T-SQL Comprehensive Guide: Functions and Statements

Set Operations

UNION

Definition: Combines result sets from two or more SELECT statements, removing duplicates.

Uses: Merging data from multiple tables with similar structures.

Efficiency: Moderate - requires sorting and duplicate removal.

When to use: When you need unique records from multiple sources.

Syntax:

```sql
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
```

Example:

```sql
SELECT Name, City FROM Customers
UNION
SELECT Name, City FROM Suppliers;
```

Practice Question: Write a query to get all unique product names from both 'Products' and 'ArchivedProducts' tables.

UNION ALL

Definition: Combines result sets from two or more SELECT statements, including duplicates.

Uses: Merging data when duplicates are needed or performance is critical.

Efficiency: High - no sorting or duplicate removal needed.

When to use: When duplicates are acceptable or required.

Syntax:

```sql
SELECT column1, column2 FROM table1
UNION ALL
SELECT column1, column2 FROM table2;
```

Example:

```sql
SELECT ProductName, Price FROM Products
UNION ALL
SELECT ProductName, Price FROM DiscontinuedProducts;
```

Practice Question: Create a report showing all sales transactions from both 'Sales2023' and 'Sales2024' tables, including duplicate entries.

INTERSECT

Definition: Returns only rows that exist in both result sets.

Uses: Finding common records between datasets.

Efficiency: Moderate - requires comparison of all rows.

When to use: When you need records that exist in multiple tables.

Syntax:

```sql
SELECT column1, column2 FROM table1
INTERSECT
SELECT column1, column2 FROM table2;
```

Example:

```sql
SELECT CustomerID FROM Orders
INTERSECT
SELECT CustomerID FROM Returns;
```

Practice Question: Find customers who have placed orders in both Q1 and Q2 of 2024.

EXCEPT / MINUS

Definition: Returns rows from the first result set that don't exist in the second.

Uses: Finding differences between datasets.

Efficiency: Moderate - requires comparison operations.

When to use: When you need records that exist in one table but not another.

Syntax:

```sql
SELECT column1, column2 FROM table1
EXCEPT
SELECT column1, column2 FROM table2;
```

Example:

```sql
SELECT CustomerID FROM Customers
EXCEPT
SELECT CustomerID FROM Orders;
```

Practice Question: Identify products that are in inventory but have never been sold.

Data Modification

INSERT Statement

Definition: Adds new rows to a table.

Uses: Adding new data records.

Efficiency: High for single rows, bulk operations for multiple rows.

When to use: When adding new data to tables.

Syntax:

```sql
INSERT INTO table_name (column1, column2) VALUES (value1, value2);
INSERT INTO table_name SELECT column1, column2 FROM source_table;
```

Example:

```sql
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('John Doe', 'New York', 'USA');

INSERT INTO CustomerBackup
SELECT * FROM Customers WHERE Country = 'USA';
```

Practice Question: Insert a new employee record with ID 1001, Name 'Sarah Wilson', Department 'IT', and Salary 75000 into the Employees table.

UPDATE Statement

Definition: Modifies existing data in a table.

Uses: Changing existing records.

Efficiency: Depends on WHERE clause and indexes.

When to use: When modifying existing data.

Syntax:

```sql
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE condition;
```

Example:

```sql
UPDATE Employees
SET Salary = Salary * 1.10
WHERE Department = 'Sales';
```

Practice Question: Update all products in the 'Electronics' category to have a 15% price increase.

DELETE Statement

Definition: Removes rows from a table.

Uses: Removing unwanted data.

Efficiency: Depends on WHERE clause and indexes.

When to use: When removing specific records.

Syntax:

```sql
DELETE FROM table_name WHERE condition;
```

Example:

```sql
DELETE FROM Orders
WHERE OrderDate < '2023-01-01';
```

Practice Question: Delete all customers who haven't placed any orders in the last 2 years.

TRUNCATE Statement

Definition: Removes all rows from a table quickly.

Uses: Clearing entire tables.

Efficiency: Very high - minimal logging.

When to use: When removing all data from a table.

Syntax:

```sql
TRUNCATE TABLE table_name;
```

Example:

```sql
TRUNCATE TABLE TempData;
```

Practice Question: Clear all data from a staging table called 'DataImport' after processing.

String Functions

CONCAT

Definition: Joins two or more strings together.

Uses: Combining text values.

