

Database Management  
System LAB

**Project Report**

|  |  |
| --- | --- |
| Group Members: |  |
| Program: | BSSE |
| Section: | 4-A |

**Lab Instructor:**

Table of Contents

[Hotel Booking Management System 3](#_Toc199640823)

[**1. Introduction** 3](#_Toc199640825)

[2. Motivation 3](#_Toc199640826)

[3. Table Structures and Columns 3](#_Toc199640827)

[Hotels 3](#_Toc199640828)

[**Rooms** 3](#_Toc199640829)

[**Customers** 4](#_Toc199640830)

[**Bookings** 4](#_Toc199640831)

[**Reviews** 4](#_Toc199640832)

[**Staff** 5](#_Toc199640833)

[**4. Screenshots and Evidence** 5](#_Toc199640834)

[**-Table Creation in MySQL** 5](#_Toc199640835)

[**- Data Insertion using (LOAD DATA INFILE TO IMPORT DATA FROM CSV FILES TO THE TABLES)** 6](#_Toc199640836)

[**- Sample Aggregate Queries** 6](#_Toc199640837)

[**- JOINs and Subqueries** 8](#_Toc199640838)

[**- Transactions and Indexing** 10](#_Toc199640839)

[**- Procedures** 12](#_Toc199640840)

[**- Views** 13](#_Toc199640841)

[**5. Key SQL Concepts Demonstrated** 15](#_Toc199640842)

[**6. SQL Script Best Practices** 15](#_Toc199640843)

[**7. Conclusion** 15](#_Toc199640844)

# Hotel Booking Management System

# 

**1. Introduction**

The **Hotel Booking Management System** is a relational database project designed to simulate real-world hotel operations. It manages room bookings, customer information, payments, reviews, and staff data. The project emphasizes performance optimization using indexing, constraints, and normalization.

The dataset also includes realistic data inconsistencies such as NULL values and duplicate entries to reflect real-world conditions and to test query robustness.

## 2. Motivation

The hotel industry deals with large volumes of daily data. Manual processes are inefficient and prone to error. This project was initiated to:

* Efficiently manage room availability and bookings
* Simulate real-time data analytics with SQL
* Ensure ACID-compliant transaction handling
* Test SQL proficiency using large and inconsistent datasets

## 3. Table Structures and Columns

### Hotels

* Hotel\_ID **(Primary Key)**
* Name
* Location
* Rating
* Contact\_Email
* Contact\_Number

**Rooms**

* Room\_ID (Primary Key)
* Hotel\_ID (Foreign Key → Hotels)
* Room\_Type
* Room\_Number
* Price\_Per\_Night
* Availability Status

**Customers**

* Customer\_ID (Primary Key)
* Full\_Name
* Email
* Phone\_Number
* Address
* Nationality
* CNIC/Passport\_Number

**Bookings**

* Booking\_ID (Primary Key)
* Room\_ID (Foreign Key → Rooms)
* Customer\_ID (Foreign Key → Customers)
* Check\_IN\_Date
* Check\_Out\_Date
* Num\_guests
* Payment Status

**Reviews**

* Review\_ID (Primary Key)
* Hotel\_ID (Foreign Key → Hotels)
* Customer\_ID (Foreign Key → Customers)
* Rating
* Comment
* Review Date

**Staff**

* Staff\_ID (Primary Key)
* Hotel\_ID (Foreign Key → Hotels)
* Full\_Name
* Designation
* Salary
* Hire\_Date

