Chapter 9

How to use functions

Objectives

Applied

- Code queries that require any of the functions presented in this chapter for working with string, numeric, and date/time data.
- Code queries that use any of the general purpose functions presented in this chapter.

Knowledge

- Describe how the use of functions can solve the problems associated with (1) sorting string data that contains numeric values, and (2) doing date or time searches.

Some of the string functions

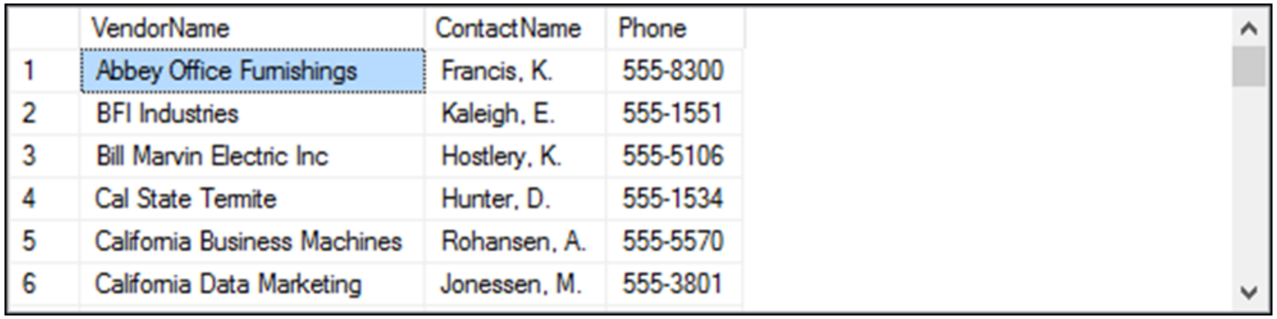
```
LEN(string)
LTRIM(string)
RTRIM(string)
TRIM(string)
LEFT(string,length)
RIGHT(string,length)
SUBSTRING(string,start,length)
REPLACE(search,find,replace)
TRANSLATE(search,find,replace)
REVERSE(string)
CHARINDEX(find,search[,start])
PATINDEX(find,search)
CONCAT(value1,value2[,value3]...)
CONCAT_WS(delimiter,value1,value2[,value3]...)
LOWER(string)
UPPER(string)
SPACE(integer)
```

String function examples

Function	Result
LEN('SQL Server')	10
LEN(' SQL Server ')	12
LEFT('SQL Server', 3)	'SQL'
LTRIM(' SQL Server ')	'SQL Server '
RTRIM(' SQL Server ')	' SQL Server'
TRIM(' SQL Server ')	'SQL Server'
LOWER('SQL Server')	'sql server'
UPPER('ca')	CA
PATINDEX('%v_r%', 'SQL Server')	8
CHARINDEX('SQL', ' SQL Server')	3
CHARINDEX('-', '(559) 555-1212')	10
SUBSTRING('(559) 555-1212', 7, 8)	555-1212
REPLACE(RIGHT('(559) 555-1212', 13), ') ', '-’)	559-555-1212
TRANSLATE('(XDG) 197.TS224', '().', '[]-')	[XDG] 197-TS224
CONCAT('Run time: ',1.52,' seconds')	Run time: 1.52 seconds
CONCAT_WS('.', 559, 555, 1212)	559.555.1212

A SELECT statement that uses three functions

```
Select VendorName, VendorContactLName + ', '
       + LEFT(VendorContactFName, 1)
       + '.' AS ContactName, RIGHT(VendorPhone, 8) AS Phone
FROM Vendors
WHERE SUBSTRING(VendorPhone, 2, 3) = 559
ORDER BY VendorName;
```

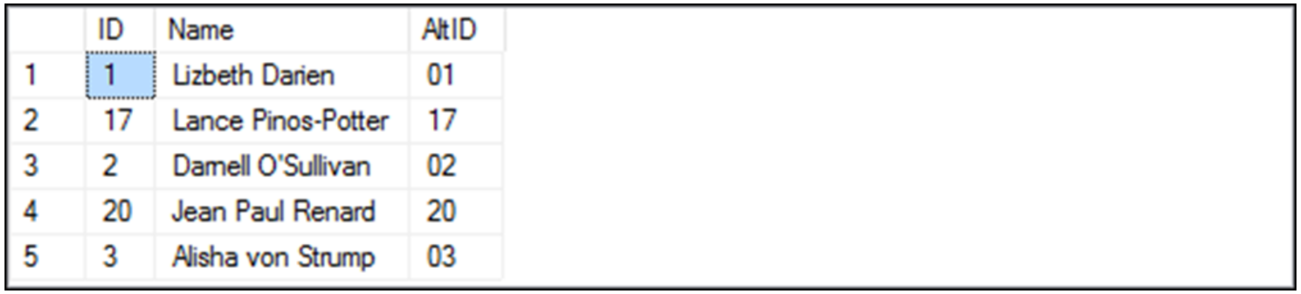


	VendorName	ContactName	Phone
1	Abbey Office Furnishings	Francis, K.	555-8300
2	BFI Industries	Kaleigh, E.	555-1551
3	Bill Marvin Electric Inc	Hostlery, K.	555-5106
4	Cal State Termite	Hunter, D.	555-1534
5	California Business Machines	Rohansen, A.	555-5570
6	California Data Marketing	Jonessen, M.	555-3801

How to sort by a string column that contains numbers (part 1)

Sorted by the ID column

```
SELECT * FROM StringSample
ORDER BY ID;
```

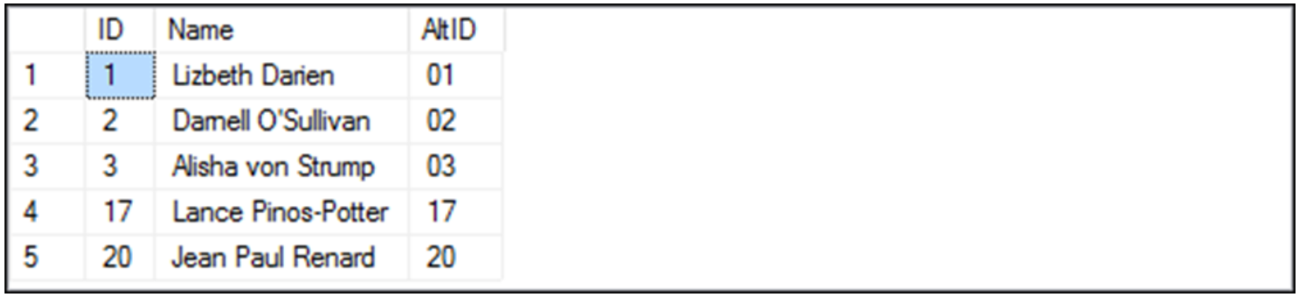


	ID	Name	AltID
1	1	Lizbeth Darien	01
2	17	Lance Pinos-Potter	17
3	2	Darnell O'Sullivan	02
4	20	Jean Paul Renard	20
5	3	Alisha von Strump	03

How to sort by a string column that contains numbers (part 2)

Sorted by the ID column cast to an integer

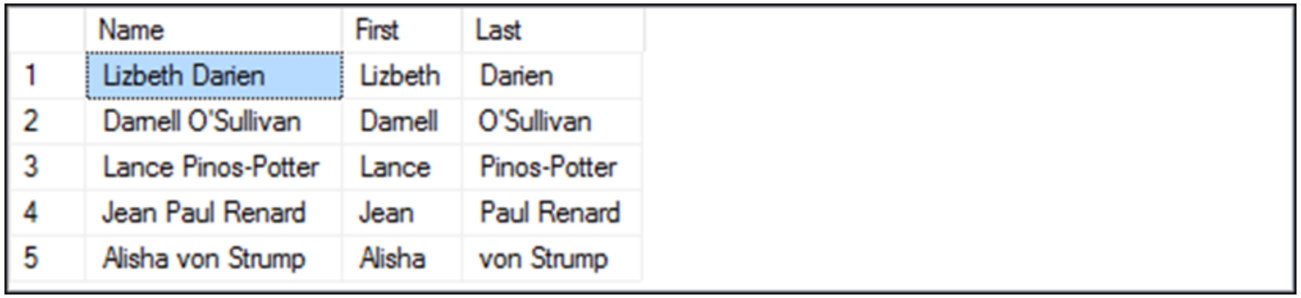
```
SELECT * FROM StringSample
ORDER BY CAST(ID AS int);
```



