LESSON 02: Creating Database Objects

Defining Data Types

THE BOTTOM LINE
In this section, you will learn what data types are, why they are important, and how they affect storage requirements. When looking at data types, you need to understand what each type is designed to do within a table, as well as how certain types work best for each column, local variable, expression, or parameter. Also, when choosing a data type to fit your requirements, you need to ensure that whatever type you choose provides the most efficient storage and querying schema. In fact, one of the key roles of a database administrator is to ensure that the data within each database is kept uniform by deciding which data type is best suited to the application module currently being worked on.

A ***data type*** is an attribute that specifies the type of data an object can hold, as well as how many bytes each data type takes up. For example, several data types handle only whole numbers, which makes them good for counting or for identification. Other data types allow decimal numbers and therefore come in handy when storing values dealing with money. Still other data types are designed to store strings or multiple characters so that you can define labels, descriptions, and comments. Last are other miscellaneous data types that can store dates, times, binary numbers consisting of 0s and 1s, and pictures. As a general rule, if you have two data types that are similar and differ only in how many bytes each uses, one of the data types will have a larger range of values and/or offer increased precision.

Using Built-in Data Types

Microsoft SQL Server includes a wide range of predefined data types called built-in data types. Most of the databases you will create or use employ only these types of data.

Microsoft SQL Server 2008's built-in data types are organized into the following general categories:

- Exact numbers
- Approximate numbers
- Date and time
- Character strings

- Unicode character strings
- Binary strings
- Other data types
- CLR data types
- Spatial data types

You'll use some of these built-in data types on a regular basis and others more sporadically. Either way, it's important to understand what these data types are and how they are utilized inside a database. Tables 2-1 and 2-2 show the most commonly used data types. Note that in Table 2-2, the asterisk (*) denotes the newest data-type additions in SQL Server 2008.

CERTIFICATION READY
What data type would you use for the cost of an automobile? What type would you use to count the number of cars you have in stock?
2.1

Table 2-1 Most commonly used data types

DATA TYPE	EXPLANATION
Money (numeric)	This numeric data type is used in places where you want money or currency involved in your database; however, if you need to calculate any percentage columns, it is best to use the "float" data type instead. Essentially, the difference between a numeric data type and a float data type rests in whether you are using the data type for approximate numbers or fixed precision. A money or numeric data type is a fixed-precision data type because it must be represented with precision and scale.
Datetime	The datetime data type is used to store date and time data in many different formats. Two main subtypes of this data type—datetime and datetime2—are available, and you should consider what you are using the stored data for when deciding which subtype to use. In particular, if you will be storing values
	between the dates of January 1, 1753, and December 31, 9999, that are accurate to 3.33 milliseconds, you should use the datetime data type. In contrast, if you will be storing values between January 1, 1900, and June 6, 2079, that are accurate to only 1 minute, then datetime2 is the data type to use. The second important difference between the two data types is that the datetime data type uses 8 bytes of storage, whereas datetime2 only requires 4 bytes.
Integer	The integer (int) numeric data type is used to store mathematical computations and is employed when you do not require a decimal point output. Examples of integers include the numbers 2 and 2.

DATA TYPE	EXPLANATION
Varchar	This character-string data type is commonly used in databases where you are supporting English attributes. If you are supporting multiple languages, use the nvarchar data type instead, as this will help minimize issues of character conversion.
Boolean	The Boolean data type is also known as the bit data type. Here, if your columns store 8 bits or fewer, the columns will be stored as 1 byte; if, they contain 9 to 16 bits, the columns will be stored as 2 bytes; and so forth. The Boolean data type converts true and false string values to bit values, with true converted to 1 and false converted to 0.
Float	The float numeric data type is commonly used in the scientific community and considered an approximate-number data type. This means that not all values within the data-type range will be represented exactly. In addition, depending on which type of float is used, a 4-byte float supports precision up to 7 digits and an 8-byte float supports precision up to 15 digits.

Table 2-2 Data types

DATA TYPE	USE/DESCRIPTION	STORAGE
Exact Numerics:		
`bit`	Integer with either a 1 or 0 value. (Columns of 9 to 16 bits are stored as 2 bytes, and storage size continues to increase as the number of bits in a column increases.)	1 byte
`tinyint`	Integer data from 0 to 255.	1 byte
`smallint`	Integer data from –2^15 (–32,768) to 2^15–1 (32,767).	2 bytes
`int`	Integer data from –2^31(–2,147,483,648) to 2^31–1 (2,147,483,647).	4 bytes
`bigint`	Integer data from –2^63 (–9,223,372,036,854,775,808) to 2^63–1 (9,223,372,036,854,775,807).	8 bytes
`numeric`	Fixed precision and scale. Valid values range from –10^38+1 through 10^38–1.	Varies
`decimal`	Fixed precision and scale. Valid values range from –10^38+1 through 10^38–1.	Varies

DATA TYPE	USE/DESCRIPTION	STORAGE
`smallmoney`	Monetary or currency values from –214,748.3648 to 214,748.3647.	4 bytes
`money`	Monetary or currency values from –922,337,203,685,477.508 to 922,337, 203,685,477.5807.	8 bytes
Approximate Numerics:		
datetime	Defines a date that is combined with a time of day with fractional seconds based on a 24-hour clock. Range: January 1, 1753, through December 31, 9999. Accuracy: Rounded to increments of .000, .003, or .007 seconds.	8 bytes
smalldatetime	Defines a date that is combined with a time of day. The time is based on a 24-hour day, with seconds always zero (:00), meaning there are no fractional seconds. Range: 1900-01-01 through 2079-06-06 (January 1, 1900, through June 6, 2079). Accuracy: one minute.	4 bytes
date*	Defines a date. Range: 0001-01-01 through 9999-12-31. (January 1, 1 AD, through December 31, 9999). Accuracy: one day.	3 bytes
time*	Defines a time of day. This time is without time-zone awareness and is based on a 24-hour clock. Range: 00:00:00.0000000 through 23:59:59.9999999. Accuracy: 100 nanoseconds.	5 bytes
datetimeoffset*	Defines a date that is combined with a time of day that has time-zone awareness and is based on a 24-hour clock. Range: 0001-01-01 through 9999-12-31 (January 1, 1 AD, through December 31, 9999). Range: 00:00:00 through 23:59:59.9999999. Accuracy: 100 nanoseconds.	10 bytes
datetime2*	Defines a date that is combined with a time of day that is based on a 24-hour clock. Range: 0001-01-01 through 999-12-31 (January 1, 1 AD, through December 31, 9999). Range: 00:00:00 through 23:59:59.9999999. Accuracy: 100 nanoseconds.	Varies
Character Strings:		
char	Character data type with fixed length.	Varies
varchar	Character data type with variable length.	Varies
text	This data type will be removed in future SQL releases; therefore, use varchar(max) instead.	Varies

DATA TYPE	USE/DESCRIPTION	STORAGE
Unicode Character Strings:		
nchar	Character data type with fixed length.	Varies
nvarchar	Character data type with variable length.	Varies
ntext	This data type will be removed in future SQL releases; therefore, use nvarchar(max) instead.	Varies
Binary Strings:		
binary	Binary data with fixed length.	Varies
varbinary	Binary data with variable length.	Varies
image	This data type will be removed in future SQL releases; therefore, use varbinary(max) instead.	Varies
Other Data Types:		
sql_variant	Stores values of various SQL Server-supported data types, except text, ntext, image, timestamp, and sql_variant.	Varies
uniqueidentifier (UUID)	16-byte GUID.	16 bytes

Remember that in SQL Server, each column, local variable, expression, and parameter always has a related data type that defines the storage characteristics of the data being stored. This is shown in Table 2-1.
Now that you have some understanding of the many data types available in Microsoft SQL Server, keep in mind that when two expressions have different data types, collation, precision, scale, or length, the characteristics of the results will be determined as follows:

- When two expressions (mathematical functions or comparison functions) have different data types, rules for data-type precedence specify that the data type with lower precedence is converted to the data type with higher precedence.

TAKE NOTE*
Each column, local variable, expression, and parameter always has a related data type, and each of the data types is an attribute.

- Collation refers to a set of rules that determine how data is sorted and compared. By default, SQL Server has predefined collation precedence. If you wish to override how data is being sorted, you must use a collation clause.
- The precision, scale, and length of the result depend on the precision of the same in the input expression. In other words, if you take several different values and perform a mathematical operation on those values, the precision, scale, and length will be based on those values on which you are performing the mathematical operations.

Now, let's go through some of the most common built-in data types in greater detail so that you are more familiar with how to use them.

Using Exact Numeric Data Types

Exact numeric data types are the most common SQL Server data types used to store numeric information. Some of these data types allow only whole numbers, whereas others allow decimal numbers.

Exact numerics include (but are not limited to) `int`, `bigint`,`bit`, `decimal`,`numeric`,

`money`, and `smallmoney`:

- `int` is the primary integer (whole number) data type.
- `bigint` is intended for use when integer values will exceed the `int` data type's range of support. Functions return `bigint` only if the original expression is a `bigint` data type. Note that SQL Server will not automatically promote other integer data types
  (i.e., `tinyint`, `smallint`, and `int`) to `bigint`.
- `bit` is a Transact-SQL integer data type that takes a value of 1, 0, or NULL and produces the following characteristics:
  - SQL Server Database Engine will optimize the storage of bit columns, meaning that if your table has columns that are 8 or fewer bits wide, these columns will be stored as 1 byte, and if it has 9- to 16-bit columns, they will be stored as 2 bytes. It is important to realize that 1 byte equals 8 bits when considering data types.
  - TRUE and FALSE string values can be converted to bit values. Specifically, TRUE is converted to 1 and FALSE is converted to `0`.

