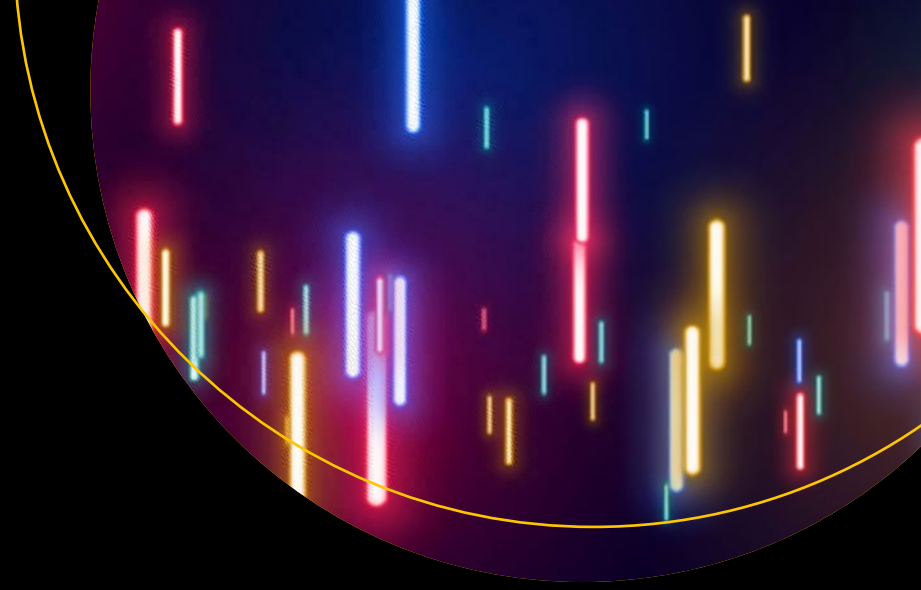

# Pro T-SQL 2019

## Toward Speed, Scalability, and Standardization for SQL Server Developers

Elizabeth Noble

Apress®

# Pro T-SQL 2019

## Toward Speed, Scalability, and Standardization for SQL Server Developers

**Elizabeth Noble**

Apress®

## *Pro T-SQL 2019: Toward Speed, Scalability, and Standardization for SQL Server Developers*

Elizabeth Noble
Roswell, GA, USA

ISBN-13 (pbk): 978-1-4842-5589-6 ISBN-13 (electronic): 978-1-4842-5590-2
https://doi.org/10.1007/978-1-4842-5590-2



Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Jonathan Gennick
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484255896. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

*This book is dedicated to #SQLFamily. You all have helped me grow as a data professional. You have given me confidence and encouraged me to pursue my dreams.*
*I hope this books helps others as much as I have been helped by all of you.*

*This book is also dedicated to my family and friends for all their love and support, especially while writing this book.*
*Eric and Danny, you can accomplish the challenges that come before you.*

# Table of Contents

# About the Author

**Elizabeth Noble** is a Director of Database Development or "E" at Pull-A-Part in the metro Atlanta area. It was love at first sight when she was introduced to her first database over 10 years ago. Her passion is to help others improve the quality and speed of deploying database changes through automation. When she is not trying to automate all things, she can be found spending time with her dogs, playing disc golf, or taking a walk. She is also a frequent speaker at SQL Saturdays across the United States and was a first-time speaker at PASS Summit in 2019.

# About the Technical Reviewer

**Warner Chaves** is a SQL Server MCM, Data Platform MVP, and Principal Consultant at Pythian. He started his career with a brief stint in .NET programming that led to working for enterprise customers in Hewlett-Packard's ITO organization. From there, he transitioned to his current position at Pythian, building and managing data solutions for some of the biggest names across many industry verticals. Warner is a frequent speaker at SQLSaturday events around the world and at the PASS Summit, the largest conference dedicated to the Microsoft Data Platform.

# Acknowledgments

I would like to thank my husband, Casey, for believing in me and encouraging me to pursue my dreams. I want to thank my Mom for teaching me that growth is the key to life. Thanks to my Dad for helping me remember to take time for myself. I also want to thank Ed, Jeff, Mike, Phil, and Rob for mentoring me and introducing me to #SQLFamily.

I want to thank my friends Angela, Joey, Jude, Lee, LeeAnn, and Tammy for their words of encouragement while writing this book. Thanks to Rie for teaching me both the importance of confidence and empathy. I want to thank Kellyn for believing in me and my ability to author a book.

