Lab 1 - Introduction to Microsoft SQL Server

Content:
- Introduction to the environment of Microsoft SQL Server
- Introduction the installation and the running platform of the database server
- How to execute/start/stop the database server
- Manage main functions in the environment
- Manage database schema and tables over environment's interface

Duration: 4 teaching periods

Learning outcome:
- Gain background knowledge about Microsoft SQL Server
- Manage fundamental functions of Microsoft SQL Server
- Use functions on the interface of the server to create tables

This lab introduces Microsoft SQL Server 2014. You may use other versions of MS SQL Server, or MySQL.

Part 1: Starting Microsoft SQL Server 2014
[refer: Beginning SQL Server 2012 for Developers book, pages 32-49]

Start the server by Select: Start > Programs > Microsoft SQL Server Management Studio.

The following dialog appears:
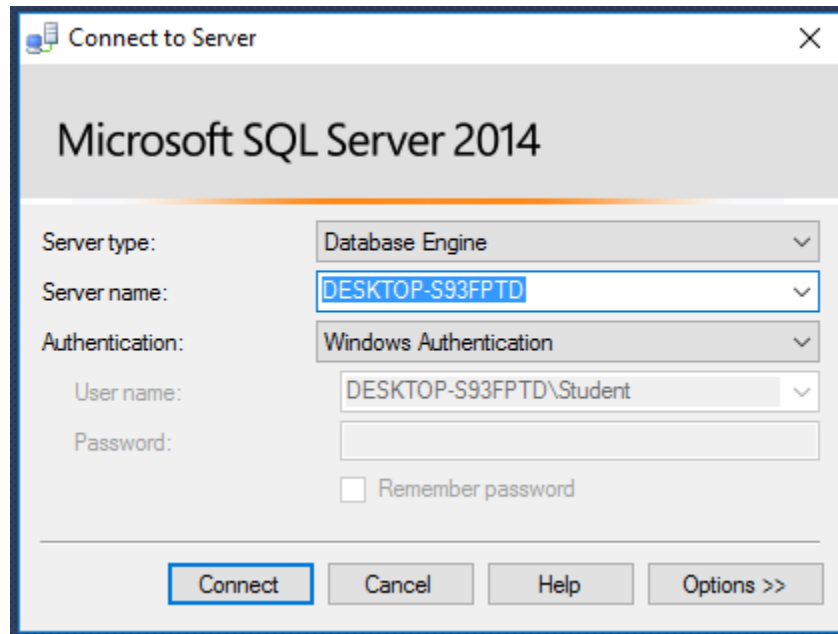
Options:

Server type

> When registering a server from Object Explorer, select the type of server to connect to: Database Engine, Analysis Services, Reporting Services, or Integration Services. The rest of the dialog shows only the options that apply to the selected server type. When registering a server from Registered Servers, the **Server type** box is read-only, and matches the type of server displayed in the Registered Servers component. To register a different type of server, select Database Engine, Analysis Services, Reporting Services, SQL Server Compact, or Integration Services from the Registered Servers toolbar before starting to register a new server.

Server name

> Select the server instance to connect to. The server instance last connected to is displayed by default.

Authentication

> Two authentication modes are available when connecting to an instance of the Database Engine.

User name

> Enter the user name to connect with. This option is only available if you have selected to connect using Windows Authentication.

Login

> Enter the login to connect with. This option is only available if you have selected to connect using SQL Server Authentication.

Password

> Enter the password for the login. This option is only editable if you have selected to connect using SQL Server Authentication.

Connect

> Click to connect to the server selected above.

Authentication Modes

There are two modes: Windows authentication mode and mixed mode.

Windows authentication mode: Microsoft SQL Server 2014 uses Windows user/password to allow users log in the server.

Mixed mode: the server can use Windows or its own user/password to allow users log in the server. You can manage user information in the server's environment.

The sa Login

The sa login is a default login that has full administration rights for SQL Server. This account is unique in SQL Server and set password during the installation. If you logged in to SQL Server as sa, you will have full control over any aspect of SQL Server. Therefore, try to limit to log in the server by this account and organize appropriate user account systems.

Practice: Main Interface of Microsoft SQL Server 2014:

1. The first area of SQL Server we will look at is the Registered Servers Explorer. Access this explorer, shown in the following figure, by selecting **View → Registered Servers** or by pressing Ctrl+Alt+G. This area details all SQL Server servers that have been registered by you in your installation. At present, there will only be the server just registered.
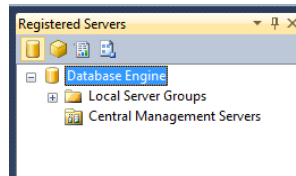