	ID	Name	AltID
1	1	Lizbeth Darien	01
2	2	Darnell O'Sullivan	02
3	3	Alisha von Strump	03
4	17	Lance Pinos-Potter	17
5	20	Jean Paul Renard	20

How to use the string functions to parse a string

```
SELECT Name,
    LEFT(Name, CHARINDEX(' ', Name) - 1) AS First,
    RIGHT(Name, LEN(Name) - CHARINDEX(' ', Name) ) AS Last
FROM StringSample;
```



	Name	First	Last
1	Lizbeth Darien	Lizbeth	Darien
2	Darnell O'Sullivan	Darnell	O'Sullivan
3	Lance Pinos-Potter	Lance	Pinos-Potter
4	Jean Paul Renard	Jean	Paul Renard
5	Alisha von Strump	Alisha	von Strump

Some of the numeric functions

```
ROUND(number,length[,function])
ISNUMERIC(expression)
ABS(number)
CEILING(number)
FLOOR(number)
SQUARE(float_number)
SQRT(float_number)
RAND([integer])
```

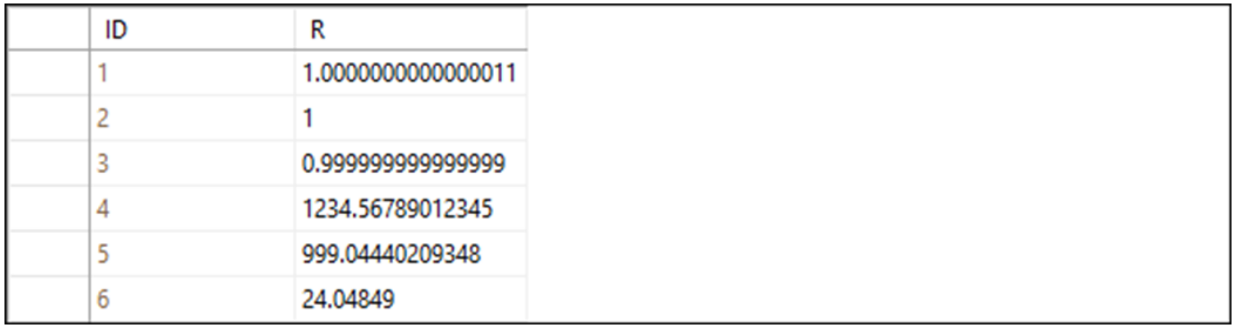
Examples that use the numeric functions (part 1)

Function	Result
`ROUND(12.5,0)`	`13.0`
`ROUND(12.4999,0)`	`12.0000`
`ROUND(12.4999,1)`	`12.5000`
`ROUND(12.4999,-1)`	`10.0000`
`ROUND(12.5,0,1)`	`12.0`
`ISNUMERIC(-1.25)`	`1`
`ISNUMERIC('SQL Server')`	`0`
`ISNUMERIC('2020-04-30')`	`0`

Examples that use the numeric functions (part 2)

Function	Result
`ABS(-1.25)`	`1.25`
`CEILING(-1.25)`	`-1`
`FLOOR(-1.25)`	`-2`
`CEILING(1.25)`	`2`
`FLOOR(1.25)`	`1`
`SQUARE(5.2786)`	`27.86361796`
`SQRT(125.43)`	`11.199553562531`
`RAND()`	`0.243729`

The RealSample table



	ID	R
	1	1.0000000000000011
	2	1
	3	0.999999999999999
	4	1234.56789012345
	5	999.04440209348
	6	24.04849

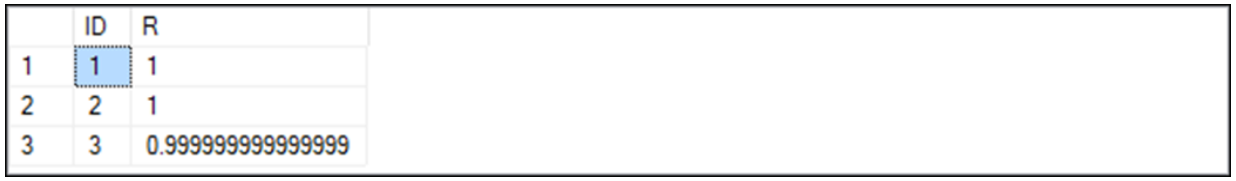
How to search for approximate real values

A SELECT statement that searches for a range of values

```
SELECT * FROM RealSample
WHERE R BETWEEN 0.99 AND 1.01;
```

A SELECT statement that searches for rounded values

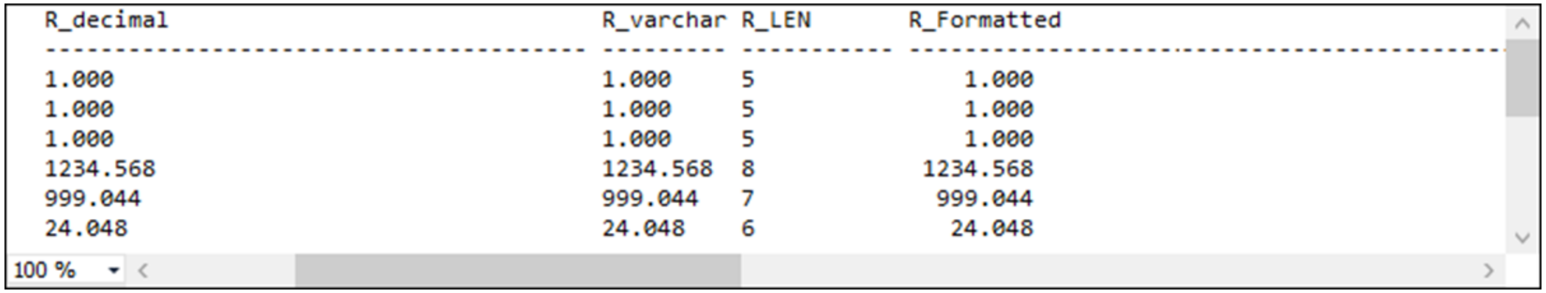
```
SELECT * FROM RealSample
WHERE ROUND(R,2) = 1;
```



	ID	R
1	1	1
2	2	1
3	3	0.999999999999999

A SELECT statement that formats real numbers

```
SELECT ID, R, CAST(R AS decimal(9,3)) AS R_decimal,
    CAST(CAST(R AS decimal(9,3)) AS varchar(9))
        AS R_varchar,
    LEN(CAST(CAST(R AS decimal(9,3)) AS varchar(9)))
        AS R_LEN,
    SPACE(9 - LEN(CAST(CAST(R AS decimal(9,3))
                  AS varchar(9)))) +
              CAST(CAST(R AS decimal(9,3))
                  AS varchar(9))
        AS R_Formatted
FROM RealSample;
```



```
R_decimal                                 R_varchar R_LEN       R_Formatted
----------------------------------------- --------- ----------- --------------------------------------------
1.000                                     1.000     5              1.000
1.000                                     1.000     5              1.000
1.000                                     1.000     5              1.000
1234.568                                  1234.568  8           1234.568
999.044                                   999.044   7            999.044
24.048                                    24.048    6             24.048
```

100 %

Some of the date/time functions (part 1)

```
GETDATE()
GETUTCDATE()

SYSDATETIME()
SYSUTCDATETIME()
SYSDATETIMEOFFSET()

DAY(date)
MONTH(date)
YEAR(date)
DATENAME(datepart,date)
DATEPART(datepart,date)
```

Some of the date/time functions (part 2)

```
DATEADD(datepart,number,date)
DATEDIFF(datepart,startdate,enddate)

TODATETIMEOFFSET(datetime2,tzoffset)
SWITCHOFFSET(datetimeoffset,tzoffset)

EOMONTH(startdate[,months])
DATEFROMPARTS(year,month,day)
ISDATE(expression)
```

Date part values and abbreviations (part 1)

Argument	Abbreviations
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw

Date part values and abbreviations (part 2)

Argument	Abbreviations
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns
tzoffset	tz

Examples that use date/time functions (part 1)

