

4

Single Table Query Basics

This chapter will cover the use of SELECT statements to retrieve/query information from the database. We will be using the SELECT statement to perform single table queries. Single table queries are used to retrieve information from one table in a database. In later chapters, we will be covering multiple table queries which using the JOIN operation to link tables to one another.

Before beginning this chapter make sure you load the Human Resources Database by following the instructions in Appendix B.

CHAPTER OBJECTIVES

This chapter will show you the basics of the SELECT statement. In addition to understand how to retrieve data from a database, you will learn how to filter your output to retrieve specific rows and columns that fit your needs. The objectives for this chapter are as follows:

- Understand how to retrieve information using simple SELECT statements
- Understand how to use alias names for clarity
- Understand how to eliminate duplicate rows using the DISTINCT clause
- Understand how to use the WHERE clause to filter output
- Understand how to sort your output using the ORDER BY clause
- Understand how to recognize errors in your SQL code

Environment Setup

Before we begin let's start SQL Server and make sure we select the proper database environment to work from. Select "New Query" from the Tool Bar Menu and use the drop-down box to select the "HumanResources" database. In Figure 1 you will see the red rectangle surrounding the database name at the top of the screen to show which database we will be working in.

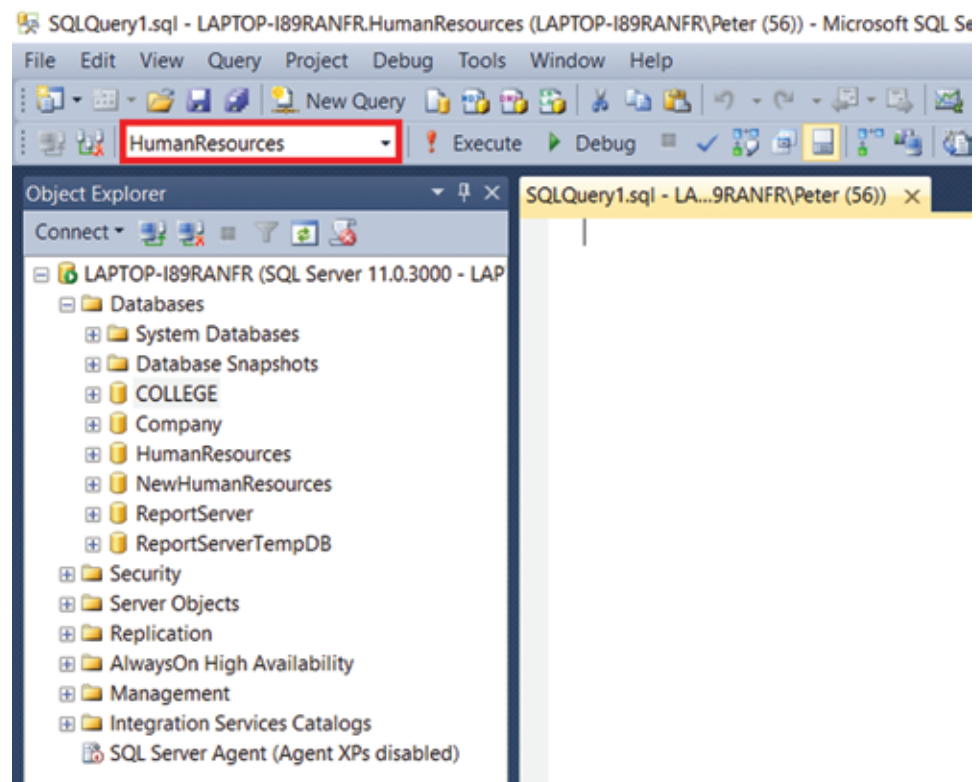


Figure 1

Now that we've set up the proper environment let's save our query window in a safe spot so that all the work we do in this chapter can be retrieved again later. This is very much the same as we do with Microsoft Word or Excel except the file extension is ".sql." See the red circled area for clarification in Figure 2.

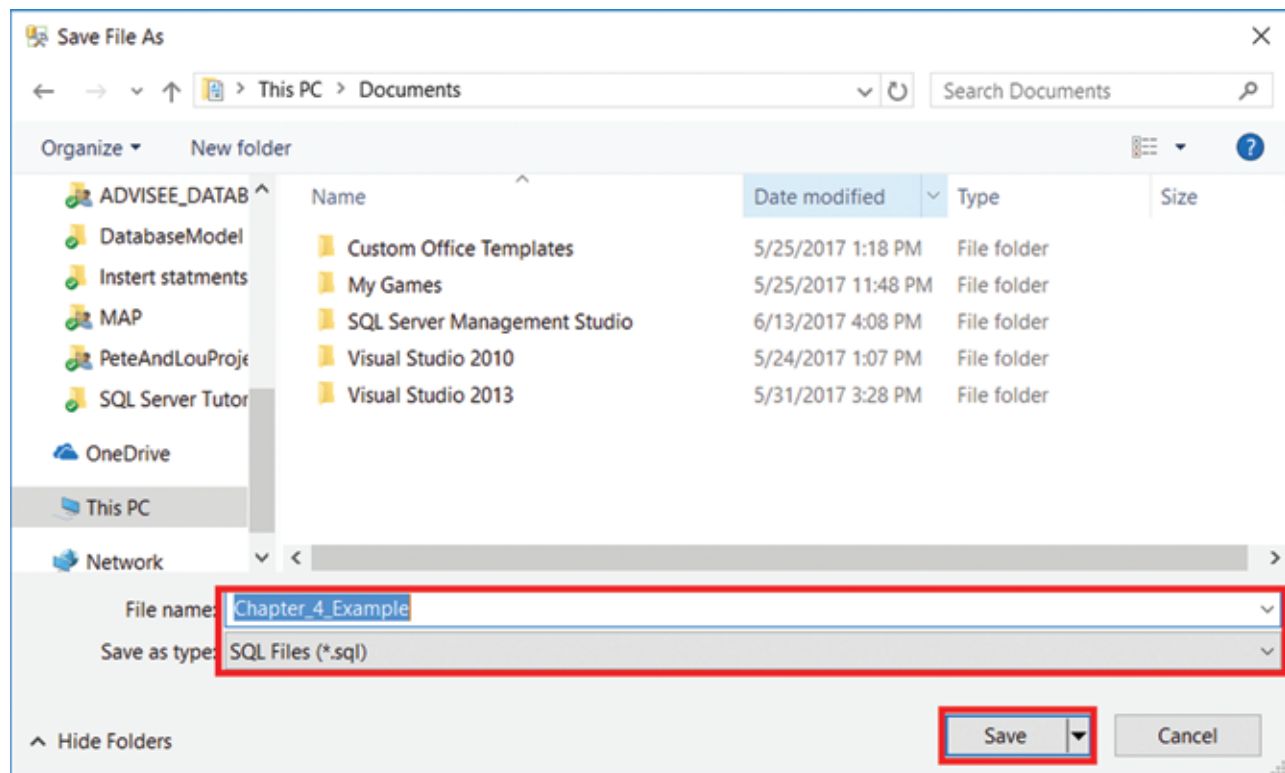


Figure 2

SELECT Statement

Ok, now that we've set up our environment let's start with our first SELECT statement. As you do in other applications such as Word and Excel, don't forget to hit the save button once in a while so you don't lose your work.

Figure 3 shows some resulting rows from using the following SELECT statement.

```
SELECT *
FROM EMPLOYEES
```

► **NOTE:** SQL Server is NOT case sensitive.

EmployeeID	FirstName	MiddleName	LastName	Title	BirthDate	MaritalStatus	Gender	HireDate	Salary	VacationHours	SickLeave
1	Kevin	F	Brown	Marketing Assistant	1977-06-03	S	M	1997-03-26	23799.93	42	41
2	Roberto	NULL	Tamburello	Engineering Manager	1964-12-13	M	M	1997-12-12	44999.97	2	21
3	Thierry	B	D'Nofre	Tool Designer	1949-08-29	M	M	1998-01-11	28730.00	9	24
4	David	M	Bradley	Marketing Manager	1965-04-19	S	M	1998-01-20	42500.02	40	40
5	Gail	A	Erickson	Design Engineer	1942-10-29	M	F	1998-02-06	66299.98	5	22
6	Josef	H	Goldberg	Design Engineer	1949-04-11	M	M	1998-02-24	37569.99	6	23
7	Toni	Lee	Duffy	Vice President of Engineering	1961-09-01	S	F	1998-03-03	85799.95	1	20
8	Ashvini	R	Sharma	Network Administrator	1967-04-28	S	M	1999-01-05	42500.02	70	55
9	Paula	M	Baneto de Mattos	Human Resources Manager	1966-03-14	M	F	1999-01-07	67999.98	54	47
10	Susan	W	Eaton	Stockler	1968-03-20	S	F	1999-01-08	15724.80	99	68
11	Vinay	N	Kumar	Shoreline and Recreations Clerk	1967-04-15	M	M	1999-01-08	14108.64	95	67

Figure 3

The asterisk (*) means to display all columns from a table specified in the FROM clause. The example above will display all columns in the **EMPLOYEES** table from the **HumanResources** database. You can also write the SELECT statement using one line as shown below, but it is standard practice to put each SQL clause on a separate line for clarity as shown in Figure 3.

```
SELECT * FROM EMPLOYEES
```

► **NOTE:** Make sure you select the HumanResources database as shown circled in Figure 1 to execute the statement. Otherwise you will receive the following error

Invalid Object Name “EMPLOYEES.”