2. If you need to register another server, right-click the **Local Server Group** under the **Database Engine** node and select **New Server Registration** to bring up a dialog box very similar to the Connect to Server dialog box shown earlier. Go ahead and do this now to familiarize yourself with the New Server Registration dialog box, shown in the following figure.
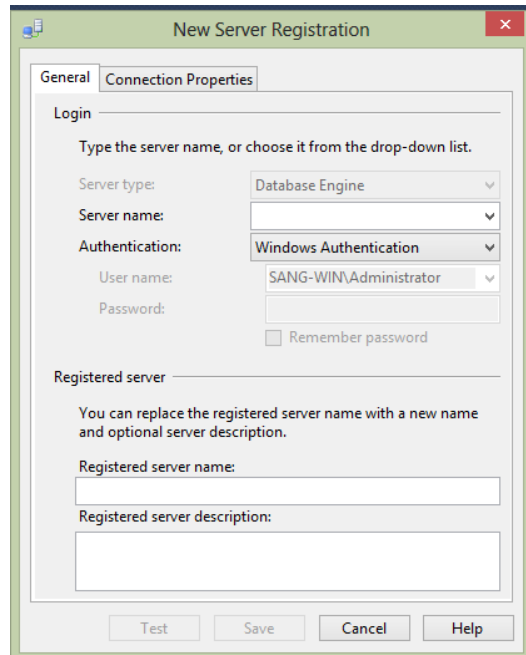
In this dialog, you can registry other servers available in your network to use in the SQL Server Management interface.

3. Moving back to **SQL Server explorer** window below the registered servers, take a look at the Object Explorer, which should have been present when you first brought up SSMSE. If it isn't there or if it disappears, you can redisplay it by selecting **View →** **Object Explorer** or by pressing F8. You will likely use this explorer the most, as it details every object, every security item, and many other areas concerning SQL Server Express. You can see that SQL Server uses nodes (which you expand by clicking the + signs) to keep much of the layout of the Object Explorer (the hierarchy) compact and hidden until needed. Let's go through each of the nodes you see in the following figure now.
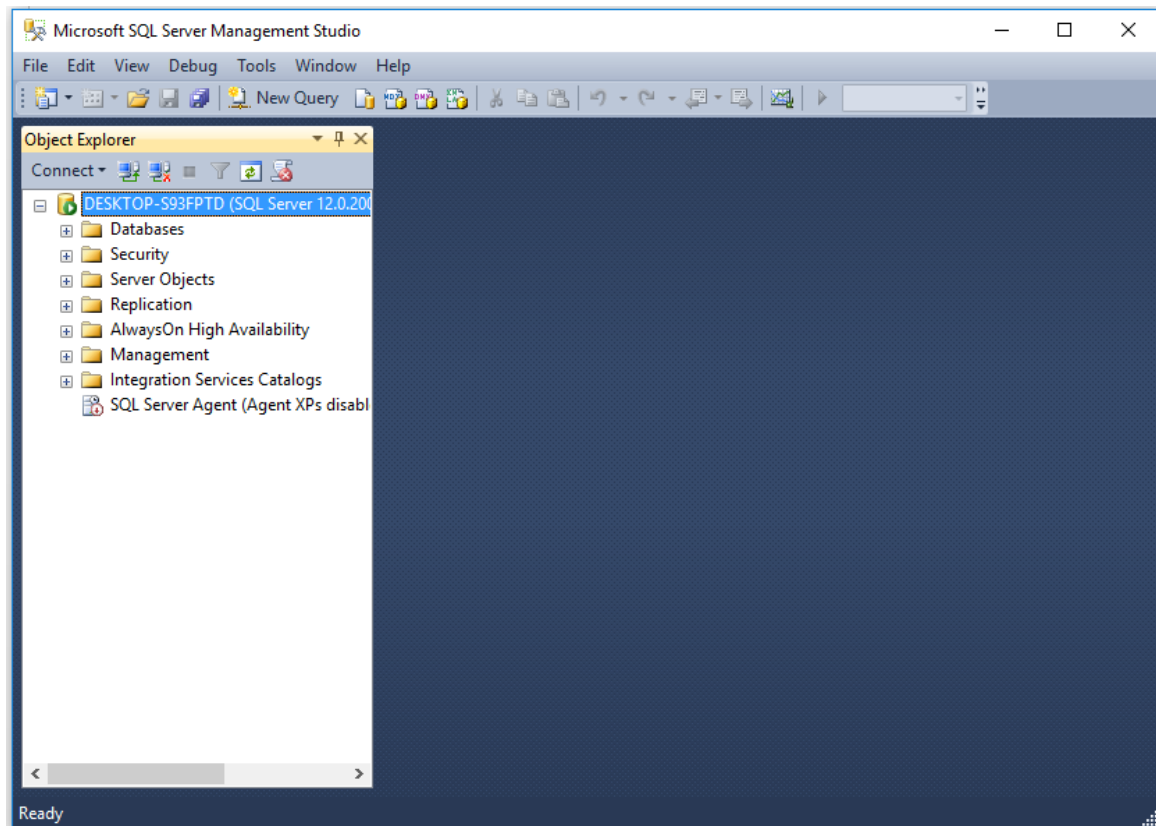
In this figure:

- *Databases*: Holds the system and user databases within the SQL Server you are connected to.
- *Security*: Details the list of SQL Server logins that can connect to SQL Server. You will see more on this in a later lab.
- *Server Objects*: Details objects such as backup devices and provides a list of linked servers, where one server is connected to another remote server.
- *Replication*: Shows the details involving data replication from a database on this server to another database (on this or another server) or vice versa.
- *Management*: Details maintenance plans, which you will learn more about in later lab, and provides a log of informational and error messages that can be very useful when troubleshooting SQL Server.

