

11

Views and Stored Procedures

This chapter will cover views and stored procedures. These are two very important database objects used in standard business practices. They are used for the purposes of providing security and efficiency to both developers and end users.

CHAPTER OBJECTIVES

The chapter objectives for this chapter are as follows:

- Understand the purpose of views.
- Understand how to create, modify, and delete views.
- Understand the purpose of stored procedures.
- Understand how to create, modify, and delete stored procedures.
- Understand how to create parameterized stored procedures.

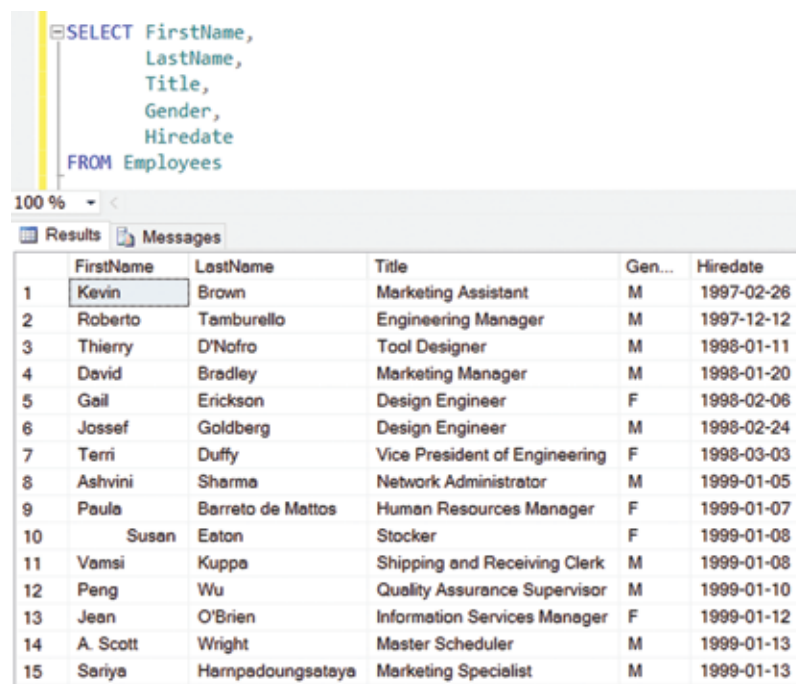
Views

Views are nothing more than virtual tables that are based on queries. You would use a view just like you would a table. Therefore, you can select from it. There are two primary reasons for creating views. One is for security reasons, to prevent end users from accessing confidential information. For example, while employee information such as name, address, and phone number should be accessible to other employees of the company, things like salary, birth date, vacation hours, and sick hours should be kept private. The second is to make access to data simpler for end users and, occasionally, developers which we will cover later.

Create Views

Views are fairly simple to create. To start, let's write out the following query and execute it to make sure it works. Your results should look similar to the results in Figure 1.

```
SELECT FirstName,
       LastName,
       Title,
       Gender,
       Hiredate
FROM Employees
```



```
SELECT FirstName,
       LastName,
       Title,
       Gender,
       Hiredate
FROM Employees
```

	FirstName	LastName	Title	Gen...	Hiredate
1	Kevin	Brown	Marketing Assistant	M	1997-02-26
2	Roberto	Tamburello	Engineering Manager	M	1997-12-12
3	Thierry	D'Nofro	Tool Designer	M	1998-01-11
4	David	Bradley	Marketing Manager	M	1998-01-20
5	Gail	Erickson	Design Engineer	F	1998-02-06
6	Jossef	Goldberg	Design Engineer	M	1998-02-24
7	Terri	Duffy	Vice President of Engineering	F	1998-03-03
8	Ashvini	Sharma	Network Administrator	M	1999-01-05
9	Paula	Barreto de Mattos	Human Resources Manager	F	1999-01-07
10	Susan	Eaton	Stocker	F	1999-01-08
11	Vamsi	Kuppa	Shipping and Receiving Clerk	M	1999-01-08
12	Peng	Wu	Quality Assurance Supervisor	M	1999-01-10
13	Jean	O'Brien	Information Services Manager	F	1999-01-12
14	A. Scott	Wright	Master Scheduler	M	1999-01-13
15	Seriya	Harnpedoungsetaya	Marketing Specialist	M	1999-01-13

Figure 1

Now, to make this query into a view, just add the following statement before the SELECT:

```
CREATE VIEW v_EmployeesInfo  
AS
```

Once the view is created, it will be called v_EmployeesInfo. The “v” at the beginning of the name stands for “view.” Although it isn’t necessary, it’s common practice in the industry to precede database object names with an acronym that helps to identify it. The underscore isn’t necessary either, but once again, it is common practice to use after the acronym and before the name of the view. Since the query this view is based on retrieves employee information, it makes sense to name it v_EmployeesInfo.

