

5

Using Logical Operators in SQL Statements

This chapter will cover the use of logical operators for increasing the functionality of SELECT statements. Logical operators provide the ability to combine one or more conditions to a WHERE clause.

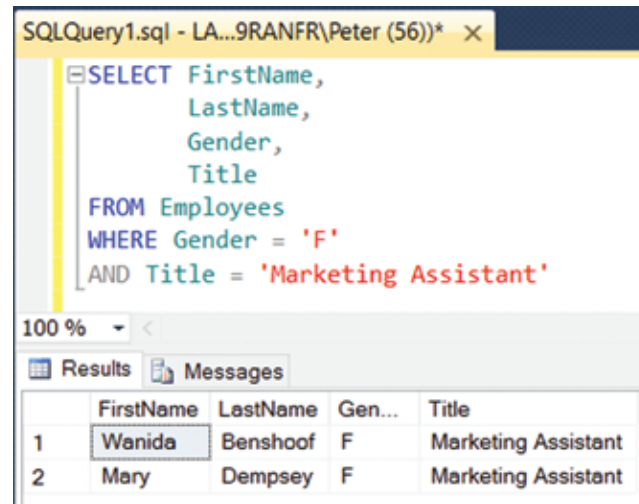
CHAPTER OBJECTIVES

Now that we've learned to use SELECT statements to retrieve data it's time to move on to learning more techniques for selecting the data using logical operators. The objectives of this chapter are as follows:

- Understand how to write complex SELECT statements using the AND operator, OR operator, and NOT operator.
- Understand how to write SELECT statements to produce lists of values and ranges using the BETWEEN operator.
- Understand how to perform comparisons using the LIKE operator to find rows based on patterns of characters.
- Understand how to check for null data using the IS NULL operator.
- Understand how to use expressions in the WHERE clause to provide specific results.

AND Operator

The AND operator is used with the WHERE clause to link two or more conditions. For example, if the manager wanted a list of all female employees whose title is “Marketing Assistant” we would use conditions to test for gender and title. Figure 1 demonstrates how to do this using the AND operator.



SQLQuery1.sql - LA...9RANFR\Peter (56))* X

```

SELECT FirstName,
       LastName,
       Gender,
       Title
FROM Employees
WHERE Gender = 'F'
AND Title = 'Marketing Assistant'

```

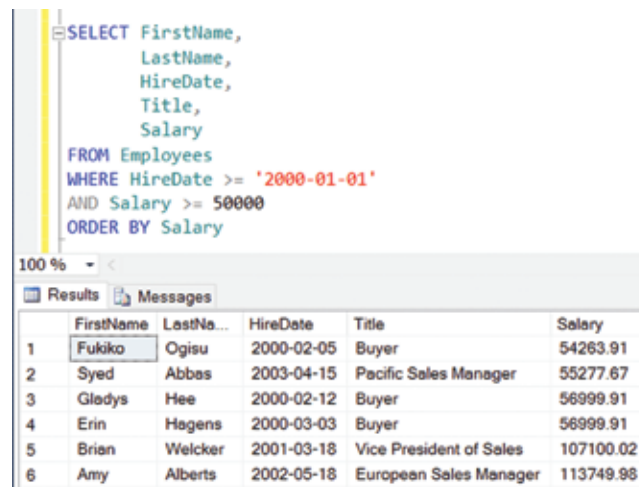
100 % <

Results Messages

	FirstName	LastName	Gen...	Title
1	Wanida	Benshoof	F	Marketing Assistant
2	Mary	Dempsey	F	Marketing Assistant

Figure 1

As another example, let's say a manager is interested in seeing a list of employees who were hired in the year 2000 or later and are making greater than or equal to \$50,000. Figure 2 demonstrates the SQL used to retrieve this information.



```

SELECT FirstName,
       LastName,
       HireDate,
       Title,
       Salary
FROM Employees
WHERE HireDate >= '2000-01-01'
AND Salary >= 50000
ORDER BY Salary

```

100 % <

Results Messages

	FirstName	LastNa...	HireDate	Title	Salary
1	Fukiko	Ogisu	2000-02-05	Buyer	54263.91
2	Syed	Abbas	2003-04-15	Pacific Sales Manager	55277.67
3	Gladys	Hee	2000-02-12	Buyer	56999.91
4	Erin	Hagens	2000-03-03	Buyer	56999.91
5	Brian	Welcker	2001-03-18	Vice President of Sales	107100.02
6	Amy	Alberts	2002-05-18	European Sales Manager	113749.98

Figure 2

We can use many different conditions in our queries. As an example, Figure 3 shows a query that is using five different conditions. The query returns a list of employees who were hired in the year 2000 or later, with a salary greater than or equal to \$50,000, is an active employee, is female, and is married.

```

SELECT FirstName,
       LastName,
       HireDate,
       Title,
       Salary
FROM Employees
WHERE HireDate >= '2000-01-01'
AND Salary >= 50000
AND Active = 'YES'
AND Gender = 'F'
AND MaritalStatus = 'M'
ORDER BY Salary

```

	FirstName	LastName	HireDate	Title	Salary
1	Gladys	Hee	2000-02-12	Buyer	56999.91
2	Amy	Alberts	2002-05-18	European Sales Manager	113749.98

Figure 3

OR Operator

When using the OR operator, it isn't necessary for both conditions to be met. If at least one condition is true, rows will be returned. Figure 4 shows results on a query that finds employees who either have vacation hours greater than 90 or sick hours less than 30. A list of all employees who meet at least one of the conditions is retrieved. Only one employee, Brian Welcker meets both conditions.

```

SELECT EmployeeID, FirstName,
       LastName,
       VacationHours,
       SickLeaveHours
FROM Employees
WHERE VacationHours > 90
OR SickLeaveHours < 30
ORDER BY VacationHours, SickLeaveHours

```

	EmployeeID	FirstName	LastName	VacationHours	SickLeaveHours
1	140	Laura	Norman	0	20
2	12	Terri	Duffy	1	20
3	3	Rob	Tamburello	2	21
4	267	Michael	Sullivan	3	21
5	270	Sharon	Selavaria	4	22
6	9	Gail	Erickson	5	22
7	11	Josief	Goldberg	6	23
8	265	Janice	Galvin	8	24
9	5	Thierry	D'Nofro	9	24
10	260	Jo	Berry	91	65
11	49	Christian	Kleinerman	92	66
12	273	Brian	Welcker	93	25
13	85	Pilar	Ackerman	93	66
14	121	Matthias	Berndt	94	67
15	35	Vamsi	Kuppa	95	67

Figure 4

Be careful when using multiple conditions as the results can be incorrect if the SQL statement is incorrectly written. For example, let's say a manager requests a list of **male employees** who have **vacation hours greater than 90 or sick hours less than 30**. The example in Figure 5 would be incorrectly written since the result set returns females in addition to males.

SQLQuery1.sql - (L...ner-PC\Owner (54))*

```

SELECT FirstName,
       LastName,
       Gender,
       HireDate,
       Title,
       Salary
FROM Employees
WHERE VacationHours >= 90
OR SickLeaveHours < 30
AND Gender = 'M'
ORDER BY VacationHours, SickLeaveHours

```

100 %

Results Messages

	FirstName	LastName	Gender	HireDate	Title	Salary
1	Roberto	Tamburello	M	1997-12-12	Engineering Manager	44999.97
2	Michael	Sullivan	M	2001-01-30	Senior Design Engineer	41437.51
3	Joself	Goldberg	M	1998-02-24	Design Engineer	37569.99
4	Thierry	D'Nofro	M	1998-01-11	Tool Designer	28730.00
5	Lori	Penor	F	2000-03-19	Janitor	16161.60
6	Jo	Bery	F	2000-04-07	Janitor	16161.60
7	Christian	Kleineman	M	1999-01-15	Maintenance Supervisor	30345.01
8	Brian	Welcker	M	2001-03-18	Vice President of Sales	107100.02
9	Pilar	Ackeman	M	1999-02-03	Shipping and Receiving Supervisor	28560.04
10	Matthias	Bemdt	M	1999-02-21	Shipping and Receiving Clerk	105000.06
11	Vamsi	Kuppa	M	1999-01-08	Shipping and Receiving Clerk	14108.64
12	Jimmy	Bischoff	M	1999-03-30	Stocker	13366.08
13	Kim	Ralls	F	1999-01-27	Stocker	15724.80
14	Willis	Johnson	M	1999-01-14	Recruiter	32299.95

Figure 5

To retrieve the correct result, set we need to consider the same order of operations used in math. Parenthesis are required around the comparison of vacation and sick hours to show that should be checked first and then gender. Otherwise, the SELECT statement in Figure 5 will be interpreted by the database as follows:

► Where vacation hours greater than 90 or sick leave hours < 30 and gender = 'M'

To fix the Select statement we need to make sure the database interprets the statement as follows:

► Where vacation hours greater than (90 or sick leave hours < 30) and gender = 'M'

Figure 6 below shows the correct version of the SELECT statement along with the correct result set.