5. Moving to the menu bar of SQL Server environment, the first item of interest is the **View** menu option. The first four options on the View menu, shown in the following figure, bring up the three explorer windows, Object Explorer, Object Explorer Details and Registered Servers Explorer. Therefore, if you ever need to close these items to give yourself more screen space, you can reopen them from the menu or with the shortcut keys you see defined.
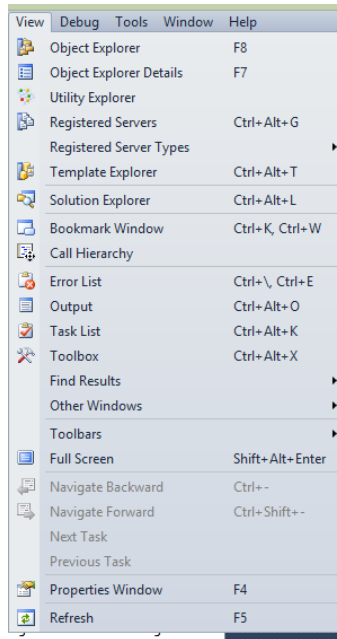
The other options on the View menu are as follows:

- Template Explorer: Provides access to code templates. In the examples in this book, we will be building objects using T-SQL. Rather than starting from scratch, we can use code templates that contain the basic code to create these objects.
- Properties Window: Displays the set of properties for each object.
- Bookmark Window: Allows you to create bookmarks, which you place into various locations in your code to allow you to jump quickly to those locations.

6. The final part of SQL Server that we will take a look at is the main SQL Server toolbar, as you can see in the following figure. Some of the icons, such as the Save icon, will be instantly recognizable, but let's go through each button, from left to right, so that it is clear what they all do.
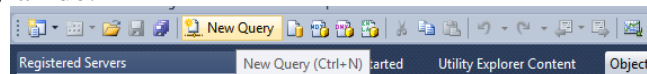
Clicking the **New Query** button will open up a new query window, which allows you to do this using the connection already made with SQL Server.

Similar to the New Query button, the **Database Engine Query** button will also create a new query window. However, this will give you the option of having a different connection to SQL Server through which to run your code. This is a good way of testing code with a different connection to ensure that you cannot see data that should be secure, such as wages, via that connection.

Part 2: Managing Tables

What is a Table?

A table is a repository for data, with items of data grouped in one or more columns. Tables contain zero or more rows of information. An Excel spreadsheet can be thought of as a table, albeit a very simple table with few or no rules governing the data.

Look at the following table in Excel:

A	B	C	D	E	F
Robin	Dewson	24/03/1964	80	Fat-Belly Dr	UK
Bernie	McGee	15/10/1955	121b	The Crescent	UK
Anthony	Jawad	31/12/1969	Flat 7	Bank Street	UK

In this table, first three columns contain data that can be assumed to be first name, last name, and date of birth, but the fourth column is free-format and varies between a hotel room number, a house number, and a flat number. There is no consistency. In fact, in Excel, all the columns could in reality contain any data.

In Microsoft SQL Server 2014, a table will have specific types of data held in each column, and a predetermined type of data defined for a column can never change without affecting every row of data within that column for that table.

At the time a table is created, every column will have a specific data type. Therefore, very careful consideration has to be made when defining a table to ensure that the column data type is the most appropriate. There is no point in selecting a generic data type (a string, for example) to cover all eventualities, as you would have to revisit the design later anyway. Leaving all data types as a string would cause problems with math functions, for example, when you needed to add taxes to a price or when you want to add a number of days to a date for delivery times over order date. A table's purpose is to hold specific information. The table requires a meaningful name and one or more columns, each given a meaningful name and a data type.

When adding or modifying a table, a user has to have a right to do so. His/her right can be assigned by more powerful users, such as `sysadmin` (the administrator of the whole server).

The rows of data that will be held in a table should be related logically to each other. If a table is defined to hold customer information, this is all it should hold. Under no circumstances should you consider putting information that was not about a customer in the table. It would be illogical to put, for example, details of customer's orders within it.

