MS SQL Common Functions — Practical Guide

> A hands-on reference for day-to-day SQL Server functions: what they do, how they behave, examples, and real-world scenarios.

Table of contents

> **Notes for samples:**
>
> - Examples assume Microsoft SQL Server (T-SQL) syntax.
> - Replace placeholder table/column names with your own.
> - Where useful, `TRY_` variants are shown to avoid runtime errors.

String functions

String functions are used for parsing, cleaning, and transforming textual data — extremely common in ETL, reporting, and data cleaning.

1. `LEN()`

- **What:** Returns length of string (number of characters). Trailing spaces are ignored.
- **Example:** `SELECT LEN('Hello ') -> 5`

- **Real world:** Validate length of user-entered IDs or codes.

2. `LTRIM()`, `RTRIM()`, `TRIM()`

- **What:** Remove whitespace from left, right, or both sides (trim added in SQL Server 2017+).
- **Example:** `SELECT TRIM(' abc ')` -> `'abc'`
- **Real world:** Clean user inputs (names, codes) before joining or comparisons.

3. `UPPER()`, `LOWER()`

- **What:** Convert case.
- **Example:** `SELECT UPPER('Anshul')` -> `'ANSHUL'`
- **Real world:** Case-insensitive comparisons (normalize before grouping or joining).

4. `SUBSTRING(expression, start, length)`

- **What:** Extracts substring starting at `start` (1-based).
- **Example:** `SELECT SUBSTRING('2025-11-21', 1, 4)` -> `'2025'`
- **Real world:** Extract year from `YYYY-MM-DD` stored as text.

5. `LEFT()` / `RIGHT()`

- **What:** Return left-most or right-most N characters.
- **Example:** `SELECT RIGHT('ABC-123', 3)` -> `'123'`
- **Real world:** Grab file extensions, last digits of phone numbers, etc.

6. `CHARINDEX(substring, expression [, start])` and `PATINDEX('%pattern%', expression)`

- **What:** Returns position of substring or pattern. `PATINDEX` supports wildcards.
- **Example:** `SELECT CHARINDEX('@', 'user@example.com')` -> `5`
- **Real world:** Validate email-looking strings, split by delimiter.

7. `REPLACE(expression, pattern, replacement)`

- **What:** Replace all occurrences of `pattern` with `replacement`.
- **Example:** `SELECT REPLACE('A/B/C','/','\')` -> `'A\B\C'`
- **Real world:** Normalize slashes, remove unwanted characters before storage.

8. `STUFF(expression, start, length, replaceWith)`

- **What:** Delete `length` characters starting at `start` and insert `replaceWith` there.
- **Example:** `SELECT STUFF('123-456-7890', 4, 1, '')` -> `'123456-7890'` (removes dash at position 4)
- **Real world:** Reformat phone numbers or IDs.

9. `CONCAT(...)`, `CONCAT_WS(separator, ...)`

- **What:** Concatenate values; `CONCAT` implicitly converts `NULL` to empty; `CONCAT_WS` uses separator and ignores NULLs.
- **Example:** `SELECT CONCAT_WS(' ', FirstName, LastName)`
- **Real world:** Build full names, addresses.

10. `FORMAT(value, format_string [, culture])` (use sparingly)

- **What:** Formats numbers/dates to string using .NET formatting rules. Slower than `CONVERT` / `CAST`.
- **Example:** `SELECT FORMAT(GETDATE(), 'yyyy-MM-dd')`
- **Real world:** Prepare human-friendly display values for reports (not recommended in large set processing).

11. `STRING_AGG(expression, delimiter) [SQL Server 2017+]`

- **What:** Aggregate strings into a single delimited string.
- **Example:** `SELECT STRING_AGG(Name, ', ') FROM Tags` -> `'tag1, tag2, tag3'`
- **Real world:** Combine multiple tag rows into a single CSV for display or export.

Numeric Functions

Common for calculations, rounding, and numeric transformations.

