# Работа с LIKE

Определяет, соответствует ли аргумент типа **String** заданному шаблону.

## Синтаксис

```
match [NOT] LIKE pattern [ESCAPE escape]
```

## Аргументы

*match*

Выражение Entity SQL, значением которого является **String**.

*pattern*

Шаблон для сопоставления с заданным значением **String**.

*escape*

Escape-символ.

NOT

Указывает, что результат оператора LIKE должен быть инвертирован.

## Возвращаемое значение

Равно **true**, если **string** соответствует шаблону, в противном случае равно **false**.

## Заметки

В основном выражения Entity SQL, в которых используется оператор LIKE, вычисляются тем же способом, что и выражения, в которых в качестве условия фильтра используется оператор равенства. Однако выражения Entity SQL с оператором LIKE могут включать как литералы, так и символы-шаблоны.

В следующей таблице описан синтаксис шаблона **string**.

| Символ-шаблон | Описание | Пример |
|---|---|---|
| % | Любое значение **string** длиной от нуля и более символов. | По условию `title like '%computer%'` будут найдены все названия, содержащие слово `"computer"` (в любом месте названия). |
| _ (символ подчеркивания) | Любой отдельный символ. | По условию `firstname like '_ean'` будут найдены имена из четырех букв, которые оканчиваются на `"ean`, например «Dean» или «Sean». |
| [ ] | Любой отдельный символ в диапазоне ([a-f]) или наборе ([abcdef]). | По условию `lastname like '[C-P]arsen'` будут найдены фамилии, которые оканчиваются на «arsen», а начинаются с любого символа между «C» и «P», например «Carsen» или «Larsen». |
| [^] | Любой символ, содержащийся в диапазоне ([^a-f]) или наборе ([^abcdef]). | По условию `lastname like 'de[^l]%'` будут найдены все фамилии, которые начинаются с символов «de» и не включают символа «l» в качестве следующего символа. |

**Примечание**

Оператор LIKE и предложение ESCAPE языка Entity SQL не могут применяться к значениям **System.DateTime** и **System.Guid**.

Оператор LIKE поддерживает сравнение по шаблонам в ASCII и Юникоде. Если все параметры имеют символьный тип ASCII, то применяется шаблон ASCII. Если один или несколько аргументов представлен в Юникоде, то выполняется преобразование всех аргументов в Юникод, а затем применяется шаблон в Юникоде. Завершающие пробелы учитываются только при работе оператора LIKE с данными в Юникоде. Для других типов данных завершающие пробелы не учитываются. Строка шаблона языка Entity SQL имеет тот же синтаксис, что и Transact-SQL.
Шаблон может включать в себя обычные символы и символы-шаблоны. Во время сравнения с шаблоном обычные символы должны в точности совпадать с символами, присутствующими в **string**. Символы-шаблоны могут совпадать с произвольными элементами символьной строки. При использовании символов-шаблонов оператор LIKE более гибок, нежели операторы сравнения строки = и !=.

**Примечание**

Для работы с конкретным поставщиком могут использоваться расширения, специфические для поставщика. Однако такие конструкции другими поставщиками могут обрабатываться иначе. Например, SqlServer поддерживает шаблоны [первый-последний] и [^первый-последний], в которых первый сопоставляет ровно один символ между первым и последним, а последний - ровно один символ, который не находится между первым и последним.

### ESC

Предложение ESCAPE позволяет искать символьные строки, в состав которых входит один или более специальных символов-шаблонов, описанных в таблице в прошлом разделе. Предположим, нужно найти все документы, содержащие в заголовке литерал «100%».Поскольку символ процента (%) является символом-шаблоном, для успешного выполнения поиска необходимо экранировать его при помощи предложения ESCAPE языка Entity SQL. Ниже приведен пример этого фильтра.

```
"title like '%100!%%' escape '!'"
```

В этом выражении поиска символ-шаблон процента (%), немедленно следующий за символом восклицательного знака (!), рассматривается как литерал, а не как символ-шаблон. В качестве escape-символа можно использовать любой символ, кроме символов-шаблонов Entity SQL и символов квадратных скобок ([ ]). В прошлом примере символ восклицательного знака (!) используется как escape-символ.