Data Types

Microsoft SQL Server 2014 has many different data types that are available for each column of data.

char

The char data type is fixed in length. If you define a column to be 20 characters long, 20 characters will be stored. If you enter less than the number of characters defined, the remaining length will be space filled to the right. Therefore, if a column were defined as char (10), "aaa" would be stored as "aaa   ". Use this data type when the column data is to be of fixed length, which tends to be the case for customer IDs and bank account IDs.

nchar

The nchar type is exactly the same as char, but will hold characters in Unicode format rather than ANSI. The Unicode format has a larger character set range than ANSI. ANSI character sets only hold up to 256 characters. However, Unicode character sets hold up to 65,536 different characters. Unicode data types do take up more storage in MS SQL server; in fact, MS SQL server allocates double the space internally, so unless there is a need in your database to hold this type of character, it is easier to stick with ANSI. If you define the column with no value, a length of 1 will be defined. However, this is not good practice, and all lengths should be defined for clarity.

varchar

The varchar data type holds alphanumeric data, just like char. The difference is that each row can hold a different number of characters up to the maximum length defined. If a column is defined as varchar(50), this means that the data in the column can be up to a maximum of 50 characters long. However, if you only store a string of 3 characters, only three storage spaces are used up. This definition is perfect for scenarios where there is no specific length of data; for example, people's names or descriptions where the length of the stored item does not matter. The maximum size of a varchar column is 8,000 characters. However, if you define the column with no size, that is, varchar(), the length will default to 1.

nvarchar

The nvarchar type is defined in a similar way to varchar, except it uses Unicode and therefore doubles the amount of space required to store the data.

text

It is useful to understand this data type in case you come across it in any legacy systems that have been upgraded to SQL Server Express 2005. If you need to hold any character data that will always be longer than 8,000 characters, you should not use varchar(max). This is where the text data type comes into play. These data types can hold up to 2GB of data, and could be used to hold notes about customers in a call center, for example. However, text data types are usually different from other data types. Because such a large amount of data can be stored in this data type, it doesn't make sense to store this data within each row of SSE. If you think about it, you would very quickly have a vast database holding very little data. Therefore, if you are storing data within this data type, the data itself is held elsewhere. A pointer is held within SSE in the column defined as a text data type, pointing to where the data is physically held. However, you can store up to 8,000 characters of physical data, if you wish, within this data type within the row; but

really, if you have decided to use text as a data type, you are expecting large amounts of data, and therefore it would be best to keep the data outside the database.

ntext
This data type is very similar to text, with the exception that the data is stored as Unicode, and only 1GB of characters can be stored because this data type takes double the amount of space to store one character of text. This data type will also be removed in a future version of SSE, and therefore you should use nvarchar(max) instead.

image
image is very much like the text data type, except this is for any type of binary data, which includes images but could also include movies, music, and so on.

int
The int, or integer, data type is used for holding numeric values that do not have a decimal point (whole numbers). There is a range limit to the value of the numbers held: int will hold any number between the values of –2,147,483,648 and 2,147,483,647.

bigint
A bigint, or big integer, data type is very similar to int, except that much larger numbers can be held. A range of –9,223,372,036,854,775,808 through to 9,223,372,036,854,775,807 can be stored.

smallint
The smallint data type, or small integer, holds small integer numbers in the range of –32,768 through to 32,767. Do take care when defining columns with this data type and make sure there really is no possibility of exceeding these limits. There is always a big danger when creating a column with this data type that you have to go back and change the data type, so if in doubt, select int.

tinyint
The tinyint, or tiny integer, data type is even smaller than smallint and holds numbers from 0 through to 255. It could be used to hold a numeric value for each US or Canadian state or perhaps every county in the United Kingdom.

decimal/numeric
Both of these data types hold the same precision and ranges of data. The range is from –10 to the power 38 + 1 through to 10 to the power 38 – 1. These are quite large ranges, from –0.00000000000000000000000000000000000001 through to 10,000,000,000,000,000,000,000,000,000. However, do take care with this, as you cannot store 38 digits to the right and left of the decimal point. You can only store up to and including 38 digits. So, the greater the precision required to the right of the decimal point, the fewer digits are left to represent the whole number.

float

This is used for numbers where the decimal point is not fixed. float data types hold very large numbers in the range of –1.79E+308 through 1.79E+308. There is a warning with this data type: the values cannot always be seen as 100% accurate, as they can be approximate. The approximation arises from the way the number is physically stored as binary code. You will have problems where a number ends in .3, .6, or .7. The value stored has to be approximated, or rounded, as some values can't be stored accurately, for they may have more decimal places than can be catered to. A well-known example is the value of Pi.

real
The real data type is very much like float, except that real can store only numbers in the range of –3.40E+38 through 3.40E+38. This data type also holds an approximate value.

money
The money data type is used for holding numeric values up to four decimal places. If you need to use more than four decimal places, you need to look to another data type, such as decimal. This data type doesn't actually store the currency symbol to signify the monetary type, so you should not use this data type for different currency values, although you can combine a column using this data type with a second column defining the currency type. The money data type has a range of –922,337,203,685,477.5808 through 922,337,203,685,477.5807. If you need to store the currency symbol of the currency that is held here ($ or USD for dollars, £ or GBP for British pounds, etc.), you would need to store this separately, as the money data type does not hold the currency symbol. A column defined as money will hold the money to 1/10,000 of a decimal unit, which is a bit of a waste if you are storing the values as Turkish Lira.

