CHAPTER 13

# Views

Contrary to popular opinion, a view isn't something you see out of a window. Not in database terms, anyway. I'm afraid it's much more mundane! A view in a database can be thought of as a virtual table. It allows you to view data in a particular way, by utilizing other database objects.

You may not be aware of it, but we've already done most of the hard work involved in creating views by understanding how to use the SELECT statement and the various joins available to us. We'll create a couple of useful views to improve our database.

## What Are Views For?

I've already mentioned that a view can be thought of as a virtual table. You may be debating why you would need virtual tables when you have real ones. There are a few reasons:

- **Efficiency**: You can display data from multiple tables and not write the same queries over and over again.
- **Performance**: You can cache the query a view uses, and you can even create something called an *indexed view*, which further improves speed.
- **Security**: You can limit the amount of data certain users can see.
- **Less coding**: Views reduce the amount of code you need to write and can be referenced in other queries using joins.

Sometimes a view just makes things easier. Think of all those SELECT statements we were writing earlier. Some of those were eminently reusable. Had we enclosed them in views, we wouldn't need to reenter the code, and that view would be available to all users of our database.

## Creating Views

Creating views is one of the simpler things you can do in SQL Server. You use the CREATE VIEW statement, which at its most basic looks like this:

```
CREATE VIEW ViewName AS SELECT...
```

We'll create a view to return verified contacts. Open a New Query Window and enter this script:

```
USE AddressBook;

CREATE VIEW dbo.VerifiedContacts AS
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, CASE C.AllowContactByPhone WHEN 1 THEN

GO
```

Press F5 to run this. The script fails! The error is:

```
Msg 111, Level 15, State 1, Line 3
'CREATE VIEW' must be the first statement in a query batch.
```

All this means is we need to put a GO after the USE AddressBook line. The batch has to begin with CREATE VIEW.

After adding the GO, run this again. You should now see **Command(s) completed successfully**. But no data is displayed at all. Did you think the SELECT statement would run? That's not what you've done here. You've created a database object, just like you did when we created our tables all those chapters ago.

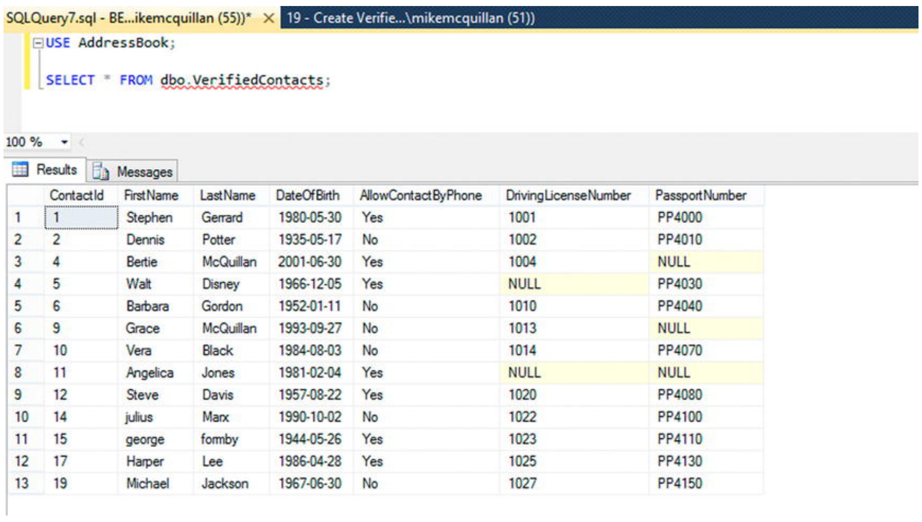
Save this script as `c:\temp\sqlbasics\apply\19 - Create VerifiedContacts View.sql`. Don't forget to add it to the end of our `00 - Apply.sql` script, too:

```
:setvar currentFile "19 - Create VerifiedContacts View.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)
```

Now you can use the view just like you would use any other table. Open a New Query Window and type this query:

```
USE AddressBook;
SELECT * FROM dbo.VerifiedContacts;
```

As you can see in Figure 13-1, just the verified contacts are returned, and the columns we specified are displayed.