Figure 2 shows the statement that creates the view as well as the results once the CREATE statement is executed.

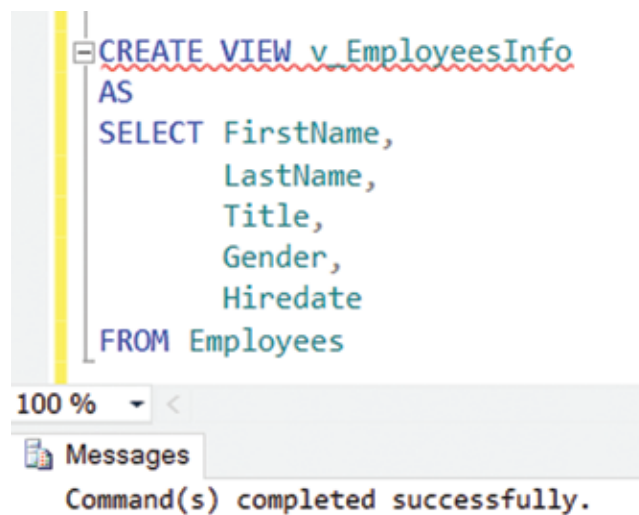


Figure 2

After you create the view, you should see it in your object explorer as shown in Figure 3. If you don't, right click on the HumanResources database and select "Refresh."

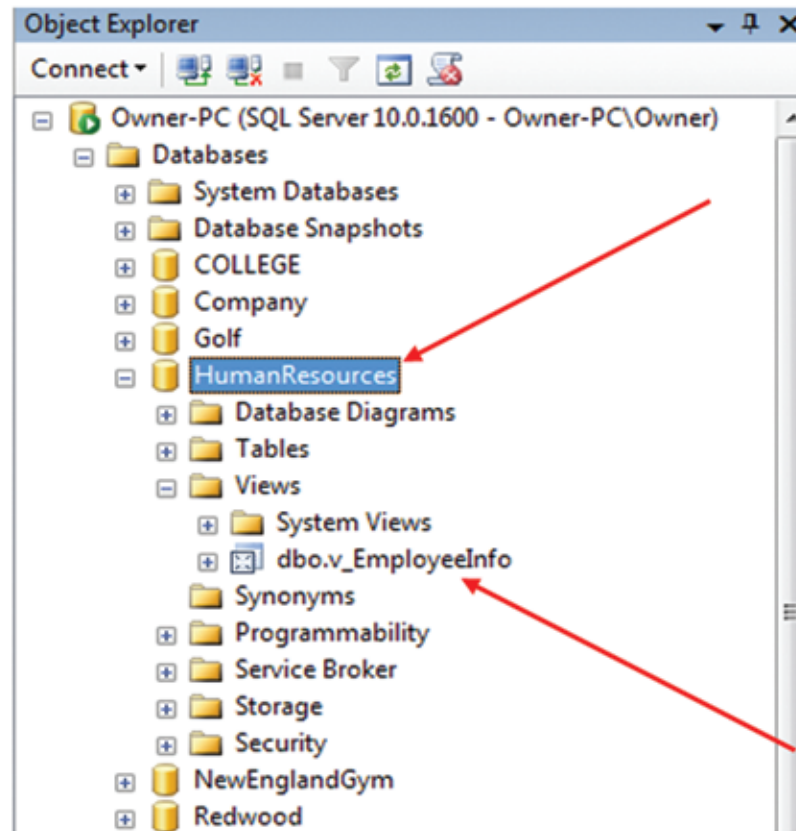


Figure 3

Now that you have created the view, you are free to query from it as if it was a table. An example is shown in Figure 4.