smallmoney
This data type is similar to money with the exception of the range, which lies between –214,748.3648 and 214,748.3647.

datetime
This will hold any date and time from January 1, 1753, through to December 31, 9999. However, it stores not only a date, but also a time alongside it. If you just populate a column defined as datetime with a date, a default time of 12:00:00 will be stored as well.

smalldatetime
This data type is very much like datetime, except the date range is January 1, 1900, through to June 6, 2079. The reason for the strange date at the end of the range lies in the binary storage representation of this datetime.

timestamp
This is an unusual data type, as it is used for a column for which you would not be expected to supply a value. The timestamp data type holds a binary number generated by SSE, which will be unique for each row within a database. Every time a record is modified, the column with this data type in the record will be modified to reflect the time

of modification. Therefore, you can use columns with this data type in more advanced techniques where you want to keep a version history of what has been changed.

uniqueidentifier
This data type holds a Globally Unique Identifier, or GUID. This is similar to the timestamp data type, in that the identifier is created by an SSE statement when a record is inserted or modified. The identifier is generated from information from the network card on a machine, processor ID, and the date and time. If you have no network card, the uniqueidentifier is generated from information from your own machine information only. These IDs should be unique throughout the world.

binary
Data held in this data type is in binary format. This data type is mainly used for data held as flags or combinations of flags. For example, perhaps you wanted to hold flags about a customer. You need to know whether the customer is active (value = 1), ordered within the last month (value = 2), last order was for more than $1,000 (value = 4), or meets loyalty criteria (value = 8). This would add up to four columns of data within a database. However, by using binary values, if a client had a value of 13 in binary, the client would have values 1 + 4 + 8, which is active, last order more than $1,000, and meets the loyalty criteria. When you define the column of a set size in binary, all data will be of that size.

varbinary
This data type is very much like binary, except the physical column size per row will differ depending on the value stored. varbinary(max) can hold values more than 8,000 characters in length and should be used for holding data such as images.

bit
This data type holds a value of 0 or 1. Usually, bit is used to determine true (1) or false (0) values.

xml
XML data can be held in its own special data type rather than in a varchar(max) column. There are special query clauses that can then be used to query and work with this data. Prior to SQL Server Express 2005, XML data was almost an afterthought with no data type, and earlier Express versions had extremely limited functionality to work with the XML data that did exist.

Default Values

As a row is added to a table, rather than requiring developers to add values to columns that could be populated by MS SQL Server, such as a column that details using a date and time when a row of data was added, it is possible to place a default value there instead. The default value can be any valid value for that data type. A default value can be overwritten.

Generating IDENTITY Values

When adding a new row to a table, you may wish to give this row a unique but easily identifiable ID number that can be used to link a row in one table with a row in another. Within the ApressFinancial database, there will be a table holding a list of transactions that needs to be linked to the customers table. Rather than trying to link on values that cannot guarantee a unique link (first name and surname, for example), a unique numeric ID value gives that possibility, providing it is used in conjunction with a unique index. If you have a customer with an ID of 100 in the Customers table and you have linked to the Transactions table via the ID, you could retrieve all the financial transactions for that customer where the foreign key is 100. However, this could mean that when you want to insert a new customer, you have to figure out which ID is next via some SQL code or using a table that just held "next number" identities. But fear not, this is where the IDENTITY option within a column definition is invaluable.

By defining a column using the IDENTITY option, what you are informing the SQL Server is that:

- The column will have a value generated by the SQL Server.
- There will be a start point (seed).
- An increment value is given, informing SSE by how much each new ID should increase.
- The SQL Server will manage the allocation of IDs.
- Values cannot be modified, as the column is totally controlled by the SQL Server internally.
- Each row will be unique by virtue of the ID being unique.
- You would have to perform all of these tasks if SSE did not do so. Therefore, by using this option in a column definition, you can use the value generated to create a solid, reliable, and unique link from one table to another, rather than relying on more imprecise selection criteria.

Null Values

Columns can be defined as NULL or NOT NULL. (In the Table Designer, you can check or uncheck the Allow Nulls option.) These two different constraints define whether data must be entered into the column or not. A NULL value means that there is absolutely nothing entered in that column—no data at all. A NULL value is in a special data state, with special meaning. It means that the data value is unknown.

If a value is a NULL value, no data has been inserted into the column for a given row. You have to use the T-SQL IS NULL and IS NOT NULL operators to test for this value. Take the example of a column defined to hold characters, but where one of the rows has a NULL value within it. If you execute a query that carries out string manipulation, the row with the NULL value might cause an error, or the row might not to be included in the result set.

