Lecture 3: SQL Mastery

1. The **Sales** table records information about **product sales**, including the **quantity sold**, **sale date**, and **total price** for each sale. It serves as a transactional data source for analyzing sales trends.

Query:

```
-- Create Sales table

CREATE TABLE Sales (
    sale_id INT PRIMARY KEY,
    product_id INT,
    quantity_sold INT,
    sale_date DATE,
    total_price DECIMAL(10, 2)
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);

-- Insert sample data into Sales table

INSERT INTO Sales (sale_id, product_id, quantity_sold, sale_date, total_price) VALUES
(1, 101, 5, '2024-01-01', 2500.00),
(2, 102, 3, '2024-01-02', 900.00),
(3, 103, 2, '2024-01-02', 60.00),
(4, 104, 4, '2024-01-03', 80.00),
(5, 105, 6, '2024-01-03', 90.00);
```

2. Products Table

The **Products** table contains details about **products**, including their **names**, **categories**, and unit prices. It provides reference data for linking product information to sales transactions.

Query:

```
-- Create Products table

CREATE TABLE Products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    unit_price DECIMAL(10, 2)
);

-- Insert sample data into Products table

INSERT INTO Products (product_id, product_name, category, unit_price) VALUES
(101, 'Laptop', 'Electronics', 500.00),
(102, 'Smartphone', 'Electronics', 300.00),
(103, 'Headphones', 'Electronics', 30.00),
(104, 'Keyboard', 'Electronics', 20.00),
(105, 'Mouse', 'Electronics', 15.00);
```

3. Retrieve the sale_id and sale_date from the Sales table.

Query:

```
SELECT sale_id, sale_date FROM Sales;
```

4. Filter the Sales table to show only sales with a total_price greater than $100.

Query:

```
SELECT * FROM Sales WHERE total_price > 100;
```

5. Filter the Products table to show only products in the 'Electronics' category.

Query:

```
SELECT * FROM Products WHERE category = 'Electronics';
```

6. Retrieve the sale_id and total_price from the Sales table for sales made on January 3, 2024.

Query:

```
SELECT sale_id, total_price
FROM Sales
WHERE sale_date = '2024-01-03';
```

7. Retrieve the product_id and product_name from the Products table for products with a unit_price greater than $100.

Query:

```
SELECT product_id, product_name
FROM Products
WHERE unit_price > 100;
```

8. Calculate the total revenue generated from sales of products in the 'Electronics' category.

Query:

```
SELECT SUM(Sales.total_price) AS total_revenue
FROM Sales
JOIN Products ON Sales.product_id = Products.product_id
WHERE Products.category = 'Electronics';
```

9. Calculate the total quantity_sold of products in the 'Electronics' category.

Query:

```
SELECT SUM(quantity_sold) AS total_quantity_sold
FROM Sales
JOIN Products ON Sales.product_id = Products.product_id
WHERE Products.category = 'Electronics';
```

10. Explain the significance of indexing in SQL databases and provide an example scenario where indexing could significantly improve query performance in the given schema.

Query:

```
-- Create an index on the sale_date column
CREATE INDEX idx_sale_date ON Sales (sale_date);

-- Query with indexing
SELECT *
FROM Sales
WHERE sale_date = '2024-01-03';
```

11. Implement a transaction that deducts the quantity sold from the Products table when a sale is made in the Sales table, ensuring that both operations are either committed or rolled back together.

Query:

```
START TRANSACTION; -- Begin the transaction

-- Deduct the quantity sold from the Products table
UPDATE Products p
JOIN Sales s ON p.product_id = s.product_id
SET p.quantity_in_stock = p.quantity_in_stock - s.quantity_sold;

-- Check if any negative quantities would result from the update
SELECT COUNT(*) INTO @negative_count
FROM Products
WHERE quantity_in_stock < 0;

-- If any negative quantities would result, rollback the transaction
IF @negative_count > 0 THEN
   ROLLBACK;
   SELECT 'Transaction rolled back due to insufficient stock.' AS Message;
ELSE
   COMMIT; -- Commit the transaction if no negative quantities would result
   SELECT 'Transaction committed successfully.' AS Message;
END IF;

START TRANSACTION;
UPDATE Products SET quantity_in_stock = 10 WHERE product_id = 101;
INSERT INTO Sales (product_id, quantity_sold) VALUES (101, 5);
COMMIT;
```

12. Analyze the performance implications of indexing the sale_date column in the Sales table, considering the types of queries commonly executed against this column.