# Introduction

Working with T-SQL allows you to write code and see results relatively quickly. There is also a great deal of flexibility when creating T-SQL statements. In many cases, there is more than one way you can achieve the same outcome from your T-SQL. This book is designed for database developers and data professionals that have a general knowledge of T-SQL but are looking to improve their overall code quality. You should understand T-SQL syntax and know how to write SELECT, INSERT, UPDATE, or DELETE statements before beginning this book. *Pro T-SQL 2019* will prepare you to write consistent code with improved performance. You will also learn how to protect your T-SQL code by using source control and improving your database deployment pipeline. Overall, the goal of this book is to provide you with a framework to write better T-SQL code. As data professionals, we can find ourselves in scenarios where there are high demands or short deadlines. *Pro T-SQL 2019* was written with the intent to help you write code that could save you time and energy in the future.

There are four sections in this book. The first section covers how to improve the readability of your T-SQL code. There is an overview of the various data types for T-SQL along with guidance on how to best use those data types. This first section explains the benefits and disadvantages of the various database objects in SQL Server. Additional chapters discuss standardizing and designing T-SQL code. The second section explains how to write T-SQL code that is efficient. This section includes using set-based design to write T-SQL code and understanding the relationship between hardware and T-SQL design. You will also learn how to use execution plans and new features in SQL Server 2019 to improve the performance of your T-SQL code. The third section discusses how to manage your T-SQL code. The chapters included in this section cover developing coding standards and using source control to store your code. To further manage your T-SQL code, you will also learn some methods to test and deploy your database code. The fourth section addresses how to write T-SQL code so that it is sustainable over time. These chapters include methods to safely add new functionality, to log changes to data within your databases, and to manage data growth over time.

# PART I

# Building Understandable T-SQL

CHAPTER 1

# Data Types

Data types are the building blocks that make the foundation of writing efficient and performant T-SQL. While many data types are either a number or a string, there are also a variety of data types that do not fit into either category. When selecting the right data type, it is important to understand what the data type is and when to use it.

## Number Data Types

While numbers may all seem to be the same, T-SQL segments numbers into different kinds of data types. These data types can include whole numbers or numbers with decimal places. Numbers are also categorized by either being exact or approximate. Understanding how to work with various number data types when performing mathematical calculations is critical to ensuring applications handle data as expected.

While it may be easiest to pick the most common seeming data types from each category, there are specific times where it is best to analyze the data that will be stored and select a more appropriate option. When choosing data types, there are various factors to consider. The most important step is to figure out what kind of data will be stored. The next logical step is to consider how the data will be used and stored. In addition, it is important to understand how T-SQL handles calculations involving various data types.

## Exact Number Data Types

There are situations when the value of a number is definite and known. These types of numbers can be referred to as exact numbers. Some examples include true or false, quantity of units sold, discount percent, or dollars and cents. One of the keys to writing good T-SQL is selecting the correct data type category to use for a given field. In some cases, the categories have more than one data type available.




E. Noble, *Pro T-SQL 2019*, https://doi.org/10.1007/978-1-4842-5590-2_1

When considering what data type to use, you will want to consider the purpose for this data type. This will provide better clarity when determining which data type should be used. You will want to consider both the benefits and drawbacks associated with each data type. You will also want to consider if SQL Server will have to perform any implicit conversions as a result of using this data type in calculations with other data types. The final piece will be to consider how the data type is stored in SQL Server.

## BIT

The term BIT is derived from the phrase binary digit. Therefore, the BIT data type can only store one of two values. In the case of SQL Server, there is a third option relating to unknown values. This third option is known as NULL. As such, the only values allowed with the BIT data type are 0, 1, and NULL. This consequently causes the BIT data type to only allow the smallest set of available values.

The BIT data type is great for data types where "either X or Y" is applicable. The information stored can be true or false, on or off, and yes or no. In the case of true or false, this BIT type could be used to indicate if a data record was translated successfully. A common use of indicating on or off with the BIT data type would involve indicating if a certain feature is enabled. One example of a yes or no value would be to record a customer's decision to opt in to receive marketing information from a company.

One of the challenges with the BIT data type is making sure to use them in a way that promotes good database design. This means there are times when you need to consider the overall purpose when selecting the BIT data type. For instance, it may seem like indicating whether an item has specific characteristics may be a good use of the BIT data type. An example could be a column in a table like IsVegetarian. However, it may be better to consider redesigning the database to record those attributes in another table. A BIT can be used to indicate a successful status for a transaction. However, there is often more status to record the state of the transaction in over a period of time. If recording status changes over time is important, then using a BIT to record if a transaction is successful may not be the best option.