Function	Result
`GETDATE()`	`2020-04-30 14:10:13.813`
`GETUTCDATE()`	`2020-04-30 21:10:13.813`
`SYSDATETIME()`	`2020-04-30 14:10:13.8160822`
`SYSUTCDATETIME()`	`2020-04-30 21:10:13.8160822`
`SYSDATETIMEOFFSET()`	`2020-04-30 14:10:13.8160822 -07.00`
`MONTH('2020-04-30')`	`4`
`DATEPART(month,'2020-04-30')`	`4`
`DATENAME(month,'2020-04-30')`	`April`
`DATENAME(m,'2020-04-30')`	`April`

Examples that use date/time functions (part 2)

Function	Result
`EOMONTH('2020-02-01')`	`2020-02-29`
`EOMONTH('2020-02-01',2)`	`2020-04-30`
`DATEFROMPARTS(2020,4,3)`	`2020-04-03`
`ISDATE('2020-04-30')`	`1`
`ISDATE('2020-04-31')`	`0`
`ISDATE('23:59:59')`	`1`
`ISDATE('23:99:99')`	`0`

Examples that use the DAY, MONTH, and YEAR functions

Function	Result
`DAY('2020-04-30')`	`30`
`MONTH('2020-04-30')`	`4`
`YEAR('2020-04-30')`	`2020`

Examples that use the DATEPART function

Function	Result
`DATEPART(day, '2020-04-30 11:35:00')`	30
`DATEPART(month, '2020-04-30 11:35:00')`	4
`DATEPART(year, '2020-04-30 11:35:00')`	2020
`DATEPART(hour, '2020-04-30 11:35:00')`	11
`DATEPART(minute, '2020-04-30 11:35:00')`	35
`DATEPART(second, '2020-04-30 11:35:00')`	0
`DATEPART(quarter, '2020-04-30 11:35:00')`	2
`DATEPART(dayofyear, '2020-04-30 11:35:00')`	121
`DATEPART(week, '2020-04-30 11:35:00')`	18
`DATEPART(weekday, '2020-04-30 11:35:00')`	5
`DATEPART(millisecond, '11:35:00.1234567')`	123
`DATEPART(microsecond, '11:35:00.1234567')`	123456
`DATEPART(nanosecond, '11:35:00.1234567')`	123456700
`DATEPART(tzoffset, '11:35:00.1234567 -07:00')`	-420

Examples that use the DATENAME function

Function	Result
`DATENAME(day, '2020-04-30 11:35:00')`	`30`
`DATENAME(month, '2020-04-30 11:35:00')`	`April`
`DATENAME(year, '2020-04-30 11:35:00')`	`2020`
`DATENAME(hour, '2020-04-30 11:35:00')`	`11`
`DATENAME(minute, '2020-04-30 11:35:00')`	`35`
`DATENAME(second, '2020-04-30 11:35:00')`	`0`
`DATENAME(quarter, '2020-04-30 11:35:00')`	`2`
`DATENAME(dayofyear, '2020-04-30 11:35:00’)`	`121`
`DATENAME(week, '2020-04-30 11:35:00’)`	`18`
`DATENAME(weekday, '2020-04-30 11:35:00')`	`Thursday`
`DATENAME(millisecond, '11:35:00.1234567')`	`123`
`DATENAME(microsecond, '11:35:00.1234567')`	`123456`
`DATENAME(nanosecond, '11:35:00.1234567')`	`123456700`
`DATENAME(tzoffset, '11:35:00.1234567 -07:00')`	`-07:00`

Examples that use the DATEADD function

Function	Result
`DATEADD(day, 1, '2020-04-30 11:35:00')`	`2020-05-01 11:35:00.000`
`DATEADD(month, 1, '2020-04-30 11:35:00')`	`2020-05-30 11:35:00.000`
`DATEADD(year, 1, '2020-04-30 11:35:00')`	`2021-04-30 11:35:00.000`
`DATEADD(hour, 1, '2020-04-30 11:35:00')`	`2020-04-30 12:35:00.000`
`DATEADD(minute, 1, '2020-04-30 11:35:00')`	`2020-04-30 11:36:00.000`
`DATEADD(second, 1, '2020-04-30 11:35:00')`	`2020-04-30 11:35:01.000`
`DATEADD(quarter, 1, '2020-04-30 11:35:00')`	`2020-07-30 11:35:00.000`
`DATEADD(week, 1, '2020-04-30 11:35:00')`	`2020-05-07 11:35:00.000`
`DATEADD(month, -1, '2020-04-30 11:35:00')`	`2020-03-30 11:35:00.000`
`DATEADD(year, 1.5, '2020-04-30 11:35:00')`	`2021-04-30 11:35:00.000`

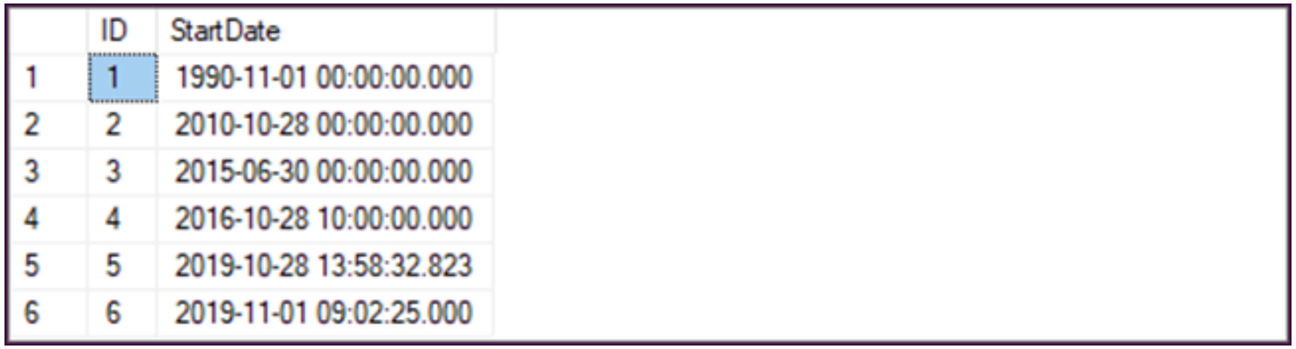
Examples that use the DATEDIFF function

Function	Result
`DATEDIFF(day, '2019-07-01', '2020-04-30')`	304
`DATEDIFF(month, '2019-07-01', '2020-04-30')`	9
`DATEDIFF(year, '2019-07-01', '2020-04-30')`	1
`DATEDIFF(hour, '06:46:45', '11:35:00')`	5
`DATEDIFF(minute, '06:46:45', '11:35:00')`	289
`DATEDIFF(second, '06:46:45', '11:35:00')`	17295
`DATEDIFF(quarter, '2019-07-01', '2020-04-30')`	3
`DATEDIFF(week, '2019-07-01', '2020-04-30')`	43
`DATEDIFF(day, '2020-04-30', '2019-07-01')`	-304

Examples that use the addition and subtraction operators

Operation	Result
`CAST('2020-04-30 11:35:00' AS smalldatetime) + 1`	`2020-05-01 11:35:00`
`CAST('2020-04-30 11:35:00' AS smalldatetime) - 1`	`2020-04-29 11:35:00`
`CAST(CAST('2020-04-30' AS datetime) - CAST('2019-07-01' AS datetime) AS int)`	`304`

The DateSample table: Date searches



	ID	StartDate
1	1	1990-11-01 00:00:00.000
2	2	2010-10-28 00:00:00.000
3	3	2015-06-30 00:00:00.000
4	4	2016-10-28 10:00:00.000
5	5	2019-10-28 13:58:32.823
6	6	2019-11-01 09:02:25.000

A search condition that fails to return a row

```
SELECT * FROM DateSample
WHERE StartDate = '2019-10-28';
```

SELECT statements that ignore time values (part 1)

Use the date type to remove time values (SQL Server 2008 or later)

```
SELECT * FROM DateSample
WHERE CONVERT(date, StartDate) = '2019-10-28';
```

Search for a range of dates

```
SELECT * FROM DateSample
WHERE StartDate >= '2019-10-28' AND
      StartDate < '2019-10-29';
```

Search for month, day, and year components

```
SELECT * FROM DateSample
WHERE MONTH(StartDate) = 10 AND
      DAY(StartDate) = 28 AND
      YEAR(StartDate) = 2019;
```

SELECT statements that ignore time values (part 2)

