Lab 4 – Database Analysis and Design

Content:

- Building a view in database
- Analyze the topic of assignments
- Design relational database schema
- Edit and install the designed database schema into the Microsoft SQL Server 2014

Duration: 4 teaching periods

Learning outcome:

- How to create and manage data view in database
- The method of analyzing a practical problem
- Database design for a practical problem and managing database diagrams in Microsoft SQL Server 2014

Part 1: Building a View

There will be times when you want to group together data from more than one table, or perhaps only allow users to see specific information from a particular table, where some of the columns may contain sensitive or even irrelevant data. A **view** is a virtual table that, in itself, doesn't contain any data or information. A view can take one or more columns from one or more tables and present this information to a user, without the user accessing the actual underlying tables. A view protects the data layer while allowing access to the data. All of these scenarios can be seen as the basis and reason for building a view rather than another method of data extraction. Because a view represents data as if it were another table, a virtual table in fact, it is also possible to create a view of a view.

Let's take a look at how a view works. As you know, we have a customer table that holds information about our customers such as their first name, last name, account number, and balances. There will be times when you want your users to have access to only the first and last names, but not to the other sensitive data. This is where a view comes into play. You would create a view that returns only a customer's first and last name but no other information.

Building a simple view is a straightforward process and can be completed in SQL Server or a Query Editor pane using T-SQL. Each of these tools has two options to build a view, and this chapter covers all four options so that you become conversant with building a view no matter which tool is currently at hand.

Creating a view can give a user enough information to satisfy a query he or she may have about data within a database without that user having to know any T-SQL commands. A view actually stores the query that creates it, and when you execute the view, the underlying query is the code that is being executed. The underlying code can be as complex as required, therefore leaving the end user with a simple SELECT * command to run with perhaps a small amount of filtering via a simple WHERE statement.

From a view, in addition to retrieving data, you can also modify the data that is being displayed, delete data, and in some situations insert new data. There are several rules and limitations for deleting, modifying, and inserting data from multi-table views.

To summarize, a view is a virtual table created by a stored SQL statement that can span multiple tables. Views can be used as a method of security within your database, and they provide a simpler front end to a user querying the data.

Using Views for Security

Security is always an issue when building your database. So far, different database-provided roles have been introduced, when to use them, how to set up different types of roles, and how useful they are. By restricting all users from accessing or modifying the data in the tables, you will then force everyone to use views and stored procedures to complete any data task.

However, by taking a view on the data and assigning which role can have select access, update access, and so on, you are protecting not only the underlying tables, but also particular columns of data. This is all covered in the discussions involving security in this chapter.

Security encompasses not only the protection of data, but also the protection of your system. At some point as a developer, you will build a view, and then someone else will come along and remove or alter a column from an underlying table that was used in the view. This causes problems; problems; however, this lab will show you how to get around this situation and secure the build of a view so that this sort of thing doesn't happen.

Imagine that you have a table holding specific security-sensitive information alongside general information—an example would be where you perhaps work for the licensing agency for driver licenses and alongside the name and address, there is a column to define the number of fines that have had to be paid. As you can see, this is information that should not be viewed by all employees within the organization. So, what do you do?

The simplest answer is to create a view on the data where you exclude the columns holding the sensitive data. In this way, you can restrict access on the table to the bare minimum of roles or logins, and leave either a view or a stored procedure as the only method of data retrieval allowed. This way, the information returned is restricted to only those columns that a general user is allowed to see.

It is also possible to place a WHERE statement within a view to restrict the rows returned. This could be useful when you don't wish all employee salaries to be listed; perhaps excluding the salaries of the top executives would be advised!

All these methods give you, as a developer, a method for protecting the physical data lying in the base tables behind the views. Combine this with what you learned with roles, and restricting table access, and you can really tighten the security surrounding your data. With more and more companies embracing initiatives like Sarbanes-Oxley, where security should be so tight a company can be defined as having secure data, views are a great method of moving toward this goal.

Another method of securing views is to encrypt the view definition.

Encrypting View Definitions

As well as restricting access to certain tables or columns within a database, views also give the option of encrypting the SQL query that is used to retrieve the data. Once a view is built and you are happy that it is functioning correctly, you would release that view to production; it is at this point that you would add the final area of security—you would encrypt the view.

The most common situation where you will find views encrypted is when the information returned by the view is of a privileged nature. To expand further, not only are you using a view to return specific information, you also don't wish anyone to see how that information was returned, for whatever reason. You would therefore encrypt the SQL code that makes up the view, which would mean that how the information was being returned would not be visible.

There is a downside to encrypting a view: once the process of encryption is completed, it is difficult to get back the details of the view. There are tools on the Internet that can decrypt an encrypted view. When you encrypt a view, the view definition is not processed via encryption algorithms, but is merely obfuscated, in other words, changed so that prying eyes cannot see the code.

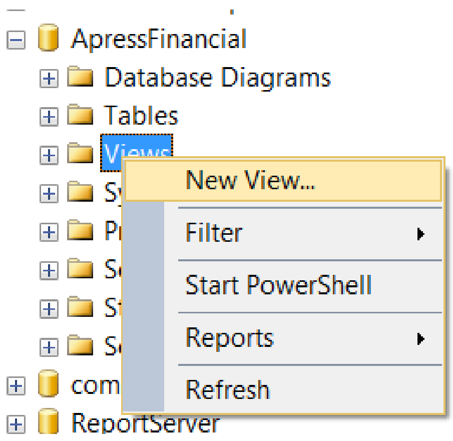
These tools can return the obfuscation back to the original code. Therefore, if you need to modify the view, you will find that it is awkward. Not only would you have to use a tool, but you would have to delete the view and re-create it, as it would not be editable. So, if you build a view and encrypt it, you should make sure that you keep a copy of the source somewhere. This is why it is recommended that encrypted views should be used with care, and really should only be placed in production, or at worst, in user testing.

Always keep a copy of the original view, before encryption, in the company's source control system, for example, SourceSafe, and make sure that regular backups are available.