1. `ABS(x)`

- **What:** Absolute value.
- **Example:** `SELECT ABS(-42)` -> `42`

2. `CEILING(x)`, `FLOOR(x)`

- **What:** Round up or down to nearest integer.
- **Example:** `SELECT CEILING(4.2)` -> `5`, `FLOOR(4.8)` -> `4`

3. `ROUND(x, length [, function])`

- **What:** Round to `length` decimal places. Third parameter `function` = 1 truncates instead of rounding.
- **Example:** `SELECT ROUND(123.4567, 2)` -> `123.46`

4. `POWER(x, y)`, `SQRT(x)`

- **What:** Power and square root.
- **Example:** `SELECT POWER(2, 10)` -> `1024`

5. `CEILING`, `FLOOR` for buckets, `NTILE()` is a window function often used for bucketing.

6. `RAND([seed])` and `CHECKSUM(NEWID())` for randomness

- **What:** `RAND()` generates pseudo random float; `CHECKSUM(NEWID()) % n` often used for row-level randomization.
- **Real world:** Sample rows for QA or A/B testing.

Date & Time Functions

Date arithmetic and formatting are central to reporting and time-window analytics.

1. `GETDATE()` vs `SYSDATETIME()`

- **What:** `GETDATE()` returns `datetime`; `SYSDATETIME()` returns `datetime2` with higher precision.
- **Real world:** Use `SYSDATETIME()` if you need sub-second precision.

2. `DATEADD(datepart, number, date)`

- **What:** Add an interval to a date.
- **Example:** `SELECT DATEADD(day, 7, GETDATE())` -> date 7 days in future
- **Real world:** Calculate subscription expiry, schedule next run.

3. `DATEDIFF(datepart, start, end)`

- **What:** Difference between two dates in the specified `datepart` (days, months, years...).
- **Note:** `DATEDIFF` counts boundaries crossed — e.g., months between `2025-01-31` and `2025-02-01` is 1.
- **Use with caution** for precise age calculations.

4. `DATEPART(datepart, date)` and `DATENAME(datepart, date)`

- **What:** Extract specific part (year, month, weekday name).
- **Example:** `DATEPART(weekday, GETDATE())`

5. `EOMONTH(date [, month_to_add])`

- **What:** End of month for the date.
- **Example:** `EOMONTH('2025-02-10')` -> `2025-02-28` (or 29 for leap year)
- **Real world:** Monthly reporting cutoffs.

6. `TRY_CONVERT`, `CONVERT` with styles

- **What:** Convert strings to date types using styles (e.g., 103 = dd/mm/yyyy) — see conversion section.

7. Time zone related: `AT TIME ZONE` (SQL Server 2016+)

- **What:** Convert datetimeoffsets between time zones, or attach time zone info.
- **Example:** `SELECT (SYSDATETIMEOFFSET()) AT TIME ZONE 'India Standard Time'`
- **Real world:** Store timestamps in UTC and convert to local time for presentation.

Conversion & Type-handling

1. `CAST(expr AS type)` and `CONVERT(type, expr [, style])`

- **What:** Convert values explicitly; `CONVERT` supports style codes for date formats.
- **Example:** `SELECT CAST('123' AS INT)`
- **Date example:** `SELECT CONVERT(varchar(10), GETDATE(), 120)` -> `yyyy-mm-dd` style

2. `TRY_CAST`, `TRY_CONVERT`

- **What:** Same as above but return `NULL` on conversion failure instead of error.
- **Real world:** Safely convert user input or CSV imports without aborting the query.

3. `PARSE()` and `TRY_PARSE()`

- **What:** Parse strings using .NET formats and optionally culture; slower than `CONVERT` — use sparingly.

NULL handling & conditional expressions

1. `ISNULL(value, replacement)`

- **What:** Replace `NULL` with `replacement`.
- **Example:** `SELECT ISNULL(Phone, 'N/A')`
- **Note:** Returns the type and length of the first argument; watch type precedence.

2. `COALESCE(val1, val2, ..., valN)`

- **What:** Returns the first non-NULL value.
- **Example:** `COALESCE(AddrLine2, AddrLine3, '—')`
- **Real world:** Pick the first available contact method.

3. `NULLIF(expr1, expr2)`

- **What:** Returns `NULL` if `expr1 = expr2`, else `expr1`.
- **Real world:** Avoid division-by-zero: `x / NULLIF(y,0)` -> returns NULL instead of error when `y=0`.