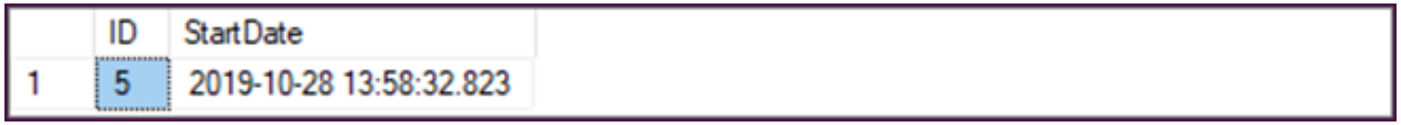
Use the CAST function to remove time values

```
SELECT * FROM DateSample
WHERE CAST(CAST(StartDate AS char(11)) AS datetime)
    = '2019-10-28';
```

Use the CONVERT function to remove time values

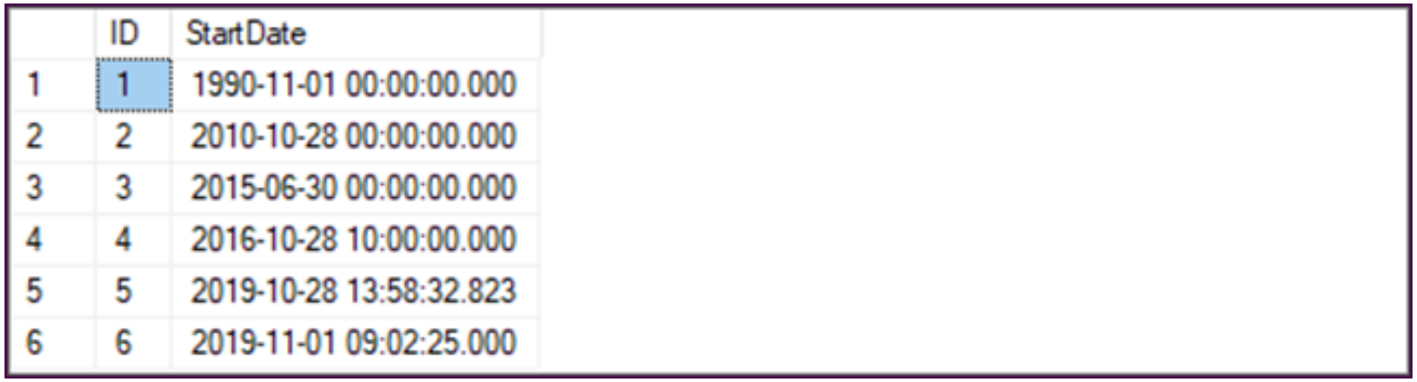
```
SELECT * FROM DateSample
WHERE CONVERT(datetime, CONVERT(char(10), StartDate,
    110)) = '2019-10-28';
```

The result set



	ID	StartDate
1	5	2019-10-28 13:58:32.823

The DateSample table: Time searches



	ID	StartDate
1	1	1990-11-01 00:00:00.000
2	2	2010-10-28 00:00:00.000
3	3	2015-06-30 00:00:00.000
4	4	2016-10-28 10:00:00.000
5	5	2019-10-28 13:58:32.823
6	6	2019-11-01 09:02:25.000

Two search conditions that fail to return a row

```
SELECT * FROM DateSample
WHERE StartDate = CAST('10:00:00' AS datetime);

SELECT * FROM DateSample
WHERE StartDate >= '09:00:00' AND
    StartDate < '12:59:59:999';
```

Two SELECT statements that ignore date values

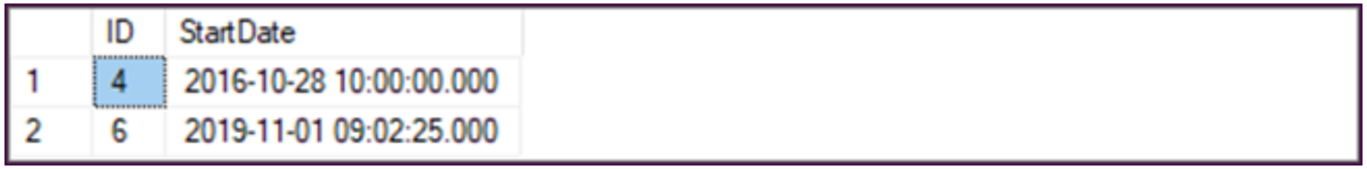
Use the time type to remove date values (SQL Server 2008 or later)

```
SELECT * FROM DateSample
WHERE CONVERT(time, StartDate) >= '09:00:00' AND
      CONVERT(time, StartDate) < '12:59:59:999';
```

Use the CONVERT function to remove date values (prior to SQL Server 2008)

```
SELECT * FROM DateSample
WHERE CONVERT(datetime, CONVERT(char(12), StartDate, 8))
          >= '09:00:00' AND
      CONVERT(datetime, CONVERT(char(12), StartDate, 8))
          < '12:59:59:999';
```

The result set



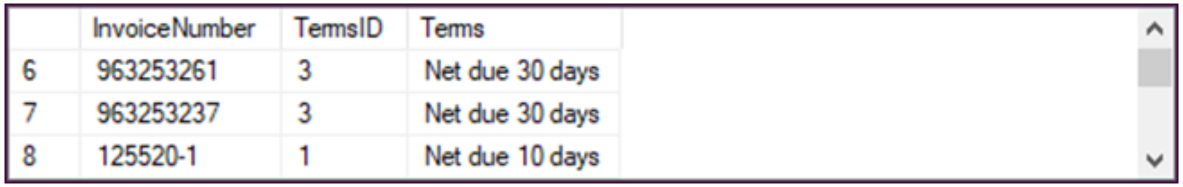
	ID	StartDate
1	4	2016-10-28 10:00:00.000
2	6	2019-11-01 09:02:25.000

The syntax of the simple CASE function

```
CASE input_expression
    WHEN when_expression_1 THEN result_expression_1
   [WHEN when_expression_2 THEN result_expression_2]...
   [ELSE else_result_expression]
END
```

A SELECT statement with a simple CASE function

```
SELECT InvoiceNumber, TermsID,
    CASE TermsID
        WHEN 1 THEN 'Net due 10 days'
        WHEN 2 THEN 'Net due 20 days'
        WHEN 3 THEN 'Net due 30 days'
        WHEN 4 THEN 'Net due 60 days'
        WHEN 5 THEN 'Net due 90 days'
    END AS Terms
FROM Invoices;
```



	InvoiceNumber	TermsID	Terms
6	963253261	3	Net due 30 days
7	963253237	3	Net due 30 days
8	125520-1	1	Net due 10 days

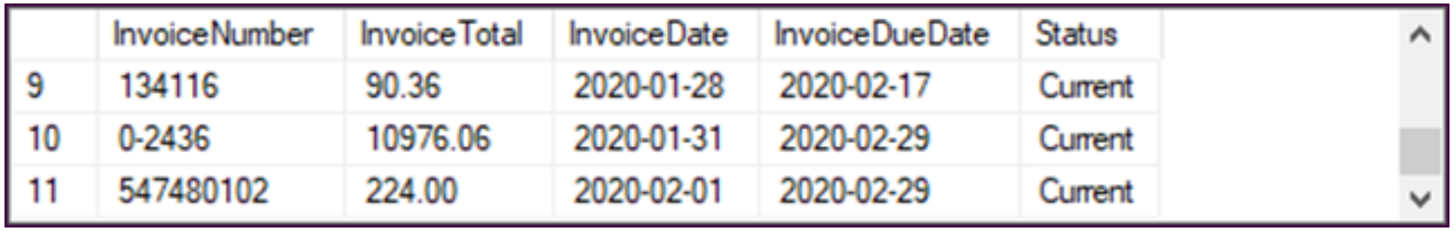
The syntax of the searched CASE function

```
CASE
    WHEN conditional_expression_1 THEN result_expression_1
   [WHEN conditional_expression_2
        THEN result_expression_2]...
   [ELSE else_result_expression]
END
```

A SELECT with a searched CASE function

```
SELECT InvoiceNumber, InvoiceTotal, InvoiceDate,
    InvoiceDueDate,
    CASE
        WHEN DATEDIFF(day, InvoiceDueDate, GETDATE()) > 30
            THEN 'Over 30 days past due'
        WHEN DATEDIFF(day, InvoiceDueDate, GETDATE()) > 0
            THEN '1 to 30 days past due'
        ELSE 'Current'
    END AS Status
FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0;
```