SELECT * FROM v_EmployeesInfo

	FirstName	LastName	Title	Gen...	Hiredate
1	Kevin	Brown	Marketing Assistant	M	1997-02-26
2	Roberto	Tamburello	Engineering Manager	M	1997-12-12
3	Thierry	D'Nofro	Tool Designer	M	1998-01-11
4	David	Bradley	Marketing Manager	M	1998-01-20
5	Gail	Erickson	Design Engineer	F	1998-02-06
6	Josief	Goldberg	Design Engineer	M	1998-02-24
7	Terri	Duffy	Vice President of Engineering	F	1998-03-03
8	Ashvini	Sharma	Network Administrator	M	1999-01-05
9	Paulo	Barreto de Mattos	Human Resources Manager	F	1999-01-07
10	Susan	Eaton	Stocker	F	1999-01-08
11	Vamsi	Kuppa	Shipping and Receiving Clerk	M	1999-01-08
12	Peng	Wu	Quality Assurance Supervisor	M	1999-01-10
13	Jean	O'Brien	Information Services Manager	F	1999-01-12
14	A. Scott	Wright	Master Scheduler	M	1999-01-13
15	Seriya	Harnpedoungsetaya	Marketing Specialist	M	1999-01-13

Figure 4

Notice that the only columns available in the view are the ones we used in the query. Restricted information such as salary and vacation hours are not available from the view.

Note: You may be experiencing red underlines under new data objects, such as the red squiggly line in Figure 5. This is because the database editor doesn't recognize the new view as available in the database, even though it is now available. However, if you close the database and start it again, those underlines would disappear. This may be an oversight on the manufacturer when SQL Server was developed.

```
CREATE VIEW v_EmployeesInfo
AS
SELECT FirstName,
       LastName,
       Title,
       Gender,
       Hiredate
FROM Employees

SELECT * FROM v_EmployeesInfo
```

Figure 5

ALTER VIEW

Now that we've created a view, let's modify it. Let's say that the end users using this view would like to see the manager ID included in the view. In addition, the end users only want to see employees that are "active". The view can be modified using the ALTER statement as follows:

```
ALTER VIEW v_EmployeesInfo
AS
SELECT FirstName,
       LastName,
       Title,
       Gender,
       Hiredate,
       ManagerID
FROM Employees
WHERE Active = 'Yes'
```

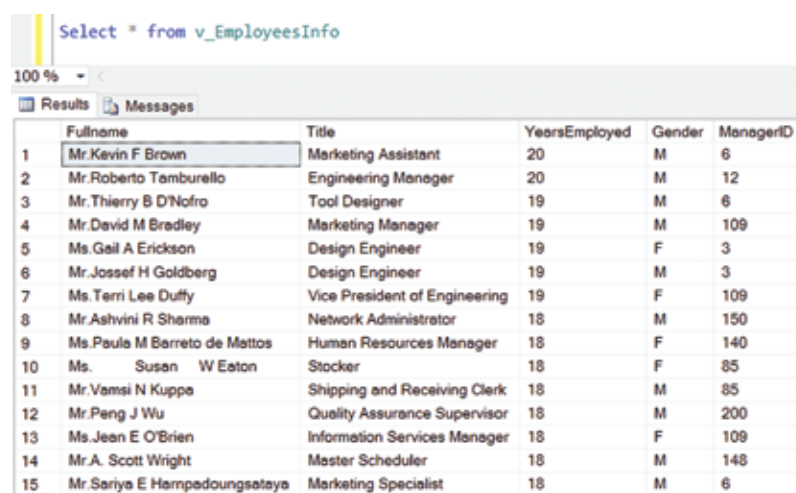
Go ahead and execute the above statement so that ManagerID is now included in the view and only active members are included. If a current active member's status changes to "not active" in the future, the view will reflect that change by not including it in the result set. This is because the view is based on the table that contains that information.

Earlier, I mentioned the primary reasons for creating views. One was for security reasons which we just discussed. The second is to make access to data simpler for end users and, occasionally, developers. Sometimes queries can be quite complicated, especially when multiple tables are involved. You are about to learn how we can use views to simplify data retrieval for end users.

Let's say our end users need to retrieve the full formal names of employees as well as the number of years employed. Well, this takes some work. But once we put the SQL code in a view, it will be easy for end users to access. Let's change the query in our view so that it looks like the one below.

```
ALTER VIEW v_EmployeesInfo
AS
SELECT CASE Gender
    WHEN 'F' THEN 'Ms. '
    ELSE 'Mr. '
END +
CASE
    WHEN MiddleName IS NULL THEN FirstName + ' ' + LastName
    ELSE FirstName + ' ' + MiddleName + ' ' + LastName
END 'FullName',
Title,
Gender,
DateDiff(YEAR, Hiredate, GETDATE()) 'YearsEmployed',
ManagerID
FROM Employees
WHERE Active = 'Yes'
```

As you can see, we are using the CASE statement and the DateDiff() function that we learned in chapter 10. The end user will never have to learn or write that code. Instead, he or she can simply retrieve the required results from the view as shown in Figure 6.