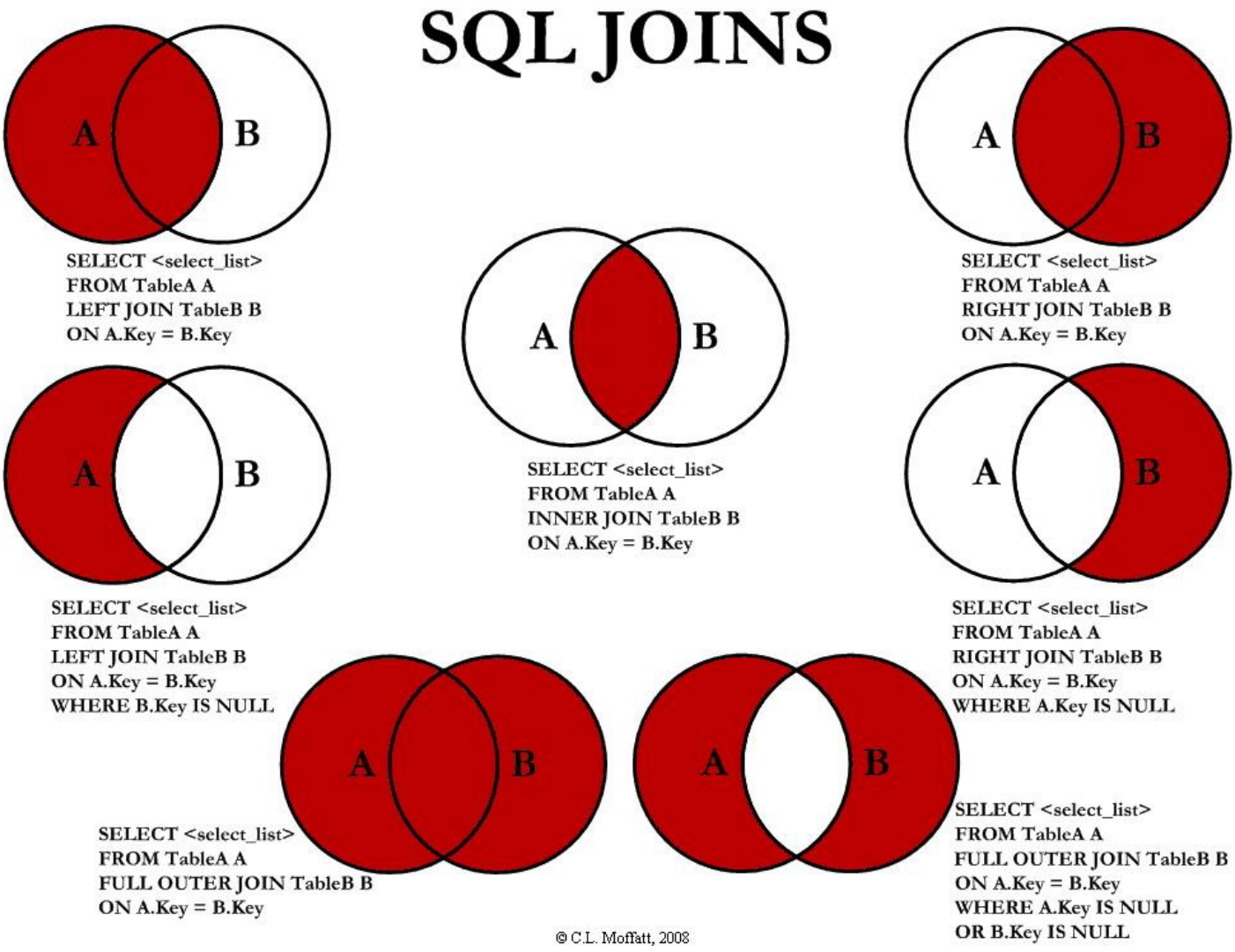
# Пример

```
-- LIKE and ESCAPE
-- If an AdventureWorksEntities.Products contained a Name
-- with the value 'Down_Tube', the following query would find that
-- value.
Select value P.Name FROM AdventureWorksEntities.Products
    as P where P.Name LIKE 'DownA_%' ESCAPE 'A'

-- LIKE
Select value P.Name FROM AdventureWorksEntities.Products
    as P where P.Name like 'BB%'
```