Tables are considered objects in the database and that is why the error refers to “object.” In Figure 4 you can see that the master database was selected rather than the HumanResources resulting in an error because the Employees table does not exist in the master database.

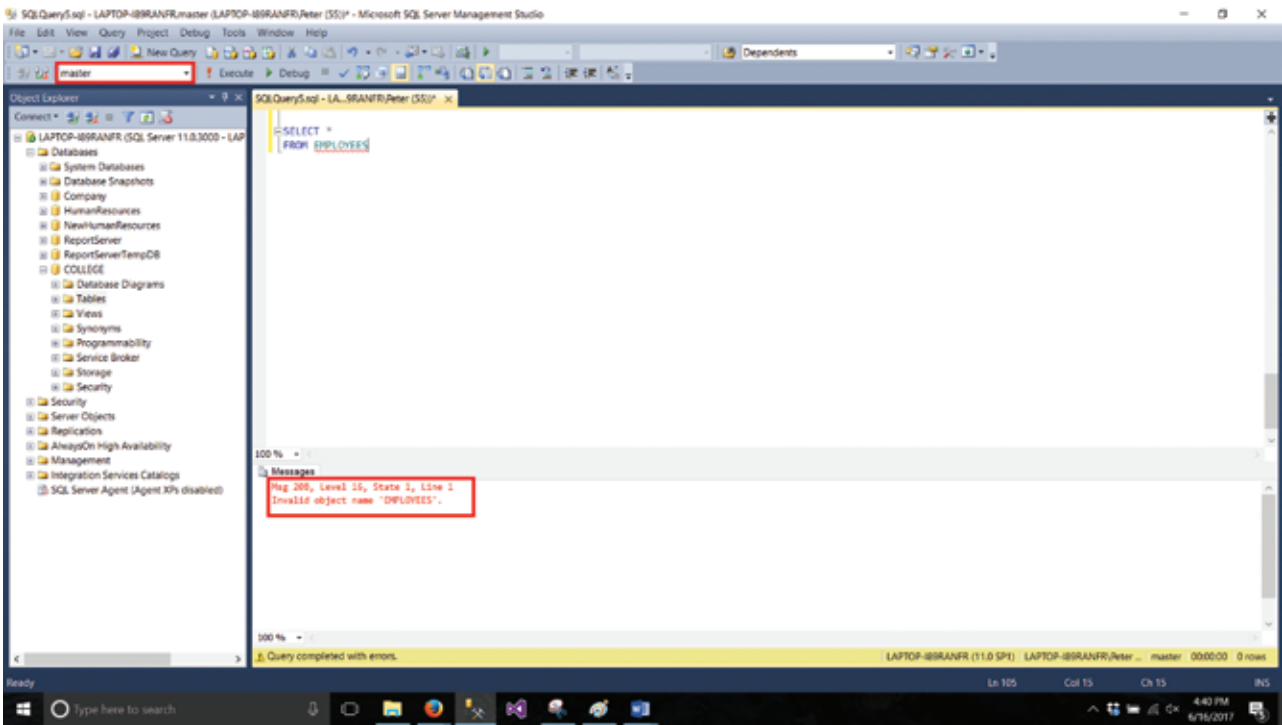


Figure 4

FROM Clause

The FROM clause in the Select statement tells SQL Server what table to use. The output from the query is called a result table. Because we didn’t filter the data to retrieve using a WHERE clause all rows in the EMPLOYEE table were retrieved. The result table should contain 102 rows as shown in Figure 5.

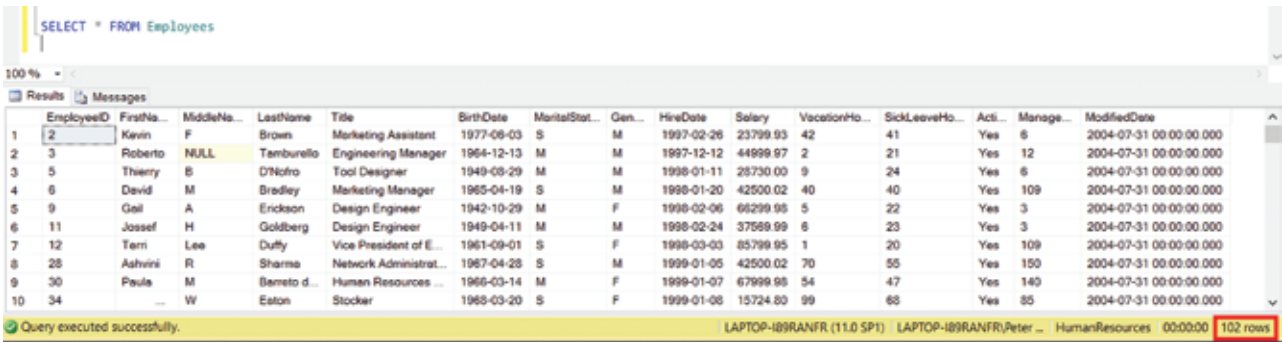


Figure 5

The SELECT statement below will produce the same results as that shown in Figure 5.

```
SELECT EmployeeID,  
       FirstName,  
       MiddleName,  
       LastName,  
       Title,  
       BirthDate,  
       MaritalStatus,  
       Gender,  
       HireDate,  
       Salary,  
       VacationHours,  
       SickLeaveHours,  
       Active,  
       ManagerID,  
       ModifiedDate  
FROM EMPLOYEES
```

Notice that all the columns are written out separated by commas. This is necessary to separate the column names so that SQL Server understands each column name. Also notice that the **last** column name does NOT have a comma following it. This is because this tells SQL Server there are no more columns to select.

Obviously, the statement using all the column names takes much more effort to code as opposed to using the asterisk, but if you only want to retrieve specific columns it's necessary. For example, the statement below uses the SELECT statement to retrieve the first name, last name, and hire date of the employee. Results are shown in Figure 6.

```
SELECT FirstName,  
       LastName,  
       HireDate  
FROM EMPLOYEES
```

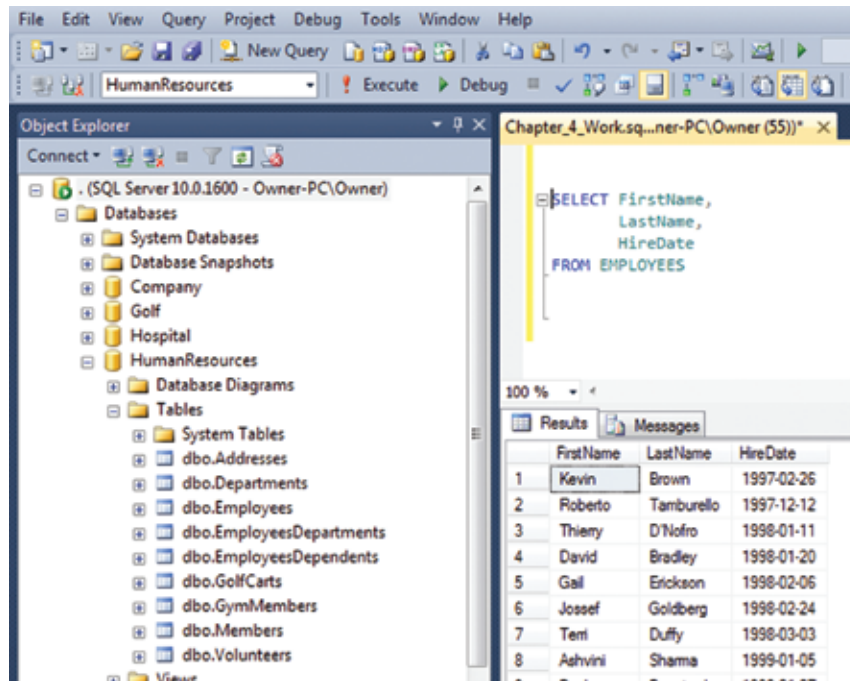


Figure 6

We can also select the columns in any order we would like. For example, the SELECT statement in Figure 7 selects the hire date first and then the last name and first name.

```
SELECT HireDate,
       LastName,
       FirstName FROM EMPLOYEES
```

