# SOME sql COMMON FUNCTIONS

The following lists some common functions you can use in SQL Server.

| **Function Name** | **Description** |
| --- | --- |
| **CAST** | Explicitly converts an expression of one data type to another. CAST and CONVERT provide similar functionality.  **Example**: When you want to convert a string to a number:  SELECT cast(‘1234’ as int) |
| **CONVERT** | Explicitly converts an expression of one data type to another. CAST and CONVERT provide similar functionality.  **Example**: When you want to convert a string to a number:  SELECT convert(int, '1234') |
| **GETDATE** | Returns the current system date and time in the standard internal format for **datetime** values  **Example**:  SELECT GETDATE() |
| **LEN** | Returns the number of characters, instead of the number of bytes, of the given string expression, excluding trailing blanks.  **Example**: Check to see the length of a constant string:  SELECT LEN('abcde')  **Example**: Get all employees whose lengths are greater than 5  SELECT FirstName, LEN(FirstName) as [Length Of First Name]  FROM Employees  WHERE LEN(FirstName) > 5 |

|  |  |
| --- | --- |
| **DATEPART** | The DATEPART and DATENAME functions produce the specified part of a **datetime** value such as the year, quarter, day, or hour, as either an integer or an ASCII string. Because **smalldatetime** is accurate only to the minute, when a **smalldatetime** value is used with either of these functions, the seconds and milliseconds returned are always zero  **Example**:  SELECT  orderid,  DATEPART(year, OrderDate) as [Order Year],  DATEPART(month, OrderDate) as [Order Month],  DATEPART(day, OrderDate) as [Order Day]  FROM Orders |
| **DATEADD** | Returns a new DT\_DBTIMESTAMP value after adding a number that represents a date or time interval to the specified datepart in a date  **Example**:  SELECT  dateadd(year, 5, GETDATE()) as [5 years after],  dateadd(month, 5, GETDATE()) as [5 months after],  dateadd(day, 5, GETDATE()) as [5 days after]  FROM Orders |
| **DATEDIFF** | Returns the number of date and time boundaries crossed between two specified dates. The *datepart* parameter identifies which date and time boundaries to compare.  **Example**: How long between ‘1-jan-10’ and ‘1-jan-11’ (by year, month, day)  SELECT  DATEDIFF(year, '1-jan-10', '1-jan-11'),  DATEDIFF(month, '1-jan-10', '1-jan-11'),  DATEDIFF(day, '1-jan-10', '1-jan-11')  FROM Orders |

|  |  |
| --- | --- |
| **REPLACE** | Replaces all occurrences of the second specified string expression in the first string expression with a third expression.  **Example**: Replace all ‘d’ character by ‘x’:  SELECT REPLACE('abcdddd', 'd','x')  **Example**: Replace all ‘d’ character by ‘x’:  SELECT REPLACE(FirstName, 'd','x')  FROM Employees |
| **UPPER** | Returns a character expression with lowercase character data converted to uppercase.  **Example**:  SELECT UPPER('abcd')  **Example**:  SELECT UPPER(firstName)  From Employees |
| **LOWER** | Returns a character expression after converting uppercase character data to lowercase.  **Example**:  SELECT LOWER('abcd')  **Example**:  SELECT LOWER(firstName)  From Employees |
| **STR** | Returns character data converted from numeric data.  **Example**:  SELECT 'I love the number: ' + STR(100) |

|  |  |
| --- | --- |
| **REVERSE** | Returns the reverse of a character expression.  **Example**:  SELECT REVERSE('abcd') |
| **LTRIM** | Returns a character expression after it removes leading blanks.  **Example**:  SELECT LTRIM(' abcd ') |
| **RTRIM** | Returns a character string after truncating all trailing blanks.  **Example**:  SELECT RTRIM(' abcd ') |
| **LEFT** | Returns the left part of a character string with the specified number of characters.  **Example**:  SELECT LEFT('1234abcd', 4) |
| **RIGHT** | Returns the right part of a character string with the specified number of characters.  **Example**:  SELECT RIGHT('1234abcd', 4) |
| **SUBSTRING** | Returns part of a character, binary, text, or image expression  **Example**: Return the substring with 6 characters from position 1 (‘I hate’):  SELECT SUBSTRING('I hate studying', 1, 6) |

|  |  |
| --- | --- |
| **ISNULL** | Replaces NULL with the specified replacement value.  **Example**: For all employees: if their region is null, replace with ‘Has no region’  SELECT FirstName, LastName,  ISNULL(Region, 'Has no region')  FROM Employees |
| **CASE** | CASE *input\_expression*  WHEN *when\_expression* THEN *result\_expression*  [ ...*n* ]  [  ELSE *else\_result\_expression*  ]  END  **Example**: For all employees: if their region is null, replace with ‘Has no region’:  SELECT FirstName, LastName,  case when Region is not null then Region  when Region is null then 'Has not region'  end  FROM Employees |

|  |  |
| --- | --- |
| **@@IDENTITY** | **Example**: Look at the Employees table:  CREATE TABLE Employees  (  EmployeeID int IDENTITY(1,1) NOT NULL,  LastName nvarchar(20) NOT NULL,  FirstName nvarchar(10) NOT NULL  )  The column EmployeeID is set to “IDENTITY”, this means that: the ID is automatically increased.  So, SQL Server will raise an error if you run the following statement:  INSERT INTO Employees(EmployeeID, LastName, FirstName)  VALUES (100, 'vinh', 'thu')  To make the above ‘insert’ statement run successfully, you must rewrite:  INSERT INTO Employees(LastName, FirstName)  VALUES ('vinh', 'thu')  But now, you dont know what EmployeeId is inserted into Employees table. To solve this problem, use @@IDENTITY as following (note that: you must run the following query in the same session with ‘Insert’ statement):  SELECT @@IDENTITY |