More about SQL Server Management Studio: https://msdn.microsoft.com/en-us/library/hh213248.aspx

Practice: Creating a Table in SQL Server (by the program interface)

1. Ensure that the SQL server on your computer is running.
2. Expand Object Explorer, right-click on Database node to restore the **ApressFinancial** database.
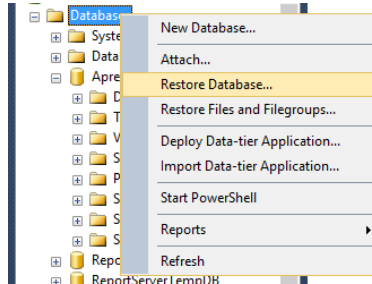

In the Restore Database dialog, you should select the Device option, and then add the **ApressFinancial** file, provided with the lab materials.
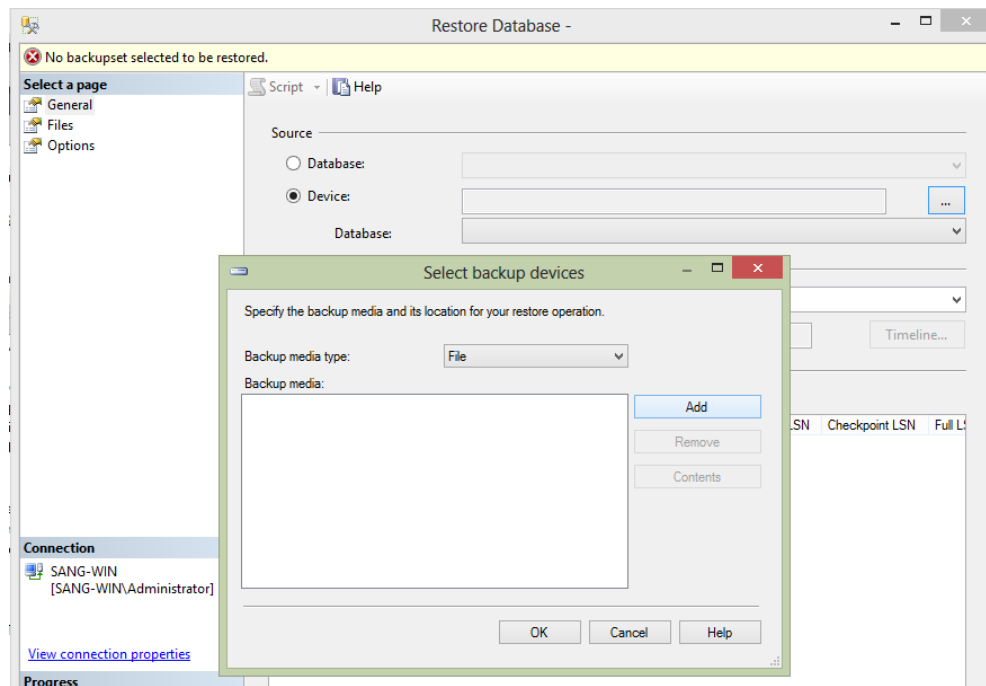
3. Expand the ApressFinancial database so that you can see the Tables node, as shown in the following figure:
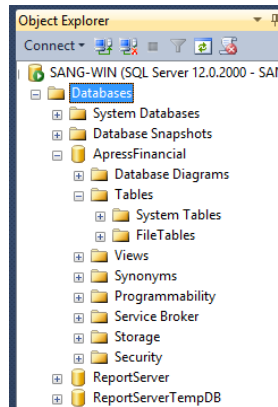
4. Right-click the Tables node and select New Table. This will take you into the Table Designer. The following figure shows how the Table Designer looks when you first enter it.
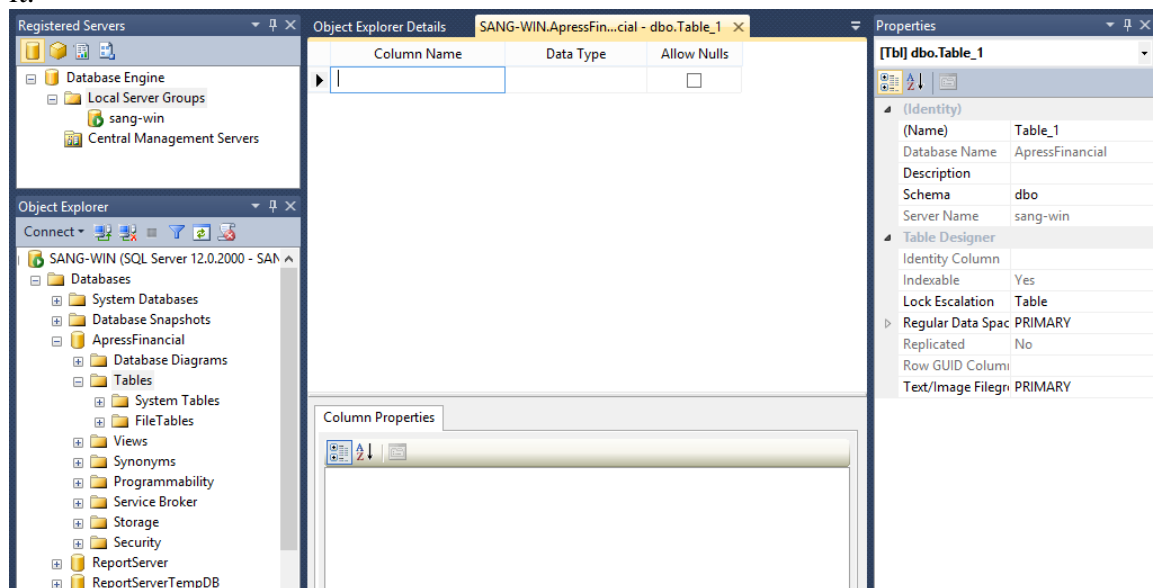
5. From this screen, you need to enter the details for each column within the table. Enter the first column, **CustomerId**, in the Column Name column. When naming columns, try to avoid using spaces. Either keep the column names without spaces, like I have done with CustomerId, or use an underscore (_) instead of a space. It is perfectly valid to have column names with spaces. However, to use these columns in SQL code, we have to surround the names by square brackets, [], which is very cumbersome.