An advantage of the BIT data type is the overall storage space required when saving BIT values in a database. As there are 8 bits in a byte, the same holds true for storing records in a database. For each 8 BIT columns in a table, those values are all stored a single byte. If there were 9 to 16 BIT columns in the table, all the BIT values would be stored in a total of 2 bytes. This small amount of storage space required indicates that it would take one table with up to 8 BIT columns and 1 million records to use one MB of space in the database.

## TINYINT, SMALLINT, INT, BIGINT

SQL Server also allows you to store whole numbers, that is, numbers that do not have decimals or fractional values. These numbers are known as integers. One example of data stored as integers can be quantities of a given item. There are several types of integer values that can be stored within SQL Server. The first integer type is TINYINT. The TINYINT value can contain any integer value between 0 and 255. Due to the limited size of this data type, this may be useful for limited configuration types or number of locations. This data type is like the BIT data type, but this data type has a slightly wider range. This data type would also be useful to configure the types of statuses in a system or categories of objects. The TINYINT is good for storing these status types as many applications do not need more than 256 statuses.

Now that we've covered TINYINT, let's discuss the possibilities of SMALLINT. The range of SMALLINT covers approximately 70,000 possible values. With this range available, you want to consider what sorts of values you would want to have between 256 and 65,435 unique values. The range for the SMALLINT starts at –32,768 and ends at 32,767. This data type would not be useful for a table that logs every single activity that happens. Many databases or data tables may have more than 70,000 transactions or records over the course of several years. This may cause this data type to be unsuitable for those tables. However, there may be other tables where the SMALLINT data type is ideal.

If you have data tables that do not experience high transactional activity but will be growing for some time, the SMALLINT data type may be beneficial. Understanding your business will help you determine if the SMALLINT is the correct data type for that value being stored.

If you were to create a table to continue to add functionality to your applications, you may want to store a record indicating each new piece of functionality was to your application. An example of this can be feature flags. Your application will likely have more than 256 enhancements over the life of the application. You may also want to store configuration values for the application. Storing these configuration values in a table may benefit from the SMALLINT data type.

Moving on to integers or INT. This is the most frequently use of the whole number data types. Many databases use this number exclusively for any sort of whole number that is being tracked. One of the reasons for this is the entire range covers about 5.4 billion records. The INT data type covers the range from –2,147,483,648 to 2,147,483,647. However, when many data tables are created, their identity column is often started, or seeded, at the integer 1. This causes the table to be limited to approximately 2.15 billion unique identities.

If you believe that your table will need more than 2.15 billion unique identity records, you may want to start the identity with the lowest number possible, –2,147,483,648.

This is often where you may need to perform some mathematical calculations. Some businesses process a couple hundred transactions per second. Other businesses process upward of 10,000 or 20,000 transactions per second. In both cases it's important to consider what kind of growth will be expected in the tables holding this transactional information. If your application has hundreds of transactions per second over several years, the number of records stored will be much smaller than if the application has tens of thousands of transactions per second for the same time period.

## DECIMAL/NUMERIC

Now that we have discussed various integers, we should consider what to do with numbers that require decimals. There are various cases where you are going to want decimal places. Some of these cases involve using dollars and cents, and other times you are going to need decimal places for precision in measurements. There are a couple of options available in these scenarios.

First there is the option for DECIMAL or alternatively it is called a NUMERIC data type. This value does not save any currency information with it; however, it does record decimal places. These decimal places can be specified by indicating both the total number of digits that should be stored and the number of digits to the right of the decimal point. You will find that the DECIMAL or NUMERIC data type is acceptable for almost all data types involving numbers. This includes general-purpose numbers, decimals, measurements, and money values.

Considering that the DECIMAL type can represent multiple different types of numbers, we should take a closer look at this data type. There is no difference between DECIMAL and NUMERIC. They are the same data type in SQL Server. There are some specific terms for the DECIMAL data type. The values that make up a DECIMAL data type are precision and scale. Precision relates to the total number of digits that are saved in a DECIMAL data type. Scale refers to the number of digits that are stored to the right of the decimal point.

## SMALLMONEY, MONEY

The next data types to discuss are MONEY and SMALLMONEY. The MONEY and SMALLMONEY data types are like the DECIMAL or NUMERIC data type. The SMALLMONEY and MONEY data types can also be used to store values for currency.