**4. Screenshots and Evidence**

**-Table Creation in MySQL**

**Query:**

create table hotels (

hotel\_id int primary key auto\_increment,

name varchar(100) not null,

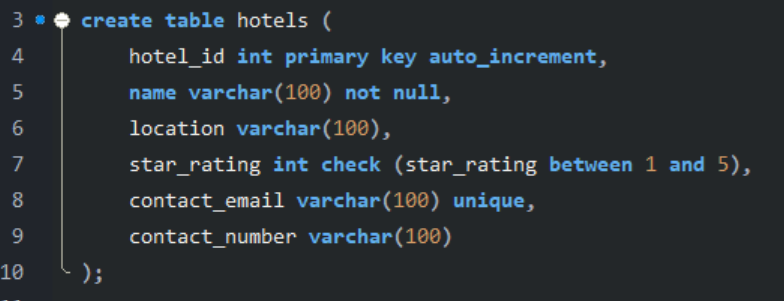
location varchar(100),

star\_rating int check (star\_rating between 1 and 5),

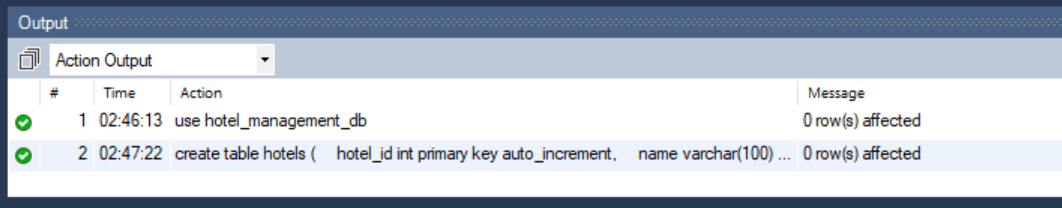
contact\_email varchar(100) unique,

contact\_number varchar(100)

);



**Output:**

****

**- Data Insertion using (LOAD DATA INFILE TO IMPORT DATA FROM CSV FILES TO THE TABLES)**

**Query:**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/hotels.csv'

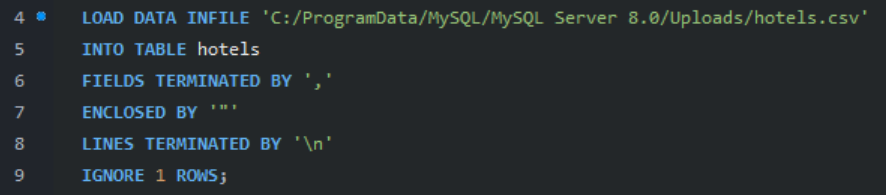
INTO TABLE hotels

FIELDS TERMINATED BY ','

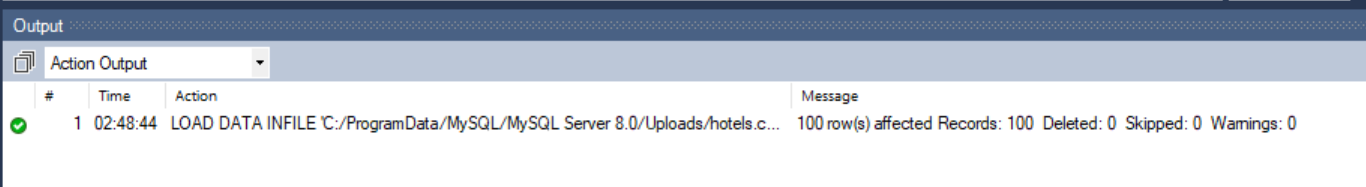
ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;



**Output:**



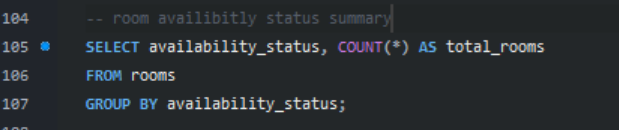
**- Sample Aggregate Queries**

**Query:**

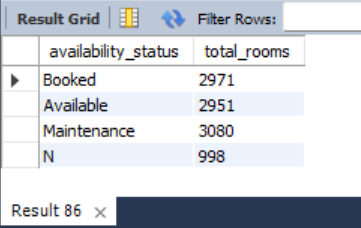
SELECT availability\_status, COUNT(\*) AS total\_rooms

FROM rooms

GROUP BY availability\_status;



**Output:**



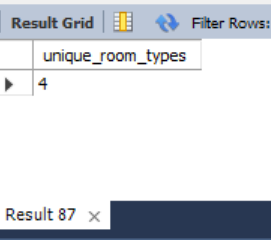
**Query:**

SELECT COUNT(DISTINCT room\_type) AS unique\_room\_types

FROM rooms;