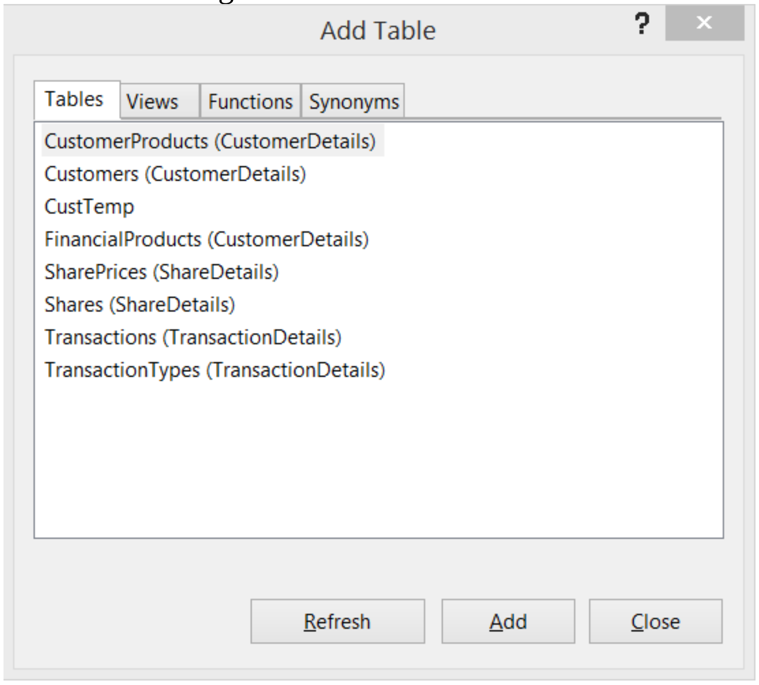
Practice: Creating a View over Interface

1. Ensure that SQL Server is running, its interface is opening, and that the ApressFinancial database is expanded.

2. Find the Views node, and right-click it—this brings up the pop-up menu shown in the following figure; from there select New View.

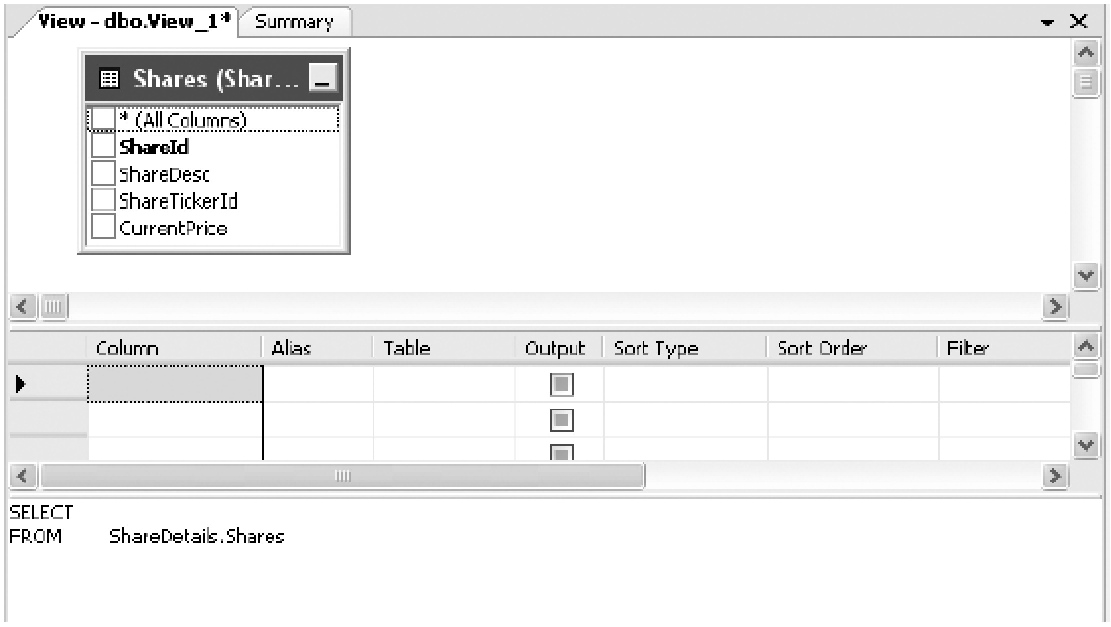


3. The next screen you will see is the View Designer, with a modal dialog box on top presenting a list of tables that you can add to make the view. The background is pretty empty at the moment (move the dialog box around if you need to). It is within the View Designer that you will see all of the information required to build a view. There are no tables in the view at this time, so there is nothing for the View Designer to show. For those of you who are familiar with Access, you will see that the View Designer is similar to the Access Query Designer, only a bit more sophisticated! We want to add our table, so moving back to the modal dialog box, shown in the following figure, select **ShareDetails.Shares** which appears on the screen as Shares (SharesDetails), click Add, and then click Close to remove the dialog box.



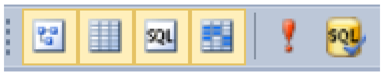
CustomerProducts (CustomerDetails)
Customers (CustomerDetails)
CustTemp
FinancialProducts (CustomerDetails)
SharePrices (ShareDetails)
Shares (ShareDetails)
Transactions (TransactionDetails)
TransactionTypes (TransactionDetails)

4. Take a moment to see how the View Designer has changed, as illustrated in the following figure. Notice that the background Query Designer area has been altered, the ShareDetails.Shares table has been added, and the beginnings of a SELECT statement now appear about two thirds of the way down the screen. By adding a table, the Query Designer is making a start to the view you wish to build.

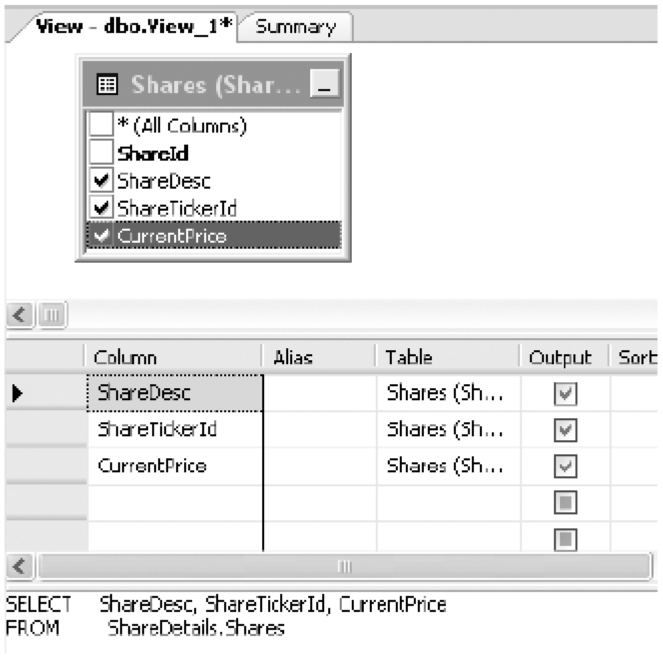


```
SELECT
FROM      ShareDetails.Shares
```

5. There are four separate parts to the View Designer, each of which can be switched on or off for viewing via the toolbar buttons on top. Take a look at these toolbar buttons, as shown close up in the following figure. The first button brings up the top pane, the diagram pane, where you can see the tables involved in the view and can access them via the leftmost toolbar button. The next button accesses the criteria pane, where you can filter the information you want to display. The third button accesses the SQL pane, and the fourth button accesses the results pane. As with Query Editor, here you also have the ability to execute a query through the Execute button (the one with the red exclamation point). The final button relates to verifying the T-SQL. When building the view, although the T-SQL is created as you build up the view, you can alter the T-SQL code, and this button will verify any changes.



