Module 9

HOW TO
WORK WITH 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 module.

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.

THE STRING FUNCTIONS

- **LEN(string)** Returns the number of characters in the string. Leading spaces are included, but trailing spaces are not.
- **LTRIM(string)** Returns the string with any leading spaces removed.
- **RTRIM(string)** Returns the string with any trailing spaces removed.
- **LEFT(string,length)** Returns the specified number of characters from the beginning of the string.
- **RIGHT(string,length)** Returns the specified number of characters from the end of the string.
- **SUBSTRING(string,start,length)** Returns the specified number of characters from the string starting at the specified position.
- **REPLACE (search,find, replace)** Returns the search string with all occurrences of the find string replaced with the replace string.

THE STRING FUNCTIONS (cont.)

University of CINCINNATI

- **REVERSE(string)** Returns the string with the characters in reverse order.

- **CHARINDEX (find, search[, start])** Returns an integer that represents the position of the first occurrence of the find string in the search string starting at the specified position.

- **PATINDEX (find, search)** Returns an integer that represents the position of the first occurrence of the find pattern in the search string.

.

- **CONCAT(valuel,value2[,value3]** Returns a string that contains a concatenation of the specified values.
The values are implicitly converted to strings. A null value is converted to an empty string.

- **LOWER(string)** Returns the string converted to lowercase letters.

- **UPPER(string)** Returns the string converted to uppercase letters.

- **SPACE(integer)** Returns a string with the specified number of space characters (blanks).

THE STRING FUNCTIONS (cont.)

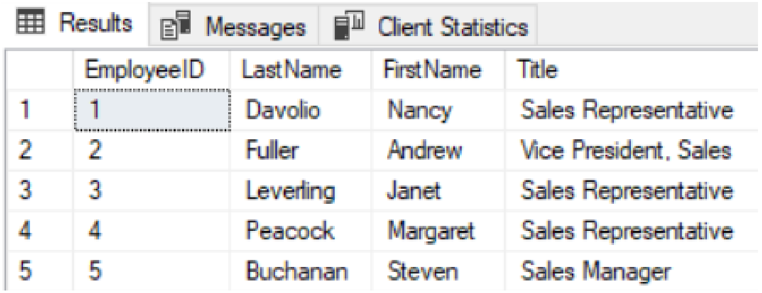
Function	Result
LEN(‘SQL Server’)	10
LEN(‘ SQL Server’)	12
LEFT(‘SQL Server’, 3)	‘SQL’
LTRIM(‘ SQL Server ’)	‘SQL Server’
RTRIM(‘ SQL Server ’)	‘ SQL Server’
RTRIM(RTRIM(‘ SQL Server ’))	‘SQL Server’
LOWER(‘SQL Server’)	‘sql server’
UPPER(‘ca’)	‘CA’
PATINDEX(‘%v_r%’, ‘SQL Server’)	8
CHARINDEX(‘SQL’, ‘ 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
CONCAT(‘Run time: ‘1.52, ‘seconds’)	Run time: 1.52 seconds

The LEFT() Function

- ❑ The **LEFT()** function: It extracts a number of characters from a string (starting from left)

➢ Original Employees table before the extraction

```
SELECT * FROM Employees;
```

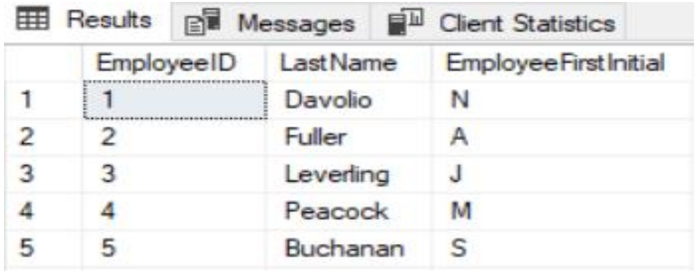


Results | Messages | Client Statistics

	EmployeeID	LastName	FirstName	Title
1	1	Davolio	Nancy	Sales Representative
2	2	Fuller	Andrew	Vice President, Sales
3	3	Leverling	Janet	Sales Representative
4	4	Peacock	Margaret	Sales Representative
5	5	Buchanan	Steven	Sales Manager

➢ Extract 1 character from the text in the "FirstName" column (starting from left):

```
SELECT EmployeeID, LastName, LEFT(FirstName, 1)
 AS EmployeeFirstInitial
FROM Employees;
```



Results | Messages | Client Statistics

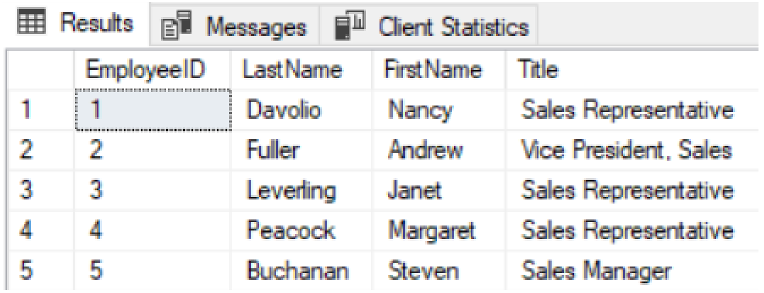
	EmployeeID	LastName	EmployeeFirstInitial
1	1	Davolio	N
2	2	Fuller	A
3	3	Leverling	J
4	4	Peacock	M
5	5	Buchanan	S

The RIGHT() Function

- ❑ The **RIGHT()** function: It extracts a number of characters from a string (starting from right)

➢ Original Employees table before the extraction

```sql
SELECT * FROM Employees;
```

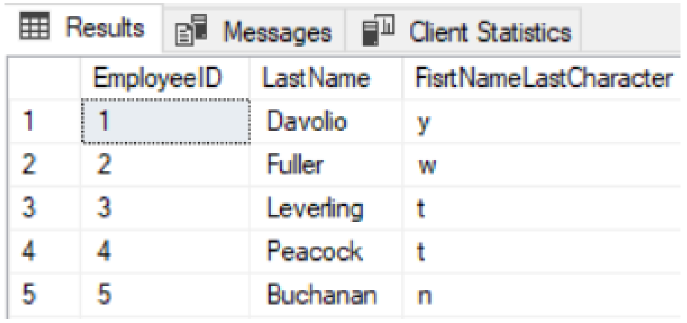


Results | Messages | Client Statistics

	EmployeeID	LastName	FirstName	Title
1	1	Davolio	Nancy	Sales Representative
2	2	Fuller	Andrew	Vice President, Sales
3	3	Leverling	Janet	Sales Representative
4	4	Peacock	Margaret	Sales Representative
5	5	Buchanan	Steven	Sales Manager

➢ Extract 1 character from the text in the “FirstName" column (starting from right):

```sql
SELECT EmployeeID, LastName, RIGHT(FirstName, 1)
 AS FisrtNameLastCharacter
FROM Employees;
```