The result set for the searched CASE example



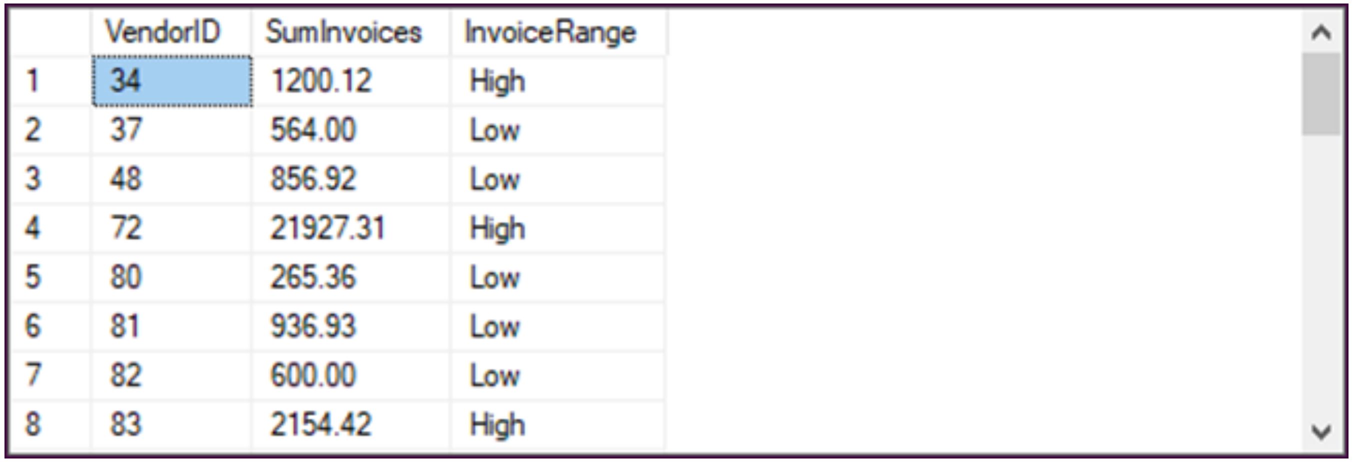
	InvoiceNumber	InvoiceTotal	InvoiceDate	InvoiceDueDate	Status
9	134116	90.36	2020-01-28	2020-02-17	Current
10	0-2436	10976.06	2020-01-31	2020-02-29	Current
11	547480102	224.00	2020-02-01	2020-02-29	Current

The syntax of the IIF function

```
IIF(conditional_expression, true_value, false_value)
```

A SELECT statement with an IIF function

```
SELECT VendorID, SUM(InvoiceTotal) AS SumInvoices,
    IIF(SUM(InvoiceTotal) < 1000, 'Low', 'High')
        AS InvoiceRange
FROM Invoices
GROUP BY VendorID;
```



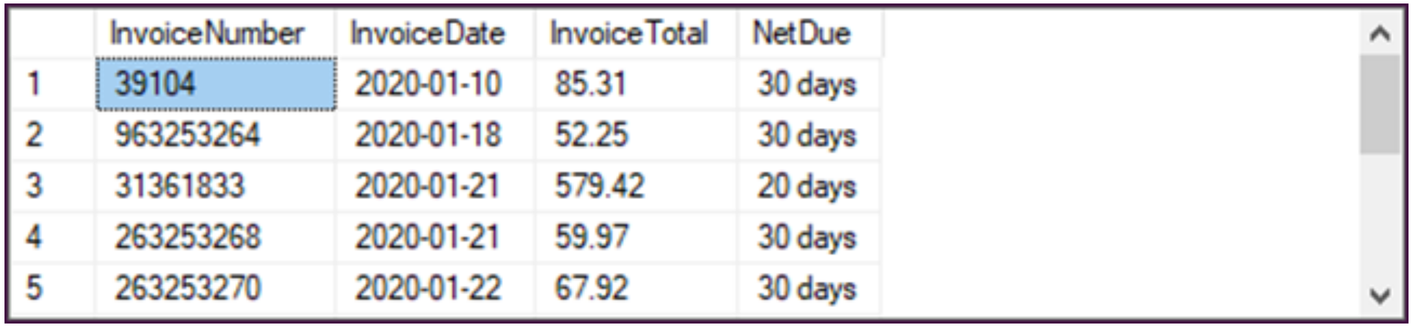
	VendorID	SumInvoices	InvoiceRange
1	34	1200.12	High
2	37	564.00	Low
3	48	856.92	Low
4	72	21927.31	High
5	80	265.36	Low
6	81	936.93	Low
7	82	600.00	Low
8	83	2154.42	High

The syntax of the CHOOSE function

```
CHOOSE(index, value1, value2 [,value3]...)
```

A SELECT statement with a CHOOSE function

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
    CHOOSE(TermsID, '10 days', '20 days', '30 days',
        '60 days', '90 days') AS NetDue
FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0;
```



	InvoiceNumber	InvoiceDate	InvoiceTotal	NetDue
1	39104	2020-01-10	85.31	30 days
2	963253264	2020-01-18	52.25	30 days
3	31361833	2020-01-21	579.42	20 days
4	263253268	2020-01-21	59.97	30 days
5	263253270	2020-01-22	67.92	30 days

The syntax of the COALESCE function

```
COALESCE(expression_1 [, expression_2]...)
```

The syntax of the ISNULL function

```
ISNULL(check_expression, replacement_value)
```

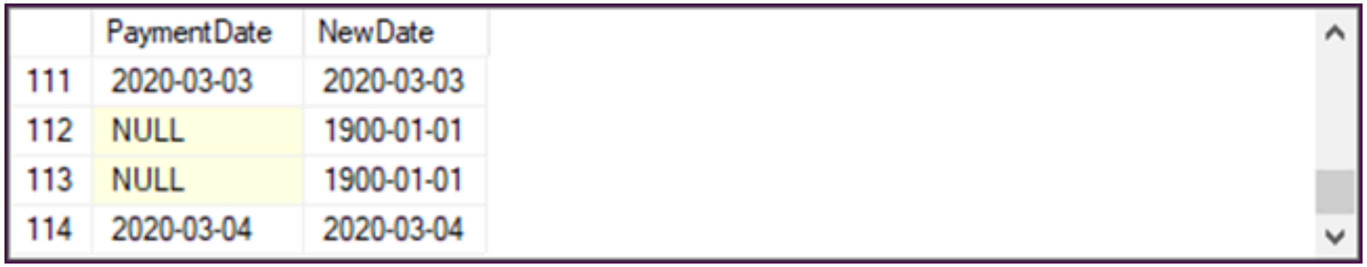
A SELECT statement with a COALESCE function

```
SELECT PaymentDate,
    COALESCE(PaymentDate, '1900-01-01') AS NewDate
FROM Invoices;
```

The same statement with an ISNULL function

```
SELECT PaymentDate,
    ISNULL(PaymentDate, '1900-01-01') AS NewDate
FROM Invoices;
```

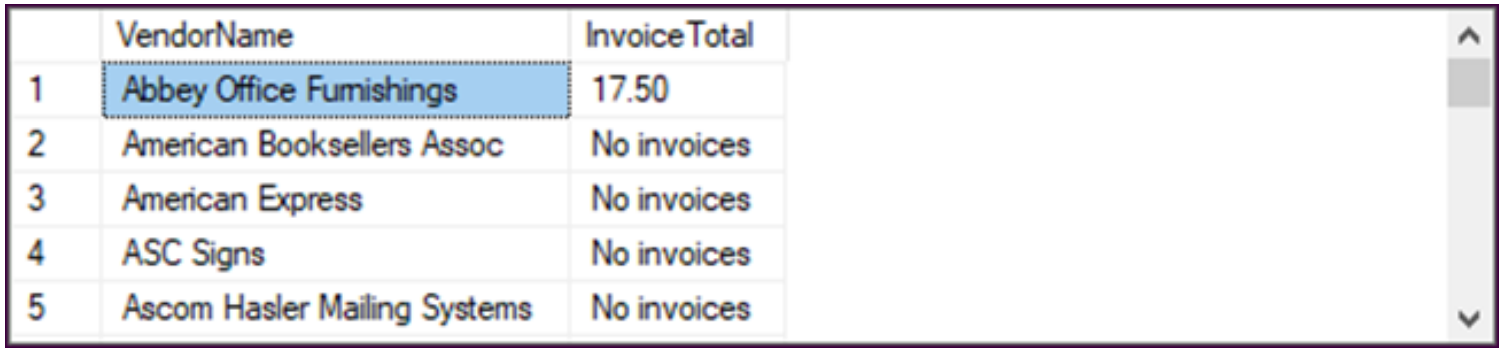
The result set for the ISNULL example



	PaymentDate	NewDate
111	2020-03-03	2020-03-03
112	NULL	1900-01-01
113	NULL	1900-01-01
114	2020-03-04	2020-03-04

A SELECT statement that substitutes a different data type

