**CHAPTER 12**

# Joining Tables

Up to this point, all of our `SELECT` statements have used the `Contacts` table. This may seem thrilling enough, but we're about to take the octane level up to Jason Statham movie territory. The `Contacts` table we've been looking at doesn't offer us much information—really, just the name and date of birth. If we wanted to phone a contact, how would we identify that contact's phone number? Based on the approach we've used so far, we could write a `SELECT` to find a record in the `Contacts` table using `FirstName` and `LastName`. From there, we could make a note of the `ContactId` and then use this to write a `SELECT` against the `ContactPhoneNumbers` table, to list that particular contact's phone numbers.

This is all well and good, but talk about going the long way around! We'll look at how joins can be used to solve this very problem. Joins let you link related tables together, allowing their contents to be displayed as a single result set. We could join `Contacts` to `ContactPhoneNumbers`, and display the various phone numbers alongside the contact names. The foreign keys we implemented earlier give our tables the means to link to each other.

SQL Server offers five types of join. In practice, I've found `INNER JOIN` and `LEFT OUTER JOIN` are used regularly, with `RIGHT OUTER JOIN` used semiregularly. The remaining two, `FULL OUTER JOIN` and `CROSS JOIN`, are seldom used. They can be very handy to know about when the need arises, though. We'll begin by seeing how joins can be applied to `SELECT` statements, but we'll also see how they can be used to `INSERT`, `UPDATE`, and `DELETE` data too.

## INNER JOIN

An `INNER JOIN` will only return data that exists in both tables specified in the join. You specify which column (or columns) you want to join on. If there is a match in both tables, a row will be returned. The black area in Figure 12-1 shows the rows that match.

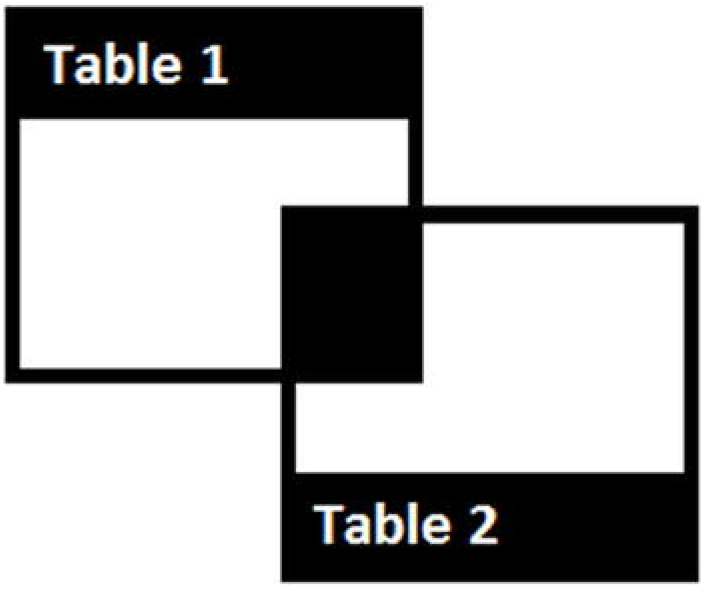


***Figure 12-1.*** *Data returned by an `INNER JOIN`*

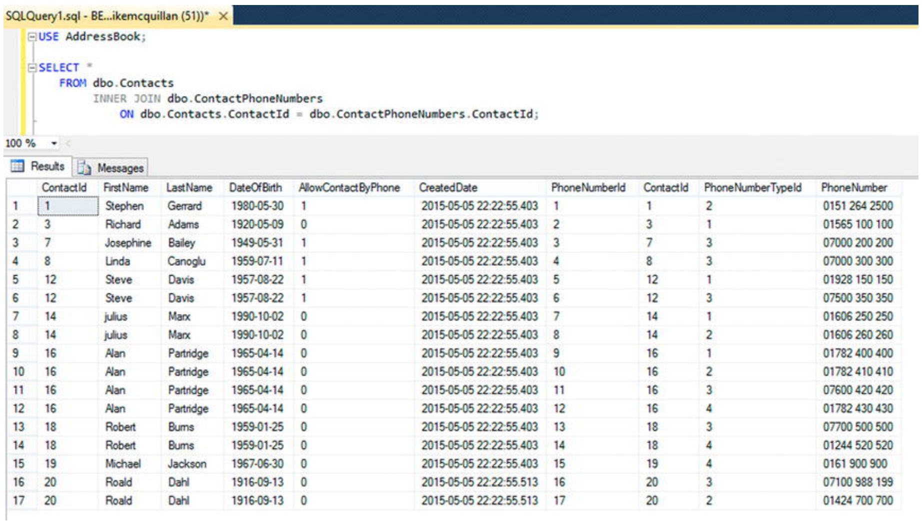
If Table 1 contains a record with an ID that is not present in Table 2, that record will not be returned. Similarly, if Table 2 contains a record with an ID that is not present in Table 1, it will not be returned.

If the ID is present on a record in both Table 1 and Table 2, it will be included in the result set. Here's our firs attempt at an `INNER JOIN`, using `SELECT *` for now (naughty me!).

```
SELECT * FROM dbo.Contacts
INNER JOIN dbo.ContactPhoneNumbers
ON dbo.Contacts.ContactId = dbo.ContactPhoneNumbers.ContactId;
```

Note that the `INNER JOIN` is so called because it only returns rows that exist in both tables. The inner data is highlighted in the diagram above, showing how the data intersects. Anything outside of this intersected area is termed `OUTER` data.

The query above returns the 17 rows displayed in Figure 12-2.


SQLQuery1.sql - BE...ikemcquillan (51))*

```
USE AddressBook;

SELECT *
    FROM dbo.Contacts
        INNER JOIN dbo.ContactPhoneNumbers
            ON dbo.Contacts.ContactId = dbo.ContactPhoneNumbers.ContactId;
```

100 %

Results | Messages

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | CreatedDate | PhoneNumberId | ContactId | PhoneNumberTypeId | PhoneNumber |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | 1 | 2015-05-05 22:22:55.403 | 1 | 1 | 2 | 0151 264 2500 |
| 2 | 3 | Richard | Adams | 1920-05-09 | 0 | 2015-05-05 22:22:55.403 | 2 | 3 | 1 | 01565 100 100 |
| 3 | 7 | Josephine | Bailey | 1949-05-31 | 1 | 2015-05-05 22:22:55.403 | 3 | 7 | 3 | 07000 200 200 |
| 4 | 8 | Linda | Canoglu | 1959-07-11 | 1 | 2015-05-05 22:22:55.403 | 4 | 8 | 3 | 07000 300 300 |
| 5 | 12 | Steve | Davis | 1957-08-22 | 1 | 2015-05-05 22:22:55.403 | 5 | 12 | 1 | 01928 150 150 |
| 6 | 12 | Steve | Davis | 1957-08-22 | 1 | 2015-05-05 22:22:55.403 | 6 | 12 | 3 | 07500 350 350 |
| 7 | 14 | julius | Marx | 1990-10-02 | 0 | 2015-05-05 22:22:55.403 | 7 | 14 | 1 | 01606 250 250 |
| 8 | 14 | julius | Marx | 1990-10-02 | 0 | 2015-05-05 22:22:55.403 | 8 | 14 | 2 | 01606 260 260 |
| 9 | 16 | Alan | Partridge | 1965-04-14 | 0 | 2015-05-05 22:22:55.403 | 9 | 16 | 1 | 01782 400 400 |
| 10 | 16 | Alan | Partridge | 1965-04-14 | 0 | 2015-05-05 22:22:55.403 | 10 | 16 | 2 | 01782 410 410 |
| 11 | 16 | Alan | Partridge | 1965-04-14 | 0 | 2015-05-05 22:22:55.403 | 11 | 16 | 3 | 07600 420 420 |
| 12 | 16 | Alan | Partridge | 1965-04-14 | 0 | 2015-05-05 22:22:55.403 | 12 | 16 | 4 | 01782 430 430 |
| 13 | 18 | Robert | Burns | 1959-01-25 | 0 | 2015-05-05 22:22:55.403 | 13 | 18 | 3 | 07700 500 500 |
| 14 | 18 | Robert | Burns | 1959-01-25 | 0 | 2015-05-05 22:22:55.403 | 14 | 18 | 4 | 01244 520 520 |
| 15 | 19 | Michael | Jackson | 1967-06-30 | 0 | 2015-05-05 22:22:55.403 | 15 | 19 | 4 | 0161 900 900 |
| 16 | 20 | Roald | Dahl | 1916-09-13 | 0 | 2015-05-05 22:22:55.513 | 16 | 20 | 3 | 07100 988 199 |
| 17 | 20 | Roald | Dahl | 1916-09-13 | 0 | 2015-05-05 22:22:55.513 | 17 | 20 | 2 | 01424 700 700 |

***Figure 12-2.*** *Data returned by an INNER JOIN—for real!*

Hmm, this query and its results seem to bring up more questions than answers. If I saw this I'd be asking:

- Why does the query specify the table names multiple times? Isn't there a cleaner way of writing the code?
- Why is the `PhoneNumberId` column included in the results? It's of no interest to me.
- Why do the results contain two `ContactId` columns?
- The `PhoneNumberTypeId` just lists numbers. How can I find out the real `PhoneNumberType`?
- Can I display something more user friendly than `1` or `0` in the `AllowContactByPhone` column?
- Why aren't all of my contacts included in this report? I can tell by the `ContactId` values that some of them are missing, like `ContactId 2`.
- Can I filter this to just include the contacts who have said I can phone them?

Whew, there's a lot to look at here! I'll try to answer and resolve each question one by one.

To start with, the query is unnecessarily long. Do you remember the column aliases we just looked at? Well, you can alias tables, too. We need to use table aliases here. We assign the alias to the table when we first declare it in the `SELECT` statement, and we can then reference it from anywhere in the query using its alias. We'll change `Contacts` to `C`, and `ContactPhoneNumbers` to `CPN`.

```
SELECT * FROM dbo.Contacts AS C
INNER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId;
```

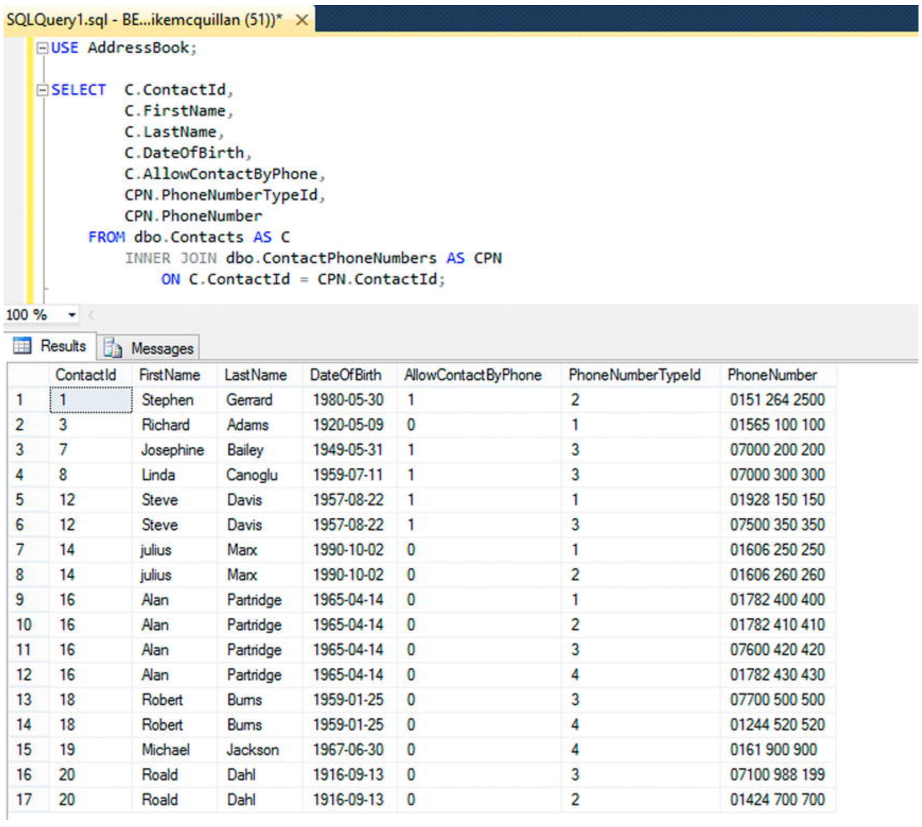
This is much easier to understand and much shorter to write, too. So now we know that anywhere we see `C` it represents the `Contacts` table, and anywhere we see `CPN` it represents the `ContactPhoneNumbers` table.

The next two questions involve the columns we are showing in this query. We're using `*` at the moment, so every column from both tables is being returned. We should limit the set to just the useful columns. `PhoneNumberId` isn't useful, so we won't include that. `ContactId` might be useful—but we don't want two `ContactId`s. Why are there two of them? Don't worry, nothing sinister is going on! The first one comes from the `Contacts` table, and the second comes from the `ContactPhoneNumbers` table. We can use our aliases to just return the `Contacts` version.

Without further ado, let's reduce the columns.

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, C.AllowContactByPhone,
   CPN.PhoneNumberTypeId, CPN.PhoneNumber FROM dbo.Contacts AS C
INNER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId;
```

If we run this, things look better, as you can see in Figure 12-3.



```
USE AddressBook;

SELECT  C.ContactId,
        C.FirstName,
        C.LastName,
        C.DateOfBirth,
        C.AllowContactByPhone,
        CPN.PhoneNumberTypeId,
        CPN.PhoneNumber
    FROM dbo.Contacts AS C
        INNER JOIN dbo.ContactPhoneNumbers AS CPN
            ON C.ContactId = CPN.ContactId;
```

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumberTypeId | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | 1 | 2 | 0151 264 2500 |
| 2 | 3 | Richard | Adams | 1920-05-09 | 0 | 1 | 01565 100 100 |
| 3 | 7 | Josephine | Bailey | 1949-05-31 | 1 | 3 | 07000 200 200 |
| 4 | 8 | Linda | Canoglu | 1959-07-11 | 1 | 3 | 07000 300 300 |
| 5 | 12 | Steve | Davis | 1957-08-22 | 1 | 1 | 01928 150 150 |
| 6 | 12 | Steve | Davis | 1957-08-22 | 1 | 3 | 07500 350 350 |
| 7 | 14 | julius | Marx | 1990-10-02 | 0 | 1 | 01606 250 250 |
| 8 | 14 | julius | Marx | 1990-10-02 | 0 | 2 | 01606 260 260 |
| 9 | 16 | Alan | Partridge | 1965-04-14 | 0 | 1 | 01782 400 400 |
| 10 | 16 | Alan | Partridge | 1965-04-14 | 0 | 2 | 01782 410 410 |
| 11 | 16 | Alan | Partridge | 1965-04-14 | 0 | 3 | 07600 420 420 |
| 12 | 16 | Alan | Partridge | 1965-04-14 | 0 | 4 | 01782 430 430 |
| 13 | 18 | Robert | Burns | 1959-01-25 | 0 | 3 | 07700 500 500 |
| 14 | 18 | Robert | Burns | 1959-01-25 | 0 | 4 | 01244 520 520 |
| 15 | 19 | Michael | Jackson | 1967-06-30 | 0 | 4 | 0161 900 900 |
| 16 | 20 | Roald | Dahl | 1916-09-13 | 0 | 3 | 07100 988 199 |
| 17 | 20 | Roald | Dahl | 1916-09-13 | 0 | 2 | 01424 700 700 |

***Figure 12-3.*** *Using table aliases and targeted columns*

What do you think would happen if you hadn't put a `C.` in front of `ContactId`? SQL Server would have thrown an error:

```
Msg 209, Level 16, State 1, Line 3
Ambiguous column name 'ContactId'.
```

This is because the table containing `ContactId` cannot be identified without the alias. To say the alias is important to making your code easy to understand is an understatement, especially as SQL Server allows you to be very loose in most cases with your aliases. This version of the query works equally as well as the preceding amended version.

