MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 8: Implementing Views

Contents

Overview	1
Introduction to Views	2
Advantages of Views	3
Defining Views	5
Modifying Data Through Views	15
Optimizing Performance by Using Views	16
Recommended Practices	22
Lab A: Implementing Views	24
Review	33

Microsoft

Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2000 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, MS-DOS, PowerPoint, Visual Basic, Visual C++, Visual Studio, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Project Lead: Rich Rose
Instructional Designers: Rich Rose, Cheryl Hoople, Marilyn McGill
Instructional Software Design Engineers: Karl Dehmer, Carl Raebler, Rick Byham
Technical Lead: Karl Dehmer
Subject Matter Experts: Karl Dehmer, Carl Raebler, Rick Byham
Graphic Artist: Kirsten Larson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Copy Editor: Edward McKillop (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Support: Lori Walker (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: TestingTesting123
Classroom Automation: Lorrin Smith-Bates
Creative Director, Media/Sim Services: David Mahlmann
Web Development Lead: Lisa Pease
CD Build Specialist: Julie Challenger
Online Support: David Myka (S&T Consulting)
Localization Manager: Rick Terek
Operations Coordinator: John Williams
Manufacturing Support: Laura King; Kathy Hershey
Lead Product Manager, Release Management: Bo Galford
Lead Product Manager, Data Base: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
45 Minutes

Lab:
30 Minutes

This module provides students with the information needed to create and use views.Views provide the ability to store a predefined query as an object in the database for later use. They offer a convenient way to hide sensitive data or the complexities of a database design and to provide information without requiring the user to write or execute Transact-SQL statements.

The module defines views and their advantages. The module then describes creating views and provides examples of projections and joins. These examples illustrate how to include computed columns and built-in functions in the view definitions. The module then covers restrictions on modifying data through views. The last section discusses how views can improve performance.

In the lab, students will create and test views, including views with encrypted definitions. Students will also make changes to source tables through a view and look at view definitions.

After completing this module, students will be able to:

- Describe the concept of a view.
- List the advantages of views.
- Define a view by using the CREATE VIEW statement.
- Modify data through views.
- Optimize performance by using views.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- The Microsoft® PowerPoint® file 2073A_08.ppt
- The C:\Moc\2073A\Demo\D08_Ex.sql example file, which contains all of the example scripts from the module, unless otherwise noted in the module.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the lab.

Module Strategy

Use the following strategy to present this module:

- Introduction to Views

  Introduce the concept of views. Mention that views are simply stored queries.

- Advantages of Views

  List the advantages of using views.

- Defining Views

  Discuss how users create, alter, and drop views, meanwhile covering the restrictions and guidelines that users must consider. Describe how users can encrypt the view definition. List the system tables that contain the view definition information.

- Modifying Data Through Views

  Describe how to modify data by using views and list the considerations and restrictions that users must heed.

- Optimizing Performance by Using Views

  Describe how views allow you to optimize performance by storing results of complex queries and partitioning data.

Customization Information

This section identifies the lab setup requirements for a module and the configuration changes that occur on student computers during the labs. This information is provided to assist you in replicating or customizing Microsoft Official Curriculum (MOC) courseware.

Important The lab in this module is dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Classroom Setup Guide* for course 2073A, *Programming a Microsoft SQL Server 2000 Database*.

Lab Setup

The following section describes the setup requirement for the lab in this module.

Setup Requirement

The lab in this module requires the **credit** database to be in a state required for this lab. To prepare student computers to meet this requirement, perform one of the following actions:

- Complete the prior lab
- Execute the C:\Moc\2073A\Batches\Restore08.cmd batch file.

Warning If this course has been customized, students must execute the C:\Moc\2073A\Batches\Restore08.cmd batch file to ensure that the lab will function properly.

Lab Results

There are no configuration changes on student computers that affect replication or customization.

Overview

Topic Objective
To provide an overview of the module topics and objectives.

Lead-in
In this module, you will learn how to create, use, and maintain data views.

- **Introduction to Views**
- **Advantages of Views**
- **Defining Views**
- **Modifying Data Through Views**
- **Optimizing Performance by Using Views**

**********ILLEGAL FOR NON-TRAINER USE**********

This module defines views and their advantages. The module then describes creating views and provides examples of projections and joins. These examples illustrate how to include computed columns and built-in functions in view definitions. The module then covers restrictions on modifying data through views. The last section discusses how views can improve performance.

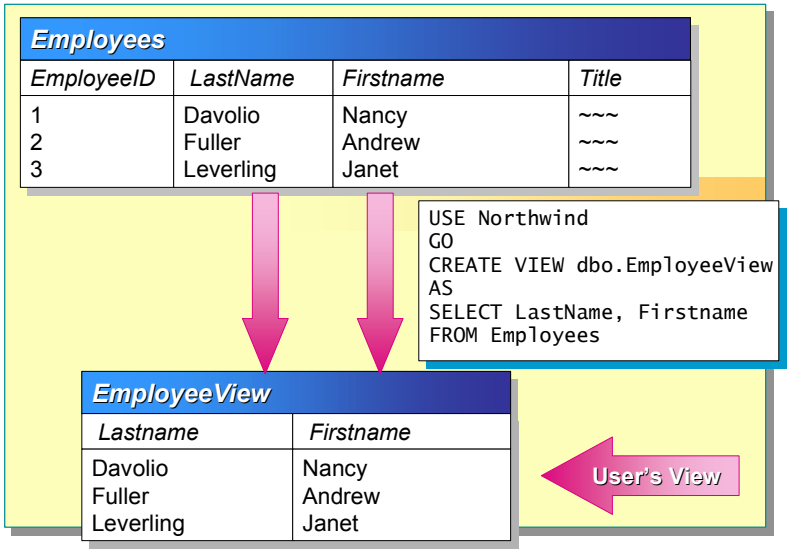
After completing this module, you will be able to:

- Describe the concept of a view.
- List the advantages of views.
- Define a view by using the CREATE VIEW statement.
- Modify data through views.
- Optimize performance by using views.

