

9

Subqueries

This chapter will cover the topic of Subqueries. Subqueries are queries within a query allowing one query to retrieve results based on the results of another query. It may sound confusing now, but after reading and working through the examples of this chapter you will understand the convenience of using subqueries for obtaining desired results.

CHAPTER OBJECTIVES

The objectives for this chapter are as follows:

- Understand the fundamentals of subqueries.
- Understand how to use the IN operator for subqueries.
- Understand how to use the ALL keyword and ANY keyword on subqueries.
- Understand the difference between correlated and noncorrelated subqueries.
- Understand how to use the EXISTS operator on subqueries.

Noncorrelated Subqueries and Comparison Operators

Noncorrelated Subqueries are very useful when a query is needed based on the results of another query. This is especially important when we don't know the values to be compared to. For example, say we wanted to list all employees who have salaries that match the highest salary found in the Employees table. We would first need to find the highest salary (186,382.56) and then write a query to find salaries equal to it as follows:

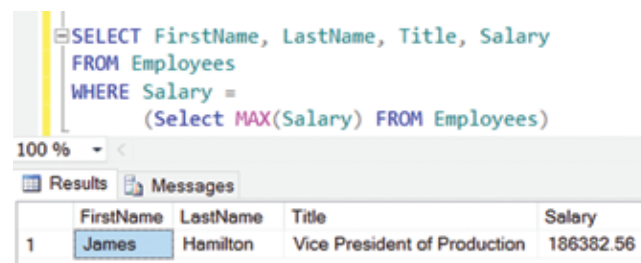
```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary = 186382.56
```

However, instead of taking the two-step approach of finding the highest salary and then finding all employees who have salaries that equal it, we could execute the following subquery instead.

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary =
    (Select MAX(Salary) FROM Employees)
```

Subqueries have outer and inner queries. The query that is highlighted in above is called the inner query and is always embedded inside parenthesis. The nonhighlighted query is called the outer query and it uses the WHERE clause to base its results on the results of the inner query.

The results in Figure 1 tell us that only one employee has the highest salary.



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

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary =
    (Select MAX(Salary) FROM Employees)
```

Below the query, there is a tab labeled "Results" which displays the following table:

	FirstName	LastName	Title	Salary
1	James	Hamilton	Vice President of Production	186382.56

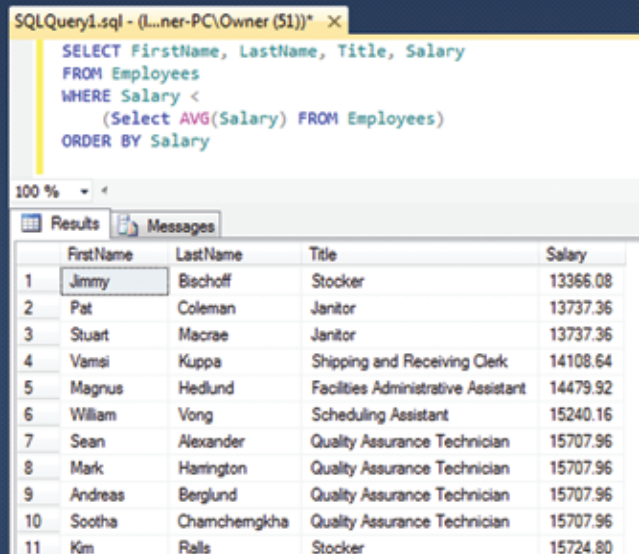
Figure 1

As another example, what if our manager wanted a list of all employees who were making less than the average pay. Let's first think about how we could retrieve the average salary. To retrieve the average of all salaries we would use the AVG function. The query written below will be our inner query. On its own it will simply give us the average salary.

```
Select AVG(Salary) FROM Employees
```

Executing the above statement returns a single value of 42268.3149. Next, we can write the outer query and compare the salary to the value the inner query produces as follows:

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary <
      (Select AVG(Salary) FROM Employees)
ORDER BY Salary
```



SQLQuery1.sql - (L...ner-PC\Owner (51))

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary <
      (Select AVG(Salary) FROM Employees)
ORDER BY Salary
```

100 %

Results Messages

	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	Andreas	Berglund	Quality Assurance Technician	15707.96
10	Sootha	Chamchemgkha	Quality Assurance Technician	15707.96
11	Kim	Ralls	Stocker	15724.80


Figure 2

Figure 2 shows a small subset of the results. There is a total of 66 employees who have salaries less than 42,268.3149.

Also, notice the ORDER BY clause in Figure 2. It pertains to the OUTER query. We can infer this because it is outside the parenthesis of the inner query.

Data Type Compatibility

You must keep in mind that like any comparisons performed in SQL, the return value of the inner join must have a compatible data type to the column being used in the WHERE clause of the outer query. For example, the query in Figure 3 below fails because the salary, which is numeric, is being compared to character data.



```
SELECT EmployeeID, FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary
      (Select MAX(FirstName) FROM Employees)
ORDER BY Salary
```

100 %

Messages

Msg 156, Level 15, State 1, Line 4
Incorrect syntax near the keyword 'Select'.

Msg 102, Level 15, State 1, Line 4
Incorrect syntax near ')'.

Figure 3

The error in Figure 3 occurs even though the inner query would work by itself. However, since it returns character data the comparison results in an error.

Comparing Multiple Return Values

In Chapter 5, we covered comparison operators and the WHERE clause in SELECT statements. This section will use comparisons with subqueries. If we look again at the very first example in this chapter, we can see that a single value is returned in the subquery. See the subquery below.