SQLQuery7.sql - BE...ikemcquillan (55))*
19 - Create Verifie...\mikemcquillan (51))
100 %
Results
Messages

```
USE AddressBook;

SELECT * FROM dbo.VerifiedContacts;
```

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | DrivingLicenseNumber | PassportNumber |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | Yes | 1001 | PP4000 |
| 2 | 2 | Dennis | Potter | 1935-05-17 | No | 1002 | PP4010 |
| 3 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | 1004 | NULL |
| 4 | 5 | Walt | Disney | 1966-12-05 | Yes | NULL | PP4030 |
| 5 | 6 | Barbara | Gordon | 1952-01-11 | No | 1010 | PP4040 |
| 6 | 9 | Grace | McQuillan | 1993-09-27 | No | 1013 | NULL |
| 7 | 10 | Vera | Black | 1984-08-03 | No | 1014 | PP4070 |
| 8 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL | NULL |
| 9 | 12 | Steve | Davis | 1957-08-22 | Yes | 1020 | PP4080 |
| 10 | 14 | julius | Marx | 1990-10-02 | No | 1022 | PP4100 |
| 11 | 15 | george | formby | 1944-05-26 | Yes | 1023 | PP4110 |
| 12 | 17 | Harper | Lee | 1986-04-28 | Yes | 1025 | PP4130 |
| 13 | 19 | Michael | Jackson | 1967-06-30 | No | 1027 | PP4150 |

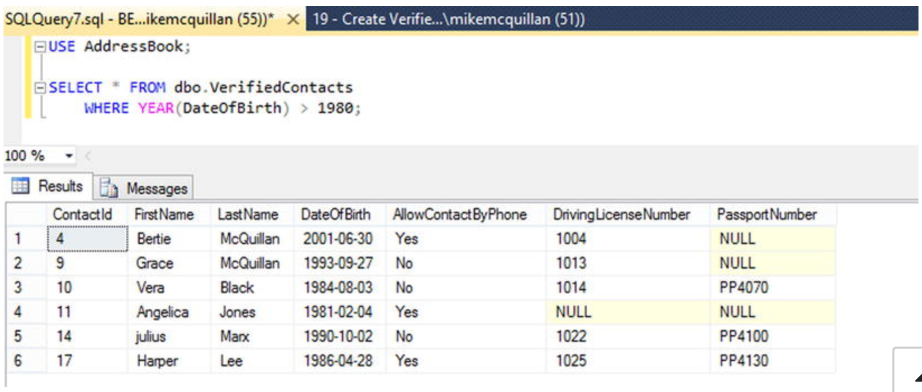
***Figure 13-1.*** *Returning everything from the* `VerifiedContacts` *view*

`dbo.VerifiedContacts` is underlined in Figure 13-1. This is because Intellisense—the technology built into SSMS to auto-complete table names and so on as you type—doesn't know the view exists yet.

**REFRESHING INTELLISENSE**

You can refresh your local Intellisense cache immediately by pressing Ctrl+Shift+R on your keyboard, or clicking **Edit ➤ Intellisense ➤ Refresh Local Cache**.

The view has worked exactly as a table would. As Figure 13-2 shows, we can even filter the view.


SQLQuery7.sql - BE...ikemcquillan (55))*
19 - Create Verifie...\mikemcquillan (51))
100 %
Results
Messages

```
USE AddressBook;

SELECT * FROM dbo.VerifiedContacts
    WHERE YEAR(DateOfBirth) > 1980;
```

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | DrivingLicenseNumber | PassportNumber |
|---|---|---|---|---|---|---|---|
| 1 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | 1004 | NULL |
| 2 | 9 | Grace | McQuillan | 1993-09-27 | No | 1013 | NULL |
| 3 | 10 | Vera | Black | 1984-08-03 | No | 1014 | PP4070 |
| 4 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL | NULL |
| 5 | 14 | julius | Marx | 1990-10-02 | No | 1022 | PP4100 |
| 6 | 17 | Harper | Lee | 1986-04-28 | Yes | 1025 | PP4130 |

