



Built-In Functions



Built-In Function Categories

- String Functions
- Mathematical / Numeric Functions
- Date Functions
- Security Functions
- Other Functions

String Functions

- SUBSTRING (value_exp , start_exp , length_exp)
 - Extracts **length_exp** number of characters from **value_exp** starting from **start_exp**

```
SELECT LastName, SUBSTRING(FirstName, 1, 1) AS Initial  
FROM Employee
```

```
SELECT x = SUBSTRING('abcdef', 2, 3)
```

- UPPER (character_expression)
 - Returns a character expression converted to uppercase
- LOWER (character_expression)
 - Returns a character expression converted to lowercase

String Functions (cont.)

- LEFT (character_expression , integer_expression)
 - Returns a specified no. of characters from the left part of a string
- RIGHT (character_expression , integer_expression)
- LEN (string_expression)
 - Returns the number of characters of the specified string expression, excluding trailing blanks
- STUFF(character_expression , start, length ,character_expression)
 - The STUFF function inserts a string into another string.
 - It deletes a specified length of characters in the first string at the start position and then inserts the second string into the first string at the start position
- REPLACE (string_expression1 , string_expression2 , string_expression3)
 - Replaces all occurrences of a specified string value with another string value

String Functions (cont.)

■ ASCII() Function

- The ASCII() function returns the ASCII value for the specific character.

- Syntax

ASCII(character)

- Examples: Return the ASCII value of the 1st character in "City" column

```
SELECT ASCII(City)
```

```
FROM Employee
```

- Example 2: Return the ASCII value of a character

```
SELECT ASCII('c')
```

■ CHAR() Function

- The CHAR() function returns the character based on the ASCII code

- Example

```
SELECT CHAR(65)
```

String Functions (cont.)

■ STR() Function

- The STR() function returns a number as a string.
- Examples

```
SELECT STR(185.5)
```

```
SELECT STR(185.476, 6, 2);
```

```
SELECT STR(185.476, 7, 4);
```

■ CONCAT() Function

- The CONCAT() function adds two or more strings together.
- Syntax

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

- Example

```
SELECT CONCAT('SQL', ' is', ' fun!');
```

String Functions (cont.)

■ CHARINDEX() Function

CHARINDEX (expToFind , expToSearch [, startAt])

- searches for one character expression inside a second character expression
- returns the starting position of the first expression if found
- If CHARINDEX does not find **expToFind** within **expToSearch**, CHARINDEX returns 0.

String Functions (cont.)

■ PATINDEX() Function

- The PATINDEX() function returns the position of a pattern in a string.
- Syntax

PATINDEX(%pattern%, string)

- If the pattern is not found, this function returns 0.
- Note: The search is case-insensitive and the first position in string is 1.
- Examples

```
SELECT PATINDEX('%oo%', 'Schools')  
SELECT PATINDEX('%s%oo%', 'Schools');
```

Mathematical Functions

- **FLOOR (numeric_expression)**
 - Returns the largest integer less than or equal to the specified numeric expression
- **CEILING (numeric_expression)**
- **ABS (numeric_expression)**
 - A mathematical function that returns the absolute (positive) value of the specified numeric expression.
- **POWER (float_expression , y)**
 - Returns the value of the specified expression to the specified power.
- **ISNUMERIC (expr)**
 - Checks if an expression is a valid numeric type



Date Functions

- GETDATE ()
 - Returns the current date of the system
- DATENAME (datepart , date)
 - Returns a string that represents the datepart of the date
- DATEPART (datepart , date)
 - Returns an integer that represents the datepart of the date
- DAY (date)
 - Returns an integer that represents the day part of the date
- MONTH (date)
 - Returns an integer that represents the month part of a date
- YEAR (date)
 - Returns an integer that represents the year part of a date

Date Functions - Examples

- Return the current month name

- `SELECT DATENAME ( month , GetDate() )`

- Return the number for the month

- `SELECT DATEPART ( month , GetDate() )`

- Return the number for day

- `SELECT DAY ( GetDate() )`

- Return the number for year

- ```
SELECT Fname, Lname, YEAR (EmploymentDate)
FROM Employee
```

# Date Functions (cont.)

- DATEDIFF ( datepart , startdate , enddate )
  - Returns the number of date or time datepart boundaries that are crossed between two specified dates
- DATEADD (datepart , number , date )
  - Returns a new datetime value by adding an interval to the specified datepart of the specified date
- ISDATE ( expression )
  - Determines whether a datetime or smalldatetime input expression is a valid date or time value.

