

***ASSIGNMENT # 3***

|  |  |
| --- | --- |
| **NAME:** | NOOR FATIMA |
| **REGISTRATION NO:** | SP23-BCS-109 |
| **COURSE:** | DATABASE MANAGEMENT SYSTEM |
| **TEACHER:** | Dr. ABID SUHAIL BHUTTA |
| **SECTION:** | C |
| **DATE :** | 21-12-2024 |

**REPORTS :**

***Procedures:***

1. **MonthlyOccupancyReport:**

The stored procedure calculates the monthly occupancy report for rooms in a hotel or accommodation setup. It takes two input parameters: @month and @year, which define the specific month and year for which the occupancy report is generated.

* **Query :**

CREATE PROCEDURE sp\_MonthlyOccupancyReport

@month INT,

@year INT

AS

BEGIN

SELECT

r.room\_type,

COUNT(a.room\_id) as occupied\_rooms,

rt.room\_capacity \* COUNT(DISTINCT r.room\_id) as total\_capacity,

CAST(COUNT(a.room\_id) \* 100.0 / COUNT(DISTINCT r.room\_id) AS DECIMAL(5,2)) as occupancy\_rate

FROM Rooms r

LEFT JOIN Allotment a ON r.room\_id = a.room\_id

JOIN Room\_type rt ON r.room\_type = rt.room\_type

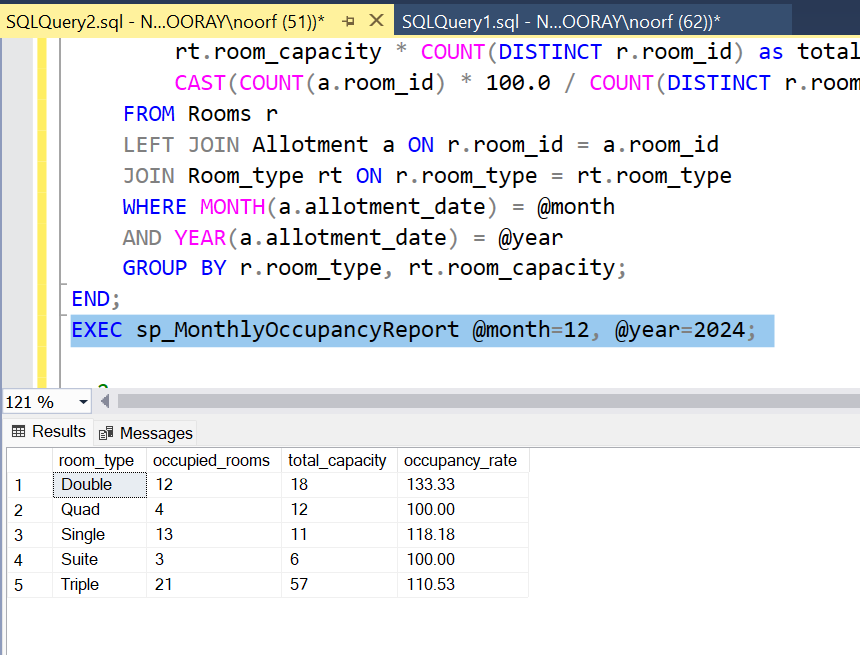
WHERE MONTH(a.allotment\_date) = @month

AND YEAR(a.allotment\_date) = @year

GROUP BY r.room\_type, rt.room\_capacity;

END;

* **Execution**:

EXEC sp\_MonthlyOccupancyReport @month=12, @year=2024;

2- **RevenueByRoomType:**

The stored procedure is designed to calculate the total revenue, total allotments, and average revenue per allotment for each room type within a specified date range. The procedure takes two input parameters**: @start\_date** and **@end\_date,** which define the date range for which the report will be generated.

* **Query:**

CREATE PROCEDURE sp\_RevenueByRoomType

@start\_date DATE,

@end\_date DATE

AS

BEGIN

SELECT

r.room\_type,

COUNT(DISTINCT a.allotment\_id) as total\_allotments,

SUM(a.amount) as total\_revenue,

AVG(a.amount) as avg\_revenue\_per\_allotment

FROM Allotment a

JOIN Rooms r ON a.room\_id = r.room\_id

WHERE a.allotment\_date BETWEEN @start\_date AND @end\_date

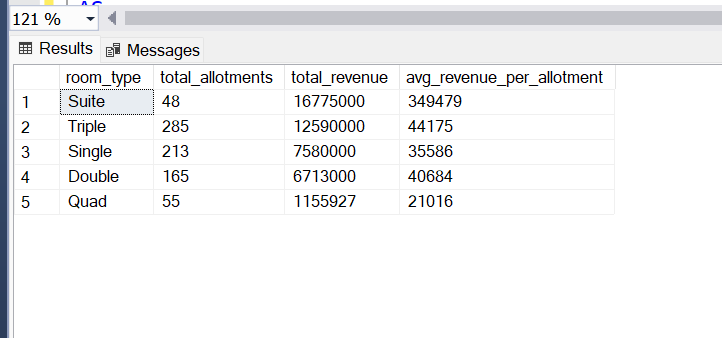
GROUP BY r.room\_type

ORDER BY total\_revenue DESC;

END;

* **EXECUTION**

EXEC sp\_RevenueByRoomType @start\_date = '2024-01-01', @end\_date = '2024-12-31';



**3- StaffPerformanceReport:**

The stored procedure is designed to generate a report that evaluates the performance of hotel staff based on the number of rooms managed, the total allotments (bookings) handled, and the revenue generated by those allotments.