Query:

```
-- Query without indexing
EXPLAIN ANALYZE
SELECT *
FROM Sales
```

```
WHERE sale_date = '2024-01-01';

-- Query with indexing
CREATE INDEX idx_sale_date ON Sales (sale_date);

EXPLAIN ANALYZE
SELECT *
FROM Sales
WHERE sale_date = '2024-01-01';
```

13. **Create a view named Product_Sales_Info that displays product details along with the total number of sales made for each product.**

Query:

```
CREATE VIEW Product_Sales_Info AS
SELECT
    p.product_id,
    p.product_name,
    p.category,
    p.unit_price,
    COUNT(s.sale_id) AS total_sales
FROM
    Products p
LEFT JOIN
    Sales s ON p.product_id = s.product_id
GROUP BY
    p.product_id, p.product_name, p.category, p.unit_price;
```

14. **Develop a stored procedure named Update_Unit_Price that updates the unit price of a product in the Products table based on the provided product_id.**

Query:

```
DELIMITER //

CREATE PROCEDURE Update_Unit_Price (
    IN p_product_id INT,
    IN p_new_price DECIMAL(10, 2)
)
BEGIN
    UPDATE Products
    SET unit_price = p_new_price
    WHERE product_id = p_product_id;
END //

DELIMITER ;
```

15. Consider the two tables, **Student** and **StudentCourse**, which share a common column **ROLL_NO**. Using SQL JOINS, we can combine data from these tables based on their **relationship**, allowing us to retrieve meaningful information like student details along with their **enrolled courses**.

Student Table

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse Table

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Let's look at the example of **INNER JOIN** clause, and understand it's working. This query will show the names and age of students enrolled in different courses.

Query:

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

16. LEFT JOIN Example

In this example, the **LEFT JOIN** retrieves all rows from the **Student** table and the matching rows from the **StudentCourse** table based on the **ROLL_NO** column.

Query:

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

17. **Employee Table:**

emp_no	emp_name	dept_no
E1	Varun Singhal	D1
E2	Amrita Aggarwal	D2
E3	Ravi Anand	D3

Department Table:

dept_no	d_name	location
D1	IT	Delhi
D2	HR	Hyderabad
D3	Finance	Pune
D4	Testing	Noida
D5	Marketing	Mathura

Example: Perform a RIGHT JOIN on Employee and Department Tables

Now, we will perform SQL RIGHT JOIN on these two tables.

Query:

```
SELECT emp_no , emp_name ,d_name, location
FROM employee
RIGHT  JOIN dept
ON employee.dept_no = department.dept_no;
```

18. we create a stored procedure called **GetCustomersByCountry**, which accepts a Country parameter and returns the **CustomerName** and **ContactName** for all customers from that country. The procedure is designed to query the **Customers** table, which contains customer information, including their **names**, **contact details**, and **country**.

CustomerID	CustomerName	ContactName	Country
1	Shubham	Thakur	India
2	Aman	Chopra	Australia
3	Naveen	Tulasi	Sri Lanka
4	Aditya	Arpan	Austria
5	Nishant	Jain	Spain

Customers Table

By passing a country as a parameter, the stored procedure dynamically fetches the relevant customer details from the table

Query:

```
-- Create a stored procedure named "GetCustomersByCountry"
CREATE PROCEDURE GetCustomersByCountry
    @Country VARCHAR(50)
AS
BEGIN
    SELECT CustomerName, ContactName
    FROM Customers
    WHERE Country = @Country;
END;

-- Execute the stored procedure with parameter "Sri lanka"
EXEC GetCustomersByCountry @Country = 'Sri lanka';
```