Efficiency: High for small strings.

When to use: When combining multiple string values.

Syntax:

```sql
CONCAT(string1, string2, ...)
```

Example:

```sql
SELECT CONCAT(FirstName, ' ', LastName) AS FullName FROM Employees;
```

Practice Question: Create a full address by concatenating Address, City, State, and ZipCode with appropriate separators.

SUBSTRING

Definition: Extracts a portion of a string.

Uses: Getting part of a string value.

Efficiency: High.

When to use: When you need only part of a string.

Syntax:

```sql
SUBSTRING(string, start_position, length)
```

Example:

```sql
SELECT SUBSTRING(ProductCode, 1, 3) AS Category FROM Products;
```

Practice Question: Extract the first 3 characters from employee IDs to determine their department code.

LENGTH / LEN

Definition: Returns the number of characters in a string.

Uses: Measuring string length.

Efficiency: High.

When to use: When you need to know string length.

Syntax:

```sql
LEN(string)  -- T-SQL uses LEN instead of LENGTH
```

Example:

```sql
SELECT ProductName, LEN(ProductName) AS NameLength FROM Products;
```

Practice Question: Find all customers whose names are longer than 20 characters.

UPPER / LOWER

Definition: Converts string to uppercase or lowercase.

Uses: Standardizing text case.

Efficiency: High.

When to use: When normalizing text data.

Syntax:

```sql
UPPER(string)
LOWER(string)
```

Example:

```sql
SELECT UPPER(CustomerName) AS UpperName, LOWER(Email) AS LowerEmail
FROM Customers;
```

Practice Question: Standardize all email addresses to lowercase and customer names to proper case.

TRIM

Definition: Removes leading and trailing spaces.

Uses: Cleaning string data.

Efficiency: High.

When to use: When cleaning imported data.

Syntax:

```sql
TRIM(string)
LTRIM(string)  -- Left trim
RTRIM(string)  -- Right trim
```

Example:

```sql
SELECT TRIM(CustomerName) AS CleanName FROM Customers;
```

Practice Question: Clean all product names by removing leading and trailing spaces.

REPLACE

Definition: Replaces occurrences of a substring with another substring.

Uses: Text substitution.

Efficiency: Moderate for large strings.

When to use: When replacing specific text patterns.

Syntax:

```sql
REPLACE(string, old_substring, new_substring)
```

Example:

```sql
SELECT REPLACE(PhoneNumber, '-', '') AS CleanPhone FROM Customers;
```

Practice Question: Replace all instances of 'Inc.' with 'Incorporated' in company names.

LEFT / RIGHT

Definition: Returns specified number of characters from left or right side of string.

Uses: Extracting characters from string ends.

Efficiency: High.

When to use: When you need characters from string ends.

Syntax:

```sql
LEFT(string, number_of_characters)
RIGHT(string, number_of_characters)
```

Example:

```sql
SELECT LEFT(ProductCode, 2) AS Category, RIGHT(ProductCode, 3) AS ID
FROM Products;
```

Practice Question: Extract the last 4 digits of credit card numbers for display purposes.

CHARINDEX

Definition: Returns the position of a substring within a string.

Uses: Finding substring positions.

Efficiency: High.

When to use: When locating text within strings.

Syntax:

```sql
CHARINDEX(substring, string, start_position)
```

Example:

```sql
SELECT CustomerName, CHARINDEX('@', Email) AS AtPosition FROM Customers;
```

Practice Question: Find the position of the first space in customer names to separate first and last names.

Date/Time Functions

GETDATE() / CURRENT_TIMESTAMP

Definition: Returns current date and time.

Uses: Getting current timestamp.

Efficiency: High.

When to use: When you need current date/time.

Syntax:

```sql
GETDATE()
CURRENT_TIMESTAMP
```

Example:

```sql
SELECT OrderID, OrderDate, GETDATE() AS ProcessedTime FROM Orders;
```

Practice Question: Add a timestamp to show when each record was last updated.

DATEADD

Definition: Adds a specified time interval to a date.

Uses: Date arithmetic.

Efficiency: High.

When to use: When calculating future or past dates.

Syntax:

```sql
DATEADD(datepart, number, date)
```

Example:

```sql
SELECT OrderDate, DATEADD(DAY, 30, OrderDate) AS DueDate FROM Orders;
```

Practice Question: Calculate the expiration date for products (30 days from manufacture date).