- `decimal` and `numeric` are also Transact-SQL data types that have a fixed precision and scale. The syntax for these data types is expressed as follows:

```
decimal[(p[,s])]

numeric[(p[,s])]
```

  - `Precision (p)` is the maximum total number of decimal digits that can be stored, both to the left and the right of the decimal point. This value must be a minimum of 1 and a maximum of 38. The default precision number is 18.
  - `Scale(s)` reflects the maximum number of decimal digits that can be stored to the right of the decimal point. This must be a value from 0 through `p`, but it can be specified only if precision is also specified. The default scale is 0.
- `money` and `smallmoney` are Transact-SQL data types you would use to represent monetary or currency values. Both data types are accurate to 1/10,000th of the monetary units they represent.

Using Approximate Numeric Data Types

Approximate numeric data types are not used as often as other SQL Server data types. However, if you need more precision (more decimal places) than is available with the exact numeric data types, you can use either `float` or `real`, although you should be aware that these data types typically require additional bytes of storage.

`float` and `real` are used in conjunction with floating-point numeric data. This means that all floating data is approximate; thus, not all values that are represented by an approximate data-type range can be expressed accurately.

The syntax of `real` is `float(n)`. `n` is the number of bits used to store the mantissa of the `float` number as represented in scientific notation; therefore, the precision and storage size are dictated if `n` is actually specified. The value of `n` must be between 1 and 53, with the default value being 53. The mantissa is the whole number and decimal part of a value but not including place holders and exponents. For example, if you have 3.42732, 3.42732 is the mantissa. But if you have 3.23×105, the value is equivalent to 323,000, the mantissa is 3.23.

USING DATE AND TIME DATA TYPES

The date and time data types, of course, deal with dates and times. These data types include

`date`, `datetime`, `datetime2`, `datetimeoffset`, `smalldatetime`, and `time`.

`date` is used to define a date starting with January 1, 1 AD, and ranging to December 31, 9999 AD. Like any data type, the `date` data type has the descriptors shown in Table 2-3. Although dates themselves are not affected by daylight saving time, you may use dates to determine whether the time on a certain day reflects daylight saving time.

While some of the information in Table 2-3 is self-explanatory, some of it is not. For example, the default string literal format means that by default, it will store the date with the year, followed by the month (two digits), and the day (two digits). It can store any day from January 1, 1 AD to December 31, 9999. The character length means that to display the date, it would take 10 characters such as 2012-03-17. The precision scale shows that it 10 whole numbers with no decimal numbers allowed. To store the date field takes 3 bytes of data. In addition, it is only accurate to one day. So you cannot use decimal number or fractions when dealing with the date value. The default value is 1900-01-01, which means if nothing is defined, it will automatically be assigned January 1, 1900. It uses the Gregorian calendar. Last, it does not use daylight savings time.

PROPERTY	VALUE
Syntax	Date
Usage	DECLARE @MyDate date
	CREATE TABLE Table1 (Column1 date)
Default string literal format (used for down-level client)	YYYY-MM-DD (This can be utilized for backward compatibility with down-level clients)
Range	0001-01-01 through 9999-12-31 January 1, 1 AD, through December 31, 9999 AD
Element ranges	YYYY is four digits from 0001 to 9999 to represent a year MM is two digits from 01 to 12 to represent a month in

PROPERTY	VALUE
	a specified year DD is two digits from 01 to 31, depending on the month, that represent a day of the specified month
Character length	10 positions
Precision, scale	10, 0
Storage size	3 bytes, fixed
Accuracy	One day
Default value	1900-01-01 This value is used for the attached date part for inherent conver sion from `time` to `datetime2` or `datetimeoffset`
Calendar	Gregorian
User-defined fractional second precision	No
Time-zone offset aware and preservation	No
Daylight-saving aware	No

In comparison, `datetime` defines a date that is combined with a time of day expressed with fractional seconds and based on a 24-hour clock. This data type is accurate to 0.00333 seconds. If you need more accuracy, you should use the `datetime2` data type, which that is accurate up to 100 nanoseconds. If, however, you don't need to keep track of seconds (which, of course, is less accurate), you can save some storage space by employing the `smalldatetime` data type instead.

The `DateTimeOffset` data type is similar to the `DateTime` data type, but it also keeps track of time zones. For example, if you use two `DateTimeOffset` values with the same Coordinated Universal Time UTC (which is Greenwich Mean Time in most cases) time in different time zones, the two values will be the same.

If you want to create a data set in which the time of day has time-zone awareness and is based on a 24-hour clock, you will need to use `datetimeoffset` .

`smalldatetime` combines a date with a time of day, with the time based on a 24-hour day and with seconds always showing zero as (:00)—meaning that fractional seconds are not provided.

Finally, `time` defines the time of day based on a 24-hour clock and is without time-zone awareness.

TAKE NOTE*
Use the `time`, `date`, `datetime2`, and `dateoffset` data types for new work because they align with the SQL standard and are more portable. All but `date` will provide the most precision for nanosecond applications.

UNDERSTANDING IMPLICIT CONVERSIONS

When working with SQL data, you may wish to convert values from one data type to another. In most situations, these conversions are done automatically. When a conversion is done automatically, it is called an implicit conversion. For example, if you multiply an item's cost (represented as a float) with the number of items (represented as an integer), the answer will be expressed as a float. Figure 2-1 courtesy of Microsoft, provides an in-depth analysis of implicit conversion between data types.

However, some implicit conversions are not allowed. For example, although a DateTime value is represented as a float, you may not implicitly convert DateTime to a float because it is meant to be a date and/or time. If you have a reason to force a conversion, you can use the Cast and Convert functions.
Cast and Convert offer similar functionality. However, Cast is compliance with ANSI standards, which allow you to import or export to other database management systems. Convert is specific to T-SQL, but is a little bit more powerful.

The syntax of the cast function is:

```
cast(source-value AS destination-type)
```

Therefore, to convert the count variable to a float, you would use the following command:

```
cast(count AS float)
```

The syntax of the convert function is:

```
CONVERT ( data_type [ ( length ) ], expression [,style ] )
```

where you can specify how many digits or characters the value will be. For example:

```
CONVERT(nvarchar(10), OrderDate, 101)
```

This will convert the OrderDate, which is a DateTime data type to nvarchar value.

The 101 style represents USA date with century. mm/dd/yyyy.

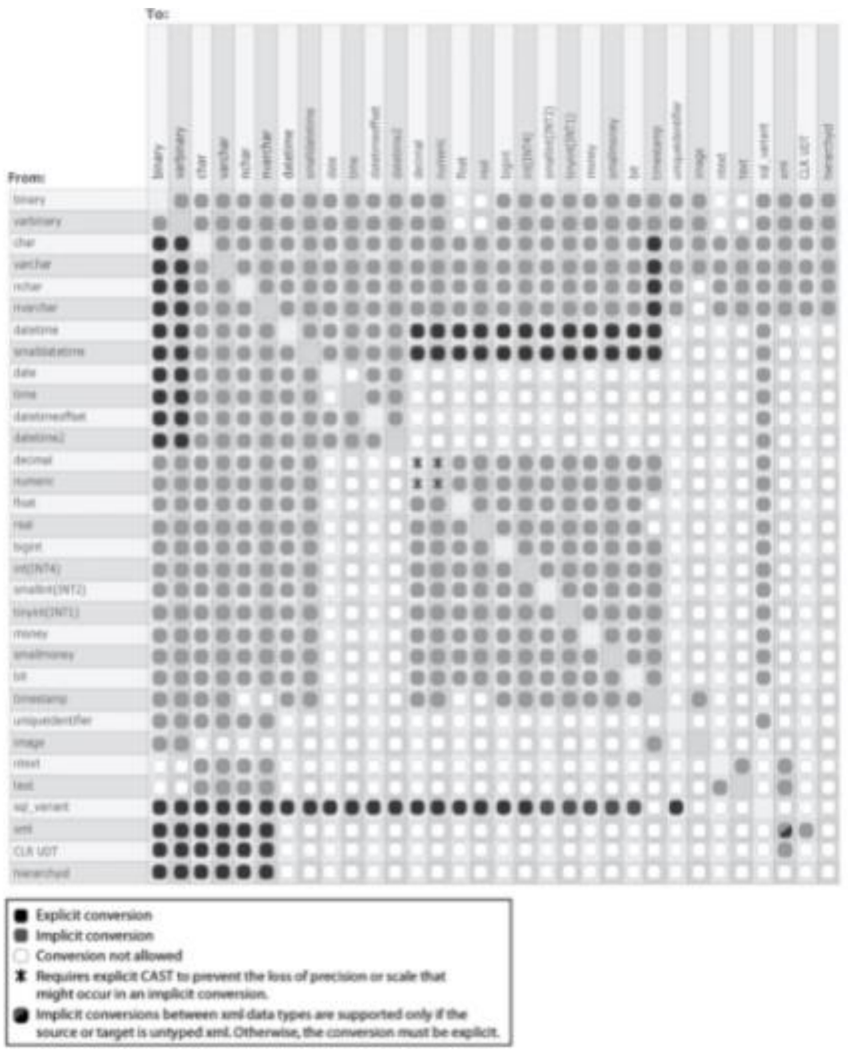

To:
From:
Explicit conversion
Implicit conversion
Conversion not allowed
Requires explicit CAST to prevent the loss of precision or scale that might occur in an implicit conversion.
Implicit conversions between xml data types are supported only if the source or target is untyped xml. Otherwise, the conversion must be explicit.