```
SELECT C.ContactId, FirstName, LastName, DateOfBirth, AllowContactByPhone,
   PhoneNumberTypeId, PhoneNumber
FROM dbo.Contacts AS C
INNER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId;
```

**BE CONSISTENT!**

Just because you *can* do something, doesn't mean that you *should*. Make your code as consistent as you possibly can at all times, even if this sometimes means a little extra typing. If your query uses more than one table, use table aliases against every column—it really will make your query easier to read.

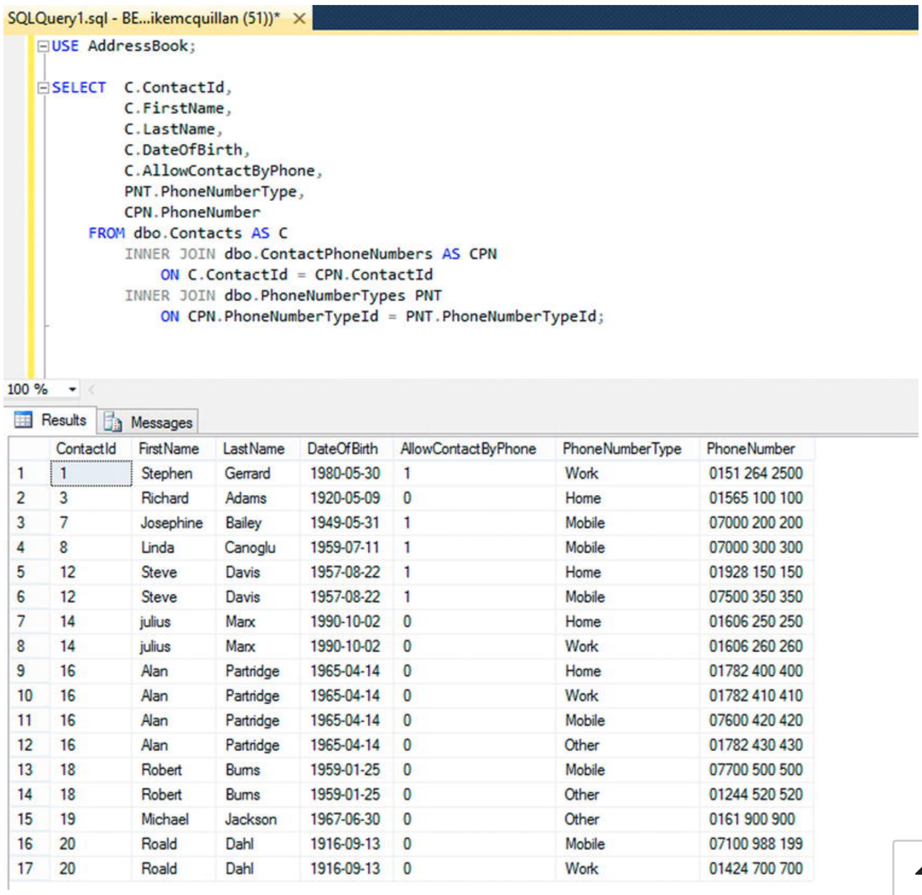
Note that only `ContactId` has an alias. We have no idea which table the other columns are coming from—it could be either table in the join. Not too much of a problem when you are only joining two tables, but if you add more joins to the mix it will become very difficult to manage. Always prefix column names with the table alias.

On to the next problem: replacing the `PhoneNumberTypeId`. To obtain the human-readable version of a phone number type, we need to join to the `PhoneNumberTypes` table. The `ContactPhoneNumbers` table has a foreign key to this table via the `PhoneNumberTypeId` column. We'll add another `INNER JOIN` to our query, and replace the `CPN.PhoneNumberTypeId` column with the `PhoneNumberType` column from the `PhoneNumberTypes` table, which we've aliased as `PNT`.

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, C.AllowContactByPhone,
   PNT.PhoneNumberType, CPN.PhoneNumber
FROM dbo.Contacts AS C
INNER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId
INNER JOIN dbo.PhoneNumberTypes PNT
ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId;
```

**Note** You can include as many joins and tables as you want in a query (SQL Server 2005 had a limit of 256 tables; now you are only constrained by local resources). I've never seen a query with more than 40 tables, and that wasn't very well written or thought out!

We're working our way through the questions; things aren't looking too bad at all now. Figure 12-4 shows our current position.



```
USE AddressBook;

SELECT  C.ContactId,
        C.FirstName,
        C.LastName,
        C.DateOfBirth,
        C.AllowContactByPhone,
        PNT.PhoneNumberType,
        CPN.PhoneNumber
    FROM dbo.Contacts AS C
        INNER JOIN dbo.ContactPhoneNumbers AS CPN
            ON C.ContactId = CPN.ContactId
        INNER JOIN dbo.PhoneNumberTypes PNT
            ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId;
```

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumberType | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | 1 | Work | 0151 264 2500 |
| 2 | 3 | Richard | Adams | 1920-05-09 | 0 | Home | 01565 100 100 |
| 3 | 7 | Josephine | Bailey | 1949-05-31 | 1 | Mobile | 07000 200 200 |
| 4 | 8 | Linda | Canoglu | 1959-07-11 | 1 | Mobile | 07000 300 300 |
| 5 | 12 | Steve | Davis | 1957-08-22 | 1 | Home | 01928 150 150 |
| 6 | 12 | Steve | Davis | 1957-08-22 | 1 | Mobile | 07500 350 350 |
| 7 | 14 | julius | Marx | 1990-10-02 | 0 | Home | 01606 250 250 |
| 8 | 14 | julius | Marx | 1990-10-02 | 0 | Work | 01606 260 260 |
| 9 | 16 | Alan | Partridge | 1965-04-14 | 0 | Home | 01782 400 400 |
| 10 | 16 | Alan | Partridge | 1965-04-14 | 0 | Work | 01782 410 410 |
| 11 | 16 | Alan | Partridge | 1965-04-14 | 0 | Mobile | 07600 420 420 |
| 12 | 16 | Alan | Partridge | 1965-04-14 | 0 | Other | 01782 430 430 |
| 13 | 18 | Robert | Burns | 1959-01-25 | 0 | Mobile | 07700 500 500 |
| 14 | 18 | Robert | Burns | 1959-01-25 | 0 | Other | 01244 520 520 |
| 15 | 19 | Michael | Jackson | 1967-06-30 | 0 | Other | 0161 900 900 |
| 16 | 20 | Roald | Dahl | 1916-09-13 | 0 | Mobile | 07100 988 199 |
| 17 | 20 | Roald | Dahl | 1916-09-13 | 0 | Work | 01424 700 700 |

***Figure 12-4.*** *Including the* `PhoneNumberTypeId` *join*

Our next problem involves the `AllowContactByPhone` column—can we make it more user friendly? Yes, we can. This is not a join problem, but it is typical of the kind of question you'll come up against when writing reporting queries for users. We can amend the values to `'Yes'` or `'No'` using something called a `CASE` statement. We'll display `Yes` for values of `1`, and `No` for values of `0`. Replace the line `C.AllowContactByPhone` with this:

```
CASE C.AllowContactByPhone
   WHEN 1 THEN 'Yes'
   ELSE 'No'
END AS AllowContactByPhone,
```

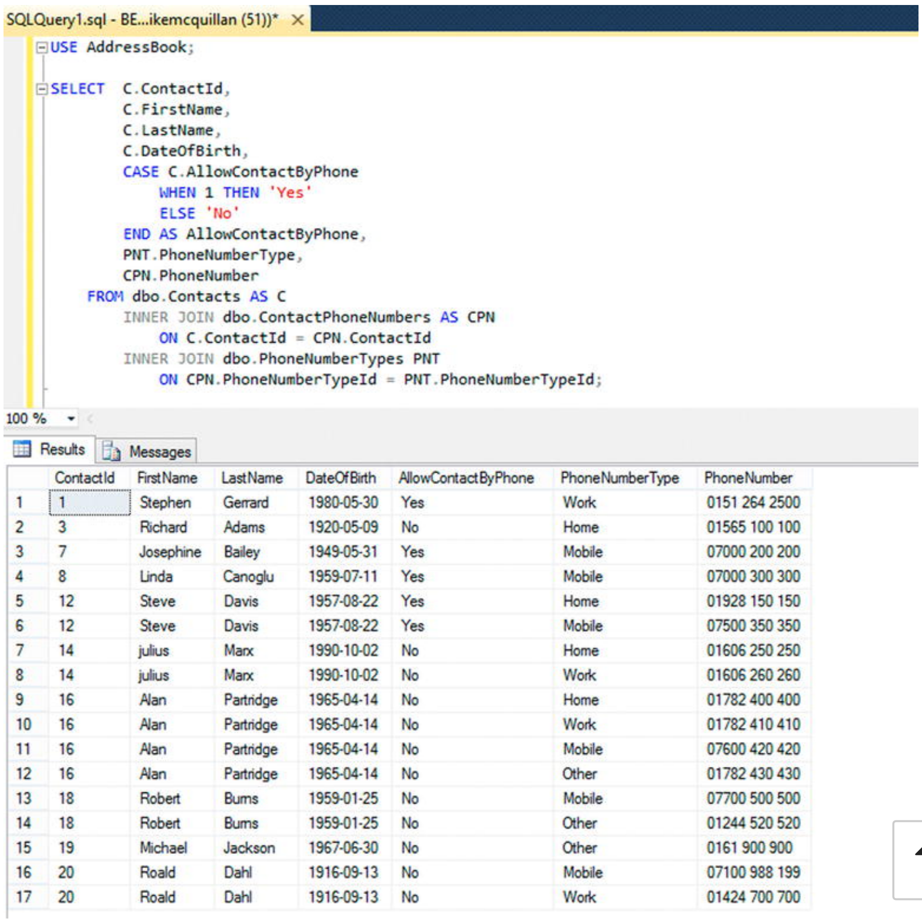
The `CASE` statement works in a similar fashion to the `IF` statement, but it operates on each row. This example says if `AllowContactByPhone` is `1`, display `Yes`. If it is anything else, display `No`. We could have written this as:

```
CASE C.AllowContactByPhone
   WHEN 1 THEN 'Yes'
   WHEN 0 THEN 'No'
   ELSE 'No'
END AS AllowContactByPhone,
```

You can specify as many `WHEN` clauses as required, but you can only have one `ELSE` clause. The optional `ELSE` clause performs the mopping up of any values you didn't include in the `WHEN` clauses for any reason. If you don't specify an `ELSE` clause and you don't explicitly specify a `WHEN` clause for a value, the column will contain a `NULL` value.

Our code doesn't include the `WHEN 0` clause, as it is redundant. If the value is `1`, display `Yes`. If it is anything else, we are not allowed to make a phone call, so just display `No`. The `ELSE` clause handles the `No` values.

Run the query now, and you'll see `Yes` and `No` in the `AllowContactByPhone` column (Figure 12-5).



```
USE AddressBook;

SELECT  C.ContactId,
        C.FirstName,
        C.LastName,
        C.DateOfBirth,
        CASE C.AllowContactByPhone
            WHEN 1 THEN 'Yes'
            ELSE 'No'
        END AS AllowContactByPhone,
        PNT.PhoneNumberType,
        CPN.PhoneNumber
    FROM dbo.Contacts AS C
        INNER JOIN dbo.ContactPhoneNumbers AS CPN
            ON C.ContactId = CPN.ContactId
        INNER JOIN dbo.PhoneNumberTypes PNT
            ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId;
```

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumberType | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | Yes | Work | 0151 264 2500 |
| 2 | 3 | Richard | Adams | 1920-05-09 | No | Home | 01565 100 100 |
| 3 | 7 | Josephine | Bailey | 1949-05-31 | Yes | Mobile | 07000 200 200 |
| 4 | 8 | Linda | Canoglu | 1959-07-11 | Yes | Mobile | 07000 300 300 |
| 5 | 12 | Steve | Davis | 1957-08-22 | Yes | Home | 01928 150 150 |
| 6 | 12 | Steve | Davis | 1957-08-22 | Yes | Mobile | 07500 350 350 |
| 7 | 14 | julius | Marx | 1990-10-02 | No | Home | 01606 250 250 |
| 8 | 14 | julius | Marx | 1990-10-02 | No | Work | 01606 260 260 |
| 9 | 16 | Alan | Partridge | 1965-04-14 | No | Home | 01782 400 400 |
| 10 | 16 | Alan | Partridge | 1965-04-14 | No | Work | 01782 410 410 |
| 11 | 16 | Alan | Partridge | 1965-04-14 | No | Mobile | 07600 420 420 |
| 12 | 16 | Alan | Partridge | 1965-04-14 | No | Other | 01782 430 430 |
| 13 | 18 | Robert | Burns | 1959-01-25 | No | Mobile | 07700 500 500 |
| 14 | 18 | Robert | Burns | 1959-01-25 | No | Other | 01244 520 520 |
| 15 | 19 | Michael | Jackson | 1967-06-30 | No | Other | 0161 900 900 |
| 16 | 20 | Roald | Dahl | 1916-09-13 | No | Mobile | 07100 988 199 |
| 17 | 20 | Roald | Dahl | 1916-09-13 | No | Work | 01424 700 700 |

*Figure 12-5. Adding a CASE statement*

We had to add a column alias at the end. If we hadn't done this, no name would have been assigned to the column. This is because the `CASE` statement generates a derived column.

This is a very simple introduction to `CASE` statements. You can do a lot with them and they'll become a useful part of your arsenal. Used wisely, `CASE` statements can greatly reduce your code in certain places.

Back to `INNER JOIN`s. Two questions left. First, why can't I see all of my contacts? If you look at the results in Figure 12-5, only 10 of the contacts are included in the results. Why is this? Are the results correct? Indeed, yes, the results are correct.

The `ContactPhoneNumbers` table contains 17 phone numbers, but several of these belong to the same contact. For example:

| Contact ID | No. of Phone Numbers |
|---|---|
| 1 | 1 |
| 3 | 1 |
| 7 | 1 |
| 8 | 1 |
| 12 | 2 |
| 14 | 2 |
| 16 | 4 |
| 18 | 2 |
| 19 | 1 |
| 20 | 2 |

Our query contains 17 results, one for each phone number. If we want to include all of our contacts, regardle of whether they have a phone number or not, we'll need to use a `LEFT JOIN`. We'll change our query a little later to do this.