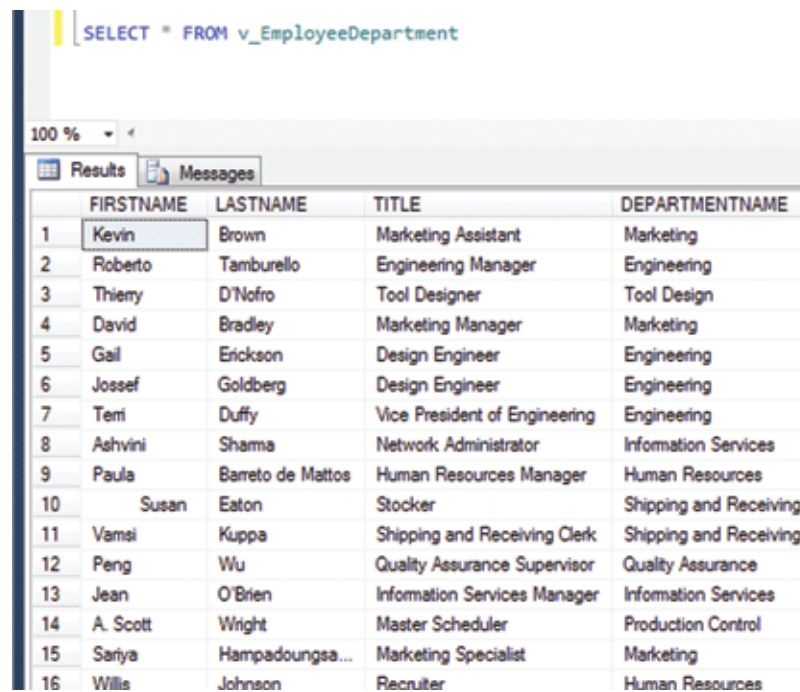
	Fullname	Title	YearsEmployed	Gender	ManagerID
1	Mr Kevin F Brown	Marketing Assistant	20	M	6
2	Mr Roberto Tamburello	Engineering Manager	20	M	12
3	Mr. Thierry B D'Nofo	Tool Designer	19	M	6
4	Mr. David M Bradley	Marketing Manager	19	M	109
5	Ms. Gail A Erickson	Design Engineer	19	F	3
6	Mr. Jossef H Goldberg	Design Engineer	19	M	3
7	Ms. Terri Lee Duffy	Vice President of Engineering	19	F	109
8	Mr. Ashvini R Sharma	Network Administrator	18	M	150
9	Ms. Paule M Barreto de Mattos	Human Resources Manager	18	F	140
10	Ms. Susan W Eaton	Stocker	18	F	85
11	Mr. Vamsi N Kuppe	Shipping and Receiving Clerk	18	M	85
12	Mr. Peng J Wu	Quality Assurance Supervisor	18	M	200
13	Ms. Jean E O'Brien	Information Services Manager	18	F	109
14	Mr. A. Scott Wright	Master Scheduler	18	M	148
15	Mr. Sariya E Hampedoungsataya	Marketing Specialist	18	M	6

Figure 6

Another example where views can make data retrieval simpler is when we need to join tables. Let's say a user wants to see the names of employees, their titles, and current departments they are assigned to. The code below will create the view to retrieve this.

```
CREATE VIEW v_EmployeeDepartment
AS
SELECT FIRSTNAME,
        LASTNAME,
        TITLE,
        DEPARTMENTNAME
FROM Employees E
INNER JOIN EmployeesDepartments ED on E.EmployeeID = ED.EmployeeId
INNER JOIN Departments D on D.DepartmentID = ED.DepartmentID
WHERE EndDate IS NULL
```

The ending results are shown in Figure 7.