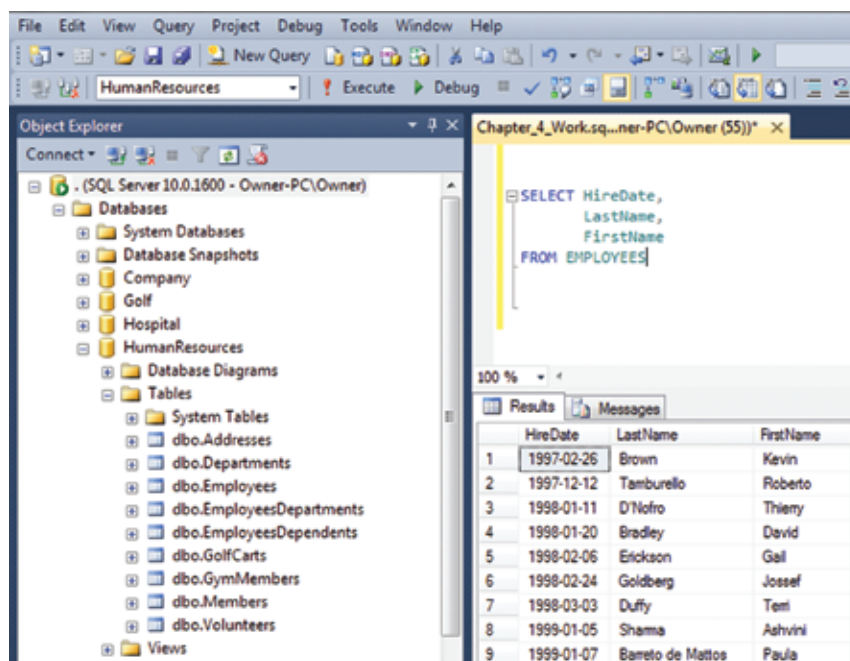


Figure 7

We can put each column on the same line as shown below.

```
SELECT HireDate, LastName, FirstName
FROM EMPLOYEES
```

However, is it a standard to put column names on separate lines to provide clarity to the developer.

```
SELECT HireDate,
       LastName,
       FirstName
FROM EMPLOYEES
```

Notice that the column names are indented to line up with under each other while the SELECT and FROM clause are left aligned. This is not necessary as the statement will execute regardless of the positioning. However, it is done this way, once again, to provide clarity.

It is very appropriate to follow standards such as indenting and starting new lines for clauses as it makes it more readable for other developers to read your code as well as yourself.

Renaming Columns

We can also rename column names to other names that might be more appropriate for our end users. For example, instead of the column name “Title” the end user may desire the word “Position.” See the example in Figure 8.

```
SELECT FirstName First,
       LastName Last,
       BirthDate DOB,
       Title Position
FROM EMPLOYEES
```

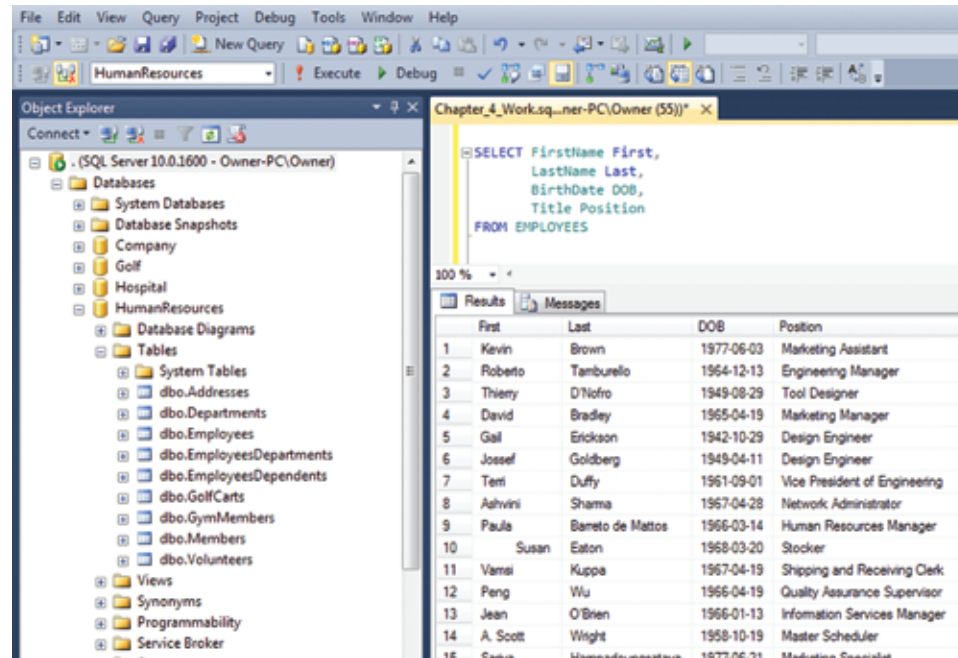



Figure 8

If we want to change column names to titles that contain multiple words we will need to enclose them in single quotes. Figure 9 is an example where the multiple words are used for column names.

```
SELECT FirstName 'First Name',
       LastName 'Last Name',
       BirthDate 'Date of Birth',
       Title 'Position Title'
FROM Employees
```

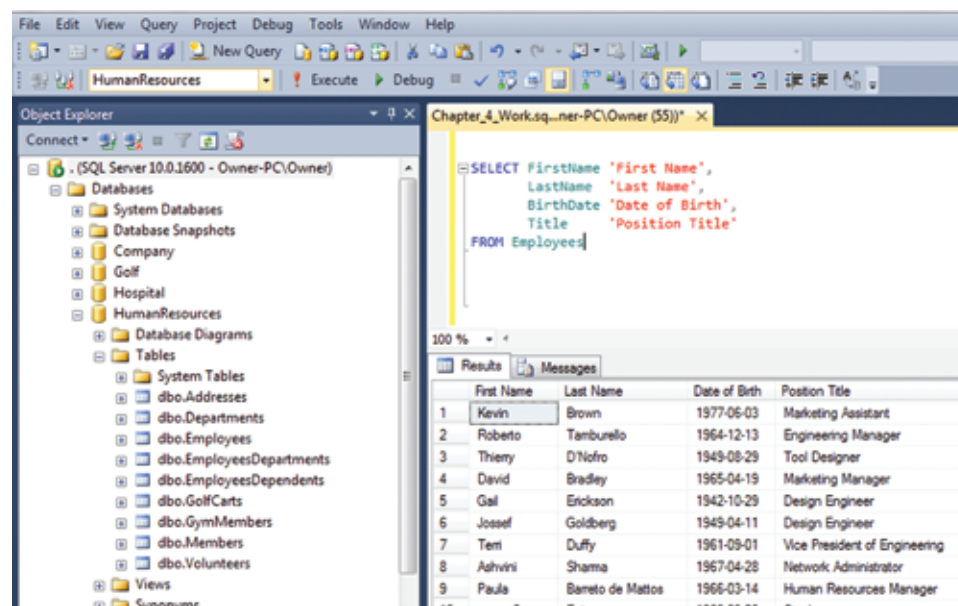


Figure 9

DISTINCT Clause

The DISTINCT clause is used to remove duplicate rows of information. While the primary key ensures uniqueness in tables, a query that doesn't contain a primary key can result in duplicate rows.

An example of duplicate rows looks at the results in Figure 10. The result set lists all the titles of employees from the Employees table. There are 102 rows returned as indicated in the lower right-hand corner of the picture. We are only seeing a small subset of the 102 rows in picture. Even though Figure 10 is showing only a small subset we can see within that small subset there are duplicate rows. The titles highlighted in blue show duplicates of “Design Engineer.”

```
SELECT Title
FROM Employees
```

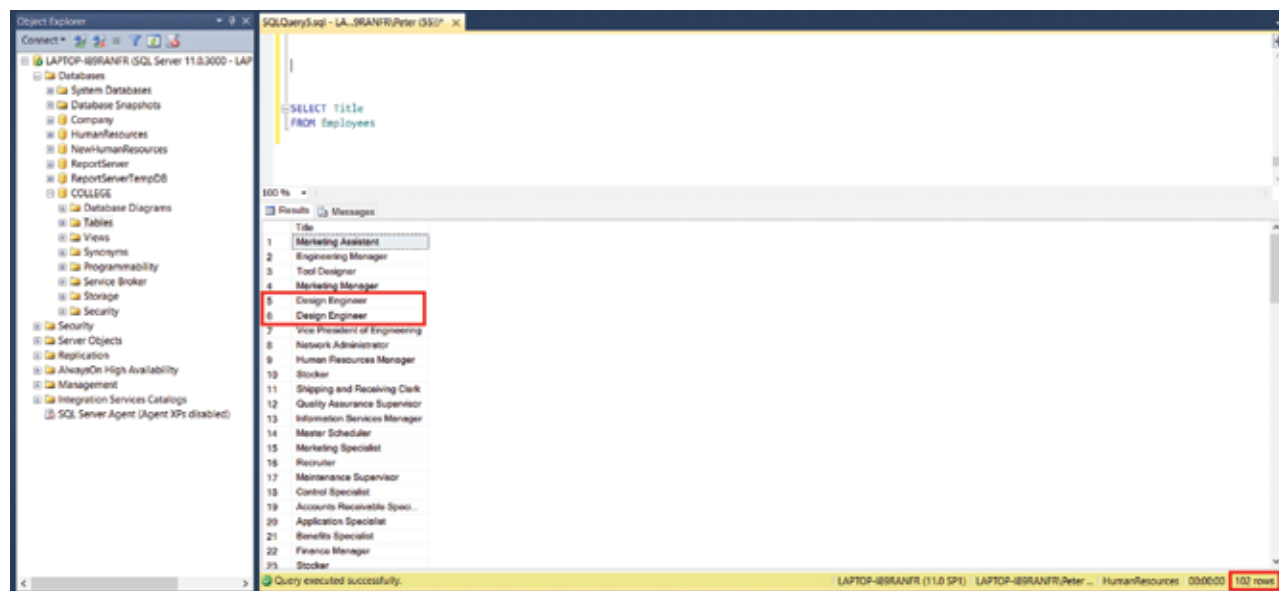


Figure 10

Figure 11 issues the same query only it has added the `DISTINCT` clause to the query. The result is no duplicates and the number of returned rows have been reduced to only 47. This can be helpful to an end user or manager who only wants to see a distinct listing of job titles for the company.

```
SELECT DISTINCT Title
FROM Employees
```