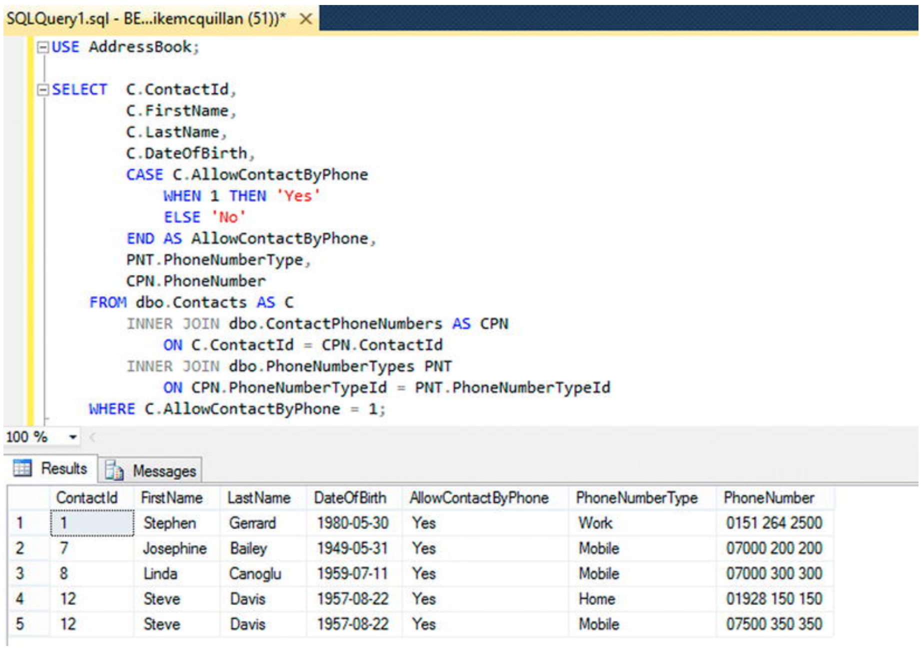
Time to answer our final question. Can we filter the list so just those contacts who have agreed to be contacted by phone are included? Of course we can, using a `WHERE` clause:

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth,
   CASE C.AllowContactByPhone WHEN 1 THEN 'Yes' ELSE 'No' END AS AllowContactByPhone,
   PNT.PhoneNumberType, CPN.PhoneNumber
FROM dbo.Contacts AS C
INNER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId
INNER JOIN dbo.PhoneNumberTypes PNT
ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId
WHERE C.AllowContactByPhone = 1;
```

We specified the alias in front of the column name—we always use the alias now for consistency. And we still use `1` to match values, even though we display `Yes` in the result set. Using the `CASE` statement in the `WHERE` clause is possible, but adds unnecessary overhead. Here's how you do it, though:

```
WHERE CASE C.AllowContactByPhone WHEN 1 THEN 'Yes' ELSE 'No' END = 'Yes'
```

We've done well here; this is a pretty useful query. If you run it, the five results in Figure 12-6 should be returned.



SQLQuery1.sql - BE...ikemcquillan (51))*

```
USE AddressBook;

SELECT  C.ContactId,
        C.FirstName,
        C.LastName,
        C.DateOfBirth,
        CASE C.AllowContactByPhone
            WHEN 1 THEN 'Yes'
            ELSE 'No'
        END AS AllowContactByPhone,
        PNT.PhoneNumberType,
        CPN.PhoneNumber
    FROM dbo.Contacts AS C
        INNER JOIN dbo.ContactPhoneNumbers AS CPN
            ON C.ContactId = CPN.ContactId
        INNER JOIN dbo.PhoneNumberTypes PNT
            ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId
    WHERE C.AllowContactByPhone = 1;
```

100 %

Results | Messages

| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumberType | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | Yes | Work | 0151 264 2500 |
| 2 | 7 | Josephine | Bailey | 1949-05-31 | Yes | Mobile | 07000 200 200 |
| 3 | 8 | Linda | Canoglu | 1959-07-11 | Yes | Mobile | 07000 300 300 |
| 4 | 12 | Steve | Davis | 1957-08-22 | Yes | Home | 01928 150 150 |
| 5 | 12 | Steve | Davis | 1957-08-22 | Yes | Mobile | 07500 350 350 |

***Figure 12-6.*** *The Completed `INNER JOIN` Query*

We've had a good introduction to `INNER JOIN`s. The key to understanding them is remembering that matching records must exist in both tables involved in the join—unlike our next contestant. . . .

## LEFT OUTER JOIN

`LEFT OUTER JOIN`s—or `LEFT JOIN`s as I'll refer to them—return every row from the left-hand table, but only matching rows from the right-hand table. Figure 12-7 illustrates how data are matched.

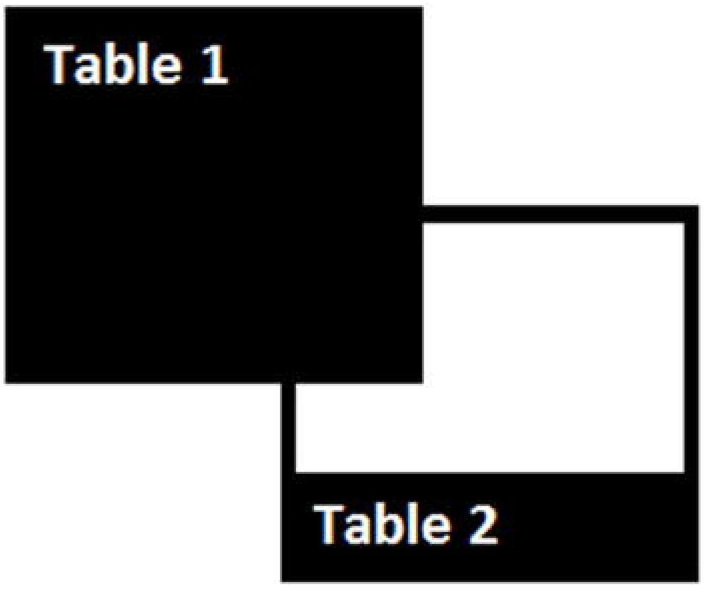


***Figure 12-7.*** *Data returned by a* `LEFT JOIN`

Everything comes back from Table 1, but only the overlapping data—the matching data—are returned from Table 2. Note that all OUTER data from Table 1 are returned; that is, we don't just returned intersected data here.

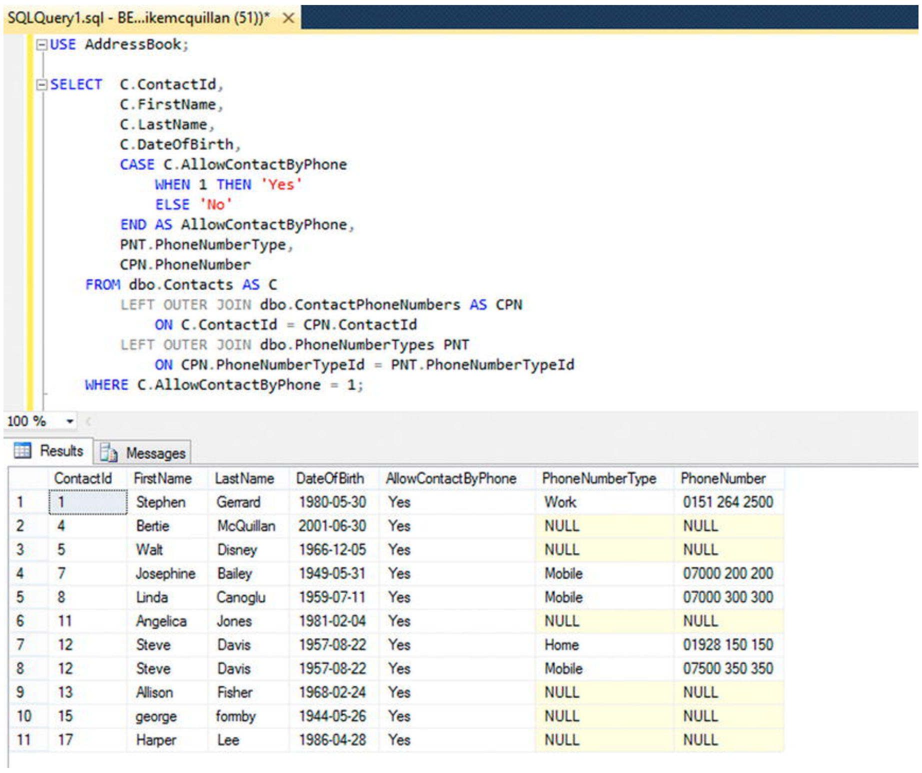
Our `INNER JOIN` query currently returns all records from `Contacts` that have `ContactPhoneNumbers` and have stated that we are allowed to contact them by phone. The first part of the problem is solved using an `INNER JOIN`, and the `WHERE` clause is used to check if contact is allowed by phone. We can change this query to return all contact records that have stated they can be contacted by phone, regardless of whether they have a phone number or not (maybe they plan to provide one in future?). Make one simple modification: change the first `INNER JOIN` to a `LEFT OUTER JOIN`.

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth,
   CASE C.AllowContactByPhone WHEN 1 THEN 'Yes' ELSE 'No' END AS AllowContactByPhone,
   PNT.PhoneNumberType, CPN.PhoneNumber
FROM dbo.Contacts AS C
LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId
INNER JOIN dbo.PhoneNumberTypes PNT
ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId
WHERE C.AllowContactByPhone = 1;
```

Run this . . . and there is no change. The same five results come back. Why is that? There are definitely some contacts that allow contact by phone, but don't have a matching record in the `ContactPhoneNumbers` table. Where are they? Don't worry, they are still in the database. The problem is the second `INNER JOIN`. This is matching records in `ContactPhoneNumbers` and `PhoneNumberTypes` that have the same `PhoneNumberTypeId`. Our contact records that don't have a phone number have a `PhoneNumberTypeId` of `NULL`, which cannot be matched. The first join, the `LEFT OUTER JOIN`, is bringing back the records, but the second `INNER JOIN` is filtering them out. The solution is to change the second `INNER JOIN` into a `LEFT OUTER JOIN`, too.

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth,
   CASE C.AllowContactByPhone WHEN 1 THEN 'Yes' ELSE 'No' END AS AllowContactByPhone,
   PNT.PhoneNumberType, CPN.PhoneNumber
FROM dbo.Contacts AS C
LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId
LEFT OUTER JOIN dbo.PhoneNumberTypes PNT
ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId
WHERE C.AllowContactByPhone = 1;
```

And now our results make a bit more sense, as Figure 12-8 tells us.


SQLQuery1.sql - BE...ikemcquillan (51))*

```
USE AddressBook;

SELECT  C.ContactId,
        C.FirstName,
        C.LastName,
        C.DateOfBirth,
        CASE C.AllowContactByPhone
            WHEN 1 THEN 'Yes'
            ELSE 'No'
        END AS AllowContactByPhone,
        PNT.PhoneNumberType,
        CPN.PhoneNumber
    FROM dbo.Contacts AS C
        LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
            ON C.ContactId = CPN.ContactId
        LEFT OUTER JOIN dbo.PhoneNumberTypes PNT
            ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId
    WHERE C.AllowContactByPhone = 1;
```

|  | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumberType | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | Yes | Work | 0151 264 2500 |
| 2 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | NULL | NULL |
| 3 | 5 | Walt | Disney | 1966-12-05 | Yes | NULL | NULL |
| 4 | 7 | Josephine | Bailey | 1949-05-31 | Yes | Mobile | 07000 200 200 |
| 5 | 8 | Linda | Canoglu | 1959-07-11 | Yes | Mobile | 07000 300 300 |
| 6 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL | NULL |
| 7 | 12 | Steve | Davis | 1957-08-22 | Yes | Home | 01928 150 150 |
| 8 | 12 | Steve | Davis | 1957-08-22 | Yes | Mobile | 07500 350 350 |
| 9 | 13 | Allison | Fisher | 1968-02-24 | Yes | NULL | NULL |
| 10 | 15 | george | formby | 1944-05-26 | Yes | NULL | NULL |
| 11 | 17 | Harper | Lee | 1986-04-28 | Yes | NULL | NULL |

*Figure 12-8.* Data returned after specifying LEFT JOINs

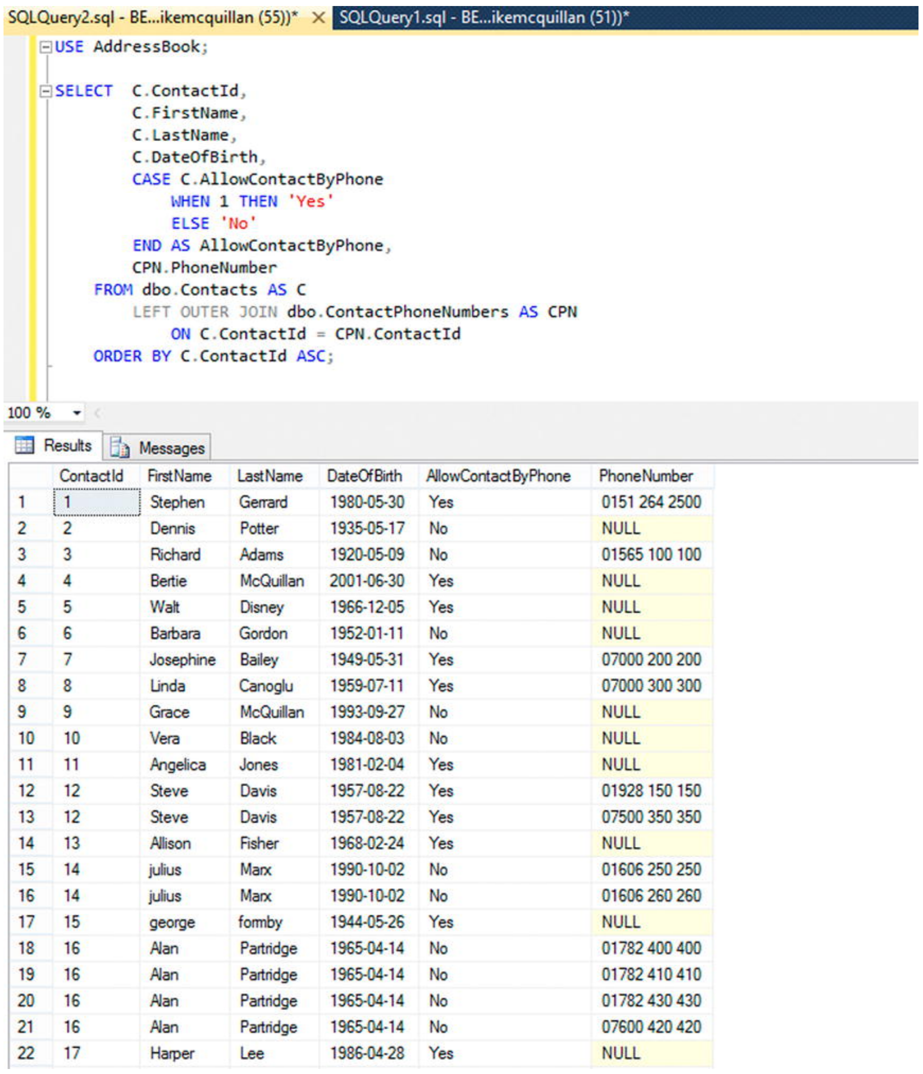
Five of these records have phone numbers and phone number types; the rest all show `NULL`.

I've used the phrase `LEFT OUTER JOIN` in the query. I could easily have written `LEFT JOIN` instead. This is an area where I have no real preference—for me, `LEFT JOIN` is just as descriptive as `LEFT OUTER JOIN`. I'll continue to use `LEFT OUTER JOIN` and so on throughout the book, but just remember to be consistent. Plump for one or the other, and stick with it.

Let's look at another `LEFT OUTER JOIN`. This is a bit simpler than our previous example.

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth,
   CASE C.AllowContactByPhone WHEN 1 THEN 'Yes' ELSE 'No' END AS AllowContactByPhone,
   CPN.PhoneNumber
FROM dbo.Contacts AS C
LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId
ORDER BY C.ContactId ASC;
```

We've removed the extra `LEFT OUTER JOIN` and the `WHERE` clause. Now every record in `Contacts` will be returned regardless of whether a phone number record is available or not. That's 27 records in total. You can see most of these in Figure 12-9.



```
USE AddressBook;

SELECT  C.ContactId,
        C.FirstName,
        C.LastName,
        C.DateOfBirth,
        CASE C.AllowContactByPhone
            WHEN 1 THEN 'Yes'
            ELSE 'No'
        END AS AllowContactByPhone,
        CPN.PhoneNumber
    FROM dbo.Contacts AS C
        LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
            ON C.ContactId = CPN.ContactId
    ORDER BY C.ContactId ASC;
```

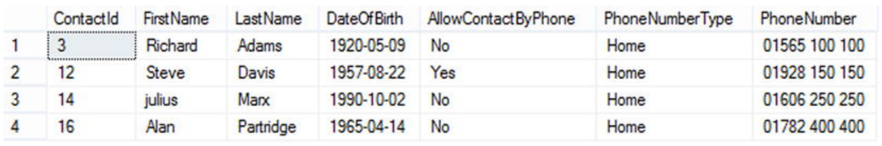
| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumber |
|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | Yes | 0151 264 2500 |
| 2 | 2 | Dennis | Potter | 1935-05-17 | No | NULL |
| 3 | 3 | Richard | Adams | 1920-05-09 | No | 01565 100 100 |
| 4 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | NULL |
| 5 | 5 | Walt | Disney | 1966-12-05 | Yes | NULL |
| 6 | 6 | Barbara | Gordon | 1952-01-11 | No | NULL |
| 7 | 7 | Josephine | Bailey | 1949-05-31 | Yes | 07000 200 200 |
| 8 | 8 | Linda | Canoglu | 1959-07-11 | Yes | 07000 300 300 |
| 9 | 9 | Grace | McQuillan | 1993-09-27 | No | NULL |
| 10 | 10 | Vera | Black | 1984-08-03 | No | NULL |
| 11 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL |
| 12 | 12 | Steve | Davis | 1957-08-22 | Yes | 01928 150 150 |
| 13 | 12 | Steve | Davis | 1957-08-22 | Yes | 07500 350 350 |
| 14 | 13 | Allison | Fisher | 1968-02-24 | Yes | NULL |
| 15 | 14 | julius | Marx | 1990-10-02 | No | 01606 250 250 |
| 16 | 14 | julius | Marx | 1990-10-02 | No | 01606 260 260 |
| 17 | 15 | george | formby | 1944-05-26 | Yes | NULL |
| 18 | 16 | Alan | Partridge | 1965-04-14 | No | 01782 400 400 |
| 19 | 16 | Alan | Partridge | 1965-04-14 | No | 01782 410 410 |
| 20 | 16 | Alan | Partridge | 1965-04-14 | No | 01782 430 430 |
| 21 | 16 | Alan | Partridge | 1965-04-14 | No | 07600 420 420 |
| 22 | 17 | Harper | Lee | 1986-04-28 | Yes | NULL |

***Figure 12-9.*** *Records returned regardless of phone number*

Say we want to display all contact records regardless of whether we have a phone number for them or not, and we only want to display their phone number if it is a home phone number. Any ideas how we may do that? What's that? Add a `WHERE` clause, you say? Okay, here's that very query with the `LEFT OUTER JOIN` to `PhoneNumberTypes` reintroduced, too.