Introduction to Views

Topic Objective
To introduce the concept of views and provide an example.

Lead-in
A view is an alternate way of looking at data from one or more tables.



**********ILLEGAL FOR NON-TRAINER USE**********

Delivery Tip
Until this point in the course, we have been writing ad-hoc queries. With views, we'll begin our discussion of storing queries as objects (views, stored procedures, and triggers) in the database.

A view provides the ability to store a predefined query as an object in the database for later use. The tables queried in a view are called *base tables*. With a few exceptions, you can name and store any SELECT statement as a view. Common examples of views are:

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

Example

This example creates the **dbo.EmployeeView** view in the **Northwind** database. The view displays two columns in the **Employees** table.

```
USE Northwind
GO
CREATE VIEW dbo.EmployeeView
AS
SELECT LastName, Firstname
FROM Employees
```

Query

```
SELECT * from EmployeeView
```

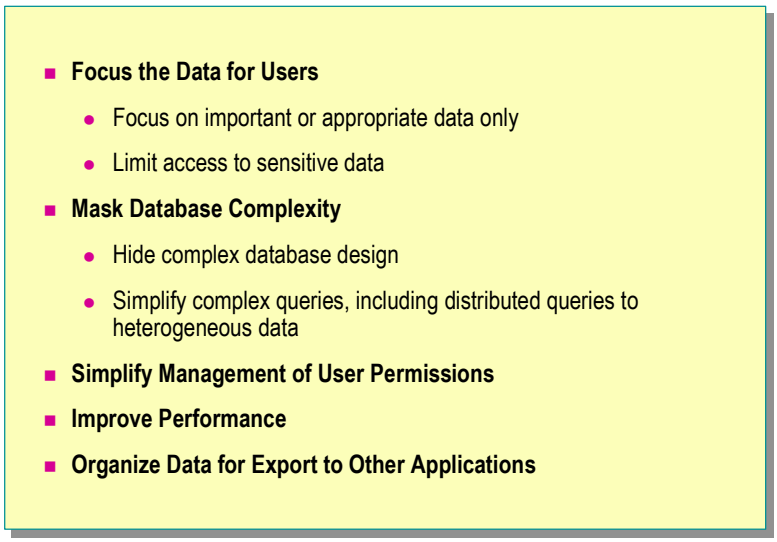
Result

```
LastName                                  FirstName
Davolio                                   Nancy
Fuller                                    Andrew
Leverling                                 Janet
.
.
.
(9 row(s) affected)
```

Advantages of Views

Topic Objective
To discuss why users would want to create or use views.

Lead-in
Views offer several advantages.



**********ILLEGAL FOR NON-TRAINER USE**********

Views offer several advantages, including focusing data for users, masking data complexity, simplifying permission management, and organizing data for export to other applications.

Focus the Data for Users

Views create a controlled environment that allows access to specific data while other data is concealed. Data that is unnecessary, sensitive, or inappropriate can be left out of a view. Users can manipulate the display of data in a view, as is possible in a table. In addition, with the proper permissions and a few restrictions, users can modify the data that a view produces.

Delivery Tip
Point out that the information schema views allow SQL Server to present system data in a consistent manner, even when significant changes have been made to the system tables.

Mask Database Complexity

Views shield the complexity of the database design from the user. This provides developers with the ability to change the design without affecting user interaction with the database. In addition, users can see a friendlier version of the data by using names that are easier to understand than the cryptic names that are often used in databases.

Complex queries, including distributed queries to heterogeneous data, can also be masked through views. The user queries the view instead of writing the query or executing a script.

Simplify Management of User Permissions

Instead of granting permission for users to query specific columns in base tables, database owners can grant permission for users to query data through views only. This also protects changes in the design of the underlying base tables. Users can continue to query the view without interruption.

Improve Performance

Views allow you to store results of complex queries. Other queries can use these summarized results. Views also allow you to partition data. You can place individual partitions on separate computers.

Organize Data for Export to Other Applications

You can create a view based on a complex query that joins two or more tables and then export the data to another application for further analysis.

Defining Views

Topic Objective
To introduce a section on working with views.

Lead-in
This section describes creating, altering and dropping views.