***Figure 13-2.*** *Filtering the* `VerifiedContacts` *view*

### Adding Columns to Views

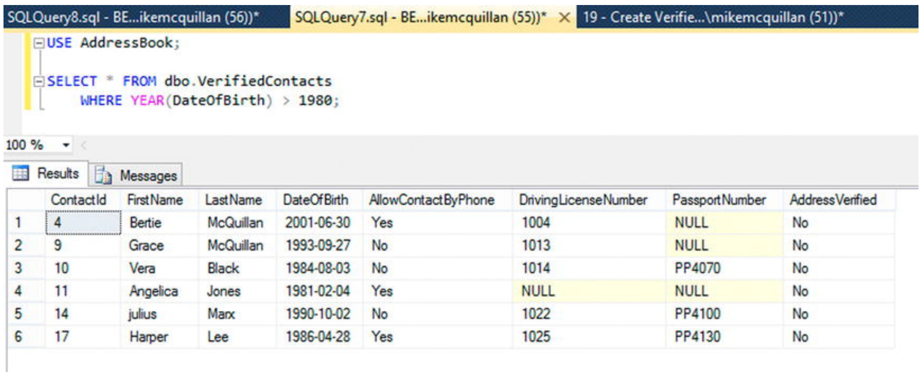
Let's say we need to add a new column, `AddressVerified`, to the `ContactVerificationDetails` table, to hold a flag denoting if a contact's address has been confirmed. Executing this script in a New Query Window adds the column without difficulty:

```
ALTER TABLE dbo.ContactVerificationDetails ADD AddressVerified  BIT NULL;
```

If you run the `SELECT` statement against the view again, you'll note the new column doesn't appear. Return to the `CREATE VIEW` script and change it to an `ALTER VIEW` statement, with the new column included in a `CASE` statement:

```
USE AddressBook;

GO

ALTER VIEW dbo.VerifiedContacts
AS
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, CASE C.AllowContactByPhone WHEN 1 THEN
CASE CVD.AddressVerified WHEN 1 THEN 'Yes' ELSE 'No' END AS AddressVerified FROM dbo.Contacts C INNER JOI

GO
```

Run this and return to the `SELECT` statement, which will now display the column (you can see it in Figure 13-3).



```
USE AddressBook;

SELECT * FROM dbo.VerifiedContacts
    WHERE YEAR(DateOfBirth) > 1980;
```

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | DrivingLicenseNumber | PassportNumber | AddressVerified |
|---|---|---|---|---|---|---|---|---|
| 1 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | 1004 | NULL | No |
| 2 | 9 | Grace | McQuillan | 1993-09-27 | No | 1013 | NULL | No |
| 3 | 10 | Vera | Black | 1984-08-03 | No | 1014 | PP4070 | No |
| 4 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL | NULL | No |
| 5 | 14 | julius | Marx | 1990-10-02 | No | 1022 | PP4100 | No |
| 6 | 17 | Harper | Lee | 1986-04-28 | Yes | 1025 | PP4130 | No |

***Figure 13-3.*** *The updated `VerifiedContacts` view*

### Checking If a View Already Exists

`ALTER VIEW` makes it easy for you to change an existing view. However, using `ALTER VIEW` will fail should we rebuild our database; the view has to exist for `ALTER VIEW` to work, and the view will not exist during a database rebuild, as no `CREATE VIEW` command has been exercised.

Fortunately, our old friends, system views, come to our rescue yet again. Modify script `19 - Create VerifiedContacts View.sql` to check if the view exists, and restore the `CREATE VIEW` statement while you are there, too.