Results | Messages | Client Statistics

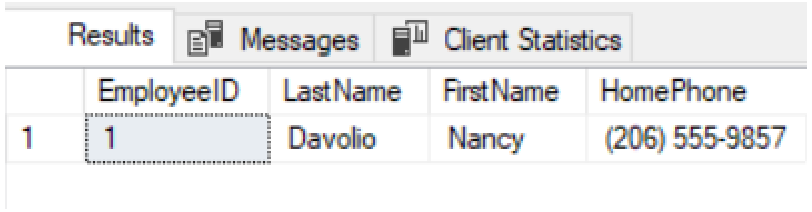
	EmployeeID	LastName	FisrtNameLastCharacter
1	1	Davolio	y
2	2	Fuller	w
3	3	Leverling	t
4	4	Peacock	t
5	5	Buchanan	n

The SUBSTRING() Function

- The **SUBSTRING()** function: It extracts a substring that starts at a specified position with a given length.
  - The position is the starting position where the substring begins. The 1st position of the string is zero (0)

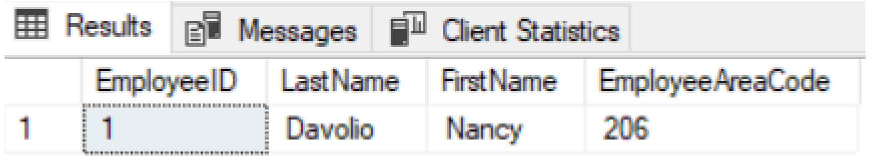
  - A substring starting at position 2 with the length 3

```
SELECT EmployeeID, LastName, FirstName, HomePhone
FROM Employees
WHERE EmployeeID = 1;
```



	EmployeeID	LastName	FirstName	HomePhone
1	1	Davolio	Nancy	(206) 555-9857

```
SELECT EmployeeID, LastName, FirstName,
SUBSTRING(HomePhone, 2, 3) AS EmployeeAreaCode
FROM Employees
WHERE EmployeeID = 1;
```

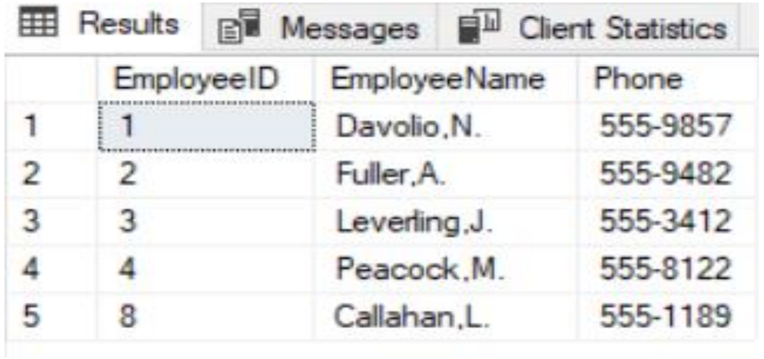


	EmployeeID	LastName	FirstName	EmployeeAreaCode
1	1	Davolio	Nancy	206

Three Functions (LEFT – RIGHT - SUBSTRING)

- A SELECT Statement Using Three Functions

```
Select EmployeeID, LastName + ','
       + LEFT(FirstName, 1)
       + '.' AS EmployeeName, RIGHT(HomePhone, 8)
 AS Phone
FROM Employees
WHERE SUBSTRING(HomePhone, 2, 3) = '206'
ORDER BY EmployeeID;
```



	EmployeeID	EmployeeName	Phone
1	1	Davolio,N.	555-9857
2	2	Fuller,A.	555-9482
3	3	Leverling,J.	555-3412
4	4	Peacock,M.	555-8122
5	8	Callahan,L.	555-1189

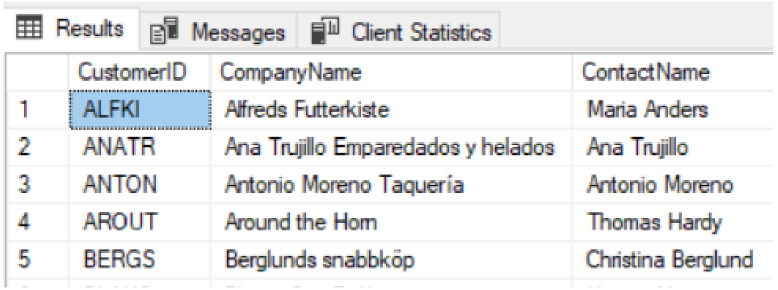
```
(5 rows affected)
```

STRINGS FUNCTIONS TO PARSE A STRING

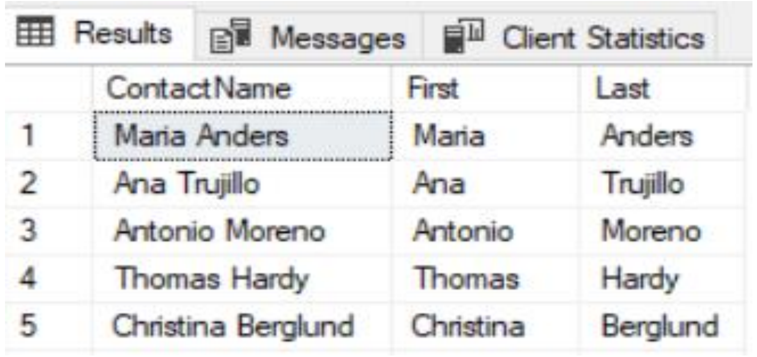
- LEFT() – RIGHT() – LEN() – CHARINDEX() Functions

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

```
SELECT * FROM Customers;
```



	CustomerID	CompanyName	ContactName
1	ALFKI	Alfreds Futterkiste	Maria Anders
2	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
3	ANTON	Antonio Moreno Taquería	Antonio Moreno
4	AROUT	Around the Horn	Thomas Hardy
5	BERGS	Berglunds snabbköp	Christina Berglund



	ContactName	First	Last
1	Maria Anders	Maria	Anders
2	Ana Trujillo	Ana	Trujillo
3	Antonio Moreno	Antonio	Moreno
4	Thomas Hardy	Thomas	Hardy
5	Christina Berglund	Christina	Berglund

NUMERIC FUNCTIONS

- Some of the numeric functions

- **ROUND(number,1ength[,function])** Returns the number rounded to the precision specified by length.
- **ISNUMERIC(expression)** Returns a value of 1 (true) if the expression is a numeric value; returns a value of 0 (false) otherwise.
- **ABS(number)** Returns the absolute value of the number.
- **CEILING(number)** Returns the smallest integer that is greater than or equal to the number.
- **FLOOR(number)** Returns the largest integer that is less than or equal to the number.
- **SQUARE (float_number)** Returns the square of a floating-point number.
- **SQRT (float_number)** Returns the square root of a floating-point number.
- **RAND([integer])** Returns a random floating-point number between 0 and 1.

NUMERIC FUNCTIONS Examples

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('2016-09-30')	0

NUMERIC FUNCTIONS Examples (cont.)