At the moment, notice that Column Properties in the middle of the figure is empty. This will fill up when you start entering a data type after entering the column name. The Column Properties section is just as crucial as the top half of the screen where you enter the column name and data type.

6. The drop-down combo box that lists the data types is one of the first areas provided by the MS server to help us with table creation. This way we don't have to remember every data type there is within SSE. By having all the necessary values listed, it is simple

enough to just select the most suitable one. In this instance we want to select **bigint**, as shown in the following figure.
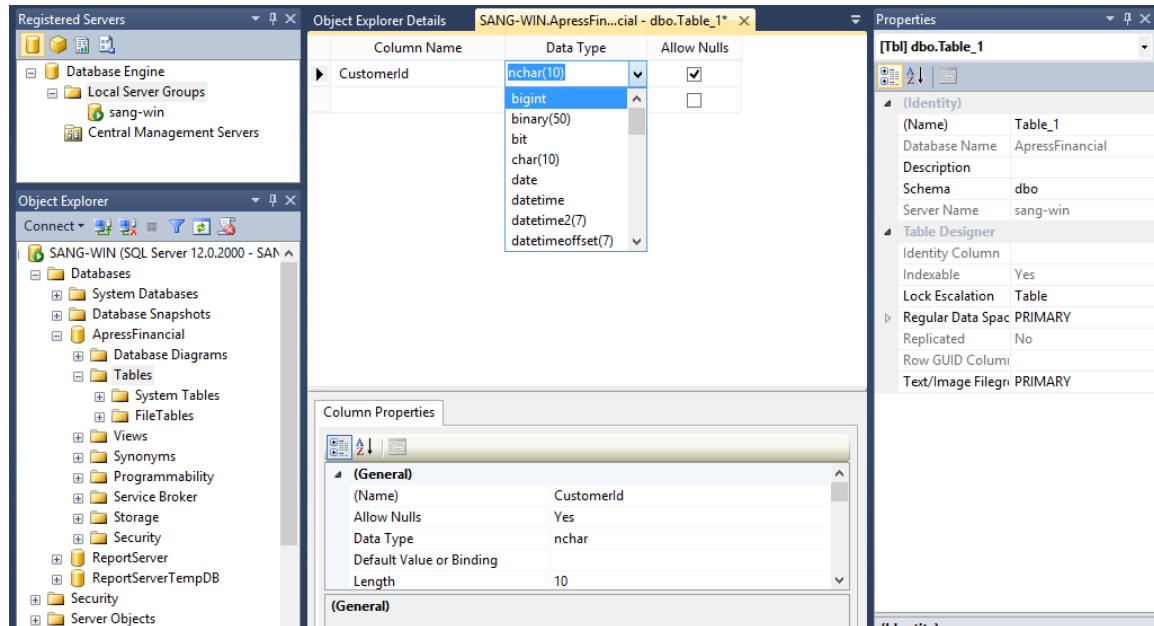
7. The final major item when creating a column within a table is the Allow Nulls check box option. If you don't check the box, some sort of data must be placed in this column. Leaving the check box in the default state will allow **NULL** values in the column, which is not recommended if the data is required (name, order number, etc.). You can also allow NULLs for numeric columns, so instead of needing to enter a zero, you can just skip over that column when it comes to entering the data. In this instance, we want data to be populated within every row, so remove the check mark.

8. The Column **Properties** section for our column will now look like the screen shown in the following figure. Take a moment to peruse this section. We can see the name, whether we are allowing NULLs, and the type of data we are storing. There will be changes to what is displayed depending on the data type chosen.