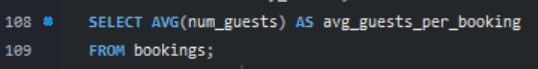
**Output:**



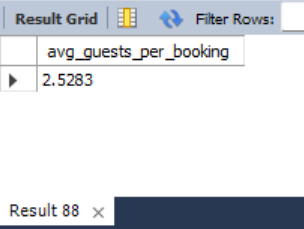
**Query:**

SELECT AVG(num\_guests) AS avg\_guests\_per\_booking

FROM bookings;



**Output:**



**- JOINs and Subqueries**

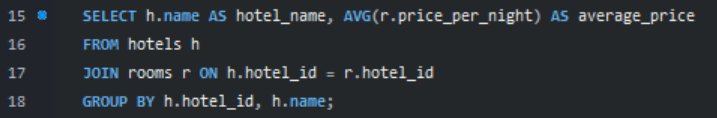
**Query:**

SELECT h.name AS hotel\_name, AVG(r.price\_per\_night) AS average\_price

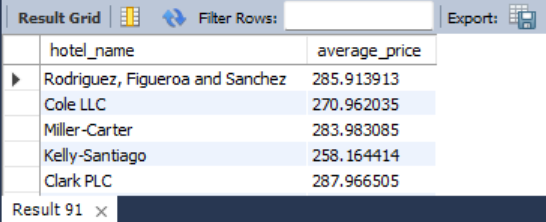
FROM hotels h

JOIN rooms r ON h.hotel\_id = r.hotel\_id

GROUP BY h.hotel\_id, h.name;



**Output:**



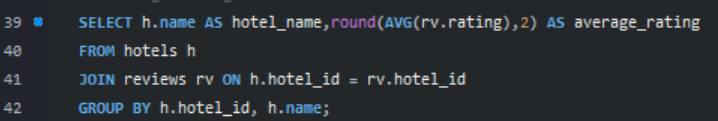
**Query:**

SELECT h.name AS hotel\_name,round(AVG(rv.rating),2) AS average\_rating

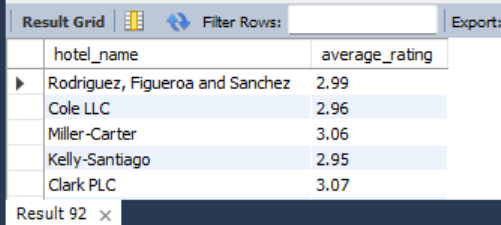
FROM hotels h

JOIN reviews rv ON h.hotel\_id = rv.hotel\_id

GROUP BY h.hotel\_id, h.name;



**Output:**



**Query:**

SELECT hotel\_id,name

FROM hotels

WHERE hotel\_id IN (

SELECT hotel\_id

FROM reviews

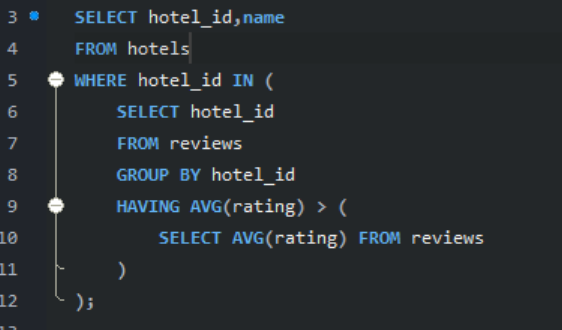
GROUP BY hotel\_id

HAVING AVG(rating) > (

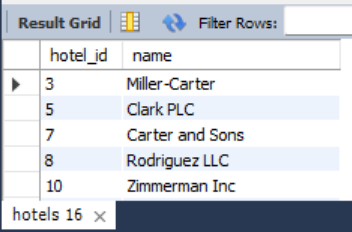
SELECT AVG(rating) FROM reviews

)

);