SQLQuery1.sql - (L...ner-PC\Owner (54))*

```

SELECT FirstName,
       LastName,
       Gender,
       HireDate,
       Title,
       Salary
FROM Employees
WHERE (VacationHours >= 90 OR SickLeaveHours < 30)
AND Gender = 'M'
ORDER BY VacationHours, SickLeaveHours

```

100 %

Results Messages

	FirstName	LastName	Gender	HireDate	Title	Salary
1	Roberto	Tamburello	M	1997-12-12	Engineering Manager	44999.97
2	Michael	Sullivan	M	2001-01-30	Senior Design Engineer	41437.51
3	Jossef	Goldberg	M	1998-02-24	Design Engineer	37569.99
4	Thierry	D'Nofro	M	1998-01-11	Tool Designer	28730.00
5	Christian	Kleinerman	M	1999-01-15	Maintenance Supervisor	30345.01
6	Brian	Welcker	M	2001-03-18	Vice President of Sales	107100.02
7	Pilar	Ackerman	M	1999-02-03	Shipping and Receiving Supervisor	28560.04
8	Matthias	Bemdt	M	1999-02-21	Shipping and Receiving Clerk	105000.06
9	Vamsi	Kuppa	M	1999-01-08	Shipping and Receiving Clerk	14108.64
10	Jimmy	Bischoff	M	1999-03-30	Stocker	13366.08
11	Willis	Johnson	M	1999-01-14	Recruiter	32299.95

Figure 6

NOT Operator

The NOT operator can be used to help reduce the size of a SELECT statement. To understand this concept let's look at Figure 7 which has a result set displaying all the rows in the Departments table. If we want to display the DepartmentName and GroupName from the table for all rows except those containing "Executive General and Administration" we could write it "the long way" using OR operators as shown in the query in Figure 8.

DepartmentID	Name	GroupName	ModifiedDate
1	Engineering	Research and Development	1998-06-01 00:00:00.000
2	Tool Design	Research and Development	1998-06-01 00:00:00.000
3	Sales	Sales and Marketing	1998-06-01 00:00:00.000
4	Marketing	Sales and Marketing	1998-06-01 00:00:00.000
5	Purchasing	Inventory Management	1998-06-01 00:00:00.000
6	Research and Development	Research and Development	1998-06-01 00:00:00.000
7	Production	Manufacturing	1998-06-01 00:00:00.000
8	Production Control	Manufacturing	1998-06-01 00:00:00.000
9	Human Resources	Executive General and Administration	1998-06-01 00:00:00.000
10	Finance	Executive General and Administration	1998-06-01 00:00:00.000
11	Information Services	Executive General and Administration	1998-06-01 00:00:00.000
12	Document Control	Quality Assurance	1998-06-01 00:00:00.000
13	Quality Assurance	Quality Assurance	1998-06-01 00:00:00.000
14	Facilities and Maintenance	Executive General and Administration	1998-06-01 00:00:00.000
15	Shipping and Receiving	Inventory Management	1998-06-01 00:00:00.000
16	Executive	Executive General and Administration	1998-06-01 00:00:00.000

Figure 7

```

SELECT GroupName,
       DepartmentName
FROM Departments
WHERE GroupName = 'Inventory Management'
OR     GroupName = 'Manufacturing'
OR     GroupName = 'Research and Development'
OR     GroupName = 'Sales and Marketing'
ORDER BY GroupName

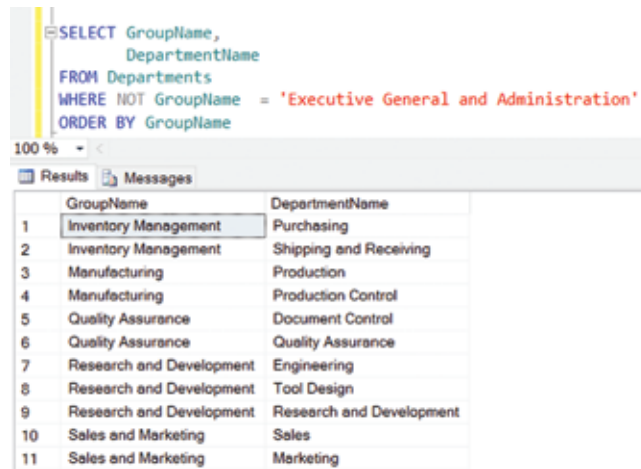
```

GroupName	DepartmentName
Inventory Management	Purchasing
Inventory Management	Shipping and Receiving
Manufacturing	Production
Manufacturing	Production Control
Research and Development	Research and Development
Research and Development	Engineering
Research and Development	Tool Design
Sales and Marketing	Sales
Sales and Marketing	Marketing

Figure 8

Although the results are what we desire, the abundance of OR operators are both time consuming to write and confusing to read.

An easier way to write the query and avoiding the abundance of OR operators is to use the NOT operator as shown in Figure 9. In this example, the query uses the WHERE NOT clause to instruct the database to NOT include the group name “Executive General and Administration.”



```

SELECT GroupName,
       DepartmentName
FROM Departments
WHERE NOT GroupName = 'Executive General and Administration'
ORDER BY GroupName

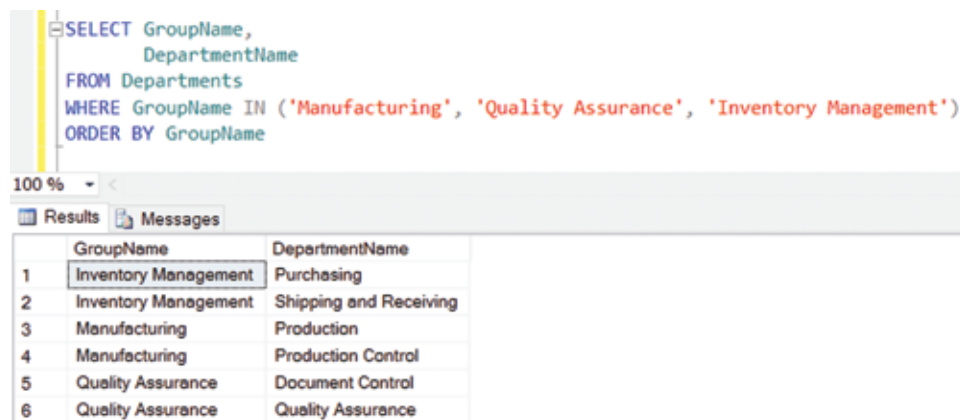
```

	GroupName	DepartmentName
1	Inventory Management	Purchasing
2	Inventory Management	Shipping and Receiving
3	Manufacturing	Production
4	Manufacturing	Production Control
5	Quality Assurance	Document Control
6	Quality Assurance	Quality Assurance
7	Research and Development	Engineering
8	Research and Development	Tool Design
9	Research and Development	Research and Development
10	Sales and Marketing	Sales
11	Sales and Marketing	Marketing