# Security Functions

## ■ CURRENT\_USER

- Returns the name of the current user. This function is equivalent to USER\_NAME().
- SELECT CURRENT\_USER

## ■ USER\_ID

- Returns the identification number for a database user.
- SELECT USER\_ID('dbo');

## ■ USER\_NAME

- Returns a database user name from a specified identification number
- Example

```
SELECT USER_NAME(4);
SELECT USER_NAME();
```

# CAST & CONVERT

## ■ CAST

- Converts a value of one type into another datatype
- Syntax:

CAST ( expression AS data\_type [ (length ) ] )

## ■ CONVERT

- Converts a value of one type into another datatype
- Syntax:

CONVERT ( data\_type [ ( length ) ] , expression [ , style ] )

# Examples (CAST & CONVERT)

```
SELECT 9.5 AS Original
 , CAST(9.5 AS int) AS int
 , CAST(9.5 AS decimal(6,4)) AS decimal
```

```
OUTPUT > 9.5 9 9.5000
```

```
SELECT 'Salary is ' + CAST(Salary AS varchar(12)) AS Salary
FROM Employee
```

```
SELECT Salary is ' + Salary AS Salary
FROM Employee
```

# Examples (CAST & CONVERT)

```
SELECT SUBSTRING(FName, 1, 3) AS EmpName, Salary
FROM Employee
WHERE CAST(Salary AS INT) LIKE '3%'
```

```
SELECT CAST(FName AS VARCHAR(2)) AS EmpName
FROM Employee
```

```
SELECT CAST(25.65 AS int)
```

```
SELECT CAST(25.65 AS varchar)
```

# COALESCE

- Returns the first non-null expression among its arguments
- At least one of the null values must be a typed NULL
- Syntax:

COALESCE ( expression [ ,...n ] )

# COALESCE - Example

## ■ Examples

```
SELECT ProductID , UnitPrice, minPrice,
 COALESCE(0.9 * UnitPrice, minPrice , 5) AS Sale
FROM Product
ORDER BY UnitPrice, minPrice , Sale
```

```
SELECT FName + ' ' + COALESCE(LName, GFName)
AS EmpName
FROM Employee
```

# COALESCE and CASE

- The COALESCE expression is a syntactic shortcut for the CASE expression
- The following two expressions are equivalent

COALESCE (expr1, expr2)

CASE

    WHEN expr1 IS NOT NULL THEN expr1

    ELSE expr2

END

# CASE Expression - Example

- Using CASE in an ORDER BY clause

```
SELECT FName , LName
FROM Employee
ORDER BY
 CASE Nationality
 WHEN 'Ethiopian' THEN FName
 ELSE LName
 END
```

- Using CASE in an UPDATE statement

```
UPDATE Employee2
SET Salary =
 (CASE
 WHEN Salary < 5000 THEN 5000
 WHEN Salary < 10000 THEN 15000
 ELSE Salary + 2000
 END
)
```

# More Functions

## ■ ISNULL Function

### □ Syntax

ISNULL ( **check\_expression** , **replacement\_value** )

- The value of **check\_expression** is returned if it is not NULL; otherwise, **replacement\_value** is returned
- Example: Find the average of the weight of all products by substituting the value 50 for all NULL entries in the Weight column of the Product table

```
SELECT AVG(ISNULL(Weight, 50))
FROM Product;
```

# More Functions

## ■ NULLIF

- Same as . . . . “Return NULL if they are equal”
- Returns NULL if two expressions are equal, otherwise it returns the first expression

- Syntax

NULLIF(expr1, expr2)