4. `CASE` and `IIF(condition, true_value, false_value)`

- **What:** Conditional logic. `CASE` is more flexible.
- **Example:**

```sql
SELECT
CASE
  WHEN Score >= 90 THEN 'A'
  WHEN Score >= 80 THEN 'B'
  ELSE 'C'
END AS Grade
FROM Scores;
```

Aggregate functions

Used in reporting and group summaries.

1. `COUNT(expr)` / `COUNT(*)` / `COUNT(DISTINCT expr)`

- **What:** Number of rows (or distinct values).

2. `SUM()`, `AVG()`, `MIN()`, `MAX()`

- **What:** Basic arithmetic aggregates.

3. `STRING_AGG()`

- **See string section** — aggregate strings.

4. `GROUPING()` / `GROUPING_ID()` in rollups/cubes

- **What:** Detect whether a row is from aggregations (useful in ROLLUP or CUBE results).

Window (analytic) functions

Window functions are essential for advanced reporting, running totals, ranking, and time-series comparisons.

1. Ranking: `ROW_NUMBER()`, `RANK()`, `DENSE_RANK()`

- **What:** Assign row numbers or ranks over partition ordered sets.
- **Example:**

```sql
SELECT
EmployeeID, Salary,
RANK() OVER (PARTITION BY Department ORDER BY Salary DESC) AS DeptRank
FROM Employees;
```

- **Real world:** Top-per-department queries, leaderboards.

2. `NTILE(n)`

- **What:** Divide partition into `n` roughly equal buckets (quartiles, percentiles).
- **Real world:** Create quartiles for customer lifetime value.

3. `LAG(expr, offset, default)` and `LEAD(expr, offset, default)`

- **What:** Access previous/next row's value in ordered partition.
- **Example:** Compute day-over-day change in sales.

4. `FIRST_VALUE()` / `LAST_VALUE()`

- **What:** Access first or last value in window frame.
- **Note:** Frame definition matters — default frame may include current row to end; often use `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`.

5. Running totals & moving averages

- **Pattern:** `SUM(Value) OVER (PARTITION BY X ORDER BY Date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)`
- **Real world:** Cumulative sales YTD, 7-day moving average of metrics.

JSON helpers (SQL Server 2016+)

JSON is common for semi-structured storage and message exchange.

1. `ISJSON(expr)`

- **What:** Returns 1 if string is valid JSON.

2. `JSON_VALUE(json, path)`

- **What:** Extract scalar value from JSON.
- **Example:** `JSON_VALUE('{"user":{"id":123}}', '$.user.id')` -> `123`

3. `JSON_QUERY(json, path)`

- **What:** Extract object or array.

4. `OPENJSON()`

- **What:** Parse JSON into rows and columns (very useful for imports).
- **Example:** `SELECT * FROM OPENJSON(@json) WITH (id int '$.id', name nvarchar(100) '$.name')`
- **Real world:** Consume webhook payloads, flatten arrays into rows.

Other utility functions & tips

`OBJECT_ID('schema.table')` and metadata queries

- Useful to check table exists before creating/dropping.

`TRY_CONVERT` / `TRY_CAST` over `CONVERT` when reading dirty data

`CHECKSUM`, `BINARY_CHECKSUM` for quick change detection (not cryptographic)

`HASHBYTES('SHA2_256', data)` for deterministic hashes (useful for de-duplication)

`NEWID()` for GUIDs, `SEQUENCE` / `IDENTITY` for numeric auto-increment

Practical recipes & real-world examples

Below are common tasks and concise patterns you can drop into ETL, reporting or production queries.

1. Normalize names and create "FullName" for reporting

```
SELECT
  TRIM(CONCAT_WS(' ', NULLIF(NULLIF(LTRIM(RTRIM(FirstName)),'') ,NULL),
NULLIF(LTRIM(RTRIM(LastName)),'') )) AS FullName
FROM dbo.Users;
```

Why: removes empty strings, trims spaces, ignores NULLs.