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary =
    (Select MAX(Salary) FROM Employee)
```

The inner query from the above example returns a single value. This is important to keep in mind because **unlike** using the IN operator, **comparison operators cannot be compared to more than one value**. Figure 4 is an example of an error that is returned when trying to execute a subquery that returns multiple values. The error message in red is clear about describing the problem.



Figure 4

If you just highlight and execute just the inner query within the parenthesis, the query executes successfully as shown in the results in Figure 5. But it returns three values so it won't work with the comparison on the outer query.

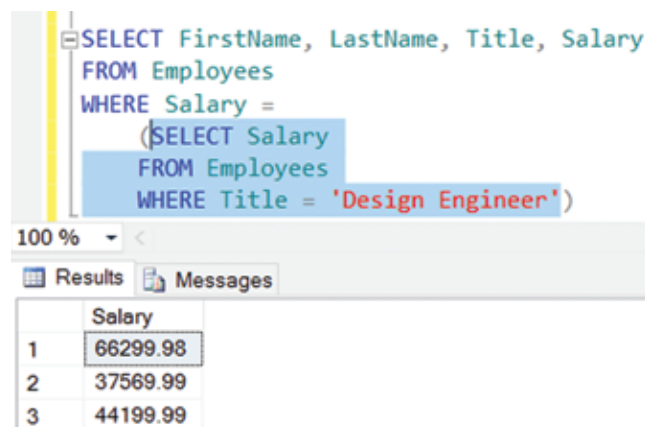


Figure 5

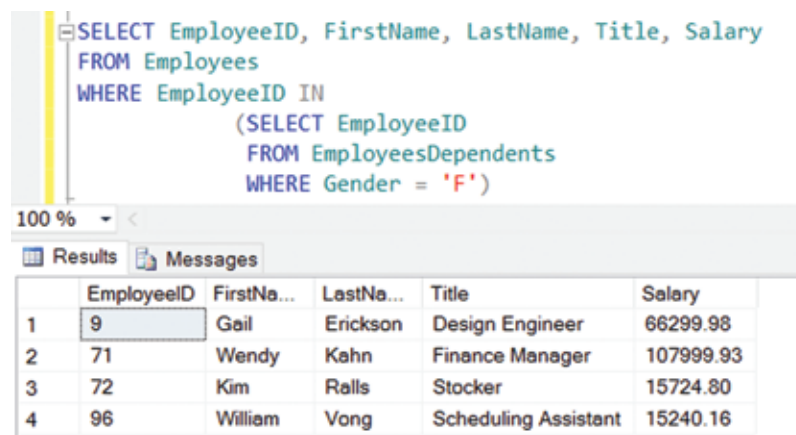
Subqueries and the IN Operator

We were introduced to the IN operator in Chapter 5, where we used hard coded values like the ones below to perform comparisons.

```
SELECT EmployeeID, FirstName, LastName, Title, Salary
FROM Employees
WHERE EmployeeID IN (9,72,117,289)
```

In the above query example, any employee who has an EmployeeID that falls into one of the values in the parenthesis will be returned in the result set. What would happen if we didn't know the values of the employee ID? For example, what if the manager wanted a list of employees who had female dependents? We could retrieve this information using the subquery as shown below. The results are shown in Figure 6.

```
SELECT EmployeeID, FirstName, LastName, Title, Salary
FROM Employees
WHERE EmployeeID IN (SELECT EmployeeID
                     FROM EmployeesDependents
                     WHERE Gender = 'F')
```



```
SELECT EmployeeID, FirstName, LastName, Title, Salary
FROM Employees
WHERE EmployeeID IN
    (SELECT EmployeeID
     FROM EmployeesDependents
     WHERE Gender = 'F')
```

	EmployeeID	FirstNa...	LastNa...	Title	Salary
1	9	Gail	Erickson	Design Engineer	66299.98
2	71	Wendy	Kahn	Finance Manager	107999.93
3	72	Kim	Rells	Stocker	15724.80
4	96	William	Vong	Scheduling Assistant	15240.16

Figure 6

Notice that the inner query uses the EmployeesDependents table to retrieve the EmployeeID of those employees who have female dependents. If we were to just run the inner query the results would be as shown in Figure 7.

```

SELECT EmployeeID, FirstName, LastName, Title, Salary
FROM Employees
WHERE EmployeeID IN
    (SELECT EmployeeID
     FROM EmployeesDependents
     WHERE Gender = 'F')
  
```

	EmployeeID
1	9
2	72
3	72
4	72
5	96
6	96
7	71
8	71
9	71

Figure 7

Figure 7 shows that duplicate values are returned. This is because Employees have more than one dependent as shown below in Figure 8.

EmployeeID	FirstName	MiddleName	LastName	Gender	BirthDate
9	Martha	S	Eaton	F	2010-06-07
72	Marcia	A	Ralls	F	1998-09-17
72	Barbara	C	Ralls	F	2001-11-12
72	Patricia	M	Ralls	F	2003-12-22
117	Heidi	NULL	Ajenstat	F	1997-10-07
117	Wendy	NULL	Ajenstat	F	1999-08-27
289	Sheila	M	Valdez	F	1997-08-22
289	Carla	C	Valdez	F	1998-12-02
289	Maria	N	Valdez	F	2000-08-01

Figure 8

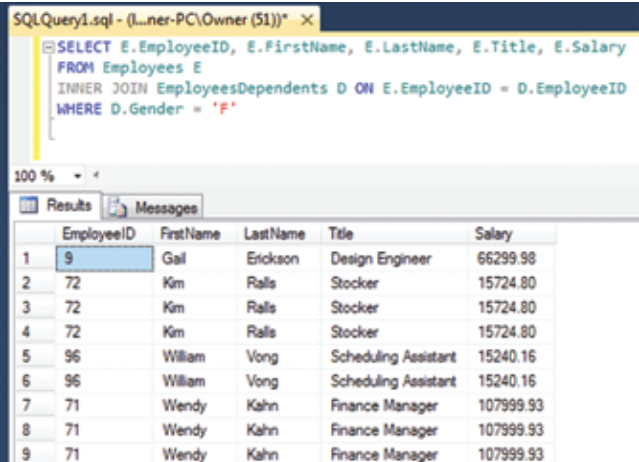
We could have used the DISTINCT keyword in the inner join to make sure only unique values were returned but it doesn't matter since all we're doing is selecting employees who have a matching EmployeeID that exists in the result set of the inner join. In that case, our employee information will not be duplicated since the employee only occurs once in the Employees table.

Using the JOIN Operator Instead

We can use the INNER JOIN below to retrieve the same results as shown in Figure 6.

