SQL practice Queries with Definitions and Examples

Create a Database

Definition: A database is a collection of organized data that can be accessed, managed, and updated.

```
CREATE DATABASE SaleOrder;
```

1.

Use a Database

Definition: Switch to the desired database to perform operations on its tables and data.

```
USE SaleOrder;
```

2.

Create Tables

Definition: Define the structure of your data using tables with specified columns and data types.

```
CREATE TABLE dbo.Customer (
    CustomerID INT NOT NULL PRIMARY KEY,
    CustomerFirstName VARCHAR(50) NOT NULL,
    CustomerLastName VARCHAR(50) NOT NULL,
    CustomerAddress VARCHAR(50) NOT NULL,
```

```
    CustomerSuburb VARCHAR(50) NULL,
    CustomerCity VARCHAR(50) NOT NULL,
    CustomerPostCode CHAR(4) NULL,
    CustomerPhoneNumber CHAR(12) NULL
);

CREATE TABLE dbo.Inventory (
    InventoryID TINYINT NOT NULL PRIMARY KEY,
    InventoryName VARCHAR(50) NOT NULL,
    InventoryDescription VARCHAR(255) NULL
);

CREATE TABLE dbo.Employee (
    EmployeeID TINYINT NOT NULL PRIMARY KEY,
    EmployeeFirstName VARCHAR(50) NOT NULL,
    EmployeeLastName VARCHAR(50) NOT NULL,
    EmployeeExtension CHAR(4) NULL
);

CREATE TABLE dbo.Sale (
```

```
    SaleID TINYINT NOT NULL PRIMARY KEY,
            CustomerID INT NOT NULL REFERENCES dbo.Customer(CustomerID),
          InventoryID TINYINT NOT NULL REFERENCES dbo.Inventory(InventoryID),
           EmployeeID TINYINT NOT NULL REFERENCES dbo.Employee(EmployeeID),
    SaleDate DATE NOT NULL,
    SaleQuantity INT NOT NULL,
    SaleUnitPrice SMALLMONEY NOT NULL
);
```

3.

View All Tables in the Database

Definition: Use the system table to check the list of tables in the current database.

```
SELECT * FROM INFORMATION_SCHEMA.TABLES;
```

4.

View Specific Rows

Definition: Use `SELECT TOP` to retrieve a limited number of rows from a table.

Example: Get the first two rows of the `Customer` table.

```
SELECT TOP 2 * FROM dbo.Customer;
```

5.

View Specific Columns

Definition: Use `SELECT` with specific column names to fetch desired fields from a table.

Example: Get customer first and last names in descending order.

```
SELECT CustomerFirstName, CustomerLastName

FROM dbo.Customer

ORDER BY CustomerLastName DESC;
```

6.

Save Results to Another Table

Definition: Use `SELECT INTO` to create a new table and insert data into it.

Example: Save distinct customer last names into a new table.

```
SELECT DISTINCT CustomerLastName

INTO TempCustomer

FROM dbo.Customer

ORDER BY CustomerLastName;


-- View the new table

SELECT * FROM TempCustomer;
```

7. **Search Using LIKE**

   Definition: Use `LIKE` with wildcards to find patterns in string data.

   - `_` matches exactly one character.
   - `%` matches zero or more characters.

     Example: Find last names starting with "r".

```
SELECT * FROM dbo.Customer

WHERE CustomerLastName LIKE '_r%';
```

8.

Search Using IN

Definition: Use `IN` to filter rows based on multiple values in a column.

Example: Find customers with specific last names.

```
SELECT * FROM dbo.Customer

WHERE CustomerLastName IN ('Brown', 'Smith', 'Johnson');
```

9.

Filter Using Comparison Operators

Definition: Use `>` to find rows where a value is greater than a specified value.

Example: Find customers with last names alphabetically after "Brown".

```
SELECT * FROM dbo.Customer

WHERE CustomerLastName > 'Brown';
```

10.

Not Equal Operator (<>):
Definition: Use `<>` to exclude rows with a specific value in a column.
Example: Find all customers except those with the last name "Smith".

```
SELECT * FROM dbo.Customer

WHERE CustomerLastName <> 'Smith';
```

11.

