

Microsoft SQL from A to Z

Intro to SELECT Statements

An **instance** is an installation of a SQL server. Each has a unique name on the network.

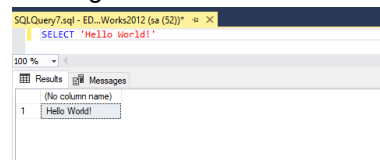
Database is an organized collection of data.

Schema is a collection of database objects associated with one particular database username.

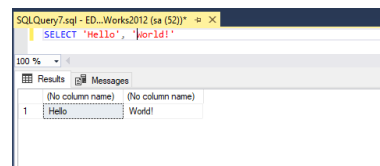
Literal SELECT statement

A statement that does not directly query a particular table and return columns, but instead returns the results of a string or mathematical expression.

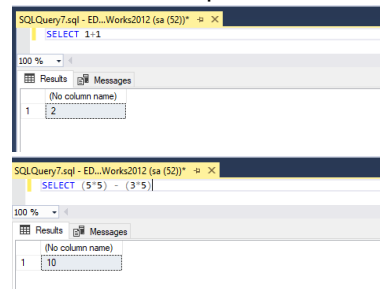
A string SELECT.



To use two columns.



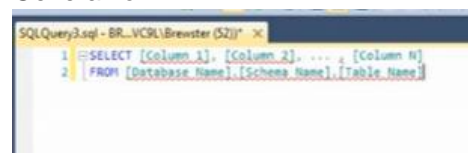
Mathematical expressions.



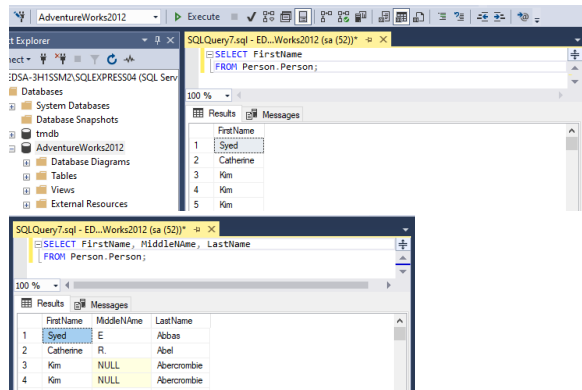
Basic SELECT statement

A statement that does directly query a particular table and return columns

General form:

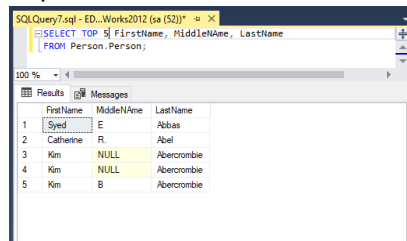


We do not specify the Database name in the form **AdventureWorks2012.Person.Person** because we are connected to the exact database.

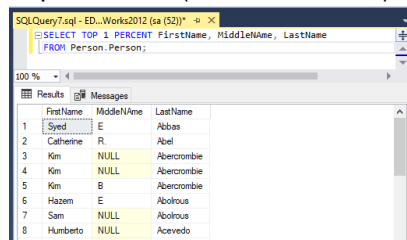


Top operator is used to limit the amount of rows displayed.

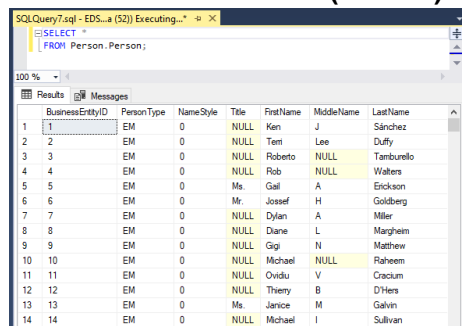
Top 5 rows.



Top 1% rows. (200 rows in total)



All rows and all columns (use of *)



Use * with the TOP operator (100 rows)

SQLQuery7.sql - ED...Works2012 (sa (52))

```
SELECT TOP 100 *
FROM Person.Person;
```

100 %

	BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName
1	1	EM	0	NULL	Ken	J	Sánchez
2	2	EM	0	NULL	Toni	Lee	Duffy
3	3	EM	0	NULL	Roberto	NULL	Tamburello
4	4	EM	0	NULL	Rob	NULL	Waters
5	5	EM	0	Ms.	Gail	A	Erickson
6	6	EM	0	Mr.	Josef	H	Goldberg
7	7	EM	0	NULL	Dylan	A	Miller
8	8	EM	0	NULL	Diane	L	Margheim
9	9	EM	0	NULL	Guy	N	Matthew
10	10	EM	0	NULL	Michael	NULL	Raheem
11	11	EM	0	NULL	Charles	V	Franklin

ALIAS is used to change the column name as viewed in the results panel, but do not change the column name in the table.

To create spaces in column names and use of ALIAS.

Use of square brackets []

SQLQuery7.sql - ED...Works2012 (sa (52))

```
SELECT TOP 10 FirstName AS [First Name], LastName AS [Last Name]
FROM Person.Person;
```

100 %

	First Name	Last Name
1	Syed	Abbas
2	Catherine	Abel
3	Kim	Abercrombie
4	Kim	Abercrombie
5	Kim	Abercrombie
6	Hazem	Abolrous
7	Sam	Abolrous
8	Humberto	Acevedo
9	Gustavo	Achong
10	Pilar	Ackerman

Use of double quotes ""

SQLQuery7.sql - ED...Works2012 (sa (52))

```
SELECT TOP 10 FirstName AS "First Name", LastName AS "Last Name"
FROM Person.Person;
```

100 %

	First Name	Last Name
1	Syed	Abbas
2	Catherine	Abel
3	Kim	Abercrombie
4	Kim	Abercrombie
5	Kim	Abercrombie
6	Hazem	Abolrous
7	Sam	Abolrous
8	Humberto	Acevedo
9	Gustavo	Achong
10	Pilar	Ackerman

A **view** is a virtual table based on the result set of an SQL statement.

Use of SQL functions, WHERE, and JOIN statements to view and present the data as if the data were coming from one single table.

A view.

SQLQuery7.sql - ED...Works2012 (sa (52))

```
SELECT *
FROM HumanResources.vEmployee
```

100 %

	BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	Job Title
1	5	Ms.	Gail	A	Erickson	NULL	Design Engineer
2	12	NULL	Thierry	B	D'Hers	NULL	Tool Designer
3	13	Ms.	Janice	M	Galvin	NULL	Tool Designer
4	14	NULL	Michael	I	Sullivan	NULL	Senior Design E
5	16	NULL	David	M	Bradley	NULL	Marketing Mana
6	17	NULL	Kevin	F	Brown	NULL	Marketing Asst.
7	18	NULL	John	L	Wood	NULL	Marketing Speci

A table.

SQLQuery7.sql - ED...Works2012 (sa (52))

```
SELECT *
FROM HumanResources.Employee
```

100 %

	BusinessEntityID	NationalIDNumber	LoginID	OrganizationNode	OrganizationLe
1	1	295947284	adventure-works\ken0	0x	0
2	2	245797967	adventure-works\tem0	0x58	1
3	3	509647174	adventure-works\rob0	0x5AC0	2
4	4	112457891	adventure-works\rob0	0x5AD6	3
5	5	695256308	adventure-works\gal0	0x5ADA	3
6	6	998320692	adventure-works\josef0	0x5ADE	3
7	7	134959118	adventure-works\dylan0	0x5AE1	3

Filtering with the WHERE clause

The **WHERE** clause is used to filter records (or rows of data).

General form:

```
SQLQuery2.sql - BR_VCS\Brewster (53) * X
1 SELECT [Column 1], [Column 2], ..., [Column N]
2 FROM [Schema Name].[Table Name]
3 WHERE [Column Name] [Comparison Operator] [some value]
```

SQL comparison operators.

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Using Equal to ==

```
SQLQuery2.sql - BR_VCS\Brewster (53) * X
1 SELECT *
2 FROM HumanResources.Employee
3 WHERE FirstName = 'Chris'
```

Results

BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	JobTitle	PhoneNumber	PhoneNumberType
1	221	NULL	Chris	K	Neved	NULL	Control Specialist	575-555-0126
2	37	NULL	Chris	O	Osakey	NULL	Production Technician - WC60	315-555-0144
3	165	NULL	Chris	T	Preston	NULL	Production Technician - WC45	205-555-0112

Using the Not equal to <>

```
SQLQuery2.sql - BR_VCS\Brewster (53) * X
1 SELECT *
2 FROM HumanResources.Employee
3 WHERE FirstName <> 'Chris'
```

Results

BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	JobTitle	PhoneNumber	PhoneNumberType
1	1	NULL	Alex	J	Sachdev	NULL	Chief Executive Officer	607-555-0142
2	2	NULL	Tam	Lee	Duffy	NULL	Vice President of Engineering	819-555-0175
3	3	NULL	Roberto	NULL	Tamburino	NULL	Engineering Manager	212-555-0187
4	4	NULL	Rita	NULL	Waters	NULL	Senior Tool Designer	612-555-0100
5	5	Ms.	Gail	A	Eckstein	NULL	Design Engineer	849-555-0139
6	6	Ms.	Josief	H	Goldberg	NULL	Design Engineer	122-555-0189
7	7	NULL	Dylan	A	Miler	NULL	Research and Development Manager	181-555-0156
8	8	NULL	Diane	L	Hardem	NULL	Research and Development Engineer	819-555-0138

Filter dates the same way you would represent strings in SQL.

```
SQLQuery3.sql - BR_VCS\Brewster (53) * X
1 SELECT *
2 FROM HumanResources.Employee
3 WHERE BirthDate >= '1/1/1988'
```

SQL logical operator

AND (conjunction)

It requires that the filtering criteria is true and returns a result to meet the condition.

SQL Query 1: Filtering by Birth Date and Gender

```
SELECT *
FROM HumanResources.Employee
WHERE BirthDate BETWEEN '1/1/1960' AND Gender = 'F'
```

EmpID	JobTitle	BirthDate	MaritalStatus	Gender	HireDate	SalaryFlag	VacationHours	SickLeaveHours	CurrentFlag
1	senior Engineer	1960-07-06	S	F	2003-01-30	1	42	51	1
2	senior Engineer	1963-06-29	M	F	2009-01-23	0	8	24	1
3	senior Engineer	1963-10-30	M	F	2003-03-02	0	14	27	1
4	senior Engineer	1960-06-20	M	F	2003-02-06	0	18	29	1
5	senior Engineer	1961-07-11	M	F	2003-03-20	0	15	27	1
6	senior Engineer	1962-12-21	M	F	2003-02-03	0	24	27	1
7	senior Engineer	1960-10-11	M	F	2004-02-02	0	89	64	1
8	senior Engineer	1962-08-07	M	F	2003-03-09	0	9	24	1

SQL Query 2: Filtering by Marital Status and Gender

```
SELECT *
FROM HumanResources.Employee
WHERE MaritalStatus = 'S' AND Gender = 'M'
```

EmpID	JobTitle	BirthDate	MaritalStatus	Gender	HireDate	SalaryFlag	VacationHours	SickLeaveHours	CurrentFlag
1	senior Engineer	1960-07-06	S	M	2003-01-30	1	42	51	1
2	senior Engineer	1963-06-29	S	M	2009-01-23	0	8	24	1
3	senior Engineer	1963-10-30	S	M	2003-03-02	0	14	27	1
4	senior Engineer	1960-06-20	S	M	2003-02-06	0	18	29	1
5	senior Engineer	1961-07-11	S	M	2003-03-20	0	15	27	1
6	senior Engineer	1962-12-21	S	M	2003-02-03	0	24	27	1
7	senior Engineer	1960-10-11	S	M	2004-02-02	0	89	64	1
8	senior Engineer	1962-08-07	S	M	2003-03-09	0	9	24	1

OR (disjunction)

It requires that one of the filtering criteria is true and returns a result to meet the condition.