```
SELECT DISTINCT E.EmployeeID, E.FirstName, E.LastName, E.Title, E.Salary
FROM Employees E
INNER JOIN EmployeesDependents D ON E.EmployeeID = D.EmployeeID
WHERE D.Gender = 'F'
```

The results would be identical but the DISTINCT clause is required here. If the DISTINCT clause isn't used duplicate rows would appear as shown in Figure 9. This is because it is JOINED with the EmployeesDependents table and if we included EmployeesDependents names each row would be unique. But since the query is only requesting Employees data it looks like we have several duplicate rows.



SQLQuery1.sql - (L:\ner-PC\Owner (51))

```
SELECT E.EmployeeID, E.FirstName, E.LastName, E.Title, E.Salary
FROM Employees E
INNER JOIN EmployeesDependents D ON E.EmployeeID = D.EmployeeID
WHERE D.Gender = 'F'
```

100 %

Results Messages

	EmployeeID	FirstName	LastName	Title	Salary
1	9	Gail	Erickson	Design Engineer	66299.98
2	72	Kim	Ralls	Stocker	15724.80
3	72	Kim	Ralls	Stocker	15724.80
4	72	Kim	Ralls	Stocker	15724.80
5	96	William	Vong	Scheduling Assistant	15240.16
6	96	William	Vong	Scheduling Assistant	15240.16
7	71	Wendy	Kahn	Finance Manager	107999.93
8	71	Wendy	Kahn	Finance Manager	107999.93
9	71	Wendy	Kahn	Finance Manager	107999.93

Figure 9

Our subquery didn't return duplicates because it is not being joined with the EmployeesDependents table. It is only using the EmployeesDependents table to retrieve EmployeeID values from the EmployeesDependents table and then listing each employee who has a matching EmployeeID to that found in the inner query results.

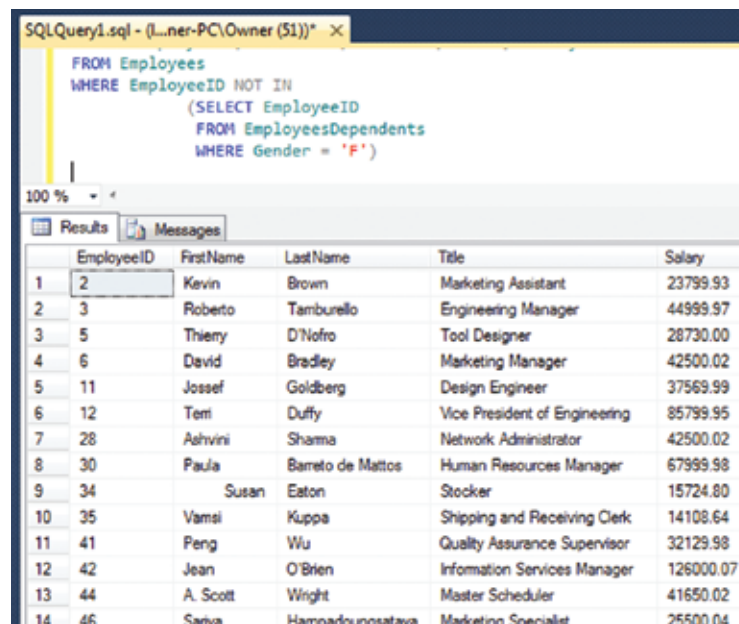
So, which type of query would be more appropriate to use in the above scenario. Although either one is fine to retrieve the results for the example we've used, there are a couple of general rules to follow to help make an appropriate decision. They are as follows:

- When the ending result columns to be retrieved are from a single table use a subquery.
- When the columns come from two or more tables use a join query.

Subqueries and the NOT IN Operator

Like the IN operator, we've learned about the NOT IN operator in Chapter 5. The NOT IN operator is used to retrieve rows where a column's values are NOT found within a group of values. For example, let's use the same query as the one in Figure 4 except let's replace the IN operator with the NOT IN operator as shown in Figure 10.

