Seenu Ananth K

Infosys SpringBoard Intern

Beyond QWERTY – Form Filling Project

---

## Basic SQL Concepts

**1. What is SQL, and why is it important in data analytics?**
SQL (Structured Query Language) is the foundation of database management, allowing users to store, retrieve, manipulate, and analyze data. In data analytics, SQL is vital for querying databases to uncover trends, patterns, and insights.

**Example Table:**

`Sales`

| SaleID | Product | Amount | Region | Date |
|---|---|---|---|---|
| 1 | Laptop | 800 | North | 2024-01-10 |
| 2 | Smartphone | 500 | South | 2024-01-12 |
| 3 | Tablet | 300 | East | 2024-02-01 |

**Query:**

```
SELECT Region, SUM(Amount) AS TotalSales
FROM Sales
GROUP BY Region;
```

**Result:**

| Region | TotalSales |
|---|---|
| North | 800 |
| South | 500 |
| East | 300 |

---

**2. Explain the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN.**

- **INNER JOIN:** Returns matching records between tables.
- **LEFT JOIN:** Includes all records from the left table and matches from the right.
- **RIGHT JOIN:** Includes all records from the right table and matches from the left.
- **FULL OUTER JOIN:** Combines LEFT and RIGHT JOIN, showing all records.

**Example Tables:**

Employees

| EmpID | Name | DeptID |
|---|---|---|
| 1 | Alice | 101 |
| 2 | Bob | 102 |
| 3 | Charlie | NULL |

Departments

| DeptID | DeptName |
|---|---|
| 101 | HR |
| 102 | IT |
| 103 | Marketing |

**Queries and Results:**

1. **INNER JOIN:**

```
SELECT Name, DeptName
FROM Employees
INNER JOIN Departments
ON Employees.DeptID = Departments.DeptID;
```

| Name | DeptName |
|---|---|
| Alice | HR |
| Bob | IT |

2. **LEFT JOIN:**

```
SELECT Name, DeptName
FROM Employees
LEFT JOIN Departments
ON Employees.DeptID = Departments.DeptID;
```

| Name | DeptName |
|---|---|
| Alice | HR |
| Bob | IT |
| Charlie | NULL |

3. **RIGHT JOIN:**

```
SELECT Name, DeptName
FROM Employees
RIGHT JOIN Departments
ON Employees.DeptID = Departments.DeptID;
```

| Name | DeptName |
|---|---|
| Alice | HR |
| Bob | IT |

| Name | DeptName |
|---|---|
| NULL | Marketing |

4. **FULL OUTER JOIN:**

```
SELECT Name, DeptName
FROM Employees
FULL OUTER JOIN Departments
ON Employees.DeptID = Departments.DeptID;
```

| Name | DeptName |
|---|---|
| Alice | HR |
| Bob | IT |
| Charlie | NULL |
| NULL | Marketing |

---

### 3. What is the difference between WHERE and HAVING clauses?

- **WHERE:** Filters rows before grouping.
- **HAVING:** Filters rows after grouping.

**Example Table:**
`Orders`

| OrderID | Customer | Amount | Status |
|---|---|---|---|
| 1 | John | 100 | Paid |
| 2 | Alice | 200 | Pending |
| 3 | Bob | 100 | Paid |

**Query Using WHERE:**

```
SELECT *
FROM Orders
WHERE Status = 'Paid';
```

**Result:**

| OrderID | Customer | Amount | Status |
|---|---|---|---|
| 1 | John | 100 | Paid |
| 3 | Bob | 100 | Paid |

**Query Using HAVING:**

```
SELECT Status, SUM(Amount) AS TotalAmount
FROM Orders
GROUP BY Status
HAVING SUM(Amount) > 150;
```

**Result:**

| Status | TotalAmount |
|---|---|
| Pending | 200 |

---

**4. How do you use GROUP BY and HAVING in a query?**
**GROUP BY** organizes data into groups; **HAVING** applies filters to these groups.

