SQL Server

Introduction

Ahmed Fekry
ahmedfikry78@gmail.com
+201120590421

SQL Functions

1. Introduction

- Brief overview of SQL functions
- Importance of using functions in SQL

SQL Functions

2. String Functions

2.1. CONCAT

- **Description**: Concatenates two or more strings.
- **Syntax**: `CONCAT(string1, string2, ...)`

sql

Copy code

```sql
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM sales.customers;
```

SQL Functions

2.2. LENGTH

- **Description**: Returns the length of a string.
- **Syntax**: `LENGTH(string)`

sql

Copy code

```sql
SELECT LENGTH(first_name) AS name_length
FROM sales.customers;
```

SQL Functions

2.3. SUBSTRING

- **Description**: Extracts a substring from a string.
- **Syntax**: `SUBSTRING(string, start, length)`

sql

Copy code

```sql
SELECT SUBSTRING(first_name, 1, 3) AS short_name
FROM sales.customers;
```

SQL Functions

2.4. UPPER and LOWER

- **Description**: Converts a string to uppercase or lowercase.
- **Syntax**: `UPPER(string)` / `LOWER(string)`

sql

Copy code

```sql
SELECT UPPER(first_name) AS upper_name, LOWER(last_name) AS lower_name
FROM sales.customers;
```

SQL Functions

3. Date and Time Functions

3.1. GETDATE

- **Description**: Returns the current date and time.
- **Syntax**: `GETDATE()`

sql

Copy code

```sql
SELECT GETDATE() AS current_date_time;
```

SQL Functions

3.2. DATEADD

- **Description**: Adds a specified number of units to a date.
- **Syntax**: `DATEADD(unit, number, date)`

sql

Copy code

```sql
SELECT DATEADD(day, 10, order_date) AS new_date
FROM sales.orders;
```

SQL Functions

3.4. FORMAT

- **Description**: Formats a date/time value according to a specified format.
- **Syntax**: `FORMAT(date, format)`

sql Copy code

```sql
SELECT FORMAT(order_date, 'dd-MM-yyyy') AS formatted_date
FROM sales.orders;
```

SQL Functions

4. Aggregate Functions

4.1. COUNT

4.2. SUM

4.3. AVG

4.4. MAX and MIN

Variables in SQL

1. Introduction

- Brief overview of variables in SQL
- Importance of using variables in SQL scripts

Variables in SQL

2. Types of Variables in SQL

- Local Variables
- Global Variables (System Variables)

Variables in SQL

3. Declaring Variables

3.1. Local Variables

- **Description**: Used to store temporary data within a session or procedure.

- Syntax:

sql Copy code

```sql
DECLARE @variable_name datatype;
```

Variables in SQL

3.2. Global Variables (System Variables)

- **Description**: Predefined variables that provide information about the server and database.

- Syntax:

```sql
SELECT @@global_variable_name;
```

- Example:

```sql
SELECT @@VERSION;
```

Variables in SQL

4. Assigning Values to Variables

4.1. SET

- **Description**: Assigns a value to a variable.

- Example:

sql

Copy code

```sql
DECLARE @total_orders INT;
SET @total_orders = (SELECT COUNT(*) FROM sales.orders);
```

Variables in SQL

4.2. SELECT

- **Description**: Assigns a value to a variable using a query.

- Example:

sql

Copy code

```sql
DECLARE @total_orders INT;
SELECT @total_orders = COUNT(*) FROM sales.orders;
```

Variables in SQL

5. Using Variables in Queries

```sql
declare @min int = 0;
declare @avg int = 0;

select @min = min(list_price),@avg = AVG(list_price)
from production.products

SELECT product_name, list_price
FROM production.products
WHERE list_price BETWEEN @min and @avg
```

Variables in SQL

6. Best Practices

- Always initialize variables to avoid unexpected results.
- Use descriptive variable names to make the code more readable.

Handling Nulls and Conditional Logic in SQL

1. Introduction

- Overview of handling null values and implementing conditional logic in SQL.
- Importance of managing null values and conditional expressions.

Handling Nulls and Conditional Logic in SQL

2. Handling Null Values

2.1. ISNULL

- **Description**: Replaces NULL with a specified replacement value.

- Example:

sql Copy code

```sql
SELECT first_name, ISNULL(phone, 'N/A') AS phone_number
FROM sales.customers;
```

Handling Nulls and Conditional Logic in SQL

2.2. NULLIF

- **Description**: Returns NULL if the two expressions are equal; otherwise, returns the first expression.