```
SELECT EmployeeID, FirstName, LastName, Title, Salary
FROM Employees
WHERE EmployeeID NOT IN
    (SELECT EmployeeID
     FROM EmployeesDependents
     WHERE Gender = 'F')
```



	EmployeeID	FirstName	LastName	Title	Salary
1	2	Kevin	Brown	Marketing Assistant	23799.93
2	3	Roberto	Tamburello	Engineering Manager	44999.97
3	5	Thierry	D'Nofro	Tool Designer	28730.00
4	6	David	Bradley	Marketing Manager	42500.02
5	11	Josief	Goldberg	Design Engineer	37569.99
6	12	Terri	Duffy	Vice President of Engineering	85799.95
7	28	Ashvini	Sharma	Network Administrator	42500.02
8	30	Paula	Barreto de Mattos	Human Resources Manager	67999.98
9	34	Susan	Eaton	Stocker	15724.80
10	35	Vamsi	Kuppa	Shipping and Receiving Clerk	14108.64
11	41	Peng	Wu	Quality Assurance Supervisor	32129.98
12	42	Jean	O'Brien	Information Services Manager	126000.07
13	44	A. Scott	Wright	Master Scheduler	41650.02
14	46	Satva	Hammadounosatava	Marketing Specialist	25500.04

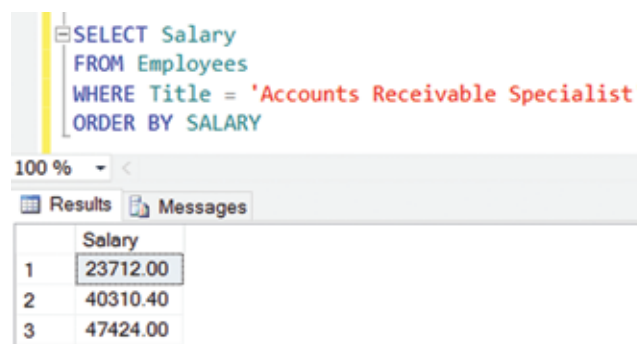
Figure 10

Figure 10 shows only a small result set of the rows returned. There are a total 98 rows returned. Since there are 102 employees this makes sense because our result set should contain employees with male dependents as well as employees with no dependents at all. There are four employees with female dependents that were NOT included because of the NOT IN operator so 102 minus 4 equals 98.

Using the ALL and ANY Keywords

The ALL keyword and ANY keyword can be used with comparison operators to produce result sets with zero, one or multiple values. These keywords can be useful when performing analysis work. For example, let's execute the following statement to retrieve all salaries for Accounts Receivable Specialists. The result set is shown in Figure 11.

```
SELECT Salary
FROM Employees
WHERE Title = 'Accounts Receivable Specialist'
ORDER BY SALARY
```



The screenshot shows a SQL query editor with the following text:

```
SELECT Salary
FROM Employees
WHERE Title = 'Accounts Receivable Specialist'
ORDER BY SALARY
```

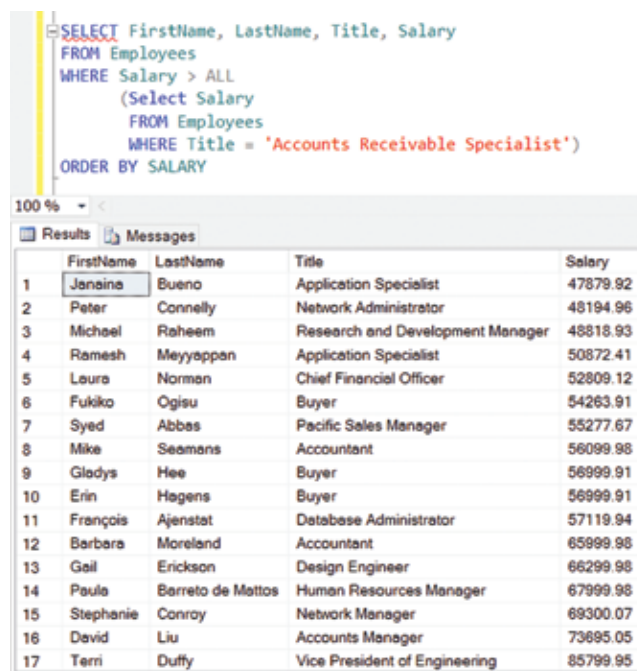
Below the query editor, there is a tab labeled "Results" and a tab labeled "Messages". The "Results" tab is active, displaying a table with the following data:

	Salary
1	23712.00
2	40310.40
3	47424.00

Figure 11

Suppose a manager wants to see a list of employees who have salaries that are higher than any of the salaries of employees with the title of “Accounts Receivable Specialist.” That means, the list must contain salaries higher than 47,424.00 since the salaries must be higher than ALL the salaries for Accounts Receivable Specialists. The subquery to perform this operation is shown below with a subset of the results following in Figure 12, 26 rows are returned.

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary > ALL
    (Select Salary FROM Employees WHERE Title = 'Accounts Receivable Specialist')
ORDER BY SALARY
```



```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary > ALL
    (Select Salary FROM Employees WHERE Title = 'Accounts Receivable Specialist')
ORDER BY SALARY
```

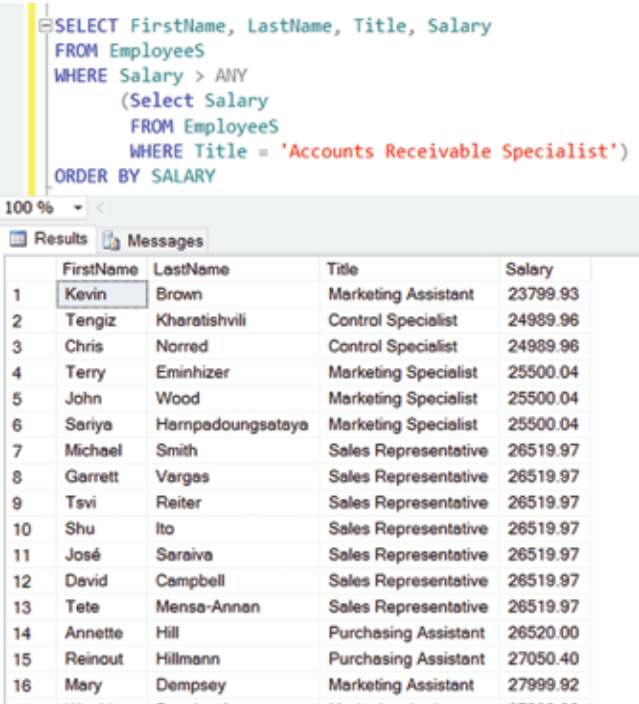
	FirstName	LastName	Title	Salary
1	Janeina	Bueno	Application Specialist	47879.92
2	Peter	Connelly	Network Administrator	48194.96
3	Michael	Raheem	Research and Development Manager	48818.93
4	Ramesh	Meyyappan	Application Specialist	50872.41
5	Leura	Norman	Chief Financial Officer	52809.12
6	Fukiko	Ogisu	Buyer	54263.91
7	Syed	Abbas	Pacific Sales Manager	55277.67
8	Mike	Seamans	Accountant	56099.98
9	Gladys	Hee	Buyer	56999.91
10	Erin	Hagens	Buyer	56999.91
11	François	Ajenstat	Database Administrator	57119.94
12	Barbara	Moreland	Accountant	65999.98
13	Gail	Erickson	Design Engineer	66299.98
14	Paule	Barreto de Mattos	Human Resources Manager	67999.98
15	Stephanie	Conroy	Network Manager	69300.07
16	David	Liu	Accounts Manager	73695.05
17	Terri	Duffy	Vice President of Engineering	85799.95

Figure 12

I included an ORDER BY clause in the subquery so that the lowest salary was listed in the first row so we can see, easily, that the result set contains salaries that exceed ALL the salaries where title is “Accounts Receivable Specialist.”