# Объяснение SQL объединений JOIN: LEFT/RIGHT/INNER/OUTER

https://www.codeproject.com/articles/33052/visual-representation-of-sql-joins



# 1. Using a simple FROM clause

```
SELECT TerritoryID, Name
FROM Sales.SalesTerritory
ORDER BY TerritoryID ;
```

```
SELECT Name, ProductNumber, ListPrice AS Price
FROM Production.Product
ORDER BY Name ASC;
```

```
SELECT Name, ProductNumber, ListPrice AS Price
FROM Production.Product
WHERE ProductLine = 'R'
AND DaysToManufacture < 4
ORDER BY Name ASC;
```

## B. Using SELECT with column headings and calculations

```
USE AdventureWorks2008;
GO
SELECT p.Name AS ProductName,
NonDiscountSales = (OrderQty * UnitPrice),
Discounts = ((OrderQty * UnitPrice) * UnitPriceDiscount)
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail AS sod
ON p.ProductID = sod.ProductID

ORDER BY ProductName DESC;
GO
```

```
USE AdventureWorks2008;
GO
SELECT 'Total income is', ((OrderQty * UnitPrice) * (1.0 - UnitPriceDiscount)), ' for ',
p.Name AS ProductName
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail AS sod
ON p.ProductID = sod.ProductID
ORDER BY ProductName ASC;
GO
```

```
USE AdventureWorks2008;
GO
SELECT DISTINCT JobTitle
FROM HumanResources.Employee
ORDER BY JobTitle;
GO
```

# 2. Using correlated subqueries

```
USE AdventureWorks2008;
GO
SELECT DISTINCT Name
FROM Production.Product AS p
WHERE EXISTS
    (SELECT *
     FROM Production.ProductModel AS pm
     WHERE p.ProductModelID = pm.ProductModelID
           AND pm.Name LIKE 'Long-Sleeve Logo Jersey%');
GO

-- OR

USE AdventureWorks2012;
GO
SELECT DISTINCT Name
FROM Production.Product
WHERE ProductModelID IN
    (SELECT ProductModelID
     FROM Production.ProductModel
     WHERE Name LIKE 'Long-Sleeve Logo Jersey%');
GO

USE AdventureWorks2008;
GO
SELECT DISTINCT p.LastName, p.FirstName
FROM Person.Person AS p
JOIN HumanResources.Employee AS e
    ON e.BusinessEntityID = p.BusinessEntityID WHERE 5000.00 IN
    (SELECT Bonus
     FROM Sales.SalesPerson AS sp
     WHERE e.BusinessEntityID = sp.BusinessEntityID);
GO

USE AdventureWorks2008;
GO
SELECT p1.ProductModelID
FROM Production.Product AS p1
GROUP BY p1.ProductModelID
HAVING MAX(p1.ListPrice) >= ALL
    (SELECT AVG(p2.ListPrice)
     FROM Production.Product AS p2
     WHERE p1.ProductModelID = p2.ProductModelID);
GO

USE AdventureWorks2008;
GO
SELECT DISTINCT pp.LastName, pp.FirstName
FROM Person.Person pp JOIN HumanResources.Employee e
ON e.BusinessEntityID = pp.BusinessEntityID WHERE pp.BusinessEntityID IN
(SELECT SalesPersonID
FROM Sales.SalesOrderHeader
WHERE SalesOrderID IN
(SELECT SalesOrderID
FROM Sales.SalesOrderDetail
WHERE ProductID IN
(SELECT ProductID
FROM Production.Product p
WHERE ProductNumber = 'BK-M68B-42')));
GO
```

## 3. Using GROUP BY

```
USE AdventureWorks2008;
GO
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
ORDER BY SalesOrderID;
GO
```

## 4. Using GROUP BY with multiple groups

```
USE AdventureWorks2008;
GO
SELECT ProductID, SpecialOfferID, AVG(UnitPrice) AS [Average Price],
    SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY ProductID, SpecialOfferID
ORDER BY ProductID;
GO
```