DATEDIFF

Definition: Returns the difference between two dates.

Uses: Calculating time intervals.

Efficiency: High.

When to use: When measuring time between dates.

Syntax:

```sql
DATEDIFF(datepart, startdate, enddate)
```

Example:

```sql
SELECT CustomerID, DATEDIFF(DAY, OrderDate, ShipDate) AS ProcessingDays
FROM Orders;
```

Practice Question: Calculate the number of days between order date and delivery date for all orders.

DATEPART / EXTRACT

Definition: Extracts a specific part of a date.

Uses: Getting specific date components.

Efficiency: High.

When to use: When you need specific date parts.

Syntax:

```sql
DATEPART(datepart, date)
YEAR(date), MONTH(date), DAY(date)  -- Specific functions
```

Example:

```sql
SELECT OrderDate, YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth
FROM Orders;
```

Practice Question: Group sales by quarter and year to analyze seasonal trends.

FORMAT

Definition: Formats a date according to specified format.

Uses: Date formatting for display.

Efficiency: Moderate.

When to use: When presenting dates in specific formats.

Syntax:

```sql
FORMAT(date, format_string)
```

Example:

```sql
SELECT OrderDate, FORMAT(OrderDate, 'MMM dd, yyyy') AS FormattedDate
FROM Orders;
```

Practice Question: Display all birth dates in 'Month Day, Year' format.

Null Handling Functions

ISNULL

Definition: Replaces NULL with a specified value.

Uses: Handling NULL values.

Efficiency: High.

When to use: When you need to replace NULLs with default values.

Syntax:

```sql
ISNULL(expression, replacement_value)
```

Example:

```sql
SELECT ProductName, ISNULL(UnitsInStock, 0) AS Stock FROM Products;
```

Practice Question: Replace NULL values in the MiddleName column with an empty string.

COALESCE

Definition: Returns the first non-NULL expression from a list.

Uses: Finding first non-NULL value.

Efficiency: High.

When to use: When checking multiple columns for non-NULL values.

Syntax:

```sql
COALESCE(expression1, expression2, ...)
```

Example:

```sql
SELECT COALESCE(WorkPhone, HomePhone, CellPhone, 'No Phone') AS ContactPhone
FROM Customers;
```

Practice Question: Create a priority contact method using Email, Phone, and Fax in order of preference.

Conversion Functions

CAST

Definition: Converts an expression from one data type to another.

Uses: Data type conversion.

Efficiency: High.

When to use: When explicit type conversion is needed.

Syntax:

```sql
CAST(expression AS target_datatype)
```

Example:

```sql
SELECT OrderID, CAST(OrderDate AS VARCHAR(10)) AS OrderDateString
FROM Orders;
```

Practice Question: Convert all salary values to strings with currency symbol for display.

CONVERT

Definition: Converts expressions between data types with formatting options.

Uses: Data type conversion with formatting.

Efficiency: High.

When to use: When you need specific formatting during conversion.

Syntax:

```sql
CONVERT(target_datatype, expression, style)
```

Example:

```sql
SELECT OrderDate, CONVERT(VARCHAR(10), OrderDate, 101) AS USFormat
FROM Orders;
```

Practice Question: Convert datetime values to different regional date formats.

NULLIF

Definition: Returns NULL if two expressions are equal, otherwise returns the first expression.

Uses: Converting specific values to NULL.

Efficiency: High.

When to use: When you want to treat certain values as NULL.

Syntax:

```sql
NULLIF(expression1, expression2)
```

Example:

```sql
SELECT ProductName, NULLIF(UnitsInStock, 0) AS AvailableStock FROM Products;
```

Practice Question: Convert zero values in the Discount column to NULL for percentage calculations.

Conditional Functions

CASE Statement

Definition: Provides conditional logic in SQL queries.

Uses: Implementing if-then-else logic.

Efficiency: High.

When to use: When you need conditional logic in queries.

Syntax:

```sql
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE result
END
```

Example:

```sql
SELECT ProductName, UnitPrice,
    CASE
        WHEN UnitPrice > 50 THEN 'Expensive'
        WHEN UnitPrice > 20 THEN 'Moderate'
        ELSE 'Affordable'
    END AS PriceCategory
FROM Products;
```

Practice Question: Categorize employees by salary ranges: 'Entry Level' (<40K), 'Mid Level' (40K-80K), 'Senior Level' (>80K).