Figure 9

The IN and NOT IN Operator

Another useful technique for reducing the number of OR operators is to use the IN operator and NOT IN operator. These operators allow the comparison of a column against several values without the use of an OR operator. Figure 10 shows an example of using the IN operator to return rows with either Manufacturing, Quality Assurance or Inventory Management as the GroupName from the Departments table.



```

SELECT GroupName,
       DepartmentName
FROM Departments
WHERE GroupName IN ('Manufacturing', 'Quality Assurance', 'Inventory Management')
ORDER BY GroupName

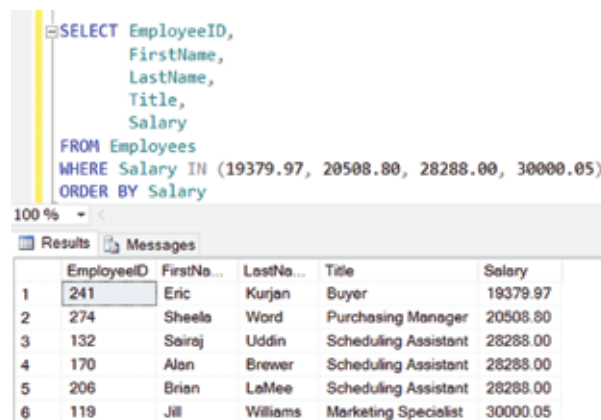
```

	GroupName	DepartmentName
1	Inventory Management	Purchasing
2	Inventory Management	Shipping and Receiving
3	Manufacturing	Production
4	Manufacturing	Production Control
5	Quality Assurance	Document Control
6	Quality Assurance	Quality Assurance

Figure 10

Take a close look at how the IN operator is formatted. The values need to be in parenthesis and separated by commas. Since this comparison is for character data the values are contained within single quotes. If we were to execute a comparison with the numeric data type the format would be slightly different by removing the single quotes.

For an example of using the IN operator with numeric data, Figure 11 shows a query based on the Employees table for salaries equaling 19379.97, 20,508.80, 28,288.00, or 30,000.05. Like the example in Figure 10, the values are placed within parenthesis and separated by commas. However, because numeric data is being compared, single quotes are not used.



```

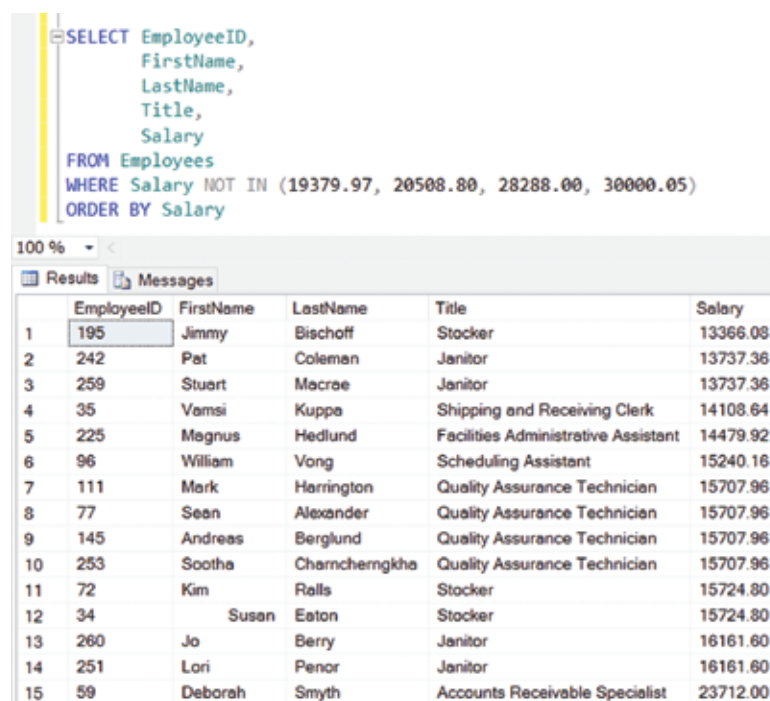
SELECT EmployeeID,
       FirstName,
       LastName,
       Title,
       Salary
FROM Employees
WHERE Salary IN (19379.97, 20508.80, 28288.00, 30000.05)
ORDER BY Salary

```

	EmployeeID	FirstNa...	LastNa...	Title	Salary
1	241	Eric	Kurjan	Buyer	19379.97
2	274	Sheela	Word	Purchasing Manager	20508.80
3	132	Sairaj	Uddin	Scheduling Assistant	28288.00
4	170	Alan	Brewer	Scheduling Assistant	28288.00
5	206	Brian	LeMee	Scheduling Assistant	28288.00
6	119	Jill	Williams	Marketing Specialist	30000.05

Figure 11

The NOT IN operator can be used the same way, except for finding rows that don't contain the values in the parenthesis. If we rewrite the query in Figure 11 to contain the NOT IN operator rather than the IN operator the results will show all salaries except the ones in the parenthesis. See Figure 12 for this example. Note: Be aware that not all rows are displayed in the example as it would be too large.



```

SELECT EmployeeID,
       FirstName,
       LastName,
       Title,
       Salary
FROM Employees
WHERE Salary NOT IN (19379.97, 20508.80, 28288.00, 30000.05)
ORDER BY Salary

```

	EmployeeID	FirstName	LastName	Title	Salary
1	195	Jimmy	Bischoff	Stockier	13366.08
2	242	Pat	Coleman	Janitor	13737.36
3	259	Stuart	Macree	Janitor	13737.36
4	35	Vamsi	Kuppa	Shipping and Receiving Clerk	14108.64
5	225	Magnus	Hedlund	Facilities Administrative Assistant	14479.92
6	96	William	Vong	Scheduling Assistant	15240.16
7	111	Mark	Harrington	Quality Assurance Technician	15707.96
8	77	Sean	Alexander	Quality Assurance Technician	15707.96
9	145	Andreas	Berglund	Quality Assurance Technician	15707.96
10	253	Sootha	Charncherngkha	Quality Assurance Technician	15707.96
11	72	Kim	Rolls	Stockier	15724.80
12	34	Susan	Eaton	Stockier	15724.80
13	260	Jo	Berry	Janitor	16161.60
14	251	Lori	Penor	Janitor	16161.60
15	59	Deborah	Smyth	Accounts Receivable Specialist	23712.00

Figure 12

The BETWEEN Operator

The BETWEEN operator provides the functionality to compare a range of values. It is an inclusive comparison as it includes both the low and high values in the comparison. When there are many values to compare, the BETWEEN operator becomes quite useful.

Figure 13 shows an example of using the BETWEEN operator to retrieve all the rows between and including the values 19,379.97 and 26,519.97.

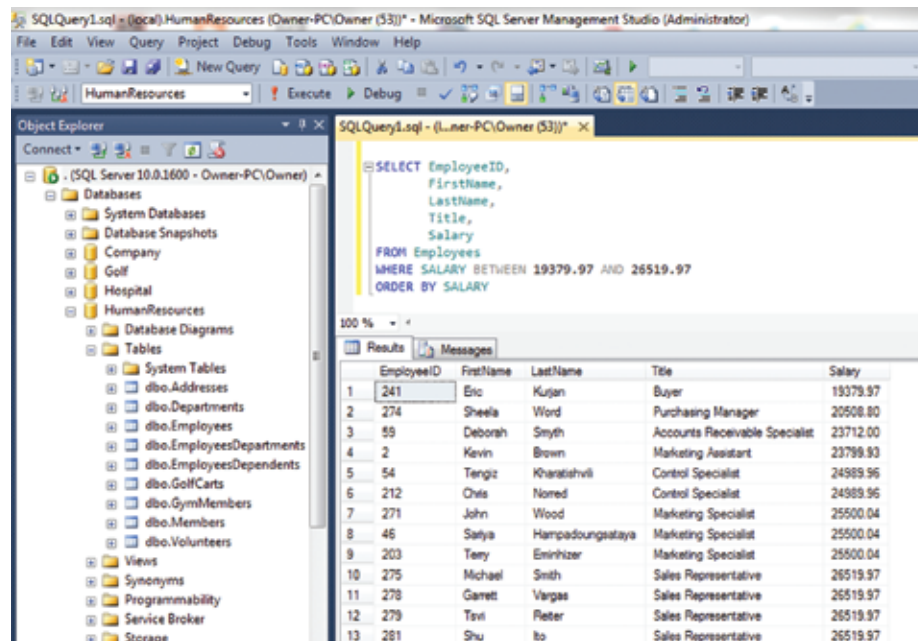


Figure 13

The query in Figure 13 can also be coded using LESS THAN, EQUAL TO, and GREATER THAN operators to retrieve the same results. Figure 14 shows how it can be done.