Let’s try the same subquery but use the ANY keyword in place of the ALL keyword as shown below. The results are shown in Figure 13.

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary > ANY
    (Select Salary
     FROM Employees
     WHERE Title = 'Accounts Receivable Specialist')
ORDER BY SALARY
```



```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary > ANY
    (Select Salary
     FROM Employees
     WHERE Title = 'Accounts Receivable Specialist')
ORDER BY SALARY
```

	FirstName	LastName	Title	Salary
1	Kevin	Brown	Marketing Assistant	23799.93
2	Tengiz	Kharatishvili	Control Specialist	24989.96
3	Chris	Norred	Control Specialist	24989.96
4	Terry	Eminhizer	Marketing Specialist	25500.04
5	John	Wood	Marketing Specialist	25500.04
6	Sariya	Hernpadoungsetaya	Marketing Specialist	25500.04
7	Michael	Smith	Sales Representative	26519.97
8	Garrett	Vargas	Sales Representative	26519.97
9	Tsvi	Reiter	Sales Representative	26519.97
10	Shu	Ito	Sales Representative	26519.97
11	José	Saraiva	Sales Representative	26519.97
12	David	Campbell	Sales Representative	26519.97
13	Tete	Mensa-Annan	Sales Representative	26519.97
14	Annette	Hill	Purchasing Assistant	26520.00
15	Reinout	Hillmann	Purchasing Assistant	27050.40
16	Mary	Dempsey	Marketing Assistant	27999.92

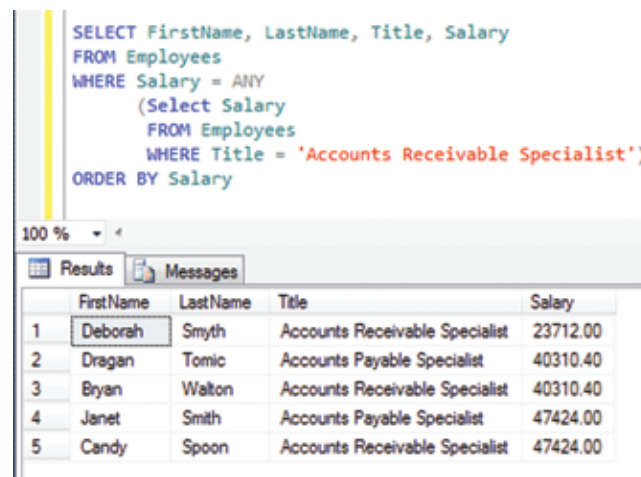
Figure 13

Notice how different our results are as the salary amounts are much lower. This is because using the ANY keyword means that any salary that is greater than the any of the salaries for the title Accounts Receivable Specialists, even the lowest for that title, would meet the criteria. The lowest salary for Accounts Receivable Specialist is 23,712.00 and the first row of the result set contains a salary of 23,799.93 which meets the criteria. In addition, many more rows were returned (85) due to a lower starting salary meeting the criteria.

"= ANY" - Equals Any Operator

Let's take the example we just used for the "> ANY" operator and substitute it with the "= ANY" operator. The syntax of the statement is shown below and the results follow in Figure 14.

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary = ANY
    (Select Salary
     FROM Employees
     WHERE Title = 'Accounts Receivable Specialist')
ORDER BY Salary
```



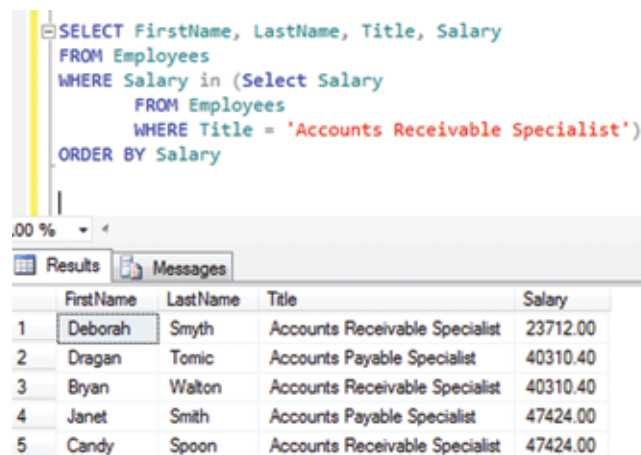
```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary = ANY
    (Select Salary
     FROM Employees
     WHERE Title = 'Accounts Receivable Specialist')
ORDER BY Salary
```

	FirstName	LastName	Title	Salary
1	Deborah	Smyth	Accounts Receivable Specialist	23712.00
2	Dragan	Tomic	Accounts Payable Specialist	40310.40
3	Bryan	Walton	Accounts Receivable Specialist	40310.40
4	Janet	Smith	Accounts Payable Specialist	47424.00
5	Candy	Spoon	Accounts Receivable Specialist	47424.00

Figure 14

As you can see from the result set, two of the three salaries for "Accounts Receivable Specialist" have matching salaries for other titles, in this case, Accounts Payable Specialist.