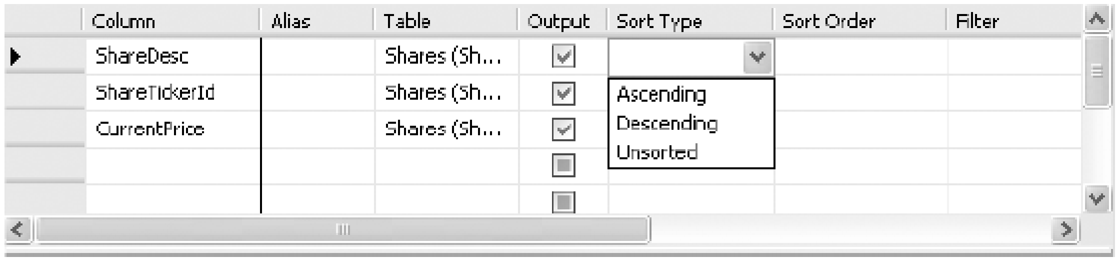
6. We will see the ShareDetails.Shares table listed in the top part of the Query Designer (the diagram pane) with no check marks against any of the column names, indicating that there are not yet any columns within the view. What we want is a view that will display the share description, the stock market ticker ID, and the current price. If we wanted all the columns displayed, we could click the check box next to * (All Columns), but for our example, just place checks against the last three columns, as shown in the following figure. Notice as you check the boxes how the two areas below the table pane alter. The middle grid pane lists all the columns selected and gives you options for sorting and giving the column an alias name. The bottom part is the underlying query of the columns selected. The finished designer will look as shown in the following figure.



Column	Alias	Table	Output	Sort
ShareDesc		Shares (Sh...	✓	
ShareTickerId		Shares (Sh...	✓	
CurrentPrice		Shares (Sh...	✓	

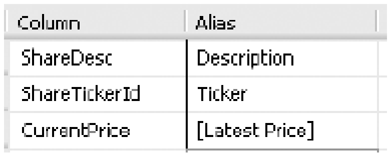
```
SELECT    ShareDesc, ShareTickerId, CurrentPrice
FROM       ShareDetails.Shares
```

7. We are going to change the details in the column grid now to enforce sorting criteria and to give the column aliases. This means that if a user just does SELECT * from the view, he or she will receive the data in the order defined by the view's query and also that some of the column names will have been altered from those of the underlying table. We want to ensure that the shares come out from the view in name order ascending. Move to the Sort Type column and click in the row that corresponds to ShareDesc. Select Ascending as shown in the following figure.



Column	Alias	Table	Output	Sort Type	Sort Order	Filter
ShareDesc		Shares (Sh...	✓			
ShareTickerId		Shares (Sh...	✓	Ascending		
CurrentPrice		Shares (Sh...	✓	Descending		
				Unsorted		

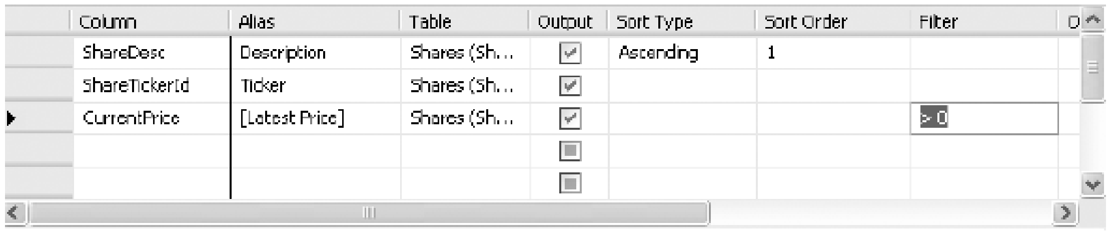
8. In the next column, if we were defining more than one column to sort, we would define the order to sort the columns in. However, we still need to add the aliases, which are found in the second column of the grid. Notice the third option, CurrentPrice. To make this column more user friendly, we make the name Latest Price, with a space. When we type this and tab out of the column, it becomes [Latest Price], as you see in the following figure; SQL Server places the square brackets around the name for us because of the space.



Column	Alias
ShareDesc	Description
ShareTickerId	Ticker
CurrentPrice	[Latest Price]

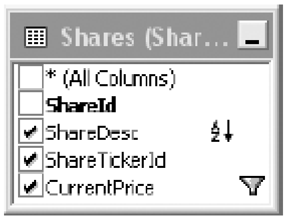
9. Scrolling to the right of the screen would allow us to define a filter for the view as well. This is ideal if we want to restrict what a user can see. Although sort orders can be changed by the T-SQL

that calls the view, filters placed within the view cannot return more data than the view allows. So going back to our salary example mentioned earlier, this would be where we would restrict users to not seeing the MD’s salary. In our example, we will only list those shares that have a current price, in other words where CurrentPrice is greater than 0, as shown in the following figure.



Column	Alias	Table	Output	Sort Type	Sort Order	Filter
ShareDesc	Description	Shares (Sh...	✓	Ascending	1	
ShareTickerId	Ticker	Shares (Sh...	✓			
CurrentPrice	[Latest Price]	Shares (Sh...	✓			> 0

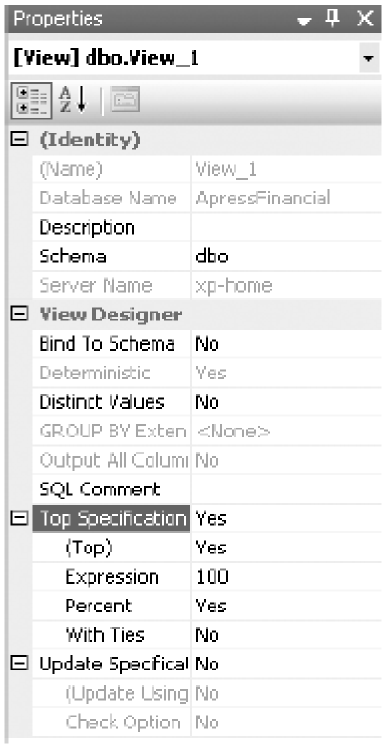
10. Notice the Query Editor pane, which now has the filter within it as well as the sorting order. Also take a look at the diagram pane and how the table display has been altered, as you see in the following figure.



11. Moving back to the T-SQL in the SQL pane, what about the TOP (100) PERCENT clause? Where did that come from? First of all, if you specify an order in a view, by default SQL Server will place the TOP (100) PERCENT clause within the SQL. It can be used if the table is very large and you don’t want to allow users to return all the data on a production system, as it would tie up resources. You can also remove that clause from the
Query Editor pane if you want; this will unlink your query from the designer and the Properties window, but you would also need to remove the ORDER BY. A final point to notice is how the column aliases are defined. The physical column is named followed by AS and then the alias.

```
SELECT TOP (100) PERCENT
    ShareDesc AS Description,
    ShareTickerId AS Ticker,
    CurrentPrice AS [Latest Price]
FROM ShareDetails.Shares
WHERE (CurrentPrice > 0)
ORDER BY Description
```

12. If you wish to remove the TOP clause, it would be better to do this within the Properties window, shown in the following figure, usually found on the bottom right of SSMS; however, you would also need to remove the sorting.
If it’s not there, it can be found by selecting View → Properties Window from the menu or by pressing F4. Within the properties, we can give the view a description—very useful—but we can also remove the TOP clause by setting Top Specification to No. We can also define whether this view is read-only by setting Update Specification to No.