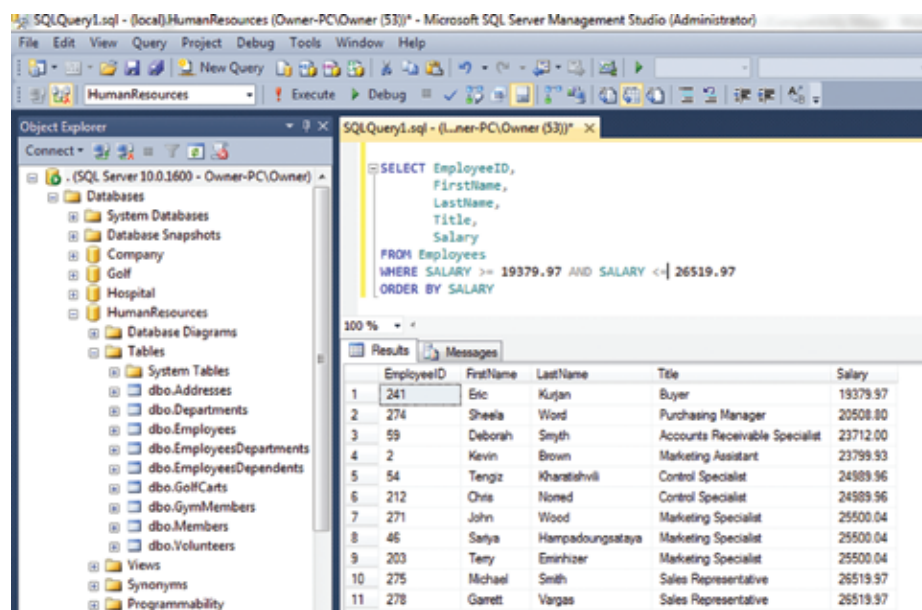
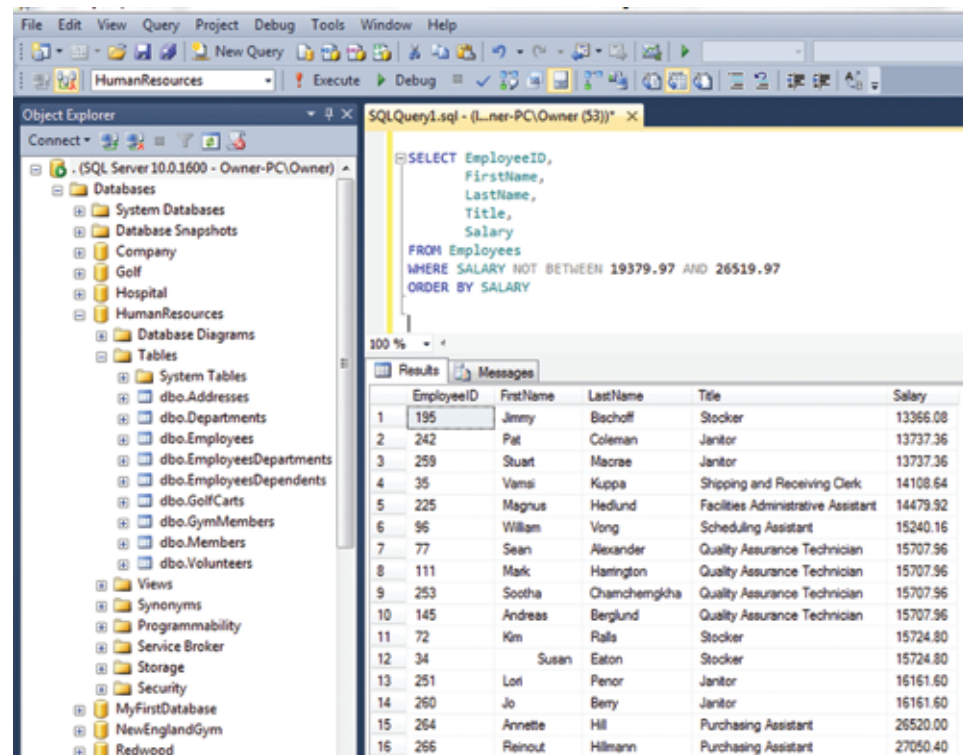


Figure 14

Although both queries retrieve the correct result set, using the BETWEEN operator makes it more user friendly to the reader and may help reduce the possibility of incorrect code.

We can use the NOT BETWEEN operator to find the values that are NOT within the certain values. For example, let's take the same query as shown in Figure 13 but add the NOT BETWEEN operator as shown in Figure 15.



The screenshot shows the SQL Server Enterprise Manager interface. The Object Explorer on the left displays the database structure for 'HumanResources'. The central pane shows a SQL query in 'SQLQuery1.sql':

```
SELECT EmployeeID,
       FirstName,
       LastName,
       Title,
       Salary
FROM Employees
WHERE SALARY NOT BETWEEN 19379.97 AND 26519.97
ORDER BY SALARY
```

The Results pane at the bottom displays a table with 16 rows of employee data. The columns are EmployeeID, FirstName, LastName, Title, and Salary. The salary values range from 13366.08 to 27050.40, with the two lowest values (13366.08 and 13737.36) being the only ones included in the result set, as they are below the specified range.

EmployeeID	FirstName	LastName	Title	Salary
1	Jimmy	Bischoff	Stocker	13366.08
2	Pat	Coleman	Janitor	13737.36
3	Stuart	Macrae	Janitor	13737.36
4	Vamsi	Kuppa	Shipping and Receiving Clerk	14108.64
5	Magnus	Hedlund	Facilities Administrative Assistant	14479.92
6	William	Vong	Scheduling Assistant	15240.16
7	Sean	Alexander	Quality Assurance Technician	15707.96
8	Mark	Harrington	Quality Assurance Technician	15707.96
9	Sootha	Chamchemkha	Quality Assurance Technician	15707.96
10	Andreas	Berglund	Quality Assurance Technician	15707.96
11	Kim	Ralls	Stocker	15724.80
12	Susan	Eaton	Stocker	15724.80
13	Lori	Penor	Janitor	16161.60
14	Jo	Berry	Janitor	16161.60
15	Annette	Hill	Purchasing Assistant	26520.00
16	Reinout	Hillmann	Purchasing Assistant	27050.40

Figure 15

Figure 15 shows only a small sample of the result set but even so you can see that any values between 19,379.97 and 26,519.97 are not included.

The NOT BETWEEN operator is useful when you need to exclude a range of values. For example, if you want to find values that are either on the low end or high end and exclude middle values, the NOT BETWEEN operator can be useful.

LIKE and NOT LIKE OPERATOR

The LIKE operator and NOT LIKE operator can be used to search for data rows containing incomplete or partial character strings within a data column. This is especially helpful when a search is required for a set of characters but the correct spelling is unknown.

The LIKE operator and NOT LIKE operator are used in conjunction with the percent sign. The percent sign acts as a wildcard for the unknown characters. For example, if we search for “Ka%”, all names

that begin with a “Ka” will be returned regardless of the characters that follow because the percent sign is acting as a wildcard. The following subset would meet the criteria:

- Kathy
- Katrina
- Karey
- Kasey

Figure 16 shows a query that is looking for employees who have the first two letters of their last names beginning with “Br.” Any employee that matches the criteria will show up no matter what letters follow “Br.”