- Example

```
SELECT NULLIF(4,4) AS Same
 , NULLIF(5,7) AS Different
```



# More Functions

- @@ERROR Function
- @@IDENTITY Function
  - SCOPE\_IDENTITY
  - IDENT\_CURRENT, and
  - @@IDENTITY
- @@ROWCOUNT Function



# @@ERROR Function

- ERROR Handling
- Error can happen
  - Duplicate key violation
  - Constraint violation
- Errors can be managed
  - Automated response
  - Error messages
- We can responding to errors by using the @@Error function

# @@ERROR Function (cont.)

- The @@ERROR system function
  - returns 0 if the last Transact-SQL statement executed successfully
  - or an error number ; if the statement generated an error
- @@ERROR gets a new value when every Transact-SQL statement completes
- To test or use @@ERROR immediately after the Transact-SQL statement
  - Save @@ERROR in an integer variable immediately after the Transact-SQL statement completes
  - Use the value stored in the variable to respond to the error

# @@ERROR Function (cont.)

- Example: FOREIGN KEY constraint violation

```
DECLARE @ErrorVar int
```

```
UPDATE Sales
```

```
SET CustomerID = 220
```

```
WHERE SalesID = 6
```

```
SET @ErrorVar = @@ERROR
```

```
IF @ErrorVar <> 0
```

```
 PRINT 'Error = ' + CAST(@ErrorVar AS NVARCHAR(8));
```

```
GO
```

# @@IDENTITY Function

- The last value inserted into an IDENTITY column by an INSERT or SELECT INTO statement.
- If a statement inserts multiple rows, @@identity reflects the IDENTITY value for the last row inserted.
  - If the affected table does not contain an IDENTITY column, @@identity is set to 0.
- The return type is: NUMERIC(38,0)
- The value of @@identity is not affected by the failure of an insert or select into statement, or the rollback of the transaction that contained it.

# SCOPE\_IDENTITY

- IDENT\_CURRENT is not limited by scope and session;
  - it is limited to a specified table.
  - IDENT\_CURRENT returns the value generated for a specific table in any session and any scope
- SCOPE\_IDENTITY and @@IDENTITY
  - return the last identity values that are generated in any table in the current session.
  - SCOPE\_IDENTITY returns values inserted only within the current scope;
- @@IDENTITY is not limited to a specific scope.

# SCOPE\_IDENTITY

- SCOPE\_IDENTITY, IDENT\_CURRENT, and @@IDENTITY are similar functions
  - They all return values that are inserted into identity columns
- IDENT\_CURRENT
  - returns the last identity value generated for a specific table in any session and any scope.
- @@IDENTITY
  - returns the last identity value generated for any table in the current session, across all scopes.
- SCOPE\_IDENTITY
  - returns the last identity value generated for any table in the current session and the current scope.
- A scope is a module; a Stored Procedure, trigger, function, or batch

# @@IDENTITY - Example

- Example

```
INSERT INTO myTable (Code) VALUES (5)
GO
SELECT @@IDENTITY AS 'Identity';
```

```
SELECT IDENT_CURRENT ('Employee') AS Current_Identity
```

```
SELECT SCOPE_IDENTITY()
```

# @@ROWCOUNT Function

- The number of rows affected by the last command.
  - The Return type: int
- If the number of rows is more than 2 billion, use ROWCOUNT\_BIG()
- Statements that make a simple assignment always set the @@ROWCOUNT value to 1.

- Example

```
UPDATE Employee
SET Salary = 7800
WHERE EmpID = 123456789
```

```
IF @@ROWCOUNT = 0
PRINT 'Warning: No rows were updated';
```

# Deterministic & Nondeterministic

- **Deterministic & Nondeterministic Functions**
- A function is said to be **Deterministic** or **Nondeterministic Function** depending on whether it returns the same or different value given the same set of input values.
  - Deterministic functions always return the **same result**
  - Nondeterministic functions **may return different results** each time they are called
- **Examples**
  - Deterministic
    - DATEDIFF, ISNUMERIC, ISNULL
  - Nondeterministic
    - NEWID, NEWSEQUENTIALID