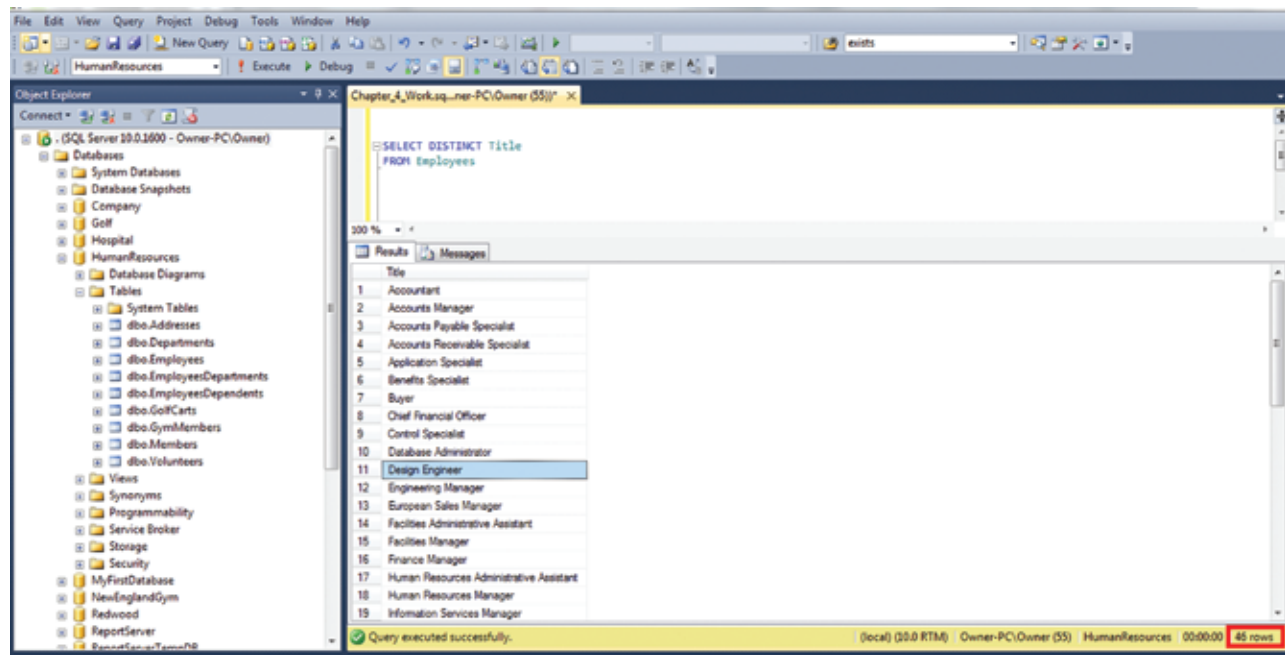


Figure 11

WHERE Clause

The WHERE clause allows us to filter our result set so that only specific rows are returned in a query. For example, a manager may be interested in employees making more than 28,730.00. The query would be written as follows with the results shown in Figure 12.

```
SELECT FirstName,
       LastName,
       Salary
FROM EMPLOYEES
WHERE Salary > 28730.00
```

► **NOTE:** Do not use a comma or any other kind of numeric format such as dollar signs when comparing numbers.

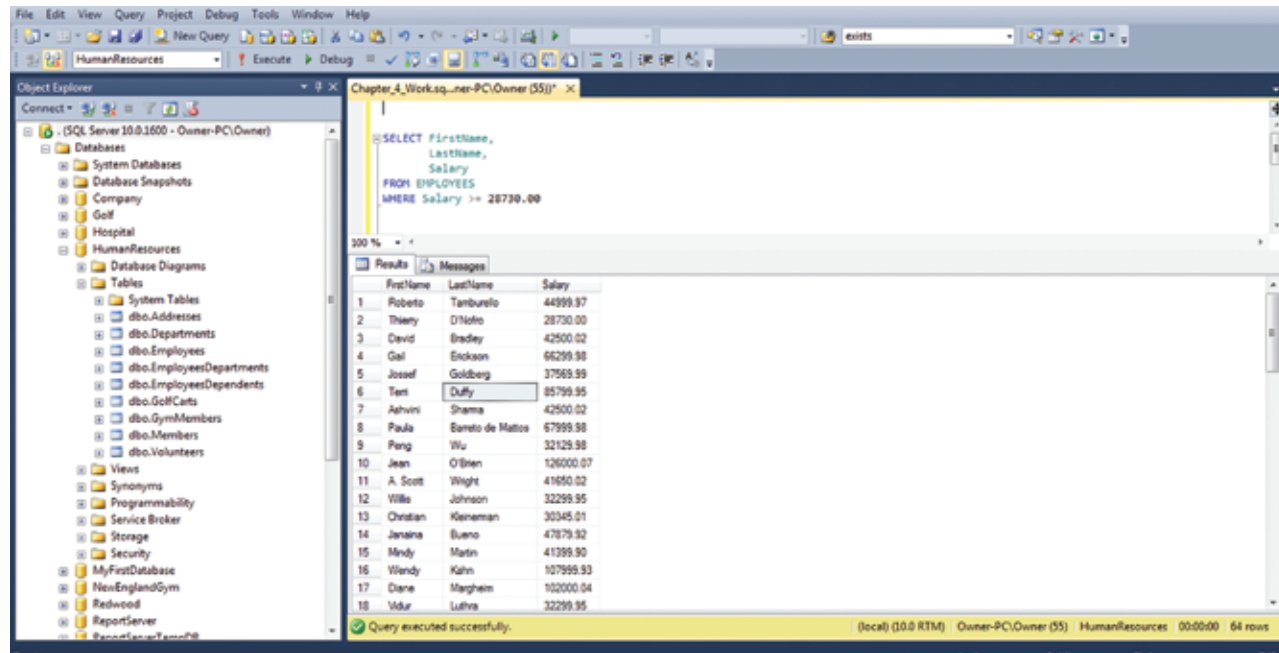


Figure 12

As another example, we can show the manager a list of employees who make 28,730.00 or less. Notice the result set in Figure 13 contains 28730.00 highlighted in red.

```
SELECT FirstName,
       LastName,
       Salary
FROM EMPLOYEES
WHERE Salary <= 28730.00
```

The screenshot shows the SQL Server Enterprise Manager interface. The Object Explorer on the left displays the database structure, including the HumanResources database and its tables. The central pane shows the query results for the query: `SELECT FirstName, LastName, Salary FROM EMPLOYEES WHERE Salary <= 28730.00`. The results are displayed in a grid with columns for FirstName, LastName, and Salary. The salary value 28730.00 for the employee Thery O'Holio is highlighted in red. The status bar at the bottom indicates the query executed successfully, returning 39 rows.