The "= ANY" operator is equivalent to using the IN operator. Below in Figure 15, the IN operator is being used in place of the "= ANY" operator and is returning the same result set as shown in Figure 14.



```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary in (Select Salary
                 FROM Employees
                 WHERE Title = 'Accounts Receivable Specialist')
ORDER BY Salary
```

	FirstName	LastName	Title	Salary
1	Deborah	Smyth	Accounts Receivable Specialist	23712.00
2	Dragan	Tomic	Accounts Payable Specialist	40310.40
3	Bryan	Walton	Accounts Receivable Specialist	40310.40
4	Janet	Smith	Accounts Payable Specialist	47424.00
5	Candy	Spoon	Accounts Receivable Specialist	47424.00

Figure 15

“!= ANY” - Not Equal Any Operator

Let’s use the same example as we did with the “= ANY” operator only, this time, let’s use “!= ANY” instead. The syntax is shown below and the results are displayed in Figure 16.

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary != ANY
    (Select Salary
     FROM Employees
     WHERE Title = 'Accounts Receivable Specialist')
ORDER BY Salary
```

Salary
23712.00
40310.40
47424.00

First Name	Last Name	Title	Salary
50	Ben	Buyer	30750.54
51	Engras	Accounts Payable Specialist	40310.40
52	Don	Application Specialist	40597.93
62	Mindy	Benefits Specialist	41396.00
63	Michael	Senior Design Engineer	41437.51
64	A. Scott	Master Scheduler	41650.02
65	Ashvin	Network Administrator	42590.02
66	David	Marketing Manager	42590.02
67	Hazel	Quality Assurance Manager	42546.07
68	Sharon	Design Engineer	44196.99
69	Hao	Human Resources Administrator	44368.97
70	Roberto	Engineering Manager	44986.97
71	Mary	Marketing Specialist	45000.08
72	Linda	Buyer	45596.93
73	Janet	Accounts Payable Specialist	47424.00
74	Janeiro	Application Specialist	48757.90
75	Peter	Network Administrator	48194.96
76	Michael	Research and Development M.	48516.93
77	Ramesh	Application Specialist	50572.41
78	Laure	Chief Financial Officer	52500.12
79	Fukiko	Buyer	54283.91

Figure 16

There are two result sets displayed purposely in Figure 16 for your understanding. The first result set contains the salaries for the title “Accounts Receivable Specialists” while the second result set contains the results from using the “!= ANY” operator. Notice the salaries enclosed in the red rectangles. They match two of the salaries for “Accounts Receivable Specialists.” At first you might ask yourself why they were returned when we used the “not any” operator. It’s because the “!= ANY” operator means for each salary amount “not 23712.00” OR “not 40310.40” OR “not 47424.00.” So, when 40310.40 is compared to 23712.00 it is not equal so it’s included in the result set.

Although the “= ANY” operator will return the same results as the IN operator, the same is not true of the “!= ANY” operator and the NOT IN operator. This is because the NOT IN operator means amount “not 23712.00” AND “not 40310.40” AND “not 47424.00.” Below is the query statement using the NOT IN operator and a subset of the results is shown in Figure 17. Notice two less rows are returned.

```
SELECT FirstName, LastName, Title, Salary
FROM Employees
WHERE Salary NOT IN
    (Select Salary
     FROM Employees
     WHERE Title = 'Accounts Receivable Specialist')
AND Title <> 'Accounts Receivable Specialist'
ORDER BY SALARY
```

	FirstName	LastName	Title	Salary
1	Jerry	Bucholt	Stockler	13986.08
2	Pat	Coleman	Junior	13797.36
3	Stuart	McInnes	Junior	13797.36
4	Vernell	Kuppe	Shipping and Receiving Clerk	14108.84
5	Magnus	Hedlund	Facilities Administration Assistant	14479.92
6	William	Vong	Scheduling Assistant	15245.16
7	Mark	Hammington	Quality Assurance Technician	15707.96
8	Seán	Alexander	Quality Assurance Technician	15707.96
9	Andrew	Berglund	Quality Assurance Technician	15707.96
10	Scottie	Chenrathangkha	Quality Assurance Technician	15707.96
11	Kim	Ralls	Stockler	15724.80
12	Susan	Eaton	Stockler	15724.80
13	Ju	Derry	Junior	16161.00
14	Lori	Penor	Junior	16161.00
15	Eric	Kurjen	Buyer	16379.87
16	Shirley	Wood	Purchasing Manager	20568.80
17	Karen	Brown	Marketing Assistant	23760.93

Figure 17

Correlated Subqueries

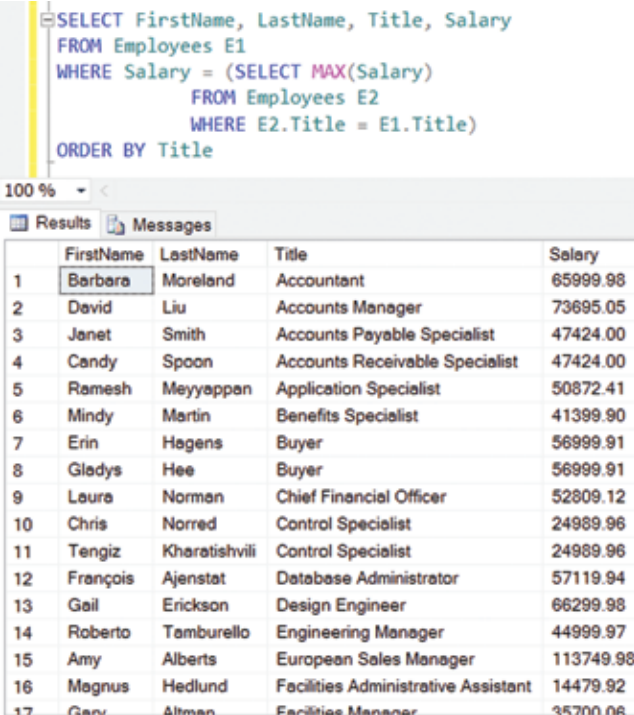
So far, we have only talked only about noncorrelated subqueries, where the outer query is dependent on the return values of the inner query to complete. In the case of correlated subqueries, the inner query is dependent on the values provided by the outer query in addition to the outer query being dependent on the values of the inner query.

As an example of a correlated subquery, let's say a manager would like to see a listing of individuals with the highest salaries for each Title. The query would be written as follows:

```
SELECT FirstName, LastName, Title, Salary
FROM Employees E1
WHERE Salary = (SELECT MAX(Salary)
                FROM Employees E2
                WHERE E2.Title = E1.Title)
ORDER BY Title
```

In the above subquery, the inner query requires the title from the outer query to understand what title to perform the aggregation on. Once the aggregation is performed, the outer query uses the value obtained from the inner query to determine whether the current row being processed by the outer query meets the criteria of maximum salary for a title. Notice how alias names are given to the Employees table in the outer query as well as the Employees table in the inner query. Alias names are required to decipher the Employees table in the outer query from the Employees table in the inner query.

A subset of the results is shown in Figure 18.



```

SELECT FirstName, LastName, Title, Salary
FROM Employees E1
WHERE Salary = (SELECT MAX(Salary)
                FROM Employees E2
                WHERE E2.Title = E1.Title)
ORDER BY Title

```

	FirstName	LastName	Title	Salary
1	Barbara	Moreland	Accountant	65999.98
2	David	Liu	Accounts Manager	73695.05
3	Janet	Smith	Accounts Payable Specialist	47424.00
4	Candy	Spoon	Accounts Receivable Specialist	47424.00
5	Ramesh	Meyyappan	Application Specialist	50872.41
6	Mindy	Martin	Benefits Specialist	41399.90
7	Erin	Hagens	Buyer	56999.91
8	Gladys	Hee	Buyer	56999.91
9	Laura	Norman	Chief Financial Officer	52809.12
10	Chris	Norred	Control Specialist	24989.96
11	Tengiz	Kharatishvili	Control Specialist	24989.96
12	François	Ajenstat	Database Administrator	57119.94
13	Gail	Erickson	Design Engineer	66299.98
14	Roberto	Tamburello	Engineering Manager	44999.97
15	Amy	Alberts	European Sales Manager	113749.98
16	Magnus	Hedlund	Facilities Administrative Assistant	14479.92
17	Gary	Altman	Facilities Manager	35700.06

Figure 18

EXISTS Operator

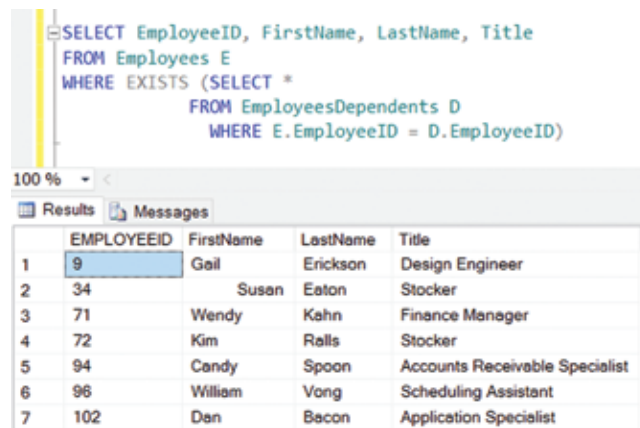
The EXISTS operator is used in a correlated subquery to determine the existence or nonexistence of data that meets the criteria of the subquery. No data is returned from the inner query. Rather the inner query produces a true or false value. As an example, we can list employees who have dependents with the following query.

```

SELECT EmployeeID, FirstName, LastName, Title
FROM Employees E
WHERE EXISTS (SELECT *
              FROM EmployeesDependents D
              WHERE E.EmployeeID = D.EmployeeID)

```

Notice the inner query uses the asterisk in place of column name. It really doesn't matter since the whole purpose of the inner query is to determine existence or nonexistence. Also notice how the EmployeeID from the Employees table (outer query) and EmployeeID from the EmployeesDependents table (inner query) are compared. This is, obviously, a correlated subquery since the inner query is dependent on the outer query. The results from this query are shown in Figure 19.