SQL Query: Filtering by Birth Date and Gender OR Marital Status and Gender

```
SELECT *
FROM HumanResources.Employee
WHERE BirthDate BETWEEN '1/1/1960' AND Gender = 'F' OR MaritalStatus = 'S' AND Gender = 'M'
```

EmpID	JobTitle	BirthDate	MaritalStatus	Gender	HireDate	SalaryFlag	VacationHours	SickLeaveHours	CurrentFlag
1	senior Engineer	1960-07-06	S	M	2003-01-30	1	42	51	1
2	senior Engineer	1963-06-29	S	M	2009-01-23	0	8	24	1
3	senior Engineer	1963-10-30	S	M	2003-03-02	0	14	27	1
4	senior Engineer	1960-06-20	S	M	2003-02-06	0	18	29	1
5	senior Engineer	1961-07-11	S	M	2003-03-20	0	15	27	1
6	senior Engineer	1962-12-21	S	M	2003-02-03	0	24	27	1
7	senior Engineer	1960-10-11	S	M	2004-02-02	0	89	64	1
8	senior Engineer	1962-08-07	S	M	2003-03-09	0	9	24	1

Conjunction

The logical conjunction is used to check that two conditions are true.

The use of parentheses changes the meaning of the expression when using logical operators.

SQL Query: Filtering by Birth Date and Gender OR Marital Status and Gender AND Organization Level

```
SELECT *
FROM HumanResources.Employee
WHERE BirthDate BETWEEN '1/1/1960' AND Gender = 'F' OR MaritalStatus = 'S' AND Gender = 'M' AND OrganizationLevel = 4
```

EmpID	JobTitle	BirthDate	MaritalStatus	Gender	HireDate	SalaryFlag	VacationHours	SickLeaveHours	CurrentFlag
1	senior Engineer	1960-07-06	S	M	2003-01-30	1	42	51	1
2	senior Engineer	1963-06-29	S	M	2009-01-23	0	8	24	1
3	senior Engineer	1963-10-30	S	M	2003-03-02	0	14	27	1
4	senior Engineer	1960-06-20	S	M	2003-02-06	0	18	29	1
5	senior Engineer	1961-07-11	S	M	2003-03-20	0	15	27	1
6	senior Engineer	1962-12-21	S	M	2003-02-03	0	24	27	1
7	senior Engineer	1960-10-11	S	M	2004-02-02	0	89	64	1
8	senior Engineer	1962-08-07	S	M	2003-03-09	0	9	24	1

SQLQueryLog - BR_VCR/Brewster (SS) - x

```
1 SELECT *
2 FROM HumanResources.Employee
3 WHERE BirthDate < '1/1/2000' AND (Gender = 'M' OR OrganizationLevel < 4)
```

100 %

Results	Messages			
OrganizationLevel	Job Title	BirthDate	MaritalStatus	Gender
0	Chief Executive Officer	1963-03-02	S	M
3	Senior Tool Designer	1963-01-23	S	M
4	Research and Development Engineer	1960-07-06	S	F
3	Senior Tool Designer	1972-02-18	S	M
3	Senior Design Engineer	1973-07-17	S	M
1	Marketing Manager	1960-04-19	S	M
2	Marketing Assistant	1981-06-03	S	M
2	Marketing Specialist	1972-04-06	S	M

Query exec... BREWSTER (31.0 SP1) WIN-APSMHMYVCR/Brewst... AdventureWorks2012 00:00:00 127 rows

Use of logical and comparison operators

Disjunction

SQLQueryLog - BR_VCR/Brewster (SS) - x

```
1 SELECT *
2 FROM Production.Product
3 WHERE ([ListPrice > 100 AND Color = 'Red']) OR StandardCost > 30
```

100 %

Results	Messages					
ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStock
514	LL Mountain Seat Assembly	SA-M1158	1	0	NULL	500
515	ML Mountain Seat Assembly	SA-M237	1	0	NULL	500
516	HL Mountain Seat Assembly	SA-M887	1	0	NULL	500
517	LL Road Seat Assembly	SA-R127	1	0	NULL	500
518	ML Road Seat Assembly	SA-R128	1	0	NULL	500
519	HL Road Seat Assembly	SA-R522	1	0	NULL	500
520	LL Touring Seat Assembly	SA-T487	1	0	NULL	500
521	ML Touring Seat Assembly	SA-T512	1	0	NULL	500

Query exec... BREWSTER (31.0 SP1) WIN-APSMHMYVCR/Brewst... AdventureWorks2012 00:00:00 235 rows

SQLQueryLog - BR_VCR/Brewster (SS) - x

```
1 SELECT *
2 FROM HumanResources.vEmployeeDepartment
3 WHERE Department = "Research and Development" OR (StartDate < '1/1/2005'
4 AND Department = "Executive")
```

100 %

Results	Messages		
JobTitle	Department	GroupName	StartDate
1 Research and Development Manager	Research and Development	Research and Development	2003-01-12
2 Research and Development Engineer	Research and Development	Research and Development	2003-01-30
3 Research and Development Engineer	Research and Development	Research and Development	2003-02-17
4 Research and Development Manager	Research and Development	Research and Development	2003-06-04
5 Chief Executive Officer	Executive	Executive General and Administration	2003-02-15

Query exec... BREWSTER (11.0 SP1) WIN-APSMHMYVCR/Brewst... AdventureWorks2012 00:00:00 5 rows

SQLQueryLog - BR_VCR/Brewster (SS) - x

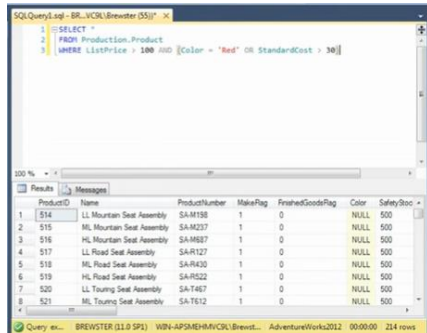
```
1 SELECT *
2 FROM Sales.vStorewithDemographics
3 WHERE (AnnualSales > 1000000 AND BusinessType = 'OS') OR
4 (
5     YearOpened < 1990 AND SquareFeet > 40000 AND
6     NumberEmployees > 10
7 )
```

100 %

Results	Messages				
BusinessEntityID	Name	AnnualSales	AnnualRevenue	BankName	Busnt...
314	Top of the Line Bikes	1500000.00	1500000.00	International Bank	OS
328	Purchase Mart	1500000.00	1500000.00	United Security	OS
330	Major Sport Suppliers	3000000.00	3000000.00	Reserve Security	OS
338	Systematic Sales	3000000.00	3000000.00	Primary International	OS
340	eCommerce Bikes	3000000.00	3000000.00	Guardian Bank	OS
346	Designated Distributors	3000000.00	3000000.00	International Security	OS
348	Sold Bike Accessories	3000000.00	3000000.00	Primary Bank & Reserve	OS
354	Acclaimed Bicycle Company	3000000.00	3000000.00	International Bank	OS

Query exec... BREWSTER (31.0 SP1) WIN-APSMHMYVCR/Brewst... AdventureWorks2012 00:00:01 239 rows

Conjunction



The screenshot shows a SQL query window with the following query:

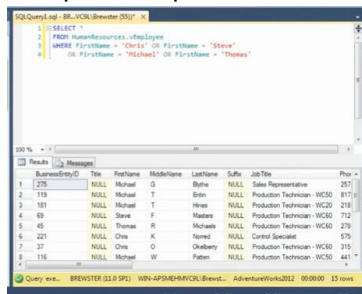
```
1 SELECT *
2 FROM Production.Product
3 WHERE ListPrice > 100 AND (Color = 'Red' OR StandardCost > 30)
```

The results pane shows the following data:

ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStock
514	LL Mountain Seat Assembly	SA-M198	1	0	NULL	500
515	ML Mountain Seat Assembly	SA-M237	1	0	NULL	500
516	HL Mountain Seat Assembly	SA-M287	1	0	NULL	500
517	LL Road Seat Assembly	SA-R127	1	0	NULL	500
518	ML Road Seat Assembly	SA-R430	1	0	NULL	500
519	HL Road Seat Assembly	SA-R522	1	0	NULL	500
520	LL Touring Seat Assembly	SA-T467	1	0	NULL	500
521	ML Touring Seat Assembly	SA-T612	1	0	NULL	500

Comparison operators, IN, BETWEEN

Comparison operator

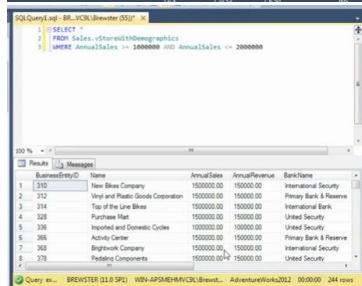


The screenshot shows a SQL query window with the following query:

```
1 SELECT *
2 FROM HumanResources.vEmployee
3 WHERE FirstName = 'Chris' OR FirstName = 'Steve'
4 OR FirstName = 'Michael' OR FirstName = 'Thomas'
```

The results pane shows the following data:

BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	JobTitle	Photo
275	NULL	Michael	G	Byrne	NULL	Sales Representative	257
119	NULL	Michael	T	Enn	NULL	Production Technician - WC30	817
181	NULL	Michael	T	Hines	NULL	Production Technician - WC30	218
69	NULL	Steve	F	Masters	NULL	Production Technician - WC30	712
45	NULL	Thomas	R	Muhale	NULL	Production Technician - WC30	278
221	NULL	Chris	K	Named	NULL	Control Specialist	575
27	NULL	Chris	O	Quiberry	NULL	Production Technician - WC30	315
116	NULL	Michael	W	Patten	NULL	Production Technician - WC30	441



The screenshot shows a SQL query window with the following query:

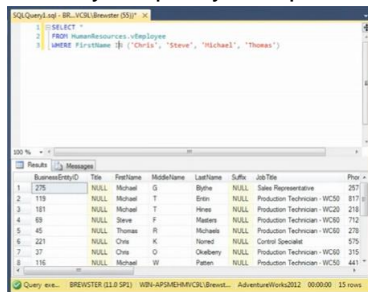
```
1 SELECT *
2 FROM Sales.vStoreWithDemographics
3 WHERE AnnualSales >= 2000000 AND AnnualRevenue <= 2000000
```

The results pane shows the following data:

BusinessEntityID	Name	AnnualSales	AnnualRevenue	BankName
210	New Blake Company	1500000.00	150000.00	International Security
312	Unyl and Plastic Goods Corporation	1500000.00	150000.00	Primary Bank & Reserve
314	Tap of the Line Bree	1500000.00	150000.00	International Bank
328	Purchase Real	1500000.00	150000.00	United Security
336	Imported and Domestic Cycles	1000000.00	100000.00	United Security
366	Active Center	1500000.00	150000.00	Primary Bank & Reserve
368	Bigbrook Company	1500000.00	150000.00	International Security
378	Pedding Components	1500000.00	150000.00	United Security

IN

Allows you specify multiple values in a WHERE clause in the form of a list.