SQL Server will save the numeric value and exclude the type of currency associated with the value saved.

The largest difference between the MONEY and SMALLMONEY data types is size and storage space. The SMALLMONEY data type covers a range from –214,000 to positive 214,000 and takes up only 4 bytes of data, whereas the MONEY data type covers a range from –922 billion to 922 billion and takes up 8 bytes of data.

The MONEY data type is accurate to store up to four decimal places. The limitation on decimal places will limit the accuracy to ten thousandths of the monetary value stored. The MONEY data type will save all values to four decimal places. The fixed number of decimal places will impact how rounding affects calculations involving the MONEY data type.

## Approximate Number Data Types

Next we will move on to the differences between exact and approximate numbers. Exact numbers exist for things that you know the exact quantity of, for instance, how many items you bought at the store or the exact amount in dollars and cents. Whereas approximate numbers exist for scenarios where the measurements may not be exact. Approximate numbers can be used to store very large or very small numbers. You may also find that your application is recording a measurement that is not exact. You may cut a length of fabric that is close to but not exactly a specific value. For example, the length of fabric may be around 12 inches. Storing the value 12 for inches would be an approximate value. This measurement of 12 inches may be so close that it would be difficult to tell that the length of fabric was not exactly 12 inches.

There are some rounding issues that come into play when dealing with approximate numbers. This is because approximate numbers are known to not be the exact measurement. In SQL Server, there is one option for approximate numbers. This data type is called FLOAT. If the floating number has 24 numbers, the synonymous data type is REAL.

When working with REAL or FLOAT numbers, there can be issues converting this data type to other data types. Converting a FLOAT data type to an INTEGER, all the values in the decimal places will be truncated. You will want to be aware that using approximate numbers may cause unexpected results. One example is when using the DECIMAL or NUMERIC data types. When converting a FLOAT or REAL number to a DECIMAL or NUMERIC data type, you are only able to keep seven decimal places.

## Converting Number Data Types

We have covered the types of numbers available, and what happens in working with the various number data types. In addition to storing numbers, you will also want to understand how various number data types interact with one another. First we should consider what happens when we are doing calculations involving fields with the same data type. In these scenarios, if all the fields for the calculation are of the same data type, the data type will remain the same. Therefore, if you were multiplying a quantity times the price and both values are stored as a NUMERIC data type of DECIMAL(5,2), it will give the same data type of DECIMAL as a result. SQL Server will determine the result precision and scale based on the starting precision and scale as well as the type of calculation performed. You will also want to consider how the precision and scale is stored in the application. The application may be expecting a value of DECIMAL(5,2). Based on the calculation performed, the data values returned may be outside the range of the data type specified. This can cause an overflow in calculation.

Things can get a little more interesting when working across various data types. To use the example of quantity times price again, we can examine what happens if you had an INT data type that was calculated with a DECIMAL (5,2) data type. SQL Server would use the process of data type precedents. First we should get familiar with the data type order of precedence related to the data types covered so far. The following list is ordered from highest order to lowest.

1. FLOAT
2. REAL
3. DECIMAL
4. MONEY
5. SMALLMONEY
6. MONEY
7. BIGINT
8. INT
9. SMALLINT
10. TINYINT
11. BIT

In the preceding scenario, we are using both the INT and DECIMAL data types. As you can see, the INT and DECIMAL data types are listed. In the case of the INT and DECIMAL data types, the INT has a lower precedence. Due to the order of precedence, SQL Server will internally convert the INT data type to the DECIMAL data type. This conversion does not change the original data value, only how SQL Server uses this value as part of the calculation. Once this conversion is complete, SQL Server moves forward with the calculation. This works well unless you are performing an action like trying to concatenate a number and string data types.

# String Data Types

Now that you know how to work with number data types, we should spend some time on the various string data types. These sorts of data types are used to store alphabetical letters, words, or combinations or letters and numbers. In addition, string data types are used to store character values that are either non-unicode or unicode. The last category of string data types includes images and binary values.

## Character String Data Types

There is information that will be stored in the database that is not related directly to numbers. This data can be names, descriptions, addresses, or other character values. Determining which data type to use will depend on what type of information is being stored and how much information needs to be stored.