FirstName	LastName	Salary	
1	Kevin	Brown	23799.93
2	Thery	O'Holio	28730.00
3	Susan	Eaton	15724.80
4	Vana	Koppa	14108.64
5	Sally	Hampdenburg	25500.04
6	Tengiz	Khanatkhvili	24969.96
7	Deborah	Smyth	23712.00
8	Kim	Ralls	15724.80
9	Sean	Alexander	15707.96
10	Pilar	Adkeman	28560.04
11	William	Vong	15240.16
12	Mark	Hammington	15707.96
13	Sairaj	Uddin	28288.00
14	Andreas	Berglund	15707.96
15	Alan	Brewer	28288.00
16	Jimmy	Bachoff	13366.08
17	Terry	Emminger	25500.04
18	Brian	LaMee	28288.00

Figure 13

We can use the WHERE clause on all columns including dates. The SELECT statement below will retrieve all employees who were hired on 01/03/2000 or later. Notice the date is placed in single quotes, whereas the numerical values were not. This is because dates are similar to character data or, as many programmers are familiar with, string data. Character data/string data requires quotes, whereas numerical data does not. The results are shown in Figure 14.

```
SELECT FirstName,
       LastName,
       HireDate
FROM Employees
WHERE HireDate >= '2000-01-03'
```

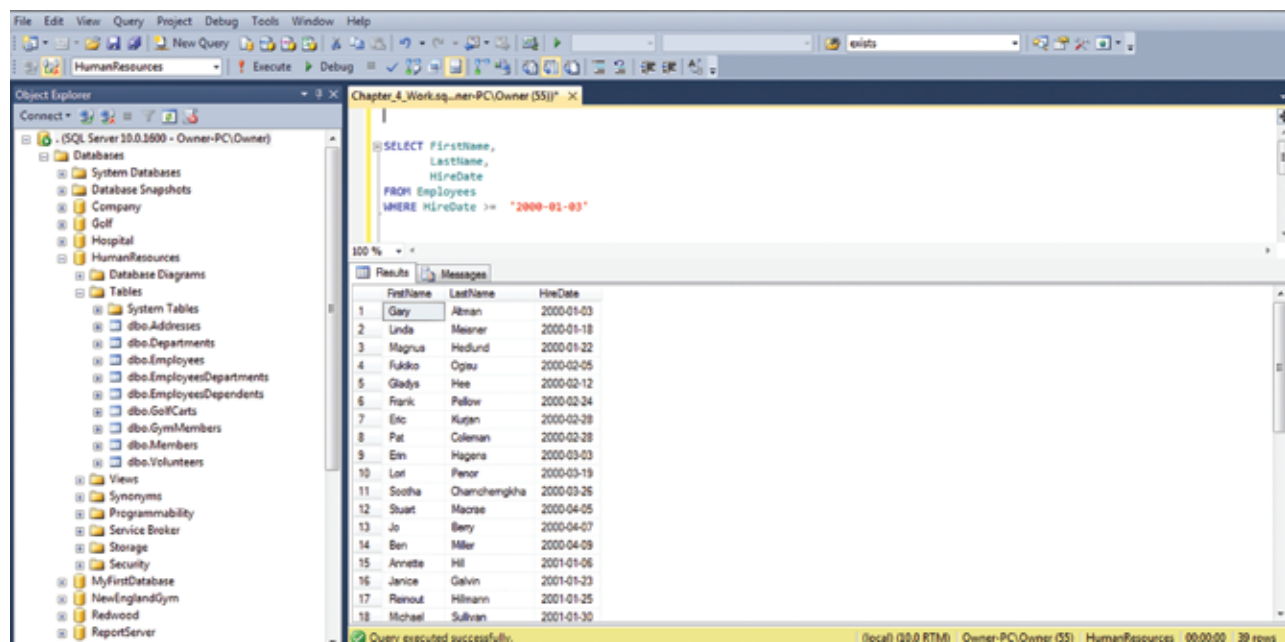


Figure 14

Let's use another WHERE clause to list all employees who are females. Notice the SQL statement below is using single quotes again in the WHERE clause because it is comparing character data.

- **NOTE:** Notice that while Gender is being compared in the WHERE clause it will not be included in the result set. This is to show you that the column in the WHERE clause is not required in the SELECT portion of the statement.

```
SELECT FirstName,
       LastName
FROM Employees
WHERE Gender = 'F'
```

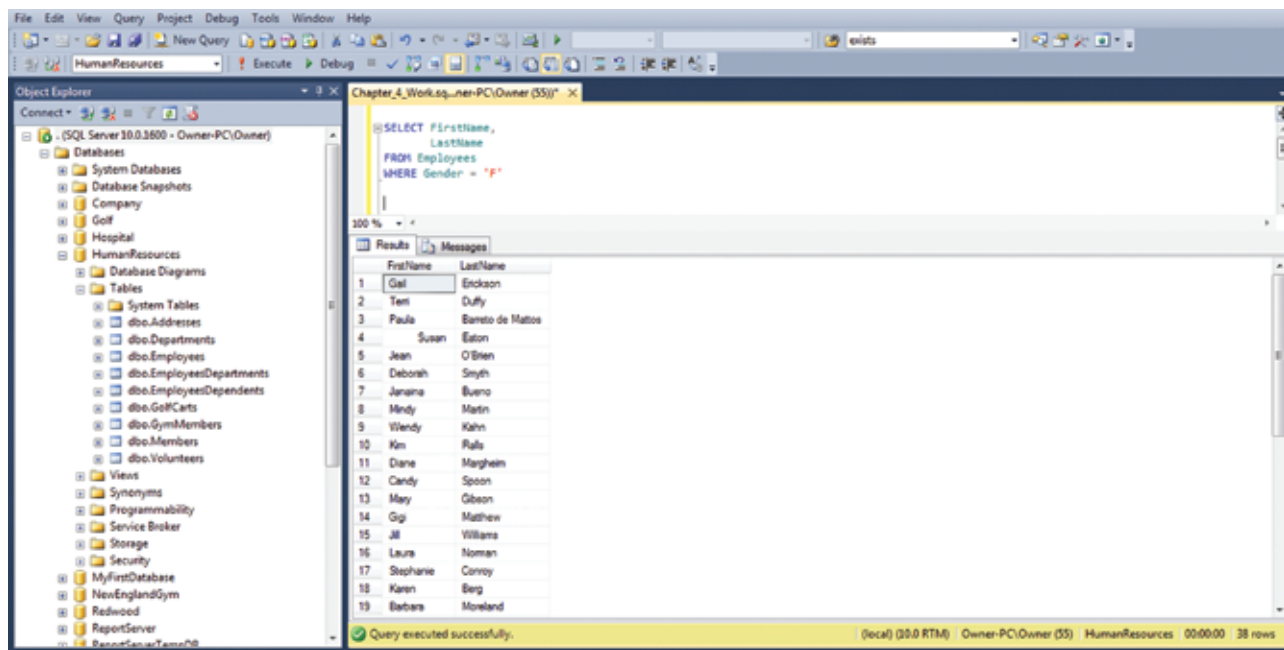


Figure 15

The statement below in Figure 16 is identical to the previous one except that the filtered value is contained in DOUBLE QUOTES rather than SINGLE QUOTES. Double quotes will NOT work as you can see in Figure 16.

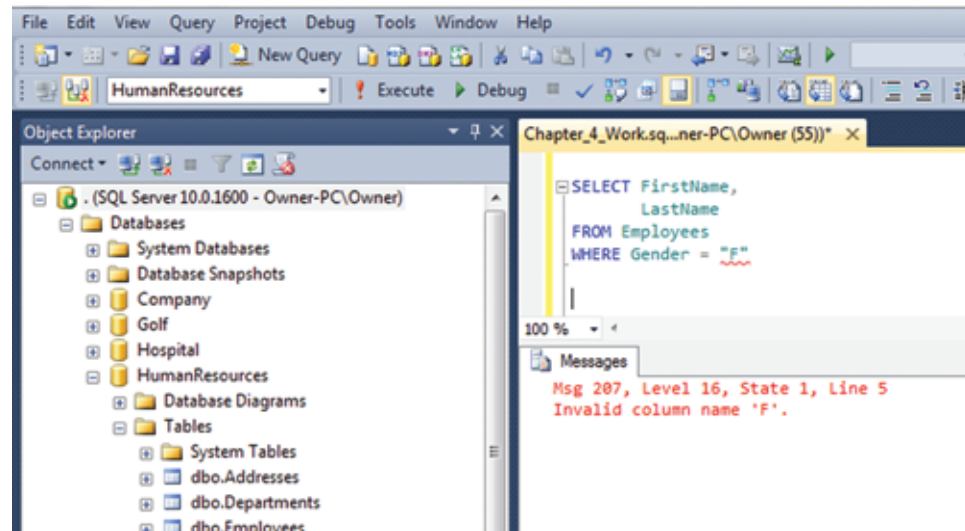


Figure 16

The next WHERE clause example finds rows by filtering on the LastName column.

```
SELECT *
FROM Employees
WHERE LastName = 'Smith'
```

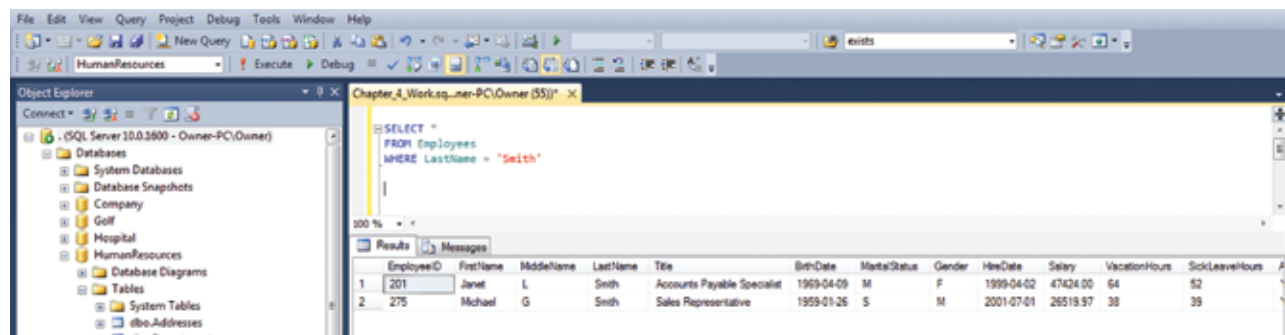


Figure 17

The above SELECT statement executed successfully. However, if we try the same statement for an employee with an apostrophe in their name such as "O'Brien" it would result in an error as shown in Figure 18. This is because SQL Server thinks the ending single quote is after the "O" in "O'Brien" and doesn't know what to do with the third single quote.

```
SELECT *
FROM Employees
WHERE LastName = 'O'Brien'
```