Figure 2-1 Implicit and explicit conversion types

USING CHARACTER STRINGS

A regular character uses one byte of storage for each character, which allows you to define one of 256 (8 bits are in a byte, and 2^8 = 256) possible characters, accommodating English and some European languages. A Unicode character uses two bytes of storage per character so that you can represent one of 65,536 (16 bits are in 2 bytes, and 2^16 = 65,536) characters. The additional space allows Unicode to store characters from just about any language, including Chinese, Japanese, Arabic, and so on.

As you write the syntax for the different data types, remember that they also differ in the way literals (fixed data value) are expressed. A regular character literal is always expressed with single quotes. For example:

```
'This is how a regular character string literal looks'
```

However, when you are expressing a Unicode character literal, it must have the letter N (for National) prefixing the single quote. For example:

TAKE NOTE*
Any data type without the VAR element (`char, nchar`) within its name is fixed length.

```
N'This is how a Unicode character string literal looks'
```

When you use a `VAR` element, SQL Server will preserve space in the row in which that element resides based on the column's defined size (and not on the actual number of characters in the character string itself), plus an extra two bytes of data for offset data. For example, if you want to specify that a string supports a maximum of only 25 characters, you would use

`VARCHAR(25).`

Storage consumption, when using Unicode data types, is reduced from that of the regular data type, thus allowing faster read operations; however, the price you pay for using this data type is in the possibility of row expansion, leading to data movement outside the current page. This means that any update of data using variable-length data types may be less efficient than updates using fixed-length data types. It is possible to define the variable-length data type with the `MAX` specifier, however, instead of using the maximum number of characters identified in the string. For example, when a column is defined with the `MAX` specifier, a value with a size identified up to a certain threshold (the default is 8,000) is stored inline in the row. Then, should you specify a value with a size greater than the default threshold, that value will be stored external to the row and identified as a large object, or LOB.

These are the most widely used character data types and are either of a fixed or variable length. Each has its own individual characteristics that you need to take into consideration when deciding which will have a positive effect on storage requirements. Both `char` and `varchar` data sets need to be defined, or assigned, within the data definition, or they may affect the maximum storage limits.

TAKE NOTE*
When `n` is not specified within a data definition or variable declaration statement, the default length is 1. When `n` is not specified within the `CAST` function, the default length is 30.

The data set `char` is identified as `char [(n)]` and is a fixed-length, non-Unicode character (in other words, regular character) with a length of `n` bytes. The value of `n` must be between 1 and 8,000, making the storage size `n` bytes. The other non-Unicode data type, `varchar[(n|max)]` , is a variable-length data set that can consist of 1 to 8,000 characters.

Microsoft SQL Server supports only two character string types: regular and Unicode. Regular data types include those identified with CHAR and VARCHAR . Unicode data types are identified with NCHAR and NVARCHAR . Simple? Yes, in the sense that the differences between regular and Unicode are the bytes of storage used for each.

TAKE NOTE*
Use `nchar` when the sizes of the column data entries will be similar.
Use `nvarchar` when the sizes of the column data entries will vary considerably, for such things as binary files, image files, SQL variant, and UUID.

The Unicode character strings `nchar` and `nvarchar` can either be fixed or variable, like the regular character strings; however, these strings use the UNICODE UCS-2 character set.

Creating and Using Tables

THE BOTTOM LINE

In this section, you will develop an understanding of the purpose of tables. You'll also explore how to create tables in a database using proper ANSI SQL syntax.

The purpose of a ***table*** is to provide a structure for storing data within a relational database. Without this structure, there is an increased probability of database failure. In Lesson 1, you learned about the purposes of tables and how to create them. Let's quickly review some of the most important points to remember when creating a table in a nongraphical user interface. As we do so, be sure to think about the purpose of a relational database in the hierarchy of database administration.

A SQL database is the central container that retrieves data from many different tables and views. You can run queries on these data, thereby interacting with the information stored in the database to obtain the output you require. One advantage of a database over a series of spreadsheets is that a database can parse out redundant storage and information obtained from various relational spreadsheets.

As in programming, when you are designing, creating, and using databases, you can easily use hundreds of objects, including databases, tables, columns, views, and stored procedures. Therefore, to make your company's database easier to manage, your organization should establish and use a single, consistent standard. Of course, this also means documenting this standard and distributing it to everyone who works with the database.

It really doesn't make any difference how you use uppercase and lowercase in a database, as long as you are consistent. Two common naming conventions are PascalCase and camelCase. Examples of PascalCase are such names as OrderDetails or CustomerAddresses, whereas examples of camelCase are names like myAddress and vendorTerms. No matter which standard you use, you should always be sure to use names that are both accurate and descriptive. You should also avoid using spaces because they add complications that make it necessary for you to use quotes. Instead, use underscores (_) as word separators or use mixed upper and lower case characters.

Let's first learn how to create a new table using SQL Server Management Studio (SSMS) before we move on to the syntax method of table creation.

CREATE A TABLE USING SSMS

GET READY. Before you begin, be sure to launch SQL Server Management Studio. Make sure you've expanded the particular database in which you wish to create the new table, then follow these steps:

1. Right-click the **Table** folder and select **New Table**, as shown in Figure 2-2:

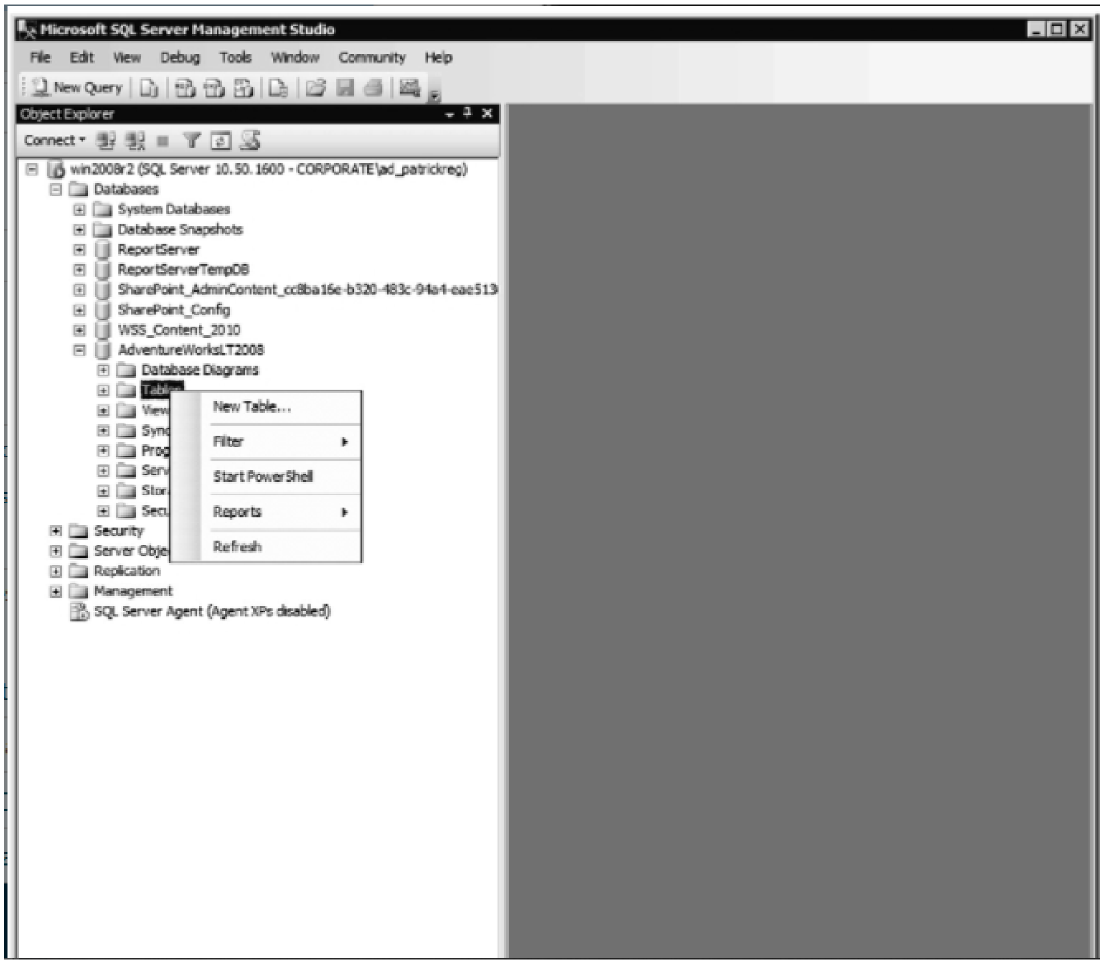
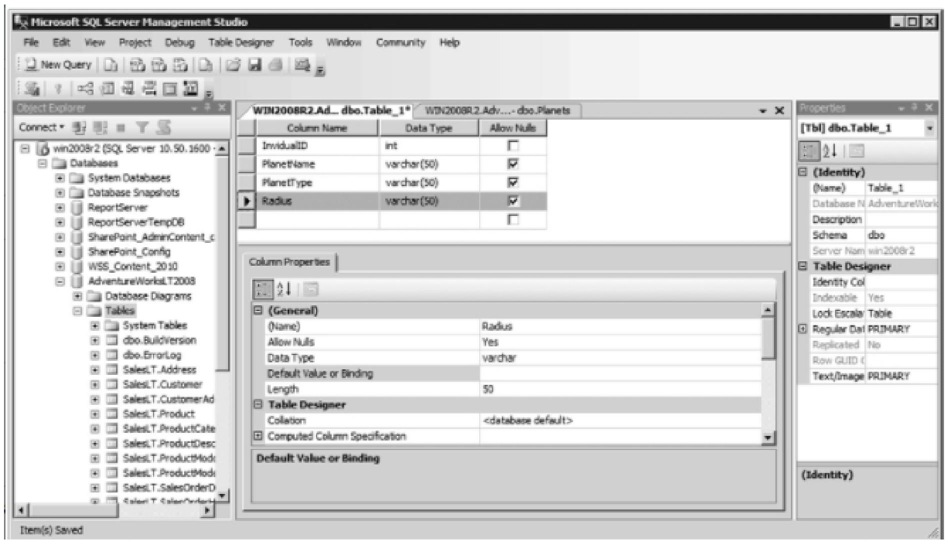


Figure 2-2 Creating a new table

2. Use the information shown in Figure 2-3 to complete the details for Column Name, Data Type, and Length, as specified in the parentheses and Allow Nulls columns.