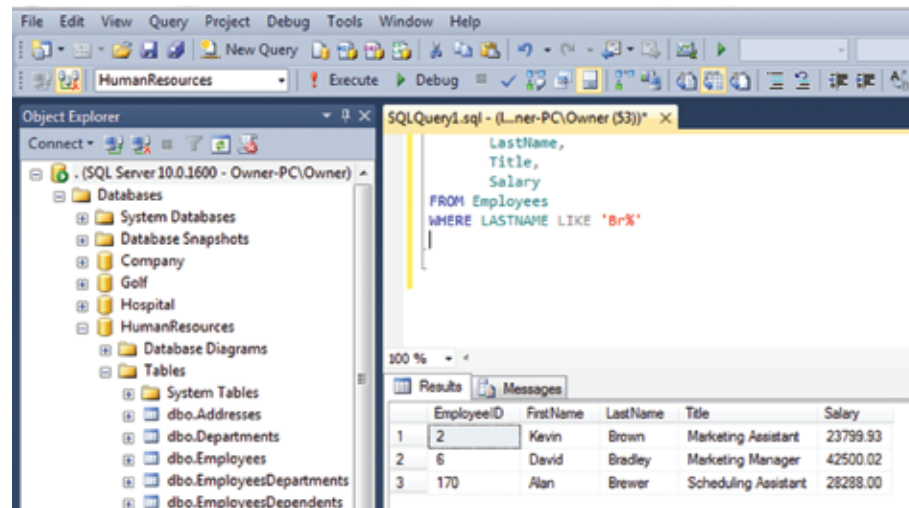


Figure 16

The example in Figure 17 helps demonstrate the flexibility of the LIKE operator. We can find any last name that has an “r” somewhere in it by placing the percent sign before and after the letter “r.”

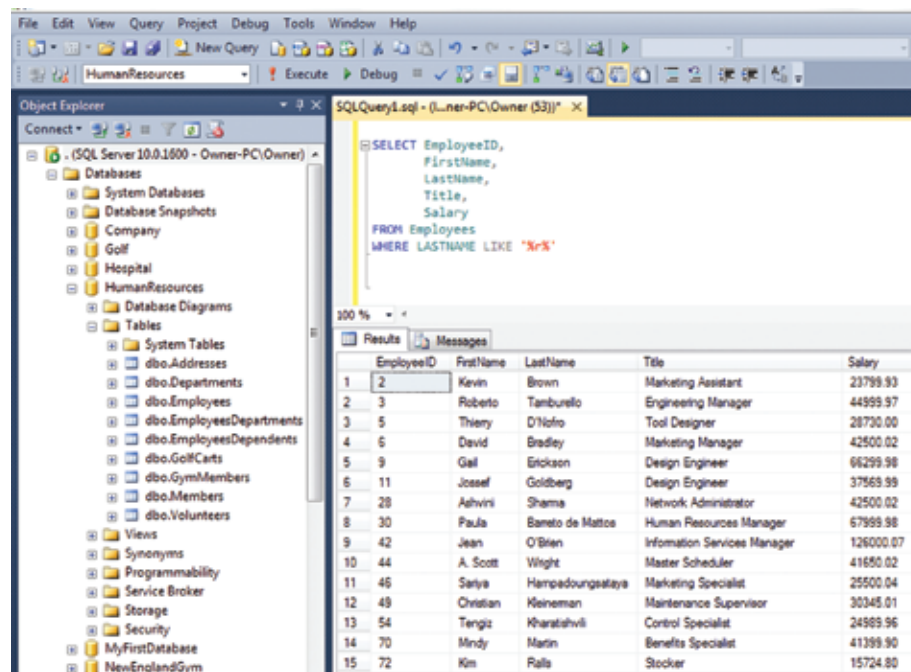


Figure 17

Figure 18 shows an example of looking for last names that contain “man” as the last three letters in it.

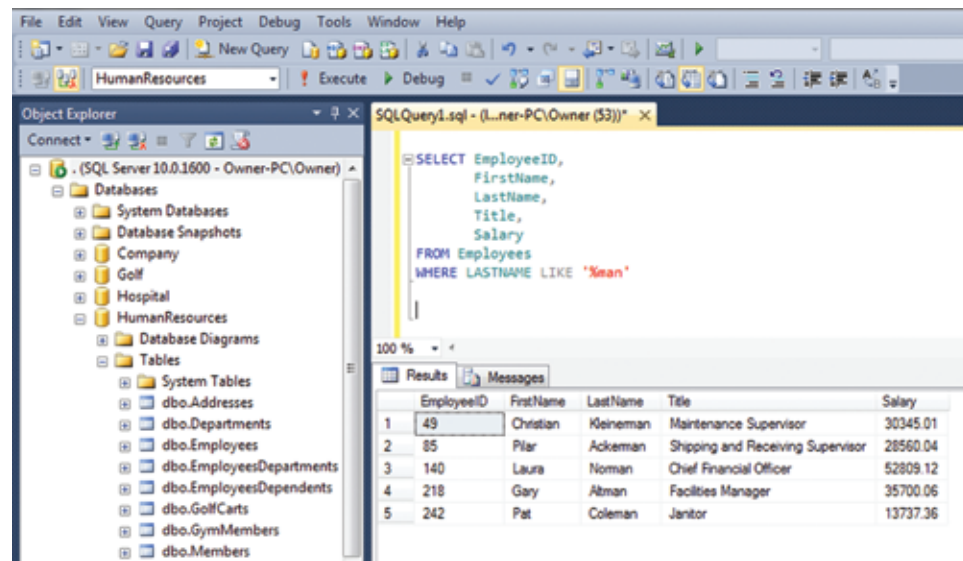


Figure 18

Figure 19 takes another approach at using the wildcard by finding employees that have the first two letters of their last name beginning with “Ha” and the last two letters of their last name ending with “on.” The wildcard is placed in the middle so anything in between will meet the criteria.

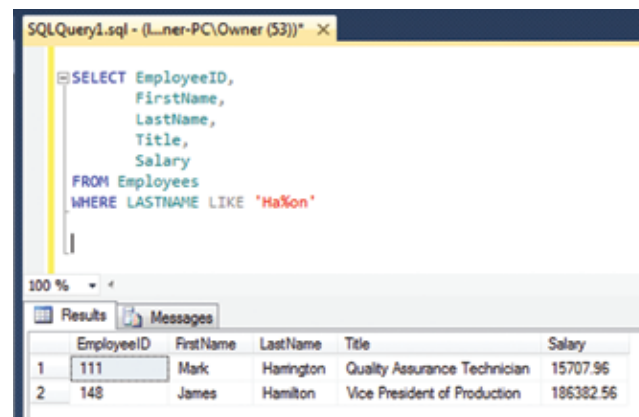
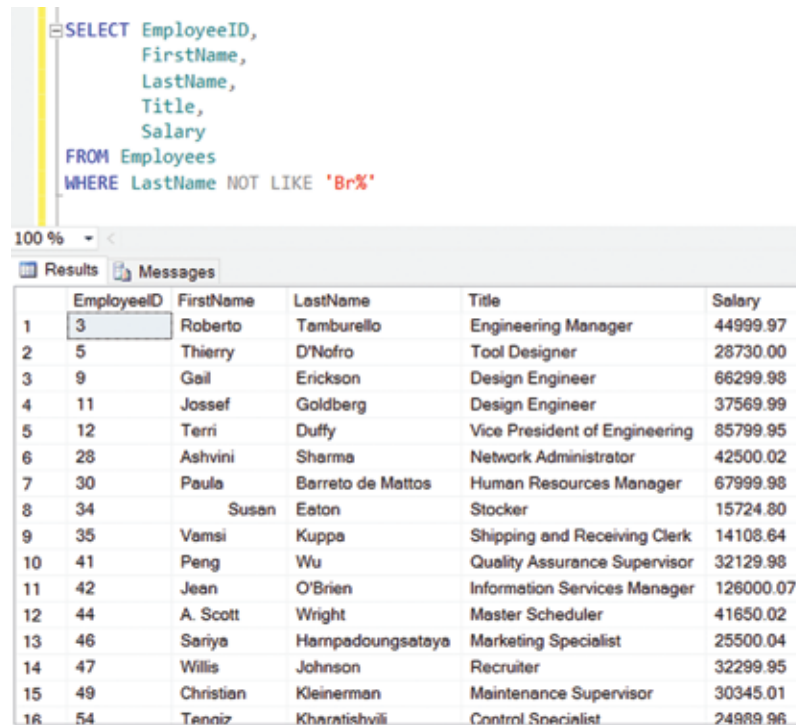