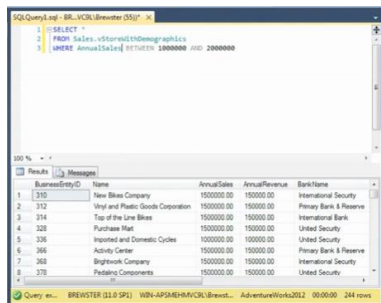
The screenshot shows a SQL query window with the following query:

```
1 SELECT *
2 FROM HumanResources.vEmployee
3 WHERE FirstName IN ('Chris', 'Steve', 'Michael', 'Thomas')
```

The results pane shows the following data:

BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	JobTitle	Photo
275	NULL	Michael	G	Byrne	NULL	Sales Representative	257
119	NULL	Michael	T	Enn	NULL	Production Technician - WC30	817
181	NULL	Michael	T	Hines	NULL	Production Technician - WC30	218
69	NULL	Steve	F	Masters	NULL	Production Technician - WC30	712
45	NULL	Thomas	R	Muhale	NULL	Production Technician - WC30	278
221	NULL	Chris	K	Named	NULL	Control Specialist	575
27	NULL	Chris	O	Quiberry	NULL	Production Technician - WC30	315
116	NULL	Michael	W	Patten	NULL	Production Technician - WC30	441

BETWEEN



Wildcard operator and characters

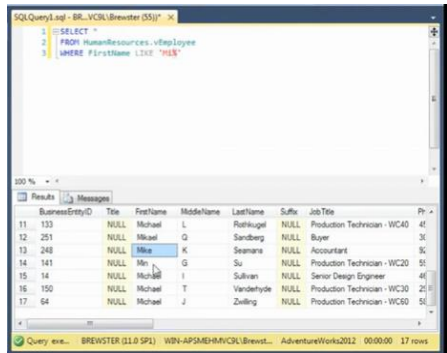
Wildcard Characters in SQL Server

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents a range of characters	c[a-b]t finds cat and cbt

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_ %'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

% represents 0 to multiple characters



_ (underscore) is used for a single character

SQL Query Log - BR_VCRU/Brewster (SSU)*

```

1 SELECT *
2 FROM HumanResources.vEmployee
3 WHERE Firstname LIKE 'HL'

```

100 %

BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	JobTitle	PhoneNumber
141		HL		Su	NULL	Production Technician - WC20	590-550

Query exec... BREWSTER (31.0 SP1) WIN-APSMHMYVCRU/Brewst... AdventureWorks2012 00:00:00 1 rows

[] to find specific characters or a range

SQL Query Log - BR_VCRU/Brewster (SSU)*

```

1 SELECT *
2 FROM HumanResources.vEmployee
3 WHERE Firstname LIKE '[a-z]'

```

100 %

BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	JobTitle	PhoneNumber
118		Don	L	Hall	NULL	Production Technician - WC50	100-550
269		Don	K	Bacon	Jr	Application Specialist	166-555-0159
271		Don	B	Wilson	NULL	Database Administrator	653-555-0144

Query exec... BREWSTER (31.0 SP1) WIN-APSMHMYVCRU/Brewst... AdventureWorks2012 00:00:00 3 rows

SQL Query Log - BR_VCRU/Brewster (SSU)*

```

1 SELECT *
2 FROM HumanResources.vEmployee
3 WHERE Firstname LIKE '[a-z]'

```

100 %

BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	JobTitle	PhoneNumber
269		Don	K	Bacon	Jr	Application Specialist	166-555-0159
271		Don	B	Wilson	NULL	Database Administrator	653-555-0144

Query exec... BREWSTER (31.0 SP1) WIN-APSMHMYVCRU/Brewst... AdventureWorks2012 00:00:00 2 rows

Not a specific character (exclude 'o')

SQL Query Log - BR_VCRU/Brewster (SSU)*

```

1 SELECT *
2 FROM HumanResources.vEmployee
3 WHERE Firstname LIKE '[a-z]'

```

100 %

BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	JobTitle	PhoneNumber
269		Don	K	Bacon	Jr	Application Specialist	166-555-0159
271		Don	B	Wilson	NULL	Database Administrator	653-555-0144

Query exec... BREWSTER (31.0 SP1) WIN-APSMHMYVCRU/Brewst... AdventureWorks2012 00:00:00 2 rows

NULL - means nothing, not a blank space.

Filter on NULL.

This sees NULL as a value. Not how to use NULL.

SQL Query - BR...VCN:Brewster (35/)

```
1 || SELECT *
2 | FROM Person.Person
3 | WHERE HiredDate = 'Null'
```

100 %

Results Messages

BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	EmailProvider
------------------	------------	-----------	-------	-----------	------------	----------	--------	---------------

Query exec... BREWSTER (11.0 SP1) WIN-AP9M6H4VCN:Brewst... AdventureWorks2012 00:00:00 0 rows

SQL Query - BR...VCN:Brewster (35/)

```
1 || SELECT *
2 | FROM Person.Person
3 | WHERE HiredDate = 'Null'
```

100 %

Results Messages

BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	EmailProvider
------------------	------------	-----------	-------	-----------	------------	----------	--------	---------------

Query exec... BREWSTER (11.0 SP1) WIN-AP9M6H4VCN:Brewst... AdventureWorks2012 00:00:00 0 rows

How to use NULL

SQL Query - BR...VCN:Brewster (35/)

```
1 || SELECT *
2 | FROM Person.Person
3 | WHERE HiredDate IS NULL
```

100 %

Results Messages

BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	Email
1	EM	0	NULL	Roberts	NULL	Tamburlo	NULL	0
2	EM	0	NULL	Rao	NULL	Walens	NULL	0
3	EM	0	NULL	Michael	NULL	Flahern	NULL	2
4	EM	0	NULL	Byrn	NULL	Bauer	NULL	0
5	EM	0	NULL	Russel	NULL	Hunter	NULL	1
6	EM	0	NULL	Ruan	NULL	Yu	NULL	0
7	EM	0	NULL	Em	NULL	Quake	NULL	2
8	EM	0	NULL	Wahne	NULL	Sunkumun	NULL	2

Query exec... BREWSTER (11.0 SP1) WIN-AP9M6H4VCN:Brewst... AdventureWorks2012 00:00:01 889 rows

SQL Query - BR...VCN:Brewster (35/)

```
1 || SELECT *
2 | FROM Person.Person
3 | WHERE HiredDate IS NOT NULL
```

100 %

Results Messages

BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	EmailProvider
1	EM	0	NULL	Ken	J	Sanchez	NULL	0
2	EM	0	NULL	Teri	Lee	Duffy	NULL	1
3	EM	0	Ms	Gal	A	Enckson	NULL	0
4	EM	0	Mr	Josef	H	Goldberg	NULL	0
5	EM	0	NULL	Dylan	A	Wier	NULL	2
6	EM	0	NULL	Diane	L	Marquess	NULL	0
7	EM	0	NULL	Op	H	Mathews	NULL	0
8	EM	0	NULL	Orville	V	Orson	NULL	0

Query exec... BREWSTER (11.0 SP1) WIN-AP9M6H4VCN:Brewst... AdventureWorks2012 00:00:01 13473 rows

Sorting data with the ORDER BY clause

The ORDER BY clause is used to sort results in ascending or descending order.

General form

```
SQLQuery1.ad - BR-VCLBrower (SQL)
1 SELECT Column1 AS [Some Alias], Column2, ..., ColumnN
2 FROM SchemaName.TableNames
3 WHERE ColumnName [Comparison Operator] [Filtering Criteria]
4 ORDER BY ColumnName, ColumnOrdinal, ColumnAlias [ASC/DESC]
```

In ascending order (automatic)

```
SQLQuery1.ad - BR-VCLBrower (SQL)
1 SELECT FirstName, LastName
2 FROM Sales.Customer
3 ORDER BY [CustomerID]
```

FirstName	LastName
Aaron	Adams
Aaron	Alvarado
Aaron	Allen
Aaron	Baker
Aaron	Bryant
Aaron	Butler
Aaron	Cardwell
Aaron	Cole
Aaron	Chen

Order columns by the second column (ordinal position). **NO OTHER CLAUSE DOES THIS.**

```
SQLQuery1.ad - BR-VCLBrower (SQL)
1 SELECT FirstName, LastName AS [Customer Last Name]
2 FROM Sales.Customer
3 ORDER BY 2
```

FirstName	Customer Last Name
Aaron	Adams
Aaron	Adams
Alan	Adams
Alexandra	Adams
Allen	Adams
Amadeo	Adams
Andrew	Adams
Archie	Adams
Arturo	Adams

Order columns by the alias

```
SQLQuery1.ad - BR-VCLBrower (SQL)
1 SELECT FirstName, LastName AS [Customer Last Name]
2 FROM Sales.Customer
3 ORDER BY [Customer Last Name] DESC
```

FirstName	Customer Last Name
Alan	Adams
Banca	Donnenman
Carolina	Donnenman
Cheryl	Donnenman
Curtis	Donnenman
Henry	Donnenman
Jack	Donnenman
Jenny	Donnenman
Kristal	Donnenman

How SQL evaluates the clauses.

- First 6 is how we write the clauses.
- Second 6 is how SQL evaluates the clauses.

```

SQLQuery1.sql - BR_VCR\Brewster (SQL)
1  SELECT
2  FROM
3  WHERE
4  GROUP BY
5  HAVING
6  ORDER BY
7
8  FROM
9  WHERE
10 GROUP BY
11 HAVING
12 SELECT
13 ORDER BY

```

Sort the LastName in ascending order and FirstName is descending order.

```

SQLQuery1.sql - BR_VCR\Brewster (SQL)
1  SELECT FirstName, LastName
2  FROM Sales.Customer
3  ORDER BY LastName, FirstName DESC

```

FirstName	LastName
Xavier	Adams
Wyatt	Adams
Thomas	Adams
Sophy	Adams
Shoshane	Adams
Seth	Adams
Sean	Adams
Srinivas	Adams
Sara	Adams

Sort the LastName and FirstName in ascending order.

```

SQLQuery1.sql - BR_VCR\Brewster (SQL)
1  SELECT FirstName AS [Customer First Name], LastName
2  FROM Sales.Customer
3  ORDER BY 2, [Customer First Name]

```

Customer First Name	LastName
Adam	Adams
Adam	Adams
Alex	Adams
Alexandra	Adams
Alison	Adams
Amanda	Adams
Andie	Adams
Andrea	Adams
Arnel	Adams

Sort SalesQuota DESC and LastName ASC where SalesQuota >= 250000

```

SQLQuery1.sql - BR_VCR\Brewster (SQL)
1  SELECT LastName, FirstName, SalesQuota
2  FROM Sales.SalesPerson
3  WHERE SalesQuota >= 250000
4  ORDER BY SalesQuota DESC, LastName ASC

```

LastName	FirstName	SalesQuota
Bayle	Michael	300000.00
Norris-Annan	Tate	300000.00
Peterson	Toni	300000.00
Arman-White	Farida	250000.00
Campbell	David	250000.00
Carson	Julian	250000.00
Lee	Shirley	250000.00
Michiel	Linda	250000.00
Pike	Jim	250000.00

Querying multiple tables with JOINS

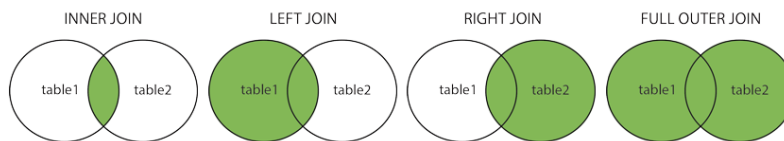
A JOIN is a means for combining columns from one (self-join) or more tables by using values common to each.

Foreign keys indicate how we can JOIN tables.

Different Types of SQL JOINS

Here are the different types of the JOINS in SQL:

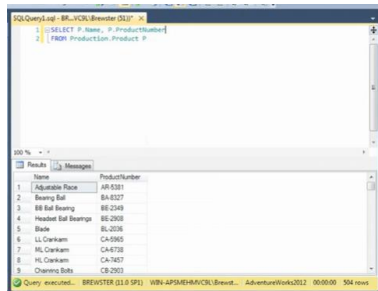
- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



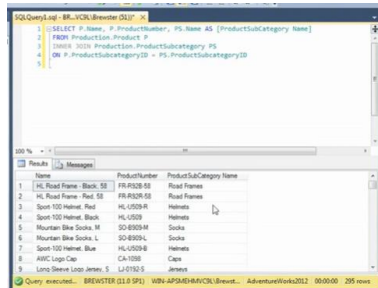
INNER JOIN

An intersection between two tables.