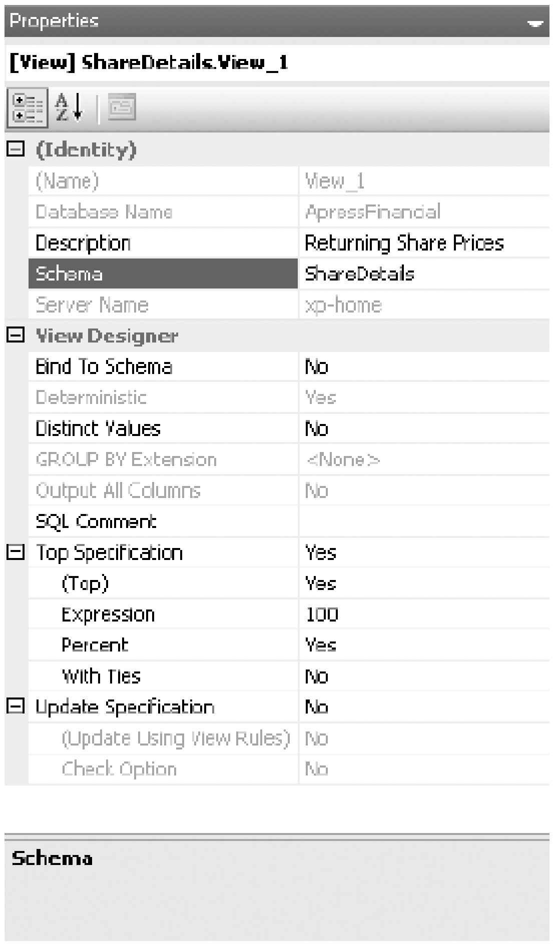


Properties

[View] dbo.View_1

(Identity)	
(Name)	View_1
Database Name	ApressFinancial
Description	
Schema	dbo
Server Name	xp-home
View Designer	
Bind To Schema	No
Deterministic	Yes
Distinct Values	No
GROUP BY Exten	<None>
Output All Columi	No
SQL Comment	
Top Specification	Yes
(Top)	Yes
Expression	100
Percent	Yes
With Ties	No
Update Specifical	No
(Update Using	No
Check Option	No

13. We do need to change some of the properties in the view definition, as shown in the following figure. First of all, it is better to give the view a description. Also, like a table, a view should belong to a schema. This can be from an existing schema, or if you have a view traversing more than one table, you may have a schema to cater to that scenario. In our case, it fits into the ShareDetails schema.


Properties

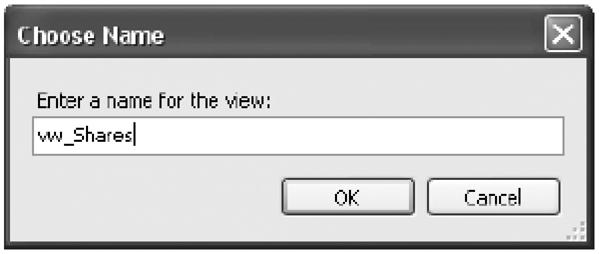
[View] ShareDetails.View_1

(Identity)	
(Name)	View_1
Database Name	ApressFinancial
Description	Returning Share Prices
Schema	ShareDetails
Server Name	xp-home
View Designer	
Bind To Schema	No
Deterministic	Yes
Distinct Values	No
GROUP BY Extension	<None>
Output All Columns	No
SQL Comment	
Top Specification	Yes
(Top)	Yes
Expression	100
Percent	Yes
With Ties	No
Update Specification	No
(Update Using View Rules)	No
Check Option	No

Schema

14. We think the view is complete, but we need to test it out. By executing the query with the Execute button (the one sporting the red exclamation point), we will see the results in the results pane.

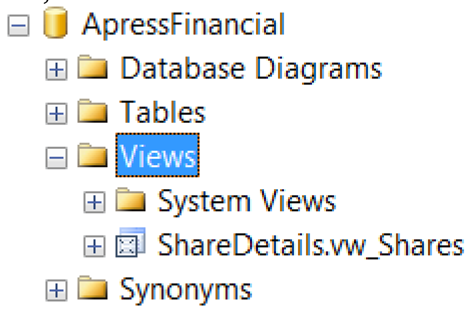
15. Now that the view is complete, it is time to save it to the database. Clicking the close button will bring up a dialog box asking whether you want to save the view. Click Yes to bring up a dialog box in which you give the view a name. You may find while starting out that there is a benefit to prefixing the name of the view with something like vw_ so that you know when looking at the object that it's a view. Many organizations do use this naming standard; however, it is not compulsory, and SQL Server makes it clear what each object is. The naming standard comes from a time when tools did not make it clear what object belonged to which group of object types. Once you have the name you wish, as shown in the following figure, click OK.


Choose Name

Enter a name for the view:

vw_Shares

16. This will bring us back to SSMS, where we will see the view saved.



CREATE VIEW Syntax

Very quickly you will find that creating a view using T-SQL is the better way forward. It is just as fast as building a view using the designer.

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] ) ]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement [ ; ]
[ WITH CHECK OPTION ]
<view_attribute> ::= {[ ENCRYPTION ][ SCHEMABINDING ]
                     [ VIEW_METADATA ]}
```

The basic CREATE VIEW syntax is very simple and straightforward. The following syntax is the most basic syntax of the CREATE VIEW statement and is the one used most often.