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth,
   CASE C.AllowContactByPhone WHEN 1 THEN 'Yes' ELSE 'No' END AS AllowContactByPhone,
   PNT.PhoneNumberType, CPN.PhoneNumber
FROM dbo.Contacts AS C
LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId
LEFT OUTER JOIN dbo.PhoneNumberTypes AS PNT
ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId
WHERE PNT.PhoneNumberType = 'Home'
ORDER BY C.ContactId ASC;
```

Does this work? No! Only the home numbers are returned. Records without a home number have been removed (check out Figure 12-10).



| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumberType | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | 3 | Richard | Adams | 1920-05-09 | No | Home | 01565 100 100 |
| 2 | 12 | Steve | Davis | 1957-08-22 | Yes | Home | 01928 150 150 |
| 3 | 14 | julius | Marx | 1990-10-02 | No | Home | 01606 250 250 |
| 4 | 16 | Alan | Partridge | 1965-04-14 | No | Home | 01782 400 400 |

***Figure 12-10.*** *Only home phone numbers returned*

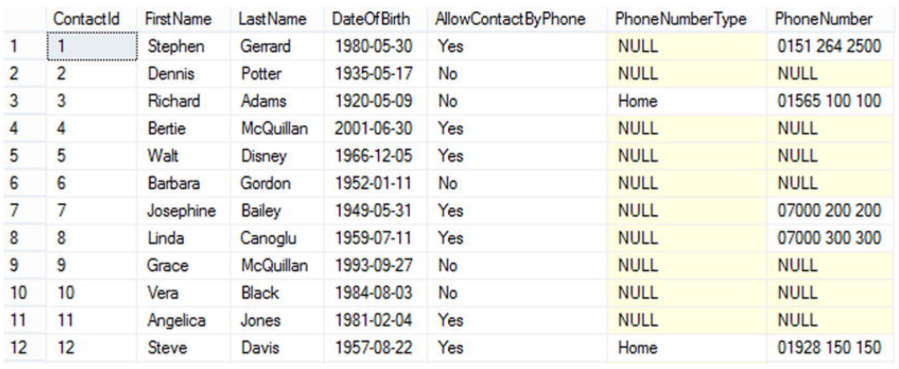
The problem is the `WHERE` clause limits what is displayed. The query is executing in this manner:

- `FROM dbo.Contacts AS C` returns all contacts. There are 20 records in the set at this point.
- `LEFT OUTER JOIN dbo.ContactPhoneNumbers` returns all contacts, and any matching phone numbers. The set now contains 27 records.
- `LEFT OUTER JOIN dbo.PhoneNumberTypes` includes each phone number's phone number type. The set still contains 27 records.
- `WHERE PNT.PhoneNumberType = 'Home'` tells the query to remove all records from the set that don't have a phone number type of `'Home'`. Only 4 records meet this criteria

Okay, that doesn't work, then. How about removing the `WHERE` clause and adding it to the last `LEFT OUTER JOIN`?

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, CASE C.AllowContactByPhone WHEN 1 THEN
```

We've now told the second `LEFT OUTER JOIN` that two conditions must be fulfilled for records to be returned. Unfortunately, this is still wrong! Twenty-seven rows are returned now (Figure 12-11 shows some of these rows)—that's all contacts and their phone numbers.



| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumberType | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | Yes | NULL | 0151 264 2500 |
| 2 | 2 | Dennis | Potter | 1935-05-17 | No | NULL | NULL |
| 3 | 3 | Richard | Adams | 1920-05-09 | No | Home | 01565 100 100 |
| 4 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | NULL | NULL |
| 5 | 5 | Walt | Disney | 1966-12-05 | Yes | NULL | NULL |
| 6 | 6 | Barbara | Gordon | 1952-01-11 | No | NULL | NULL |
| 7 | 7 | Josephine | Bailey | 1949-05-31 | Yes | NULL | 07000 200 200 |
| 8 | 8 | Linda | Canoglu | 1959-07-11 | Yes | NULL | 07000 300 300 |
| 9 | 9 | Grace | McQuillan | 1993-09-27 | No | NULL | NULL |
| 10 | 10 | Vera | Black | 1984-08-03 | No | NULL | NULL |
| 11 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL | NULL |
| 12 | 12 | Steve | Davis | 1957-08-22 | Yes | Home | 01928 150 150 |

***Figure 12-11.*** *All contacts and phone numbers are returned—not good!*

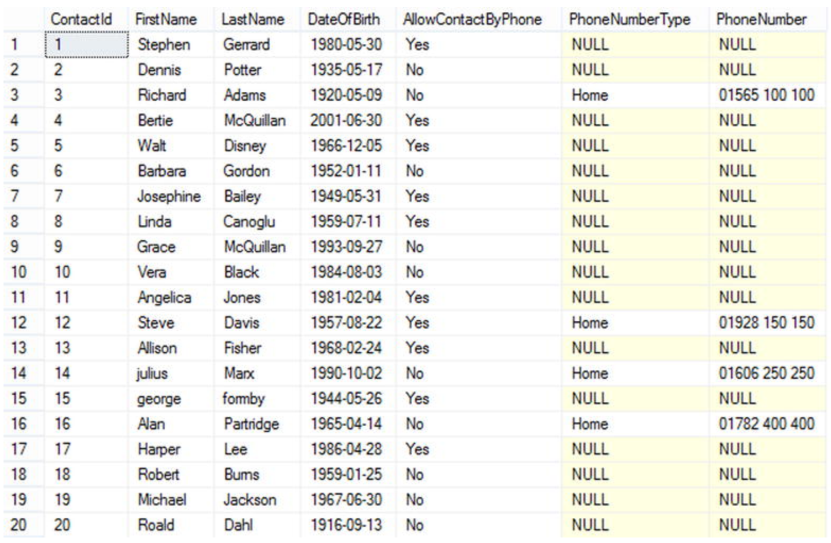
Yes, the `PhoneNumberType` column correctly shows nothing except the `'Home'` value, but there are several rows with a `PhoneNumber` value and no `PhoneNumberType`. What the heck happened here?

- `FROM dbo.Contacts AS C` returns all contacts. There are 20 records in the set at this point.
- `LEFT OUTER JOIN dbo.ContactPhoneNumbers` returns all contacts, and any matching phone numbers. The set now contains 27 records.
- `LEFT OUTER JOIN dbo.PhoneNumberTypes` has been limited to only return the `'Home'` phone number type. The set still contains 27 records. The `'Home'` filter only applies to records returned from the `PhoneNumberTypes` table—it's in the wrong place.

This result set is actually worse than our first effort, as the data are inconsistent. A user would ask why we have phone numbers without phone number types. To successfully return all contacts and only the home phone numbers, we need to obtain all contacts, and then only records from `ContactPhoneNumbers` that have a `PhoneNumberTypeId` of `1`. The `PhoneNumberTypes` table is irrelevant in terms of limiting the result set. This query should do the trick; Figure 12-12 has the result.

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth,
   CASE C.AllowContactByPhone WHEN 1 THEN 'Yes' ELSE 'No' END AS AllowContactByPhone,
   PNT.PhoneNumberType, CPN.PhoneNumber
FROM dbo.Contacts AS C
LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId AND CPN.PhoneNumberTypeId = 1
LEFT OUTER JOIN dbo.PhoneNumberTypes AS PNT
```

```
ON CPN.PhoneNumberTypeId = PNT.PhoneNumberTypeId
ORDER BY C.ContactId ASC;
```



| | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | PhoneNumberType | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | Yes | NULL | NULL |
| 2 | 2 | Dennis | Potter | 1935-05-17 | No | NULL | NULL |
| 3 | 3 | Richard | Adams | 1920-05-09 | No | Home | 01565 100 100 |
| 4 | 4 | Bertie | McQuillan | 2001-06-30 | Yes | NULL | NULL |
| 5 | 5 | Walt | Disney | 1966-12-05 | Yes | NULL | NULL |
| 6 | 6 | Barbara | Gordon | 1952-01-11 | No | NULL | NULL |
| 7 | 7 | Josephine | Bailey | 1949-05-31 | Yes | NULL | NULL |
| 8 | 8 | Linda | Canoglu | 1959-07-11 | Yes | NULL | NULL |
| 9 | 9 | Grace | McQuillan | 1993-09-27 | No | NULL | NULL |
| 10 | 10 | Vera | Black | 1984-08-03 | No | NULL | NULL |
| 11 | 11 | Angelica | Jones | 1981-02-04 | Yes | NULL | NULL |
| 12 | 12 | Steve | Davis | 1957-08-22 | Yes | Home | 01928 150 150 |
| 13 | 13 | Allison | Fisher | 1968-02-24 | Yes | NULL | NULL |
| 14 | 14 | julius | Marx | 1990-10-02 | No | Home | 01606 250 250 |
| 15 | 15 | george | formby | 1944-05-26 | Yes | NULL | NULL |
| 16 | 16 | Alan | Partridge | 1965-04-14 | No | Home | 01782 400 400 |
| 17 | 17 | Harper | Lee | 1986-04-28 | Yes | NULL | NULL |
| 18 | 18 | Robert | Burns | 1959-01-25 | No | NULL | NULL |
| 19 | 19 | Michael | Jackson | 1967-06-30 | No | NULL | NULL |
| 20 | 20 | Roald | Dahl | 1916-09-13 | No | NULL | NULL |

***Figure 12-12**. A perfect record set!*

Eureka!

As Figure 12-12 highlights, all 20 contacts are returned, and just the home phone numbers are displayed. Why did this query work when the others failed?

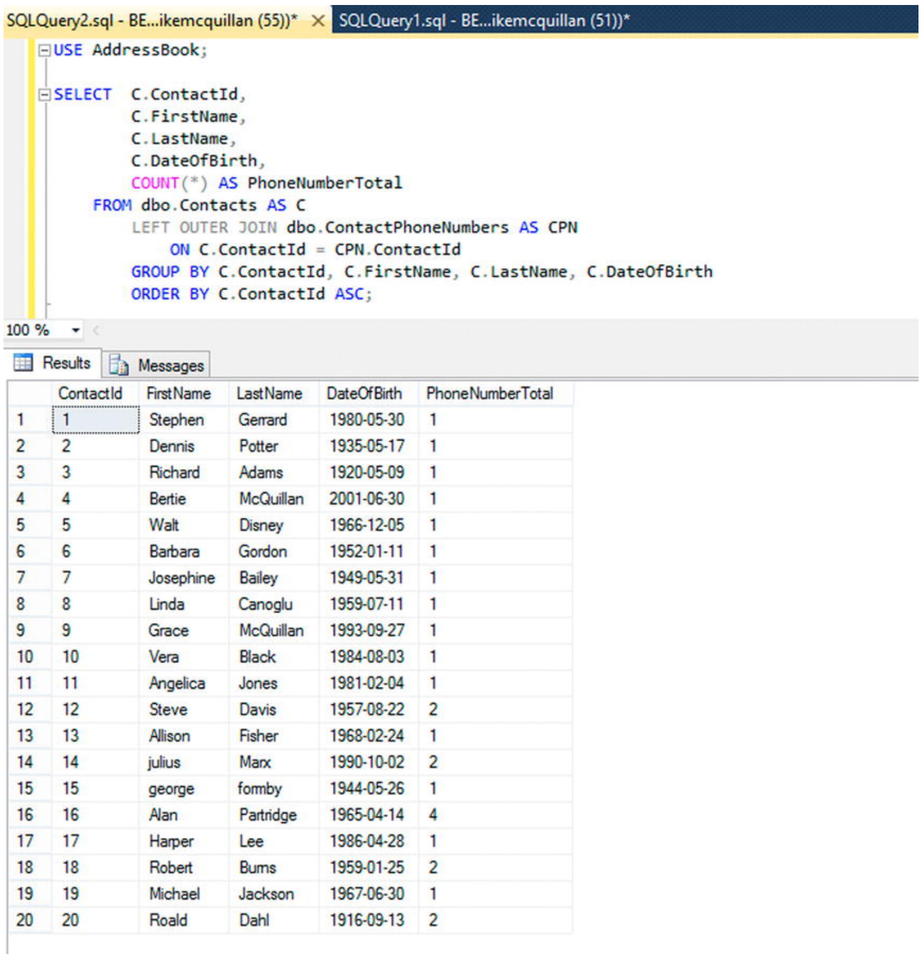
- `FROM dbo.Contacts AS C` returns all contacts. There are 20 records in the set at this point.
- `LEFT OUTER JOIN dbo.ContactPhoneNumbers` returns all contacts, and any matching phone numbers that are of type `'Home'` (`PhoneNumberTypeId = 1`). The set still contains 20 records.
- `LEFT OUTER JOIN dbo.PhoneNumberTypes` returns the phone number type for matching records. As the set only contains blank phone numbers or home phone numbers, only the `'Home'` type will be displayed. The set still contains 20 records.

These examples illustrate just how important it is to think about how you construct your `JOIN`s. With great power comes great responsibility!

We have one last example to look at for `LEFT OUTER JOIN`. We can use `LEFT OUTER JOIN` to figure out how many phone numbers we have per contact, using `GROUP BY`. Pop this query into SSMS and run it.

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, COUNT(*) AS PhoneNumberTotal FROM dbo.C
```

Figure 12-13 has the results.