Creating an alias for the table. (Production.Product is now P)



Use INNER join to connect the rows where there is a matching column.



SQLQuery1.sql - MS_VCLBrewster (SQL) -

```

1 SELECT
2   PS.Name AS ProductSubCategoryName,
3   PC.Name AS ProductCategoryName
4 FROM Production.Product PS
5 INNER JOIN Production.ProductCategory PC
6 ON PS.ProductCategoryID = PC.ProductCategoryID

```

100 %

Results Messages

ProductSubCategoryName	ProductCategoryName
Mountain Bikes	Bikes
Road Bikes	Bikes
Touring Bikes	Bikes
Handbikes	Components
Bottom Brackets	Components
Brakes	Components
Chains	Components
Cranksets	Components
Derailleurs	Components

Query executed... BREWSTER (11.9 SP1) WIN-APSMHVM/CL/Brewst... AdventureWorks2012 00:00:00 37 rows

Use one INNER JOIN

SQLQuery1.sql - MS_VCLBrewster (SQL) -

```

1 SELECT P.FirstName, P.LastName, E.EmailAddress
2 FROM Person.Person P
3 INNER JOIN HumanResources.Employee E
4 ON P.BusinessEntityID = E.BusinessEntityID

```

100 %

Results Messages

FirstName	LastName	EmailAddress
Adrian	Abbas	adrian@adventureworks.com
Carlene	Abel	carlene@adventureworks.com
Ken	Abeniorde	ken@adventureworks.com
Ken	Abeniorde	ken@adventureworks.com
Ken	Abeniorde	ken@adventureworks.com
Hazen	Abolous	hazen@adventureworks.com
Sam	Abolous	sam@adventureworks.com
Humberto	Acosta	humberto@adventureworks.com
Gustavo	Achong	gustavo@adventureworks.com

Query executed... BREWSTER (11.9 SP1) WIN-APSMHVM/CL/Brewst... AdventureWorks2012 00:00:00 10072 rows

Use two INNER JOINS

SQLQuery1.sql - MS_VCLBrewster (SQL) -

```

1 SELECT P.FirstName, P.LastName, E.EmailAddress, PP.PhoneNumber
2 FROM Person.Person P
3 INNER JOIN HumanResources.Employee E
4 ON P.BusinessEntityID = E.BusinessEntityID
5 INNER JOIN HumanResources.Employee PP
6 ON PP.BusinessEntityID = P.BusinessEntityID

```

100 %

Results Messages

FirstName	LastName	EmailAddress	PhoneNumber
Adrian	Abbas	adrian@adventureworks.com	526-555-0102
Carlene	Abel	carlene@adventureworks.com	747-555-0171
Ken	Abeniorde	ken@adventureworks.com	334-555-0137
Ken	Abeniorde	ken@adventureworks.com	319-555-0100
Hazen	Abolous	hazen@adventureworks.com	208-555-0114
Sam	Abolous	sam@adventureworks.com	360-555-0123
Humberto	Acosta	humberto@adventureworks.com	339-555-0127
Gustavo	Achong	gustavo@adventureworks.com	339-555-0132

Query executed... BREWSTER (11.9 SP1) WIN-APSMHVM/CL/Brewst... AdventureWorks2012 00:00:00 10072 rows

LEFT OUTER JOIN

All information for table A and an intersection between two tables .

LEFT OUTER JOIN

There is NULL under the ProductSubCategoryName as there is no match between the two tables for the ProductSubCategoryID column.

SQLQuery1.sql - MS_VCLBrewster (SQL) -

```

1 SELECT P.Name, P.ProductNumber, PS.Name AS ProductSubCategoryName
2 FROM Production.Product P
3 LEFT OUTER JOIN Production.ProductSubCategory PS
4 ON P.ProductSubCategoryID = PS.ProductSubCategoryID

```

100 %

Results Messages

Name	ProductNumber	ProductSubCategoryName
Adjustable Race	AR-5381	NULL
Bearing Ball	BA-5327	NULL
Ball Bearing	BB-2389	NULL
Headset Ball Bearings	BB-2908	NULL
Brake	BL-2036	NULL
LL-Crankarm	CA-5985	NULL
ML-Crankarm	CA-6738	NULL
HL-Crankarm	CA-7487	NULL
Chainset Bolt	CB-2903	NULL

Query executed... BREWSTER (11.9 SP1) WIN-APSMHVM/CL/Brewst... AdventureWorks2012 00:00:00 504 rows

RIGHT OUTER JOIN

All information for table B and an intersection between two tables .

RIGHT OUTER JOIN (follows above example from LEFT OUTER JOIN)

```
SELECT P.Name, P.ProductNumber, PS.Name AS ProductSubcategoryName
FROM Production.Product P
LEFT OUTER JOIN Production.ProductSubcategory PS
ON P.ProductSubcategoryID = PS.ProductSubcategoryID
UNION ALL
SELECT P.Name, P.ProductNumber, PS.Name AS ProductSubcategoryName
FROM Production.Product P
RIGHT OUTER JOIN Production.ProductSubcategory PS
ON P.ProductSubcategoryID = PS.ProductSubcategoryID
```

Name	ProductNumber	ProductSubcategoryName
Bearing Ball	BA-8327	NULL
BB Ball Bearing	BB-2349	NULL
Headset Ball Bearings	BE-2908	NULL
Bike	BL-2036	NULL
LL Crankarm	CA-8845	NULL
ML Crankarm	CA-8738	NULL
HL Crankarm	CA-7487	NULL
Adjustable Race	AR-8337	NULL

INNER JOIN (follows above example from LEFT OUTER JOIN)

```
SELECT P.Name, P.ProductNumber, PS.Name AS ProductSubcategoryName
FROM Production.Product P
INNER JOIN Production.ProductSubcategory PS
ON P.ProductSubcategoryID = PS.ProductSubcategoryID
```

Name	ProductNumber	ProductSubcategoryName
HL Road Frame - Black, 52	FR-RD28-S	Road Frames
HL Road Frame - Red, 53	FR-RD29-S	Road Frames
Sport 100 Helmet, Red	HL-US24-M	Helmets
Sport 100 Helmet, Black	HL-US25-M	Helmets
Mountain Bike Socks, M	SO-BK24-M	Socks
Mountain Bike Socks, L	SO-BK24-L	Socks
Sport 100 Helmet, Blue	HL-US26-M	Helmets
AWC Logo Cap	CA-1038	Caps

Example: LEFT OUTER JOIN

```
SELECT P.FirstName, P.LastName,
SOH.SalesOrderNumber,
SOH.TotalDue AS SalesAmount
FROM Sales.SalesOrderHeader SOH
LEFT OUTER JOIN Sales.SalesPerson SP
ON SP.BusinessEntityID = SOH.SalesPersonID
LEFT OUTER JOIN HumanResources.Employee E
ON E.BusinessEntityID = SP.BusinessEntityID
LEFT OUTER JOIN Person.Person P
ON P.BusinessEntityID = E.BusinessEntityID
```

FirstName	LastName	SalesOrderNumber	SalesAmount
Tavi	Rietter	SO43659	23153.2339
Tavi	Rietter	SO43660	1457.3288
José	Saraiva	SO43661	36885.8012
José	Saraiva	SO43662	32474.9324
Linda	Mitchell	SO43663	472.3108
Pamela	Anaman-Wolfe	SO43664	27510.4109
David	Campbell	SO43665	16150.6961
Linda	Mitchell	SO43666	5694.8564
Jillian	Carson	SO43667	6876.3649
José	Saraiva	SO43668	40487.7233
David	Campbell	SO43669	807.2585
Michael	Blythe	SO43670	6893.2549
David	Campbell	SO43671	9153.6054
José	Saraiva	SO43672	6895.41

```
SELECT P.FirstName, P.LastName,
SOH.SalesOrderNumber,
SOH.TotalDue AS SalesAmount,
T.Name AS TerritoryName
FROM Sales.SalesOrderHeader SOH
LEFT OUTER JOIN Sales.SalesPerson SP
ON SP.BusinessEntityID = SOH.SalesPersonID
LEFT OUTER JOIN HumanResources.Employee E
ON E.BusinessEntityID = SP.BusinessEntityID
LEFT OUTER JOIN Person.Person P
ON P.BusinessEntityID = E.BusinessEntityID
LEFT OUTER JOIN Sales.SalesTerritory T
ON T.TerritoryID = SOH.TerritoryID
WHERE T.Name = 'Northwest'
ORDER BY SOH.TotalDue DESC
```

FirstName	LastName	SalesOrderNumber	SalesAmount	TerritoryName
Stephen	Jiang	SO51830	126352.1615	Northwest
Linda	Mitchell	SO51705	124349.4919	Northwest
David	Campbell	SO46643	123497.0664	Northwest
Linda	Mitchell	SO51783	120955.8851	Northwest
Tate	Mensa-Annan	SO68508	119641.1984	Northwest
Tate	Mensa-Annan	SO50297	116390.7349	Northwest
David	Campbell	SO51711	114536.7764	Northwest
Linda	Mitchell	SO51721	109948.7407	Northwest
Pamela	Anaman-Wolfe	SO47033	105493.6317	Northwest
Linda	Mitchell	SO69471	105415.7003	Northwest
David	Campbell	SO51123	105281.9588	Northwest
Shu	Ito	SO44762	104545.315	Northwest
Pamela	Anaman-Wolfe	SO67297	103227.0017	Northwest
Pamela	Anaman-Wolfe	SO53518	99023.6736	Northwest

Aggregate Functions

An **aggregate function** performs a calculation on a set of values, and returns a single value.

Except for COUNT , **aggregate functions** ignore null values.

Aggregate functions (COUNT, MAX, MIN, SUM, AVG) are often used with the GROUP BY clause of the SELECT statement to group the result-set by one or more columns.

MAX

SQL Query 1 - BR_VCL/Brewster (SSP) - x

```
1 SELECT MAX(TotalDue)
2 FROM Sales.SalesOrderHeader
3
4 SELECT TotalDue
5 FROM Sales.SalesOrderHeader
6 ORDER BY TotalDue DESC
```

100 %

Results Messages

TotalDue
157407825
152764217
170512689
166375089
165028742
165065449
145741853
...