### CHAR and VARCHAR

Two of the character string data types available are CHAR and VARCHAR. These data types are similar and only vary regarding some specific considerations. The data field can be configured to determine how data can be stored. The columns can be configured to toggle case sensitivity, accent sensitivity, sensitivity for kana, or width sensitivity. The sensitivity that is stored is referred to as collation. The type of data that can be stored usually matches the database collation. The collation of a column is the same as the database unless there is a specific override in place. Both data types are used to store text data, and the characters that can be stored in these fields are the same characters allowed by the collation of the column.

When determining which data type to use, keep in mind what kinds of data will be stored. If the data will have similar lengths, like phone numbers or zip codes, then CHAR may be the preferred data type. However, if the column widths can vary significantly as is the case with address lines or notes columns, then VARCHAR will be a better option. Consider limiting the use of VARCHAR(MAX) for situations where you expect to save more than 8000 characters. If VARCHAR(MAX) is specified, then the maximum storage size is 2 GB.

There are some additional considerations to keep in mind when using CHAR and VARCHAR. When using CHAR and VARCHAR for data definition or variable declaration, remember that the default value is one character. However, when using the CAST or CONVERT functions, the default number of characters will be 30. In order to minimize truncating data, make sure to always specify the number of characters explicitly when using the CHAR or VARCHAR data types.

For collations using single-byte encoding characters, such as Latin, the storage size in bytes for CHAR is equal to the number of characters. When working with VARCHAR, the number of characters plus 2 additional bytes is equal the total number of bytes stored. It is also possible to save multi-byte encoding characters in the CHAR and VARCHAR data types. For both data types, the number of characters saved may be less than the total number of bytes.

Starting with SQL Server 2019, it is now possible to save unicode values in CHAR or VARCHAR. However, this is only possible if UTF-8 encoding is enabled.

## TEXT

The TEXT data types have been previously used when needing to store very large strings of characters. However, this data type has been deprecated. As this data type has been deprecated, you will want to avoid using the TEXT data type for new development. If you need to use the TEXT data you, you will want to consider the data type VARCHAR(MAX) for new development instead. However, consider if you need this functionality or if using VARCHAR with a smaller number of characters may be more appropriate. The only consideration for using the TEXT data type going forward should be as needed for backward compatibility of your applications. The TEXT data type is primarily used in situations with very large strings that have variable length. In many cases, this would be where the length of data saved is more than 8000 characters. The maximum number of characters that can be saved to the TEXT data type is 2,147,483,647. There may be occasions where the total number of characters that can be stored is less than this number.

# Unicode String Data Types

Prior to SQL Server 2019, any Unicode text data would need to be saved as a special data type. This is still true for situations where UTF-8 encoding cannot be enabled.

## NCHAR and NVARCHAR

When using Unicode values, there are a couple of options available. These options include storing a fixed or variable-length string. In order to avoid unexpected results, you should understand how these data types work if the number of characters or collation is not specified.

Once you have determined that you need to use the NCHAR or NVARCHAR data types, choosing between them gets easy. If the data being stored will have generally similar lengths, then the NCHAR data type is the correct choice. However, if the values stored will vary significantly, then the NVARCHAR data type may be a better choice. In addition, if the number of characters to be stored is over 4000, it is recommended to use NVARCHAR(MAX).

Typically, it is best practice to specify the number of characters when declaring the NCHAR or NVARCHAR data types. The default number of characters for data definition or variable declaration is one character for NCHAR or NVARCHAR. However, when using the CAST or CONVERT function, the default number of characters is 30 if none are specified. If a collation is not specified for the NCHAR or NVARCHAR data type, the default database collation will be used.

Understanding the amount of space required to store this data type also allows you to make better decisions about if this is the correct data type and the number of characters that need to be stored. Storing NCHAR takes up twice as many bytes as the string length of the byte pairs, while using NVARCHAR the number of bytes stored is twice the string length in byte pairs plus 2 bytes.

## NTEXT

Previously, storing very large variable-length Unicode data was accomplished using the NTEXT data type. If this data type is still in use in your systems, you can expect it to store up to 1,073,741,823 characters. However, due the size associated with Unicode values, the total length stored may be less. Going forward, it is no longer best practice to use this data type. Instead, use the NVARCHAR(MAX) data type.

# Binary String Data Types

At some point, you may want to store data that is neither a number nor a character. In these cases, the use of binary strings may be appropriate. There are a couple alternatives when using binary string data types.

## BINARY and VARBINARY

The options available for storing binary string data involved storing either fixed length or variable-length character strings. Like the other string data types discussed, there are also some considerations when dealing with these data types.