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

DATE/TIME Functions

- **GETDATE()** Returns a datetime value for the current local date and time based on the system's clock.
- **GETUTCDATE()** Returns a datetime value for the current UTC date and time based on the system's clock and time zone setting.
- **SYSDATETIME()** Returns a datetime2(7) value for the current local date and time based on the system's clock.
- **SYSUTCDATETIME()** Returns a datetime2(7) value for the current UTC date and time based on the system's clock and time zone setting.
- **SYSDATETIMEOFFSET()** Returns a datetimeoffset(7) value for the current UTC date and time based on the system's clock and time zone setting with a time zone offset

DATE/TIME Functions (cont.)

- **DAY(date)** Returns the day of the month as an integer.
- **MONTH(date)** Returns the month as an integer.
- **YEAR(date)** Returns the 4-digit year as an integer.
- **DATENAME(dat e p a r t , d a t e)** Returns the part of the date specified by datepart as a character string.
- **DATEPART(datepart,date)** Returns the part of the date specified by datepart as an integer.

DATE/TIME Functions (cont.)

- **DATEADD(datepart,number,date)** Returns the date that results from adding the specified number of datepart units to the date.
- **DATEDIFF(datepart,startdate,enddate)** Returns the number of datepart units between the specified start and end dates.
- **TODATETIMEOFFSET(datetime2,tzoffset)** Returns a datetimeoffset value that results from adding the specified time zone offset to the specified datetime2 value.
- **SWITCHOFFSET(datetimeoffset,tzoffset)** Returns a datetimeoffset value that results from switching the time zone offset for the specified datetimeoffset value to the specified offset.
- **EOMONTH(startdate[,months])** Returns a date for the last day of the month specified by the start date.
- **DATEFROMPARTS(year,month,day)** Returns a date for the specified year, month, and day.
- **ISDATE(expression)** Returns a value of 1 (true) if the expression is a valid date/time value; returns a value of 0 (false) otherwise.

DATE PART Values and Abbreviations

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

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

DATE/TIME Functions Examples

University of CINCINNATI

Function	Result
GETDATE()	2019-07-15 14:10:13.813
GETUTCDATE()	2019-07-15 21:10:13.813
SYSDATETIME()	2019-07-15 14:10:13.8160822
SYSUTCDATETIME()	2019-07-15 21:10:13.8160822
SYSDATETIMEOFFSET()	2019-07-15 14:10:13.8160822-07.00
MONTH(‘2019-07-15’)	7
DATEPART(month, ‘2019-07-15’)	7
DATENAME(month, ‘2019-07-15’)	July
DATENAME(m, ‘2019-07-15’)	July

DATE/TIME Functions Examples (cont.)

University of CINCINNATI

Function	Result
EOMONTH('2019-02-01')	2019-02-28
EOMONTH('2019-02-01',2)	2019-04-30
DATEFROMPARTS(2019,5,2)	2019-05-02
ISDATE('2019-07-15')	1
ISDATE('2019-07-16')	0
ISDATE('2019-07-16')	0
ISDATE('23:99:99')	0

DATE/TIME Functions Examples (cont.)

Function	Result
DAY('20169-07-15')	15
MONTH('2019-07-15')	7
YEAR('2019-07-15')	2019

DATEPART Function Examples

Function	Result
DATEPART(day, '2019-07-15 11:35:00')	15
DATEPART(month, '2019-07-15 11:35:00')	7
DATEPART(year, '2019-07-15 11:35:00')	2019
DATEPART(hour, '2019-07-15 11:35:00')	11
DATEPART(minute, '2019-07-15 11:35:00')	35
DATEPART(second, '2019-07-15 11:35:00')	0
DATEPART(quarter, '2019-09-30 11:35:00')	3
DATEPART(dayofyear, '2019-09-30 11:35:00')	272
DATEPART(week, '2019-09-30 11:35:00')	40
DATEPART(weekday, '2019-09-30 11:35:00')	1
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

DATENAME Function Examples

Function	Result
DATENAME(day, '2019-07-15 11:35:00')	15
DATENAME(month, '2019-07-15 11:35:00')	July
DATENAME(year, '2019-07-15 11:35:00')	2019
DATENAME(hour, '2019-07-15 11:35:00')	11
DATENAME(minute, '2019-07-15 11:35:00')	35
DATENAME(second, '2019-09-15 11:35:00')	0
DATENAME(quarter, '2016-09-30 11:35:00')	3
DATENAME(dayofyear, '2019-09-30 11:35:00')	273
DATENAME(week, '2019-09-30 11:35:00')	40
DATENAME(weekday, '2019-07-15 11:35:00')	Monday
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

DATEADD Function Examples

Function	Result
DATEADD(day, 1, '2019-07-15 11:35:00')	2019-07-16 11:35:00.000
DATEADD(month, 1, '2019-07-15 11:35:00')	2019-08-15 11:35:00.000
DATEADD(year, 1, '2019-07-15 11:35:00')	2020-07-15 11:35:00.000
DATEADD(hour, 1, '2019-07-15 11:35:00')	2019-07-15 12:35:00.000
DATEADD(minute, 1, '2019-07-15 11:35:00')	2019-07-15 11:36:00.000
DATEADD(second, 1, '2019-07-15 11:35:00')	2019-07-15 11:35:01.000
DATEADD(quarter, 1, '2019-07-15 11:35:00')	2019-10-15 11:35:00.000
DATEADD(week, 1, '2019-07-15 11:35:00')	2019-07-22 11:35:00.000
DATEADD(month, -1, '2019-07-15 11:35:00')	2019-06-15 11:35:00.000
DATEADD(year, 1.5, '2019-07-15 11:35:00')	2029-07-15 11:35:00.000

DATEDIFF Function Examples

Function	Result
DATEDIFF(day, '2019-07-15', '2020-05-15')	305
DATEDIFF(month, '2019-07-15', '2019-12-15')	5
DATEDIFF(year, '2019-07-15', '2020-09-15')	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, '2019-07-15', '2019-12-15')	1
DATEDIFF(week, '2019-07-15', '2020-07-15')	52
DATEDIFF(day, '2019-07-15', '2018-05-15')	-426

Addition and Subtraction Operators Examples

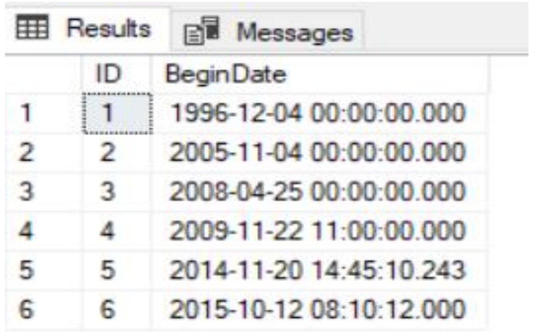
Operation	Result
CAST('2019-07-15 10:30:00' AS smalldatetime) + 1	2019-07-16 10:30:00
CAST('2019-07-15 10:30:00' AS smalldatetime) – 1	2010-07-14 10:30:00
CAST(CAST('2019-07-15' AS datetime) -CAST('2018-09-15' AS datetime) AS int)	303

DATE Values search

- ❑ Use **SOIT** (Database)