9. We want this column to be an **identity column**. If you have not already done so, within the Column Properties area expand the Identity Specification node, as we need to set the Is Identity property to Yes. This will set the Identity Increment to 1 and the Identity Seed to 1 as well, as shown in the following figure.
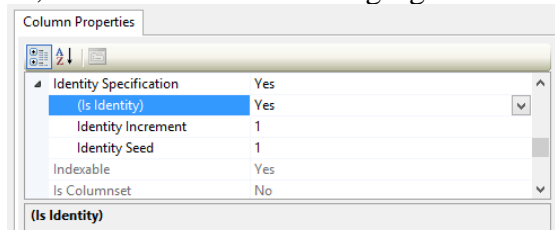
10. It is now possible to add in a few more columns before we get to the next interesting item as in the following figure. Go ahead and do so now. Not everybody will have more than a first name and last name, although some people may have initials. Therefore, we will allow NULL values for any initials they may have. We leave the box checked on the **CustomerOtherInitials** column, as shown in the figure. We also alter the length of this column to 10 characters, which should be more than enough.


11. We can now define our final columns, which you see in the below figure. The last column will record when the account was opened. This can be done by setting the default value of the **DateAdded** column. The default value can be a constant value, the value from a function, or a value bound to a formula defined here. For the moment we will use an SSE function that **returns the current date and time**, **GETDATE()**, as shown in the following figure. Then every time a row is added, it is not necessary for a value to be entered for this column, as SSE will put the date and time in for you.
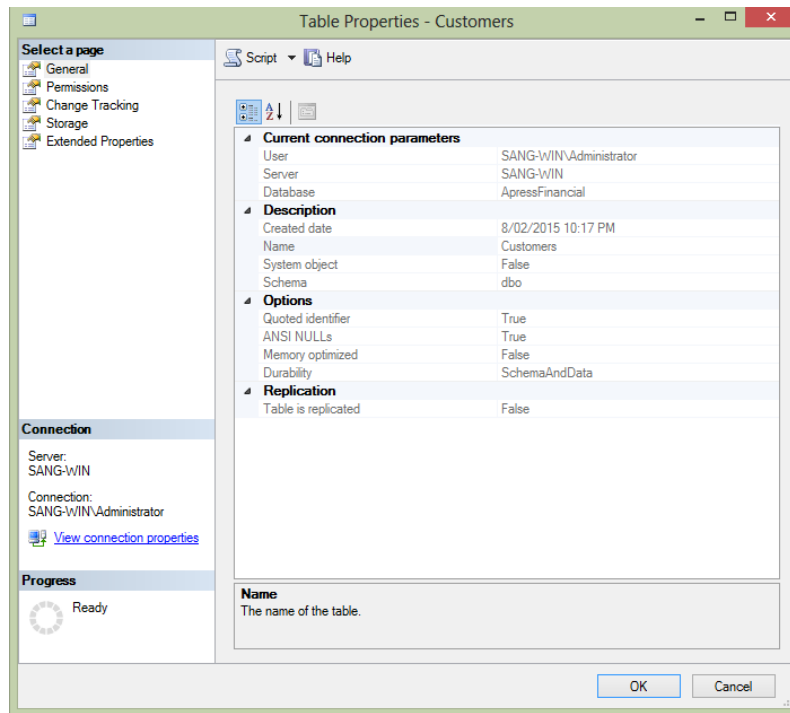

12. Before we save the table, we need to define some properties for it such as the schema owner. On the right, you should see the Table Properties dialog window, as shown in the following figure. If this is not displayed, you can press F4, or from the menu select View → **Properties Window**. First of all, give the **table a name**, **Customers**, and give the table some sort of description. We then move to the schema owner details. When you click the **Schema** combo box, it presents you with a list of possible schemas the table can belong to.
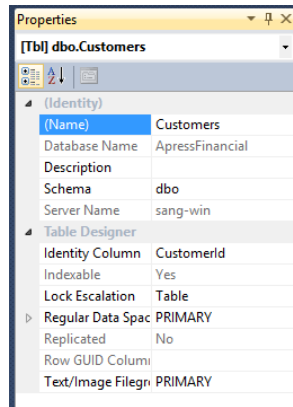
13. Now that we are finished, we can **save** the table either by clicking the Save toolbar button, which sports a floppy disk icon, or by clicking the X (close) button on the Table Designer to close the window, which should bring up the Save dialog box, asking whether we want to save the changes. By clicking Yes, we get a second box asking for the name of the table if we didn't enter a table name in the Table Properties dialog window as shown in the following figure, or the table is saved using the name specified and we are returned to the MS SQL Server main interface.
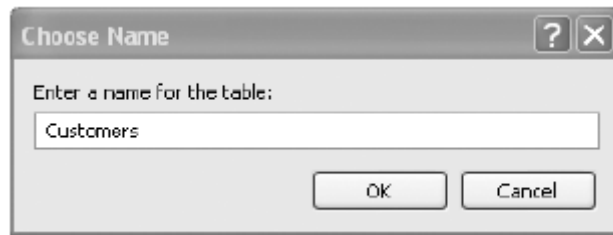
14. If you now right-click the table and select Properties, you can see important details about the table, as shown in the following figure. The first section details who is currently connected. Then we see the date the table was created, its name, and the schema name of the owner of the table.

End of Lab 1