SQLQuery2.sql - BE...ikemcquillan (55))* SQLQuery1.sql - BE...ikemcquillan (51))*

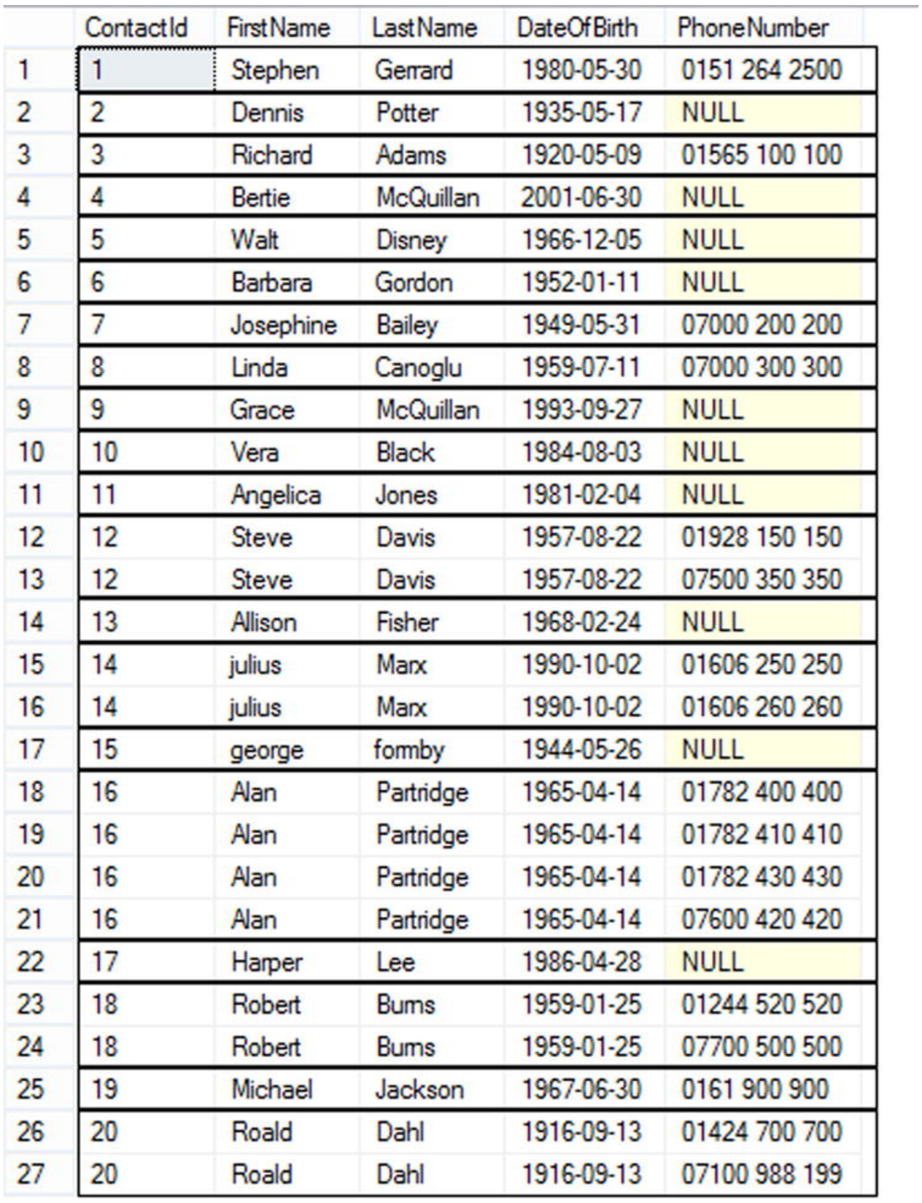
```
USE AddressBook;

SELECT  C.ContactId,
        C.FirstName,
        C.LastName,
        C.DateOfBirth,
        COUNT(*) AS PhoneNumberTotal
    FROM dbo.Contacts AS C
        LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
            ON C.ContactId = CPN.ContactId
        GROUP BY C.ContactId, C.FirstName, C.LastName, C.DateOfBirth
        ORDER BY C.ContactId ASC;
```

|  | ContactId | FirstName | LastName | DateOfBirth | PhoneNumberTotal |
|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | 1 |
| 2 | 2 | Dennis | Potter | 1935-05-17 | 1 |
| 3 | 3 | Richard | Adams | 1920-05-09 | 1 |
| 4 | 4 | Bertie | McQuillan | 2001-06-30 | 1 |
| 5 | 5 | Walt | Disney | 1966-12-05 | 1 |
| 6 | 6 | Barbara | Gordon | 1952-01-11 | 1 |
| 7 | 7 | Josephine | Bailey | 1949-05-31 | 1 |
| 8 | 8 | Linda | Canoglu | 1959-07-11 | 1 |
| 9 | 9 | Grace | McQuillan | 1993-09-27 | 1 |
| 10 | 10 | Vera | Black | 1984-08-03 | 1 |
| 11 | 11 | Angelica | Jones | 1981-02-04 | 1 |
| 12 | 12 | Steve | Davis | 1957-08-22 | 2 |
| 13 | 13 | Allison | Fisher | 1968-02-24 | 1 |
| 14 | 14 | julius | Marx | 1990-10-02 | 2 |
| 15 | 15 | george | formby | 1944-05-26 | 1 |
| 16 | 16 | Alan | Partridge | 1965-04-14 | 4 |
| 17 | 17 | Harper | Lee | 1986-04-28 | 1 |
| 18 | 18 | Robert | Burns | 1959-01-25 | 2 |
| 19 | 19 | Michael | Jackson | 1967-06-30 | 1 |
| 20 | 20 | Roald | Dahl | 1916-09-13 | 2 |

*Figure 12-13.* Results of a `LEFT JOIN` with `GROUP BY`

This doesn't look right—the `PhoneNumberTotal` column is showing at least one phone number exists for every contact. We know this isn't the case. Why has this happened? Well, in this particular instance the fault doesn't lie with the `LEFT OUTER JOIN`; it lies with the `COUNT(*)` statement. `COUNT(*)` tells SQL Server to include every row when counting. Because every contact exists in the result set at least once, regardless of whether they have a phone number or not, a value of at least 1 will always be returned. Higher numbers will be seen for those records that have more than one phone number. Figure 12-14 shows the set without the `GROUP BY`.



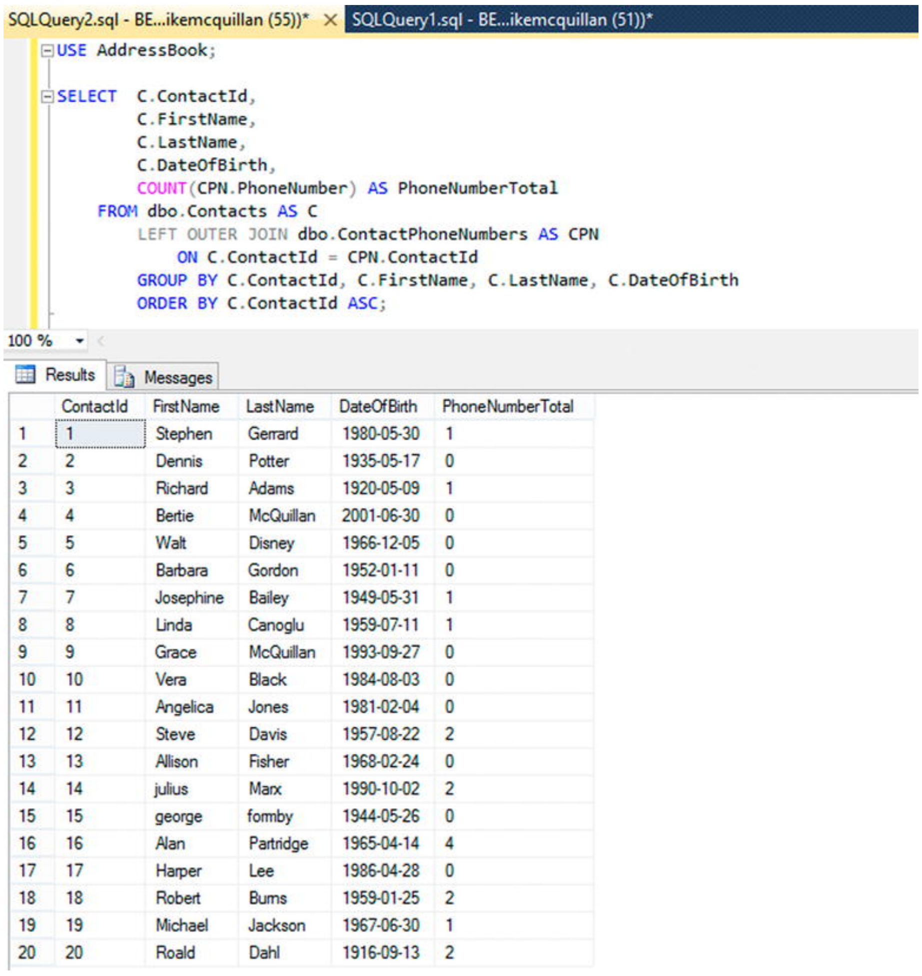
| | ContactId | FirstName | LastName | DateOfBirth | PhoneNumber |
|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | 0151 264 2500 |
| 2 | 2 | Dennis | Potter | 1935-05-17 | NULL |
| 3 | 3 | Richard | Adams | 1920-05-09 | 01565 100 100 |
| 4 | 4 | Bertie | McQuillan | 2001-06-30 | NULL |
| 5 | 5 | Walt | Disney | 1966-12-05 | NULL |
| 6 | 6 | Barbara | Gordon | 1952-01-11 | NULL |
| 7 | 7 | Josephine | Bailey | 1949-05-31 | 07000 200 200 |
| 8 | 8 | Linda | Canoglu | 1959-07-11 | 07000 300 300 |
| 9 | 9 | Grace | McQuillan | 1993-09-27 | NULL |
| 10 | 10 | Vera | Black | 1984-08-03 | NULL |
| 11 | 11 | Angelica | Jones | 1981-02-04 | NULL |
| 12 | 12 | Steve | Davis | 1957-08-22 | 01928 150 150 |
| 13 | 12 | Steve | Davis | 1957-08-22 | 07500 350 350 |
| 14 | 13 | Allison | Fisher | 1968-02-24 | NULL |
| 15 | 14 | julius | Marx | 1990-10-02 | 01606 250 250 |
| 16 | 14 | julius | Marx | 1990-10-02 | 01606 260 260 |
| 17 | 15 | george | formby | 1944-05-26 | NULL |
| 18 | 16 | Alan | Partridge | 1965-04-14 | 01782 400 400 |
| 19 | 16 | Alan | Partridge | 1965-04-14 | 01782 410 410 |
| 20 | 16 | Alan | Partridge | 1965-04-14 | 01782 430 430 |
| 21 | 16 | Alan | Partridge | 1965-04-14 | 07600 420 420 |
| 22 | 17 | Harper | Lee | 1986-04-28 | NULL |
| 23 | 18 | Robert | Burns | 1959-01-25 | 01244 520 520 |
| 24 | 18 | Robert | Burns | 1959-01-25 | 07700 500 500 |
| 25 | 19 | Michael | Jackson | 1967-06-30 | 0161 900 900 |
| 26 | 20 | Roald | Dahl | 1916-09-13 | 01424 700 700 |
| 27 | 20 | Roald | Dahl | 1916-09-13 | 07100 988 199 |

***Figure 12-14****. Phone number data grouped by contact*

Most of these groups contain one record. We can see a few with two, and Alan Partridge has four records. The key to fixing our `GROUP BY` is the `PhoneNumber` column. Some of the records have a `NULL PhoneNumber` value—we need to `COUNT` using this column. The `NULL` will cause a zero value to be returned. If we change `COUNT(*)` to `COUNT(CPN.PhoneNumber)`:

```
SELECT C.ContactId, C.FirstName, C.LastName, C.DateOfBirth, COUNT(CPN.PhoneNumber) AS PhoneNumberTo
FROM dbo.Contacts AS C
LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
ON C.ContactId = CPN.ContactId GROUP BY C.ContactId, C.FirstName, C.LastName, C.DateOfBirth
ORDER BY C.ContactId ASC;
```

Our query will work as expected. You can see the outcome in Figure 12-15.


SQLQuery2.sql - BE...ikemcquillan (55))*
SQLQuery1.sql - BE...ikemcquillan (51))*
100 %
Results
Messages

```
USE AddressBook;

SELECT  C.ContactId,
        C.FirstName,
        C.LastName,
        C.DateOfBirth,
        COUNT(CPN.PhoneNumber) AS PhoneNumberTotal
    FROM dbo.Contacts AS C
        LEFT OUTER JOIN dbo.ContactPhoneNumbers AS CPN
            ON C.ContactId = CPN.ContactId
        GROUP BY C.ContactId, C.FirstName, C.LastName, C.DateOfBirth
        ORDER BY C.ContactId ASC;
```

| | ContactId | FirstName | LastName | DateOfBirth | PhoneNumberTotal |
|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | 1 |
| 2 | 2 | Dennis | Potter | 1935-05-17 | 0 |
| 3 | 3 | Richard | Adams | 1920-05-09 | 1 |
| 4 | 4 | Bertie | McQuillan | 2001-06-30 | 0 |
| 5 | 5 | Walt | Disney | 1966-12-05 | 0 |
| 6 | 6 | Barbara | Gordon | 1952-01-11 | 0 |
| 7 | 7 | Josephine | Bailey | 1949-05-31 | 1 |
| 8 | 8 | Linda | Canoglu | 1959-07-11 | 1 |
| 9 | 9 | Grace | McQuillan | 1993-09-27 | 0 |
| 10 | 10 | Vera | Black | 1984-08-03 | 0 |
| 11 | 11 | Angelica | Jones | 1981-02-04 | 0 |
| 12 | 12 | Steve | Davis | 1957-08-22 | 2 |
| 13 | 13 | Allison | Fisher | 1968-02-24 | 0 |
| 14 | 14 | julius | Marx | 1990-10-02 | 2 |
| 15 | 15 | george | fomby | 1944-05-26 | 0 |
| 16 | 16 | Alan | Partridge | 1965-04-14 | 4 |
| 17 | 17 | Harper | Lee | 1986-04-28 | 0 |
| 18 | 18 | Robert | Burns | 1959-01-25 | 2 |
| 19 | 19 | Michael | Jackson | 1967-06-30 | 1 |
| 20 | 20 | Roald | Dahl | 1916-09-13 | 2 |

***Figure 12-15***. *Returning home number counts with* `LEFT JOIN`

We've covered some pretty interesting stuff here. The crucial thing to understand is how easy it is to output the wrong results, even when the query appears at face value to be correct. Now that we've thoroughly explored the `LEFT OUTER JOIN`, we'll take a look at its twin brother, `RIGHT OUTER JOIN`.

### RIGHT OUTER JOIN

`LEFT OUTER JOIN`s return everything from the left-hand table along with matching records from the right-hand table. As you might have guessed, `RIGHT OUTER JOIN`s return everything from the right-hand table, along with matching records from the left-hand table. The black area in Figure 12-16 shows how data is returned from a `RIGHT OUTER JOIN`.

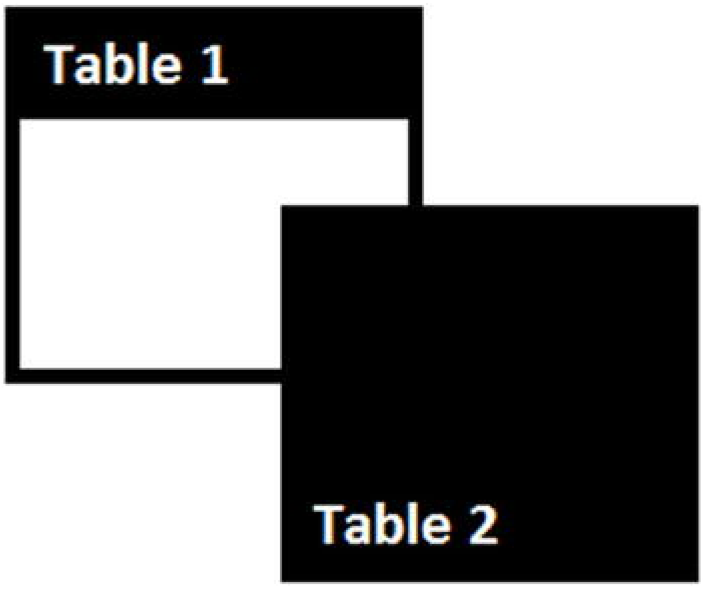
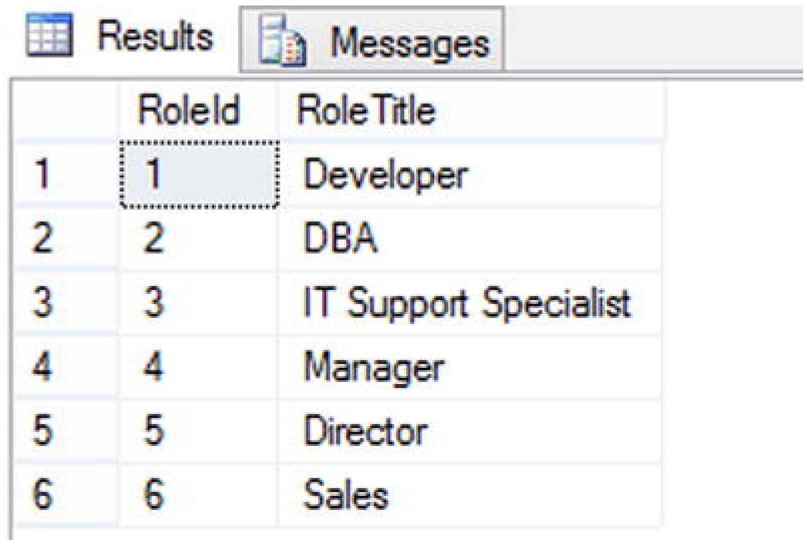


*Figure 12-16.* *Data returned by a* `RIGHT JOIN`

To demonstrate how this works, we first need to add a new `Role` record.

```
INSERT INTO dbo.Roles (RoleTitle) VALUES ('Sales');
```

This leaves us with the six roles you can see in Figure 12-17.