```
SELECT VendorName,
    COALESCE(CAST(InvoiceTotal AS varchar), 'No invoices')
        AS InvoiceTotal
FROM Vendors LEFT JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
ORDER BY VendorName;
```



	VendorName	InvoiceTotal
1	Abbey Office Furnishings	17.50
2	American Booksellers Assoc	No invoices
3	American Express	No invoices
4	ASC Signs	No invoices
5	Ascom Hasler Mailing Systems	No invoices

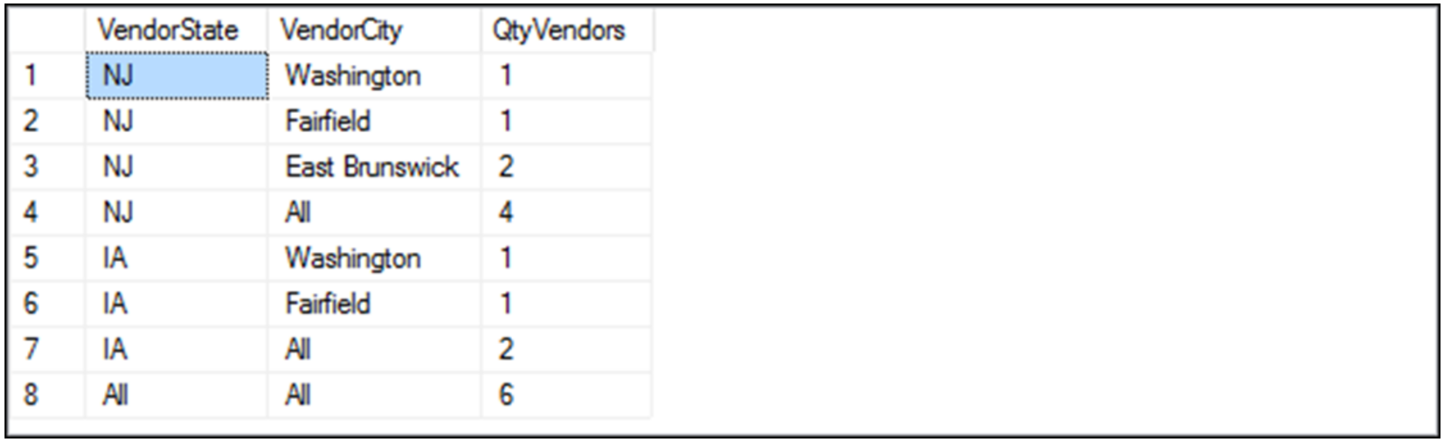
The syntax of the GROUPING function

```
GROUPING(column_name)
```

A summary query with a GROUPING function

```
SELECT
    CASE
        WHEN GROUPING(VendorState) = 1 THEN 'All'
        ELSE VendorState
    END AS VendorState,
    CASE
        WHEN GROUPING(VendorCity) = 1 THEN 'All'
        ELSE VendorCity
    END AS VendorCity,
    COUNT(*) AS QtyVendors
FROM Vendors
WHERE VendorState IN ('IA', 'NJ')
GROUP BY VendorState, VendorCity WITH ROLLUP
ORDER BY VendorState DESC, VendorCity DESC;
```

The result set for the GROUPING example



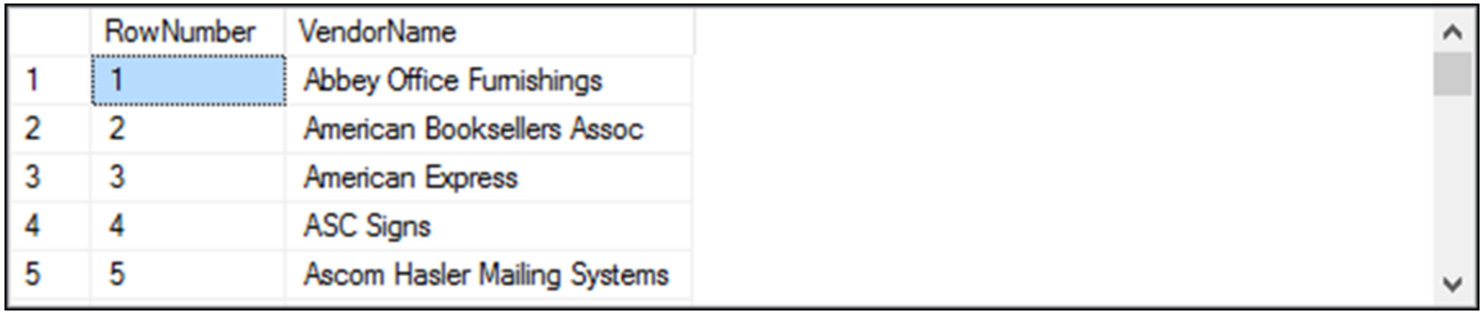
	VendorState	VendorCity	QtyVendors
1	NJ	Washington	1
2	NJ	Fairfield	1
3	NJ	East Brunswick	2
4	NJ	All	4
5	IA	Washington	1
6	IA	Fairfield	1
7	IA	All	2
8	All	All	6

The syntax for the four ranking functions