Microsoft SQL Server Management Studio
File Edit View Project Debug Table Designer Tools Window Community Help
New Query
Object Explorer
Connect
WIN2008R2.Ad... dbo.Table_1*
WIN2008R2.Adv....- dbo.Planets
Properties
Column Properties
Item(s) Saved

Column Name	Data Type	Allow Nulls
InvidualID	int	☐
PlanetName	varchar(50)	☑
PlanetType	varchar(50)	☑
Radius	varchar(50)	☑
		☐

(General)	
(Name)	Radius
Allow Nulls	Yes
Data Type	varchar
Default Value or Binding	
Length	50
Table Designer	
Collation	<database default>
Computed Column Specification	

Default Value or Binding

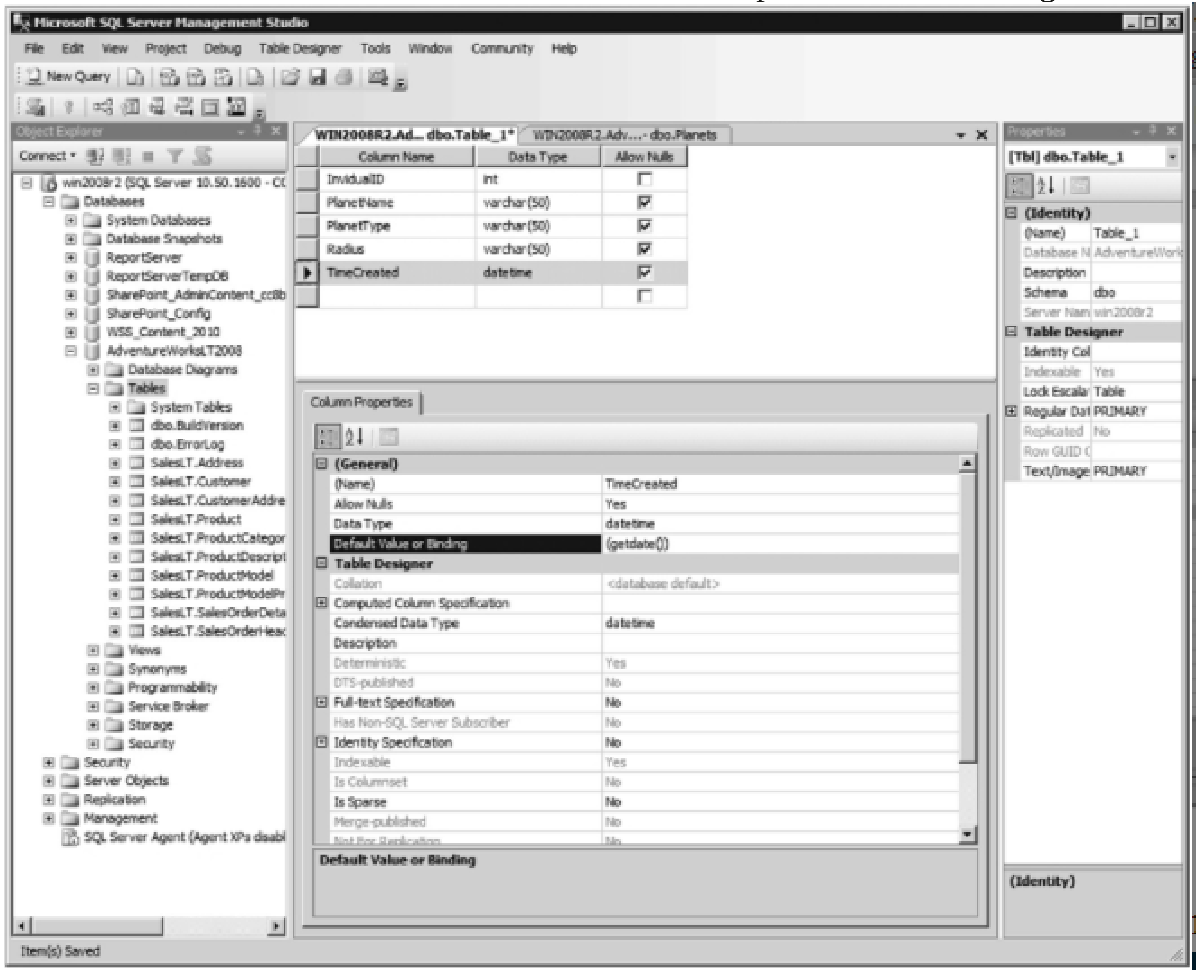
[Tbl] dbo.Table_1

(Identity)	
(Name)	Table_1
Database N	AdventureWork
Description	
Schema	dbo
Server Nam	win2008r2
Table Designer	
Identity Col	
Indexable	Yes
Lock Escala	Table
Regular Dat	PRIMARY
Replicated	No
Row GUID C	
Text/Image	PRIMARY

(Identity)

Figure 2-3 Column names and identifying information

3. Set the Default Value of the DateCreated column to *(getdate())*; this will insert the current date within each new record for that specific field. See Figure 2-4.


Microsoft SQL Server Management Studio
File Edit View Project Debug Table Designer Tools Window Community Help
New Query
Object Explorer
Connect
WIN2008R2.Ad... dbo.Table_1*
WIN2008R2.Adv....- dbo.Planets
Properties
Column Properties
Item(s) Saved

Column Name	Data Type	Allow Nulls
InvidualID	int	☐
PlanetName	varchar(50)	☑
PlanetType	varchar(50)	☑
Radius	varchar(50)	☑
TimeCreated	datetime	☑
		☐

(General)	
(Name)	TimeCreated
Allow Nulls	Yes
Data Type	datetime
Default Value or Binding	(getdate())
Table Designer	
Collation	<database default>
Computed Column Specification	
Condensed Data Type	datetime
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	No
Indexable	Yes
Is Columnset	No
Is Sparse	No
Merge-published	No
Not For Replication	No

Default Value or Binding

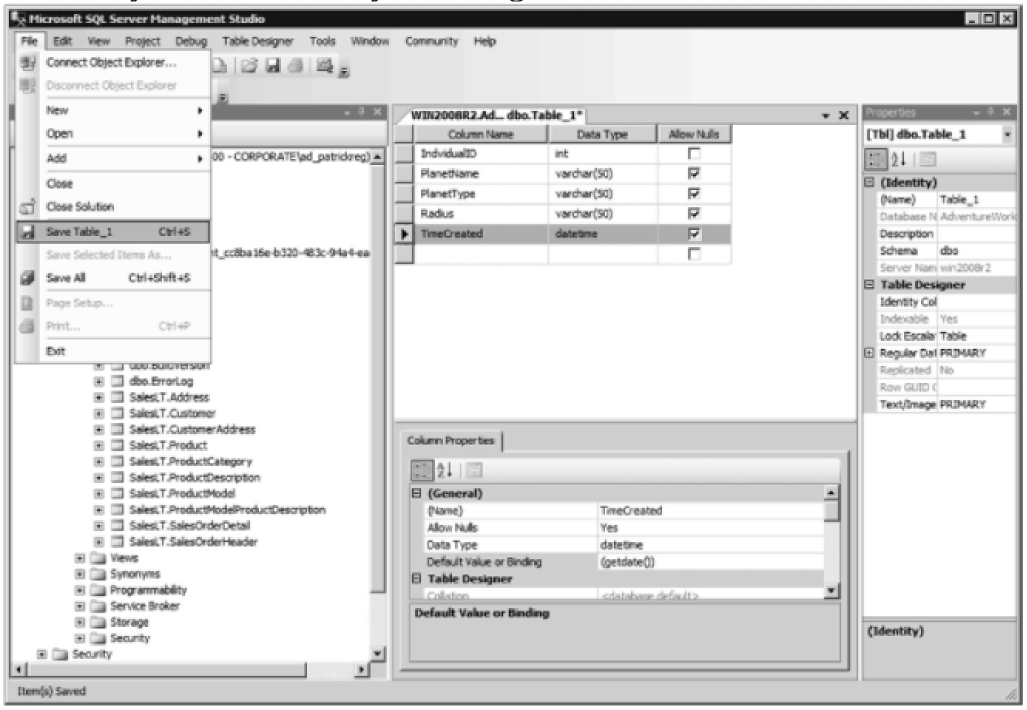
[Tbl] dbo.Table_1

(Identity)	
(Name)	Table_1
Database N	AdventureWork
Description	
Schema	dbo
Server Nam	win2008r2
Table Designer	
Identity Col	
Indexable	Yes
Lock Escala	Table
Regular Dat	PRIMARY
Replicated	No
Row GUID C	
Text/Image	PRIMARY