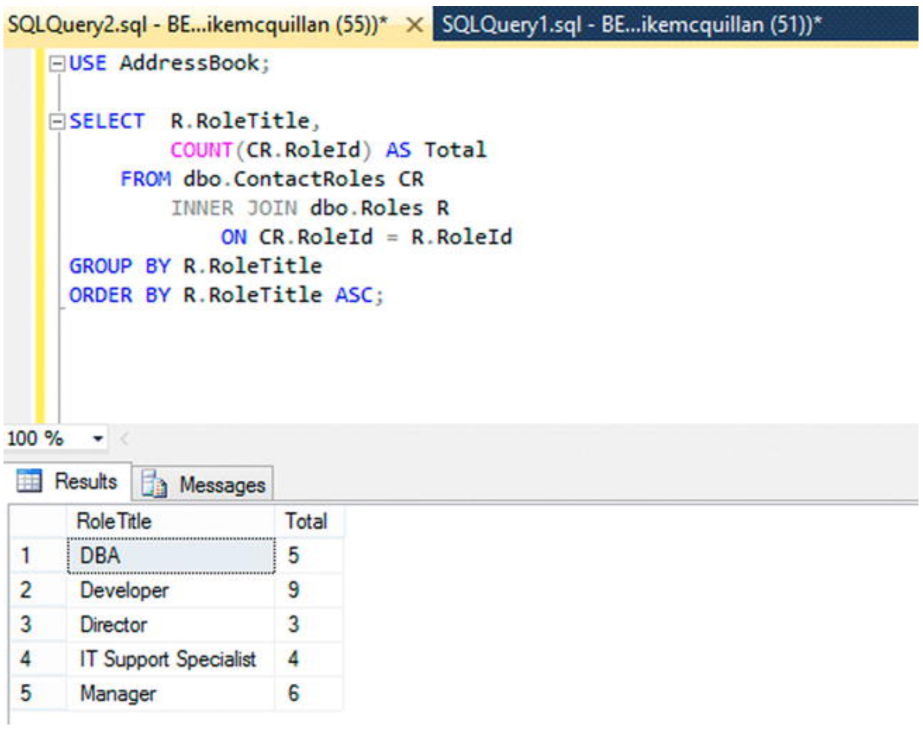
| | RoleId | RoleTitle |
|---|---|---|
| 1 | 1 | Developer |
| 2 | 2 | DBA |
| 3 | 3 | IT Support Specialist |
| 4 | 4 | Manager |
| 5 | 5 | Director |
| 6 | 6 | Sales |

*Figure 12-17.* *A new Sales role*

Our aim here is to return how many contacts there are per role in the database. We'll start off with an `INNER JOIN` to create this.

```
SELECT R.RoleTitle, COUNT(CR.RoleId) AS Total
FROM dbo.ContactRoles CR INNER JOIN dbo.Roles R
ON CR.RoleId = R.RoleId GROUP BY R.RoleTitle
ORDER BY R.RoleTitle ASC;
```

This returns the five results displayed in Figure 12-18.


SQLQuery2.sql - BE...ikemcquillan (55))*
SQLQuery1.sql - BE...ikemcquillan (51))*
100 %
Results
Messages

```
USE AddressBook;

SELECT  R.RoleTitle,
        COUNT(CR.RoleId) AS Total
    FROM dbo.ContactRoles CR
        INNER JOIN dbo.Roles R
            ON CR.RoleId = R.RoleId
GROUP BY R.RoleTitle
ORDER BY R.RoleTitle ASC;
```

| | RoleTitle | Total |
|---|---|---|
| 1 | DBA | 5 |
| 2 | Developer | 9 |
| 3 | Director | 3 |
| 4 | IT Support Specialist | 4 |
| 5 | Manager | 6 |

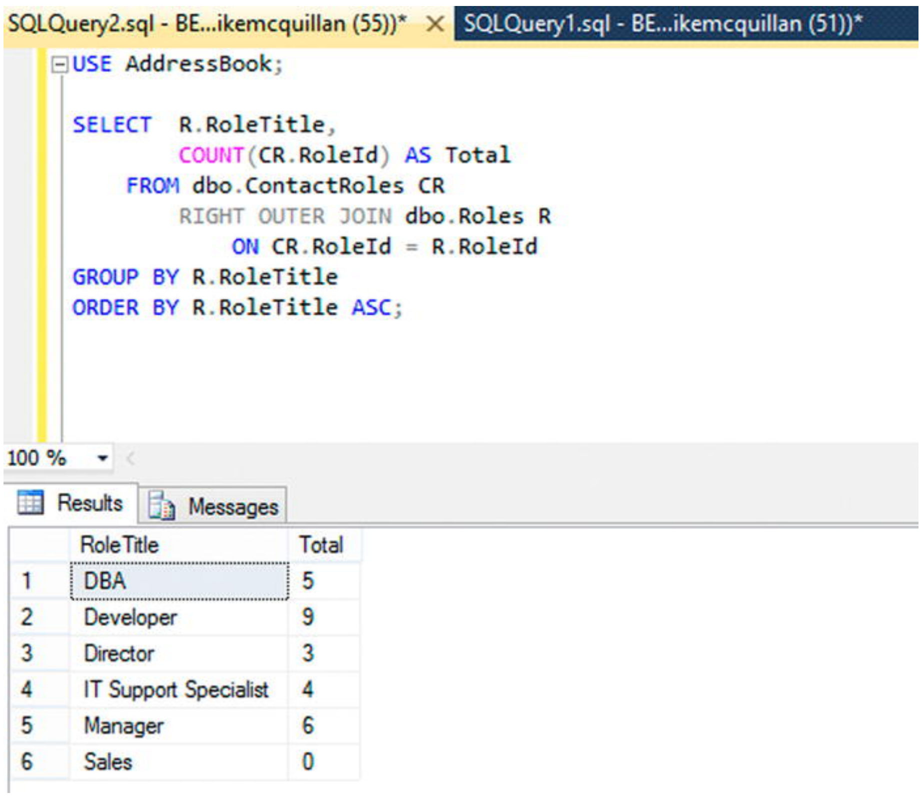
***Figure 12-18.*** *Showing the number of contacts per role—no Sales*

We are missing the total for the new Sales role we just added. This is because we are using an `INNER JOIN`, and no records in `ContactRoles` are linked to this new role. Changing to a `LEFT JOIN` brings back exactly the same result, as it brings back everything from the left-hand table—`ContactRoles`—and matching records from the `Roles` table on the right-hand side.

The fix for this is to change to a `RIGHT OUTER JOIN`. This will return all records from the right-hand table—`Roles`—before executing the `GROUP BY` on the matching records returned by the left-hand table, `ContactRoles`.

```
SELECT R.RoleTitle, COUNT(CR.RoleId) AS Total
FROM dbo.ContactRoles CR
RIGHT OUTER JOIN dbo.Roles R
ON CR.RoleId = R.RoleId
GROUP BY R.RoleTitle
ORDER BY R.RoleTitle ASC;
```

This time, success (see Figure 12-19). We rock!


SQLQuery2.sql - BE...ikemcquillan (55))*
SQLQuery1.sql - BE...ikemcquillan (51))*
100 %
Results
Messages

```
USE AddressBook;

SELECT  R.RoleTitle,
        COUNT(CR.RoleId) AS Total
    FROM dbo.ContactRoles CR
        RIGHT OUTER JOIN dbo.Roles R
            ON CR.RoleId = R.RoleId
GROUP BY R.RoleTitle
ORDER BY R.RoleTitle ASC;
```

|   | RoleTitle | Total |
|---|---|---|
| 1 | DBA | 5 |
| 2 | Developer | 9 |
| 3 | Director | 3 |
| 4 | IT Support Specialist | 4 |
| 5 | Manager | 6 |
| 6 | Sales | 0 |

***Figure 12-19.*** *`RIGHT JOIN` shows Sales!*

If you want to delve further into `RIGHT OUTER JOIN`s, try changing the queries we wrote when exploring `LEFT OUTER JOIN`s to use `RIGHT OUTER JOIN`s. As I mentioned earlier, you'll usually use a `LEFT OUTER JOIN` rather than a `RIGHT OUTER JOIN` if you are writing a query from scratch. I tend to use `RIGHT OUTER JOIN`s when I'm refactoring an existing query and using a `RIGHT OUTER JOIN` will save me from moving tables around.

One `OUTER JOIN` type left to look at. . . .

### FULL OUTER JOIN

This type of join is seldom used, but when it's needed it can be a real time saver. So far, the joins we've seen allow us to

- return exact matches from two tables (`INNER JOIN`), and
- return all records from one table, and matching records from another (`LEFT` and `RIGHT OUTER JOIN`s).

`FULL OUTER JOIN` does something different. It returns every row from both tables (represented by the completely black squares in Figure 12-20). We can bring back all contacts and all roles that match, but we can also return contacts that are not associated with a role, and roles that are not associated with a contact.

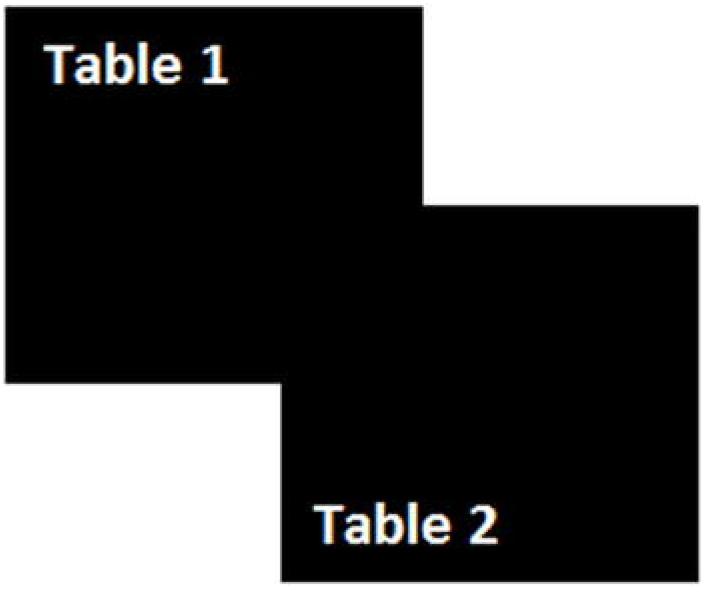


*Figure 12-20. Data returned by a* `FULL OUTER JOIN`

To illustrate this join, we need to add a new row to the `Contacts` table. We'll add Keith Chegwin ("Cheggers" to his friends):

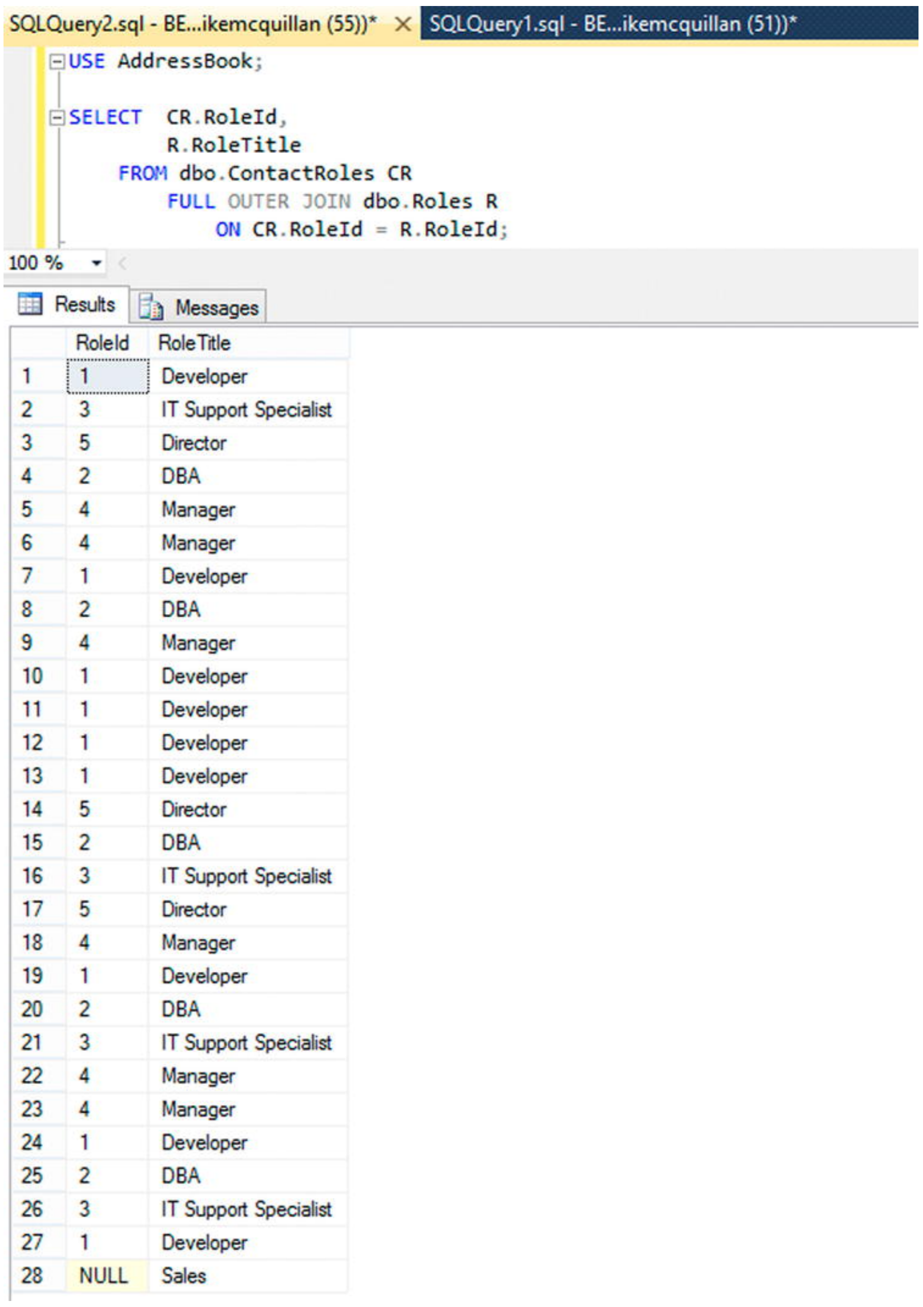
```
INSERT INTO dbo.Contacts(FirstName, LastName, DateOfBirth, AllowContactByPhone)
VALUES ('Keith', 'Chegwin', '1957-01-17', 1);
```

We'll use three tables in our example: `ContactRoles`, `Contacts`, and `Roles`. These three tables together form a many-to-many relationship between `Contacts` and `Roles`. We know we have one role (Sales) that is not linked to any `Contacts`. And we now have a contact (Keith Chegwin) who is not linked to a role. What we want to do is return all contacts with their matching roles, but also the Sales role and the Keith Chegwin contact.