Query executed successfully. BREWSTER (31.0 SP1) WIN-APSMH-MVCL/Brewster... AdventureWorks2012 00:00:00 31465 rows

MIN

SQL Query 1 - BR_VCL/Brewster (SSP) - x

```
1 SELECT MIN(TotalDue)
2 FROM Sales.SalesOrderHeader
3
```

100 %

Results Messages

TotalDue
15183

Query executed successfully. BREWSTER (31.0 SP1) WIN-APSMH-MVCL/Brewster... AdventureWorks2012 00:00:00 1 rows

COUNT

SQL Query 1 - BR_VCL/Brewster (SSP) - x

```
1 SELECT COUNT(*)
2 FROM Sales.SalesOrderHeader
3
```

100 %

Results Messages

(No column name)
31465

Query executed successfully. BREWSTER (31.0 SP1) WIN-APSMH-MVCL/Brewster... AdventureWorks2012 00:00:00 1 rows

SQL Query 1 - BR_VCL/Brewster (SSP) - x

```
1 SELECT COUNT(DISTINCT SalesPersonID)
2 FROM Sales.SalesOrderHeader
3
4 SELECT COUNT(*)
5 FROM Sales.SalesOrderHeader
6 WHERE SalesPersonID IS NOT NULL
```

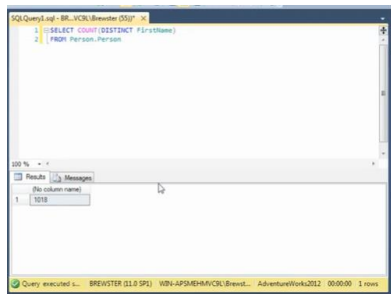
100 %

Results Messages

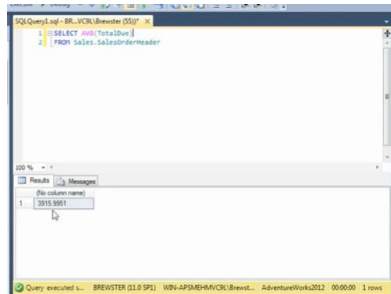
(No column name)
2658

Query executed successfully. BREWSTER (31.0 SP1) WIN-APSMH-MVCL/Brewster... AdventureWorks2012 00:00:00 2 rows

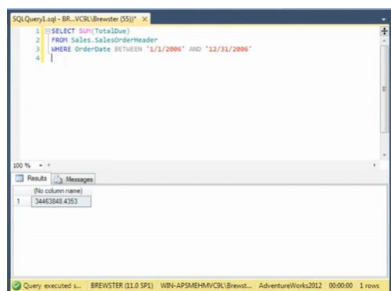
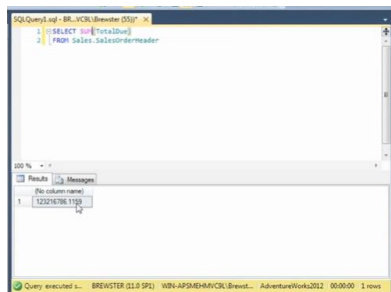
Use **COUNT** to get unique values



AVG



SUM

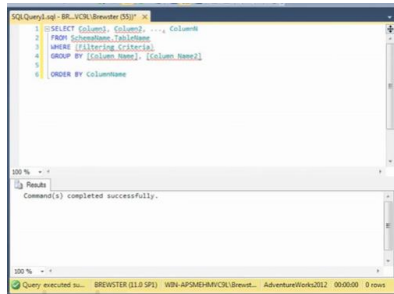


Grouping Data with the GROUP BY Clause

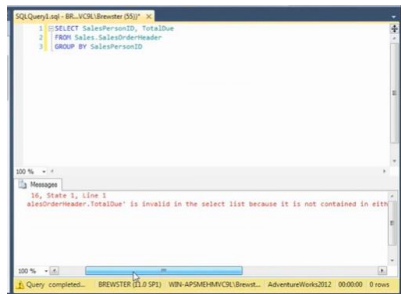
The **GROUP BY** clause is a SQL command that is used to **group** rows that have the same values. The **GROUP BY** clause is used in the SELECT statement.

Optionally it is used in conjunction with aggregate functions to produce summary reports from the database. That's what it does, summarizing data from the database

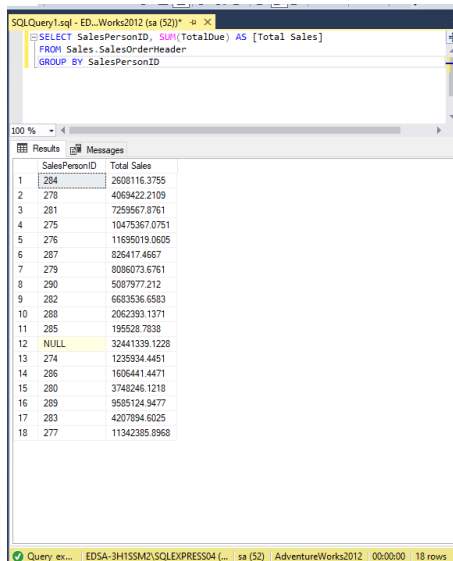
General form



DO NOT DO.



SQL requires that every column in the SELECT clause be included in either an aggregate function or in the GROUP BY clause.



One aggregate function.

SQLQuery1.sql - ED...Works2012 (sa (123)) - n - x

```

1 SELECT ProductID, SUM(Quantity) AS [Total In Stock]
2 FROM Production.ProductInventory
3 GROUP BY ProductID

```

100 % - 4

Results Messages

ProductID	Total In Stock
1	1085
2	1109
3	1352
4	1322
5	1361
6	893
7	439
8	797
9	1136
10	1750
11	1684
12	1684
13	1629
14	1750
15	1684
16	1364
17	1629
18	1601
19	1571
20	1405
21	844
22	1103
23	1093
24	1673
25	1072

Query executed successfully. AdventureWorks2012 00:00:01 432 rows

Multiple aggregate functions.

SQLQuery1.sql - ED...Works2012 (sa (123)) - n - x

```

1 SELECT ProductID, SUM(Quantity) AS [Total In Stock],
2 COUNT(*) AS [Total Locations]
3 FROM Production.ProductInventory
4 GROUP BY ProductID

```

100 % - 7

Results Messages

ProductID	Total In Stock	Total Locations
1	1085	3
2	1109	3
3	1352	3
4	1322	3
5	1361	3
6	893	3
7	439	3
8	797	3
9	1136	3
10	1750	3
11	1684	3
12	1684	3
13	1629	3
14	1750	3
15	1684	3
16	1364	3
17	1629	3
18	1601	3
19	1571	3
20	1405	3
21	844	3
22	1103	3
23	1093	3
24	1673	3
25	1072	3

Query executed successfully. AdventureWorks2012 00:00:00 432 rows

Multiple clauses

SQLQuery1.sql - BW...VCOLibender (DBP) - x

```

1 SELECT TerritoryID, SalesPersonID, SUM(TotalSales) AS [Total Sales]
2 FROM Sales.SalesOrderHeader SOH
3 WHERE OrderDate BETWEEN '12/1/2008' AND '12/31/2008'
4 GROUP BY TerritoryID, SalesPersonID
5 ORDER BY 1, 2

```

100 % - 7

Results Messages

TerritoryID	SalesPersonID	Total Sales
1	NULL	69823.354
2	1	6547054
3	1	69843.055
4	1	128617.701
5	1	281459.073
6	1	1327676.9342
7	1	286363.687

Query executed successfully. AdventureWorks2012 00:00:00 35 rows

Example

SQLQuery1.sql - BW...VCOLibender (DBP) - x

```

1 SELECT
2     ST.Name AS [Territory Name],
3     P.FirstName, P.LastName,
4     SUM(TotalSales) AS [Total Sales]
5 FROM Sales.SalesOrderHeader SOH
6 INNER JOIN Sales.SalesPerson SP
7 ON SP.BusinessEntityID = SOH.BusinessEntityID
8 INNER JOIN Person.Person P
9 ON P.BusinessEntityID = SP.BusinessEntityID
10 INNER JOIN Sales.SalesTerritory ST
11 ON ST.TerritoryID = SOH.TerritoryID
12 WHERE OrderDate BETWEEN '12/1/2008' AND '12/31/2008'
13 GROUP BY ST.Name, P.FirstName, P.LastName
14 ORDER BY 1, 2

```

100 % - 7

Results Messages

Territory Name	FirstName	LastName	Total Sales
Canada	Stephen	Jorg	76455.4218
Canada	Allen	Canon	1570545.046
Canada	Luís	Michael	554160.3187
Canada	Michael	Barthe	555551.042
Canada	Shirley	Bo	235168.4652
France	Amy	Aberts	555551.042
France	Paul	Salem	555551.042

Query executed successfully. AdventureWorks2012 00:00:00 27 rows

Filtering Groups with the HAVING Clause

A **HAVING** clause in **SQL** specifies that an **SQL SELECT** statement should only return rows where aggregate values meet the specified conditions.

HAVING filters records that work on summarized GROUP BY results. **HAVING** applies to summarized group records, whereas WHERE applies to individual records.

Only the groups that meet the HAVING criteria will be returned.

General form:

```
SQLQueryLog-BE-VCLBrewster (SQL) >
1 SELECT Column1, Column2, ..., ColumnN
2 FROM SchemaName.TableName
3 WHERE (filtering condition)
4 GROUP BY (ColumnName1)
5 HAVING (aggregate function) (comparison operator) (filtering criteria)
6 ORDER BY (ColumnName, Ordinal, Alias)
```

Territory Name	SalesPersonName	Total Sales
Canada	Christy Morgan	136059.9594
Canada	Joe Pal	2542719.5744
Canada	Joel Simoes	1156424.6549
Canada	Stephan Jiang	74654.4205
Central	Allan Carson	1570545.046
Central	Linda Michiel	556160.2167
Central	Michael Balle	595531.1542

Query executed successfully. BREWSTER (31.0 SP1) WIN-APSMHMYVCLBrewst... AdventureWorks2012 00:00:00 27 rows

Example

```
SQLQueryLog-BE-VCLBrewster (SQL) >
1 SELECT
2 ST.Name AS [Territory Name],
3 SUM(TotalSales) AS [Total Sales - 2006]
4 FROM Sales.SalesOrderHeader SOH
5 INNER JOIN Sales.SalesTerritory ST
6 ON ST.TerritoryID = SOH.TerritoryID
7 WHERE OrderDate BETWEEN '1/1/2006' AND '12/31/2006'
8 GROUP BY ST.Name
9 ORDER BY 1
```

Territory Name	Total Sales - 2006
Australia	2380454.8387
Canada	6130270.7554
Central	2595946.5541
France	1536232.896
Germany	570560.0274
Northwest	2751513.4245
Northwest	4855977.1032

Query executed successfully. BREWSTER (31.0 SP1) WIN-APSMHMYVCLBrewst... AdventureWorks2012 00:00:00 10 rows

Use HAVING

```
SQLQueryLog-BE-VCLBrewster (SQL) >
1 SELECT
2 ST.Name AS [Territory Name],
3 SUM(TotalSales) AS [Total Sales - 2006]
4 FROM Sales.SalesOrderHeader SOH
5 INNER JOIN Sales.SalesTerritory ST
6 ON ST.TerritoryID = SOH.TerritoryID
7 WHERE OrderDate BETWEEN '1/1/2006' AND '12/31/2006'
8 GROUP BY ST.Name
9 HAVING SUM(TotalSales) > 4000000
10 ORDER BY 1
```

Territory Name	Total Sales - 2006
Canada	6130270.7554
Northwest	4855977.1032
Southwest	8603200.3625

Query executed successfully. BREWSTER (31.0 SP1) WIN-APSMHMYVCLBrewst... AdventureWorks2012 00:00:00 3 rows

```
SQLQueryLog-BE-VCLBrewster (SQL) >
1 (SELECT PS.Name AS [SubCategory Name], COUNT(*) AS [Product Count])
2 FROM Production.Product P
3 INNER JOIN Production.ProductSubcategory PS
4 ON P.ProductSubcategoryID = PS.ProductSubcategoryID
5 GROUP BY PS.Name
6 HAVING COUNT(*) >= 10
7 ORDER BY 1
```

SubCategory Name	Product Count
Mountain Bikes	32
Mountain Frames	28
Road Bikes	43
Road Frames	33
Touring Bikes	22
Touring Frames	18

Query executed successfully. BREWSTER (31.0 SP1) WIN-APSMHMYVCLBrewst... AdventureWorks2012 00:00:00 6 rows

3 examples of the HAVING function with aggregate functions and operators.