Figure 19

While all of the examples thus far find employees with a wildcard criteria, we are able to do the opposite by finding employees that don't meet the criteria using the NOT operator in conjunction with the LIKE operator. The example in Figure 20 returns all employees who DO NOT have the first two letters of their last name beginning with "Br."



```

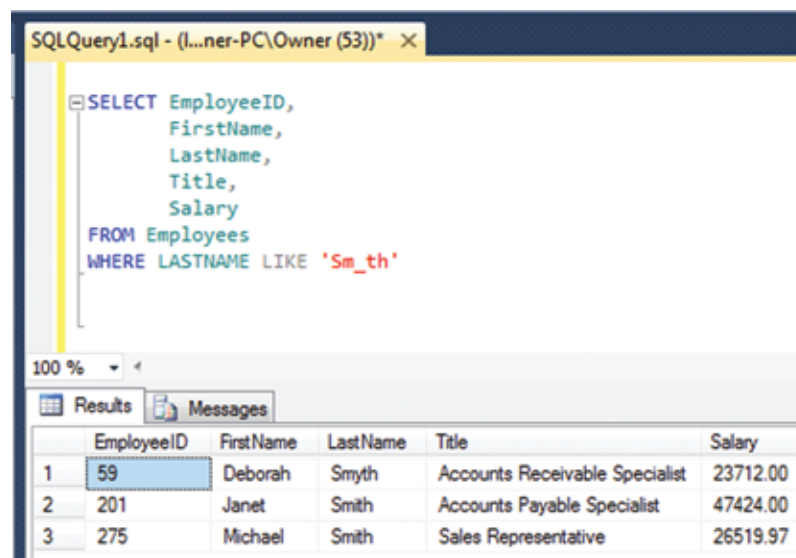
SELECT EmployeeID,
       FirstName,
       LastName,
       Title,
       Salary
FROM Employees
WHERE LastName NOT LIKE 'Br%'

```

	EmployeeID	FirstName	LastName	Title	Salary
1	3	Roberto	Tamburello	Engineering Manager	44999.97
2	5	Thierry	D'Nofro	Tool Designer	28730.00
3	9	Gail	Erickson	Design Engineer	66299.98
4	11	Josief	Goldberg	Design Engineer	37569.99
5	12	Terri	Duffy	Vice President of Engineering	85799.95
6	28	Ashvini	Sharma	Network Administrator	42500.02
7	30	Paula	Barreto de Mattos	Human Resources Manager	67999.98
8	34	Susan	Eaton	Stocker	15724.80
9	35	Vamsi	Kuppa	Shipping and Receiving Clerk	14108.64
10	41	Peng	Wu	Quality Assurance Supervisor	32129.98
11	42	Jean	O'Brien	Information Services Manager	126000.07
12	44	A. Scott	Wright	Master Scheduler	41650.02
13	46	Sariya	Harnpadoungsetaya	Marketing Specialist	25500.04
14	47	Willis	Johnson	Recruiter	32299.95
15	49	Christian	Kleinerman	Maintenance Supervisor	30345.01
16	54	Tenniz	Kharatishvili	Control Specialist	24989.96

Figure 20

The underscore can be used as a wildcard in lieu of an individual letter. You can also use multiple underscores to substitutes as wildcards for multiple letters. Figure 21 shows a query where all last names that are no longer than five characters beginning with "Sm" and ending with "th" are found. The middle character is replaced by the underscore wildcard. As a result both "Smith" and "Smyth" are part of the result set.



```

SELECT EmployeeID,
       FirstName,
       LastName,
       Title,
       Salary
FROM Employees
WHERE LASTNAME LIKE 'Sm_th'

```

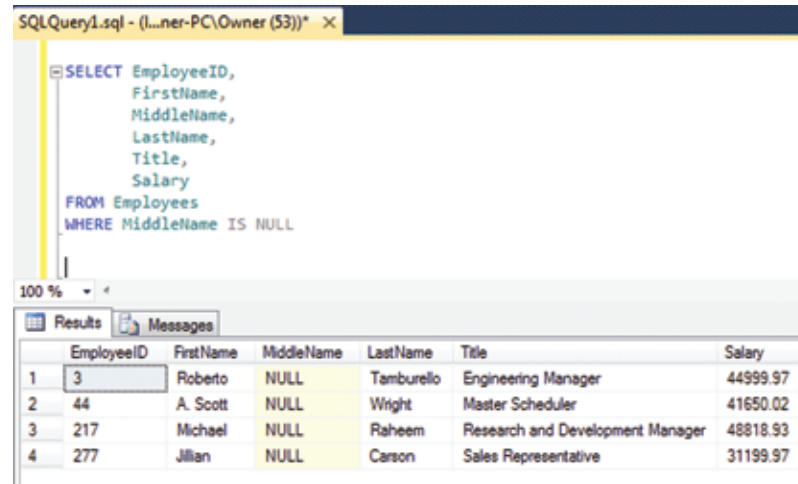
	EmployeeID	FirstName	LastName	Title	Salary
1	59	Deborah	Smyth	Accounts Receivable Specialist	23712.00
2	201	Janet	Smith	Accounts Payable Specialist	47424.00
3	275	Michael	Smith	Sales Representative	26519.97

Figure 21

NULL and NOT NULL Operators

NULL value is not the same as “zero” (numerical values) or “blank” (character values). It is really nothing. No characters exist. NULL values allow users to distinguish between a deliberate entry of zero/blank and a non-entry of data.

To retrieve rows where NULL values exist, the IS NULL operator is used. Figure 22 shows an example of a query that is written to retrieve rows where middle name has no entry.



The screenshot shows a SQL Query window titled "SQLQuery1.sql - (L...ner-PC\Owner (53))". The query is as follows:

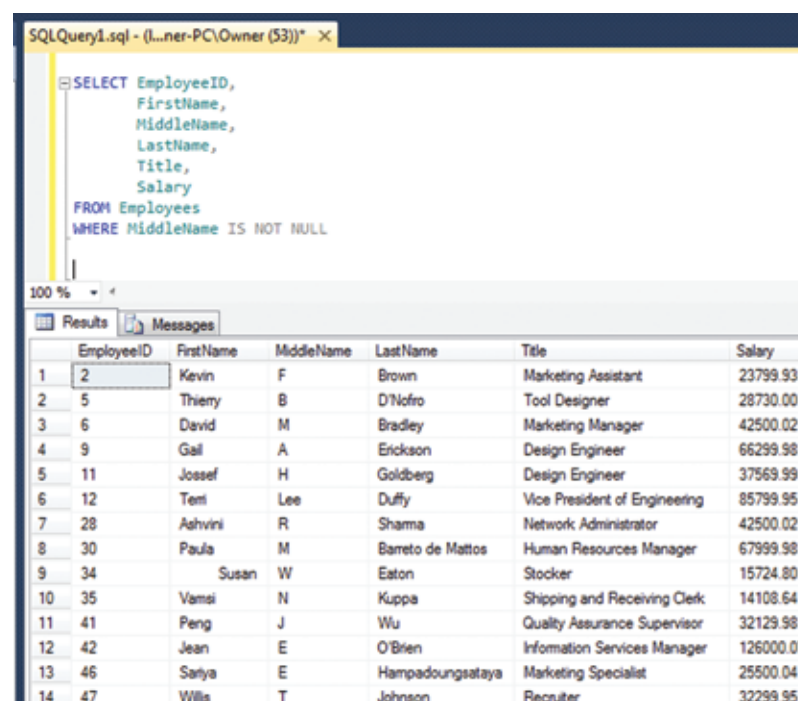
```
SELECT EmployeeID,
       FirstName,
       MiddleName,
       LastName,
       Title,
       Salary
FROM Employees
WHERE MiddleName IS NULL
```

The Results tab shows the following data:

	EmployeeID	FirstName	MiddleName	LastName	Title	Salary
1	3	Roberto	NULL	Tamburello	Engineering Manager	44999.97
2	44	A. Scott	NULL	Wright	Master Scheduler	41650.02
3	217	Michael	NULL	Raheem	Research and Development Manager	48818.93
4	277	Jillian	NULL	Carson	Sales Representative	31199.97

Figure 22

Figure 23 does the opposite of that in Figure 22 by retrieving rows where middle name contains a value. It does this by using the NOT operator along with IS NULL which together form “IS NOT NULL.”