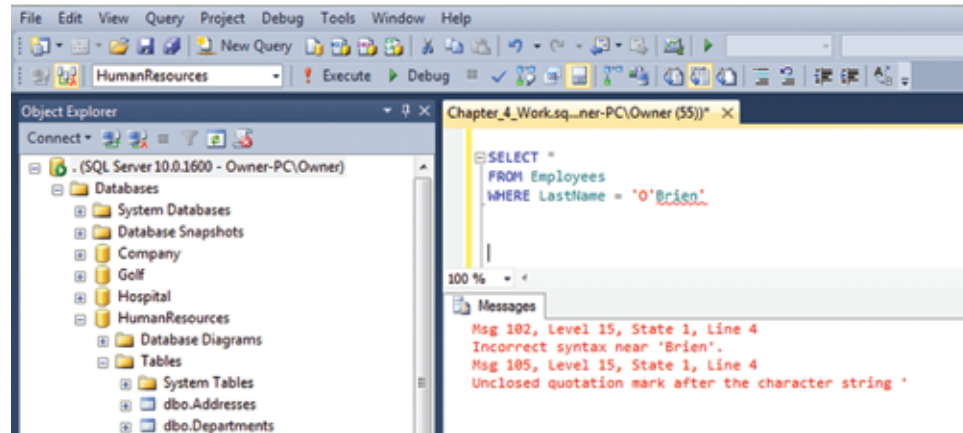


Figure 18

In order to get a search on “O’Brien” to work we need to let SQL Server know that the apostrophe is part of the name and we do this by typing a second apostrophe (or single quote) right after the first so that it looks like ‘O’Brien.’ See Figure 19 for an example of a successful execution searching for “O’Brien.”

```
SELECT *
FROM Employees
WHERE LastName = 'O''Brien'
```

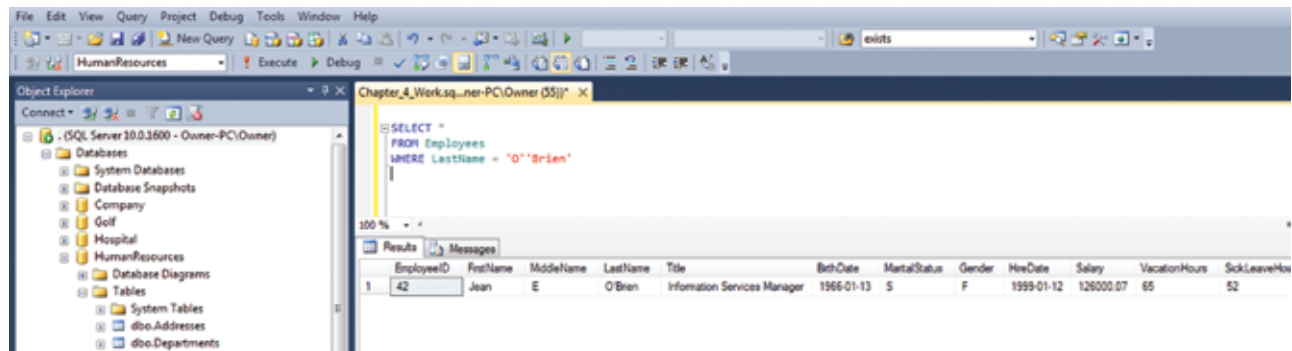


Figure 19

Comparison Operators

Below is a list of comparison operators that can be used with the WHERE clause.

Operator	Meaning
=	equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to
!=	not equal to
<>	not equal to
!>	not greater than
!<	not less than

Comparing Character Data

The meaning of the less than and greater than operators are clear when comparing numeric data. However, it might not be so clear when using these operators for character data such as CHAR and VARCHAR. Figure 20 shows a small result set from retrieving all employees whose last name is less than “Duffy.”

```
SELECT *  
FROM Employees  
WHERE LastName < 'Duffy'
```

The result will not include any last name that has a first letter greater than the letter “D.” The name “Duffy” is not included because we didn’t state equal to. But notice the blue highlight in the result set that contains the name “Dudenhoefer.” This is because the comparison works letter by letter from left to right. The first two letters of “Dudenhoefer” are equal to the first two letters in “Duffy.” However, the third letter in “Dudenhoefer” is less than the third letter in “Duffy” so it is included in the result set.

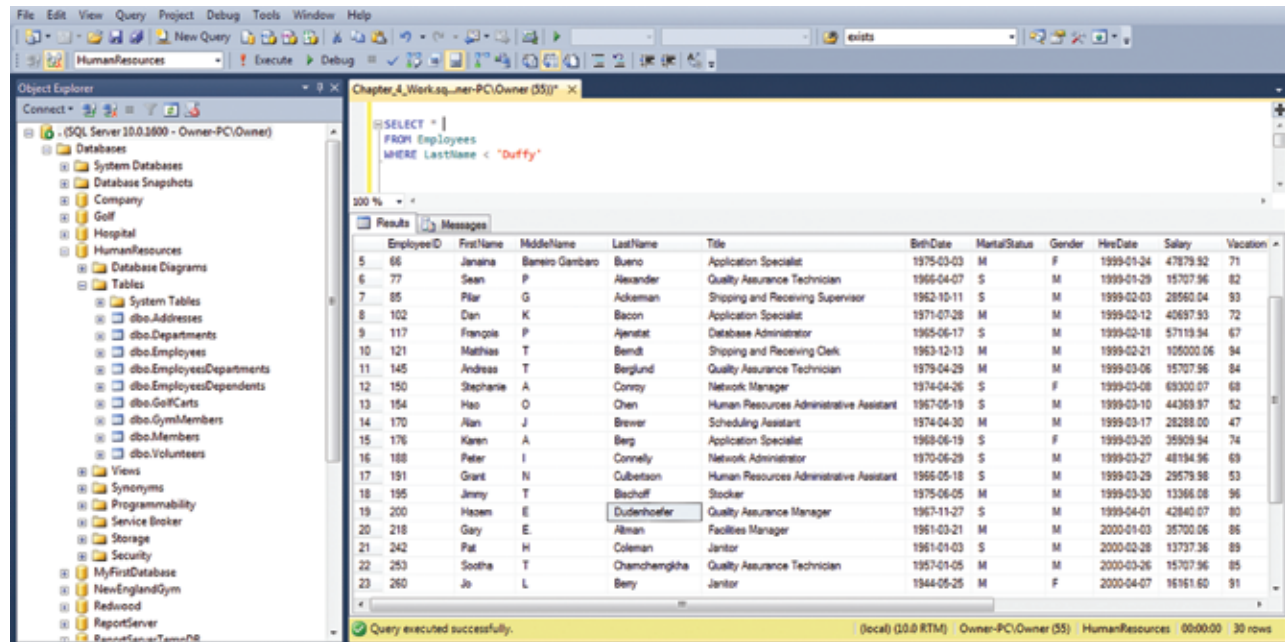


Figure 20

Figure 21 shows the results of finding last names greater than “Duffy.” Notice the name “Dugle” highlighted in blue was included in the result set. That’s because the third letter in “Dugle” is greater than the third letter in “Duffy.”

```
SELECT *
FROM Employees
WHERE LastName > 'Duffy'
```