SQLQuery1.sql - BR...VCRLibreuter (SSP) - X

```

1  SELECT
2      Department AS [Department Name],
3      COUNT(*) AS [Employee Count]
4  FROM HumanResources.vEmployeeDepartment
5  GROUP BY Department
6  HAVING COUNT(*) >= 6

```

100 %

Results Messages

	Department Name	Employee Count
1	Finance	10
2	Information Services	10
3	Marketing	9
4	Production	179
5	Purchasing	12
6	Sales	10

Query executed successfully. BREWSTER (21.9 SP1) WIN-APSMH-HVCRLibreuter... AdventureWorks2012 00:00:00 6 rows

SQLQuery1.sql - BR...VCRLibreuter (SSP) - X

```

1  SELECT
2      Department AS [Department Name],
3      COUNT(*) AS [Employee Count]
4  FROM HumanResources.vEmployeeDepartment
5  GROUP BY Department
6  HAVING COUNT(*) >= [6, 10]

```

100 %

Results Messages

	Department Name	Employee Count
1	Engineering	6
2	Finance	10
3	Human Resources	6
4	Information Services	10
5	Production Control	6
6	Quality Assurance	6
7	Shipping and Receiving	6

Query executed successfully. BREWSTER (21.9 SP1) WIN-APSMH-HVCRLibreuter... AdventureWorks2012 00:00:00 7 rows

SQLQuery1.sql - BR...VCRLibreuter (SSP) - X

```

1  SELECT
2      Department AS [Department Name],
3      COUNT(*) AS [Employee Count]
4  FROM HumanResources.vEmployeeDepartment
5  GROUP BY Department
6  HAVING COUNT(*) >= [6] AND [6]

```

100 %

Results Messages

	Department Name	Employee Count
1	Engineering	6
2	Facilities and Maintenance	7
3	Finance	10
4	Human Resources	6
5	Information Services	10
6	Marketing	9
7	Production Control	6
8	Quality Assurance	6
9	Shipping and Receiving	6

Query executed successfully. BREWSTER (21.9 SP1) WIN-APSMH-HVCRLibreuter... AdventureWorks2012 00:00:00 9 rows

Example

SQLQuery1.sql - BR...VCRLibreuter (SSP) - X

```

1  SELECT
2      SalesPersonID,
3      SUM(TotalDue) AS [Total Sales Amount],
4      COUNT(*) AS [Total Sales Count]
5  FROM Sales.SalesOrderHeader
6  WHERE OrderDate BETWEEN '12/1/2006' AND '12/31/2006'
7  AND SalesPersonID IS NOT NULL
8  GROUP BY SalesPersonID
9  HAVING SUM(TotalDue) >= 2000000
10 AND COUNT(*) >= 75
11 ORDER BY [Total Sales Amount] DESC

```

100 %

Results Messages

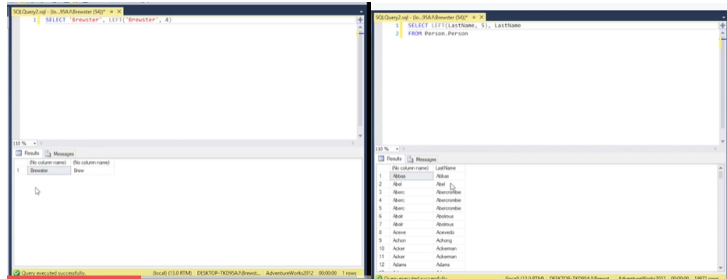
	SalesPersonID	Total Sales Amount	Total Sales Count
1	277	420604.7229	140
2	276	365326.6626	127
3	275	347028.7471	129
4	289	284279.9744	84
5	279	278719.4816	153
6	282	254917.8727	82

Query executed successfully. BREWSTER (21.9 SP1) WIN-APSMH-HVCRLibreuter... AdventureWorks2012 00:00:00 6 rows

Using SQL Server Functions

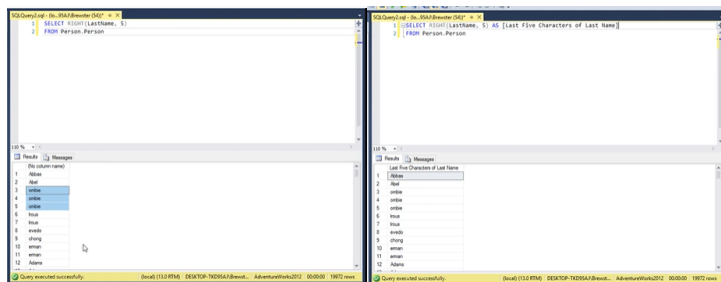
LEFT function

To pass a string and only return an x amount of the LEFT most characters within that string.



RIGHT function

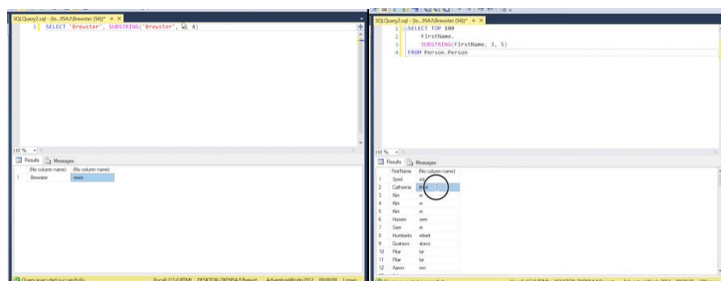
To pass a string and only return an x amount of the RIGHT most characters within that string.



SUBSTRING function

It extracts a **substring** with a specified length starting from a location in an input string.

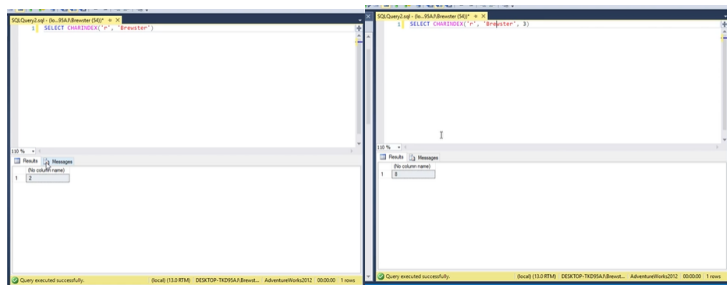
The length is a positive integer that specifies the number of characters of the **substring** to be returned.



CHARINDEX function

The CHARINDEX() function searches for a substring in a string, and returns the position. If the substring is not found, this function returns 0.

It has a 3rd argument that specifies the starting position.

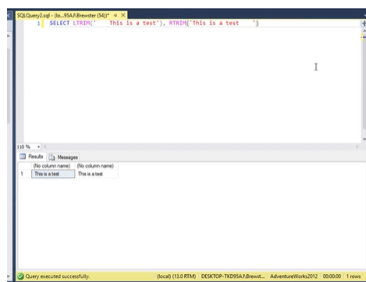


LTRIM function

The LTRIM function removes all space characters from the left-hand side of a string.

RTRIM function

The RTRIM function removes all space characters from the right-hand side of a string.

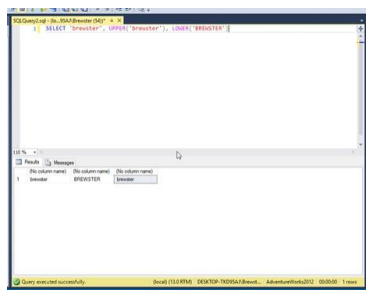


UPPER function

It is used to capitalize all characters in a string.

LOWER function

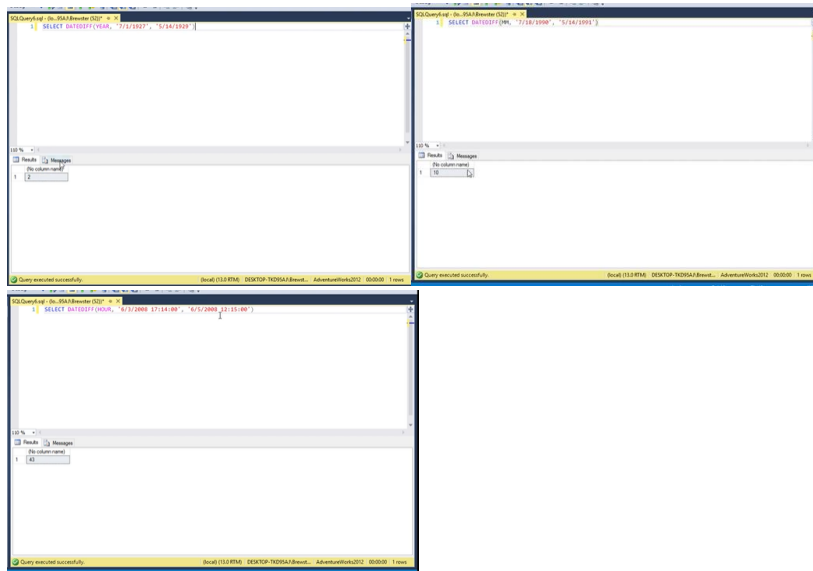
Converts a string to lower-case.



LEN function

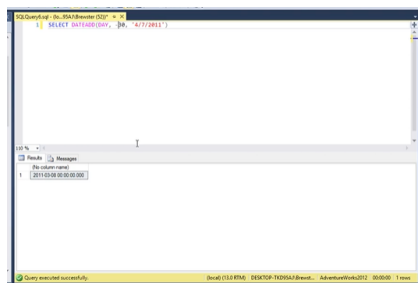
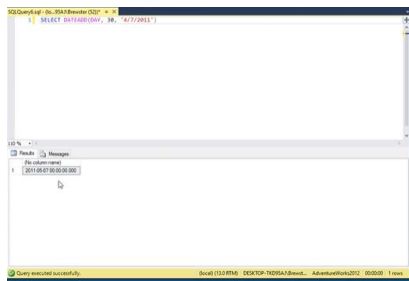
Find the length of a string.

Returns the difference between two dates.

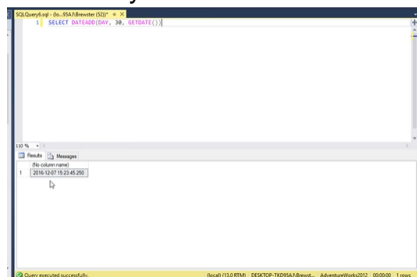


DATEADD function

To add a specified value to a specified date part of a date.



Add 30 days to current date



NULL handling functions

COALESCE function

Take any cell with NULL and convert it to a blank space.

The screenshot shows a SQL query in a text editor and its results in a table.

SQL Query:

```
SELECT TOP 100
FROM   PERSON, CONTACT(cid,lastname) AS cid_lastname, lastname
FROM   PERSON, PERSON
```

Results Table:

	First Name	MidName	Last Name
1	Isabel	E	Adams
2	Catherine	B	And
3	My		Baroness
4	Ken		Baroness
5	Ken		Baroness
6	Norman	E	Barth
7	Sam		Barth
8	Subalini		Barth
9	Subalini		Barth
10	Peter		Barth
11	Ken	C	Barth
12	Aaron	B	Adams

NULLIF function

Returns **NULL** if two expressions are equal, otherwise it returns the first expression.

```

1 select BillAddressID, ShipAddressID
2      , NullIf(BillAddressID, ShipAddressID)
3      FROM Sales.SalesOrderHeader

```

10 rows ->

10 Rows | 3 Columns

BillAddressID	ShipAddressID	Row count (avg)
15071	15071	Null
15068	15068	Null
15067	15067	Null
15067	15067	Null
15036	15036	Null
15196	15196	Null
15169	15169	Null
15099	15099	Null
14675	14675	Null
20771	20771	Null
20754	20754	Null
20753	20753	Null

Query executed successfully. (SQL) (11.0.0.6000) DEVCON NINJA@bentley... Adventureworks2012 06/06/2016 1:48:55

COALESCE and NULLIF

SQL Server Enterprise Manager

```

SELECT BillAddressID, ShipAddressID,
       COUNTRYID, BillAddressID, ShipAddressID, 1
FROM Sales.SalesOrderHeader

```

Results (1) Messages

	BillAddressID	ShipAddressID	This column is empty
1	865	865	1
2	821	821	1
3	817	817	1
4	482	482	1
5	1075	1075	1
6	876	876	1
7	845	845	1
8	1074	1074	1
9	629	629	1
10	629	629	1
11	896	896	1
12	892	892	1

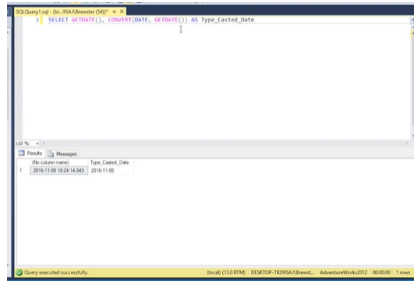
Query executed successfully. (local) [11.0.6002] DEKTOP-7676564-8... Adventureworks2008 200802 21483

SQL Server Data Types and Type Casting

CONVERT

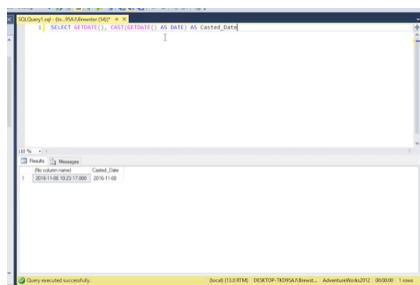
The **CONVERT** function **converts** an expression from one datatype to another datatype.