**Example Table:**
`Sales`

| SaleID | Product | Amount | Region |
|---|---|---|---|
| 1 | Laptop | 800 | North |
| 2 | Smartphone | 500 | South |
| 3 | Tablet | 300 | North |

**Query:**

```
SELECT Region, SUM(Amount) AS TotalSales
FROM Sales
GROUP BY Region
HAVING SUM(Amount) > 500;
```

**Result:**

| Region | TotalSales |
|---|---|
| North | 1100 |

---

**5. Write a query to find duplicate records in a table.**
Identify records with the same values in specific columns.

**Example Table:**
`Users`

| UserID | Name | Email |
|---|---|---|
| 1 | Alice | alice@example.com |
| 2 | Bob | bob@example.com |
| 3 | Alice | alice@example.com |

**Query:**

```
SELECT Name, Email, COUNT(*) AS DuplicateCount
FROM Users
GROUP BY Name, Email
HAVING COUNT(*) > 1;
```

**Result:**

| Name | Email | DuplicateCount |
|---|---|---|
| Alice | alice@example.com | 2 |

**6. How do you retrieve unique values from a table using SQL?**
The `DISTINCT` keyword is used to eliminate duplicate rows and retrieve unique values.

**Example Table:**
`Products`

| ProductID | Category | Price |
|---|---|---|
| 1 | Electronics | 800 |
| 2 | Clothing | 50 |
| 3 | Electronics | 800 |

**Query:**

```
SELECT DISTINCT Category
FROM Products;
```

**Result:**

| Category |
|---|
| Electronics |
| Clothing |

---

**7. Explain the use of aggregate functions like COUNT(), SUM(), AVG(), MIN(), and MAX().**

Aggregate functions perform calculations on data sets and return a single value.

- **`COUNT()`**: Counts rows.
- **`SUM()`**: Adds values.
- **`AVG()`**: Calculates average.
- **`MIN()`**: Finds smallest value.
- **`MAX()`**: Finds largest value.

**Example Table:**
`Sales`

| SaleID | Product | Amount |
|---|---|---|
| 1 | Laptop | 800 |
| 2 | Smartphone | 500 |
| 3 | Tablet | 300 |

**Query Example:**

1. **COUNT:**

```
SELECT COUNT(*) AS TotalSales
FROM Sales;
```

**Result:**

| TotalSales |
|---|
| 3 |

2. **SUM:**

```
SELECT SUM(Amount) AS TotalRevenue
FROM Sales;
```

**Result:**

| TotalRevenue |
|---|
| 1600 |

3. **AVG:**

```
SELECT AVG(Amount) AS AverageSale
FROM Sales;
```

**Result:**

| AverageSale |
|---|
| 533.33 |

4. **MIN and MAX:**

```
SELECT MIN(Amount) AS LowestSale, MAX(Amount) AS HighestSale
FROM Sales;
```

**Result:**

| LowestSale | HighestSale |
|---|---|
| 300 | 800 |

---

**8. What is the purpose of a DISTINCT keyword in SQL?**
The `DISTINCT` keyword ensures unique results by removing duplicate rows from the output.

**Example Table:**
`Customers`

| CustomerID | Name | Country |
|---|---|---|
| 1 | John | USA |
| 2 | Alice | India |
| 3 | John | USA |

**Query:**

```
SELECT DISTINCT Name
FROM Customers;
```

**Result:**

| Name |
|---|
| John |
| Alice |

---

## Intermediate SQL

**1. Write a query to find the second-highest salary from an employee table.**
Use the `LIMIT` clause or a subquery to achieve this.

**Example Table:**
`Employees`

| EmpID | Name | Salary |
|---|---|---|
| 1 | Alice | 6000 |
| 2 | Bob | 8000 |
| 3 | Charlie | 7000 |

**Query Using Subquery:**

```
SELECT MAX(Salary) AS SecondHighestSalary
FROM Employees
WHERE Salary < (SELECT MAX(Salary) FROM Employees);
```

**Result:**

| SecondHighestSalary |
|---|
| 7000 |

---

**2. What are subqueries, and how do you use them?**

Subqueries are nested queries within another SQL query. They are used to perform operations that depend on query results.

**Example Table:**
`Orders`

| OrderID | Customer | Amount |
|---|---|---|
| 1 | John | 100 |
| 2 | Alice | 200 |
| 3 | Bob | 300 |

**Query:**
Find customers with orders above the average order amount.

```
SELECT Customer, Amount
FROM Orders
WHERE Amount > (SELECT AVG(Amount) FROM Orders);
```

**Result:**

| Customer | Amount |
|---|---|
| Bob | 300 |

---

**3. What is a Common Table Expression (CTE)? Give an example of when to use it.**
CTEs are temporary result sets named within a `WITH` clause for easier readability and modularization of complex queries.