We'll handle the `Roles` part first.

```
SELECT CR.RoleId, R.RoleTitle
FROM dbo.ContactRoles CR FULL OUTER JOIN dbo.Roles R
ON CR.RoleId = R.RoleId;
```

As it stands, this query returns all records that match in `ContactRoles` and `Roles`, as well as any records in `Roles` that do not exist in `ContactRoles` (you can see the results in Figure 12-21). We could have done the same thing with a `RIGHT OUTER JOIN`.



SQLQuery2.sql - BE...ikemcquillan (55))* × SQLQuery1.sql - BE...ikemcquillan (51))*

```
USE AddressBook;

SELECT   CR.RoleId,
         R.RoleTitle
     FROM dbo.ContactRoles CR
         FULL OUTER JOIN dbo.Roles R
             ON CR.RoleId = R.RoleId;
```

100 %

Results | Messages

| | RoleId | RoleTitle |
|---|---|---|
| 1 | 1 | Developer |
| 2 | 3 | IT Support Specialist |
| 3 | 5 | Director |
| 4 | 2 | DBA |
| 5 | 4 | Manager |
| 6 | 4 | Manager |
| 7 | 1 | Developer |
| 8 | 2 | DBA |
| 9 | 4 | Manager |
| 10 | 1 | Developer |
| 11 | 1 | Developer |
| 12 | 1 | Developer |
| 13 | 1 | Developer |
| 14 | 5 | Director |
| 15 | 2 | DBA |
| 16 | 3 | IT Support Specialist |
| 17 | 5 | Director |
| 18 | 4 | Manager |
| 19 | 1 | Developer |
| 20 | 2 | DBA |
| 21 | 3 | IT Support Specialist |
| 22 | 4 | Manager |
| 23 | 4 | Manager |
| 24 | 1 | Developer |
| 25 | 2 | DBA |
| 26 | 3 | IT Support Specialist |
| 27 | 1 | Developer |
| 28 | NULL | Sales |

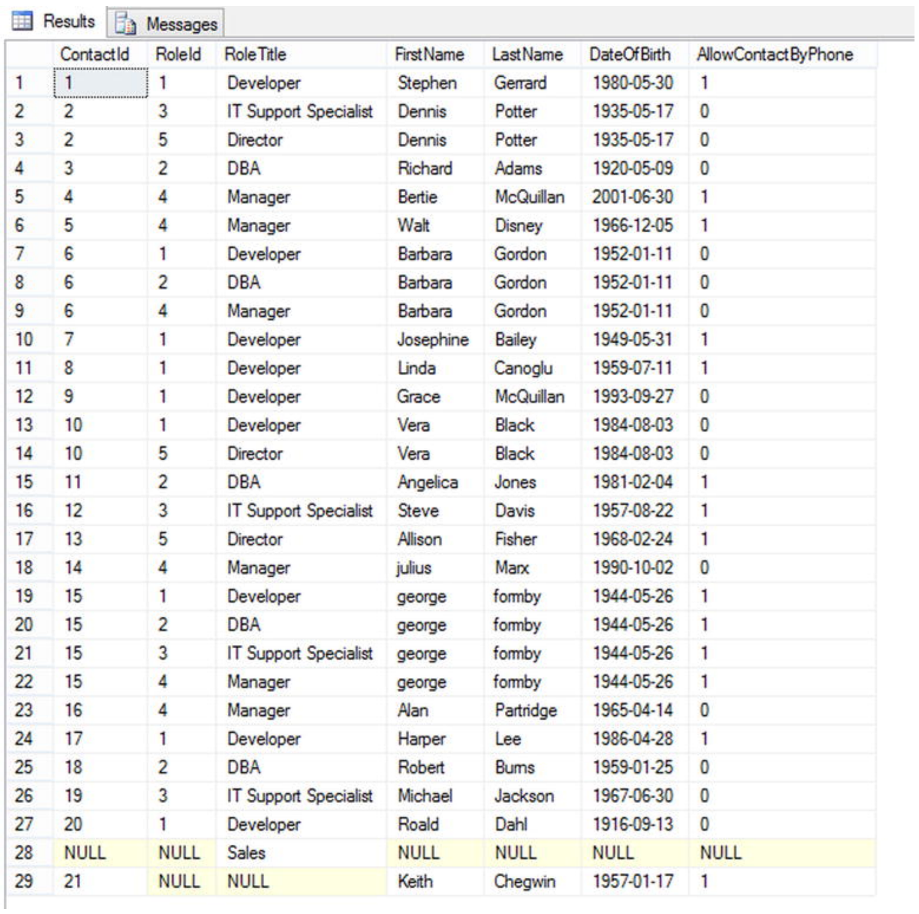
***Figure 12-21***. *Returning all* `ContactRoles` *and* `Roles`

Note the `NULL RoleId` in the final record, for Sales. This is because no records exist for Sales in the `ContactRoles` table.

Now it's time to introduce the power of `FULL OUTER JOIN`. Add a second `FULL OUTER JOIN`, linking to `Contacts` this time.

```
SELECT C.ContactId, CR.RoleId, R.RoleTitle, C.FirstName,
  C.LastName, C.DateOfBirth, C.AllowContactByPhone
FROM dbo.ContactRoles CR FULL OUTER JOIN dbo.Roles R
ON CR.RoleId = R.RoleId
FULL OUTER JOIN dbo.Contacts C ON CR.ContactId = C.ContactId;
```

What should happen now is all records that exist in all three tables should be returned. The Sales record in tl `Roles` table should be returned with mostly `NULL` values, and the Keith Chegwin record in `Contacts` shoul also be returned, again with `NULL` values in all columns except for columns returned via the `Contacts` table The full result set is shown in Figure 12-22; the last two rows show the unmatched records.



Results | Messages

| | ContactId | RoleId | RoleTitle | FirstName | LastName | DateOfBirth | AllowContactByPhone |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | Developer | Stephen | Gerrard | 1980-05-30 | 1 |
| 2 | 2 | 3 | IT Support Specialist | Dennis | Potter | 1935-05-17 | 0 |
| 3 | 2 | 5 | Director | Dennis | Potter | 1935-05-17 | 0 |
| 4 | 3 | 2 | DBA | Richard | Adams | 1920-05-09 | 0 |
| 5 | 4 | 4 | Manager | Bertie | McQuillan | 2001-06-30 | 1 |
| 6 | 5 | 4 | Manager | Walt | Disney | 1966-12-05 | 1 |
| 7 | 6 | 1 | Developer | Barbara | Gordon | 1952-01-11 | 0 |
| 8 | 6 | 2 | DBA | Barbara | Gordon | 1952-01-11 | 0 |
| 9 | 6 | 4 | Manager | Barbara | Gordon | 1952-01-11 | 0 |
| 10 | 7 | 1 | Developer | Josephine | Bailey | 1949-05-31 | 1 |
| 11 | 8 | 1 | Developer | Linda | Canoglu | 1959-07-11 | 1 |
| 12 | 9 | 1 | Developer | Grace | McQuillan | 1993-09-27 | 0 |
| 13 | 10 | 1 | Developer | Vera | Black | 1984-08-03 | 0 |
| 14 | 10 | 5 | Director | Vera | Black | 1984-08-03 | 0 |
| 15 | 11 | 2 | DBA | Angelica | Jones | 1981-02-04 | 1 |
| 16 | 12 | 3 | IT Support Specialist | Steve | Davis | 1957-08-22 | 1 |
| 17 | 13 | 5 | Director | Allison | Fisher | 1968-02-24 | 1 |
| 18 | 14 | 4 | Manager | julius | Marx | 1990-10-02 | 0 |
| 19 | 15 | 1 | Developer | george | formby | 1944-05-26 | 1 |
| 20 | 15 | 2 | DBA | george | formby | 1944-05-26 | 1 |
| 21 | 15 | 3 | IT Support Specialist | george | formby | 1944-05-26 | 1 |
| 22 | 15 | 4 | Manager | george | formby | 1944-05-26 | 1 |
| 23 | 16 | 4 | Manager | Alan | Partridge | 1965-04-14 | 0 |
| 24 | 17 | 1 | Developer | Harper | Lee | 1986-04-28 | 1 |
| 25 | 18 | 2 | DBA | Robert | Burns | 1959-01-25 | 0 |
| 26 | 19 | 3 | IT Support Specialist | Michael | Jackson | 1967-06-30 | 0 |
| 27 | 20 | 1 | Developer | Roald | Dahl | 1916-09-13 | 0 |
| 28 | NULL | NULL | Sales | NULL | NULL | NULL | NULL |
| 29 | 21 | NULL | NULL | Keith | Chegwin | 1957-01-17 | 1 |

***Figure 12-22.*** *The last two rows are unmatched*

All of the matched records have returned just as an `INNER JOIN` would. You can think of the `FULL OUTER JOIN` as executing all three of `INNER JOIN`, `LEFT OUTER JOIN`, and `RIGHT OUTER JOIN`. Very useful at times.

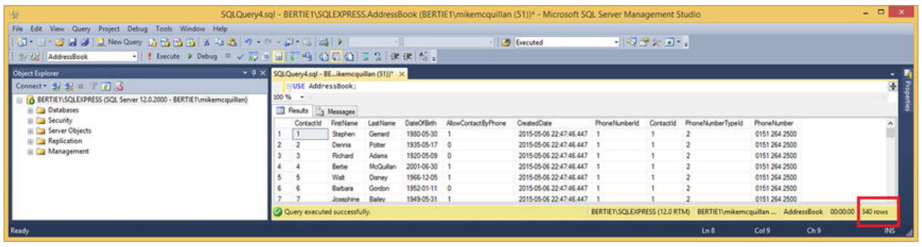
**CROSS JOIN**

Our final join is not used on a regular basis. A `CROSS JOIN` is usually used to generate test data—it has few other practical uses. There is no image for this join as it's quite difficult to show graphically—going straight to an example will work better here. I've rebuilt my database, so any records I've added during this chapter are gone now.

Assume the `Contacts` table has 20 records in it, and the `ContactPhoneNumbers` table has 17 records in it. We've already seen via our other joins how we can return this information, ensuring the correct parent is mapped to the correct children by specifying a join condition. There's no such condition here—a `CROSS JOIN` doesn't allow you to specify one. Write this:

```
SELECT * FROM dbo.Contacts AS C
CROSS JOIN dbo.ContactPhoneNumbers AS CPN;
```

Without any criteria specified to dictate how the tables should be joined, how many rows do you think will be returned here? Go on, try it. Make sure you look at the row count returned in the bottom right-hand corner (I've put a box around it in Figure 12-23).



```
USE AddressBook;
```

|  | ContactId | FirstName | LastName | DateOfBirth | AllowContactByPhone | CreatedDate | PhoneNumberId | ContactId | PhoneNumberTypeId | PhoneNumber |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Stephen | Gerrard | 1980-05-30 | 1 | 2015-05-06 22:47:46.447 | 1 | 1 | 2 | 0151 264 2500 |
| 2 | 2 | Dennis | Potter | 1935-05-17 | 0 | 2015-05-06 22:47:46.447 | 1 | 1 | 2 | 0151 264 2500 |
| 3 | 3 | Richard | Adams | 1920-05-09 | 0 | 2015-05-06 22:47:46.447 | 1 | 1 | 2 | 0151 264 2500 |
| 4 | 4 | Bertie | McQuillan | 2001-06-30 | 1 | 2015-05-06 22:47:46.447 | 1 | 1 | 2 | 0151 264 2500 |
| 5 | 5 | Walt | Disney | 1966-12-05 | 1 | 2015-05-06 22:47:46.447 | 1 | 1 | 2 | 0151 264 2500 |
| 6 | 6 | Barbara | Gordon | 1952-01-11 | 0 | 2015-05-06 22:47:46.447 | 1 | 1 | 2 | 0151 264 2500 |
| 7 | 7 | Josephine | Bailey | 1949-05-31 | 1 | 2015-05-06 22:47:46.447 | 1 | 1 | 2 | 0151 264 2500 |

Query executed successfully. BERTIE1\SQLEXPRESS (12.0 RTM) BERTIE1\mikemcquillan ... AddressBook 00:00:00 340 rows

***Figure 12-23.*** *A Cartesian Product via* `CROSS JOIN`

It returns 340 rows! Why, when we should have no more than 20? Well, 20 rows in `Contacts`, and 17 in `ContactPhoneNumbers`: 20 * 17 = 340. This join returns every possible combination of the two tables, so for every row in the `Contacts` table, 17 combinations are returned—one for each `ContactPhoneNumbers` record.

Obviously this is not good, as we have no way of telling which phone number belongs to which contact. But it is only not good if we were trying to use the data in a realistic scenario (i.e., we wanted to call the contact)—we'd use one of the other join types for that. This join would be useful for us if we were trying to generate lots and lots of contacts so we could test the database. Keep this join in mind if you are ever asked to generate some test data.

**CARTESIAN PRODUCTS**

The result set the `CROSS JOIN` returned is known as a *Cartesian Product*. A Cartesian Product occurs when a database query returns every possible row combination across two tables. It is easy to accidentally create Cartesian Product result sets, so ensure you test your queries thoroughly for accidental duplicates.

### Using JOINs In DML Statements

I've covered all the various join types available to us in SQL Server. I'm sure you'll agree there's been quite a bit to take in. Before I wrap up this chapter, we'll look at how `JOIN`s can be used as part of the `INSERT INTO`, `UPDATE` and `DELETE` statements.

Whenever I'm tasked with writing a DML statement, I always begin by writing a `SELECT` statement. Let's say I've been asked to insert a new Senior Developer record into the `Roles` table. Once this is available, I need to change all developers in the database into senior developers. I then have to remove the Developer role from these senior developers. Last, I've been asked to change IT Support Specialists to Developers. Let's break this down into the statements we need to write:

- `INSERT` a new Senior Developer record into the `Roles` table
- `INSERT` Senior Developer records into `ContactRoles`, for those `Contacts` who currently hold the Developer role
- `DELETE` all `ContactRoles` records that have the Developer role assigned to them
- `UPDATE` all `ContactRoles` records that have the IT Support Specialist role assigned to them, and change them to Developer

Let's do the easy part first: adding the new Senior Developer record.

```
INSERT INTO dbo.Roles(RoleTitle) VALUES ('Senior Developer');
```

Run this to add the new role. Now, before we write the `INSERT` Senior Developers statement, we'll write a `SELECT` to obtain the existing Developer `ContactRoles` records. Do this in a New Query Window.