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.views WHERE [Name] = 'VerifiedContacts')
BEGIN
DROP VIEW dbo.VerifiedContacts;
END;

GO

CREATE VIEW dbo.VerifiedContacts
AS
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, CASE C.AllowContactByPhone WHEN 1 TH
CASE CVD.AddressVerified WHEN 1 THEN 'Yes' ELSE 'No' END AS AddressVerified FROM dbo.Contacts C INN
```

```
GO
```

We use the `sys.views` system view to check if the view exists, in the same way we checked if a table existed. The `DROP VIEW` command will drop the view if the view exists. `DROP VIEW` can be used to drop any view, and it will not prompt you when doing so. Be careful!

Save this script, rerun the view, and rerun your `SELECT` statement. Everything should be hunky-dory.

**CAN I USE VIEWS TO INSERT AND UPDATE DATA?**

You can use views should you wish to insert and update data. However, I recommend against doing this. It leads to confusion. It isn't too bad if a view only represents one table, but usually a view joins multiple tables together. Which table should the view insert to if there are multiple tables present?

Just use views to `SELECT` data from your tables. Think of them as read-only tables and use them for their intended purpose!

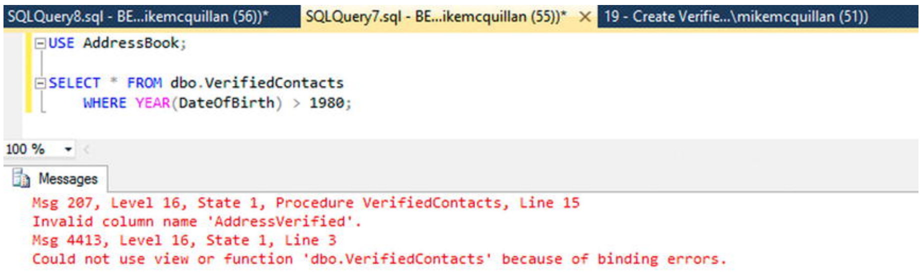
## Broken Views

Some bright spark has decided we don't need an `AddressVerified` column in the `ContactVerificationDetails` table after all. Some people! Not to worry. We whiz off to a New Query Window, and execute the command:

```
USE AddressBook;

ALTER TABLE dbo.ContactVerificationDetails
DROP COLUMN AddressVerified;
```

Bang! The column has gone, obliterated by the confirmation message of **Command(s) completed successfully**. We ring the bright spark to confirm the deed is done and put our feet up, feeling pretty good about life.

Five minutes later the phone rings—the system isn't working! You take a look at it and indeed, there is some kind of database error. After investigating you narrow it down to the `VerifiedContacts` view. You run a `SELECT` statement against the view and are shocked to see it fail (the errors are shown in Figure 13-4).


SQLQuery8.sql - BE...ikemcquillan (56))*
SQLQuery7.sql - BE...ikemcquillan (55))*
19 - Create Verifie...\mikemcquillan (51))
100 %
Messages

```
USE AddressBook;