(Identity)

Figure 2-4 Setting the Table Designer properties

4. Save your new table by selecting File > Save Table_1, as shown in Figure 2-5.



Column Name	Data Type	Allow Nulls
IndividualID	int	☐
PlanetName	varchar(50)	☑
PlanetType	varchar(50)	☑
Radius	varchar(50)	☑
TimeCreated	datetime	☑
		☐

(General)

(Name)	TimeCreated
Allow Nulls	Yes
Data Type	datetime
Default Value or Binding	(getdate())

Table Designer

Collation	<database default>

Default Value or Binding

(Identity)

(Name)	Table_1
Database N	AdventureWork
Description	
Schema	dbo
Server Nam	win2008r2

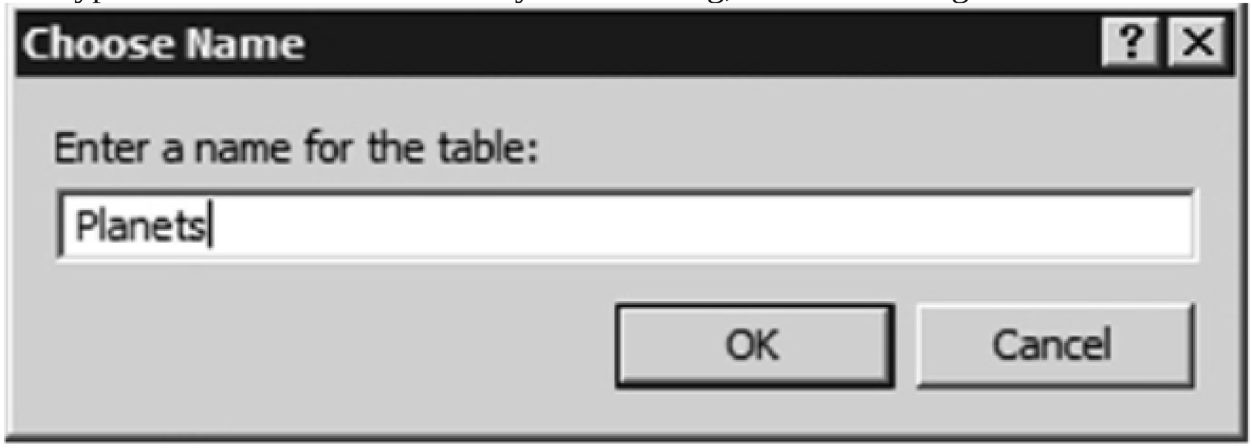
Table Designer

Identity Col	
Indexable	Yes
Lock Escala	Table
Regular Dat	PRIMARY
Replicated	No
Row GUID C	
Text/Image	PRIMARY

(Identity)

Figure 2-5 Saving the new table

5. Type the new name of the table you are saving, as shown in Figure 2-6.



Enter a name for the table:

Planets

Figure 2-6 Naming the table

Your new table will appear under the **Tables** section, as depicted in Figure 2-7.

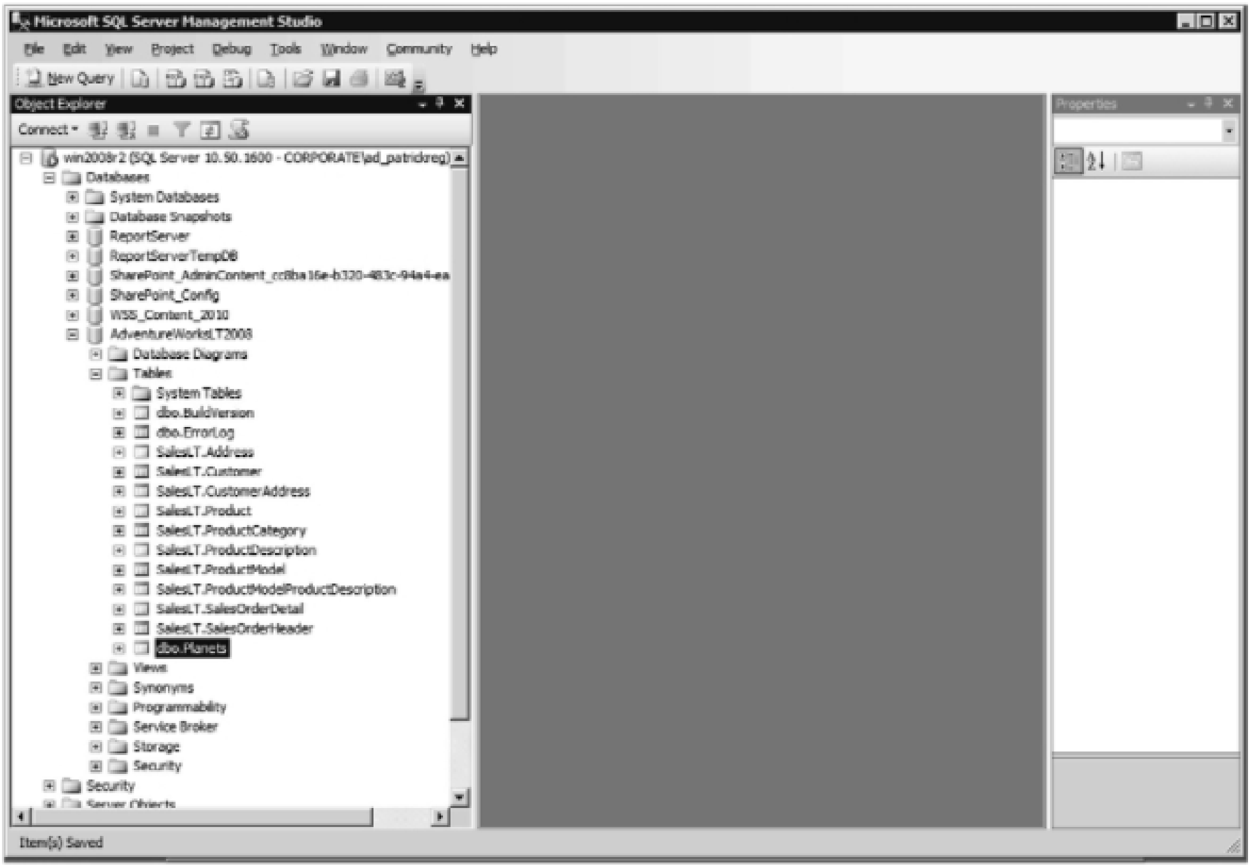


Figure 2-7 The newly created table

PAUSE. Leave the SSMS interface open for the next exercise.

CREATE A TABLE USING TRANSACT-SQL COMMANDS

Creating tables within SSMS is simple because SSMS is an easy-to-use graphical interface. But how can you create tables using ANSI SQL syntax? Quite simply, you will use the `create table` statement to accomplish this task. An example of proper Transact-SQL syntax for creating a table is as follows:

```
CREATE TABLE planets (name varchar(50), diameter varchar(50))

INSERT INTO planets (name, diameter) VALUES (‘earth’, 10000)
```

Note that if SQL Server didn’t support implicit conversion, the following syntax would be needed:

CERTIFICATION READY
How would you create a table using SSMS, and how would you create a table using Transact-SQL commands?
2.2

```
CREATE TABLE planets (name varchar(50), diameter varchar(50))

INSERT INTO planets (name, diameter) VALUES (‘earth’, CAST (10000 as

varchar(50)))
```

Creating Views

THE BOTTOM LINE
As a database administrator, you must understand when to use views. You should also know how to create views by using either a Transact-SQL statement or the graphical designer.

A ***view*** is simply a virtual table consisting of different columns from one or more tables. Unlike a table, a view is stored in a database as a query object; therefore, a view is an object that obtains its data from one or more tables. Views that are based on this definition are referred to as underlying tables. Once you have defined a view, you can reference it as you would any other table in a database.

A view is meant to be a security mechanism; that is, a view ensures that users can retrieve and modify only the data seen by them through their permissions, thus ensuring they cannot see or access the remaining data in the underlying tables. A view is also a mechanism to simplify query execution. Complex queries can be stored in the form of a view and data from the view can then be mined using simple query statements.

Views ensure the security of data by restricting access to the following data:

- Specific rows of tables
- Specific columns of tables
- Specific rows and columns of tables
- Rows obtained by using joins
- Statistical summaries of data in given tables
- Subsets of another view or subsets of views and tables

Some common examples of views include the following:

- A subset of rows or columns of a base table
- A union of two or more tables
- A join of two or more tables
- A statistical summary of base tables
- A subset of another view or some combination of views and base tables

Database views are designed to create a virtual table that is representative of one or more tables in an alternative way. There are two major reasons you might want to provide a view instead of enabling users to access the underlying tables in your database:

- Views allow you to limit the type of data users can access. You can grant view permissions in designated tables, and you can also choose to deny permissions for certain information.
- Views reduce complexity for end users so they don't have to learn how to write complex SQL queries. Instead, you can write those queries on their behalf and hide them in a view.

When creating a view, be sure to consider database performance in your design. As discussed briefly in Lesson 1, indexing plays a role in query time and an even greater role in database performance improvements. But tread lightly: Adding indexes to the schema can actually increase the overhead of your database due to the ongoing maintenance of these indexes.

