By now, you really are becoming proficient in SQL Server 2012 and writing code to work with the data and the objects within the database. Already you have seen some T-SQL code and encountered some scenarios that have advanced your skills as a T-SQL developer. You can now look at more advanced areas of T-SQL programming to round out your knowledge and really get you going with queries that do more than the basics.

This chapter will cover the occasions when you need a query within a query, known as a subquery. This is ideal for producing a list of values to search for, or for producing a value from another table to set a column or a variable with. It is also possible to create a transient table of data to use within a query, known as a common table expression. You will look at both subqueries and common table expressions in this chapter.

From there, you will explore how to take a set of data and pivot the results, just as you can do within Excel. You will also take a look at different types of ranking functions, where you can take your set of data and attach rankings to rows or groups of rows of data.

Finally, you will see how to deal with varchar(max) and varbinary(max) data types when loading large text, images, movies, and so on.

**Sequences Instead of IDENTITY**

In Chapter 5, you were introduced to IDENTITY columns, in which an identity value would be automatically generated when there was an attempt to add a new row. As I mentioned, gaps could be created either when a single- or multiple-row insertion failed or was rolled back as part of a transaction. IDENTITY-defined columns are ideal for small, low-frequency row insertions, but there is a small performance overhead each time a value is created. It has been requested for some time from developers that Microsoft look at this problem and find a way to improve performance, and so from this problem, Microsoft has developed sequences.

The first benefit of sequences is that it is possible for SQL Server not only to retrieve the next value one at a time from the underlying sequence table (like IDENTITY), but also to cache a range of numbers in memory for faster access and retrieval. A cached set of numbers will work faster than using the IDENTITY property, which is generated with each insertion.

You can also define a sequence so that the sequence can loop around when a maximum value is reached. This allows you to reuse numbers.

It is also possible to retrieve the next sequence value and place it into a local variable before the row is inserted. This will give you the flexibility of knowing the “identity” column value for any child tables’ insertions. With IDENTITY, you can retrieve the value assigned to a row only after the row is inserted. By having the ability to assign the value to a variable with sequence, you can avoid gaps through insertions failing by reusing the value if, for example, a rollback occurs. Gaps will still form, though, if the server is
rebooted or the SQL Server Windows service stopped, so using a local variable is not a complete solution to the gapping issue.

Sequences can deal with generating a value for multiple tables and potentially different columns all from the single sequencing object, unlike the IDENTITY keyword, which is bound to one column in one table. The ability to split sequencing across multiple tables allows you to keep the sequence of data intact if, for example, you have a table holding incoming records but two tables for processed data—one for accepted records and one for rejected records. By inspecting the sequence column value on both these tables, you would be able to see the order in which the rows had been processed.

- **Caution** If you use sequences within a transaction and the transaction is rolled back, then the sequence is *not* rolled back and the values retrieved will look as if they have been used.

Using sequences, you can assign a value to a column programmatically. This functionality removes the problems that can occur when using IDENTITY because IDENTITY column values are awkward to alter within a column after they are assigned. When completing a bulk insertion or even modifying the value within a column, an incorrectly inserted value can be modified when using sequences, as you will see in the ALTER SEQUENCE section of the exercise that follows. The downside, however, is that developers have to be aware that a column is logically linked to a sequence and use the sequence functionality rather than enter values manually. You can apply a sequence as a default value to the relevant column, as shown in Figure 14-1. This will still allow you to insert the sequence value like IDENTITY. However, when I use a sequence in the exercises in this section, I will not use it as a default value so that you can see explicitly what is happening.

![Figure 14-1. Applying a sequence as a default value for a column](image)

Let’s take a look at how to create a sequence, including the syntax and the options that are available.