SELECT * FROM dbo.VerifiedContacts
    WHERE YEAR(DateOfBirth) > 1980;
```

```
Msg 207, Level 16, State 1, Procedure VerifiedContacts, Line 15
Invalid column name 'AddressVerified'.
Msg 4413, Level 16, State 1, Line 3
Could not use view or function 'dbo.VerifiedContacts' because of binding errors.
```

***Figure 13-4.*** *A broken view*

The first message tells you that the view is using the `AddressVerified` column you just dropped. The second error states that the view cannot be used because of binding errors. The binding error occurs because the columns the view is expecting to find in its `SELECT` statement no longer exist.

Why didn't SQL Server tell you about this? How did it let you delete the column when you are using it?

SCHEMABINDING

As usual, SQL Server isn't at fault when we incorrectly remove a column—we're at fault (could it be any other way?). We should have told SQL Server it must check if the table is being used by any other objects before it allows the table's structure to be modified. We can do this by using the `SCHEMABINDING` directive.

SCHEMABINDING tells SQL Server to link the view directly to the table. When SCHEMABINDING is not specified, the view just exists as an entity in its own right. Even though it uses one or more tables, it has no relationship with those tables—the tables have no idea that the view exists. By specifying SCHEMABINDING we bind the view to the tables. This means that if anybody tries to modify a column in the table and the view will be affected by that change, the change will be prevented.

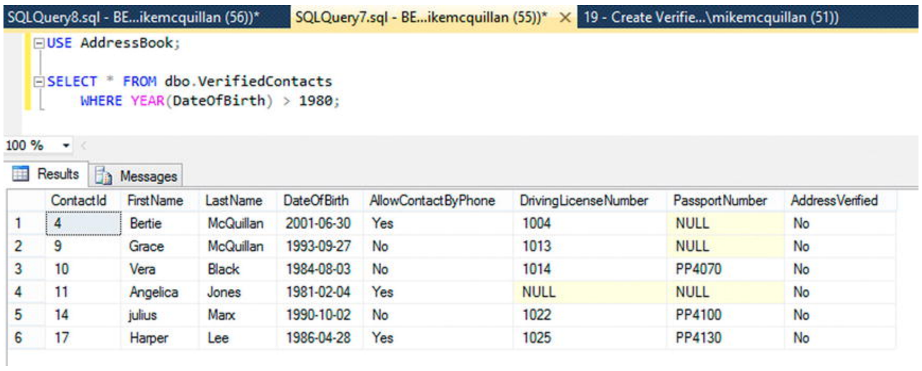
Execute the script to add the AddressVerified column back to the database, in order to fix the view:

```
ALTER TABLE dbo.ContactVerificationDetails
ADD AddressVerified     BIT NULL;
```

To modify the CREATE VIEW statement, add WITH SCHEMABINDING immediately after the view name. Change your CREATE VIEW script to include WITH SCHEMABINDING:

```
CREATE VIEW dbo.VerifiedContacts
WITH SCHEMABINDING
AS...
```

Run this and the view will be created. Execute the SELECT statement to make sure everything is still working as expected (hopefully your results will match Figure 13-5).


SQLQuery8.sql - BE...ikemcquillan (56))*
SQLQuery7.sql - BE...ikemcquillan (55))*
19 - Create Verifie...\mikemcquillan (51))
100 %
Results
Messages

```
USE AddressBook;

SELECT * FROM dbo.VerifiedContacts
    WHERE YEAR(DateOfBirth) > 1980;
```

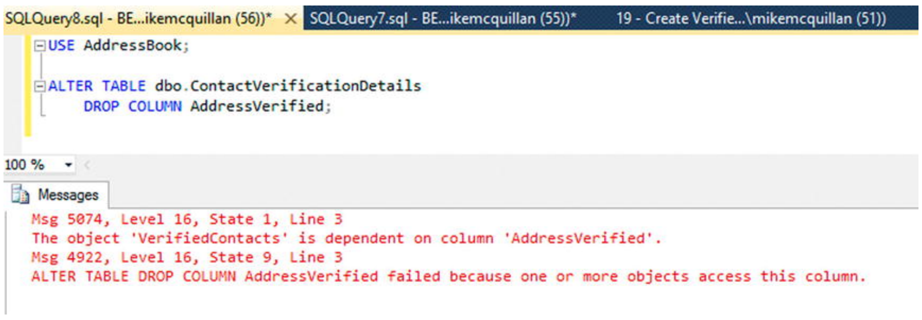
| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | DrivingLicenseNumber | PassportNumber | AddressVerified |
|---|---|---|---|---|---|---|---|---|
| 1 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | 1004 | NULL | No |
| 2 | 9 | Grace | McQuillan | 1993-09-27 | No | 1013 | NULL | No |
| 3 | 10 | Vera | Black | 1984-08-03 | No | 1014 | PP4070 | No |
| 4 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL | NULL | No |
| 5 | 14 | julius | Marx | 1990-10-02 | No | 1022 | PP4100 | No |
| 6 | 17 | Harper | Lee | 1986-04-28 | Yes | 1025 | PP4130 | No |

***Figure 13-5.*** *Making sure the view works*

Now return to the query window containing your DROP COLUMN statement and run it (or type it back in if necessary).