```
ROW_NUMBER()
    OVER ([partition_by_clause] order_by_clause)
RANK()
    OVER ([partition_by_clause] order_by_clause)
DENSE_RANK()
    OVER ([partition_by_clause] order_by_clause)
NTILE(integer_expression)
    OVER ([partition_by_clause] order_by_clause)
```

A query with a ROW_NUMBER function

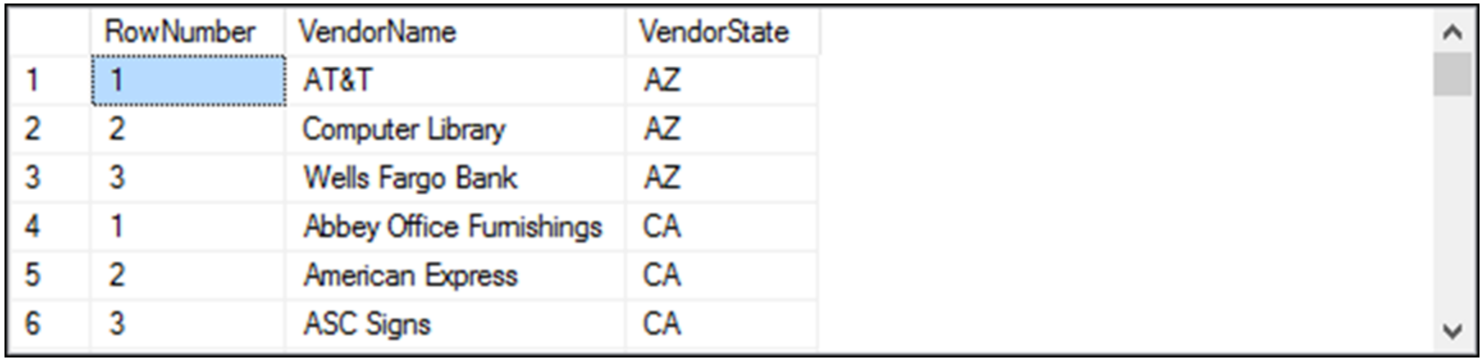
```
SELECT ROW_NUMBER() OVER(ORDER BY VendorName) AS RowNumber,
    VendorName
FROM Vendors;
```



	RowNumber	VendorName
1	1	Abbey Office Furnishings
2	2	American Booksellers Assoc
3	3	American Express
4	4	ASC Signs
5	5	Ascom Hasler Mailing Systems

A query that uses the PARTITION BY clause

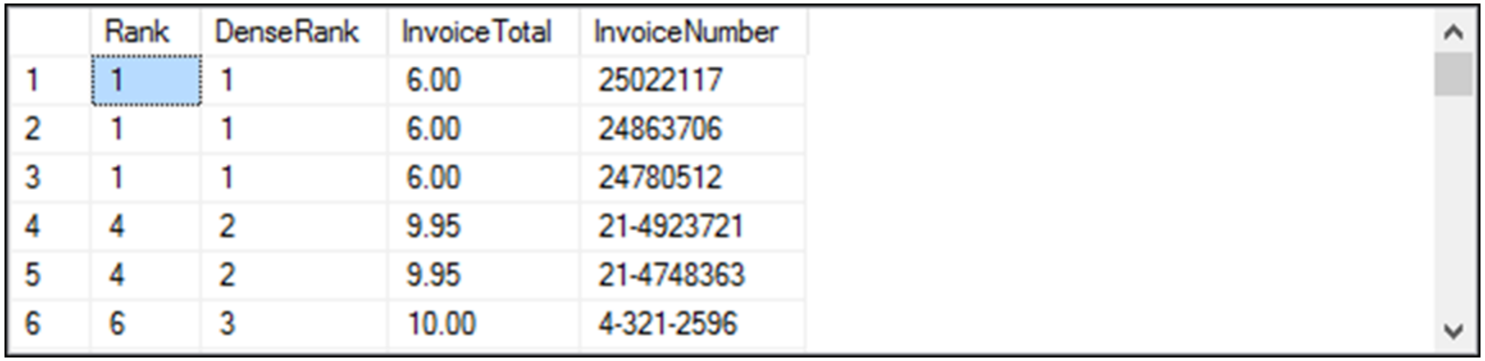
```
SELECT ROW_NUMBER() OVER(PARTITION BY VendorState
    ORDER BY VendorName) As RowNumber, VendorName,
    VendorState
FROM Vendors;
```



	RowNumber	VendorName	VendorState
1	1	AT&T	AZ
2	2	Computer Library	AZ
3	3	Wells Fargo Bank	AZ
4	1	Abbey Office Furnishings	CA
5	2	American Express	CA
6	3	ASC Signs	CA

A query with RANK and DENSE_RANK functions

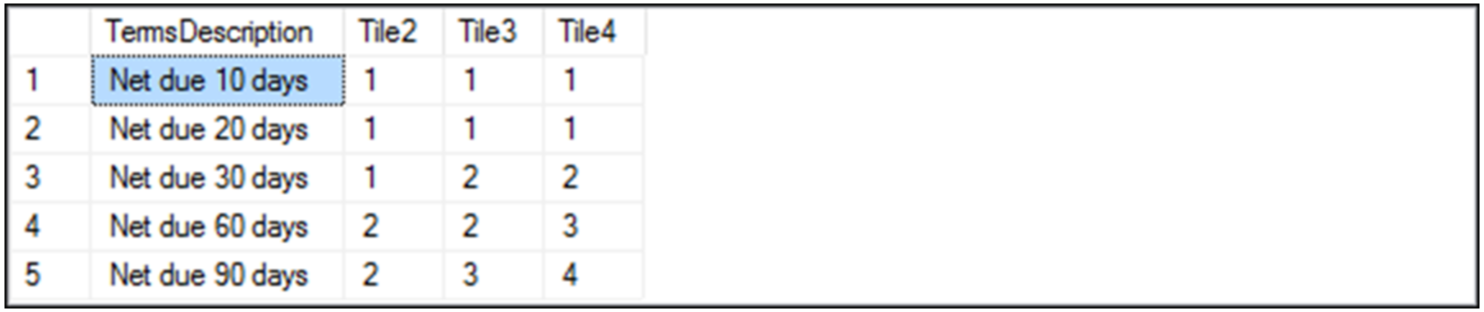
```
SELECT RANK() OVER (ORDER BY InvoiceTotal) As Rank,
       DENSE_RANK() OVER (ORDER BY InvoiceTotal)
            As DenseRank, InvoiceTotal, InvoiceNumber
FROM Invoices;
```



	Rank	DenseRank	InvoiceTotal	InvoiceNumber
1	1	1	6.00	25022117
2	1	1	6.00	24863706
3	1	1	6.00	24780512
4	4	2	9.95	21-4923721
5	4	2	9.95	21-4748363
6	6	3	10.00	4-321-2596

A query that uses the NTILE function

```
SELECT TermsDescription,
    NTILE(2) OVER (ORDER BY TermsID) AS Tile2,
    NTILE(3) OVER (ORDER BY TermsID) AS Tile3,
    NTILE(4) OVER (ORDER BY TermsID) AS Tile4
FROM Terms;
```



	TermsDescription	Tile2	Tile3	Tile4
1	Net due 10 days	1	1	1
2	Net due 20 days	1	1	1
3	Net due 30 days	1	2	2
4	Net due 60 days	2	2	3
5	Net due 90 days	2	3	4

The syntax of the analytic functions

```
{FIRST_VALUE|LAST_VALUE}(scalar_expression)
    OVER ([partition_by_clause] order_by_clause
          [rows_range_clause])

{LEAD|LAG}(scalar_expression [, offset [, default]])
    OVER ([partition_by_clause] order_by_clause)

{PERCENT_RANK()|CUME_DIST}
    OVER ([partition_by_clause] order_by_clause)

{PERCENTILE_CONT|PERCENTILE_DISC}(numeric_literal)
    WITHIN GROUP (ORDER BY expression [ASC|DESC])
    OVER (partition_by_clause)
```

The columns in the SalesReps table

Column name	Data type
RepID	int
RepFirstName	varchar(50)
RepLastName	varchar(50)

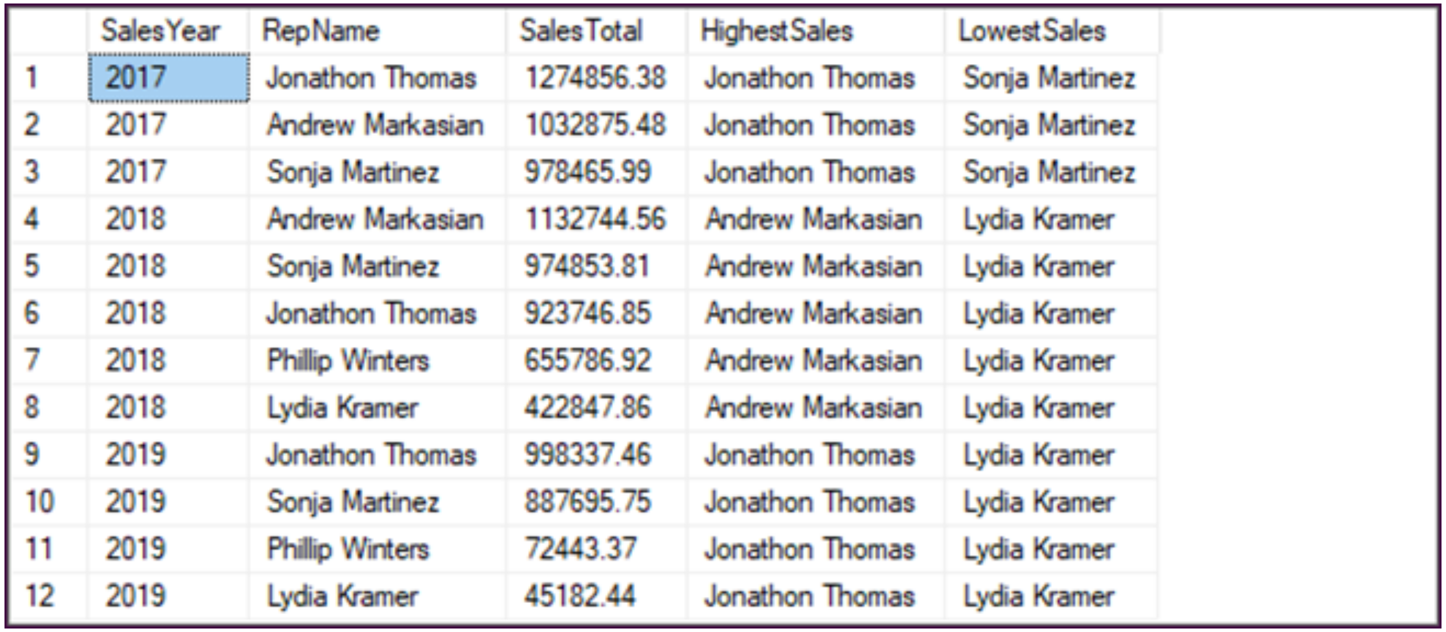
The columns in the SalesTotals table

Column name	Data type
RepID	int
SalesYear	char(4)
SalesTotal	money

A query that uses the FIRST_VALUE and LAST_VALUE functions