- ➢ Contents of DatetimeVsDatevalue Table

```
SELECT * FROM DatetimeVsDatevalue;
```



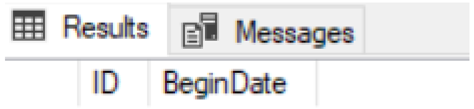
Results | Messages

	ID	BeginDate
1	1	1996-12-04 00:00:00.000
2	2	2005-11-04 00:00:00.000
3	3	2008-04-25 00:00:00.000
4	4	2009-11-22 11:00:00.000
5	5	2014-11-20 14:45:10.243
6	6	2015-10-12 08:10:12.000

```
(6 rows affected)
```

- ➢ No rows returned after this search condition

```
SELECT * FROM DatetimeVsDatevalue
WHERE BeginDate = '2014-11-20';
```



Results | Messages

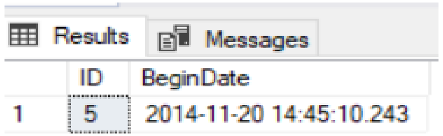
ID	BeginDate

```
(0 rows affected)
```

IGNORE **Time values** Techniques

- **Tech#1**. Use the date type to remove time values (SQL Server 2008 or later)

```
SELECT * FROM DatetimeVsDatevalue
WHERE CONVERT(date, BeginDate) = '2014-11-20';
```

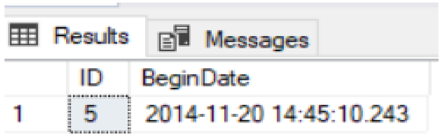


Results | Messages

```
   ID  BeginDate
1  5   2014-11-20 14:45:10.243
```

- **Tech#2.** Search for a range of dates

```
SELECT * FROM DatetimeVsDatevalue
WHERE BeginDate >= '2014-11-20'
  AND BeginDate < '2014-11-21';
```

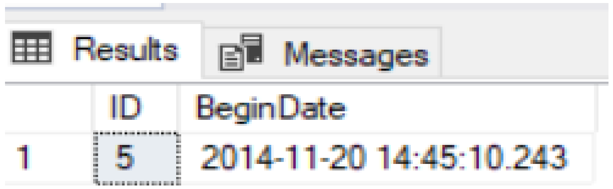


Results | Messages

```
   ID  BeginDate
1  5   2014-11-20 14:45:10.243
```

- **Tech#3.** Search for month, day, and year components

```
SELECT * FROM DatetimeVsDatevalue
WHERE MONTH(BeginDate) = 11 AND
      DAY(BeginDate) = 20 AND
      YEAR(BeginDate) = 2014;
```



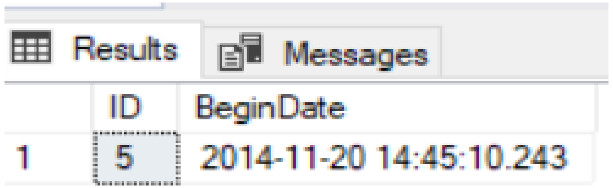
Results | Messages

```
   ID  BeginDate
1  5   2014-11-20 14:45:10.243
```

IGNORE **Time values** Techniques (cont.)

- **Tech#4.** Use the **CAST** function to remove time values

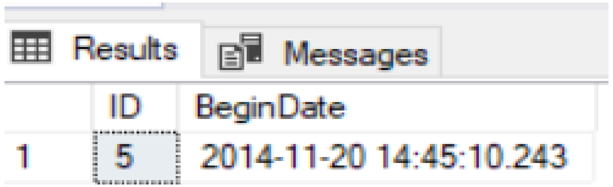
```
SELECT * FROM DatetimeVsDatevalue
WHERE CAST(CAST(BeginDate AS char(11)) AS
datetime)
    = '2014-11-20';
```



```
Results    Messages
   ID   BeginDate
1  5    2014-11-20 14:45:10.243
```

- **Tech#5.** Use the **CONVERT** function to remove time values

```
SELECT * FROM DatetimeVsDatevalue
WHERE CONVERT(datetime, CONVERT(char(10),
BeginDate, 110)) = '2014-11-20';
```

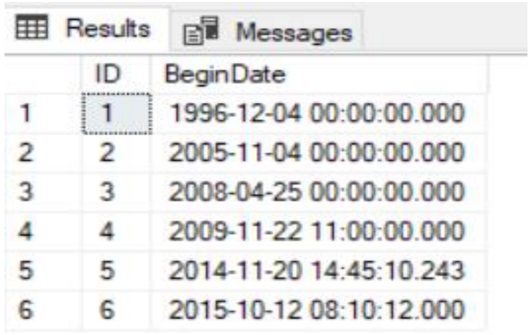


```
Results    Messages
   ID   BeginDate
1  5    2014-11-20 14:45:10.243
```

TIME Values search

- Contents of DatetimeVsDatevalue Table

```
SELECT * FROM DatetimeVsDatevalue;
```

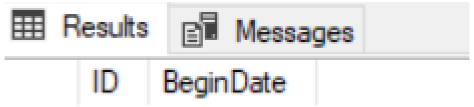


	ID	BeginDate
1	1	1996-12-04 00:00:00.000
2	2	2005-11-04 00:00:00.000
3	3	2008-04-25 00:00:00.000
4	4	2009-11-22 11:00:00.000
5	5	2014-11-20 14:45:10.243
6	6	2015-10-12 08:10:12.000

```
(6 rows affected)
```

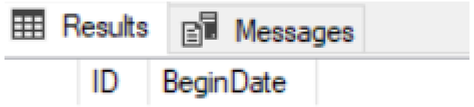
- No rows returned for both search conditions

```
SELECT * FROM DatetimeVsDatevalue
WHERE BeginDate = CAST('11:00:00' AS datetime);
```



ID	BeginDate

```
SELECT * FROM DatetimeVsDatevalue
WHERE BeginDate >= '18:00:00' AND
    BeginDate < '13:59:59:999';
```

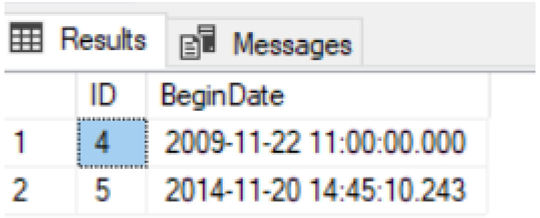


ID	BeginDate

IGNORE **Date values** Techniques

- **Tech#1.** Use the **time type** to remove date values (SQL Server 2008 or later)

```
SELECT * FROM DatetimeVsDatevalue
WHERE CONVERT(time, BeginDate) >= '10:00:00' AND
      CONVERT(time, BeginDate) < '14:59:59:999';
```