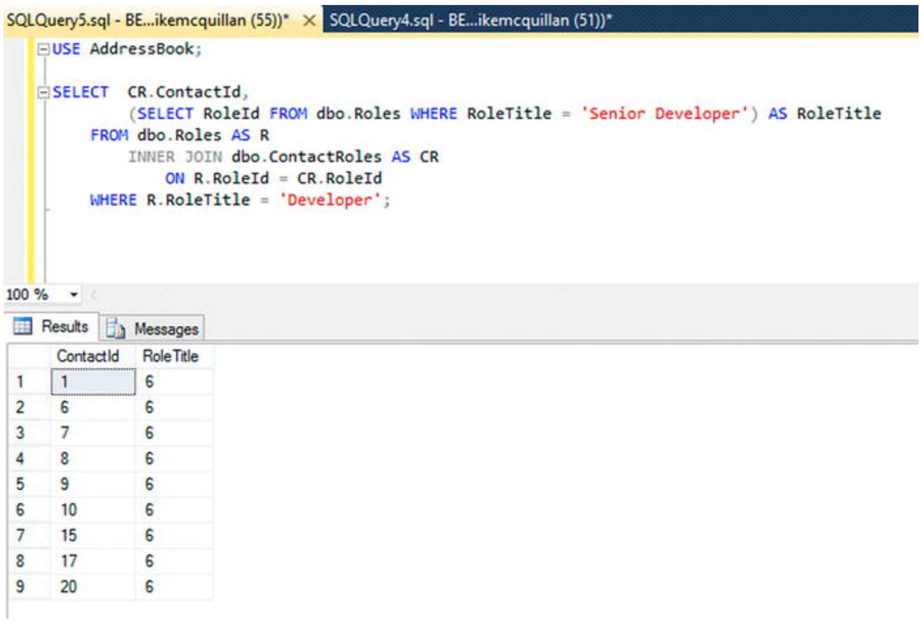
```
SELECT CR.ContactId, CR.RoleId
FROM dbo.Roles AS R INNER JOIN dbo.ContactRoles AS CR
ON R.RoleId = CR.RoleId WHERE R.RoleTitle = 'Developer';
```

This returns the nine `ContactRoles` records. Using `JOIN`s in `INSERT`s is as simple as including a `SELECT` statement under the `INSERT INTO` statement. So we just change the `SELECT` statement to output the

ContactId and the RoleId of the Senior Developer role.

```
SELECT CR.ContactId,
   (SELECT RoleId FROM dbo.Roles WHERE RoleTitle = 'Senior Developer') AS RoleTitle
FROM dbo.Roles AS R INNER JOIN dbo.ContactRoles AS CR
ON R.RoleId = CR.RoleId WHERE R.RoleTitle = 'Developer';
```

We've used something new here—a subquery, to return the RoleId of the Senior Developer role. It's possible to embed queries inside other queries as part of SELECT statements. We have no easy way of using a JOIN to obtain the Senior Developer RoleId (because none of the records we are modifying currently have a Senior Developer role), so we just embedded the SELECT statement to pull it back into our main query. The subquery is seen as a column within the result set (RoleTitle in Figure 12-24).


SQLQuery5.sql - BE...ikemcquillan (55))*
SQLQuery4.sql - BE...ikemcquillan (51))*
100 %
Results
Messages

```
USE AddressBook;

SELECT  CR.ContactId,
        (SELECT RoleId FROM dbo.Roles WHERE RoleTitle = 'Senior Developer') AS RoleTitle
    FROM dbo.Roles AS R
        INNER JOIN dbo.ContactRoles AS CR
            ON R.RoleId = CR.RoleId
    WHERE R.RoleTitle = 'Developer';
```

| | ContactId | RoleTitle |
|---|---|---|
| 1 | 1 | 6 |
| 2 | 6 | 6 |
| 3 | 7 | 6 |
| 4 | 8 | 6 |
| 5 | 9 | 6 |
| 6 | 10 | 6 |
| 7 | 15 | 6 |
| 8 | 17 | 6 |
| 9 | 20 | 6 |

***Figure 12-24.*** *SELECT statement preparation for an INSERT*

Now we can just prepend an INSERT over the top of the SELECT statement.

```
INSERT INTO dbo.ContactRoles(ContactId, RoleId) SELECT CR.ContactId, (SELECT RoleId FROM dbo.Roles
```

Run this and you should see 9 row(s) affected.

Now we need to implement our DELETE statement. We want to delete all Developer records from the ContactRoles table. Again, we begin by writing a SELECT statement to obtain the relevant records. Another reason for writing a SELECT statement to do this is it allows you to check that you are obtaining the correct data. If you just run an UPDATE or DELETE statement without checking the data first, you could process the wrong data.

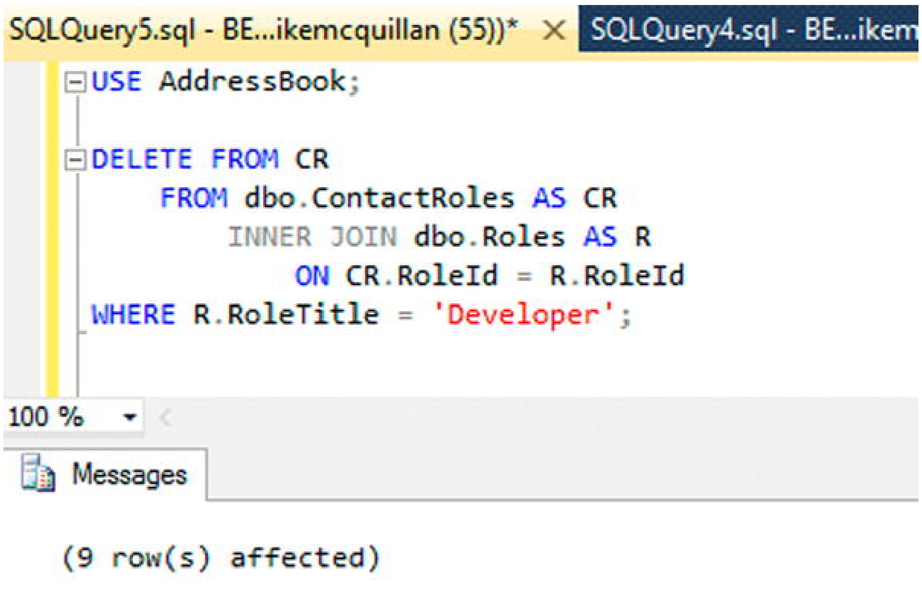
The SELECT statement is pretty simple.

```
SELECT CR.ContactId, CR.RoleId
FROM dbo.ContactRoles AS CR INNER JOIN dbo.Roles AS R
ON CR.RoleId = R.RoleId WHERE R.RoleTitle = 'Developer';
```

This also returns nine rows. This makes perfect sense; we were switching the Developers to Senior Developers. To change this SELECT statement into a DELETE statement, you replace everything above the FROM line with the DELETE statement. As multiple tables are involved in the statement, you must provide the alias of the tal
you wish to delete from.

```
DELETE FROM CR FROM dbo.ContactRoles AS CR
INNER JOIN dbo.Roles AS R
ON CR.RoleId = R.RoleId WHERE R.RoleTitle = 'Developer';
```

Remember, once you have specified a table alias, you should always use that to reference the table in the query. Run this and the same nine rows that were returned by the `SELECT` statement will be deleted (look at the message in Figure 12-25).



SQLQuery5.sql - BE...ikemcquillan (55))*
SQLQuery4.sql - BE...ikem
100 %
Messages

```
USE AddressBook;

DELETE FROM CR
    FROM dbo.ContactRoles AS CR
        INNER JOIN dbo.Roles AS R
            ON CR.RoleId = R.RoleId
WHERE R.RoleTitle = 'Developer';
```

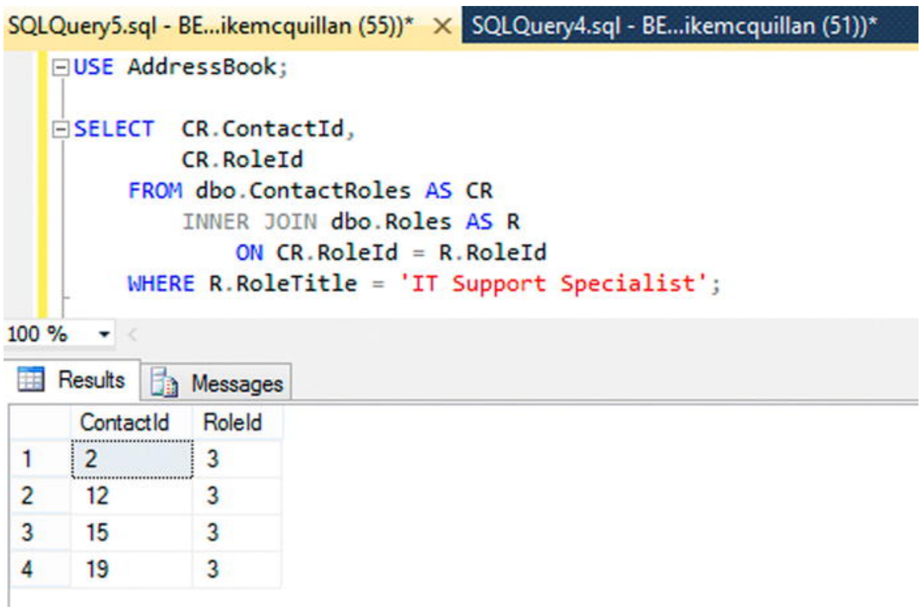
```
(9 row(s) affected)
```

***Figure 12-25.*** *Deleting data using `INNER JOIN`*

If you change this back to a `SELECT` statement now, no rows will be returned. All is as it should be. We can move on to the last task: updating IT Support Specialists to make them Developers. Again, we begin by creating the `SELECT` statement to obtain the IT Support Specialists.

```
SELECT CR.ContactId, CR.RoleId
FROM dbo.ContactRoles AS CR
INNER JOIN dbo.Roles AS R ON CR.RoleId = R.RoleId
WHERE R.RoleTitle = 'IT Support Specialist';
```

This returns the four records shown in Figure 12-26.



SQLQuery5.sql - BE...ikemcquillan (55))* SQLQuery4.sql - BE...ikemcquillan (51))*

```
USE AddressBook;

SELECT  CR.ContactId,
        CR.RoleId
    FROM dbo.ContactRoles AS CR
        INNER JOIN dbo.Roles AS R
            ON CR.RoleId = R.RoleId
    WHERE R.RoleTitle = 'IT Support Specialist';
```

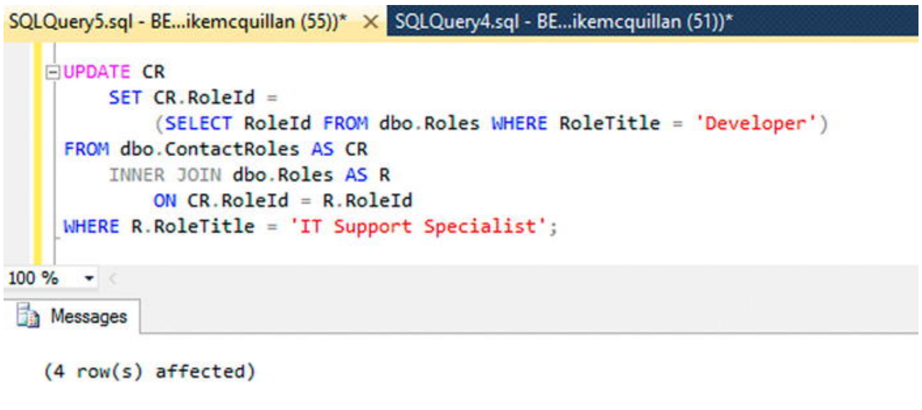
|   | ContactId | RoleId |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 12 | 3 |
| 3 | 15 | 3 |
| 4 | 19 | 3 |

***Figure 12-26.*** *More* `SELECT` *statement preparation for an* `INSERT`

Now we remove the `SELECT` part and replace it with an `UPDATE`, again using the table alias. We also use a subquery again, to obtain the RoleId of the Developer record.

```
UPDATE CR SET CR.RoleId = (SELECT RoleId FROM dbo.Roles WHERE RoleTitle = 'Developer')
   FROM dbo.ContactRoles AS CR INNER JOIN dbo.Roles AS R ON CR.RoleId = R.RoleId
   WHERE R.RoleTitle = 'IT Support Specialist';
```

We already know that everything in the statement from `FROM` onwards will ensure the correct records are processed. We've specified that we want to update `CR` (`ContactRoles`), setting the `RoleId` to Developer. We don't need to change the `ContactId` value. Running this tells us four records have been updated (Figure 12-27).



SQLQuery5.sql - BE...ikemcquillan (55))* SQLQuery4.sql - BE...ikemcquillan (51))*

```
UPDATE CR
    SET CR.RoleId =
        (SELECT RoleId FROM dbo.Roles WHERE RoleTitle = 'Developer')
FROM dbo.ContactRoles AS CR
    INNER JOIN dbo.Roles AS R
        ON CR.RoleId = R.RoleId
WHERE R.RoleTitle = 'IT Support Specialist';
```

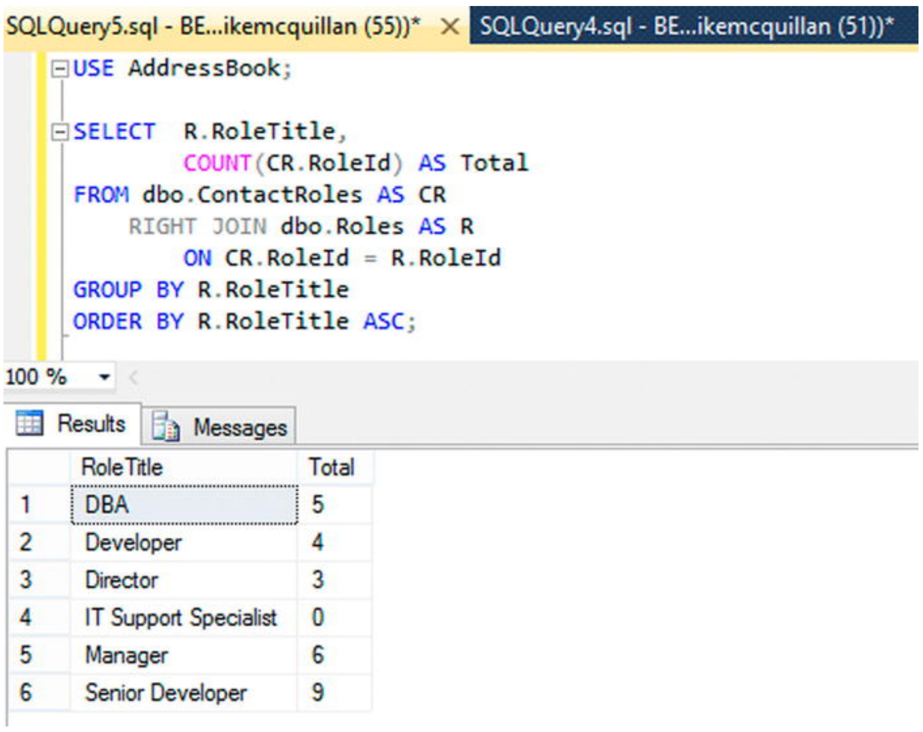
```
(4 row(s) affected)
```

***Figure 12-27.*** *The* `SELECT` *statement converted to an* `UPDATE`

We've successfully modified our data using `JOIN`s. We can run a quick `SELECT` to show how many records of each type we have now.

```
SELECT R.RoleTitle, COUNT(CR.RoleId) AS Total
   FROM dbo.ContactRoles AS CR RIGHT JOIN dbo.Roles AS R ON CR.RoleId = R.RoleId
   GROUP BY R.RoleTitle ORDER BY R.RoleTitle ASC;
```

We should have no IT Support Specialists, four Developers, and nine Senior Developers. Figure 12-28 confir this.


SQLQuery5.sql - BE...ikemcquillan (55))*
SQLQuery4.sql - BE...ikemcquillan (51))*
100 %
Results
Messages

```
USE AddressBook;

SELECT  R.RoleTitle,
        COUNT(CR.RoleId) AS Total
FROM dbo.ContactRoles AS CR
    RIGHT JOIN dbo.Roles AS R
        ON CR.RoleId = R.RoleId
GROUP BY R.RoleTitle
ORDER BY R.RoleTitle ASC;
```

| | RoleTitle | Total |
|---|---|---|
| 1 | DBA | 5 |
| 2 | Developer | 4 |
| 3 | Director | 3 |
| 4 | IT Support Specialist | 0 |
| 5 | Manager | 6 |
| 6 | Senior Developer | 9 |

***Figure 12-28.*** *The final outcome*

**Summary**

Well done, you've learned how one of the most important aspects of SQL Server development works. Joins are essential if you want to do anything useful with SQL Server, especially when it comes to querying. We've actually gone beyond querying, looking at how joins can help us to insert, update, and delete multiple records.

You have many of the pieces you need to become proficient with SQL Server in place. There is still more to learn (isn't there always?). We'll continue to improve our `SELECT` skills by delving into views. Let's walk into the distance. . . .

PREV
Chapter 11 : The SELECT Statem…

NEXT
Chapter 13 : Views