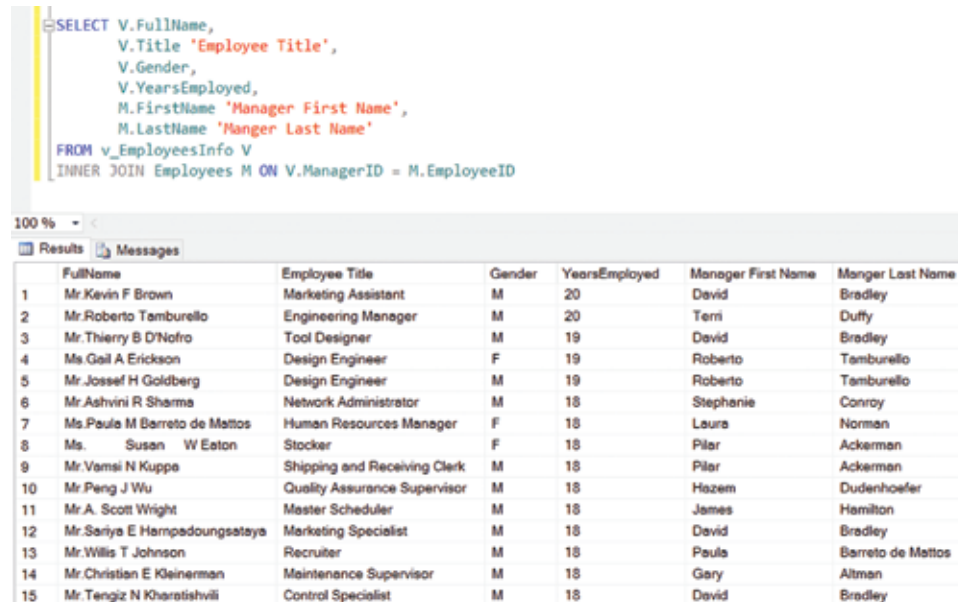


	FIRSTNAME	LASTNAME	TITLE	DEPARTMENTNAME
1	Kevin	Brown	Marketing Assistant	Marketing
2	Roberto	Tamburello	Engineering Manager	Engineering
3	Thierry	D'Nofro	Tool Designer	Tool Design
4	David	Bradley	Marketing Manager	Marketing
5	Gail	Erickson	Design Engineer	Engineering
6	Josief	Goldberg	Design Engineer	Engineering
7	Teri	Duffy	Vice President of Engineering	Engineering
8	Ashvini	Sharma	Network Administrator	Information Services
9	Paula	Barreto de Mattos	Human Resources Manager	Human Resources
10	Susan	Eaton	Stocker	Shipping and Receiving
11	Vamsi	Kuppa	Shipping and Receiving Clerk	Shipping and Receiving
12	Peng	Wu	Quality Assurance Supervisor	Quality Assurance
13	Jean	O'Brien	Information Services Manager	Information Services
14	A. Scott	Wright	Master Scheduler	Production Control
15	Sariya	Hampadounsa...	Marketing Specialist	Marketing
16	Willis	Johnson	Recruiter	Human Resources

Figure 7

Joining Views and Tables

Because views are treated like virtual tables, you can join them with other tables. Let's try that by joining to the Employees table to retrieve the managers' names and titles, since only the manager ID's are included in our view. The code and results are shown in Figure 8.



```

SELECT V.FullName,
       V.Title 'Employee Title',
       V.Gender,
       V.YearsEmployed,
       M.FirstName 'Manager First Name',
       M.LastName 'Manger Last Name'
FROM v_EmployeesInfo V
INNER JOIN Employees M ON V.ManagerID = M.EmployeeID

```

	FullName	Employee Title	Gender	YearsEmployed	Manager First Name	Manger Last Name
1	Mr Kevin F Brown	Marketing Assistant	M	20	David	Bradley
2	Mr Roberto Tamburello	Engineering Manager	M	20	Terri	Duffy
3	Mr Thierry B D'Nofo	Tool Designer	M	19	David	Bradley
4	Ms Gail A Erickson	Design Engineer	F	19	Roberto	Tamburello
5	Mr Joesef H Goldberg	Design Engineer	M	19	Roberto	Tamburello
6	Mr Ashvini R Sharma	Network Administrator	M	18	Stephanie	Conroy
7	Ms Paula M Barreto de Mattos	Human Resources Manager	F	18	Laura	Norman
8	Ms. Susan W Eaton	Stocker	F	18	Pilar	Ackerman
9	Mr Vamsi N Kuppe	Shipping and Receiving Clerk	M	18	Pilar	Ackerman
10	Mr Peng J Wu	Quality Assurance Supervisor	M	18	Hazem	Dudenhofer
11	Mr A. Scott Wright	Master Scheduler	M	18	James	Hamilton
12	Mr Saniya E Hampedoungsetays	Marketing Specialist	M	18	David	Bradley
13	Mr Willis T Johnson	Recruiter	M	18	Paula	Barreto de Mattos
14	Mr Christian E Kleiner	Maintenance Supervisor	M	18	Gary	Altman
15	Mr Tengiz N Kharatishvili	Control Specialist	M	18	David	Bradley