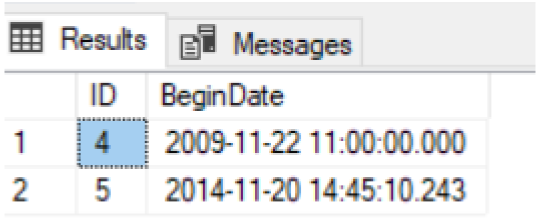


Results | Messages

	ID	BeginDate
1	4	2009-11-22 11:00:00.000
2	5	2014-11-20 14:45:10.243

- **Tech#2.** Use the **CONVERT** function to remove date values (prior to SQL Server 2008)

```
SELECT * FROM DatetimeVsDatevalue
WHERE CONVERT(datetime, CONVERT(char(12), BeginDate,
8)) >= '10:00:00' AND
      CONVERT(datetime, CONVERT(char(12), BeginDate,
8))< '14:59:59:999';
```



Results | Messages

	ID	BeginDate
1	4	2009-11-22 11:00:00.000
2	5	2014-11-20 14:45:10.243

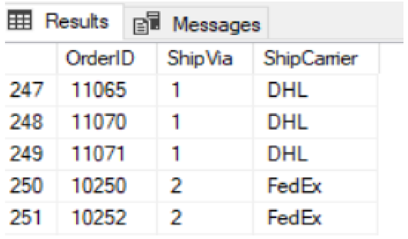
The CASE Function

- The **CASE** Function: It returns a value that's determined by the conditions you specify.

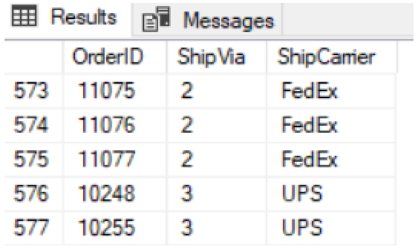
- Syntax of **CASE** Function

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN condition3 THEN result3
    ELSE result
END;
```

```
SELECT OrderID, ShipVia,
    CASE ShipVia
        WHEN 1 THEN 'DHL'
        WHEN 2 THEN 'FedEx'
        WHEN 3 THEN 'UPS'
        WHEN 4 THEN 'USPS'
    END AS ShipCarrier
FROM Orders;
```



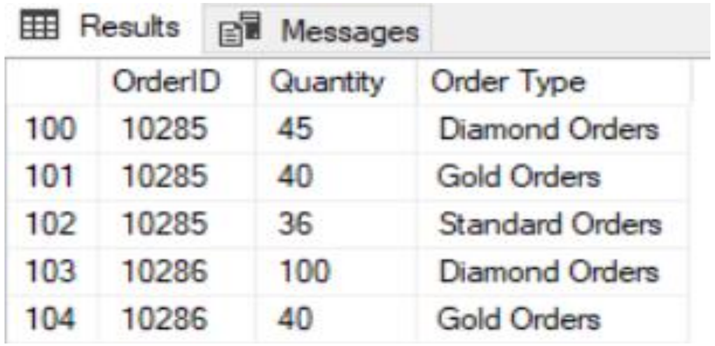
	OrderID	ShipVia	ShipCarrier
247	11065	1	DHL
248	11070	1	DHL
249	11071	1	DHL
250	10250	2	FedEx
251	10252	2	FedEx



	OrderID	ShipVia	ShipCarrier
573	11075	2	FedEx
574	11076	2	FedEx
575	11077	2	FedEx
576	10248	3	UPS
577	10255	3	UPS

The CASE Function (with ELSE clause)

```sql
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 40 THEN 'Diamond Orders'
    WHEN Quantity = 40 THEN 'Gold Orders'
    ELSE 'Standard Orders'
END AS [Order Type]
FROM [Order Details];
```



Results | Messages

	OrderID	Quantity	Order Type
100	10285	45	Diamond Orders
101	10285	40	Gold Orders
102	10285	36	Standard Orders
103	10286	100	Diamond Orders
104	10286	40	Gold Orders

The IIF Function

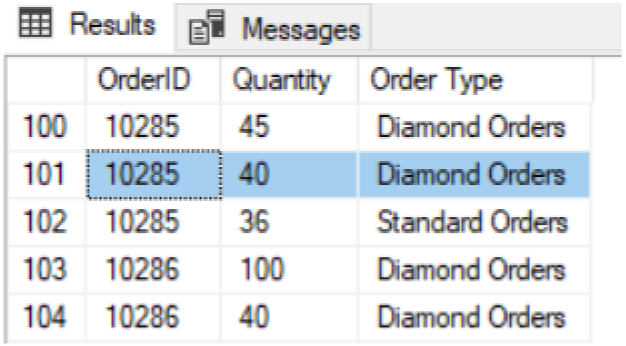
- The **IIF()** function: It returns a value if a condition is TRUE, or another value if a condition is FALSE.

  - Syntax of the **IIF** Function

```
IIF(condition, This value_if_true, This value_if_false)
```

- A SELECT statement with an **IIF** function

```
SELECT OrderID, Quantity,
IIF(Quantity>=40, 'Diamond Orders', 'Standard
Orders')
AS [Order Type]
FROM [Order Details];
```



Results | Messages

	OrderID	Quantity	Order Type
100	10285	45	Diamond Orders
101	10285	40	Diamond Orders
102	10285	36	Standard Orders
103	10286	100	Diamond Orders
104	10286	40	Diamond Orders

The CHOOSE Function

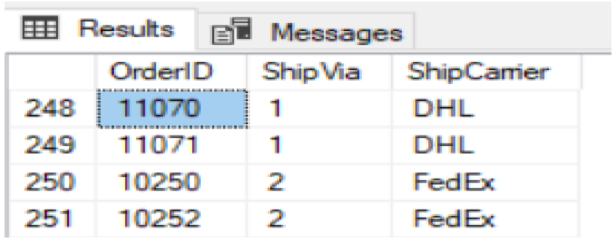
- The **CHOOSE** function: It returns the item from a list of items at a specified index.

  - Syntax of the **CHOOSE** Function

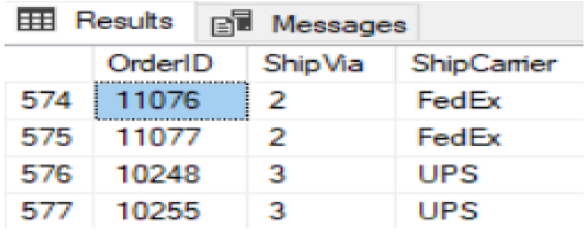
```
CHOOSE(index, value1, value2 ,value3...)
```

- A SELECT statement with a **CHOOSE** function