- Creating Views
- Example: View of Joined Tables
- Altering and Dropping Views
- Avoiding Broken Ownership Chains
- Locating View Definition Information
- Hiding View Definitions

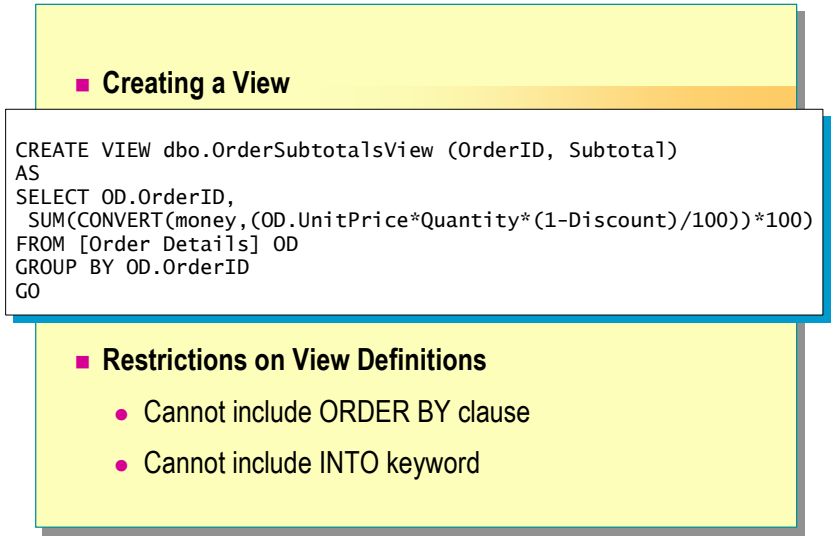
**********ILLEGAL FOR NON-TRAINER USE**********

This section describes creating, altering and dropping views. It also covers how to avoid broken ownership chains, to hide view definitions, and to obtain information on views within your database.

Creating Views

Topic Objective
To introduce creating and dropping views.

Lead-in
Now that we have defined views, let's discuss how to create a view.



**********ILLEGAL FOR NON-TRAINER USE**********

You can create views by using the Create View Wizard, SQL Server Enterprise Manager, or Transact-SQL. You can create views only in the current database.

Creating a View

Delivery Tip
Recommend that students develop a consistent naming convention to distinguish views from tables and that they specify **dbo** as the owner name.

When you create a view, Microsoft® SQL Server™ 2000 verifies the existence of objects that are referenced in the view definition. Your view name must follow the rules for identifiers. Specifying a view owner name is optional. You should develop a consistent naming convention to distinguish views from tables. For example, you could add the word view as a suffix to each view object that you create. This allows similar objects (tables and views) to be easily distinguished when you query the **INFORMATION_SCHEMA.TABLES** view.

Syntax

CREATE VIEW *owner.view_name* [(*column* [,*n*])]
[WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA} [,n]]
AS
select_statement

[WITH CHECK OPTION]

Delivery Tip
It is possible to have been granted permission to create a view and not have permission on the underlying tables. However, the view that is created in this situation would not return a result set.

To execute the CREATE VIEW statement, you must be a member of the system administrators (**sysadmin**) role, database owner (**db_owner**) role, or the data definition language administrator (**db_ddladmin**) role, or you must have been granted the CREATE VIEW permission. You must also have SELECT permission on all tables or views that are referenced within the view.

To avoid situations in which the owner of a view and the owner of the underlying tables differ, it is recommended that the **dbo** user own all objects in a database. Always specify the **dbo** user as the owner name when you create the object; otherwise, the object will be created with your user name as the object owner.

You specify the contents of a view by using a SELECT statement. With a few limitations, views can be as complex as you like. You must specify column names if:

- Any of the columns of the view are derived from an arithmetical expression, built-in function, or constant.
- Any columns in tables that will be joined share the same name.

Delivery Tip
You can specify column names in one of two ways: in the SELECT statement, by using column aliasing, or in the CREATE VIEW statement.

Important When you create views, it is important to test the SELECT statement that defines the view to ensure that SQL Server returns the expected result set. After you have written and tested the SELECT statement and verified the results, create the view.

Restrictions on View Definitions

When you create views, consider the following restrictions:

- The CREATE VIEW statement cannot include the COMPUTE, or COMPUTE BY clauses. The CREATE VIEW statement cannot include the INTO keyword.
- The CREATE VIEW statement can include the ORDER BY clause, only if the TOP keyword is used.
- Views cannot reference temporary tables.
- Views cannot reference more than 1,024 columns.
- The CREATE VIEW statement cannot be combined with other Transact-SQL statements in a single batch.

Example 1

Here is an example of a view that creates a column (**Subtotal**) that calculates the subtotals of an order from the **UnitPrice**, **Quantity**, and **Discount** columns.

```
CREATE VIEW dbo.OrderSubtotalsView (OrderID, Subtotal)
AS
SELECT OD.OrderID,
   SUM(CONVERT
    (money,(OD.UnitPrice*Quantity*(1-  Discount)/100))*100)
FROM [Order Details] OD
GROUP BY OD.OrderID
GO
```

Example 2

This example queries the view to see the results.

```
SELECT * FROM OrderSubtotalsView
```