**Example Table:**
`Sales`

| SaleID | Product | Amount | Region |
|---|---|---|---|
| 1 | Laptop | 800 | North |
| 2 | Smartphone | 500 | South |
| 3 | Tablet | 300 | North |

**Query:**

```
WITH RegionalSales AS (
    SELECT Region, SUM(Amount) AS TotalSales
    FROM Sales
    GROUP BY Region
)
SELECT Region
FROM RegionalSales
WHERE TotalSales > 500;
```

**Result:**

| Region |
|---|
| North |

---

**4. Explain window functions like ROW_NUMBER(), RANK(), and DENSE_RANK().**

Window functions operate over a subset of rows, defined by the `OVER()` clause, without collapsing the result set.

**Example Table:**
`Sales`

| SaleID | Product | Amount |
|---|---|---|
| 1 | Laptop | 800 |
| 2 | Smartphone | 800 |
| 3 | Tablet | 500 |

**Query:**

```
SELECT Product, Amount,
       ROW_NUMBER() OVER (ORDER BY Amount DESC) AS RowNumber,
       RANK() OVER (ORDER BY Amount DESC) AS RankNumber,
       DENSE_RANK() OVER (ORDER BY Amount DESC) AS DenseRank
FROM Sales;
```

**Result:**

| Product | Amount | RowNumber | RankNumber | DenseRank |
|---|---|---|---|---|
| Laptop | 800 | 1 | 1 | 1 |
| Smartphone | 800 | 2 | 1 | 1 |
| Tablet | 500 | 3 | 3 | 2 |

**5. How do you combine results of two queries using UNION and UNION ALL?**
`UNION` combines the results of two queries while removing duplicates. `UNION ALL` retains all rows, including duplicates.

**Example Tables:**

`TableA`

| ID | Name |
|---|---|
| 1 | Alice |
| 2 | Bob |

`TableB`

| ID | Name |
|---|---|
| 2 | Bob |
| 3 | Charlie |

**Query Using UNION:**

```
SELECT Name FROM TableA
UNION
SELECT Name FROM TableB;
```

**Result (No Duplicates):**

| Name |
|---|
| Alice |
| Bob |
| Charlie |

**Query Using UNION ALL:**

```
SELECT Name FROM TableA
UNION ALL
SELECT Name FROM TableB;
```

**Result (Includes Duplicates):**

| Name |
|---|
| Alice |
| Bob |
| Bob |
| Charlie |

---

**6. What are indexes in SQL, and how do they improve query performance?**
Indexes are special lookup tables that the database uses to speed up data retrieval. They reduce the time complexity of queries but may increase the cost of data modifications.

**Example Table:**
`Employees`

| EmpID | Name | Department |
|---|---|---|
| 1 | Alice | HR |
| 2 | Bob | IT |
| 3 | Charlie | HR |

**Creating an Index:**

```
CREATE INDEX idx_department ON Employees(Department);
```

**Query Before Index:**

```
SELECT * FROM Employees WHERE Department = 'HR';
```

**Result Without Index:**
Full table scan, slower retrieval.

**Query With Index:**
Indexes enable faster retrieval by directly locating rows in the "HR" department.

---

### 7. Write a query to calculate the total sales for each month using GROUP BY.

Use the `GROUP BY` clause to summarize data, grouping rows based on a column (e.g., `Month`).

**Example Table:**
`Sales`

| SaleID | Month | Amount |
|---|---|---|
| 1 | January | 100 |
| 2 | January | 200 |
| 3 | February | 150 |

**Query:**

```
SELECT Month, SUM(Amount) AS TotalSales
FROM Sales
GROUP BY Month;
```

**Result:**

| Month | TotalSales |
|---|---|
| January | 300 |
| February | 150 |

---

## Advanced SQL

### 1. How do you optimize a slow-running SQL query?

Optimizing queries involves:

1. Using indexes for frequently queried columns.
2. Avoiding `SELECT *` by selecting only necessary columns.
3. Reducing joins and subqueries where possible.
4. Using appropriate data types and constraints.
5. Analyzing query execution plans (`EXPLAIN` or `ANALYZE` commands).