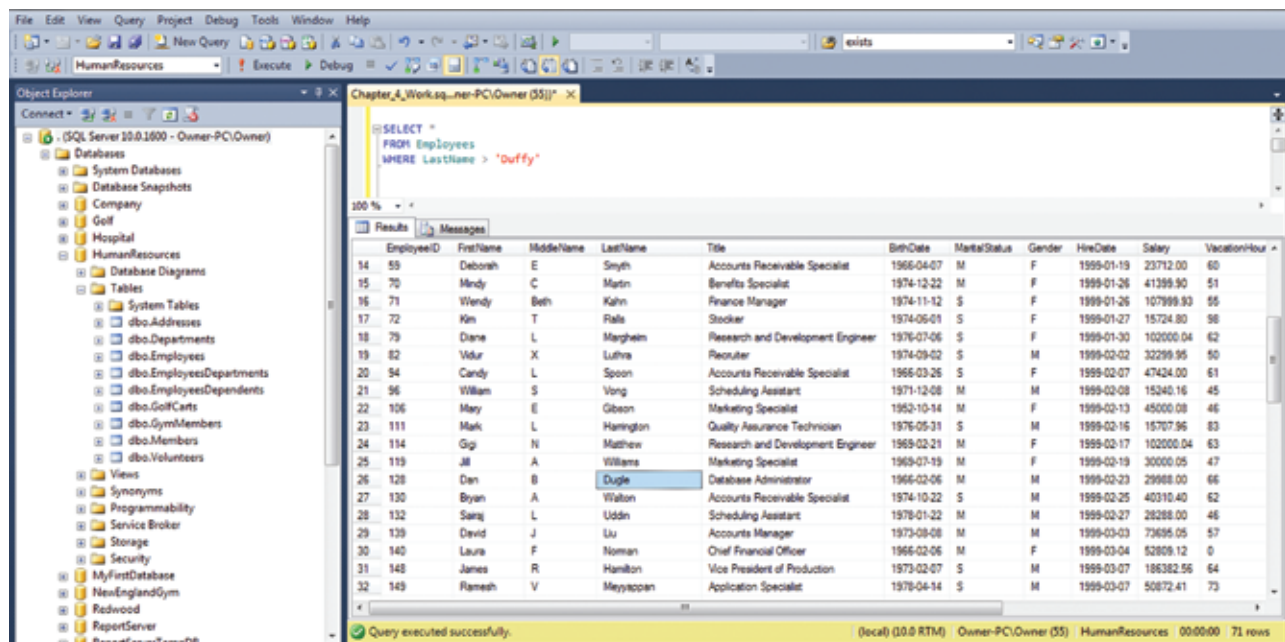


Figure 21

Order By Clause

We can sort our result set on a column or multiple columns of our choosing. Being able to sort the result sets is a significant function to assist end users with the ability to analyze the data. Figure 22 shows an example of a result set sorted by gender.

```
SELECT Gender,
       Title,
       FirstName,
       LastName
FROM Employees
WHERE Salary > 50000
ORDER BY Gender
```

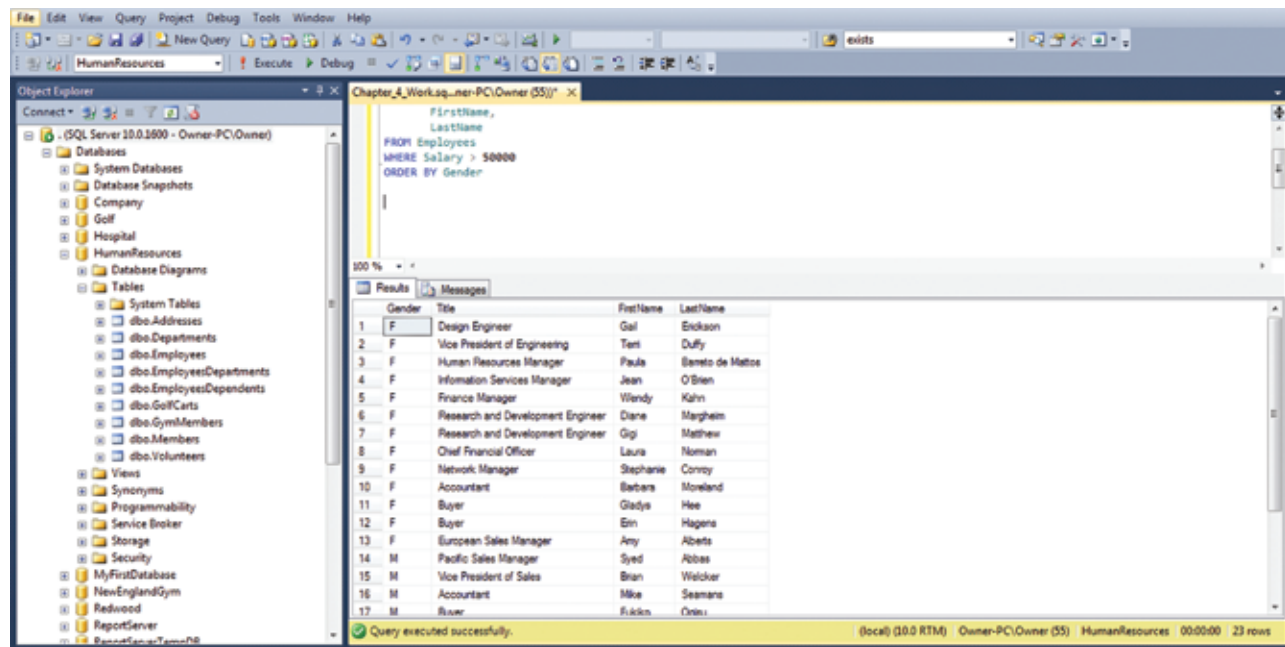


Figure 22

Figure 23 shows an example of using the ORDER BY clause using two columns, gender and title.

```
SELECT Gender,
       Title,
       FirstName,
       LastName
FROM Employees
WHERE Salary > 50000
ORDER BY Gender, Title
```

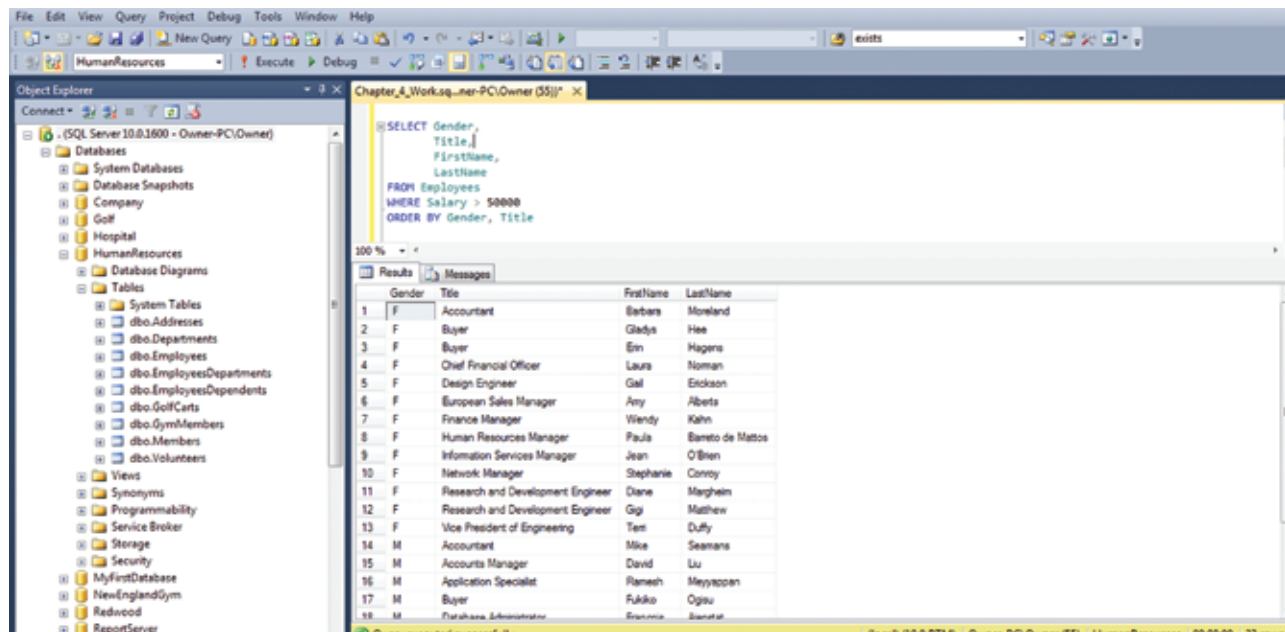


Figure 23

As an option, you can use the numeric representation of the column in the ORDER BY clause instead of the column names. While this reduces typing I would not suggest it if you are going to save the SQL code for future use because it is not immediately clear what the columns are. Having said that below is the SQL statement to produce the same results shown in Figure 23 using column numbers.

```
SELECT Gender,
       Title,
       FirstName,
       LastName
FROM Employees
WHERE Salary > 50000
ORDER BY 1, 2
```


By default, columns are sorted in ascending order. Although unnecessary you can use the ASC clause to achieve the same results shown in Figure 23. Below is the SQL statement using the ASC clause.

```
SELECT Gender,
       Title,
       FirstName,
       LastName
FROM Employees
WHERE Salary > 50000
ORDER BY Gender, Title ASC
```

Columns are sorted in descending order by using the DESC clause. When using multiple columns, you even have the option of sorting each column in ascending or descending order. Figure 24 shows a result set sorting gender by ascending order and title by descending order.

```
SELECT Gender,
       Title,
       FirstName,
       LastName
FROM Employees
WHERE Salary > 50000
ORDER BY Gender ASC, Title DESC
```