Result

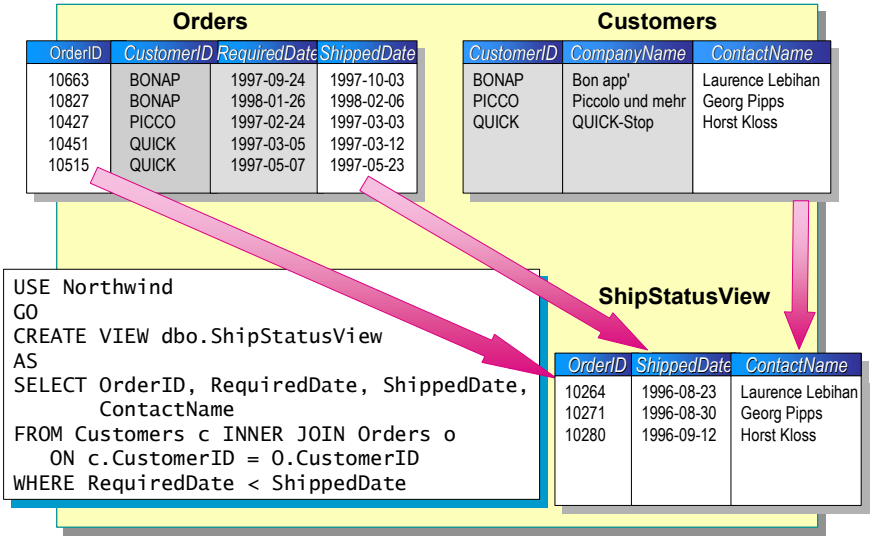
OrderID	Subtotal
10271	48.0000
10977	2233.0000
10440	4924.1400
.	
.	
.	

```
(830 row(s) affected)
```

Example: View of Joined Tables

Topic Objective
To give an example of a view of two or more joined tables.

Lead-in
You can create several types of views. One type is a subset of columns, as we saw in an earlier slide. Another, more common type is a view of two or more joined tables.



**********ILLEGAL FOR NON-TRAINER USE**********

You often create views to provide a convenient way of looking at information from two or more joined tables in one central location.

Example 1

In this example, **ShipStatusView** joins the **Customers** and **Orders** tables.

```
USE Northwind
GO
CREATE VIEW dbo.ShipStatusView
AS
SELECT OrderID, ShippedDate, ContactName
FROM Customers c INNER JOIN Orders o
   ON c.CustomerID = O.CustomerID
WHERE RequiredDate < ShippedDate

SELECT * FROM ShipStatusView
```

Result

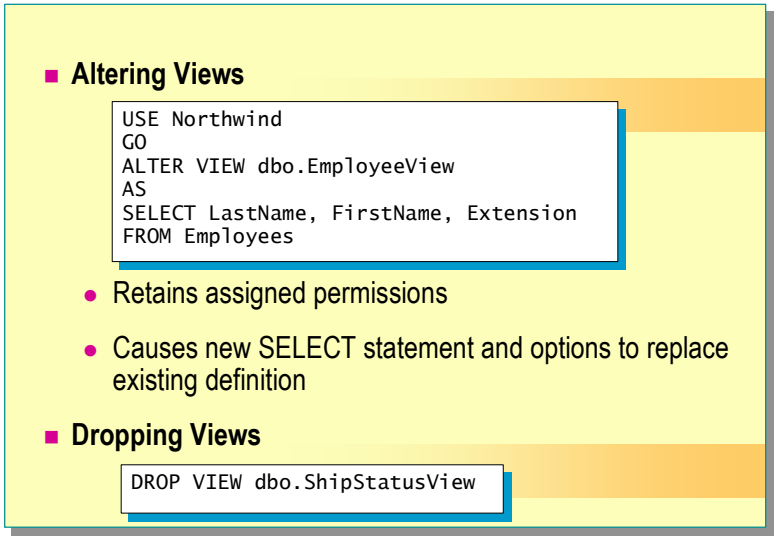
OrderID	ShippedDate	ContactName
10264	1996-08-23	Maria Larsson
10271	1996-08-30	Art Braunschweiger
10280	1996-09-12	Christina Berglund

```
.
.
.
(37 row(s) affected)
```

Altering and Dropping Views

Topic Objective
To introduce how to alter a view.

Lead-in
It is possible to alter a view.



**********ILLEGAL FOR NON-TRAINER USE**********

You often alter views in response to requests from users for additional information or to changes in the underlying table definition. You can alter a view by dropping and recreating it or by executing the ALTER VIEW statement.

Altering Views

The ALTER VIEW statement changes the definition of a view, including indexed views, without affecting dependent stored procedures or triggers. This allows you to retain permissions for the view. This statement is subject to the same restrictions as the CREATE VIEW statement. If you drop a view and then recreate it, you must reassign permissions to it.

Syntax

```
ALTER VIEW owner.view_name
  [(column [,...n ])]
  [WITH  {ENCRYPTION | SCHEMABINDING | VIEW_METADATA} [,...n]]
  AS
  select_statement
  [WITH CHECK OPTION]
```

Note If you use the WITH CHECK OPTION, WITH ENCRYPTION, WITH SCHEMABINDING, or WITH VIEW_METADATA option when you create the view, you must include it in the ALTER VIEW statement if you want to retain the functionality that the option provides.

Example

The following example alters **EmployeeView** to add the **Extension** column.

```
USE Northwind
GO
ALTER VIEW dbo.EmployeeView
AS
SELECT LastName, FirstName, Extension
FROM Employees
```

Query

```
SELECT * from  dbo.EmployeeView
```

Result

LastName	FirstName	Extension
Davolio	Nancy	5467
Fuller	Andrew	3457
Leverling	Janet	3355
.		
.		
.		

