# Funktionen

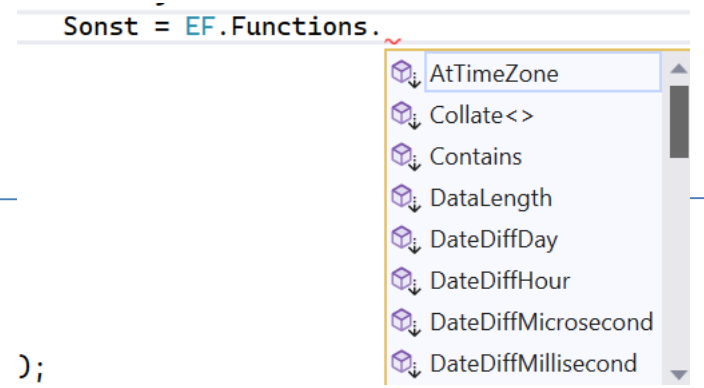
- In SQL-Datenbanken gibt es sowohl integrierte Funktionen (Buildt-In) als auch nutzerdefinierte Funktionen (udf)
- In EF Core-Abfragen können sowohl die integrierten als auch die benutzerdefinierten Funktionen aufgerufen werden
- Der Zugriff auf eine **benutzerdefinierte Skalarwert-Funktion** wird in der DBContext-Klasse als statische Methode definiert:

```
[DbFunction(Name = "udfBerechneAlter", Schema = "dbo", IsBuiltIn = false, IsNullable = true)]
public static int Alter(DateOnly? Geburtsdatum)
    => throw new InvalidOperationException($"{nameof(GetAlter)} hat zu einem Fehler geführt");
```

- In einer Abfrage kann dann diese Funktion integriert werden:

```
var mitListe = dbctx.MitarbeiterListe
    .OrderBy(m => m.Nachname)
    .Select(m => new
    {
        MitarbeiterId = m.MitarbeiterId,
        Name = m.Nachname,
        Vorname = m.Vorname,
        Alter = EFCTestDBContext.GetAlter(m.Geburtsdatum),
    });
```

# Integrierte Funktionen – EF.Functions




- Die in den SQL-Datenbanken integrierten Funktionen sind als Erweiterungsmethoden über EF.Functions definiert und können in LINQ-Abfragen verwendet werden.
- Diese Methoden sind providerspezifisch, da sie eng mit bestimmten Datenbankfunktionen verknüpft sind. Eine Methode, die bei einem Anbieter funktioniert, funktioniert also wahrscheinlich nicht bei einem anderen Anbieter.
- Beispiel: Der LIKE-Operator und die Funktion DateDiffYear:

```
var mitListe = dbctx.MitarbeiterListe
    .OrderBy(m => m.Nachname)
    .Where(m => EF.Functions.Like(m.Nachname,"P%")) //Zugriff auf integrierte Funktion bzw. Operator
    .Select(m => new
    {
        MitarbeiterId = m.MitarbeiterId,
        Name = m.Nachname,
        Vorname = m.Vorname,
        Anrede = GetAnrede(m.Geschlecht), //Lokale Funktion
        Alter = EFCTestDBContext.GetAlter(m.Geburtsdatum),//Nutzerdefinierte Datenbankfunktion
        Dienstjahre = EF.Functions.DateDiffYear(m.DatumEintritt,m.DatumAustritt) //EF-Funktion
    });
```

# Beispiele: String-Funktionen

| EF.Function | SQL |
|---|---|
| EF.Functions.Collate(operand, collation) | @operand COLLATE @collation |
| EF.Functions.Like(matchExpression, pattern) | @matchExpression LIKE @pattern |
| EF.Functions.Like(matchExpression, pattern, escapeCharacter) | @matchExpression LIKE @pattern ESCAPE @escapeCharacter |
| string.Compare(strA, strB) | CASE WHEN @strA = @strB THEN 0 ... END |
| string.Concat(str0, str1) | @str0 + @str1 bzw. CONCAT(@str0, @str1) |
| string.IsNullOrEmpty(value) | @value IS NULL OR @value = '' |
| string.IsNullOrWhiteSpace(value) | @value IS NULL OR trim(@value) = '' |
| stringValue.CompareTo(strB) | CASE WHEN @stringValue = @strB THEN 0 ... END |
| stringValue.Contains(value) | instr(@stringValue, @value) > 0 |
| stringValue.EndsWith(value) | @stringValue LIKE '%' |
| stringValue.FirstOrDefault() | substr(@stringValue, 1, 1) |
| stringValue.IndexOf(value) | instr(@stringValue, @value) - 1 |
| stringValue.LastOrDefault() | substr(@stringValue, length(@stringValue), 1) |

# String-Funktionen - Fortsetzung

| EF.Function | SQL |
|---|---|
| stringValue.Length | length(@stringValue) |
| stringValue.Replace(oldValue, newValue) | replace(@stringValue, @oldValue, @newValue) |
| stringValue.StartsWith(value) | @stringValue LIKE @value \|\| '%' |
| stringValue.Substring(startIndex, length) | substr(@stringValue, @startIndex + 1, @length) |
| stringValue.ToLower() | lower(@stringValue) |
| stringValue.ToUpper() | upper(@stringValue) |
| stringValue.Trim() | trim(@stringValue) |
| stringValue.Trim(trimChar) | trim(@stringValue, @trimChar) |
| stringValue.TrimEnd() | rtrim(@stringValue) |
| stringValue.TrimEnd(trimChar) | rtrim(@stringValue, @trimChar) |
| stringValue.TrimStart() | ltrim(@stringValue) |
| stringValue.TrimStart(trimChar) | ltrim(@stringValue, @trimChar) |