Figure 8

A very simple join from the view to our Employees table was all it took to retrieve the managers' first and last names. While this was simple enough, perhaps the view would be better if it already contained the managers' first and last names along with the manager ID's. This is an example of the type of analysis and consideration you will experience when creating views for your end users.

Dropping Views

If views are no longer needed, we can remove them from the database simply by issuing a DROP statement as shown in Figure 9.

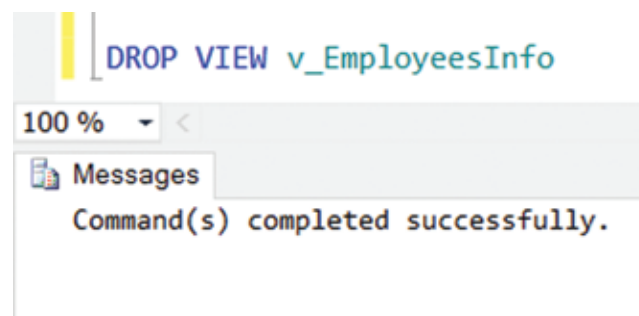


Figure 9

Stored Procedures

Stored procedures, like views, are database objects. However, because they are compiled and become part of the database execution plan, they perform much faster than views or issuing SQL statements. They are often used by developers who code Windows, Web and Mobile applications that need to perform database activity such as querying, inserting, updating, and deleting data. We will be working on stored procedures that query data in this chapter.

Let's create our first stored procedure. Much like we did with our view earlier, we do this with a simple SELECT statement, as follows:

```
CREATE PROCEDURE sp_EmployeesInfo
AS
SELECT FirstName,
        LastName,
        Title,
        Gender,
        Hiredate
FROM Employees
```

There are only two differences between the first view we created earlier and the stored procedure statement above. Instead of "CREATE VIEW", we write "CREATE PROCEDURE." Then, of course, we should change the name of the object to reflect that it's a stored procedure. As a standard, "sp" is used to represent that it's a stored procedure. Therefore, the full name of the stored procedure above is "sp_EmployeesInfo."

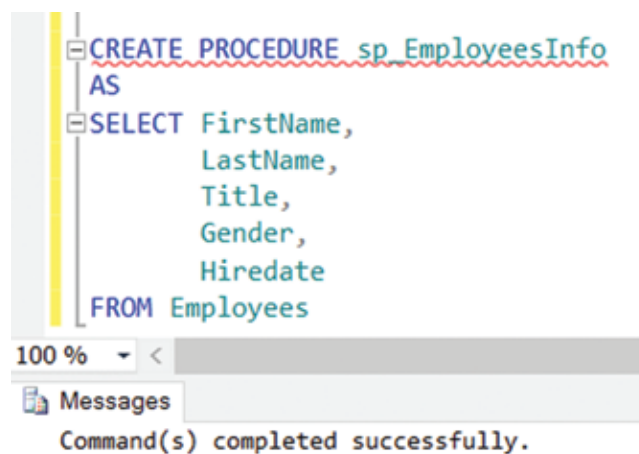


Figure 10

Once you've created your stored procedure, you can find it in the browser under "Programmability" and then "Stored Procedures". See Figure 11 as a reference to assist you in locating your stored procedure.