(9 row(s) affected)

Delivery Tip
When the view is created, the column list is stored in the **syscolumns** table.

Note If you define a view with a SELECT * statement, and then alter the structure of the underlying tables by adding columns, the new columns do not appear in the view. When all columns are selected in a CREATE VIEW statement, the column list is interpreted only when you first create the view. To see the new columns in the view, you must alter the view.

Dropping Views

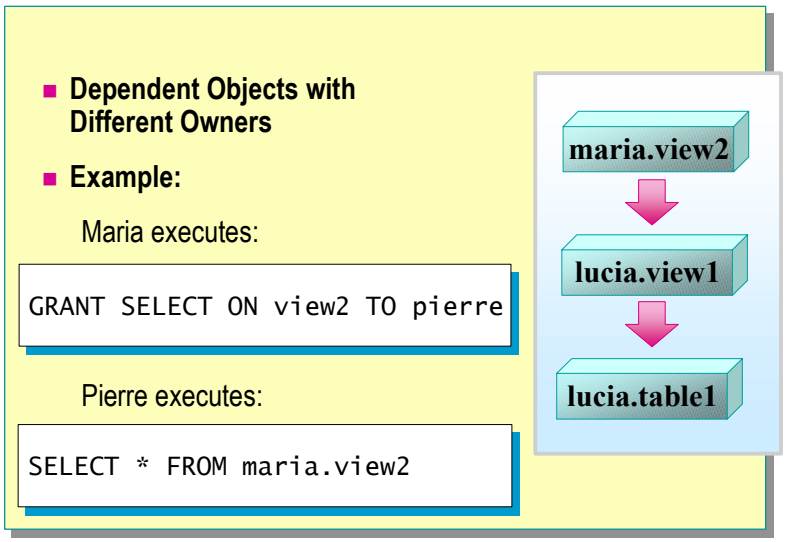
If you no longer need a view, you can remove its definition from the database by executing the DROP VIEW statement. Dropping a view removes its definition and all permissions assigned to it. Furthermore, if users query any views that reference the dropped view, they receive an error message. However, dropping a table that references a view does not drop the view automatically. You must drop it explicitly.

Note The permission to drop a view goes to the view owner and is nontransferable. This is the default. However, the system administrator or database owner can drop any object by specifying the owner name in the DROP VIEW statement.

Avoiding Broken Ownership Chains

Topic Objective
To introduce the concept of ownership chains.

Lead-in
To avoid broken ownership chains, the **dbo** user should own all views.



**********ILLEGAL FOR NON-TRAINER USE**********

SQL Server allows the owner of the original object to retain control over users who are authorized to access the object.

Dependent Objects with Different Owners

View definitions depend on underlying objects (views or tables). These dependencies can be thought of as the ownership chain. If the owner of a view also owns the underlying objects, the owner only has to grant permission for the view. When the object is used, permissions are checked only on the view.

To avoid broken ownership chains, the **dbo** user should own all views. When the object is used, permissions are checked on each dependent object with a different owner.

Example

Maria creates **view2**. With the following statement, she grants permission to Pierre to query it.

Syntax

```
GRANT select ON view2 TO pierre
```

However, **maria.view2** depends on an object (**view1**) owned by another user (Lucia). Permissions are checked on each dependent object with a different owner.

Pierre queries the view by using the following statement:

Syntax

```
SELECT * FROM maria.view2
```

Because **maria.view2** depends on **lucia.view1**, SQL Server checks the permissions on **maria.view2** and **lucia.view1**. If Lucia has previously granted permission to Pierre on **view1**, Pierre is allowed access. If Lucia has not previously granted permission to Pierre, access is denied, allowing Lucia to retain control over individuals who are authorized to access the objects that she creates.

Locating View Definition Information

Topic Objective
To describe how to see information about views.

Lead-in
As you create, alter, or drop tables, you will want to see information about the views in your database.

- **Locating View Definitions**
  - Not available if view was created using WITH ENCRYPTION option
- **Locating View Dependencies**
  - Lists objects upon which view depends
  - Lists objects that depend on a view

**********ILLEGAL FOR NON-TRAINER USE**********

You may want to see the definition of a view in order to alter the view definition or to understand how its data is derived from the base tables.

Locating View Definitions

You can locate view definition information with SQL Server Enterprise Manager or by querying the following views and system tables.

Information schema view or system table	Displays information on
INFORMATION_SCHEMA.TABLES or **sysobjects**	View names.
INFORMATION_SCHEMA.VIEW_TABLE_USAGE or **sysdepends**	Base object names.
INFORMATION_SCHEMA.VIEWS or **syscomments**	View definition.
INFORMATION_SCHEMA.VIEW_COLUMN_USAGE or **syscolumns**	Columns that are defined in a view.

Note **INFORMATION_SCHEMA.VIEW_TABLE_USAGE** and **INFORMATION_SCHEMA.VIEW_COLUMN_USAGE** display information for your user name only.

To display the text that was used to create a view, use SQL Server Enterprise Manager, query **INFORMATION_SCHEMA.VIEWS**, or execute the **sp_helptext** system stored procedure with the view name as the parameter.

Syntax

sp_helptext *objname*

Delivery Tip
Demonstrate how to display dependency information by using SQL Server Enterprise Manager.