The screenshot shows a SQL Query window titled "SQLQuery1.sql - (L...ner-PC\Owner (53))". The query is as follows:

```
SELECT EmployeeID,
       FirstName,
       MiddleName,
       LastName,
       Title,
       Salary
FROM Employees
WHERE MiddleName IS NOT NULL
```

The Results tab shows the following data:

	EmployeeID	FirstName	MiddleName	LastName	Title	Salary
1	2	Kevin	F	Brown	Marketing Assistant	23799.93
2	5	Thierry	B	D'Nofro	Tool Designer	28730.00
3	6	David	M	Bradley	Marketing Manager	42500.02
4	9	Gail	A	Erickson	Design Engineer	66299.98
5	11	Jossef	H	Goldberg	Design Engineer	37569.99
6	12	Terri	Lee	Duffy	Vice President of Engineering	85799.95
7	28	Ashvini	R	Sharma	Network Administrator	42500.02
8	30	Paula	M	Barreto de Mattos	Human Resources Manager	67999.98
9	34	Susan	W	Eaton	Stockier	15724.80
10	35	Vamsi	N	Kuppa	Shipping and Receiving Clerk	14108.64
11	41	Peng	J	Wu	Quality Assurance Supervisor	32129.98
12	42	Jean	E	O'Brien	Information Services Manager	126000.07
13	46	Sariya	E	Hampadourasataya	Marketing Specialist	25500.04
14	47	Willis	T	Johnson	Recruiter	32299.95

Figure 23

To further demonstrate that NULL is truly nothing, Figure 24 shows that NULL values cannot be compared to anything, including spaces. Figure 24 shows a query using two quotes in the WHERE clause, one after the other with nothing, not even a space in between. No results are returned. That's because a blank space is not a NULL and still considered a value and no one in the database has a middle name consisting of a blank value. The data contained within a database is either NOT NULL or NULL.

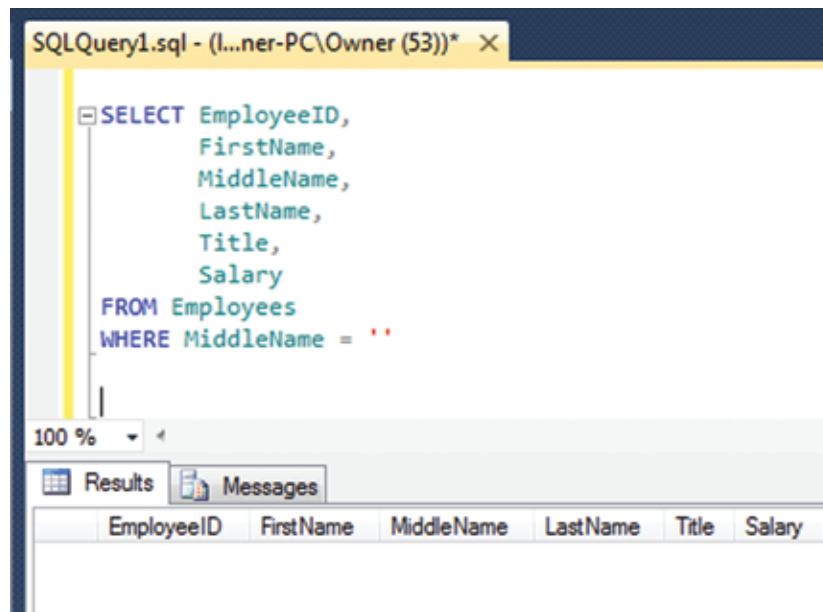


Figure 24

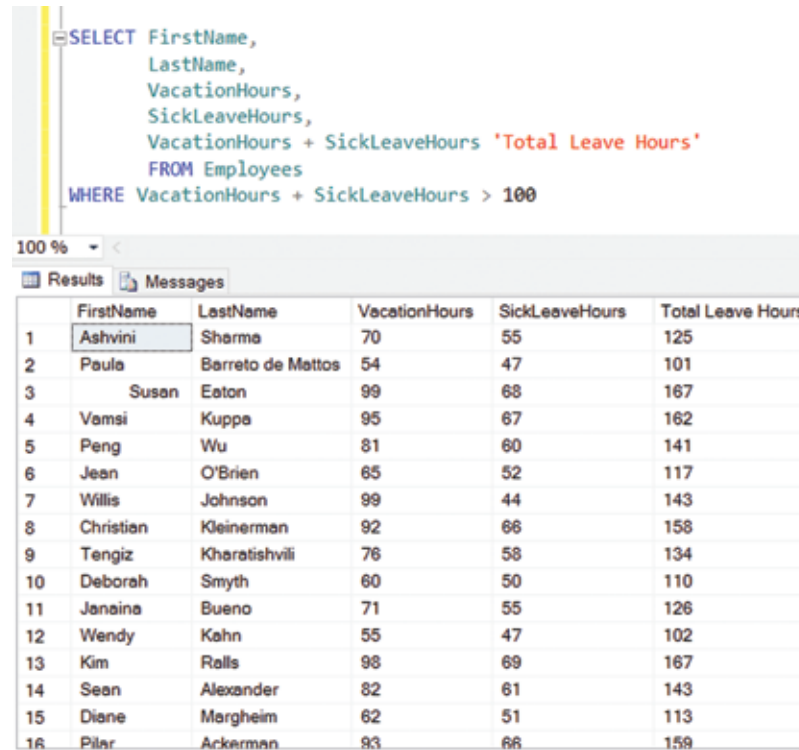
Expressions

An expression is formed by combining a column name or constant with an arithmetic operator.

The arithmetic operators used in SQL are shown in the table below.

Symbol	Operation	Order
*	Multiplication	1
/	Division	1
%	Modulo (remainder)	1
+	Addition	2
-	Subtraction	2

Expressions can be used in the WHERE clause so that manipulated values can be compared. For example, to find employees who have vacation days and sick days totaling to more than 100 days off can be written as shown in Figure 25.



```

SELECT FirstName,
       LastName,
       VacationHours,
       SickLeaveHours,
       VacationHours + SickLeaveHours 'Total Leave Hours'
FROM Employees
WHERE VacationHours + SickLeaveHours > 100

```

	FirstName	LastName	VacationHours	SickLeaveHours	Total Leave Hours
1	Ashvini	Sharma	70	55	125
2	Paula	Berreto de Mattos	54	47	101
3	Susan	Eaton	99	68	167
4	Vamsi	Kuppa	95	67	162
5	Peng	Wu	81	60	141
6	Jean	O'Brien	65	52	117
7	Willis	Johnson	99	44	143
8	Christian	Kleinerman	92	66	158
9	Tengiz	Kharatishvili	76	58	134
10	Deborah	Smyth	60	50	110
11	Jenaina	Bueno	71	55	126
12	Wendy	Kohn	55	47	102
13	Kim	Rolls	98	69	167
14	Sean	Alexander	82	61	143
15	Diane	Margheim	62	51	113
16	Pilar	Ackerman	83	66	149

Figure 25

To help understand the query I've included the columns for vacation hours and sick leave hours. In addition, I included the expression of adding vacation hours and sick leave hours in the SELECT portion of the statement and providing an alias name of "Total Leave Hours." Only employees who have total leave hours that exceed 100 hours will end up in the result set.

Order of Precedence

The standard math order of precedence applies to SQL. For example, multiplication and division operations are executed before addition and subtraction. When operators are equal the order is left to right. Also, any operations in parenthesis occur first.