**Output:**



**Query:**

SELECT \* FROM customers

WHERE customer\_id IN (

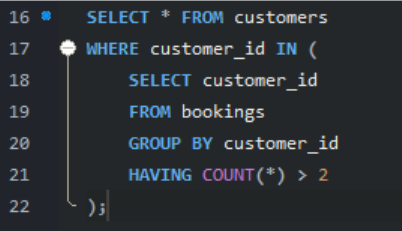
SELECT customer\_id

FROM bookings

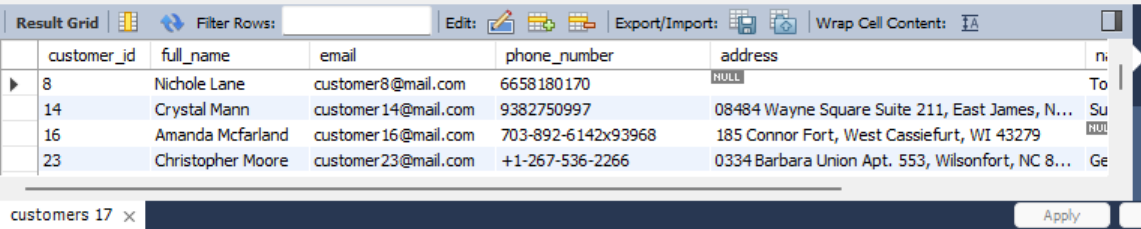
GROUP BY customer\_id

HAVING COUNT(\*) > 2

);



**Output:**



**- Transactions and Indexing**

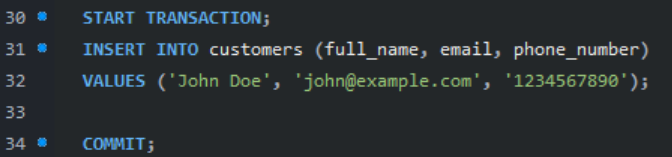
**Query:**

START TRANSACTION;

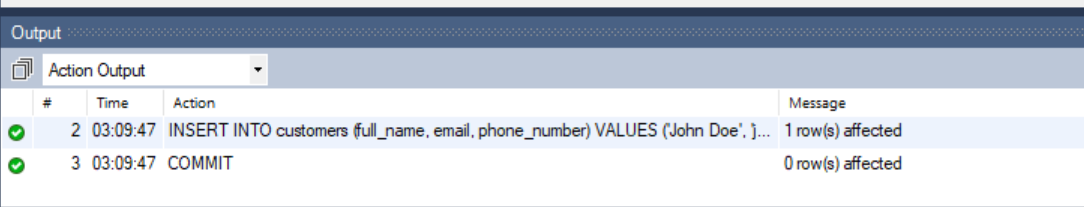
INSERT INTO customers (full\_name, email, phone\_number)

VALUES ('John Doe', 'john@example.com', '1234567890');

COMMIT;



**Output:**

****

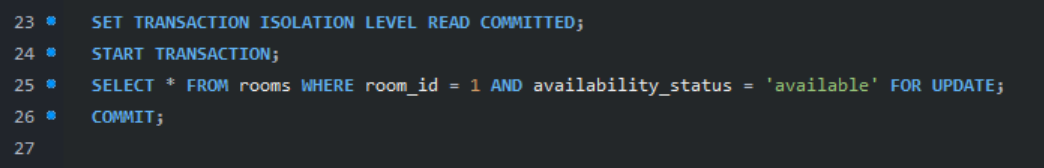
**Query:**

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

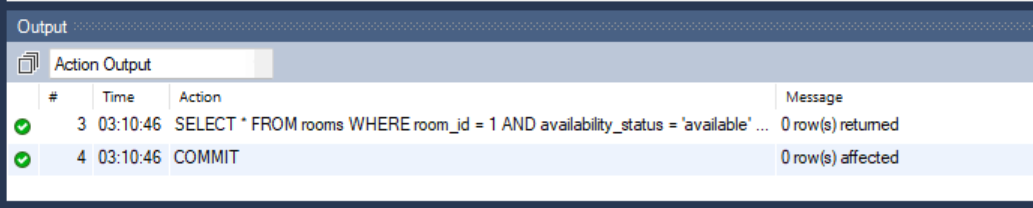
START TRANSACTION;

SELECT \* FROM rooms WHERE room\_id = 1 AND availability\_status = 'available' FOR UPDATE;