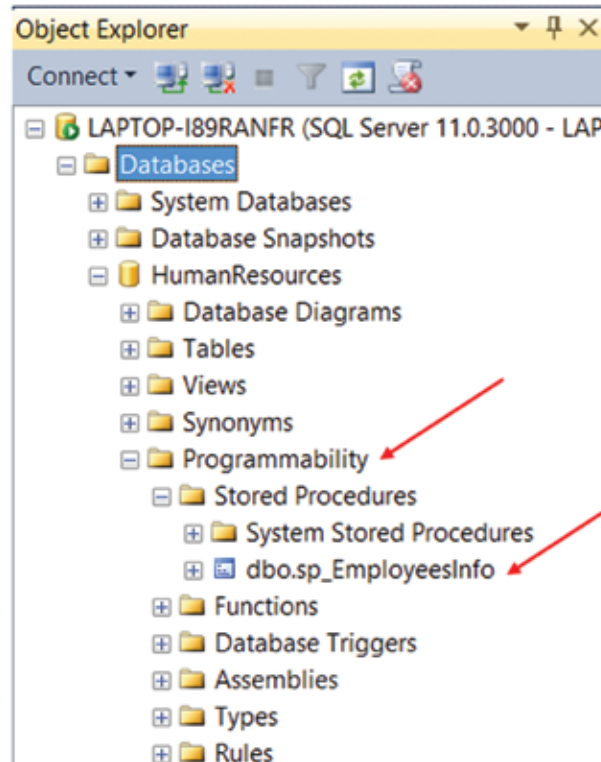


Figure 11

To execute the stored procedure, we can use one of two statements listed below.

- EXECUTE
- EXEC

Either one of these statements will work. So, let's try it out. Your results should look like those in Figure 12.

EXEC sp_EmployeesInfo

100 %

Results Messages

	FirstName	LastName	Title	Gender	Hiredate
1	Kevin	Brown	Marketing Assistant	M	1997-02-26
2	Roberto	Temburello	Engineering Manager	M	1997-12-12
3	Thierry	D'Nofro	Tool Designer	M	1998-01-11
4	David	Bradley	Marketing Manager	M	1998-01-20
5	Gail	Erickson	Design Engineer	F	1998-02-06
6	Jossef	Goldberg	Design Engineer	M	1998-02-24
7	Terri	Duffy	Vice President of Engineering	F	1998-03-03
8	Ashvini	Sharma	Network Administrator	M	1999-01-05
9	Paula	Berreto de Mattos	Human Resources Manager	F	1999-01-07
10	Susan	Eaton	Stocker	F	1999-01-08
11	Vamsi	Kuppa	Shipping and Receiving Clerk	M	1999-01-08
12	Peng	Wu	Quality Assurance Supervisor	M	1999-01-10
13	Jean	O'Brien	Information Services Manager	F	1999-01-12
14	A. Scott	Wright	Master Scheduler	M	1999-01-13
15	Sariya	Hampedoungsa...	Marketing Specialist	M	1999-01-13

Figure 12

ALTER Stored Procedures

We can alter stored procedures, much like we did views, by using the ALTER statement. Figure 13 shows us the statement to alter our stored procedure so that it selects only active employees.

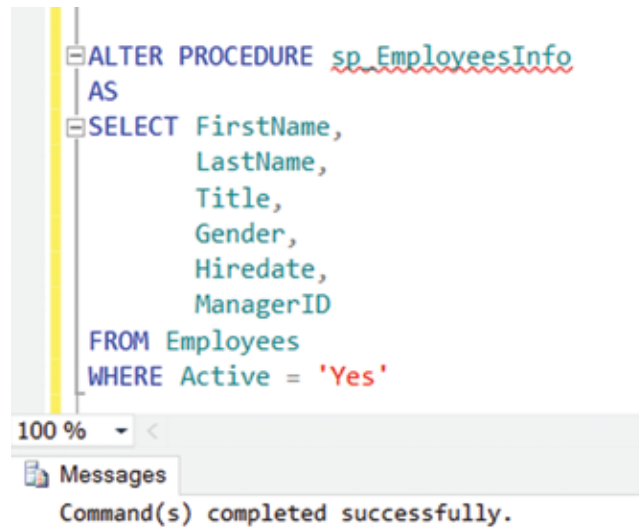


Figure 13

Parameters

Unlike views, stored procedures can accept parameters so that the results can be more specific to the end users' needs. For parameters to be accepted by the stored procedure, it needs to be created or altered to do so.

Let's change our stored procedure so that instead of finding all employees that are active, we find the employee whose ID has been entered as a parameter. Below is the code to alter our stored procedure.

```
ALTER PROCEDURE sp_EmployeesInfo (@EmployeeID int)
AS
SELECT FirstName,
        LastName,
        Title,
        Gender,
        Hiredate,
        ManagerID
FROM Employees
WHERE EmployeeID = @EmployeeID
```

Two red arrows point to the parameter definition `(@EmployeeID int)` in the first line and the variable `@EmployeeID` in the `WHERE` clause of the last line.