Using binary strings for storing items that are strings without characters may be useful. These can include audio, video, images, or other similar items. Two of the available data types are BINARY and VARBINARY. The best option for storing binary strings with similar lengths is the BINARY data type. Conversely, when storing binary strings with significantly varying lengths of data, the VARBINARY data type is a better choice. If the total length of the binary string is expected to exceed 4000 characters, then it is suggested to use VARBINARY(MAX).

Using the BINARY and VARBINARY data types for data definition or variable declaration will have a default length of one if the number of characters is not specified. When converting the BINARY to VARBINARY with the CAST or CONVERT function, the default number of characters will be 30. Use caution when converting to BINARY or VARBINARY from a variable with a different length as SQL Server may pad or truncate the binary data as necessary.

The BINARY data types stores the same number of bytes as the length of data being stored, whereas VARBINARY uses 2 bytes plus the same number of bytes as the length of data being stored. In both data types, the length can be up to 8000. For VARBINARY(MAX), the maximum storage size is 2 GB.

## IMAGE

One of the items that can be stored in a binary string is image. When working with images, it is important to consider how this data should be stored and if so what data type should be used.

This IMAGE data type has been used to store large variable-length binary data. While this data type can have a length of 2,147,483,647, there are times where the allowable length stored may be less. In the case of the IMAGE data type, you should use the VARBINARY(MAX) data type going forward as the IMAGE data type is deprecated.

# Date and Time Data Types

Each database transaction occurs at a specific point in time. There may be a need to reference or know when a transaction happened. Your application may record important dates for a person including birthdays or anniversaries. Dates and times can also be used to determine pricing and functionality. By using dates and times, you can determine when functionality should be enabled or disabled. Dates and times can also show when a user account is inactive or access to a given system is enabled or expired. Pricing and billing rates can cover multiple different date ranges. When one set of pricing becomes inactive, another set may be active. Due to regulations, your company may need to record the pricing over a period of time. This includes indicating when the pricing rates may have started and stopped. Depending on the purpose for tracking this information, you may only need to know only the date or time of the transaction. There are other situations where it is best to know both the date and time associated with a certain action.

## DATE

When working with transactions, there may be a specific occasion where you want to record when something happened. In some cases, it may only matter on what day in which the transaction happened. The DATE data type can also be used to store aggregated data for a given day. While recording the date of the activity, there may also be some options available as to how that data is displayed. When choosing if the DATE data type is right for you, it is also important to consider not only how much data is stored for the DATE data type but any possible limitations in how the data can be stored.

There will be times when an application or a user needs to know when a specific action happened. When deciding if a DATE data type is the right choice, you will want to consider the need for the information both in terms of user and application usage. In some cases, it is easier to think about when a DATE data type would not be preferable.

For any action where you would want to know a specific time when something happened, the DATE data type would not be a good choice. However, if it is only necessary to know on what day an action occurred, then the DATE data type would be a great option.

For the DATE data types, there are several options as to how a DATE can be displayed. With the date format, the default is YYYY-MM-DD. In this case YYYY represents the four-digit year with the range of 0001 to 9999. MM represents the month number from 01 to 12, and DD stands for the day ranging from 01 to 31, per the number of days in a month. The DATE can be displayed in a variety of numeric and alphabetic formats. However, the format ydm is not supported.

The DATE values that can be stored range from 0001-01-01 to 9999-12-31. The DATE data type has a ten-digit character length with a precision of 10 and a scale of 0. The DATE data type takes up 3 bytes and is stored as an INT.

Dates can be converted to DATETIME, SMALLDATETIME, DATETIME2, or DATETIMEOFFSET. However, the time value will be set to midnight. However, dates cannot be converted to the TIME data type, and any attempts to perform this conversion will fail with an error. In addition, dates do not have a time zone offset and are not daylight saving time aware.

## TIME

Another data type related to when an action happened is the TIME data type. It is useful to understand how time is stored and formatted. When using the TIME data type, it is helpful to know the implications of converting the data type to other DATE and DATETIME data types. There are also some limitations when using the TIME data type.

TIME can be used to record a specific time when a transaction or activity occurred. When this happens, the time is recorded independently of the date and the date may not be able to be determined in the future. One way around this issue could be to store the date separately from time. The accuracy of TIME is up to 100 nanoseconds, and the default value for TIME is 00:00:00.