```
ALTER TABLE dbo.ContactVerificationDetails
DROP COLUMN AddressVerified;
```

This time the attempt to remove the column fails. Figure 13-6 shows the errors.


SQLQuery8.sql - BE...ikemcquillan (56))*
SQLQuery7.sql - BE...ikemcquillan (55))*
19 - Create Verifie...\mikemcquillan (51))
100 %
Messages

```
USE AddressBook;

ALTER TABLE dbo.ContactVerificationDetails
    DROP COLUMN AddressVerified;
```

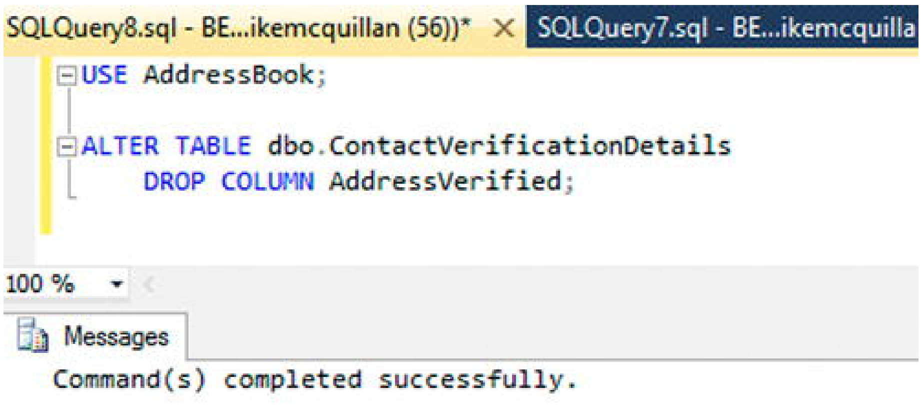
```
Msg 5074, Level 16, State 1, Line 3
The object 'VerifiedContacts' is dependent on column 'AddressVerified'.
Msg 4922, Level 16, State 9, Line 3
ALTER TABLE DROP COLUMN AddressVerified failed because one or more objects access this column.
```

***Figure 13-6.*** *Attempting to remove a schema-bound column*

This is much better. Now SQL Server is telling us the column we want to remove is used by our VerifiedContacts view. To remove it, we return to our CREATE VIEW statement, and remove the AddressVerified piece. Here's the final version of the view. Make sure you save this as script 19.

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.views WHERE [Name] = 'VerifiedContacts')
BEGIN
DROP VIEW dbo.VerifiedContacts;
END;

GO

CREATE VIEW dbo.VerifiedContacts
WITH SCHEMABINDING
AS
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, CASE C.AllowContactByPhone WHEN 1 THEN

GO
```

Run this, and then try to drop the column again. This time, it succeeds (Figure 13-7)!



```
USE AddressBook;

ALTER TABLE dbo.ContactVerificationDetails
    DROP COLUMN AddressVerified;
```

```
Command(s) completed successfully.
```

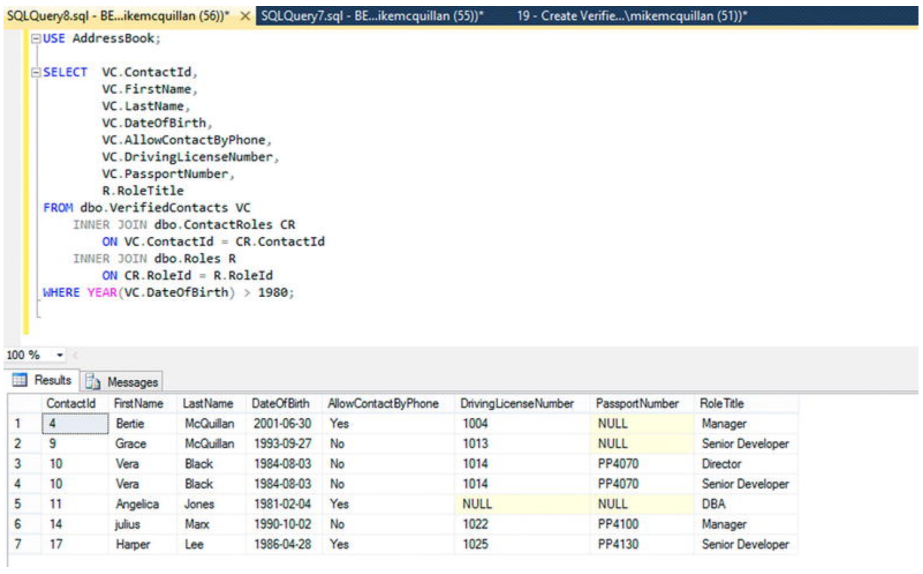
***Figure 13-7.*** *Column dropped successfully after view modification*