```
CREATE VIEW [database_name.][schema_name.]view_name
WITH {ENCRYPTION | SCHEMABINDING}
AS
SELECT_statement
```

Taking a look at the first section of the syntax, notice that the name of the view can be prefixed with the name of the schema and the name of the database to which it belongs; however, the database name and the schema are optional. Providing that we are in the correct database and are logged in with the ID we wish to create the view for, the database_name and schema_name options are not required, as they will be assumed from the connection details. For production views, rather than views used purely by a single SQL Server user, it is recommended that they be built by the database owner. If the view is built by a non-database owner, then when someone tries to execute the view, that user will need to prefix the name of the view with the login of the person who created it.

Following on from these options, we build the query, typically formed with a SELECT statement that makes up the view itself. As you saw in the previous example, the SELECT statement can cover one or many tables or views, many columns, and as many filtering options using the WHERE statement as you wish. You can also order the data in a view; however, if you recall from our earlier example, to place an ORDER BY clause on a SELECT statement within a view, it is necessary to use the TOP statement. We specified the TOP 100 PERCENT in our first example to get around this problem. Failure to do so will result in an error, and the view will not be created. We also cannot reference any temporary variable or temporary table within a view, or create a new table from a view by using the INTO clause. To clarify, it is not possible to have a SELECT column INTO newtable.

The ENCRYPTION option will take the view created and encrypt the schema contained so that the view is secure and no one can see the underlying code or modify the contents of the SELECT statement within. However (I know I keep repeating this, but it is so important), do keep a backup of the contents of the view in a safe place in development in case any modifications are required.

The SCHEMABINDING option ensures that any column referenced within the view cannot be dropped from the underlying table without dropping the view built with SCHEMABINDING first. This, therefore, keeps the view secure with the knowledge that there will be no run-time errors when columns have been altered or dropped from the underlying table, and the view is not altered in line with those changes. If you try to remove a column from the table that is contained within a schema bound view, for example, you will receive an error. There is one knock-on effect when using SCHEMABINDING: all tables or other views named within the SELECT statement must be prefixed with the name of the schema of the table or view, even if the owner of these objects is the same as the schema of the view.

Go back to the two options that will be used less often, the first being WITH CHECK OPTION. If the view is being used as the basis of completing updates to the underlying table, any modification call, such as UPDATE/DELETE/INSERT, will still make the data visible through the view.

The final possible option, VIEW_METADATA, exposes the view’s metadata if you are calling the view via ODBC, OLE DB, etc.—in other words, from a program that is external to SQL Server.

Practice: Creating a View by T-SQL

1. Ensure that SQL Server is running, its interface is opening and that there is an empty Query Editor pane. First of all, let's get the T-SQL correct. We need to link in three tables, the CustomerDetails.Customers table to get the name and address, the TransactionDetails.Transactions table so we can get a list of transactions for the customer, and finally the TransactionDetails.TransactionTypes table so that each transaction type has its full description. The code is as follows:

```
SELECT c.AccountNumber,c.CustomerFirstName,c.CustomerOtherInitials,
    tt.TransactionDescription,t.DateEntered,t.Amount,t.ReferenceDetails
FROM CustomerDetails.Customers c
JOIN TransactionDetails.Transactions t ON t.CustomerId = c.CustomerId
JOIN TransactionDetails.TransactionTypes tt ON
    tt.TransactionTypeId = t.TransactionType
ORDER BY c.AccountNumber ASC, t.DateEntered DESC
```

2. Once done, execute the code by pressing F5 or Ctrl+E, or clicking the Execute button.

3. We can now wrap the CREATE VIEW statement around our code. Execute this code to store the view in the ApressFinancial database. As there is an ORDER BY clause, we need to add to the query a TOP statement, so we have TOP 100 Percent.

```
CREATE VIEW CustomerDetails.vw_CustTrans
AS
SELECT TOP 100 PERCENT
    c.AccountNumber,c.CustomerFirstName,c.CustomerOtherInitials,
    tt.TransactionDescription,t.DateEntered,t.Amount,t.ReferenceDetails
FROM CustomerDetails.Customers c
JOIN TransactionDetails.Transactions t ON t.CustomerId = c.CustomerId
JOIN TransactionDetails.TransactionTypes tt ON
    tt.TransactionTypeId = t.TransactionType
ORDER BY c.AccountNumber ASC, t.DateEntered DESC
```

This view is a straightforward view with no ENCRYPTION or SCHEMABINDING options. The one complication within the view concerns the ORDER BY clause: one of the stipulations for this view is that it returns the data of financial transactions with the most recent transaction first. Therefore, an ORDER BY statement is required on the DateEntered column to return the rows in descending order. To avoid receiving an error message when building the view, it has been necessary to place a TOP option within the SELECT statement; in the case of the example, a TOP 100 PERCENT statement has been chosen so that all the rows are returned. The remainder of the SELECT statement syntax is very straightforward.

Practice: Creating a View with option SCHEMABINDING

The following example will bind the columns used in the view to the actual tables that lie behind the view, so that if any column contained within the view is modified, an error message will be displayed and the changes will be canceled. The error received will be shown so that we can see for ourselves what happens.

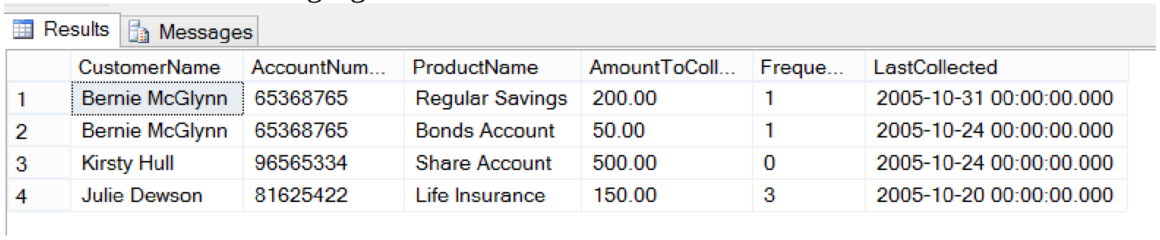
1. Create a new Query Editor pane and connect it to the ApressFinancial database.We can then create the T-SQL that will form the basis of our view.

```
SELECT c.CustomerFirstName + ' ' + c.CustomerLastName AS CustomerName,
c.AccountNumber, fp.ProductName, cp.AmountToCollect, cp.Frequency,
cp.LastCollected
FROM CustomerDetails.Customers c
JOIN CustomerDetails.CustomerProducts cp ON cp.CustomerId = c.CustomerId
JOIN CustomerDetails.FinancialProducts fp ON
fp.ProductId = cp.FinancialProductId
```

2. We need some test data within the system to test this out. This is detailed in the following code. Enter this code and execute it.