IIF (Inline IF)

Definition: Returns one of two values based on a Boolean expression.

Uses: Simple conditional logic.

Efficiency: High.

When to use: For simple true/false conditions.

Syntax:

```sql
IIF(boolean_expression, true_value, false_value)
```

Example:

```sql
SELECT ProductName, IIF(UnitsInStock > 0, 'In Stock', 'Out of Stock') AS Status
FROM Products;
```

Practice Question: Create a column showing 'Active' or 'Inactive' based on employee status.

Window Functions (Analytical)

ROW_NUMBER

Definition: Assigns a unique sequential number to rows within a result set.

Uses: Ranking and pagination.

Efficiency: High with proper indexing.

When to use: When you need unique row numbers.

Syntax:

```sql
ROW_NUMBER() OVER (PARTITION BY column ORDER BY column)
```

Example:

```sql
SELECT CustomerID, OrderDate, OrderTotal,
    ROW_NUMBER() OVER (PARTITION BY CustomerID ORDER BY OrderDate) AS OrderSequence
FROM Orders;
```

Practice Question: Number each employee's projects in chronological order of assignment.

RANK

Definition: Assigns ranks to rows, with gaps for tied values.

Uses: Ranking with tied values.

Efficiency: High with proper indexing.

When to use: When you need rankings that handle ties with gaps.

Syntax:

```sql
RANK() OVER (PARTITION BY column ORDER BY column)
```

Example:

```sql
SELECT StudentName, TestScore,
    RANK() OVER (ORDER BY TestScore DESC) AS Rank
FROM TestResults;
```

Practice Question: Rank salespersons by their total sales, showing tied rankings appropriately.

DENSE_RANK

Definition: Assigns ranks to rows without gaps for tied values.

Uses: Ranking without gaps.

Efficiency: High with proper indexing.

When to use: When you need consecutive rankings despite ties.

Syntax:

```sql
DENSE_RANK() OVER (PARTITION BY column ORDER BY column)
```

Example:

```sql
SELECT ProductName, UnitsSold,
    DENSE_RANK() OVER (ORDER BY UnitsSold DESC) AS SalesRank
FROM ProductSales;
```

Practice Question: Rank products by popularity ensuring consecutive rank numbers.

LAG

Definition: Accesses data from a previous row in the result set.

Uses: Comparing current row with previous row.

Efficiency: High.

When to use: When you need to compare with previous values.

Syntax:

```sql
LAG(column, offset, default) OVER (PARTITION BY column ORDER BY column)
```

Example:

```sql
SELECT SaleDate, SaleAmount,
    LAG(SaleAmount, 1, 0) OVER (ORDER BY SaleDate) AS PreviousSale,
    SaleAmount - LAG(SaleAmount, 1, 0) OVER (ORDER BY SaleDate) AS Difference
FROM Sales;
```

Practice Question: Calculate month-over-month sales growth by comparing current month with previous month.

LEAD

Definition: Accesses data from a subsequent row in the result set.

Uses: Comparing current row with next row.

Efficiency: High.

When to use: When you need to compare with following values.

Syntax:

```sql
LEAD(column, offset, default) OVER (PARTITION BY column ORDER BY column)
```

Example:

```sql
SELECT EmployeeID, SalaryDate, Salary,
    LEAD(Salary, 1, 0) OVER (PARTITION BY EmployeeID ORDER BY SalaryDate) AS NextSalary
FROM SalaryHistory;
```

Practice Question: Identify employees who will receive salary increases by comparing current salary with next scheduled salary.

Complex Practice Scenarios

Scenario 1: Sales Analysis

Create a comprehensive sales report that shows:

- Monthly sales totals with year-over-year comparison
- Ranking of products by sales volume
- Customer segmentation based on purchase behavior
- Identification of seasonal trends

Scenario 2: Employee Management

Develop queries for:

- Employee performance ranking within departments

- Salary analysis with percentile calculations
- Career progression tracking
- Department efficiency metrics

Scenario 3: Inventory Optimization

Build a system to:

- Track product movement patterns
- Identify slow-moving inventory
- Calculate reorder points based on historical data
- Analyze supplier performance

These scenarios combine multiple functions and concepts to solve real-world business problems, demonstrating the practical application of T-SQL functions in enterprise environments.