```
SELECT OrderID, ShipVia,
 CHOOSE(ShipVia, 'DHL', 'FedEx', 'UPS','USPS')
 AS ShipCarrier
FROM Orders;
```



	OrderID	ShipVia	ShipCarrier
248	11070	1	DHL
249	11071	1	DHL
250	10250	2	FedEx
251	10252	2	FedEx



	OrderID	ShipVia	ShipCarrier
574	11076	2	FedEx
575	11077	2	FedEx
576	10248	3	UPS
577	10255	3	UPS

The **COALESCE** Function

- The **COALESCE** function: It returns the first non-null expression in the list. If all expressions evaluate to null, then the COALESCE function will return null.

- Syntax of the **COALESCE** function

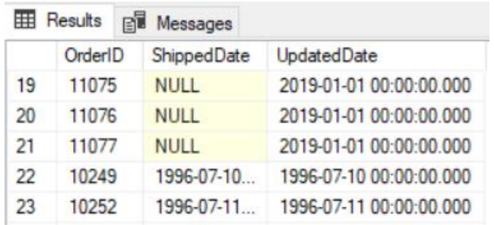
```
COALESCE( expression1, expression2, ... expression_n )
```

- Syntax of the **ISNULL** function

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

- A SELECT statement with a **COALESCE** function

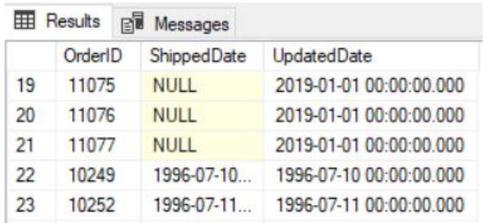
```
SELECT OrderID, ShippedDate,
 COALESCE(ShippedDate, '2019-01-01')
 AS UpdatedDate
FROM Orders;
```



	OrderID	ShippedDate	UpdatedDate
19	11075	NULL	2019-01-01 00:00:00.000
20	11076	NULL	2019-01-01 00:00:00.000
21	11077	NULL	2019-01-01 00:00:00.000
22	10249	1996-07-10...	1996-07-10 00:00:00.000
23	10252	1996-07-11...	1996-07-11 00:00:00.000

- The same statement with an ISNULL function

```
SELECT OrderID, ShippedDate,
 ISNULL(ShippedDate, '2019-01-01')
 AS UpdatedDate
FROM Orders;
```

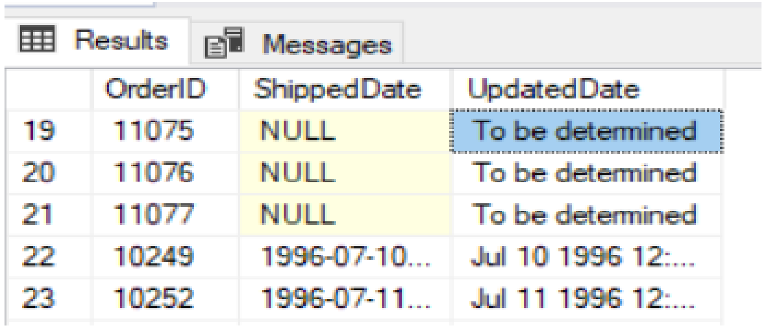


	OrderID	ShippedDate	UpdatedDate
19	11075	NULL	2019-01-01 00:00:00.000
20	11076	NULL	2019-01-01 00:00:00.000
21	11077	NULL	2019-01-01 00:00:00.000
22	10249	1996-07-10...	1996-07-10 00:00:00.000
23	10252	1996-07-11...	1996-07-11 00:00:00.000

The COALESCE Function: Substitute DATA Types

- A SELECT statement that substitutes a different data type.

```
SELECT OrderID, ShippedDate,
 COALESCE(CAST(ShippedDate AS VARCHAR), 'To be determined')
 AS UpdatedDate
FROM Orders;
```



Results | Messages

	OrderID	ShippedDate	UpdatedDate
19	11075	NULL	To be determined
20	11076	NULL	To be determined
21	11077	NULL	To be determined
22	10249	1996-07-10...	Jul 10 1996 12:...
23	10252	1996-07-11...	Jul 11 1996 12:...

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)
  ```

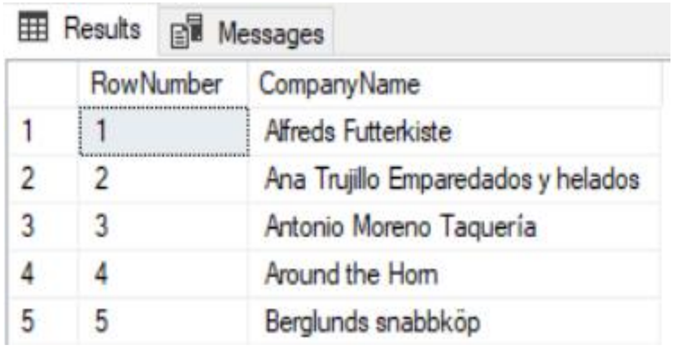
ROW_NUMBER Function

University of CINCINNATI

- ❑ **ROW_NUMBER** function: It returns the sequential number of a row starting at 1 for first row in each partition.
- ▪ ORDER BY clause is required
- ▪ PARTITION clause is optional

➢ Query with a **ROW_NUMBER** function

```
SELECT ROW_NUMBER() OVER(ORDER BY CompanyName) AS RowNumber,
  CompanyName
FROM Customers;
```



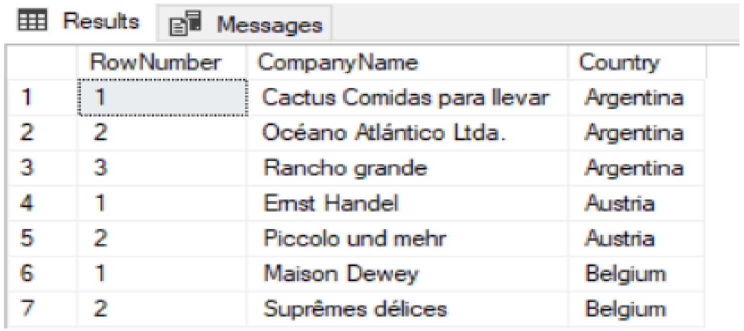
	RowNumber	CompanyName
1	1	Alfreds Futterkiste
2	2	Ana Trujillo Emparedados y helados
3	3	Antonio Moreno Taquería
4	4	Around the Horn
5	5	Berglunds snabbköp

PARTITION BY Clause

- **PARTITION BY:** It allows to specify a column that's used to divide the result set into groups

- Query with a **PARTITION BY** clause

```
SELECT ROW_NUMBER() OVER(PARTITION BY Country
ORDER BY CompanyName) As RowNumber,
CompanyName,Country
FROM Customers;
```



Results | Messages

	RowNumber	CompanyName	Country
1	1	Cactus Comidas para llevar	Argentina
2	2	Océano Atlántico Ltda.	Argentina
3	3	Rancho grande	Argentina
4	1	Ernst Handel	Austria
5	2	Piccolo und mehr	Austria
6	1	Maison Dewey	Belgium
7	2	Suprêmes délices	Belgium

RANK and DENSE_RANK

- **RANK() Function:** It assigns a rank to each row within a partition of a result set.
- **DENSE_RANK() Function:** It assigns a rank to each row within a partition of a result set.
  It also returns consecutive rank values.
  Rows in each partition receive the same ranks if they have the same values.