Check for NULL Values
Definition: Use `IS NULL` to identify rows with missing data in a column.
Example: Find customers without a postcode.

```
SELECT * FROM dbo.Customer

WHERE CustomerPostCode IS NULL;
```

12.

Check for NOT NULL Values
Definition: Use `IS NOT NULL` to exclude rows with missing data in a column.

```
SELECT * FROM dbo.Customer

WHERE CustomerPostCode IS NOT NULL;
```

13.

Filter Using BETWEEN
Definition: Use `BETWEEN` to find rows within a range of values.

Example: Get sales where the unit price is between 5 and 10.

```
SELECT * FROM dbo.Sale

WHERE SaleUnitPrice BETWEEN 5 AND 10;
```

14.

Aggregate Function COUNT

Definition: Use `COUNT` to return the number of rows matching a condition.

Example: Count customers whose first names start with "B".

```
SELECT COUNT(*) AS [Number of Records]

FROM dbo.Customer

WHERE CustomerFirstName LIKE 'B%';
```

15.

Aggregate Function SUM

Definition: Use `SUM` to calculate the total of numeric values.

Example: Total sale quantities grouped by employees.

```
SELECT Sale.EmployeeID, EmployeeFirstName, EmployeeLastName,

       COUNT(*) AS [Number of Orders],

       SUM(SaleQuantity) AS [Total Quantity]

FROM dbo.Sale

JOIN dbo.Employee ON Sale.EmployeeID = Employee.EmployeeID
```

```
GROUP BY Sale.EmployeeID, EmployeeFirstName, EmployeeLastName;
```

16.

---

18. Find the Maximum Salary

Definition: Use `MAX` to find the highest value in a column.

Query:

```
SELECT MAX(Salary) AS [Highest Salary]
FROM dbo.EmployeeSalary;
```

Hypothetical Output:

Highest Salary
120,000

---

19. Find the Minimum Salary

Definition: Use `MIN` to find the smallest value in a column.

Query:

```
SELECT MIN(Salary) AS [Lowest Salary]
FROM dbo.EmployeeSalary;
```

Hypothetical Output:

Lowest Salary
40,000

---

20. Calculate the Average Salary

Definition: Use `AVG` to compute the average of numeric data in a column.

Query:

```
SELECT AVG(Salary) AS [Average Salary]
FROM dbo.EmployeeSalary;
```

Hypothetical Output:

Average Salary
80,000

---

21. Use HAVING with Aggregates

Definition: Filter grouped data using the `HAVING` clause.

Query 1: Count job titles with more than one employee.

```
SELECT JobTitle, COUNT(JobTitle) AS [Number of Employees]
FROM dbo.EmployeeDemographics
GROUP BY JobTitle
HAVING COUNT(JobTitle) > 1;
```

Hypothetical Output:

JobTitle	Number of Employees
Manager	3

Salesman	5

---

22. Change Data Types Using CAST/CONVERT

Definition: Convert one data type into another.

Query 1: Use `CAST` to convert a datetime to a date.

```
SELECT CAST('2024-12-11 14:30:00' AS DATE) AS [Converted Date];
```

Output:

Converted Date
2024-12-11

Query 2: Use `CONVERT` to achieve the same.

```
SELECT CONVERT(DATE, '2024-12-11 14:30:00') AS [Converted Date];
```

Output:

Converted Date

2024-12-11

---

23. Use CASE for Conditional Logic

Definition: The `CASE` statement allows conditional logic in SQL queries.

Query 1: Categorize employees by age.

```
SELECT FirstName, LastName, Age,
       CASE
           WHEN Age > 30 THEN 'Old'
           WHEN Age BETWEEN 27 AND 30 THEN 'Young'
           ELSE 'Baby'
       END AS [Age Category]
FROM dbo.EmployeeDemographics;
```

Hypothetical Output:

FirstName	LastName	Age	Age Category

John	Doe	35	Old
Jane	Smith	28	Young
Tim	Brown	25	Baby

---

24. Use PARTITION BY

Definition: Compute values for a specific group within the table.

Query: Count employees by gender.

```
SELECT FirstName, LastName, Gender, Salary,
          COUNT(Gender) OVER (PARTITION BY Gender) AS [Total by Gender]
FROM dbo.EmployeeDemographics;
```

Hypothetical Output:

FirstName	LastName	Gender	Salary	Total by Gender
John	Doe	Male	80,000	3

Tim	Brown	Male	90,000	3
Jane	Smith	Female	70,000	2

25. String Functions

Definition: SQL provides functions to manipulate and format strings, such as trimming spaces, replacing characters, and changing cases.

