	Dangal	Sahil Patankar	UPI	500.00
1008	Chhichhore	Neha Jadhav	Cash	600.00

Insight: This detailed view ensures that all linked data across tables is accurate and consistent.

Question: Show top 5 movie bookings with the highest amount.

Input:

```
SELECT Movies.Movie_id, Title, MAX(total_amount) AS Amounts
FROM Movies JOIN Bookings
ON Movies.Movie_id= Bookings.Movie_id
GROUP BY Movies.Movie_id, Title
ORDER BY Amounts DESC
LIMIT 5;
```

Output:

Movie_id	Title	Amounts
14	Jawaan	1250.00
25	Dunki	1250.00
18	Drishyam 2	1000.00
5	Bahubali: The Conclusion	1000.00
12	RRR	900.00

Insight: This highlights the most expensive bookings, usually linked to premium seating or bulk reservations.

Question: Show the top 5 customers by total spending.

Input:

```
SELECT Customer_name, SUM(total_amount) AS Total_spent
FROM Customers JOIN Bookings
ON Customers.Customer_id = Bookings.Customer_id
GROUP BY Customers.Customer_id, Customers.Customer_name
ORDER BY total_spent DESC
LIMIT 5;
```

Output:

Customer_name	Total_spent
Rahul Pawar	2450.00
Meera Shinde	2250.00
Rutuja More	2050.00
Sagar Desai	1250.00
Neha Jadhav	1100.00

Insight: These customers are the most valuable in terms of revenue. The query demonstrates combining GROUP BY with ORDER BY and LIMIT.

Question: Calculate total booking earnings based on movie type.

Input:

```
SELECT Movie_Type, SUM(Bookings.total_amount) AS Earning
FROM Movies JOIN Bookings
ON Movies.Movie_id = Bookings.Movie_id
GROUP BY Movie_Type ORDER BY Earning DESC ;
```

Output:

Movie_Type	Earning
Action	5500.00
Fantasy	3600.00
Comedy	2500.00
Drama	2250.00
Thriller	1900.00
Romance	1750.00
Sci-Fi	1050.00
Biography	900.00
Historical	400.00
Adventure	200.00

Insight: Movie categories can be compared to understand which genre performs better in the market.

Question: Count total bookings per language.

Input:

```
SELECT Language, COUNT(Bookings.Booking_id) AS Total_Bookings
FROM Movies JOIN Bookings
ON Movies.Movie_id = Bookings.Movie_id
GROUP BY Language;
```

Output:

Language	Total_Bookings
Hindi	18
Kannada	3
Telugu	6
English	5

Insight: Language preferences become clear, showing which languages attract more viewers.

Question: Display bookings made on the latest booking date.

Input:

```
SELECT * FROM Bookings
WHERE Booking_date = (SELECT MAX(Booking_date) FROM Bookings);
```

Output:

Booking_id	Movie_id	Customer_id	Seats	Booking_date	total_amount
1029	3	110	3	2025-10-16	600.00
1030	4	111	4	2025-10-16	800.00
1032	5	111	4	2025-10-16	1000.00
NULL	NULL	NULL	NULL	NULL	NULL

Insight: This uses a subquery to identify fresh/active entries. It confirms that the booking table is updated recently.

CONCLUSION

This project shows how a Movie Booking System can be made using SQL. It handles movie details, customer information, bookings, and payments in an organized way. We learned how to create tables, connect them using primary and foreign keys, and write SQL queries like JOINs, aggregates, ranking and windows functions and subqueries The system helps us easily check bookings, payments, and total earnings. Overall, this project improved our understanding of database design and how SQL works in real life.

In the future, we can add more features like automatic messages, better seat selection, and simple dashboards.