* **QUERY**:

cREATE PROCEDURE sp\_StaffPerformanceReport

AS

BEGIN

SELECT

s.staff\_id,

s.staff\_name,

COUNT(DISTINCT r.room\_id) as rooms\_managed,

COUNT(DISTINCT a.allotment\_id) as total\_allotments,

SUM(a.amount) as revenue\_generated

FROM Staff s

LEFT JOIN Rooms r ON s.staff\_id = r.staff\_id

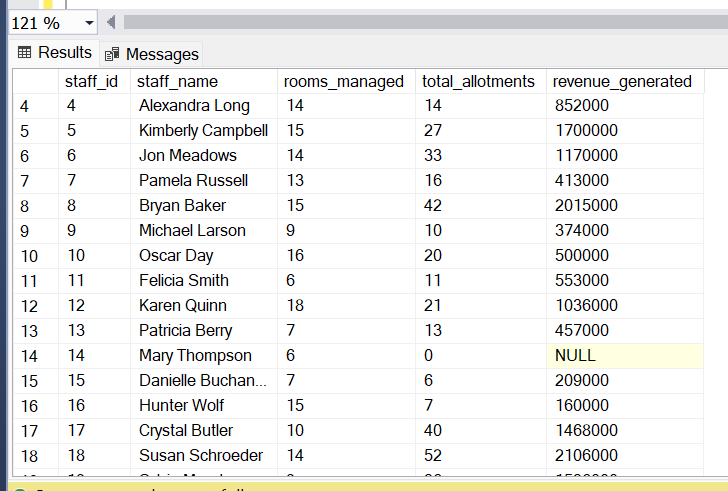
LEFT JOIN Allotment a ON r.room\_id = a.room\_id

GROUP BY s.staff\_id, s.staff\_name;

END;

* EXECUTION:

EXEC sp\_StaffPerformanceReport



***VIEWS***

1. **ResidentStayAnalysis:**

The view is designed to analyze the stay details of residents in a hotel or accommodation setup. It provides a comprehensive report about each resident’s stay, total stay days, total money spent, and the date of their most recent stay. This view combines data from the Resident table (which holds resident details) and the Allotment table (which tracks individual stays and bookings).

* **QUERY**:

CREATE VIEW vw\_ResidentStayAnalysis AS

SELECT

res.resident\_id,

res.resident\_name,

COUNT(DISTINCT a.allotment\_id) as total\_stays,

SUM(a.stay\_time\_period) as total\_stay\_days,

SUM(a.amount) as total\_spent,

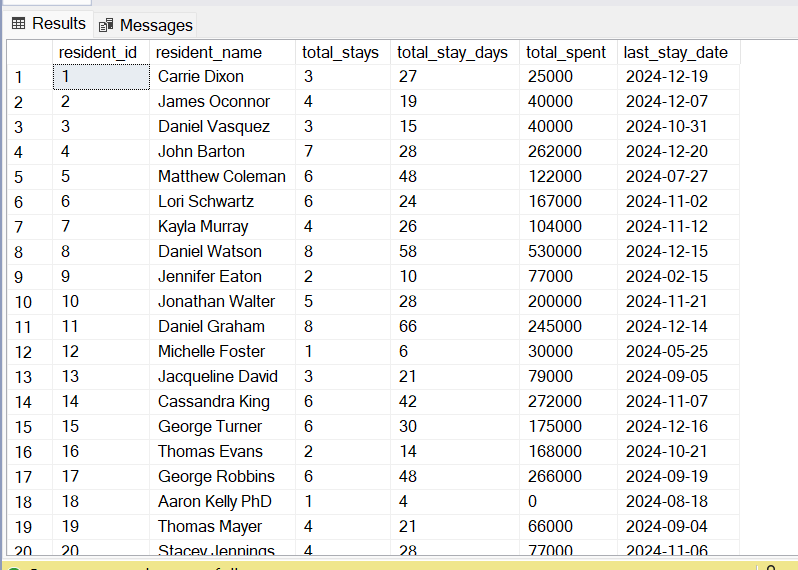
MAX(a.allotment\_date) as last\_stay\_date

FROM Resident res

LEFT JOIN Allotment a ON res.resident\_id = a.resident\_id

GROUP BY res.resident\_id, res.resident\_name;

* **Execution**:



2- **MonthlyRevenue**

The view is designed to provide a monthly revenue report for each room type. It aggregates the total revenue, the number of allotments (bookings), and the average revenue per allotment for each room type within each month. This view uses the Allotment table to track the bookings and the Rooms table to identify the type of room booked

* **QUERY**:

CREATE VIEW vw\_MonthlyRevenue AS

SELECT

FORMAT(a.allotment\_date, 'yyyy-MM') AS month,

r.room\_type,

COUNT(DISTINCT a.allotment\_id) AS total\_allotments,

SUM(a.amount) AS total\_revenue,

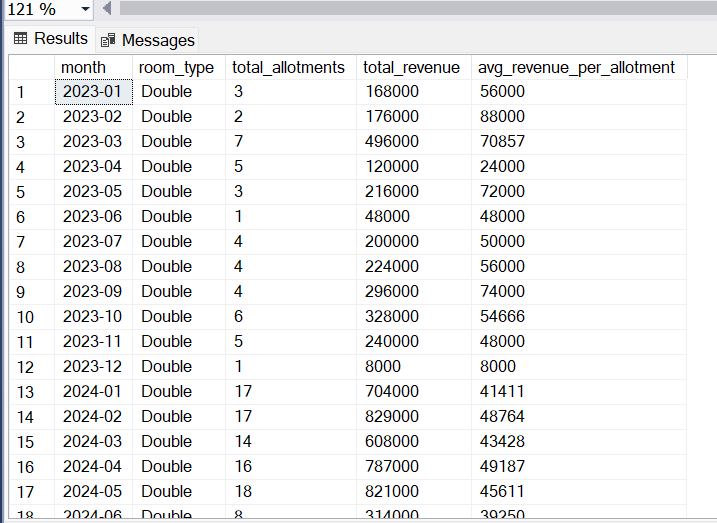
AVG(a.amount) AS avg\_revenue\_per\_allotment

FROM Allotment a

JOIN Rooms r ON a.room\_id = r.room\_id

GROUP BY FORMAT(a.allotment\_date, 'yyyy-MM'), r.room\_type;

* **EXECUTION:**

****

1. **VisitorActivity:**

The view is designed to provide a summary of the activity of visitors who meet residents in a hotel or accommodation setting. It aggregates information such as the total number of meetings a visitor has attended, the first and last meeting dates, and the resident they visited.

* **QUERY**:

CREATE VIEW vw\_VisitorActivity AS

SELECT

v.visitor\_id,

v.visitor\_name,

res.resident\_name AS visited\_resident,

COUNT(m.meeting\_id) AS meeting\_count,

MAX(m.meeting\_datetime) AS last\_visit,

MIN(m.meeting\_datetime) AS first\_visit

FROM Visitor v

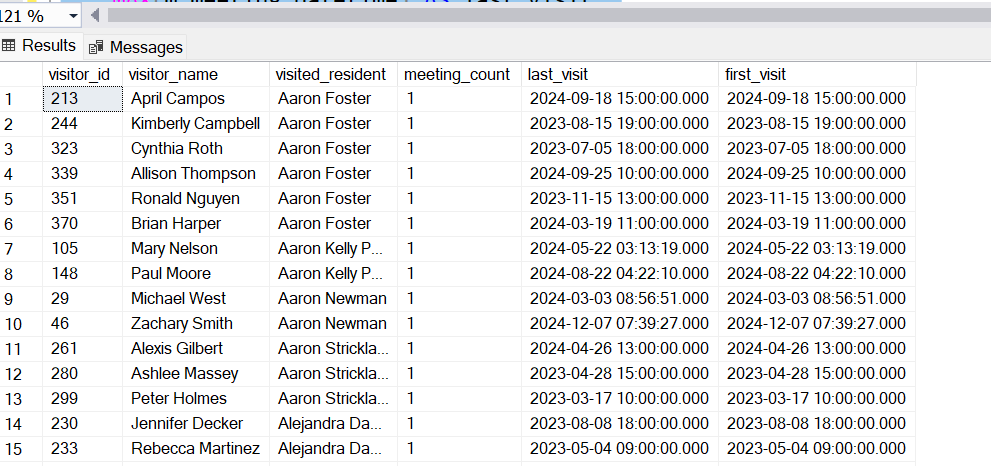
JOIN Meeting m ON v.visitor\_id = m.visitor\_id

JOIN Resident res ON m.resident\_id = res.resident\_id

GROUP BY v.visitor\_id, v.visitor\_name, res.resident\_name;

* **EXECUTION**:

Select \* from vw\_VisitorActivity



***Materialized View***

1- **MonthlyRevenue**

The view is designed to provide a detailed report on the monthly revenue generated from residents, broken down by room type. It calculates the total revenue, the number of residents, and the average revenue per resident for each month and room type. The ROLLUP clause adds summary rows, which include aggregates for each month and overall totals across all months.

**QUERY** :

CREATE VIEW vw\_MonthlyRevenueDenorm AS

SELECT

YEAR(allotment\_date) as year,

MONTH(allotment\_date) as month,

room\_type,

COUNT(DISTINCT resident\_id) as total\_residents,

SUM(total\_amount) as total\_revenue,

AVG(total\_amount) as avg\_revenue\_per\_resident

FROM resident\_denormalized

GROUP BY YEAR(allotment\_date), MONTH(allotment\_date), room\_type

WITH ROLLUP;

SELECT \* FROM vw\_MonthlyRevenueDenorm;