There are two methods for creating a view:

- By using SSMS
- By writing a Transact-SQL statement

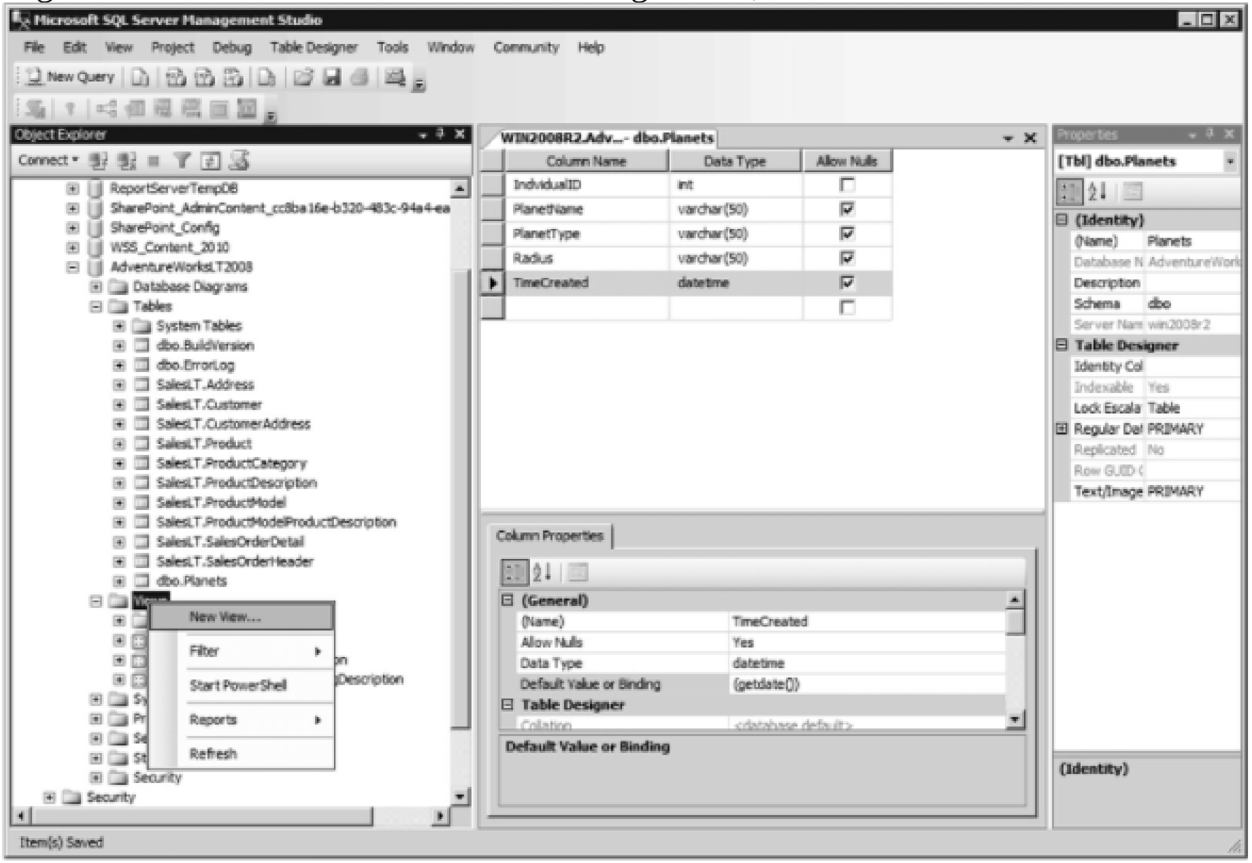
We'll cover both procedures in this section.

CREATE A VIEW USING SSMS

GET READY. Before you begin these steps, make sure SSMS is open and the database to which you wish to add a view is highlighted. Then, follow these steps to create your view:

Right-click the Views folder as shown in Figure 2-8, then select New View.

1. Expand the Views section by clicking the plus sign (+) next to Views. Right-click the Views folder as shown in Figure 2-8, then select New View.



Column Name	Data Type	Allow Nulls
IndividualID	int	☐
PlanetName	varchar(50)	☑
PlanetType	varchar(50)	☑
Radius	varchar(50)	☑
TimeCreated	datetime	☑
		☐

(General)

(Name)	TimeCreated
Allow Nulls	Yes
Data Type	datetime
Default Value or Binding	(getdate())

Table Designer

Collation	<database default>

Default Value or Binding

[Tbl] dbo.Planets

(Identity)

(Name)	Planets
Database N	AdventureWork
Description	
Schema	dbo
Server Nam	win2008r2

Table Designer

Identity Col	
Indexable	Yes
Lock Escala	Table
Regular Dat	PRIMARY
Replicated	No
Row GUID C	
Text/Image	PRIMARY

(Identity)

Figure 2-8 Creating a new view

The Add Table dialog box will open (see Figure 2-9).

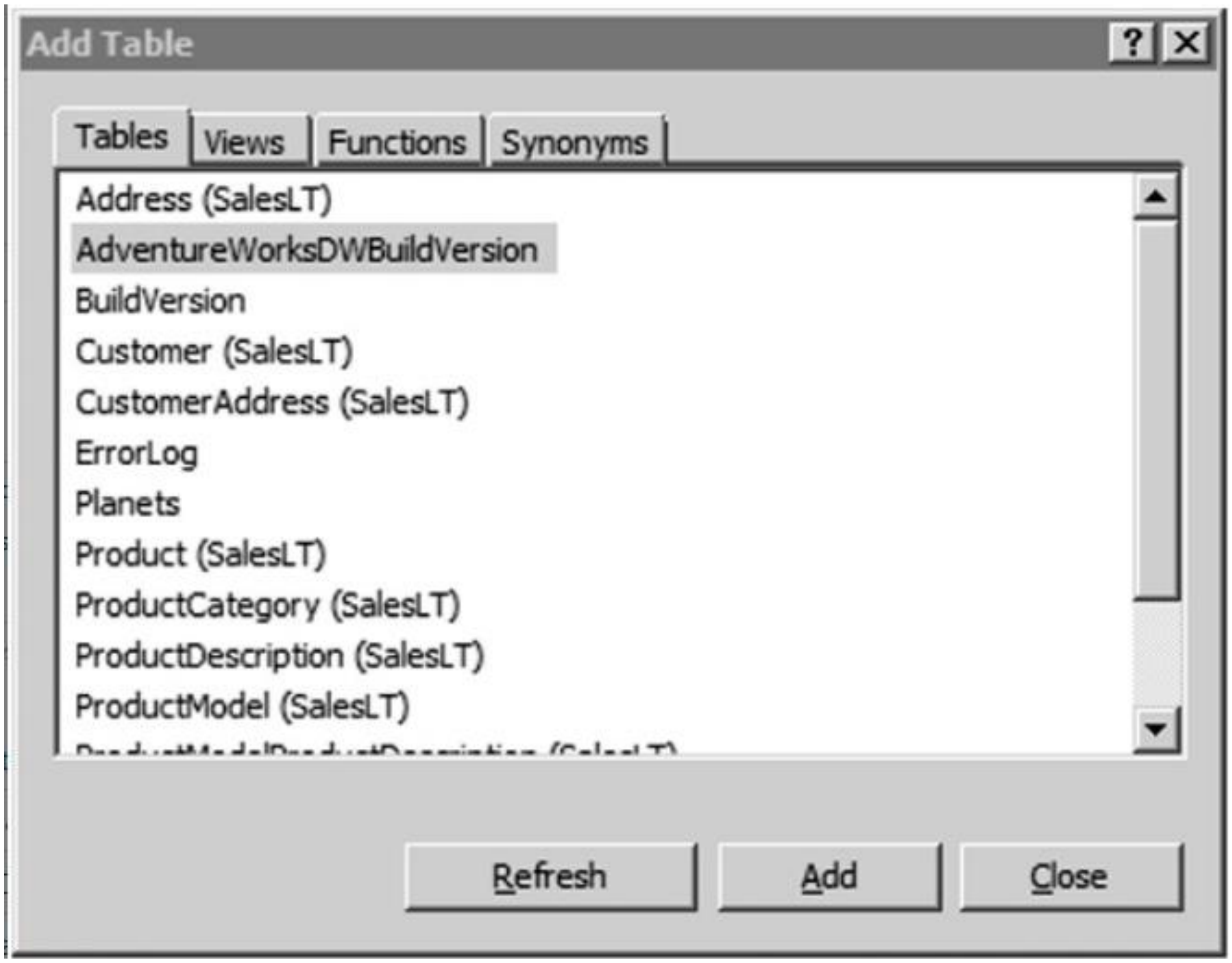


Figure 2-9 Add Table dialog box

Let's explain a little bit about what this dialog box allows you to do:

- To specify the table to be used as the primary source, click the appropriate table in t Tables tab of the dialog box.
- To use another existing view, click the Views tab of the dialog box.
- If you want to generate records from a function, you will find that on the Functions ta
- If you want to use more than one source, you can click each of the different tabs to fi the table, view, or function you wish to add to your query.
- Once you have selected the desired source(s), simply click the Add button for each on
- Once you have selected and added all your desired sources, click the Close button to e the Add Table dialog box.