COMMIT;



**Output:**

****

**Query:**

CREATE INDEX idx\_rooms\_hotel ON rooms(hotel\_id);

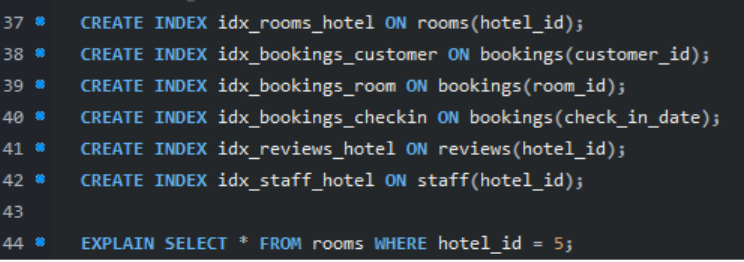
CREATE INDEX idx\_bookings\_customer ON bookings(customer\_id);

CREATE INDEX idx\_bookings\_room ON bookings(room\_id);

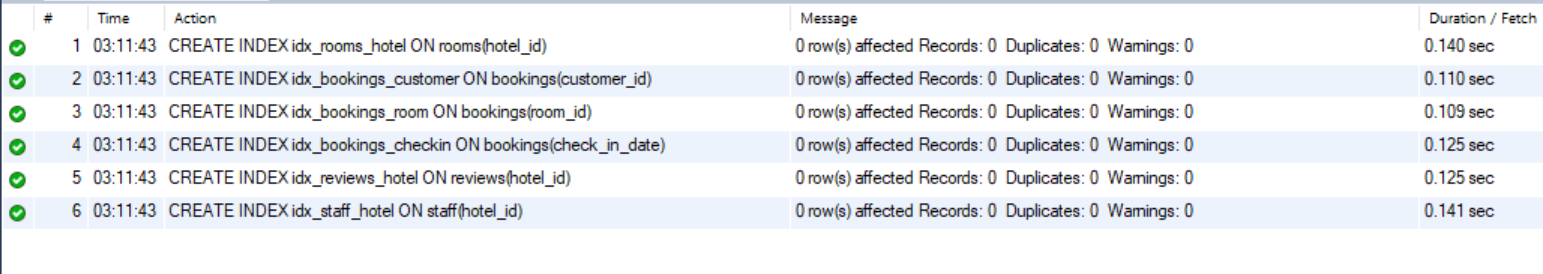
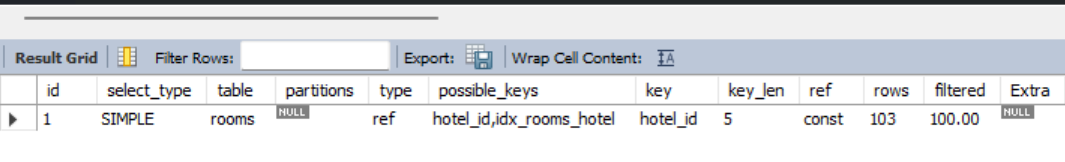
CREATE INDEX idx\_bookings\_checkin ON bookings(check\_in\_date);

CREATE INDEX idx\_reviews\_hotel ON reviews(hotel\_id);

CREATE INDEX idx\_staff\_hotel ON staff(hotel\_id);



**Output:**

**** ****

**- Procedures**

**Query:**

DELIMITER //

CREATE PROCEDURE MarkBookingPaid (

IN p\_booking\_id INT

)

BEGIN

UPDATE bookings

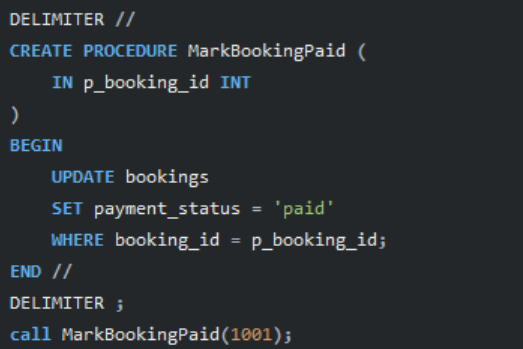
SET payment\_status = 'paid'