```
INSERT INTO CustomerDetails.FinancialProducts (ProductId,ProductName)
VALUES (1,'Regular Savings')
INSERT INTO CustomerDetails.FinancialProducts (ProductId,ProductName)
VALUES (2,'Bonds Account')
INSERT INTO CustomerDetails.FinancialProducts (ProductId,ProductName)
VALUES (3,'Share Account')
INSERT INTO CustomerDetails.FinancialProducts (ProductId,ProductName)
VALUES (4,'Life Insurance')
INSERT INTO CustomerDetails.CustomerProducts
(CustomerFinancialProductId, CustomerId,FinancialProductId,
AmountToCollect,Frequency,LastCollected,LastCollection,Renewable)
VALUES (1, 1,1,200,1,'31 October 2005','31 October 2025',0)
INSERT INTO CustomerDetails.CustomerProducts
(CustomerFinancialProductId, CustomerId,FinancialProductId,
AmountToCollect,Frequency,LastCollected,LastCollection,Renewable)
VALUES (2, 1,2,50,1,'24 October 2005','24 March 2008',0)
INSERT INTO CustomerDetails.CustomerProducts
(CustomerFinancialProductId, CustomerId,FinancialProductId,
AmountToCollect,Frequency,LastCollected,LastCollection,Renewable)
VALUES (3, 2,4,150,3,'20 October 2005','20 October 2005',1)
INSERT INTO CustomerDetails.CustomerProducts
(CustomerFinancialProductId, CustomerId,FinancialProductId,
AmountToCollect,Frequency,LastCollected,LastCollection,Renewable)
VALUES (4,3,3,500,0,'24 October 2005','24 October 2005',0)
```

3. Test out that the T-SQL works as required by executing it. The results you get returned should look similar to the following figure:



Results | Messages

	CustomerName	AccountNum...	ProductName	AmountToColl...	Freque...	LastCollected
1	Bernie McGlynn	65368765	Regular Savings	200.00	1	2005-10-31 00:00:00.000
2	Bernie McGlynn	65368765	Bonds Account	50.00	1	2005-10-24 00:00:00.000
3	Kirsty Hull	96565334	Share Account	500.00	0	2005-10-24 00:00:00.000
4	Julie Dewson	81625422	Life Insurance	150.00	3	2005-10-20 00:00:00.000

4. We now need to create the CREATE VIEW. First of all, we are completing a test to see whether the view already exists within the system catalogs. If it does, we DROP it. Then we define the view using the WITH SCHEMABINDING clause. The other change to the T-SQL is to prefix the tables we are using with the schema that the tables come from. This is to ensure that the SCHEMABINDING is successful and can regulate when a column is dropped.

```
IF EXISTS(SELECT TABLE_NAME FROM INFORMATION_SCHEMA.VIEWS
WHERE TABLE_NAME = N'vw_CustFinProducts'
AND TABLE_SCHEMA = N'CustomerDetails')
DROP VIEW CustomerDetails.vw_CustFinProducts
GO
CREATE VIEW CustomerDetails.vw_CustFinProducts WITH SCHEMABINDING
AS
SELECT c.CustomerFirstName + ' ' + c.CustomerLastName AS CustomerName,
c.AccountNumber, fp.ProductName, cp.AmountToCollect, cp.Frequency,
cp.LastCollected
FROM CustomerDetails.Customers c
JOIN CustomerDetails.CustomerProducts cp ON cp.CustomerId = c.CustomerId
JOIN CustomerDetails.FinancialProducts fp ON
fp.ProductId = cp.FinancialProductId
```

5. Once done, execute the code by pressing F5 or Ctrl+E, or clicking the Execute button. You should then see the following message:

```
The command(s) completed successfully.
```

6. Now that our vw_CustFinProducts view is created, which we can check by looking in Object Explorer, it is possible to demonstrate what happens if we try to alter a column used in the view and so affect one of the underlying tables. Enter the following code, and then execute it:

```
ALTER TABLE CustomerDetails.Customers
ALTER COLUMN CustomerFirstName nvarchar(100)
```

7. You will then see in the results pane two error messages: the first shows that an alteration has been attempted on the CustomerDetails.Customers table and has been disallowed and names the view stopping this, and the second shows that the alteration failed.