2. As you click Add to add each source, you will see the information shown in Figure 2-10.

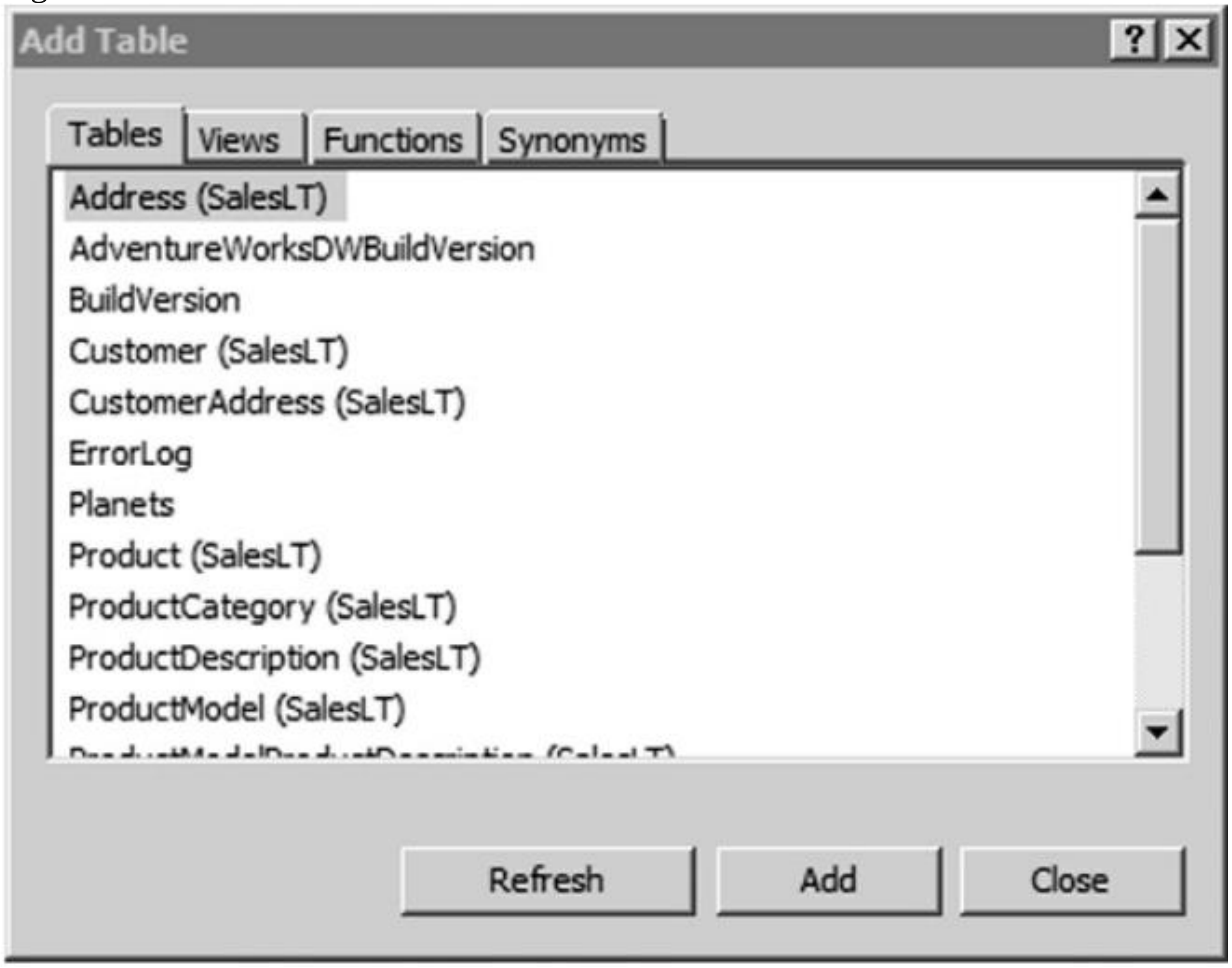


Figure 2-10 Add Table dialog box output

After you have selected the objects you wish to use, the View Designer toolbar will be added, in which you can further map out the views you wish to incorporate into your query.

You can also create views using Transact-SQL. Here, once you add your sources to the diagram pane, the syntax for these sources is shown in the SQL pane.

CERTIFICATION READY
How do you create a view using SSMS?
2.3

To create a view using Transact-SQL syntax, a simple convention is as follows:

```
CREATE VIEW vwCustomer

AS

SELECT CustomerId, Company Name, Phone

 FROM Customers
```

This creates a view called vwCustomer that will be stored as an object. Here, the data that is queried from the columns comes from the Customers table.

Creating Stored Procedures

THE BOTTOM LINE
By creating stored procedures and functions, you make it possible to select, insert, update, or delete data using these statements.

So far, you've learned how to use different data types to create tables and views through the SSMS interface as well as through Transact-SQL syntax statements. Now it's time to learn how to create stored procedure statements using the same graphic interface.

A ***stored procedure*** is a previously written SQL statement that has been "stored" or saved into a database. One way to save time when running the same query over and over again is to create a stored procedure that you can then execute from within the database's command environment. An example of executing a stored procedure is as follows:

```
exec usp_displayallusers
```

Here, the name of the stored procedure is "usp_displayallusers," and "exec" tells SQL Server to execute the code in the stored procedure. Indeed, when you create your own stored procedure, it will have the designation "usp" in front of it, which indicates to SQL that this is a user-created stored procedure.

Now, say that your stored procedure named "displayallusers" has a simple code inside it, such as the following:

```
SELECT * FROM USERLIST
```

What this select statement does is return all the data that is found in the USERLIST table. One question you may be asking right now is, "Why can't I just run the query I want to return the information I need? In other words, you may wonder why you should bother with creating a stored procedure. Note that the "*" you see in the above statement means you are not defining criteria you would like matched in the output. In other words, you are returning *all* records from the userlist table.

Perhaps you are working on a website built with ASP pages and you need to call a stored procedure from that, or from within another application such as Visual Basic, or from another application entirely. Using a stored procedure allows you to store all the logic inside the database, so by using a simple command, you can query and retrieve all information from all sources.

A stored procedure is an already-written SQL statement that is stored in a database. If you are continually using the same SQL statement within your database, it is simpler to create a stored procedure for it. Now, simple statements like a "select" statement would not entirely benefit from a stored procedure, but if you are creating complex query statements, your best bet is to create a stored procedure for them and run that stored procedure from within the Query Analyzer using an execute (exec) command.

CREATE A STORED PROCEDURE

GET READY. Before you begin these steps, make sure SSMS is open and the database to which you wish to add a view is highlighted. Then, follow these steps to create a stored procedure:

1. Expand the Programmability section by clicking the appropriate + sign, then expand the Stored Procedure section by clicking the appropriate + sign.
2. Right-click Stored Procedures and choose New Stored Procedure (see Figure 2-11).