The default format for TIME is hh:mm:ss[.nnnnnnn]. In this format, hh stands for a two-digit hour ranging from 0 to 23, mm is for a two-digit minute ranging from 0 to 59, and ss is for a two-digit second from 0 to 59. The TIME data type allows for varying precision, and if specified, up to seven decimal places can be used for fractional seconds as represented by nnnnnnn. These values can range from 0 to 9999999.

Due to how AM and PM are used to differentiate between morning and evening, there are additional considerations when working with TIME. If AM or PM is not provided and the value for hour is between 00 and 11, the time will be recorded as AM. For hours 12 to 23, the time will be saved as PM. When writing TIME, if 12 AM is entered, this value will be converted to the 0 hour.

The range of TIME is 00:00:00.0000000 to 23:59:59.9999999. The character length can vary from 8 to 16 digits, depending on the precision specified for TIME. In either scenario, TIME will be saved as fixed 5 bytes. If TIME is converted to any data type with a date and time, the day value will be represented as 1900-01-01. If the fractional precision is higher for TIME than the new data type, the value will be truncated. Any attempt to convert the TIME data type to a DATE will fail. Like DATE, TIME is neither time zone nor daylight saving time aware.

## SMALLDATETIME, DATETIME, DATETIME2, DATETIMEOFFSET

There are occasions where saving the date or time may not be enough. For these scenarios, it may be best to combine the date and time values together. Sometimes these values can be somewhat simpler, need more precision, or need to be time zone aware.

One such data type is the SMALLDATETIME. This data type is used to record both a specific date and time. It has a default value of 1900-01-01 00:00:00. While the accuracy of this data type is listed as 1 second, it is important to note that the seconds will always be saved as 00 in the database.

As with the date data type, the SMALLDATETIME data type can be displayed in a variety of numeric and alphabetical formats. The range for the SMALLDATETIME is somewhat limited as compared to other DATE and DATETIME data types. The day portion of this data type can span 1900-01-01 to 2079-06-06. While the time entered can range from 00:00:00 to 23:59:59, the value saved in the database will be 00:00:00 to 23:59:00. The overall length of the SMALLDATETIME is up to 19 characters, and the storage size required is a total of 4 fixed bytes.

When converting SMALLDATETIME to other DATETIME data types, keep in mind that any additional precision needed will be recorded with 0s. While it may be tempting to use the SMALLDATETIME, this data type is not ANSI compliant. As stated previously, the seconds for this data type will be rounded depending on the value passed for the

seconds. If the seconds passed are less than or equal to 29.998, the minute will be rounded down. Otherwise, the minute will be rounded up. Like the date and time data types, SMALLDATETIME is also not time zone or daylight saving time aware.

There are more options available than just SMALLDATETIME. DATETIME has been an option for a higher level of precision than the previously mentioned data types. There are also several key considerations when using this data type.

While the DATETIME data type can record a specific day and time, it may not comply with the SQL Standard. One of the key issues with this data type has to do with the limitations related to accuracy. The DATETIME data type can record three decimal places for fractional seconds; the third decimal place is always rounded to an increment ending in .000, .003, or .007.

If a value is not specified, the default for DATETIME will be 1900-01-01 00:00:00. There are many numeric and alphabetical formats available when using this data type. The year range for DATETIME is 1753-01-01 to 2999-12-31, and the time can range from 00:00:00.000 to 23:59:59.997. The size of this data type is 8 bytes with a character length ranging from 19 to 26.

While it is possible to convert other data types to DATETIME, it is not recommended as this data type does not meet SQL Standards and is not ANSI compliant. The DATETIME data type is also limited due to the rounding that occurs allowing only increments of .000, .003, and .007. This data type is also not time zone or daylight saving time aware.

The DATETIME2 data type has some additional advantages over the data types previously mentioned. While some of the previously mentioned data types have a fixed size, this data type works a little differently. We will also look at storing and formatting available for this data type.

The DATETIME2 data type allows for a specific date and time to be recorded with an accuracy of up to 100 nanoseconds. The default value for DATETIME2 is 1900-01-01 00:00:00. Due to this level of precision, this is a great data type to use for scenarios where the time must be known to a fraction of a second. As DATETIME2 doesn't have the same rounding issues as DATETIME, it is also more straightforward to work with this data type when writing code.