Locating View Dependencies

To retrieve a report of the tables or views on which a view depends and of objects that depend on a particular view, use SQL Server Enterprise Manager or execute the **sp_depends** system stored procedure.

You should view dependencies before you drop any object. Before you alter or drop a table, use the **sp_depends** system stored procedure to determine whether any objects reference the table.

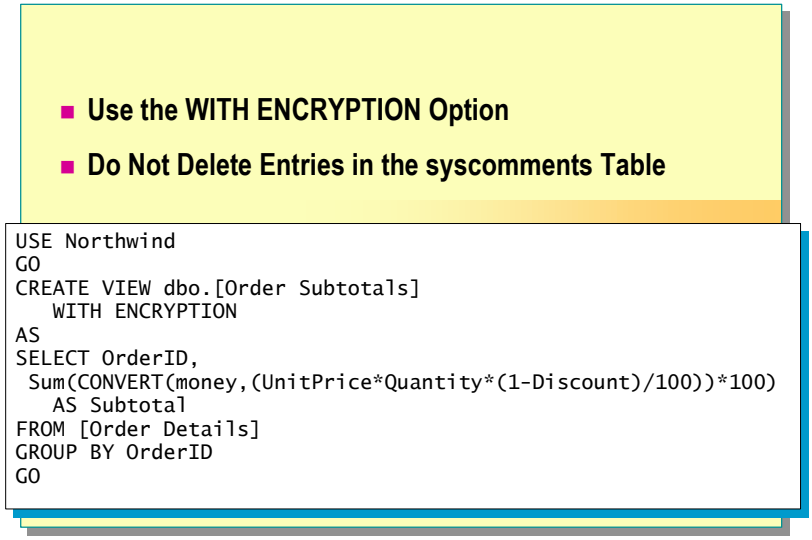
Syntax

sp_depends *objname*

Hiding View Definitions

Topic Objective
To discuss how to encrypt view definitions.

Lead-In
You can encrypt the definition of views to hide the details of the base tables that the view queries.



**********ILLEGAL FOR NON-TRAINER USE**********

Because users may display the definition of a view by using SQL Server Enterprise Manager, by querying **INFORMATION_SCHEMA.VIEWS**, or by querying the **syscomments** system table, you might want to hide certain view definitions.

Use the WITH ENCRYPTION Option

You can encrypt the **syscomments** table entries that contain the text of the CREATE VIEW statement by specifying the WITH ENCRYPTION option in the view definition.

Before you encrypt a view, ensure that the view definition (script) is saved to a file. To decrypt the text of a view, you must drop the view and recreate it, or alter the view and use the original syntax.

Example

In this example, **dbo.[Order Subtotals]** is created by using the WITH ENCRYPTION option so that the view definition is hidden.

```
USE Northwind
GO
CREATE VIEW dbo.[Order Subtotals]
   WITH ENCRYPTION
AS
SELECT OrderID,
 Sum(CONVERT(money,(UnitPrice*Quantity*(1-Discount)/100))*100)
   AS Subtotal
FROM [Order Details]
GROUP BY OrderID
```

Do Not Delete Entries in the syscomments Table

When security considerations require that the view definition be unavailable to users, use encryption. Never delete entries from the **syscomments** table. This prevents you from using the view, and it prevents SQL Server from recreating the view when you upgrade a database to a newer version of SQL Server.

Modifying Data Through Views

Topic Objective
To introduce considerations of which students must be aware when they modify data with views.

Lead-in
Data changes that you make through a view modify the underlying tables.

- **Cannot Affect More Than One Underlying Table**
- **Cannot Be Made to Certain Columns**
- **Can Cause Errors If They Affect Columns That Are Not Referenced in the View**
- **Are Verified If the WITH CHECK OPTION Has Been Specified**

**********ILLEGAL FOR NON-TRAINER USE**********

Views do not maintain a separate copy of data. Instead, they show the result set of a query on one or more base tables. Therefore, whenever you modify data in a view, you are actually modifying the base table.

With some restrictions, you can insert, update, or delete table data freely through a view. In general, the view must be defined on a single table and must not include aggregate functions or GROUP BY clauses in the SELECT statement.

Specifically, modifications that are made by using views:

- Cannot affect more than one underlying table.

  You can modify views that are derived from two or more tables, but each update or modification can affect only one table.

- Cannot be made on certain columns.

  SQL Server does not allow you to change a column that is the result of a calculation, such as columns that contain computed values, built-in functions, or row aggregate functions.

- Can cause errors if modifications affect columns that are not referenced in the view.

  For example, you will receive an error message if you insert a row into a view that is defined on a table that contains columns that are not referenced in the view and that do not allow NULLs or contain default values.

- Are verified if the WITH CHECK OPTION has been specified in the view definition.

  The WITH CHECK OPTION forces all data modification statements that are executed against the view to adhere to certain criteria. These criteria are specified within the SELECT statement that defines the view. If the changed values are out of the range of the view definition, SQL Server rejects the modifications.

Optimizing Performance by Using Views

Topic Objective
To describe how to optimize performance by using views.

Lead-in
This section describes how to optimize performance by using views.