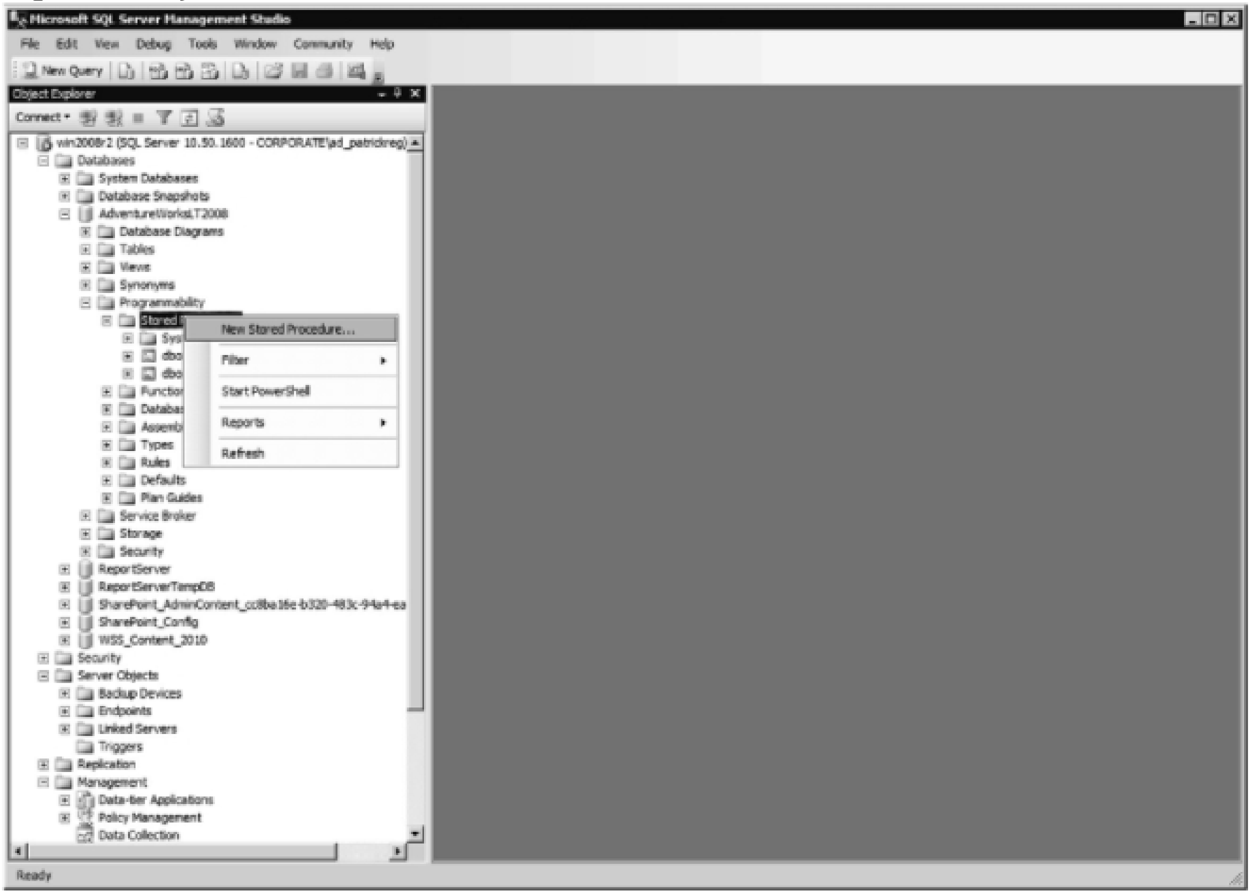


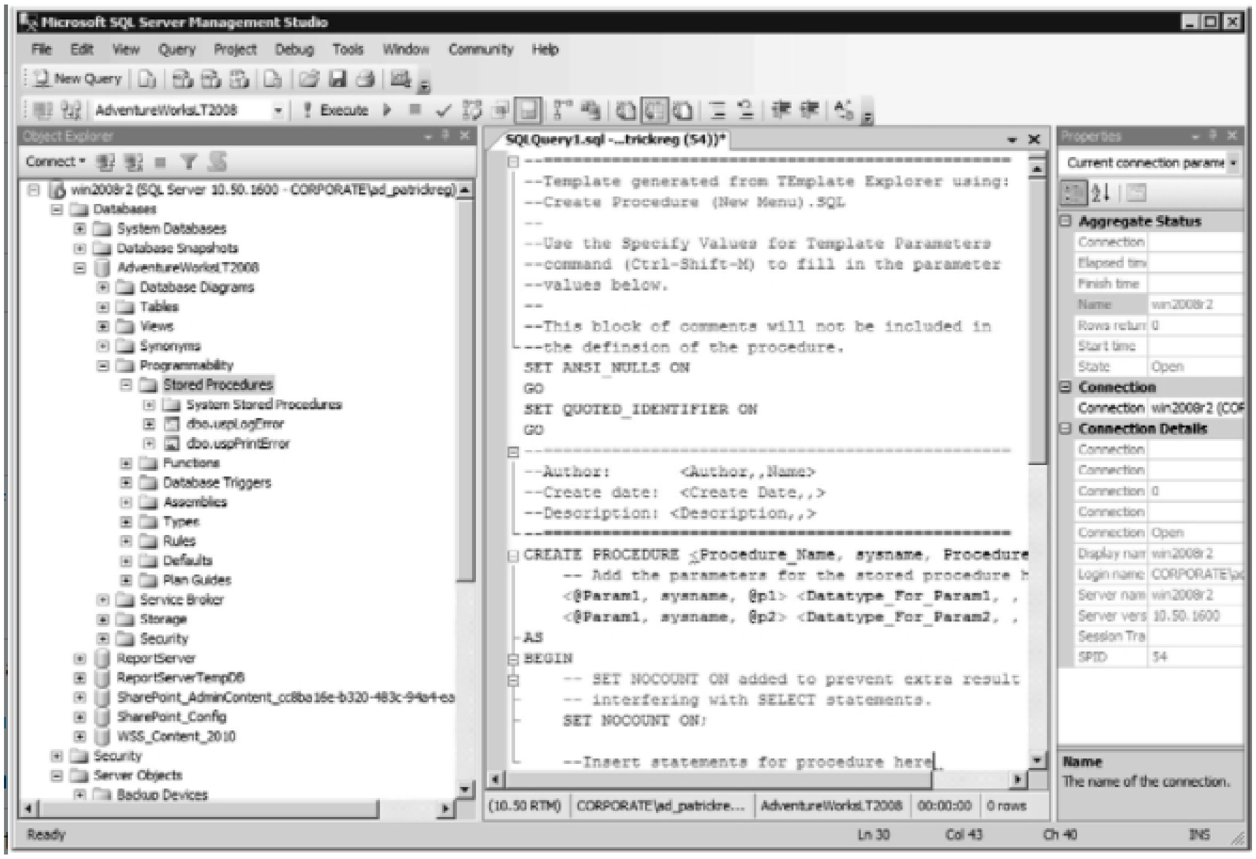
Figure 2-11 New Stored Procedure selection menu

The Text Editor window will open (see Figure 2-12), displaying the syntax. The window contains a ready-made stored procedure template for you to add your own view parameters.

CERTIFICATION READY

How would you create a stored procedure with SSMS?

2.4


Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Community Help
New Query
AdventureWorksLT2008
Execute
Object Explorer
Connect
SQLQuery1.sql -...trickreg (54))*
Properties
Current connection parame
Aggregate Status
Connection
Connection Details
Name
The name of the connection.
Ready
Ln 30
Col 43
Ch 40
INS

```
--==============================================================
--Template generated from TEmplate Explorer using:
--Create Procedure (New Menu).SQL
--
--Use the Specify Values for Template Parameters
--command (Ctrl-Shift-M) to fill in the parameter
--values below.
--
--This block of comments will not be included in
--the definsion of the procedure.
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
--==============================================================
--Author:         <Author,,Name>
--Create date:    <Create Date,,>
--Description: <Description,,>
--==============================================================
CREATE PROCEDURE <Procedure_Name, sysname, Procedure
     -- Add the parameters for the stored procedure h
     <@Param1, sysname, @p1> <Datatype_For_Param1, ,
     <@Param1, sysname, @p2> <Datatype_For_Param2, ,
AS
BEGIN
     -- SET NOCOUNT ON added to prevent extra result
     -- interfering with SELECT statements.
     SET NOCOUNT ON;

     --Insert statements for procedure here
```

Figure 2-12 Sample Text Editor window

Microsoft SQL Server already has hundreds of system-stored procedures so that you can perform basic functions. For example, you can use the Select Stored Procedure to retrieve or select rows from a database. Some of the more popular stored procedures will be covered in the next lesson including `SELECT` , `INSERT` , `UPDATE` , and `DELETE` .

Understanding SQL Injections

TAKE NOTE*
A SQL injection is an attack in which malicious code is inserted into strings to be passed on later when parsing or executing statements.

Before you learn the syntax statements for selecting, inserting, updating, and deleting data, you need to understand what a SQL injection is. In short, a ***SQL injection*** is an attack in which malicious code is inserted into strings that are later passed on to instances of SQL Server waiting for parsing and execution. Any procedure that constructs SQL statements should be reviewed continually for injection vulnerabilities because SQL Server will execute all syntactically valid queries from any source.

The primary form of SQL injection is a direct insertion of code into user-input variables that are concatenated with SQL commands and then executed. A less direct method of attack injects malicious code into strings that are destined for storage in a table or are considered metadata. When these stored strings are subsequently concatenated into the dynamic SQL command, the malicious code will be executed.

The injection process's function is to terminate a text string prematurely and append a new command directed from it; because the inserted command may have additional strings appended to it before it is executed, the malefactor terminates the injected string with a comment mark "—", making subsequent text ignored at execution time.

SKILL SUMMARY

IN THIS LESSON, YOU LEARNED THE FOLLOWING:	
1	A data type is an attribute that specifies the type of data an object can hold, as well as how many bytes each data type takes up.
2	As a general rule, if you have two data types that differ only in how many bytes each uses, the one with more bytes has a larger range of values and/or increased precision.
3	Microsoft SQL Server includes a wide range of predefined data types called built-in data types. Most of the databases you will create or use will employ only these data types.
4	Exact numeric data types are the most common SQL Server data types used to store numeric information.
5	`int` is the primary integer (whole number) data type.
6	Precision (`p`) is the maximum total number of decimal digits that can be stored in a numeric data type, both to the left and to the right of the decimal point; this value must be at least 1 and at most 38. The default precision number is 18.
7	`money` and `smallmoney` are Transact-SQL data types you would use to represent monetary or currency values. Both data types are accurate to 1/10,000th of the monetary units they represent.
8	Approximate numeric data types are not as commonly used as other SQL Server data types. If you need more precision (more decimal places) than is available with the exact numeric data types, you should use the `float` or `real` data types, both of which typically take additional bytes of storage.
9	The date and time data types, of course, deal with dates and times. These data types include `date` , `datetime2` , `datetime` , `datetimeoffset` , `smalldatetime` , and `time` .
10	SQL Server supports implicit conversions, which can occur without specifying the actual callout function `(cast or convert).` Explicit conversions require you to use the functions `cast` or `convert` specifically.
11	A regular character uses one byte of storage for each character, which allows you to define one of 256 possible characters; this accommodates English and some European languages
12	A Unicode character uses two bytes of storage per character so that you can represent one of 65,536 characters. This added capacity means that Unicode can store characters from almost any language.

13	When you use a VAR element, SQL Server will preserve space in the row in which this element resides on the basis of on the column’s defined size and not the actual number of characters in the character string itself.
14	The Unicode character strings `nchar` and `nvarchar` can either be fixed or variable, like regular character strings; however, they use the UNICODE UCS-2 character set.
15	The purpose of a table is to provide a structure for storing data within a relational database.
16	A view is simply a virtual table that consists of different columns from one or more tables. Unlike a table, a view is stored in a database as a query object; therefore, a view is an object that obtains its data from one or more tables.
17	A stored procedure is a previously written SQL statement that has been stored or saved into a database.
18	A SQL injection is an attack in which malicious code is inserted into strings that are later passed on to instances of SQL Server for parsing and execution.