The DATETIME2 data type supports multiple numeric and alphabetical ways to display the information. The date range for DATETIME is from 1753-01-01 to 2999-12-31, and the time range is from 00:00:00 to 23:59:59.9999999. Multiple precision options are allowed, thus causing the character length to range from 19 for the precision to the second all the way up to 27 for the precision to 0.0000001 nanoseconds.

The variation in the precision also affects the storage size of the DATETIME2 data type. One byte is used to store the precision of DATETIME2 plus the number of bytes needed depending on the precision of time. If the precision is less than three decimal places for nanoseconds, then there are another 6 bytes used to store the DATETIME2 value. If the precision is 3 or 4, there is 1 byte to store the precision and 7 bytes to store the value, for a total of 8 bytes. However, the total will be 9 bytes for any values with a precision of more than four decimal places.

Due to the high level of accuracy, the probability of converting values to DATETIME2 is highly likely. If a date is converted to DATETIME2, the time component will be recorded as 00:00:00. If time is converted to DATETIME2, the day will be 1900-01-01. In the case of SMALLDATETIME to DATETIME2, the date and time will be copied. Any additional precision will be represented with 0s. Going from DATETIMEOFFSET to DATETIME2 will cause the time zone to be truncated. When going from DATETIME to DATETIME2, make sure to use explicit conversions to avoid unexpected results. The main limitation of using DATETIME2 is that the data type is not time zone aware or daylight saving aware.

The final data type for dates and times is DATETIMEOFFSET. When discussing the DATETIMEOFFSET, there is some additional functionality that has not been seen before with the other data types. There are also things to keep in mind when formatting, storing, or converting to this data type.

The DATETIMEOFFSET data type records the specific date and time, with a high level of accuracy, for transactions or actions that have taken place. One of the key advantages to this data type is the ability to have an offset on the time, thus allowing databases from multiple geographic locations to not only be aware of when something happened in relation to their local time but also in relation to local time at another location.

The DATETIMEOFFSET is accurate to 100 nanoseconds and has a default value of 1900-01-01 00:00:00. The format of DATETIMEOFFSET is YYYY-MM-DD hh:mm:ss. nnnnnnn +|– hh:mm. The +|– hh:mm portion of this data type is related to the offset. The offset can range from +14 to –14 for the number of hours that a given time can have an offset. As with the other time and DATETIME data types, this date can be formatted or displayed numerically or alphabetically.

The dates can range from 0001-01-01 to 2999-12-31. The time that can be saved ranges from 00:00:00 to 23:59:59.9999999. When the precision is saved as YYYY-MM-DD hh:mm:ss {+|–} hh:mm, the character length is 26. The character length can go up to 34 when the precision is YYYY-MM-DD hh:mm:ss.0000000 {+|–} hh:mm. The storage space required for the DATETIMEOFFSET data type is a fixed 10 bytes.

# Other Data Types

In addition to the data types discussed previously, SQL Server has several other data types that are available. Some of these data types can be used in table definition and may have special purposes, while some of the other data types may only be usable as variables or inside stored procedures.

## UNIQUEIDENTIFIER

This data type can be a column in a table or used as a variable. The UNIQUEIDENTIFIER takes up 16 bytes and has a maximum number of characters that can be stored in this data type is 36. While non-unicode character strings can be converted to UNIQUEIDENTIFIER, if the total number of characters exceeds 36, those results will be truncated.

This data type is a GUID, or Globally Unique Identifier. The concept is that these unique values will only ever be used once. However, there have been reports of this not being true. Either way, the UNIQUEIDENTIFIER can be populated one of several ways. These include using the functions NEWID() and NEWSEQUENTIALID(). Otherwise, these values can be manually populated if the overall format of the GUID is correct and uses valid hexadecimal values of 0–9 and a–f.

While the UNIQUEIDENTIFIER can be used in place of IDENTITY, I would only recommend it for scenarios where it is absolutely required. Not only does it take up significantly more space than an INT or BIGINT, but UNIQUEIDENTIFIER is limited in the types of constraints that can be used with this data type. UNIQUEIDENTIFER can be an IDENTITY but other table constraints are not allowed.

## XML

Various systems and applications send, use, or store XML data. While there is the option to parse this data and save it in tables, there are also times where it may be necessary to store the XML data intact. When storing XML data, there are other considerations that include what data is in the XML.

For the XML data type, the data must be in a valid XML format. In order to be valid, there are several requirements. These include all starting tags must have matching end tags. In addition, nested elements must begin and end within the same parent element. XML elements cannot have more than one attribute and markup characters must be