- Performance Considerations
- Using Indexed Views
- Using Views to Partition Data

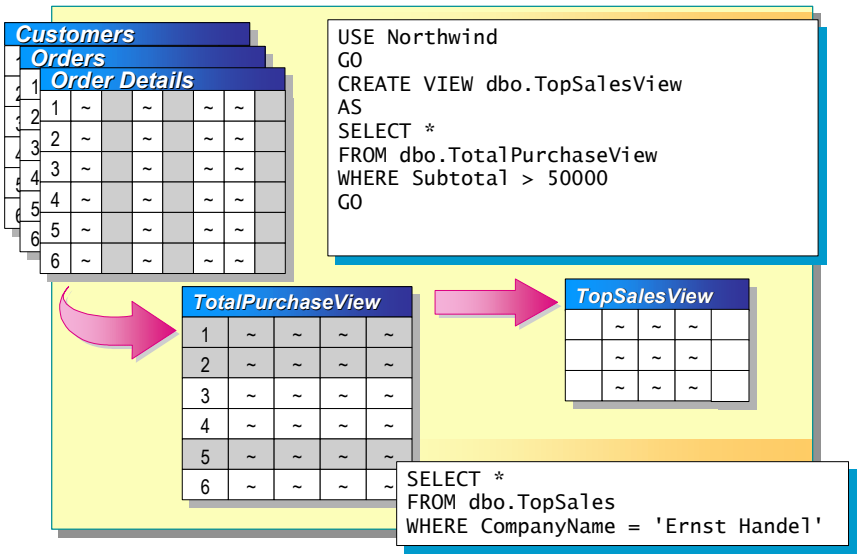
**********ILLEGAL FOR NON-TRAINER USE**********

This section describes performance considerations for using views, and how views allow you optimize performance by storing results of complex queries and partitioning data.

Performance Considerations

Topic Objective
To show how to create a view that contains another view.

Lead-in
If you want to use work that you've already done, you can create a view of another view. However, you should be aware of potential performance problems.



**********ILLEGAL FOR NON-TRAINER USE**********

When views that join several tables and evaluate complex expressions are nested within another view, the immediate source of any performance problems may be difficult to determine. Therefore, you may want to consider creating separate view definitions rather than nesting views.

Example

In the following example, **TopSalesView** queries a subset of rows from **TotalPurchaseView**.

Delivery Tip
Ask: How far can you nest a view of a view?

Answer: Available resources are the only limit. Generally, do not nest more than three levels in order to avoid hiding potential performance problems.

```
USE Northwind
GO
CREATE VIEW dbo.TopSalesView
AS
SELECT *
FROM dbo.TotalPurchaseView
WHERE Subtotal > 50000
GO
```

The view definition of **dbo.TopSalesView** hides the complexity of the underlying query that is used to create TotalPurchaseView, which joins three base tables.

```
USE Northwind
GO
CREATE VIEW dbo.TotalPurchaseView
AS
SELECT CompanyName, Sum(CONVERT(money,
   (UnitPrice*Quantity*(1-Discount)/100))*100) AS Subtotal
FROM Customers c INNER JOIN Orders o
     ON c.CustomerID=o.CustomerID
   INNER JOIN [Order Details] od
     ON o.OrderID = od.OrderID
GROUP BY CompanyName
GO
```

Query

If users experience any performance problems when they execute the following query to list the available French language books, the source of the problem will not be readily apparent.

```
SELECT *
FROM dbo.TopSales
WHERE CompanyName = 'Ernst Handel'
```

Result

CompanyName	Subtotal
Ernst Handel	104874.98

(1 row(s) affected)

Using Indexed Views

Topic Objective
To describe indexed views.

Lead-in
You can create indexes on views.

- **Indexed Views Store the Result Sets in the Database**
- **Creating an Indexed View**
- **Guidelines for Creating Indexed Views**

  Use when:

  - Performance gains outweigh maintenance costs
  - Underlying data is infrequently updated
  - Queries perform many joins and aggregations
- **Restrictions on Creating Indexed Views**

**********ILLEGAL FOR NON-TRAINER USE**********

You can create indexes on views. An *indexed view* stores the result set of a view in the database. Because of the fast retrieval time, you can use indexed views to improve query performance.

Creating an Indexed View

Create an indexed view by implementing a UNIQUE CLUSTERED index on a view. The results of the view are stored in the leaf-level pages of the clustered index. After you create the UNIQUE CLUSTERED index, you can create other indexes on that view.

An indexed view automatically reflects modifications made to data in the base tables. As data changes, the UNIQUE CLUSTERED index is updated.

Guidelines for Creating Indexed Views

The query optimizer automatically determines whether a given query will benefit from using an indexed view. It can determine this even if the query does not reference the indexed view. As a general practice, allow the query optimizer to determine when to use indexed views.

By using the Index Tuning Wizard, you can greatly enhance your ability to determine the best mix of indexes and indexed views to optimize query performance.

Create indexed views when:

- The performance gain of improved speed in retrieving results outweighs the increased maintenance cost.
- The underlying data is infrequently updated.
- Queries perform a significant amount of joins and aggregations that either process many rows or are performed frequently by many users.