➢ Syntax of the **RANK** functions:

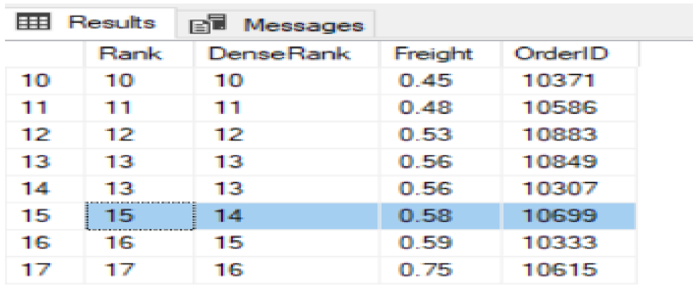
```
RANK()
OVER ([partition_by_clause] order_by_clause)
```

➢ Syntax of the **DENSE_RANK** functions:

```
DENSE_RANK()
OVER ([partition_by_clause] order_by_clause)
```

➢ A query with **RANK** and **DENSE_RANK** functions

```
SELECT RANK() OVER (ORDER BY Freight) As Rank,
   DENSE_RANK() OVER (ORDER By Freight)
   As DenseRank, Freight, OrderID
FROM Orders;
```



Results | Messages

	Rank	DenseRank	Freight	OrderID
10	10	10	0.45	10371
11	11	11	0.48	10586
12	12	12	0.53	10883
13	13	13	0.56	10849
14	13	13	0.56	10307
15	15	14	0.58	10699
16	16	15	0.59	10333
17	17	16	0.75	10615

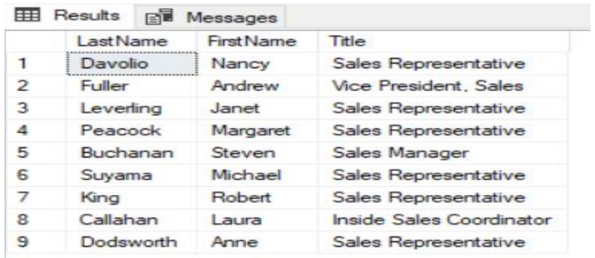
NTILE Function

❑ **NTILE() function:** It is used to divide the rows in a partition into the specified number of groups.

- ORDER BY clause is required
- PARTITION clause is optional

➢ A query that retrieves Lasname, Firstname and Title for all the employees.

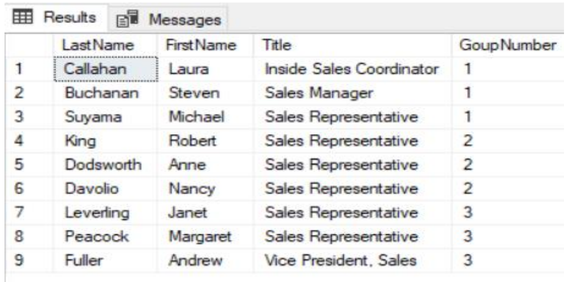
```
SELECT
LastName, FirstName, Title
FROM Employees;
```


Results | Messages

	LastName	FirstName	Title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
5	Buchanan	Steven	Sales Manager
6	Suyama	Michael	Sales Representative
7	King	Robert	Sales Representative
8	Callahan	Laura	Inside Sales Coordinator
9	Dodsworth	Anne	Sales Representative

➢ This **NTILE** function divides the Employees table in 3 groups

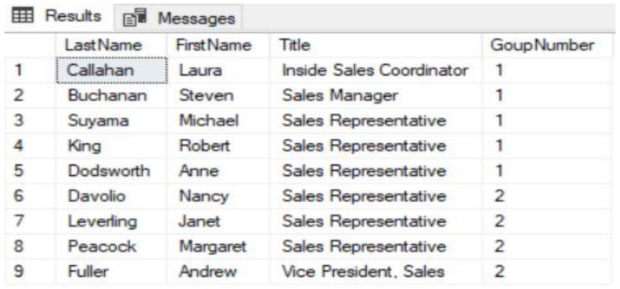
```
SELECT LastName, FirstName, Title,
NTILE(3) OVER (ORDER BY TITLE) AS
GoupNumber
FROM Employees;
```


Results | Messages

	LastName	FirstName	Title	GoupNumber
1	Callahan	Laura	Inside Sales Coordinator	1
2	Buchanan	Steven	Sales Manager	1
3	Suyama	Michael	Sales Representative	1
4	King	Robert	Sales Representative	2
5	Dodsworth	Anne	Sales Representative	2
6	Davolio	Nancy	Sales Representative	2
7	Leverling	Janet	Sales Representative	3
8	Peacock	Margaret	Sales Representative	3
9	Fuller	Andrew	Vice President, Sales	3

➢ This **NTILE** function divides the Employees table in 2 groups

```
SELECT LastName, FirstName, Title,
NTILE(2) OVER (ORDER BY TITLE) AS
GoupNumber
FROM Employees;
```


Results | Messages

	LastName	FirstName	Title	GoupNumber
1	Callahan	Laura	Inside Sales Coordinator	1
2	Buchanan	Steven	Sales Manager	1
3	Suyama	Michael	Sales Representative	1
4	King	Robert	Sales Representative	1
5	Dodsworth	Anne	Sales Representative	1
6	Davolio	Nancy	Sales Representative	2
7	Leverling	Janet	Sales Representative	2
8	Peacock	Margaret	Sales Representative	2
9	Fuller	Andrew	Vice President, Sales	2

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 **FitnessInc** Database

➢ SalesReps Table

Column Name	Data Type
RepID	int
RepFirstName	varchar(50)
RepLastName	varchar(50)

➢ SalesTotals Table

Column Name	Data Type
RepID	int
SalesYear	char(4)
SalesTotal	money

❑ Using the **FIRST_VALUE** and **LAST_VALUE** functions we can find the name of the sales rep with the highest and lowest sales for each year.

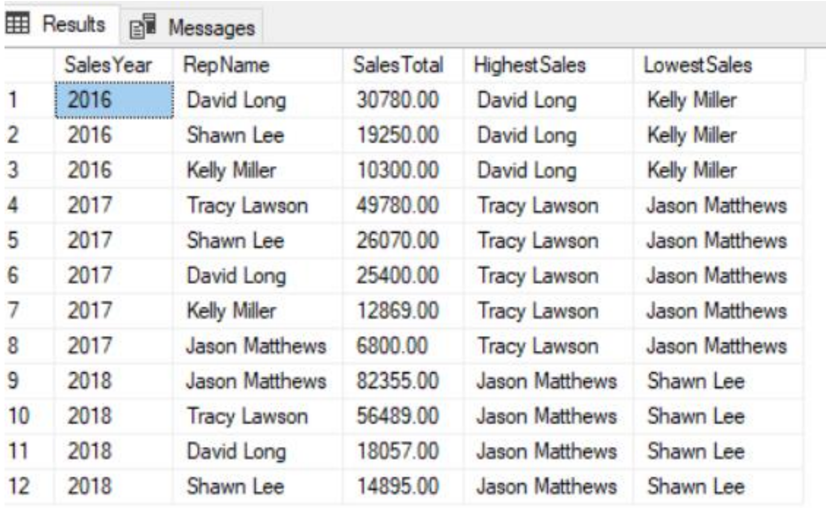
FIRST_VALUE and LAST_VALUE

- **FIRST_VALUE** function: It returns the first value in an ordered set of values
- **LAST_VALUE** function: It returns the last value in an ordered set of values