  - Example:

```sql
SELECT order_id, NULLIF(discount, 0) AS discount
FROM sales.order_items;
```

Handling Nulls and Conditional Logic in SQL

2.3. COALESCE

- **Description**: Returns the first non-null value in a list of expressions.

- Example:

sql

Copy code

```sql
SELECT first_name, COALESCE(phone, email, 'N/A') AS contact_info
FROM sales.customers;
```

Handling Nulls and Conditional Logic in SQL

3. Implementing Conditional Logic

3.1. CASE

- **Description**: Evaluates a list of conditions and returns one of multiple possible result expressions.

Handling Nulls and Conditional Logic in SQL

- Syntax:

sql Copy code

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

Handling Nulls and Conditional Logic in SQL

- Example:

sql

Copy code

```sql
SELECT order_id, order_status,
CASE
  WHEN order_status = 1 THEN 'Pending'
  WHEN order_status = 2 THEN 'Processing'
  WHEN order_status = 3 THEN 'Rejected'
  WHEN order_status = 4 THEN 'Completed'
  ELSE 'Unknown'
END AS status_description
FROM sales.orders;
```

Handling Nulls and Conditional Logic in SQL

3.2. IF ELSE

- **Description**: Executes a statement if a condition is true; otherwise, executes an alternative statement.

- Syntax:

sql Copy code

```sql
IF condition
  statement1
ELSE
  statement2
```

Handling Nulls and Conditional Logic in SQL

- **Example** (in a stored procedure or script):

sql Copy code

```sql
DECLARE @total_orders INT;
SET @total_orders = (SELECT COUNT(*) FROM sales.orders);

IF @total_orders > 100
  PRINT 'High order volume'
ELSE
  PRINT 'Normal order volume';
```

Handling Nulls and Conditional Logic in SQL

3.3. IIF

- **Description**: Returns one of two values, depending on whether the Boolean expression evaluates to true or false.

  - Syntax:

    sql Copy code

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

  - Example:

    sql Copy code

    ```sql
    SELECT order_id, IIF(order_status = 4, 'Completed', 'Not Completed') AS order_status
    FROM sales.orders;
    ```

Handling Nulls and Conditional Logic in SQL

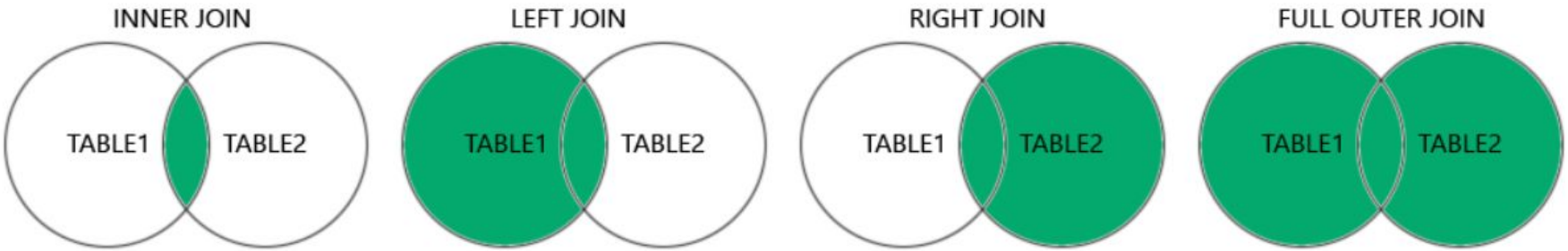
5. Best Practices

- Use `ISNULL` and `COALESCE` to handle null values effectively.
- Choose `CASE` for more complex conditional logic within queries.
- Use `IIF` for simple inline conditional checks.

Joins in SQL

1. Introduction

- Overview of joins in SQL
- Importance of joining tables



Joins in SQL

Joins in SQL

- **Description**: Joins are used to combine rows from two or more tables based on a related column between them.

- Syntax:

sql Copy code

```sql
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Joins in SQL

- Example:

sql

Copy code

```sql
SELECT o.order_id, c.first_name, c.last_name
FROM sales.orders o
INNER JOIN sales.customers c
ON o.customer_id = c.customer_id;
```

Joins in SQL

2.2. Left (Outer) Join

- **Description**: Returns all records from the left table and the matched records from the right table. The result is NULL from the right side if there is no match.

Joins in SQL

- Syntax:

sql Copy code

```sql
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column = table2.column;
```

Joins in SQL

- Example:

sql

Copy code

```sql
SELECT c.first_name, c.last_name, o.order_id
FROM sales.customers c
LEFT JOIN sales.orders o
ON c.customer_id = o.customer_id;
```