Query 1: Remove leading and trailing spaces using `LTRIM`, `RTRIM`, and `TRIM`.

```
SELECT EmployeeID,
       LTRIM(EmployeeID) AS [Left Trimmed ID],
       RTRIM(EmployeeID) AS [Right Trimmed ID],
       TRIM(EmployeeID) AS [Fully Trimmed ID]
FROM dbo.EmployeeErrors;
```

Hypothetical Output:

Employee ID	Left Trimmed ID	Right Trimmed ID	Fully Trimmed ID
" 1234 "	"1234 "	" 1234"	"1234"

Query 2: Replace substrings in a column using REPLACE.

```
SELECT LastName,
           REPLACE(LastName, '- Fired', '') AS [Fixed LastName]
FROM dbo.EmployeeErrors;
```

Hypothetical Output:

LastName	Fixed LastName
Smith-Fired	Smith
Brown	Brown

Query 3: Extract substrings using SUBSTRING.

```
SELECT SUBSTRING(FirstName, 1, 3) AS [First 3 Letters],
        SUBSTRING(LastName, 1, 3) AS [First 3 Letters of Last]
```

```
FROM dbo.EmployeeErrors;
```

Hypothetical Output:

First 3 Letters	First 3 Letters of Last
Joh	Doe
Jan	Smi

Query 4: Change case using `UPPER` and `LOWER`.

```
SELECT FirstName,
       UPPER(FirstName) AS [Upper Case],
       LOWER(FirstName) AS [Lower Case]
FROM dbo.EmployeeErrors;
```

Hypothetical Output:

FirstName	Upper Case	Lower Case
John	JOHN	john

Jane JANE jane

---

26. Stored Procedures

Definition: A stored procedure is a reusable SQL script that can accept parameters and return results.

Query: Create and execute a stored procedure to calculate employee statistics.

```
CREATE PROCEDURE Temp_Employee
@JobTitle NVARCHAR(100)
AS
BEGIN
    DROP TABLE IF EXISTS #temp_employee;
    CREATE TABLE #temp_employee (
        JobTitle NVARCHAR(100),
        EmployeesPerJob INT,
        AvgAge INT,
        AvgSalary MONEY
    );
```

```
    INSERT INTO #temp_employee
         SELECT  JobTitle,  COUNT(JobTitle),  AVG(Age),
AVG(Salary)
    FROM dbo.EmployeeDemographics emp
    JOIN dbo.EmployeeSalary sal
        ON emp.EmployeeID = sal.EmployeeID
    WHERE JobTitle = @JobTitle
    GROUP BY JobTitle;

    SELECT * FROM #temp_employee;
END;
GO

EXEC Temp_Employee @JobTitle = 'Salesman';
```

Hypothetical Output:

JobTitle	EmployeesPerJob	AvgAge	AvgSalary
Salesman	5	32	80,000

27. Subqueries

Definition: Subqueries allow nesting one query within another to refine results.

Query 1: Subquery in SELECT.

```
SELECT EmployeeID, Salary,
        (SELECT AVG(Salary) FROM dbo.EmployeeSalary) AS [All Avg Salary]
FROM dbo.EmployeeSalary;
```

Hypothetical Output:

Employee ID	Salary	All Avg Salary
1	80,000	70,000
2	60,000	70,000

Query 2: Subquery in FROM.

```
SELECT a.EmployeeID, a.Salary, a.AllAvgSalary
FROM (
     SELECT EmployeeID, Salary, AVG(Salary) OVER () AS AllAvgSalary
    FROM dbo.EmployeeSalary
) a
ORDER BY a.EmployeeID;
```

Hypothetical Output:

Employee ID	Salary	AllAvgSalary
1	80,000	70,000
2	60,000	70,000

Query 3: Subquery in `WHERE`.

```
SELECT EmployeeID, JobTitle, Salary
```

```
FROM dbo.EmployeeSalary
WHERE Salary IN (SELECT MAX(Salary) FROM dbo.EmployeeSalary);
```

Hypothetical Output:

Employee ID	JobTitle	Salary
5	Manager	120,000