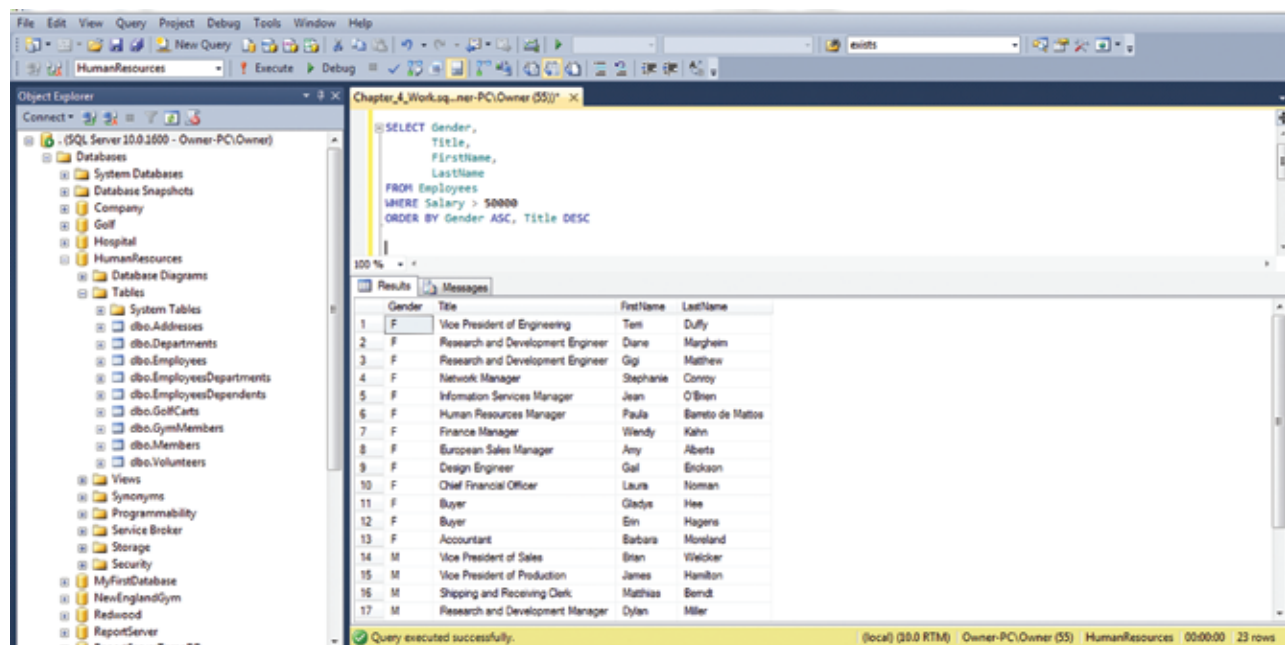


Figure 24

TOP Keyword

The TOP keyword can be used to return a specified number of rows. Figure 25 shows an example of using the TOP keyword to return the first 10 rows from the Employee table.

```
SELECT TOP 10
    FirstName,
    LastName,
    Salary
FROM Employees
```

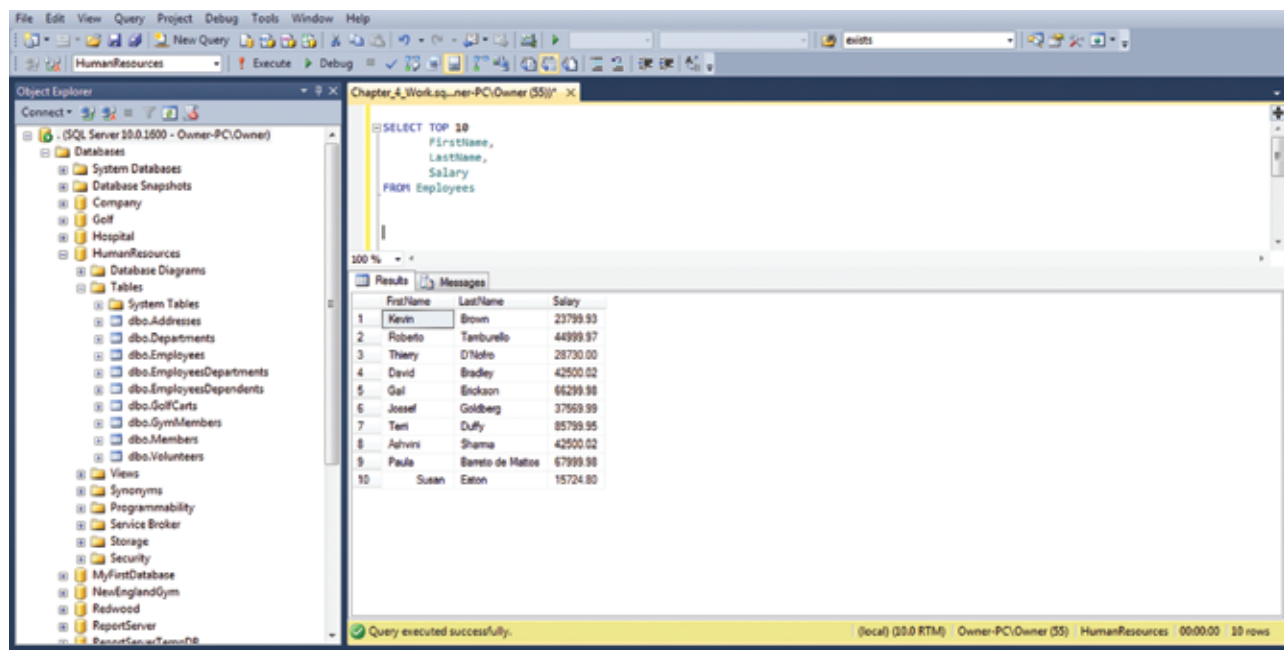


Figure 25

Using the TOP keyword is very useful when selecting, for example, the top two highest salaries in the Employees table. To do so however, salary will need to be in descending order so that the top largest salaries show first in the result set. See Figure 25 for an example of this scenario.

```
SELECT TOP 2
    FirstName,
    LastName,
    Salary
FROM Employees
ORDER BY Salary DESC
```

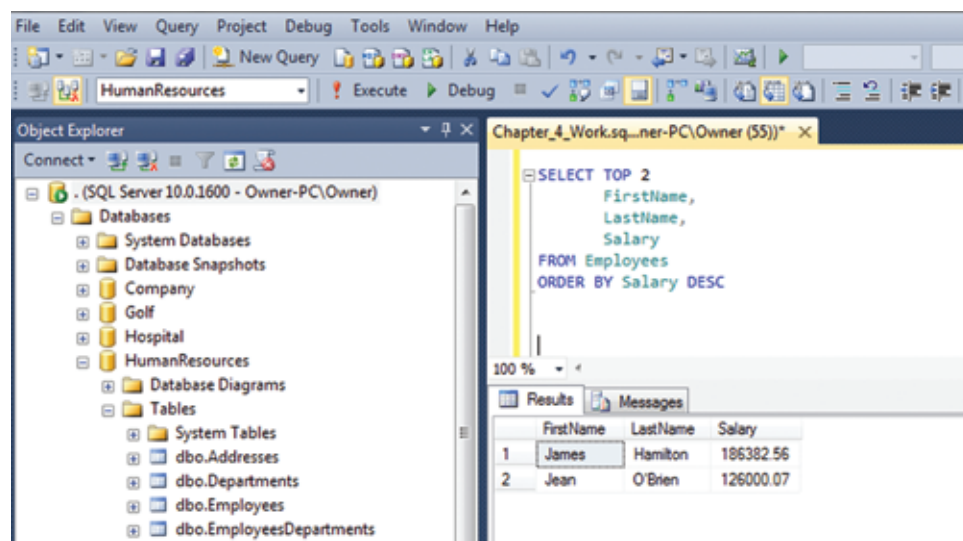


Figure 26

Summary

This chapter introduced you to the SQL query fundamentals using the SELECT statement. You learned how to select all columns of a table using the asterisk as well as select specific columns by including them in the SELECT clause. You learned to filter rows retrieved by using the WHERE clause and to sort them in either ascending or descending order. You also learned the DISTINCT clause to avoid duplicate rows in the result set.

Exercises

1. Select all columns from the Addresses table using the asterisk.
2. Select all columns from the Addresses table using all the columns from the table.
3. Select EmployeeID, AddressLine1, City, State, and Zip1 from the Addresses table.
4. Select EmployeeID, AddressLine1, City, State, and Zip1 from the Addresses table as you did in number 3 above, but this time give alias names of your choosing for EmployeeID, AddressLine1, and Zip1.
5. Use the DISTINCT clause to list all unique states from the Addresses table.
6. Select EmployeeID, AddressLine1, City, State, and Zip1 from the Addresses table, where the state equals California.
7. Select all columns from the Addresses table where the modified date is greater than or equal to 06/21/2000.
8. Select all columns from the Addresses table where the modified date is less than or equal to 06/21/2001. Order the results by EmployeeID.
9. Select all columns from the Addresses table order by State ascending and ModifiedDate descending.
10. Select the top five rows from the Addresses table with the latest modified dates.
11. Select AddressLine1 and Zip1 from the Addresses table and change the column title of AddressLine1 to "Street" and Zip1 to "ZipCode."
12. Select all employees with the last name O'Brien.
13. Select all from the Employees table and find all the last names that are "less than" Miller.
14. Select Gender, Title, FirstName, LastName, and Salary from the Employees table, where salary is less than \$40,000, and order by gender and title.
15. Select EmployeeID, FirstName, LastName, and Salary from the Employees table and find the top 10 highest paid employees and order them using the descending clause.