2. Extract domain from email

```
SELECT
  Email,
  SUBSTRING(Email, CHARINDEX('@',Email)+1, LEN(Email)) AS Domain
FROM dbo.Contacts
WHERE CHARINDEX('@',Email) > 0;
```

Real world: Group signups by email provider.

3. Safe numeric conversion (bulk CSV import)

```
SELECT
  TRY_CAST(col1 AS INT) AS col1_int,
  TRY_CAST(col2 AS DECIMAL(18,2)) AS price
FROM Staging.ImportCsv;
```

Why: avoids runtime errors on bad formats.

4. Calculate customer age

```
SELECT
  BirthDate,
  DATEDIFF(year, BirthDate, GETDATE()) -
    CASE WHEN DATEADD(year, DATEDIFF(year, BirthDate, GETDATE()), BirthDate) >
GETDATE() THEN 1 ELSE 0 END AS Age
FROM Customers;
```

Why: `DATEDIFF(year, ...)` alone can overcount; this adjusts for birthdays.

5. Last non-null value per partition (carry-forward fill)

```
SELECT *,
  MAX(Value) OVER (PARTITION BY Customer ORDER BY EventDate ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS LastKnownValue
FROM Events
WHERE Value IS NOT NULL OR 1=1; -- depends how you want to handle NULLs
```

6. Running total of sales YTD

```
SELECT
  SalesDate,
  Amount,
  SUM(Amount) OVER (ORDER BY SalesDate ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT
ROW) AS RunningTotal
FROM Sales
WHERE YEAR(SalesDate) = 2025
ORDER BY SalesDate;
```

7. Pivoting strings into rows using `STRING_SPLIT` (SQL Server 2016+)

```
-- store tags as comma separated string and split into rows
SELECT t.Id, value AS Tag
FROM MyTable t
CROSS APPLY STRING_SPLIT(t.TagsCsv, ',');
```

8. Aggregate with `STRING_AGG`

```
SELECT p.ProductID, STRING_AGG(t.TagName, ', ') AS AllTags
FROM ProductTags t
JOIN Products p ON p.ProductID = t.ProductID
GROUP BY p.ProductID;
```

Performance & best-practices notes

1. **Prefer proper types**: store dates in `datetime2`, numeric in `decimal` / `int` etc. Avoid storing numbers/dates as strings.
2. **Use `TRY_` conversions** when ingesting dirty data to prevent query failures.
3. **Avoid `FORMAT()` in large result sets** — it's slow because it uses .NET formatting.
4. **Be careful with `LEN()` vs `DATALENGTH()`**: `LEN()` counts characters (ignores trailing spaces), `DATALENGTH()` returns bytes (useful with varbinary).
5. **Watch implicit conversions**: mismatched types can cause scans rather than seeks and harm performance.
6. **Use computed columns and persisted columns** when you repeatedly compute the same transformation for indexing.

Quick reference cheat-sheet (select commonly-used functions)

- String: `LEN`, `SUBSTRING`, `LEFT`, `RIGHT`, `CHARINDEX`, `REPLACE`, `STUFF`, `CONCAT`, `CONCAT_WS`, `STRING_AGG`
- Numeric: `ABS`, `ROUND`, `CEILING`, `FLOOR`, `POWER`, `SQRT`, `RAND`
- Date: `GETDATE`, `SYSDATETIME`, `DATEADD`, `DATEDIFF`, `DATEPART`, `EOMONTH`, `AT TIME ZONE`
- Conversion: `CAST`, `CONVERT`, `TRY_CAST`, `TRY_CONVERT`, `PARSE`
- Null/conditional: `ISNULL`, `COALESCE`, `NULLIF`, `CASE`, `IIF`
- Window: `ROW_NUMBER`, `RANK`, `DENSE_RANK`, `NTILE`, `LAG`, `LEAD`, `SUM() OVER(...)`
- JSON: `ISJSON`, `JSON_VALUE`, `JSON_QUERY`, `OPENJSON`

Suggested next steps (if you want me to extend this document)

- Add **30 practice problems** with increasing difficulty (I can generate AdventureWorks-based practice queries).
- Create a **cheat-sheet PDF** or slide deck.
- Provide **performance-tuned examples** showing index usage and execution plans.

End of file.