Figure 26 shows an example of a query that retrieves rows where the number of weeks for both vacation hours and sick hours exceeds 2 weeks. This is calculated by adding vacation hours and sick leave hours together and then dividing by 40. The example result set in Figure 26, however, is incorrect

because the parenthesis is missing around the addition of the vacation hours and sick hours to force that to happen before the division occurs. Instead, the division occurs first against the sick hours resulting in an incorrect result set.

```

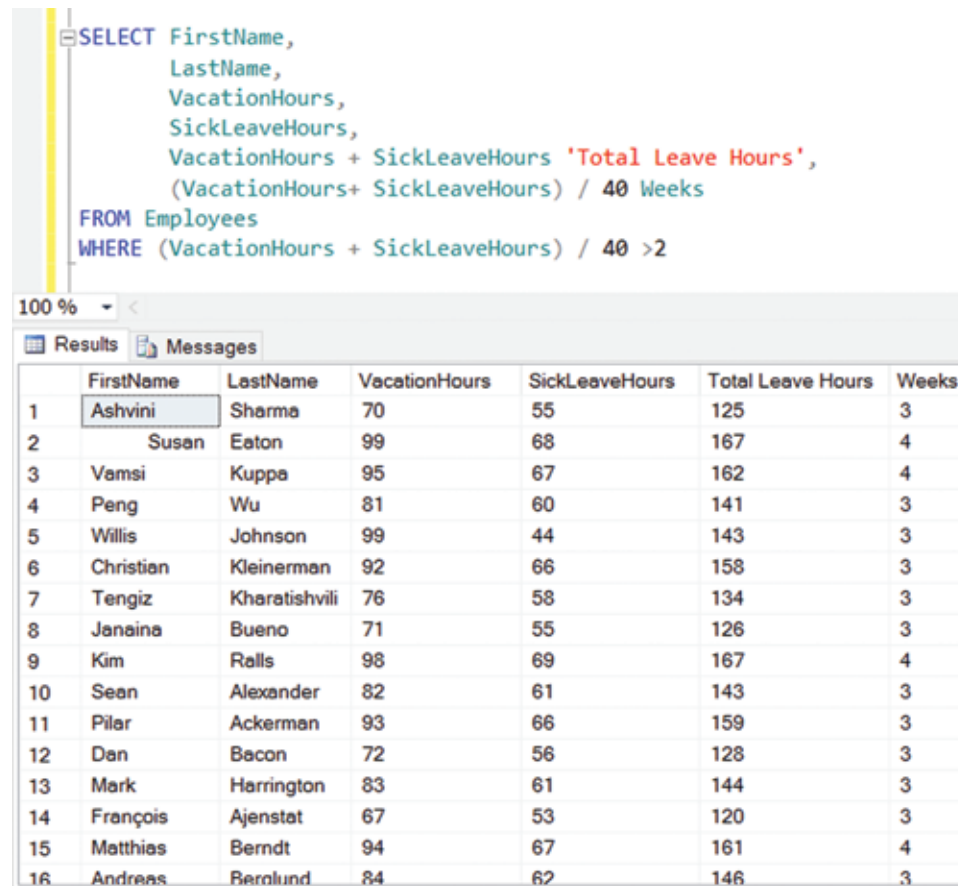
SELECT FirstName,
       LastName,
       VacationHours,
       SickLeaveHours,
       VacationHours + SickLeaveHours 'Total Leave Hours',
       VacationHours+ SickLeaveHours / 40 Weeks
FROM Employees
WHERE VacationHours + SickLeaveHours / 40 >2

```

	FirstName	LastName	VacationHours	SickLeaveHours	Total Leave Hours	Weeks
1	Kevin	Brown	42	41	83	43
2	Thierry	D'Nofro	9	24	33	9
3	David	Bradley	40	40	80	41
4	Gail	Erickson	5	22	27	5
5	Jossef	Goldberg	6	23	29	6
6	Ashvini	Sharma	70	55	125	71
7	Paula	Barreto de Mattos	54	47	101	55
8	Susan	Eaton	99	68	167	100
9	Vamsi	Kuppa	95	67	162	96
10	Peng	Wu	81	60	141	82
11	Jean	O'Brien	65	52	117	66
12	A. Scott	Wright	44	42	86	45
13	Sariya	Harnpadoungsataya	45	42	87	46
14	Willis	Johnson	99	44	143	100
15	Christian	Kleinerman	92	66	158	93
16	Tenoiz	Kharatishvili	76	58	134	77

Figure 26

To correct the example in Figure 26 we need to enclose the addition of vacation hours and sick hours within parenthesis as shown in Figure 27.



```

SELECT FirstName,
       LastName,
       VacationHours,
       SickLeaveHours,
       VacationHours + SickLeaveHours 'Total Leave Hours',
       (VacationHours+ SickLeaveHours) / 40 Weeks
FROM Employees
WHERE (VacationHours + SickLeaveHours) / 40 >2

```

	FirstName	LastName	VacationHours	SickLeaveHours	Total Leave Hours	Weeks
1	Ashvini	Sharma	70	55	125	3
2	Susan	Eaton	99	68	167	4
3	Vamsi	Kuppa	95	67	162	4
4	Peng	Wu	81	60	141	3
5	Willis	Johnson	99	44	143	3
6	Christian	Kleinerman	92	66	158	3
7	Tengiz	Kharatishvili	76	58	134	3
8	Janeina	Bueno	71	55	126	3
9	Kim	Rolls	98	69	167	4
10	Sean	Alexander	82	61	143	3
11	Pilar	Ackerman	93	66	159	3
12	Dan	Bacon	72	56	128	3
13	Mark	Harrington	83	61	144	3
14	François	Ajenstat	67	53	120	3
15	Matthias	Berndt	94	67	161	4
16	Andreas	Berolund	84	62	146	3

Figure 27

Summary

This chapter demonstrated how the use of various logical operators and expressions can be used to retrieve specific result sets making your queries more efficient and useful to end users. This chapter included learning about logical operators to provide criteria that help make appropriate selections using the WHERE clause. The IN, NOT IN, BETWEEN, NOT BETWEEN operators were covered to show how they can be used to empower query selection.

The understanding of NULL values was covered as well as how they are used in queries. It was explained that NULL values are in a class of their own and can only be compared by using the IS NULL and IS NOT NULL operators.

This chapter also covered the value of using expressions in SELECT and WHERE clauses and the rules of doing so.

Exercises

1. Select the first name, middle name, last name, and title from the Employees table of all male employees who are single and have the title “Buyer.”
2. Select the first name, middle name, last name, and title of all female employees who are married and have the title “Marketing Assistant” or “Marketing Specialist.”
3. Select the first name, middle name, last name, and title of all female employees who do NOT have the title “Sales Representative.”
4. Using the OR operator, retrieve the first name, middle name, last name, title, and gender of all male employees who have the title “Sales Representative” AND all females who are married and have the title “Sales Representative.”
5. Using the IN operator, retrieve the first name, last name, and title of all employees who have a title of “Design Engineer,” “Sales Representative,” “Marketing Specialist,” or “Network Manager.”
6. Using the IN operator, retrieve the first name, last name, title, and salary of all employees who have a salary of \$42,500.02, \$44,999.97, \$67,999.98, or \$105,000.06.
7. Using the IN operator, retrieve the first name, last name, title, and salary of all employees who have a salary that is NOT equal to \$42,500.02, \$44,999.97, \$67,999.98, or \$105,000.06
8. Using the BETWEEN operator, retrieve the first name, last name, title, and salary of all employees who have a salary between \$25,000 and \$44,300. Order by salary.
9. Retrieve the first name, last name, title, and salary of all employees who have a last name beginning with “Va.”
10. Retrieve the first name, last name, title and salary of all employees who have a last name starting with “B” and is seven characters long.
11. Retrieve the first name, last name, title, and salary of all employees who have a last name beginning or ending with “k”.
12. Retrieve the first name, last name, title, and salary of all employees who have a first name that does not contain “a” and “i.”
13. Retrieve the AddressLine1, City, State, Zip1, and Zip2 from the Addresses table where Zip 2 is NOT null.
14. Retrieve the first name, last name, title, and salary of all employees who have a combined total of vacation hours and sick hours that are equal to or less than 25. Include the total hours in the result set.
15. Retrieve the first name, last name, title and salary of all employees who have combined total of sick hours and vacation hours equaling less than 2 weeks’ worth. Include the total hours and number of weeks in the result set.