### Using Views in Joins

You can use views in any `SELECT` statement you care to write, including joins. It's no different to using a table. Here's a `SELECT` statement that joins our `ContactVerificationDetails` view to the `ContactRoles` and `Roles` tables.

```
USE AddressBook;

SELECT VC.ContactId, VC.FirstName, VC.LastName, VC.DateOfBirth, VC.AllowContactByPhone, VC.DrivingL
```

Figure 13-8 shows that this returns the expected results.


SQLQuery8.sql - BE...ikemcquillan (56))* SQLQuery7.sql - BE...ikemcquillan (55))* 19 - Create Verifie...\mikemcquillan (51))*

```
USE AddressBook;

SELECT  VC.ContactId,
        VC.FirstName,
        VC.LastName,
        VC.DateOfBirth,
        VC.AllowContactByPhone,
        VC.DrivingLicenseNumber,
        VC.PassportNumber,
        R.RoleTitle
FROM dbo.VerifiedContacts VC
    INNER JOIN dbo.ContactRoles CR
        ON VC.ContactId = CR.ContactId
    INNER JOIN dbo.Roles R
        ON CR.RoleId = R.RoleId
WHERE YEAR(VC.DateOfBirth) > 1980;
```

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | DrivingLicenseNumber | PassportNumber | RoleTitle |
|---|---|---|---|---|---|---|---|---|
| 1 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | 1004 | NULL | Manager |
| 2 | 9 | Grace | McQuillan | 1993-09-27 | No | 1013 | NULL | Senior Developer |
| 3 | 10 | Vera | Black | 1984-08-03 | No | 1014 | PP4070 | Director |
| 4 | 10 | Vera | Black | 1984-08-03 | No | 1014 | PP4070 | Senior Developer |
| 5 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL | NULL | DBA |
| 6 | 14 | julius | Marx | 1990-10-02 | No | 1022 | PP4100 | Manager |
| 7 | 17 | Harper | Lee | 1986-04-28 | Yes | 1025 | PP4130 | Senior Developer |

***Figure 13-8.*** *Joining the view to other tables*

## Using Views in Other Views

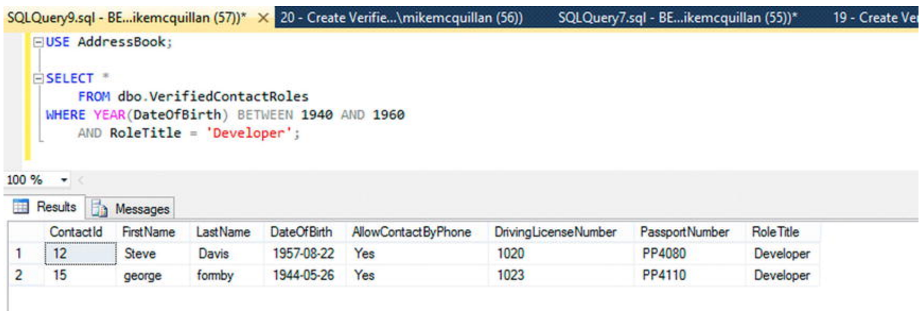
Much as you can use any table in any view, you can use any view you've already created in other views. We've just written a `SELECT` statement to join the `VerifiedContacts` view to the `ContactRoles` and `Roles` tables. Let's convert this into a view, removing the `WHERE` clause while we're at it.