WHERE booking\_id = p\_booking\_id;

END //

DELIMITER ;

call MarkBookingPaid(1001);

****

**Output:**

****

**Query:**

DELIMITER //

CREATE PROCEDURE GetBookingStatusCount()

BEGIN

SELECT payment\_status, COUNT(\*) AS count

FROM bookings

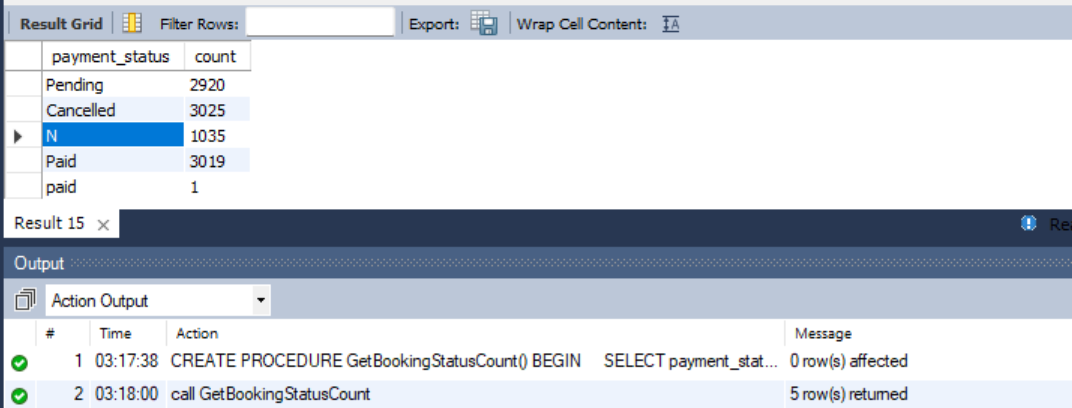
GROUP BY payment\_status;

END //

DELIMITER ;

call GetBookingStatusCount;

**Output:**

****

**- Views**

**Query:**

CREATE VIEW view\_customer\_bookings AS

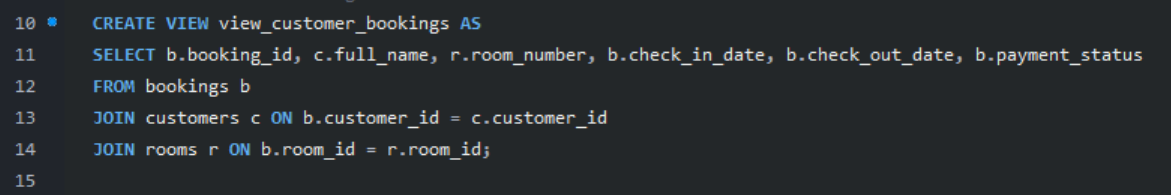
SELECT b.booking\_id, c.full\_name, r.room\_number, b.check\_in\_date, b.check\_out\_date, b.payment\_status

FROM bookings b

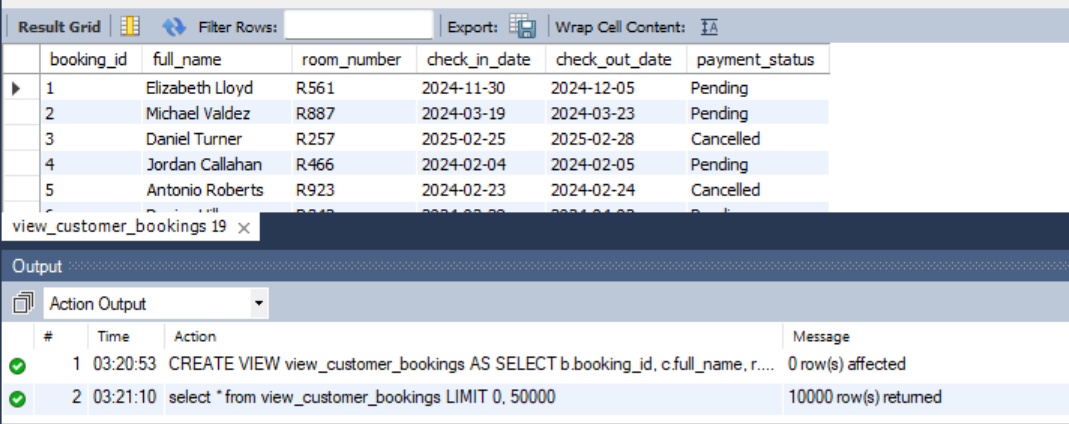
JOIN customers c ON b.customer\_id = c.customer\_id