Can be one of the following: bigint, int, smallint, tinyint, bit, decimal, numeric, money, float etc.



CAST

The CAST function in SQL converts data from one data type to another, such as from numeric data into character string data.



Working with Table Expressions

Derived table

The **SQL Derived Table** is nothing but a Subquery used in the From Clause. It has to have a name.

```
1 SELECT *
2 FROM (
3     SELECT BusinessEntityID, FirstName, LastName
4     FROM Person.Person
5     ) AS PersonTable
```

BusinessEntityID	FirstName	LastName
1	Adrian	Adrian
2	Adrian	Adrian
3	Adrian	Adrian
4	Adrian	Adrian
5	Adrian	Adrian
6	Adrian	Adrian
7	Adrian	Adrian
8	Adrian	Adrian
9	Adrian	Adrian
10	Adrian	Adrian
11	Adrian	Adrian
12	Adrian	Adrian

Example to reduce redundancy

```
1 SELECT *
2 FROM (SELECT OrderYear, COUNT(*) AS SalesCount
3       FROM Sales.SalesOrderHeader
4       WHERE YEAR(OrderDate) = 2008
5       GROUP BY YEAR(OrderDate)) AS SalesCount
6
7 SELECT OrderYear, COUNT(*) AS SalesCount
8 FROM (
9     SELECT YEAR(OrderDate) AS OrderYear, SalesOrderID
10    FROM Sales.SalesOrderHeader
11    ) AS SalesOrderTable
12 WHERE OrderYear = 2008
13 GROUP BY OrderYear
```

OrderYear	SalesCount
2008	350

Example

```
1 SELECT *
2 FROM (
3     SELECT
4         BusinessEntityID,
5         NationalIDNumber,
6         YEAR(OrderDate) AS OrderYear,
7         YEAR(OrderDate) AS OrderYear
8     FROM HumanResources.Employee
9     ) AS HR_Employee
10 WHERE OrderYear = 2008
```

BusinessEntityID	NationalIDNumber	OrderYear
1	69020000	2008
2	69020000	2008
3	69020000	2008
4	69020000	2008
5	69020000	2008
6	69020000	2008
7	69020000	2008
8	69020000	2008
9	69020000	2008
10	69020000	2008
11	69020000	2008
12	69020000	2008

```
1 SELECT *
2 FROM (
3     SELECT
4         BusinessEntityID,
5         NationalIDNumber,
6         YEAR(OrderDate) AS OrderYear,
7         YEAR(OrderDate) AS OrderYear
8     FROM HumanResources.Employee
9     ) AS HR_Employee
10 WHERE OrderYear = 2008
```

BusinessEntityID	NationalIDNumber	OrderYear
1	69020000	2008
2	69020000	2008
3	69020000	2008
4	69020000	2008
5	69020000	2008
6	69020000	2008
7	69020000	2008
8	69020000	2008
9	69020000	2008
10	69020000	2008
11	69020000	2008
12	69020000	2008

Nested derived tables

These work inside out.

Query 1: Nested Subqueries

```
1 SELECT *
2 FROM (
3     SELECT BusinessEntityID, NationalIDNumber, BirthYear, YEAR(MinDate) AS HireYear
4     FROM (
5         SELECT BusinessEntityID, NationalIDNumber, YEAR(BirthDate) AS BirthYear, MinDate
6         FROM HumanResources.Employee
7         ) AS Innermost
8     ) AS Outermost
```

BusinessEntityID	NationalIDNumber	BirthYear	HireYear
1	1000000000	1968	2000
2	2000000000	1968	2002
3	3000000000	1968	2002
4	4000000000	1968	2002
5	5000000000	1968	2002
6	6000000000	1968	2002
7	7000000000	1968	2002
8	8000000000	1968	2002
9	9000000000	1968	2002
10	1000000000	1979	2001
11	1100000000	1979	2001
12	1200000000	1980	2002

Query 2: Simplified Query

```
1 SELECT *
2 FROM (
3     SELECT BusinessEntityID, NationalIDNumber, BirthYear, YEAR(MinDate) AS HireYear
4     FROM (
5         SELECT BusinessEntityID, NationalIDNumber, YEAR(BirthDate) AS BirthYear, MinDate
6         FROM HumanResources.Employee
7         ) AS Innermost
8     ) AS Outermost
```

BusinessEntityID	NationalIDNumber	BirthYear	HireYear
1	1000000000	1968	2000
2	2000000000	1968	2002
3	3000000000	1968	2002
4	4000000000	1968	2002
5	5000000000	1968	2002
6	6000000000	1968	2002
7	7000000000	1968	2002
8	8000000000	1968	2002
9	9000000000	1968	2002
10	1000000000	1979	2001

Using derived tables with JOINS

Query 1: Complex Query with JOINS

```
1 SELECT SalesByYear.SalesYear, SalesByYear.TotalRevenue, HireByYear.HireYear
2 FROM (
3     SELECT SalesYear, SUM(TotalDue) AS TotalRevenue
4     FROM (
5         SELECT YEAR(OrderDate) AS SalesYear, TotalDue
6         FROM Sales.SalesOrderHeader
7         ) AS SalesDetails
8     GROUP BY SalesYear
9     ) AS SalesByYear
10 LEFT JOIN (
11     SELECT HireYear, COUNT(BusinessEntityID) AS NumHires
12     FROM (
13         SELECT YEAR(MinDate) AS HireYear, BusinessEntityID
14         FROM HumanResources.Employee
15         ) AS HireDetails
16     GROUP BY HireYear
17     ) AS HireByYear
18 ON SalesByYear.SalesYear = HireByYear.SalesYear
19 ORDER BY 1
```

SalesYear	TotalRevenue	HireYear
2000	1000000000	0
2001	2000000000	1
2002	3000000000	2
2003	4000000000	3

Part One

Query 1: Part One Query

```
1 SELECT SalesByYear.SalesYear, SalesByYear.TotalRevenue, HireByYear.HireYear
2 FROM (
3     SELECT SalesYear, SUM(TotalDue) AS TotalRevenue
4     FROM (
5         SELECT YEAR(OrderDate) AS SalesYear, TotalDue
6         FROM Sales.SalesOrderHeader
7         ) AS SalesDetails
8     GROUP BY SalesYear
9     ) AS SalesByYear
10 LEFT JOIN (
11     SELECT HireYear, COUNT(BusinessEntityID) AS NumHires
12     FROM (
13         SELECT YEAR(MinDate) AS HireYear, BusinessEntityID
14         FROM HumanResources.Employee
15         ) AS HireDetails
16     GROUP BY HireYear
17     ) AS HireByYear
18 ON SalesByYear.SalesYear = HireByYear.SalesYear
19 ORDER BY 1
```

SalesYear	TotalRevenue	HireYear
2000	1000000000	0
2001	2000000000	1
2002	3000000000	2
2003	4000000000	3

Part Two

Query 1: Part Two Query

```
1 SELECT SalesByYear.SalesYear, SalesByYear.TotalRevenue, HireByYear.HireYear
2 FROM (
3     SELECT SalesYear, SUM(TotalDue) AS TotalRevenue
4     FROM (
5         SELECT YEAR(OrderDate) AS SalesYear, TotalDue
6         FROM Sales.SalesOrderHeader
7         ) AS SalesDetails
8     GROUP BY SalesYear
9     ) AS SalesByYear
10 LEFT JOIN (
11     SELECT HireYear, COUNT(BusinessEntityID) AS NumHires
12     FROM (
13         SELECT YEAR(MinDate) AS HireYear, BusinessEntityID
14         FROM HumanResources.Employee
15         ) AS HireDetails
16     GROUP BY HireYear
17     ) AS HireByYear
18 ON SalesByYear.SalesYear = HireByYear.SalesYear
19 ORDER BY 1
```

SalesYear	TotalRevenue	HireYear
2000	1000000000	0
2001	2000000000	1
2002	3000000000	2
2003	4000000000	3

Working with CTEs (Common Table Expression)

A **CTE** (Common Table Expression) is a temporary result set that you can reference within another **SELECT**, **INSERT**, **UPDATE**, or **DELETE** statement.

A derived table vs. CTE

The left screenshot shows a query using a derived table in the FROM clause. The right screenshot shows a query using a CTE defined with the WITH clause.

```
1 SELECT OrderYear, COUNT(*) AS SalesCount
2 FROM (
3     SELECT YEAR(OrderDate) AS OrderYear, SalesOrderID
4     FROM Sales.SalesOrderHeader
5     ) AS SalesOrderID
6 WHERE OrderYear = 2000
7 GROUP BY OrderYear
8 GO
9
10
11 WITH SalesOrderID
12 AS (
13     SELECT YEAR(OrderDate) AS OrderYear, SalesOrderID
14     FROM Sales.SalesOrderHeader
15 )
16 SELECT OrderYear, COUNT(*) AS SalesCount
17 FROM SalesOrderID
18 WHERE OrderYear = 2000
19 GROUP BY OrderYear
```

The right screenshot shows a query using a CTE. The CTE is defined with the WITH clause and used in the FROM clause.

```
1 SELECT
2 FROM
3     SELECT SalesOrderID, NationalIDNumber, YEAR(BirthDate) AS BirthYear,
4     FROM Sales.SalesOrderHeader
5     JOIN HumanResources.Employee
6     ON SOH.SalesOrderID = EMP.EmployeeID
7 WHERE BirthYear = 2000
8 GO
9
10
11 WITH CTE_Sales
12 AS (
13     SELECT SalesOrderID, NationalIDNumber, YEAR(BirthDate) AS BirthYear,
14     FROM Sales.SalesOrderHeader
15     JOIN HumanResources.Employee
16     ON SOH.SalesOrderID = EMP.EmployeeID
17 WHERE BirthYear = 2000
18 )
19 SELECT
20 FROM CTE_Sales
21 WHERE BirthYear = 2000
```

To reduce redundancy using a CTE after a derived table.