  - Use **FitnessInc** (Database)

➢ 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;
```



Results Messages

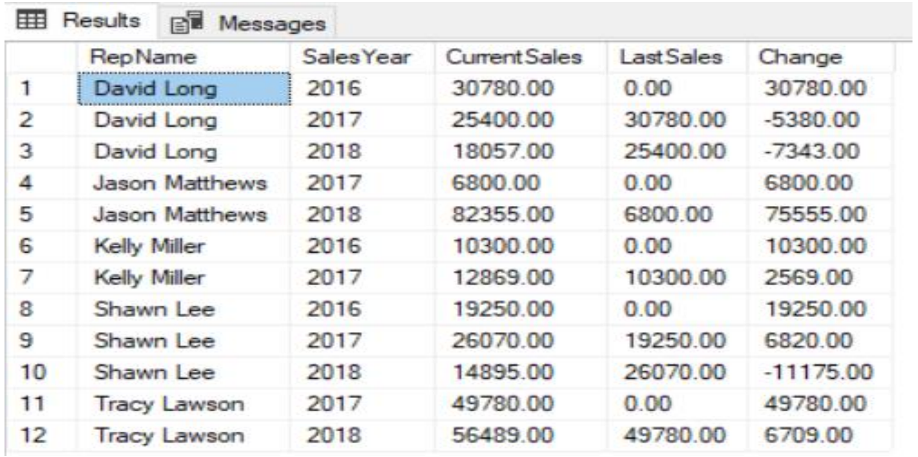
	SalesYear	RepName	SalesTotal	HighestSales	LowestSales
1	2016	David Long	30780.00	David Long	Kelly Miller
2	2016	Shawn Lee	19250.00	David Long	Kelly Miller
3	2016	Kelly Miller	10300.00	David Long	Kelly Miller
4	2017	Tracy Lawson	49780.00	Tracy Lawson	Jason Matthews
5	2017	Shawn Lee	26070.00	Tracy Lawson	Jason Matthews
6	2017	David Long	25400.00	Tracy Lawson	Jason Matthews
7	2017	Kelly Miller	12869.00	Tracy Lawson	Jason Matthews
8	2017	Jason Matthews	6800.00	Tracy Lawson	Jason Matthews
9	2018	Jason Matthews	82355.00	Jason Matthews	Shawn Lee
10	2018	Tracy Lawson	56489.00	Jason Matthews	Shawn Lee
11	2018	David Long	18057.00	Jason Matthews	Shawn Lee
12	2018	Shawn Lee	14895.00	Jason Matthews	Shawn Lee

The LAG Function

- The **LAG** function: It provides access to a row at a specified physical offset which comes before the current row.
  - Use **FitnessInc** (Database)

➢ A query that uses the LAG function

```
SELECT  RepFirstName + ' ' +  RepLastName AS
RepName, SalesYear,
SalesTotal AS CurrentSales,
 LAG(SalesTotal, 1, 0)OVER (PARTITION BY
RepFirstName ORDER BY SalesYear)
  AS LastSales,
  SalesTotal - LAG(SalesTotal, 1, 0)
  OVER (PARTITION BY RepFirstName ORDER BY
SalesYear)
 AS Change
FROM SalesTotals JOIN SalesReps
ON SalesTotals.RepID = SalesReps.RepID;
```


Results | Messages

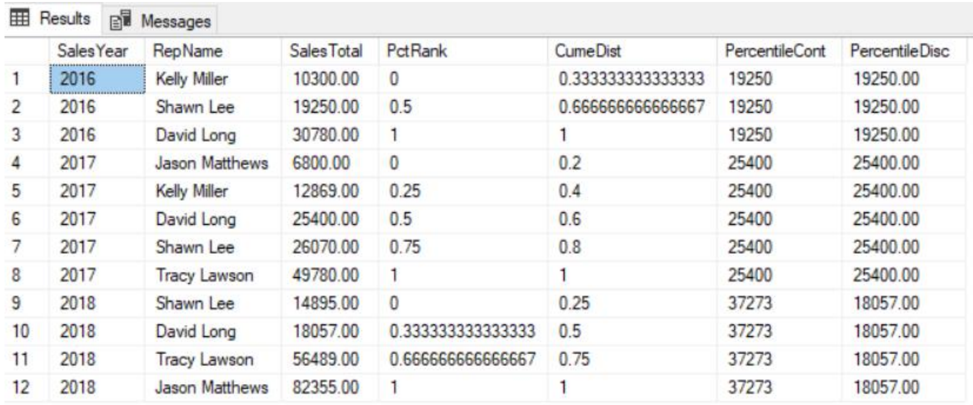
	RepName	SalesYear	CurrentSales	LastSales	Change
1	David Long	2016	30780.00	0.00	30780.00
2	David Long	2017	25400.00	30780.00	-5380.00
3	David Long	2018	18057.00	25400.00	-7343.00
4	Jason Matthews	2017	6800.00	0.00	6800.00
5	Jason Matthews	2018	82355.00	6800.00	75555.00
6	Kelly Miller	2016	10300.00	0.00	10300.00
7	Kelly Miller	2017	12869.00	10300.00	2569.00
8	Shawn Lee	2016	19250.00	0.00	19250.00
9	Shawn Lee	2017	26070.00	19250.00	6820.00
10	Shawn Lee	2018	14895.00	26070.00	-11175.00
11	Tracy Lawson	2017	49780.00	0.00	49780.00
12	Tracy Lawson	2018	56489.00	49780.00	6709.00

Four ANALYTIC Functions in a Query

- A query that uses four more functions

- Use **FitnessInc** (Database)

```
SELECT SalesYear, RepFirstName + ' ' +  RepLastName AS RepName, SalesTotal,
 PERCENT_RANK() OVER (PARTITION BY SalesYearORDER 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 JOIN SalesReps
ON SalesTotals.RepID = SalesReps.RepID;
```



	SalesYear	RepName	SalesTotal	PctRank	CumeDist	PercentileCont	PercentileDisc
1	2016	Kelly Miller	10300.00	0	0.333333333333333	19250	19250.00
2	2016	Shawn Lee	19250.00	0.5	0.666666666666667	19250	19250.00
3	2016	David Long	30780.00	1	1	19250	19250.00
4	2017	Jason Matthews	6800.00	0	0.2	25400	25400.00
5	2017	Kelly Miller	12869.00	0.25	0.4	25400	25400.00
6	2017	David Long	25400.00	0.5	0.6	25400	25400.00
7	2017	Shawn Lee	26070.00	0.75	0.8	25400	25400.00
8	2017	Tracy Lawson	49780.00	1	1	25400	25400.00
9	2018	Shawn Lee	14895.00	0	0.25	37273	18057.00
10	2018	David Long	18057.00	0.333333333333333	0.5	37273	18057.00
11	2018	Tracy Lawson	56489.00	0.666666666666667	0.75	37273	18057.00
12	2018	Jason Matthews	82355.00	1	1	37273	18057.00

Questions?