```
Msg 5074, Level 16, State 1, Line 1
```

```
The object 'vw_CustFinProducts' is dependent on column 'CustomerFirstName'.
Msg 4922, Level 16, State 9, Line 1
ALTER TABLE ALTER COLUMN CustomerFirstName failed because one or more
objects access this column.
```

Part 2: Diagramming the Database

Now that the database has been built, the tables have been created, the indexes have been inserted, and relationships link some of the tables, it's time to start documenting. To help with this, SQL Server offers us the database diagram tool, which is the topic of this section.

One of the worst things about database documentation is documenting tables and showing how they relate to one another in a diagram. The database diagram tool can do all of this very quickly and simply, with one caveat: if more than one person is using the database diagram tool on the same database, and there are two sets of changes to be applied to the same table, the person who saves his or her changes last will be the person who creates the final table layout. In other words, the people who save before the last person will lose their changes.

As you developed tables within your database, hopefully you will have commented the columns and tables as you have gone along to say what each column and table is. This is a major part of documentation anyway, and providing you comment columns and tables at the start, it will be less of a chore to add in further comments when you add new columns. If you do have comments on each of your columns within a table, this will help overall with the documentation shown within the diagram.

SQL Server's database diagram feature is more than just a documentation aid. This tool provides us with the ability to develop and maintain database solutions. It is perhaps not always the quickest method of building a solution, but it is one that allows the entire solution to be completed in one place. Alternatively, you can use it to build up sections of a database into separate diagrams, breaking the whole solution into more manageable parts, rather than switching between nodes in SQL Server.

Database Diagramming Basics

So far, with the creation of databases, tables, indexes, and relationships, as much documentation as SQL Server will allow has so far been maintained. However, there is no documentation demonstrating how the tables relate to one another within the database. This is where the database diagram comes in.

A database diagram is a useful and easy tool to build simple but effective documentation on these aspects. You build the diagram yourself, and you control what you want to see within the diagram. When you get to a large database solution, you may want diagrams for sections of the database that deal with specific aspects of the system, or perhaps you want to build a diagram showing information about process flows. Although there are other external tools to do this, none is built into SSE that can allow diagrams to be kept instantly up to date.

A diagram will only show tables, columns within those tables, and the relationships between tables in a bare form. You will also see a yellow “key,” which denotes a primary key on the table where one has been defined, but that is all the information displayed. It is possible to define the information that is to be displayed about the columns in the table, whether it is just the column name or more in-depth information, such as a column’s data type and length, comments, and so on. However, to display more than just the bare essentials, a little bit of work is required.

Although the diagram shows the physical and logical attributes of the database that is being built or has been built, it also allows developers and administrators to see exactly what is included with the database at a glance and how the database fits together.

The SQL Server Database Diagram Tool

SQL Server’s database diagram tool aids in the building of diagrams that detail aspects of the database that a developer wishes to see. Although it is a simple and straightforward tool, and it’s not as powerful as some other tools on the market for building database diagrams, it is perfect for SQL Server.

For example, one of the market leaders in database design tools is a product called ERwin, currently owned by Computer Associates. ERwin is a powerful database utility that not only builds diagrams of databases, but also provides data dictionary language output, which can be used to build database solutions. Through links such as OLE DB data providers, these tools can interact directly with databases and so can be used as a front end for creating databases. They can also, at the same time, keep the created source in alignment and under control from a change control perspective, not only ensuring that the code exists within the database, but also issuing a command to create a new database quickly, if necessary. An example of where this might be useful is when you’re creating a new test database. If you want to go further than the SQL Server’s database diagram tool provides, you should be looking at more powerful tools, which can cost a great deal of money.

SQL Server’s database diagram utility offers more than just the ability to create diagrams. As mentioned earlier, it can also be used as a front end for building database solutions. Through this utility, SQL Server allows you to add and modify tables, build relationships, add indexes, and do much more. Any changes built in the tool are held in memory until they are committed using a save command within the tool. However, there are limitations to its overall usefulness.

First of all, the biggest restriction of any diagram-based database tool comes down to the amount of screen space available to view the diagram. As soon as your database solution consists of more than a handful of tables, you will find yourself scrolling around the diagram, trying to find the table you are looking for.

Second, you cannot add stored procedures, schemas, users, views, or any object that is not a table. Other products allow you to include these objects, or they may even build some of them for you.

Finally, for the moment, when altering any of the information you can change within this tool, you are usually using the same dialog boxes and screens as you would in SQL Server.

As you will see as you go through the chapter, the database diagram tool is quite powerful in what it can achieve, but there are some areas of concern that you have to be aware of when working with diagrams. Keep in mind that the database diagram tool is holding all the changes in memory until you actually save the diagram.

For example, if you have a database diagram open, and a table within that diagram is deleted outside of the diagram, perhaps in Query Editor or SQL Server by yourself or another valid user ID, one of two things can happen. First, if you have unsaved changes to the deleted table, saving your diagram will re-create the table, but don't forget that through the earlier deletion all the data will be removed. If, however, you have no changes pending to that table, the table will not be re-created. When you come to reopen the diagram, the table will have been removed.

With several developers working on the database at once, any changes made from the diagramming tool of your SQL Server will not be reflected in any other developer's diagram until their changes are saved and their diagrams are refreshed. If you have multiple diagrams open, and you alter a table and insert or remove a column, this will reflect immediately in all the open diagrams within your own SQL Server only. Don't forget this is an in-memory process, so this process can't reflect on anyone else's diagrams until the changes are saved and the diagrams are refreshed.