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.views WHERE [Name] = 'VerifiedContactRoles')
BEGIN
DROP VIEW dbo.VerifiedContactRoles;
END;

GO

CREATE VIEW dbo.VerifiedContactRoles
WITH SCHEMABINDING
AS
SELECT VC.ContactId, VC.FirstName, VC.LastName, VC.DateOfBirth, VC.AllowContactByPhone, VC.DrivingL

GO
```

Save this as `c:\temp\sqlbasics\apply\20 - Create VerifiedContactRoles View.sql`. Run the script, open a New Query Window, and execute a `SELECT` statement.

```
USE AddressBook;

SELECT * FROM dbo.VerifiedContactRoles WHERE YEAR(DateOfBirth) BETWEEN 1940 AND 1960 AND RoleTitle
```

As demonstrated in Figure 13-9, the query runs without issue.


SQLQuery9.sql - BE...ikemcquillan (57))* 20 - Create Verifie...\mikemcquillan (56)) SQLQuery7.sql - BE...ikemcquillan (55))* 19 - Create Ve

```
USE AddressBook;

SELECT *
    FROM dbo.VerifiedContactRoles
WHERE YEAR(DateOfBirth) BETWEEN 1940 AND 1960
    AND RoleTitle = 'Developer';
```

|  | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | DrivingLicenseNumber | PassportNumber | RoleTitle |
|---|---|---|---|---|---|---|---|---|
| 1 | 12 | Steve | Davis | 1957-08-22 | Yes | 1020 | PP4080 | Developer |
| 2 | 15 | george | formby | 1944-05-26 | Yes | 1023 | PP4110 | Developer |

***Figure 13-9.*** *Querying using the* `VerifiedContactRoles` *view*

Add the `CREATE VIEW` script to the bottom of the `00 - Apply.sql` script:

```
:setvar currentFile "20 - Create VerifiedContactRoles View.sql"
PRINT 'Executing $(path)$(currentFile)'
:r $(path)$(currentFile)
```

Remember, you can use views to greatly reduce your code. If you find yourself constantly writing the same joins, consider putting them in a view to reduce your code.

### Indexed Views

It's possible to improve view performance by indexing views. We'll take a look at indexing views in the very next chapter.

### Dropping the Views

Dropping a view works in the same way as dropping a table or database. The command to use is `DROP VIEW`, followed by the view name. To drop our `VerifiedContacts` view we'd write:

```
DROP VIEW dbo.VerifiedContacts;
```

If you did want to drop this view, you'd need to drop the `VerifiedContactRoles` view first, as it references the `VerifiedContacts` view.

We need to create some rollback scripts for the two views we created, so let's incorporate `DROP VIEW` into those. Here is `c:\temp\sqlbasics\rollback\19 - Create VerifiedContacts View Rollback.sql`:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.views WHERE [Name] = 'VerifiedContacts')
BEGIN
DROP VIEW dbo.VerifiedContacts;
END;

GO
```

And here is `c:\temp\sqlbasics\rollback\20 - Create VerifiedContactRoles View Rollback.sql`:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.views WHERE [Name] = 'VerifiedContactRoles')
BEGIN
DROP VIEW dbo.VerifiedContactRoles;
END;

GO
```

To finish off, just add the relevant SQLCMD calls to the top of the `00 - Rollback.sql` script.

```
:setvar currentFile "20 - Create VerifiedContactRoles View Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "19 - Create VerifiedContacts View Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)
```

## Summary

Views offer a fantastic mechanism that is often underused (I include myself here!). They can save you a world of pain and make queries easier to understand by hiding complexity.

I mentioned it is possible to index views for better performance. That's something we'll see in our next chapter, as we take a look at indexing.

R
S
PREV
Chapter 12 : Joining Tables
NEXT
Chapter 14 : Indexes