**Example:**

**Slow Query:**

```
SELECT * FROM Employees WHERE Department = 'HR';
```

**Optimized Query:**

1. Add an index:

```
CREATE INDEX idx_department ON Employees(Department);
```

2. Use specific columns:

```
SELECT EmpID, Name FROM Employees WHERE Department = 'HR';
```

---

**2. What are views in SQL, and when would you use them?**
Views are virtual tables created from a query. They simplify complex queries and improve security by restricting access to specific data.

**Example Table:**
`Orders`

| OrderID | Customer | Amount |
|---|---|---|
| 1 | John | 100 |
| 2 | Alice | 200 |

**Create a View:**

```
CREATE VIEW HighValueOrders AS
SELECT * FROM Orders WHERE Amount > 150;
```

**Query the View:**

```
SELECT * FROM HighValueOrders;
```

**Result:**

| OrderID | Customer | Amount |
|---|---|---|
| 2 | Alice | 200 |

**3. What is the difference between a stored procedure and a function in SQL?**

Stored procedures and functions are reusable blocks of code in SQL, but they differ in their purpose and behavior.

| Feature | Stored Procedure | Function |
|---|---|---|
| **Purpose** | Perform multiple operations, including DML. | Perform calculations and return a value. |
| **Return Type** | Can return zero, one, or multiple values. | Returns a single value (scalar or table). |
| **Use in Queries** | Cannot be directly used in SELECT statements. | Can be used in SELECT statements. |

| Feature | Stored Procedure | Function |
|---|---|---|
| **Parameters** | Supports input, output, and in-out params. | Supports only input parameters. |
| **Transaction Control** | Can have COMMIT and ROLLBACK statements. | Cannot have transaction control statements. |

---

**Example Table:**

`Sales`

| SaleID | Product | Amount |
|---|---|---|
| 1 | Laptop | 2000 |
| 2 | Phone | 800 |

---

**Stored Procedure Example:**

Calculate total sales of a product.

```
CREATE PROCEDURE GetTotalSales (IN productName VARCHAR(50))
BEGIN
    SELECT SUM(Amount) AS TotalSales
    FROM Sales
    WHERE Product = productName;
END;
```

**Execution:**

```
CALL GetTotalSales('Laptop');
```

**Result:**

| TotalSales |
|---|
| 2000 |

---

**Function Example:**

Calculate the tax for a given amount.

```
CREATE FUNCTION CalculateTax(amount DECIMAL(10, 2)) RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    RETURN amount * 0.18; -- Assuming 18% tax
END;
```

**Usage in Query:**

```
SELECT SaleID, Product, Amount, CalculateTax(Amount) AS Tax FROM Sales;
```

**Result:**

| SaleID | Product | Amount | Tax |
|---|---|---|---|
| 1 | Laptop | 2000 | 360.00 |
| 2 | Phone | 800 | 144.00 |

**4. Explain the difference between TRUNCATE, DELETE, and DROP commands.**

TRUNCATE, DELETE, and DROP are SQL commands used to remove data, but they differ in their purpose, functionality, and performance.

| Command | Purpose | Behavior | Rollback Possible | Use Case |
|---|---|---|---|---|
| **TRUNCATE** | Removes all rows from a table. | Resets table to its initial empty state. | No | Clear data but keep table. |
| **DELETE** | Removes specific rows from a table. | Can filter rows using a WHERE clause. | Yes | Delete selected data only. |
| **DROP** | Deletes the table structure and data. | Completely removes the table from schema. | No | Remove table permanently. |

---

**Example Table:**
`Products`

| ProductID | ProductName | Price |
|---|---|---|
| 1 | Laptop | 2000 |
| 2 | Phone | 800 |
| 3 | Tablet | 500 |

---

**TRUNCATE Example:**

```
TRUNCATE TABLE Products;
```

**Result:** Table exists, but no rows are present.

| ProductID | ProductName | Price |
|---|---|---|

---

**DELETE Example:**

```
DELETE FROM Products WHERE Price > 1000;
```

**Result:**

| ProductID | ProductName | Price |
|---|---|---|
| 2 | Phone | 800 |
| 3 | Tablet | 500 |

---

**DROP Example:**

```
DROP TABLE Products;
```

**Result:** The table `Products` no longer exists.

---

**5. What are windowing functions, and how are they used in analytics?**

Windowing functions perform calculations across a set of table rows that are related to the current row, allowing for advanced analytics and reporting.