Also, if you remove an object in your diagram, when you then save the diagram, the object will be removed and any changes completed by others will be lost. Effectively, the last person who closes his or her diagram wins!

To summarize, if you use the database diagram tool, use it with care. Because many of the processes are in memory, you could inadvertently cause problems.

The Default Database Diagram

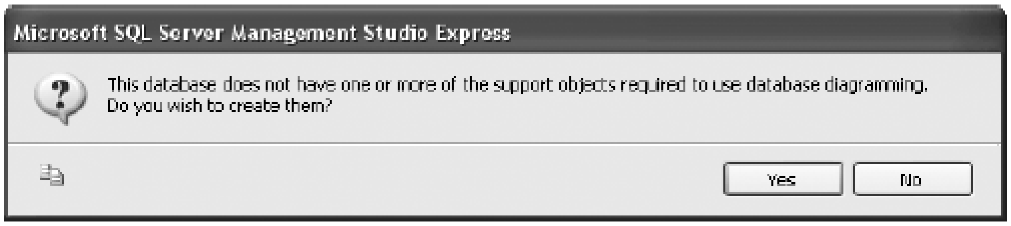
Although it's not mandatory, it is necessary that every SQL Server database solution should have a default database diagram built into it so that any developer—new or experienced—can instantly see how the database being inspected fits together.

A default database diagram should include every table and every relationship that is held for that database. Unlike other diagrams that may take a more sectionalized view of things, the default database diagram should be all-encompassing.

As mentioned earlier, it is imperative that you keep this diagram up to date. You will notice this statement repeated a few times in this chapter. Don't use the default diagram as the source of development for your database solution. The default diagram includes all the tables, which means that if you're using the database diagram tool for development, you are potentially logically locking out all other users from touching any other table as part of their development, in case their changes are lost. Only update the diagram with tables and relationships once they have been inserted in the database.

Practice: Creating a Database Diagram

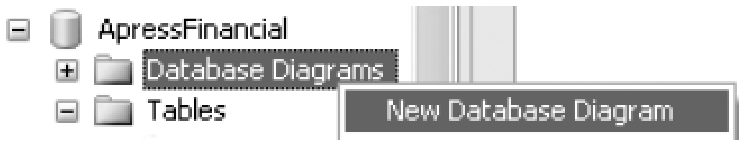
1. Ensure that SQL Server is running, its management environment is opening and that the ApressFinancial database is expanded so that you see the Database Diagrams and Tables nodes. Select the Database Diagrams node. If this is the first diagram you are creating for the database, you'll need to install support objects. Without them, you cannot create the diagram, so click Yes at the next dialog box prompt:



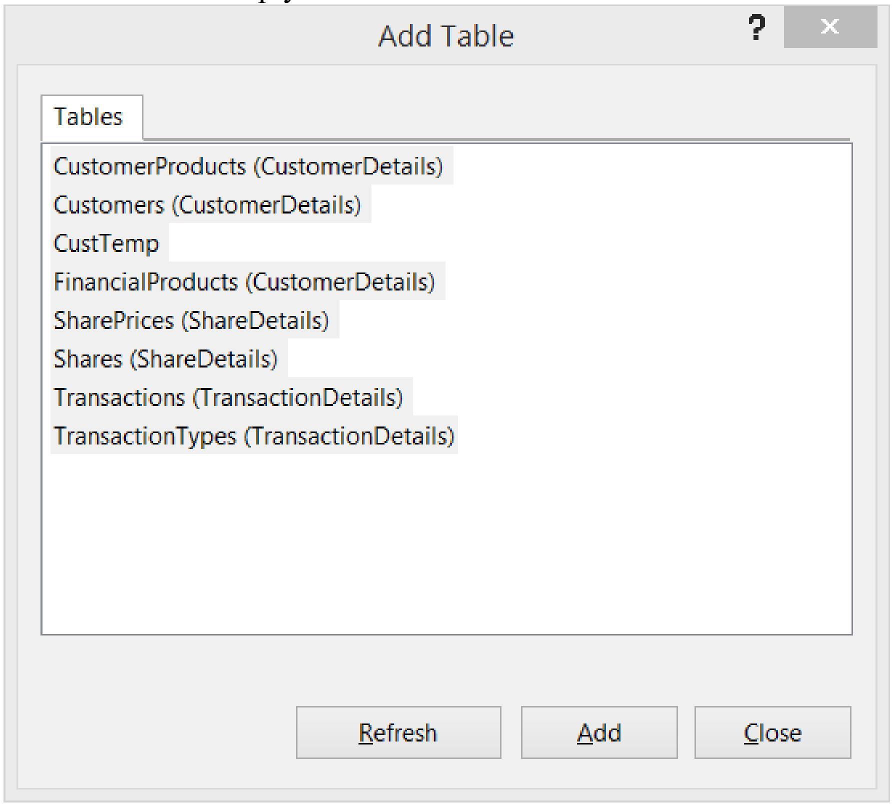
Microsoft SQL Server Management Studio Express

This database does not have one or more of the support objects required to use database diagramming. Do you wish to create them?

2. Now right-click and select New Database Diagram:



3. The first screen you'll see when creating the diagram is the Add Table dialog box (see the following figure). Select all of the tables listed, as you want to use all the tables in your diagram, and then click Add. This will "empty" the screen. Click Close.

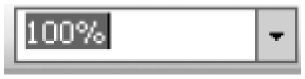


Add Table

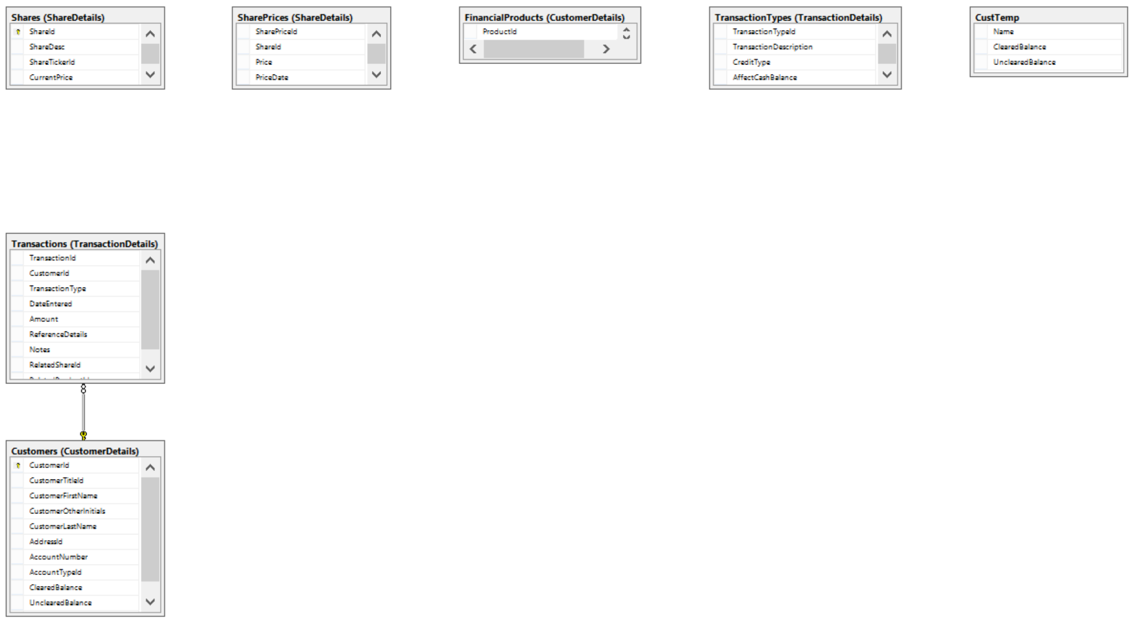
Tables

CustomerProducts (CustomerDetails)
Customers (CustomerDetails)
CustTemp
FinancialProducts (CustomerDetails)
SharePrices (ShareDetails)
Shares (ShareDetails)
Transactions (TransactionDetails)
TransactionTypes (TransactionDetails)

4. After a few moments, you will be returned to SQL management environment, but with the database diagram now built. The diagram will not show all the tables at this point and will be very large. You can reduce the size through the Size combo box in the diagram toolbar, as shown in the following figure:



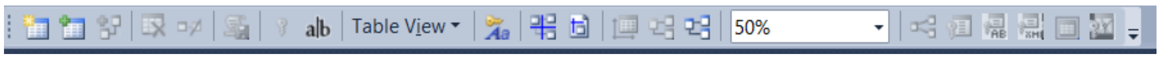
5. You’ll then see a diagram similar to that shown in the following figure (Don’t be surprised if the layout is different, though)



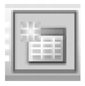
That’s all there is to building a basic diagram.

The Database Diagram Toolbar

Let’s next take a look at the toolbar and how each of the buttons works within the diagram. The whole toolbar is:



The first button is the New Table button, as shown in the following image. You click this button to create a new table within the Database Designer. The difference is that you need to use the Properties window for each column rather than having the properties at the bottom of the Table Designer.



When building the diagram, you selected every table. If you hadn't done so and you wanted to include a table added since you created the diagram, clicking the Add Table button would bring up the Add Table dialog box shown earlier to add in any missing tables.

The Add Related Tables button, shown next, adds tables related to the selected table in the designer.

It is also possible to delete a table from a database from the designer using this button.

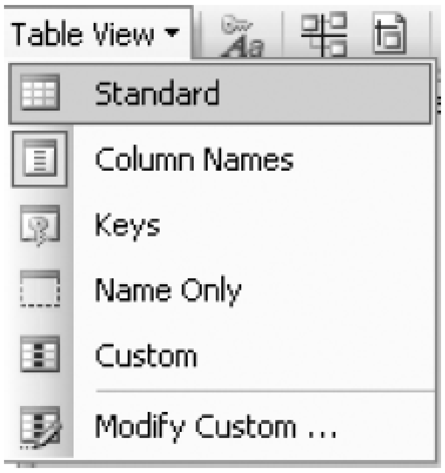
If you just wish to remove the database from the diagram rather than the database, you can use the next button to accomplish this. You would use this button, for example, if a table no longer formed part of the "view" the database diagram was built for.

Any changes made to the database within the designer can be saved as a script. Use the following Generate Change Script button to accomplish this.

If you wish to set a column to be the primary key, select the relevant column within a table and click the Set Primary Key button (shown next).

It is possible to create a place within the diagram to put ad hoc text. Use the following New Text Annotation button to do this.

Each table displayed is set to a standard layout. It is possible to change this to a different predefined layout or to create your own custom view. The following Table View drop-down list enables you to change the layout or create your own custom version.



Relationships that exist between tables will, by default, show as a line. However, it is possible to show the name of the relationship as a label by clicking the following button.

Diagrams are ideal methods of documenting the database. Diagrams can be printed for meetings, discussions about future development, and so on. The following button shows the line breaks in pages that will be printed.

Page breaks in diagrams remain as they were first set up until they are recalculated. You are able to view the page breaks, arrange your tables accordingly, and then recalculate the page breaks based on the new layout. Clicking the following button will do this recalculation for you.

Tables can be expanded or shrunk manually, but when you select one or more tables using the Ctrl button, click the relevant tables, and then click the following button, you can resize the tables to a uniform size.

It is possible, by clicking the following button, to rearrange tables that have been selected and let SQL Server make the arrangement choices.

It is possible to rearrange the tables shown in the diagram. SSE will do its best to get the best layout for the tables and relationships when you click the following button.

In a previous lab, you saw how to build a relationship between two tables. The button you use to do this appears next. This button will bring up the same dialog box as you saw in that chapter.

You can also manage the indexes and keys using the dialog box by clicking the following button.

It is possible to create an index on a table or column called a full-text index. This index allows for searching on text data a bit like Google. For example, Google would hold the web pages within a text data type, and using a full-text index would allow for full searching on that page, clicking the following button will manage these indexes.

If you have indexes placed on any XML data types, clicking the next button will allow you to manage these indexes.

Previously, you learned how to build constraints for tables. Clicking the following Manage Check Constraints button brings up the same dialog box you saw then.

End of Lab 4