Notice the parameter is placed after the name of the stored procedure within parenthesis. Also, the “@” sign must precede the parameter name. The parameter name can be any name you want, but it makes sense to keep the name consistent with the column name you will be comparing it to in the WHERE clause. The “@” sign will be enough to differentiate it from the column name.

Also, notice that the data type is required. EmployeeID is an integer, so our parameter should be an integer as well.

The parameter is then used in the WHERE clause to find the specific employee that was requested. Figure 14 shows the results of executing the stored procedure with a parameter.



	FirstName	LastName	Title	Gender	Hiredate	ManagerID
1	Roberto	Tamburello	Engineering Manager	M	1997-12-12	12

Figure 14

Multiple parameters can be used in stored procedures. For example, let's say we wanted to change our stored procedure so that it retrieved employees based on the gender and manager ID that was entered as parameters. The code to do this is as follows:

```
ALTER PROCEDURE sp_EmployeesInfo (@Gender char(1), @ManagerID int)
AS
SELECT FirstName,
       LastName,
       Title,
       Gender,
       Hiredate,
       ManagerID
FROM Employees
WHERE Gender = @Gender
AND ManagerID = @ManagerID
```

In the code above, notice that a comma separates the two parameters. When the stored procedure is executed, the parameters entered will require separation by commas as well, as shown in Figure 14. Notice, also, that one parameter is a character data type and needs single quotes, while the other parameter is an integer and does not need quotes.



	FirstName	LastName	Title	Gender	Hiredate	ManagerID
1	Gail	Erickson	Design Engineer	F	1998-02-06	3
2	Sharon	Salavaria	Design Engineer	F	2001-02-18	3

Figure 15

Dropping Stored Procedures

Like views, stored procedures can easily be removed from the database using the DROP statement. Figure 16 shows the use of the DROP statement and the resulting message from SQL Server.

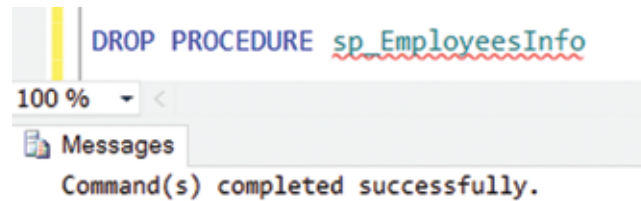


Figure 16

Summary

This chapter introduced you to the fundamentals of views and stored procedures. It demonstrated the creation, alteration, and deletion of views and stored procedures. It showed how these database objects can be efficient resources for both end users and developers.

This chapter also explained the fact that views are nothing more than virtual tables based on queries and the two primary reasons for creating views are for security reasons and ease of access to data. The reasons for creating stored procedures include the fact that they're much faster to execute than views and they're often used by developers who code Windows, Web and Mobile applications that need to query, insert, update, or delete data.

Furthermore, this chapter covered how to create or alter a stored procedure with parameters so that results can be more specific to the end users' needs.

Exercises

1. Create a view showing the formal name of employees (Example: Mr. John P Smith) and full address.
2. Alter the view in exercise 1 to include the age of the employee.
3. Join the view from exercise 2 to the necessary tables to retrieve the department name.
4. Create a stored procedure to retrieve the first name, last name, and department name for each employee.
5. Alter the stored procedure from exercise 4 to allow a parameter to retrieve employees by Department ID.
6. Create a stored procedure to list employee first names, last names, titles, and genders. Allow parameters for title and gender.
7. Create a stored procedure listing the full address from the address table where Zip1 is equal to the parameter entered.
8. Create a stored procedure to show the first name, last name, salary, and a bonus based on salary percentage entered as a parameter.
9. Create a view showing the first name, last name, and the year each employee was born.
10. Create a view showing the first name, last name, and hire date of each employee. Format the hire date as mm/dd/yyyy.
11. Create a stored procedure using the LIKE operator to search employee first and last names for those containing a specified parameter.
12. Create a stored procedure that displays employees with salaries between two integer parameters. Order by salary.
13. Create a stored procedure showing member names and the golf cart information for any golf carts they have checked out, search members' names using the LIKE operator, and accepting a search term as a parameter. (Hint: An outer join will be necessary to show golf cart information.)
14. Create a stored procedure to list the count of employees in a state entered as a parameter.
15. Create a stored procedure showing volunteer information where first and last name are entered as parameter