**Common Window Functions:**

- **ROW_NUMBER():** Assigns a unique number to each row within a partition.
- **SUM():** Provides cumulative totals.
- **AVG():** Calculates moving averages.

---

**Example Table:**
`Sales`

| SaleID | Product | Amount | Region |
|---|---|---|---|
| 1 | Laptop | 2000 | East |
| 2 | Phone | 800 | East |
| 3 | Laptop | 1500 | West |
| 4 | Phone | 900 | West |

---

**Window Function Query:**
Calculate the running total of sales by region.

```
SELECT Region, Product, Amount, SUM(Amount) OVER (PARTITION BY Region ORDER
BY SaleID) AS RunningTotal
FROM Sales;
```

**Result:**

| Region | Product | Amount | RunningTotal |
|---|---|---|---|
| East | Laptop | 2000 | 2000 |
| East | Phone | 800 | 2800 |
| West | Laptop | 1500 | 1500 |
| West | Phone | 900 | 2400 |

---

### 6. How do you use PARTITION BY and ORDER BY in window functions?

`PARTITION BY` divides the result set into groups, and `ORDER BY` defines the order of rows within each partition.

---

**Example Table:**
`Sales`

| SaleID | Product | Amount | Region |
|---|---|---|---|
| 1 | Laptop | 2000 | East |
| 2 | Phone | 800 | East |
| 3 | Laptop | 1500 | West |
| 4 | Phone | 900 | West |

---

**Query:** Calculate the rank of sales within each region by amount.

```
SELECT Region, Product, Amount, RANK() OVER (PARTITION BY Region ORDER BY
Amount DESC) AS SalesRank
FROM Sales;
```

**Result:**

| Region | Product | Amount | SalesRank |
|---|---|---|---|
| East | Laptop | 2000 | 1 |
| East | Phone | 800 | 2 |
| West | Laptop | 1500 | 1 |
| West | Phone | 900 | 2 |

### 7. How do you handle NULL values in SQL, and what functions help with that (e.g., COALESCE, ISNULL)?

Handling `NULL` values is essential in SQL to ensure accuracy in data processing and reporting. `NULL` represents missing or unknown data, and functions like `COALESCE` and `ISNULL` help manage these scenarios effectively.

---

### Common Techniques to Handle `NULL`:

| Function/Technique | Purpose | Example Usage |
|---|---|---|
| **IS NULL / IS NOT NULL** | Check if a value is `NULL` or not. | `SELECT * FROM Products WHERE Price IS NULL;` |
| **COALESCE()** | Replaces `NULL` with the first non-NULL value. | `COALESCE(Discount, 0)` |
| **ISNULL()** | Similar to `COALESCE`, replaces `NULL` with a value. | `ISNULL(Price, 0)` |
| **NULLIF()** | Returns `NULL` if two expressions are equal. | `NULLIF(SalePrice, CostPrice)` |

---

### Example Table:

`Products`

| ProductID | ProductName | Price | Discount |
|---|---|---|---|
| 1 | Laptop | 2000 | NULL |
| 2 | Phone | 800 | 10 |
| 3 | Tablet | NULL | NULL |

---

**Example Queries:**

### 1. Use IS NULL to Find Missing Prices:

```
SELECT ProductID, ProductName
FROM Products
WHERE Price IS NULL;
```

**Result:**

| ProductID | ProductName |
|---|---|
| 3 | Tablet |

---

### 2. Use COALESCE to Replace NULL Values:

```
SELECT ProductID, ProductName, COALESCE(Price, 0) AS Price
FROM Products;
```

**Result:**

| ProductID | ProductName | Price |
|---|---|---|
| 1 | Laptop | 2000 |
| 2 | Phone | 800 |
| 3 | Tablet | 0 |

---

### 3. Use NULLIF to Compare Two Columns:

```
SELECT ProductID, NULLIF(Price, Discount) AS EffectivePrice
FROM Products;
```

**Result:**

| ProductID | EffectivePrice |
|---|---|
| 1 | 2000 |
| 2 | NULL |
| 3 | NULL |