The left screenshot shows a query using a derived table in the FROM clause. The right screenshot shows a query using a CTE defined with the WITH clause.

```
1 SELECT
2     SalesCurrentYear.SalesYear,
3     SalesCurrentYear.TotalSales AS AnnualSales,
4     SalesPriorYear.TotalSales AS PriorYearSales
5 FROM (
6     SELECT YEAR(OrderDate) AS SalesYear, SUM(TotalDue) AS TotalSales
7     FROM Sales.SalesOrderHeader
8     GROUP BY YEAR(OrderDate)
9 ) AS SalesCurrentYear
10
11 SELECT YEAR(OrderDate) AS SalesYear, SUM(TotalDue) AS TotalSales
12 FROM Sales.SalesOrderHeader
13 GROUP BY YEAR(OrderDate)
14 ) AS SalesPriorYear
15 ON SalesCurrentYear.SalesYear = 1 - SalesPriorYear.SalesYear
16 ORDER BY 1
```

The right screenshot shows a query using a CTE. The CTE is defined with the WITH clause and used in the FROM clause.

```
1 SELECT
2     SalesCurrentYear.SalesYear,
3     SalesCurrentYear.TotalSales AS AnnualSales,
4     SalesPriorYear.TotalSales AS PriorYearSales
5 FROM (
6     SELECT YEAR(OrderDate) AS SalesYear, SUM(TotalDue) AS TotalSales
7     FROM Sales.SalesOrderHeader
8     GROUP BY YEAR(OrderDate)
9 ) AS SalesCurrentYear
10
11 SELECT
12     SalesCurrentYear.SalesYear,
13     SalesCurrentYear.TotalSales AS AnnualSales,
14     SalesPriorYear.TotalSales AS PriorYearSales
15 FROM (
16     SELECT YEAR(OrderDate) AS SalesYear, SUM(TotalDue) AS TotalSales
17     FROM Sales.SalesOrderHeader
18     GROUP BY YEAR(OrderDate)
19 ) AS SalesPriorYear
20 ON SalesCurrentYear.SalesYear = 1 - SalesPriorYear.SalesYear
21 ORDER BY 1
```

Nested CTEs

The screenshot shows a query using nested CTEs. The CTEs are defined with the WITH clause and used in the FROM clause.

```
1 WITH CTE_Sales
2 AS (
3     SELECT YEAR(OrderDate) AS SalesYear, SalesOrderID, TotalDue
4     FROM Sales.SalesOrderHeader
5 )
6 AS 1
7 SELECT SalesYear, COUNT(*) AS SalesCount,
8     SUM(TotalDue) AS AnnualSales
9 FROM CTE_Sales
10 GROUP BY SalesYear
11
12
13 SELECT SalesYear, SalesCount, AnnualSales
14 FROM CTE_Sales
15 WHERE SalesCount > 5000
16 ORDER BY 1
```

CASE statements

The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement).

So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

Left Screenshot Query:

```
SELECT ProductID, ListPrice
FROM Product
WHERE ListPrice >= 100
UNION ALL
SELECT ProductID, ListPrice
FROM Product
WHERE ListPrice < 100
```

Right Screenshot Query:

```
SELECT ProductID, ListPrice
FROM Product
WHERE ListPrice >= 100
UNION ALL
SELECT ProductID, ListPrice
FROM Product
WHERE ListPrice < 100
```

Example

FLOOR function

Returns an integer after removing the decimal point.

Step 1

Complete

Left Screenshot Query:

```
SELECT P.ProductID, P.ListPrice,
       CASE WHEN P.ListPrice >= 100 THEN 'Expensive Product'
            ELSE 'Inexpensive Product' END AS ProductType
FROM Product P
```

Right Screenshot Query:

```
SELECT P.ProductID, P.ListPrice,
       CASE WHEN P.ListPrice >= 100 THEN 'Expensive Product'
            ELSE 'Inexpensive Product' END AS ProductType
FROM Product P
GROUP BY ProductType
```

Group by age and count.

Place the last block in a second CTE.

Query:

```
WITH CTE1 AS (
    SELECT P.ProductID, P.ListPrice,
           CASE WHEN P.ListPrice >= 100 THEN 'Expensive Product'
                ELSE 'Inexpensive Product' END AS ProductType
    FROM Product P
)
SELECT ProductType, COUNT(*) AS Count
FROM CTE1
GROUP BY ProductType
```

[illegible][illegible][illegible]

```

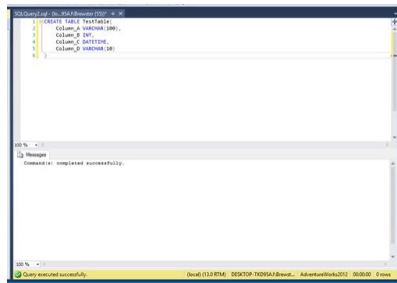
10) Query (SQL) [SELECT] - 10
11)
12)
13)
14)
15)
16)
17)
18)
19)
20)
21)
22)
23)
24)
25)
26)
27)
28)
29)
30)
31)
32)
33)
34)
35)
36)
37)
38)
39)
40)
41)
42)
43)
44)
45)
46)
47)
48)
49)
50)
51)
52)
53)
54)
55)
56)
57)
58)
59)
60)
61)
62)
63)
64)
65)
66)
67)
68)
69)
70)
71)
72)
73)
74)
75)
76)
77)
78)
79)
80)
81)
82)
83)
84)
85)
86)
87)
88)
89)
90)
91)
92)
93)
94)
95)
96)
97)
98)
99)
100)
101)
102)
103)
104)
105)
106)
107)
108)
109)
110)
111)
112)
113)
114)
115)
116)
117)
118)
119)
120)
121)
122)
123)
124)
125)
126)
127)
128)
129)
130)
131)
132)
133)
134)
135)
136)
137)
138)
139)
140)
141)
142)
143)
144)
145)
146)
147)
148)
149)
150)
151)
152)
153)
154)
155)
156)
157)
158)
159)
160)
161)
162)
163)
164)
165)
166)
167)
168)
169)
170)
171)
172)
173)
174)
175)
176)
177)
178)
179)
180)
181)
182)
183)
184)
185)
186)
187)
188)
189)
190)
191)
192)
193)
194)
195)
196)
197)
198)
199)
200)
201)
202)
203)
204)
205)
206)
207)
208)
209)
210)
211)
212)
213)
214)
215)
216)
217)
218)
219)
220)
221)
222)
223)
224)
225)
226)
227)
228)
229)
230)
231)
232)
233)
234)
235)
236)
237)
238)
239)
240)
241)
242)
243)
244)
245)
246)
247)
248)
249)
250)
251)
252)
253)
254)
255)
256)
257)
258)
259)
260)
261)
262)
263)
264)
265)
266)
267)
268)
269)
270)
271)
272)
273)
274)
275)
276)
277)
278)
279)
280)
281)
282)
283)
284)
285)
286)
287)
288)
289)
290)
291)
292)
293)
294)
295)
296)
297)
298)
299)
300)
301)
302)
303)
304)
305)
306)
307)
308)
309)
310)
311)
312)
313)
314)
315)
316)
317)
318)
319)
320)
321)
322)
323)
324)
325)
326)
327)
328)
329)
330)
331)
332)
333)
334)
335)
336)
337)
338)
339)
340)
341)
342)
343)
344)
345)
346)
347)
348)
349)
350)
351)
352)
353)
354)
355)
356)
357)
358)
359)
360)
361)
362)
363)
364)
365)
366)
367)
368)
369)
370)
371)
372)
373)
374)
375)
376)
377)
378)
379)
380)
381)
382)
383)
384)
385)
386)
387)
388)
389)
390)
391)
392)
393)
394)
395)
396)
397)
398)
399)
400)
401)
402)
403)
404)
405)
406)
407)
408)
409)
410)
411)
412)
413)
414)
415)
416)
417)
418)
419)
420)
421)
422)
423)
424)
425)
426)
427)
428)
429)
430)
431)
432)
433)
434)
435)
436)
437)
438)
439)
440)
441)
442)
443)
444)
445)
446)
447)
448)
449)
450)
451)
452)
453)
454)
455)
456)
457)
458)
459)
460)
461)
462)
463)
464)
465)
466)
467)
468)
469)
470)
471)
472)
473)
474)
475)
476)
477)
478)
479)
480)
481)
482)
483)
484)
485)
486)
487)
488)
489)
490)
491)
492)
493)
494)
495)
496)
497)
498)
499)
500)
501)
502)
503)
504)
505)
506)
507)
508)
509)
510)
511)
512)
513)
514)
515)
516)
517)
518)
519)
520)
521)
522)
523)
524)
525)
526)
527)
528)
529)
530)
531)
532)
533)
534)
535)
536)
537)
538)
539)
540)
541)
542)
543)
544)
545)
546)
547)
548)
549)
550)
551)
552)
553)
554)
555)
556)
557)
558)
559)
560)
561)
562)
563)
564)
565)
566)
567)
568)
569)
570)
571)
572)
573)
574)
575)
576)
577)
578)
579)
580)
581)
582)
583)
584)
585)
586)
587)
588)
589)
590)
591)
592)
593)
594)
595)
596)
597)
598)
599)
600)
601)
602)
603)
604)
605)
606)
607)
608)
609)
610)
611)
612)
613)
614)
615)
616)
617)
618)
619)
620)
621)
622)
623)
624)
625)
626)
627)
628)
629)
630)
631)
632)
633)
634)
635)
636)
637)
638)
639)
640)
641)
642)
643)
644)
645)
646)
647)
648)
649)
650)
651)
652)
653)
654)
655)
656)
657)
658)
659)
660)
661)
662)
663)
664)
665)
666)
667)
668)
669)
670)
671)
672)
673)
674)
675)
676)
677)
678)
679)
680)
681)
682)
683)
684)
685)
686)
687)
688)
689)
690)
691)
692)
693)
694)
695)
696)
697)
698)
699)
700)
701)
702)
703)
704)
705)
706)
707)
708)
709)
710)
711)
712)
713)
714)
715)
716)
717)
718)
719)
720)
721)
722)
723)
724)
725)
726)
727)
728)
729)
730)
731)
732)
733)
734)
735)
736)
737)
738)
739)
740)
741)
742)
743)
744)
745)
746)
747)
748)
749)
750)
751)
752)
753)
754)
755)
756)
757)
758)
759)
760)
761)
762)
763)
764)
765)
766)
767)
768)
769)
770)
771)
772)
773)
774)
775)
776)
777)
778)
779)
780)
781)
782)
783)
784)
785)
786)
787)
788)
789)
790)
791)
792)
793)
794)
795)
796)
797)
798)
799)
800)
801)
802)
803)
804)
805)
806)
807)
808)
809)
810)
811)
812)
813)
814)
815)
816)
817)
818)
819)
820)
821)
822)
823)
824)
825)
826)
827)
828)
829)
830)
831)
832)
833)
834)
835)
836)
837)
838
```


Creating Tables and Inserting/Updating Data

CREATE TABLE

Creating a basic **table** involves naming the **table** and defining its columns and each column's data type. The **SQL CREATE TABLE** statement is used to **create** a new **table**.

General form

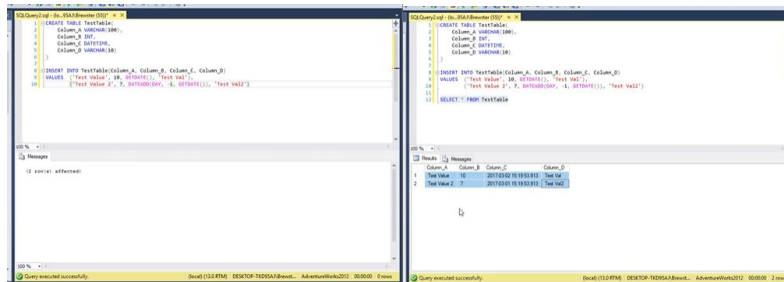


INSERT INTO

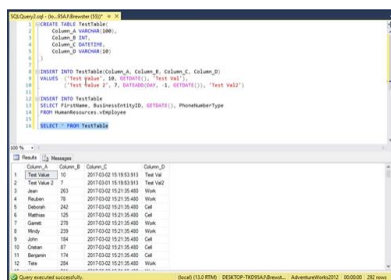
The **INSERT INTO** statement is used to add new data to a database. The **INSERT INTO** statement adds a new record to a table.

Can contain values for some or all of its columns.
Can be combined with a **SELECT** to insert records.

Manually inserting data.



Use **SELECT** statement



UPDATE

An **SQL UPDATE** statement changes the data of one or more records in a table. Either all the rows can be **updated**, or a subset may be chosen using a condition.

Query is used to modify the existing records in a table.

You can use the WHERE clause with the **UPDATE** query to **update** the selected rows, otherwise all the rows would be affected.

Update all the rows

[illegible]

Update a row

[illegible]