```
SELECT SalesYear, RepFirstName + ' ' +
       RepLastName AS RepName, SalesTotal,
       FIRST_VALUE(RepFirstName + ' ' + RepLastName)
           OVER (PARTITION BY SalesYear
               ORDER BY SalesTotal DESC)
           AS HighestSales,
       LAST_VALUE(RepFirstName + ' ' + RepLastName)
           OVER (PARTITION BY SalesYear
               ORDER BY SalesTotal DESC
              RANGE BETWEEN UNBOUNDED PRECEDING AND
                            UNBOUNDED FOLLOWING)
           AS LowestSales
FROM SalesTotals JOIN SalesReps
  ON SalesTotals.RepID = SalesReps.RepID;
```

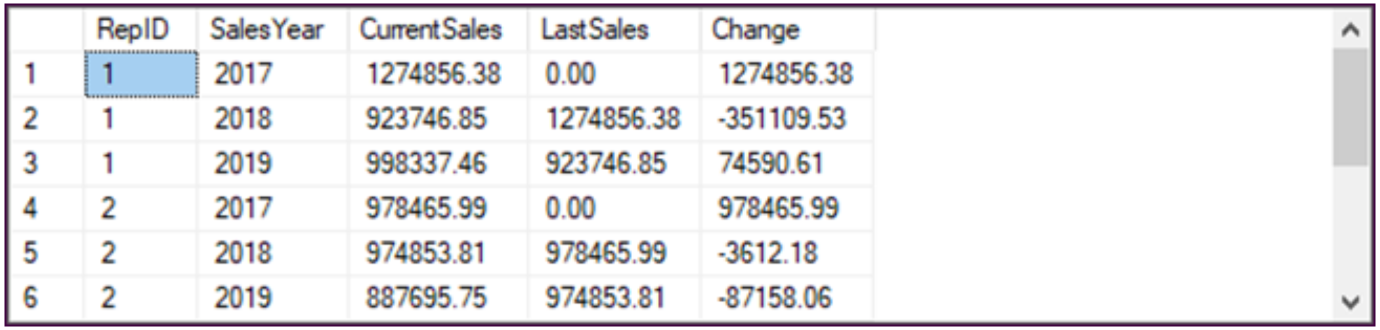
The result set for the FIRST_VALUE and LAST_VALUE example



	SalesYear	RepName	SalesTotal	HighestSales	LowestSales
1	2017	Jonathon Thomas	1274856.38	Jonathon Thomas	Sonja Martinez
2	2017	Andrew Markasian	1032875.48	Jonathon Thomas	Sonja Martinez
3	2017	Sonja Martinez	978465.99	Jonathon Thomas	Sonja Martinez
4	2018	Andrew Markasian	1132744.56	Andrew Markasian	Lydia Kramer
5	2018	Sonja Martinez	974853.81	Andrew Markasian	Lydia Kramer
6	2018	Jonathon Thomas	923746.85	Andrew Markasian	Lydia Kramer
7	2018	Phillip Winters	655786.92	Andrew Markasian	Lydia Kramer
8	2018	Lydia Kramer	422847.86	Andrew Markasian	Lydia Kramer
9	2019	Jonathon Thomas	998337.46	Jonathon Thomas	Lydia Kramer
10	2019	Sonja Martinez	887695.75	Jonathon Thomas	Lydia Kramer
11	2019	Phillip Winters	72443.37	Jonathon Thomas	Lydia Kramer
12	2019	Lydia Kramer	45182.44	Jonathon Thomas	Lydia Kramer

A query that uses the LAG function

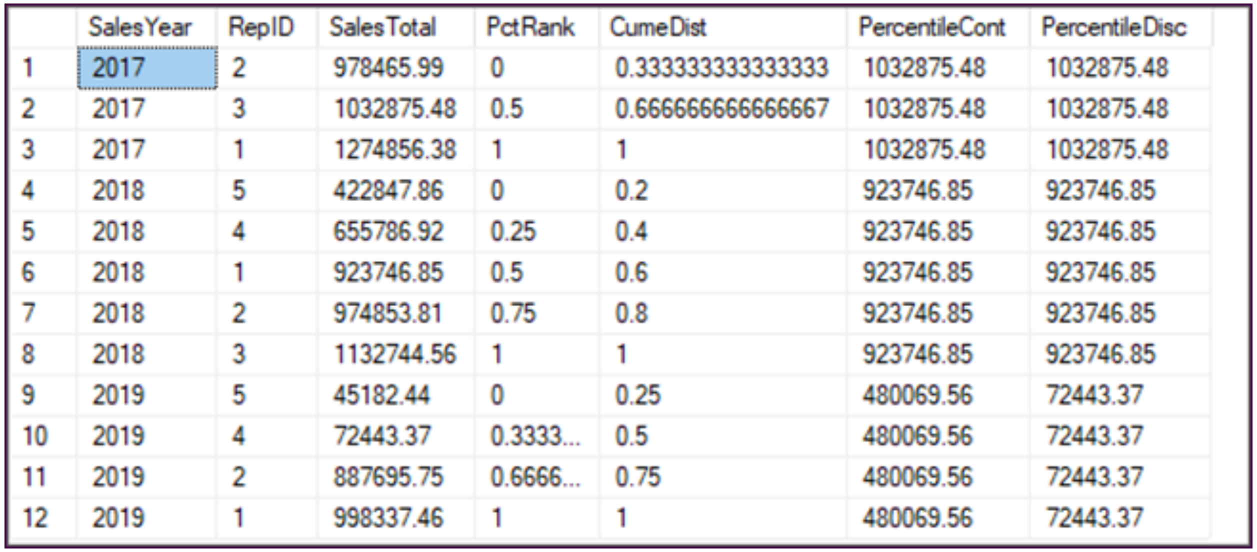
```
SELECT RepID, SalesYear, SalesTotal AS CurrentSales,
    LAG(SalesTotal, 1, 0)
        OVER (PARTITION BY RepID ORDER BY SalesYear)
            AS LastSales,
    SalesTotal - LAG(SalesTotal, 1, 0)
        OVER (PARTITION BY REPID ORDER BY SalesYear)
            AS Change
FROM SalesTotals;
```



	RepID	SalesYear	CurrentSales	LastSales	Change
1	1	2017	1274856.38	0.00	1274856.38
2	1	2018	923746.85	1274856.38	-351109.53
3	1	2019	998337.46	923746.85	74590.61
4	2	2017	978465.99	0.00	978465.99
5	2	2018	974853.81	978465.99	-3612.18
6	2	2019	887695.75	974853.81	-87158.06

A query that uses four more functions

```
SELECT SalesYear, RepID, SalesTotal,
    PERCENT_RANK() OVER (PARTITION BY SalesYear
        ORDER BY SalesTotal) AS PctRank,
    CUME_DIST() OVER (PARTITION BY SalesYear
        ORDER BY SalesTotal) AS CumeDist,
    PERCENTILE_CONT(.5) WITHIN GROUP (ORDER BY SalesTotal)
        OVER (PARTITION BY SalesYear) AS PercentileCont,
    PERCENTILE_DISC(.5) WITHIN GROUP (ORDER BY SalesTotal)
        OVER (PARTITION BY SalesYear) AS PercentileDisc
FROM SalesTotals;
```



	SalesYear	RepID	SalesTotal	PctRank	CumeDist	PercentileCont	PercentileDisc
1	2017	2	978465.99	0	0.333333333333333	1032875.48	1032875.48
2	2017	3	1032875.48	0.5	0.666666666666667	1032875.48	1032875.48
3	2017	1	1274856.38	1	1	1032875.48	1032875.48
4	2018	5	422847.86	0	0.2	923746.85	923746.85
5	2018	4	655786.92	0.25	0.4	923746.85	923746.85
6	2018	1	923746.85	0.5	0.6	923746.85	923746.85
7	2018	2	974853.81	0.75	0.8	923746.85	923746.85
8	2018	3	1132744.56	1	1	923746.85	923746.85
9	2019	5	45182.44	0	0.25	480069.56	72443.37
10	2019	4	72443.37	0.3333...	0.5	480069.56	72443.37
11	2019	2	887695.75	0.6666...	0.75	480069.56	72443.37
12	2019	1	998337.46	1	1	480069.56	72443.37

Terms to know

- Logical functions
- Ranking functions
- Analytic functions