JOIN rooms r ON b.room\_id = r.room\_id;

select \* from view\_customer\_bookings;



**Output:**

****

**Query:**

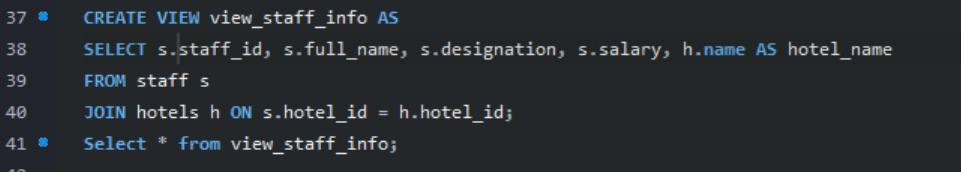
CREATE VIEW view\_staff\_info AS

SELECT s.staff\_id, s.full\_name, s.designation, s.salary, h.name AS hotel\_name

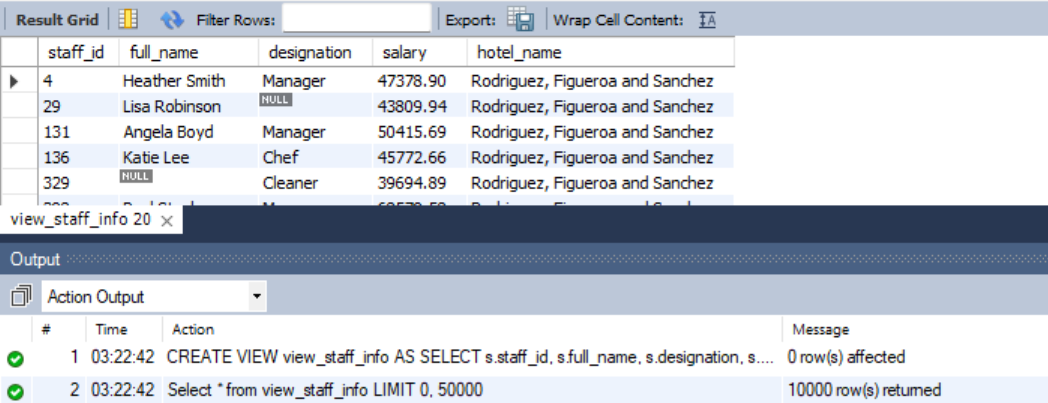
FROM staff s

JOIN hotels h ON s.hotel\_id = h.hotel\_id;

Select \* from view\_staff\_info;



**Output:**

****

**5. Key SQL Concepts Demonstrated**

| **Concept** | **Example Usage** |
| --- | --- |
| Aggregate Functions | COUNT(), SUM(), AVG(), MIN(), MAX() |
| JOINs | INNER JOIN, LEFT JOIN, etc. |
| Subqueries | WHERE NOT IN (SELECT ...) |
| Constraints | PRIMARY KEY, FOREIGN KEY, CHECK, UNIQUE |
| Transactions | START TRANSACTION, COMMIT, ROLLBACK |
| Indexing | CREATE INDEX ON rooms(availability\_status) |

**6. SQL Script Best Practices**

* Used consistent indentation and comments in queries
* Included NULL and duplicate data for robustness testing
* All tables follow 3NF normalization
* No constraint violations in inserted data

**7. Conclusion**

The Hotel Booking Management System project offers a complete simulation of a real-world hotel database. It highlights key SQL features including joins, subqueries, indexing, and transaction handling. With over 10,000 records in each table, the system demonstrates scalability and practical use cases in database design and performance testing.