```

SELECT EmployeeID, FirstName, LastName, Title
FROM Employees E
WHERE EXISTS (SELECT *
              FROM EmployeesDependents D
              WHERE E.EmployeeID = D.EmployeeID)
  
```

	EMPLOYEEID	FirstName	LastName	Title
1	9	Gail	Erickson	Design Engineer
2	34	Susan	Eaton	Stocker
3	71	Wendy	Kahn	Finance Manager
4	72	Kim	Rolls	Stocker
5	94	Candy	Spoon	Accounts Receivable Specialist
6	96	William	Vong	Scheduling Assistant
7	102	Dan	Bacon	Application Specialist

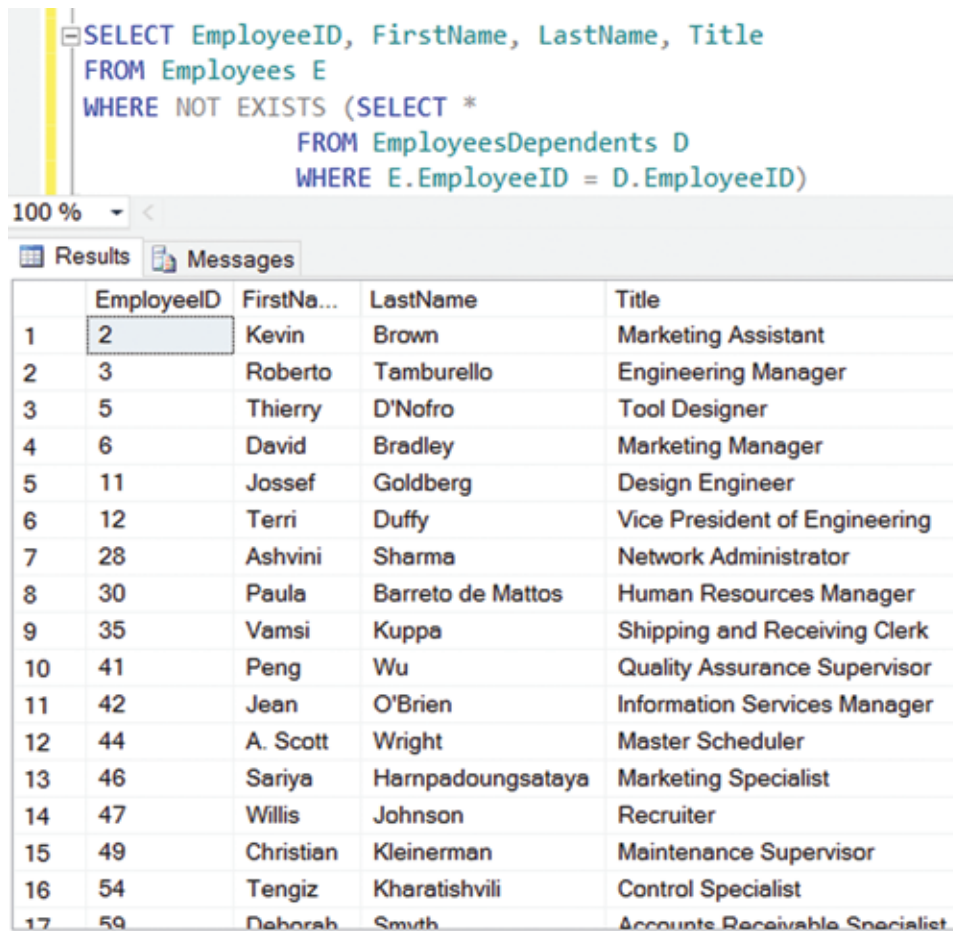
Figure 19

NOT EXISTS Operator

The NOT EXISTS operator test for nonexistence of rows. For example, if we use the same query as that in the EXISTS example but add the “NOT” keyword, our result set would contain employees without dependents. The query is shown below and a subset of the result set follows in Figure 20.

```

SELECT EmployeeID, FirstName, LastName, Title
FROM Employees E
WHERE NOT EXISTS (SELECT *
                  FROM EmployeesDependents D
                  WHERE E.EmployeeID = D.EmployeeID)
  
```

```

SELECT EmployeeID, FirstName, LastName, Title
FROM Employees E
WHERE NOT EXISTS (SELECT *
                  FROM EmployeesDependents D
                  WHERE E.EmployeeID = D.EmployeeID)

```

	EmployeeID	FirstNa...	LastName	Title
1	2	Kevin	Brown	Marketing Assistant
2	3	Roberto	Tamburello	Engineering Manager
3	5	Thierry	D'Nofro	Tool Designer
4	6	David	Bradley	Marketing Manager
5	11	Jossef	Goldberg	Design Engineer
6	12	Terri	Duffy	Vice President of Engineering
7	28	Ashvini	Sharma	Network Administrator
8	30	Paula	Barreto de Mattos	Human Resources Manager
9	35	Vamsi	Kuppa	Shipping and Receiving Clerk
10	41	Peng	Wu	Quality Assurance Supervisor
11	42	Jean	O'Brien	Information Services Manager
12	44	A. Scott	Wright	Master Scheduler
13	46	Sariya	Harnpadoungsataya	Marketing Specialist
14	47	Willis	Johnson	Recruiter
15	49	Christian	Kleinerman	Maintenance Supervisor
16	54	Tengiz	Kharatishvili	Control Specialist
17	59	Deborah	Smith	Accounts Receivable Specialist

Figure 20

Summary

This chapter introduced you to the functionalities and fundamentals of correlated and noncorrelated subqueries. The chapter provided explanations and examples where subqueries can be used to help simplify complex SQL coding to retrieve specific information along with the correct syntax. Topics included the use of various operators with subqueries including the IN, NOT IN, ALL, and ANY comparison operators. Also, the chapter discussed the usage of EXISTS and NOT EXISTS operators to determine the existence or nonexistence of data that meets the criteria of the subquery.

Exercises

1. Write a subquery to determine the employee or employees with the maximum number of vacation days.
2. Write a subquery to list the employees who have an amount of vacation hours that are greater than the average number of vacation hours.
3. Use the IN operator to list employees' names who live in the state of California. Hint: You need to use the addresses table in your inner query.
4. Use an inner join to obtain the same results as the exercise in number 3.
5. Use the NOT IN operator to list employees' names who DO NOT live in the state of Washington.
6. Use the ALL operator to list employees' names and their salaries for those who have salaries that are greater than all the salaries of employees that have ManagerID = 41.
7. Use the ANY operator to list employees' names and their salaries for those who have salaries that are less than any of the salaries of employees that have ManagerID = 41.
8. Use a correlated subquery to list employees' names, titles, and salaries for those who have the highest salaries for each manager.
9. Use the EXISTS operator to find employees who live in the state of California.
10. Use the NOT EXISTS operator to find employees who DO NOT live in the state of Washington.
11. Using a subquery, list all male employees with salaries higher than any female employee.
12. Use the ANY operator to list all employees paid more than any of the Design Engineers.
13. Use a subquery to list all employees who do NOT have any dependents.
14. Use a subquery to find all employees who have more sick leave hours than the average employee.
15. Use a correlated subquery to select the lowest paid employees under each manager.