## 5. Using GROUP BY and WHERE

```
USE AdventureWorks2008;
GO
SELECT ProductModelID, AVG(ListPrice) AS [Average List Price]
FROM Production.Product
WHERE ListPrice > $1000
GROUP BY ProductModelID
ORDER BY ProductModelID;
GO
```

## 6. Using GROUP BY with an expression

```
USE AdventureWorks2008;
GO
SELECT AVG(OrderQty) AS [Average Quantity],
NonDiscountSales = (OrderQty * UnitPrice)
FROM Sales.SalesOrderDetail
GROUP BY (OrderQty * UnitPrice)
ORDER BY (OrderQty * UnitPrice) DESC;
GO
```

## 7. Using GROUP BY with ORDER BY

```
USE AdventureWorks2008;
GO
SELECT ProductID, AVG(UnitPrice) AS [Average Price]
FROM Sales.SalesOrderDetail
WHERE OrderQty > 10
GROUP BY ProductID
ORDER BY AVG(UnitPrice);
GO
```

## 8. Using the HAVING clause

```
USE AdventureWorks2008;
GO
SELECT ProductID
FROM Sales.SalesOrderDetail
GROUP BY ProductID
HAVING AVG(OrderQty) > 5
ORDER BY ProductID;
GO

USE AdventureWorks2012 ;
GO
SELECT SalesOrderID, CarrierTrackingNumber
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID, CarrierTrackingNumber
```

```
HAVING CarrierTrackingNumber LIKE '4BD%'
ORDER BY SalesOrderID ;
GO
```

## 9. Using HAVING and GROUP BY

```
USE AdventureWorks2008;
GO
SELECT ProductID
FROM Sales.SalesOrderDetail
WHERE UnitPrice < 25.00
GROUP BY ProductID
HAVING AVG(OrderQty) > 5
ORDER BY ProductID;
GO
```

## 10. Using HAVING with SUM and AVG

```
USE AdventureWorks2008;
GO
SELECT ProductID, AVG(OrderQty) AS AverageQuantity, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
GROUP BY ProductID
HAVING SUM(LineTotal) > $1000000.00
AND AVG(OrderQty) < 3;
GO

USE AdventureWorks2008;
GO
SELECT ProductID, Total = SUM(LineTotal)
FROM Sales.SalesOrderDetail
GROUP BY ProductID
HAVING SUM(LineTotal) > $2000000.00;
GO

USE AdventureWorks2008;
GO
SELECT ProductID, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
GROUP BY ProductID
HAVING COUNT(*) > 1500;
GO
```

## 11. Using the SQL-92 CROSS JOIN syntax

```
SELECT e.BusinessEntityID, d.Name AS Department
FROM HumanResources.Employee AS e
CROSS JOIN HumanResources.Department AS d
ORDER BY e.BusinessEntityID, d.Name ;
```

## 12. Using the SQL-92 FULL OUTER JOIN syntax

```
SELECT p.Name, sod.SalesOrderID
FROM Production.Product AS p
FULL OUTER JOIN Sales.SalesOrderDetail AS sod
ON p.ProductID = sod.ProductID
ORDER BY p.Name ;
```

## 13. Using the SQL-92 LEFT OUTER JOIN syntax

```
SELECT p.Name, sod.SalesOrderID
FROM Production.Product AS p
LEFT OUTER JOIN Sales.SalesOrderDetail AS sod
ON p.ProductID = sod.ProductID
ORDER BY p.Name ;
```

## 14. Using the SQL-92 INNER JOIN syntax

```
SELECT p.Name, sod.SalesOrderID
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail AS sod
ON p.ProductID = sod.ProductID
ORDER BY p.Name ;
```

## 15. Using the SQL-92 RIGHT OUTER JOIN syntax

```
SELECT st.Name AS Territory, sp.BusinessEntityID
FROM Sales.SalesTerritory AS st
RIGHT OUTER JOIN Sales.SalesPerson AS sp
ON st.TerritoryID = sp.TerritoryID ;
```