Restrictions on Creating Indexed Views

Consider the following guidelines when you create indexed views:

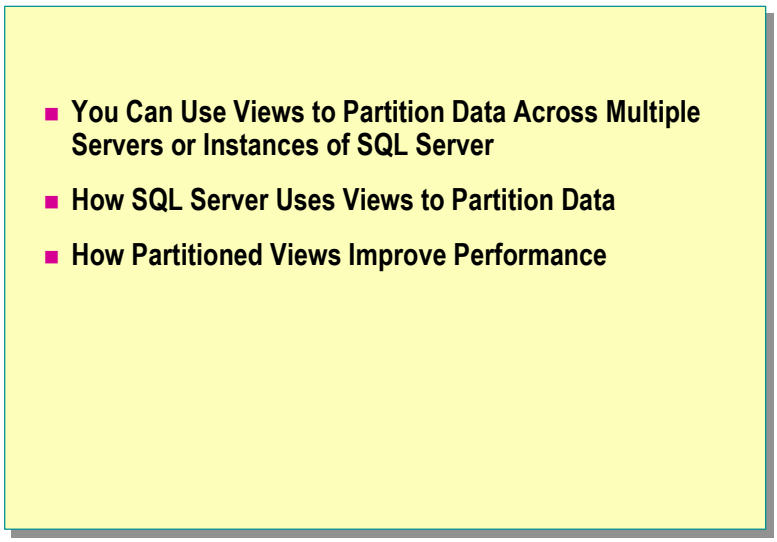
- The first index that you create on a view must be a unique clustered index.
- You must create the view with the SCHEMABINDING option.
- The view can reference base tables, but it cannot reference other views.
- You must use two-part names to reference tables and user-defined functions.
- Subsequent connections must have the same option settings to use the indexed view.

Note You should use the **IsIndexable** property of the OBJECTPROPERTY function to make sure that you can index a view.

Using Views to Partition Data

Topic Objective
To introduce partitioned views.

Lead-in
You can use views to partition data across multiple databases or instances of SQL Server.



**********ILLEGAL FOR NON-TRAINER USE**********

You can use views to partition data across multiple databases or instances of SQL Server to improve performance.

How SQL Server Uses Views to Partition Data

You can use the UNION set operator within a view to combine the results of two or more queries from separate tables into a single result set. This appears to the user as a single table called a *partitioned view*. You can update partitioned views even though they reference multiple tables.

Partitioned views can be based on data from multiple heterogeneous sources, such as remote servers, not just tables in the same database. This allows you to distribute database processing across a group of servers. The group of servers can support the processing needs for large e-commerce applications or corporate data centers.

How Partitioned Views Improve Performance

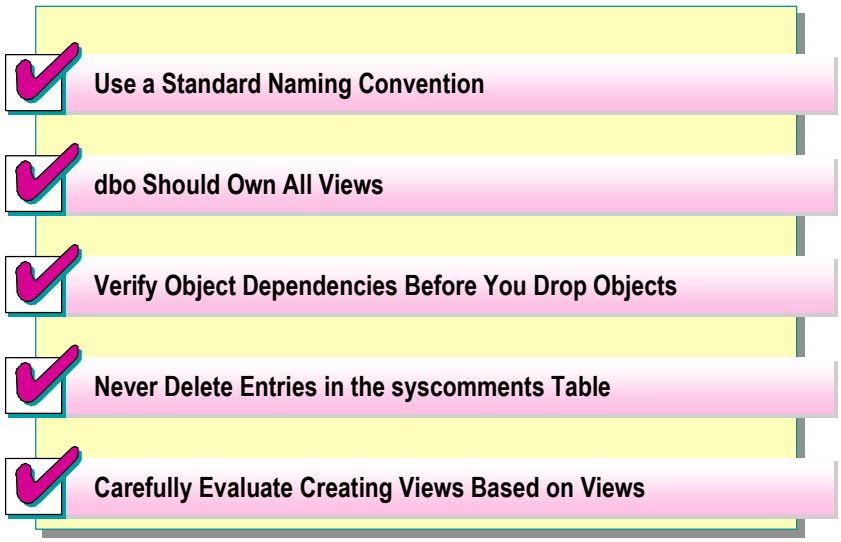
If the tables in a partitioned view are on different servers, or on a computer with multiple processors, each table involved in the query can be scanned in parallel, thereby improving query performance. In addition, maintenance tasks, such as rebuilding indexes or backing up a table, can execute faster.

Note You cannot create an index on a partitioned view. The view definition required to build the indexed view only allows two-part names; a partitioned view requires the use of three- or four-part names, such as, *Servername.databasename.ownername.objectname.*

Recommended Practices

Topic Objective
To present recommended practices for using views.

Lead-in
The following are recommended practices for using views.



**********ILLEGAL FOR NON-TRAINER USE**********

The following recommended practices should help you use and manage views in your databases:

- You should develop a consistent naming convention to distinguish views from tables.
- Specify **dbo** as the owner when you create views. The **dbo** should own all objects referenced in the view definition. This prevents the need to specify the owner name when you query the view, because the database owner is the default owner. The database owner also has permission on all underlying objects in the database, thereby preventing potential broken ownership chains.
- Verify object dependencies before you drop objects from the database. Execute the **sp_depends** system stored procedure, or display the dependencies in SQL Server Enterprise Manager to ensure that dependencies do not exist on an object that you plan to drop.
- Never delete entries in the **syscomments** system table. If your application requires that the definition is invisible to others, include the WITH ENCRYPTION option with either the CREATE VIEW or ALTER VIEW statement. Be sure to save your script definition before you encrypt the script.
- Carefully evaluate whether to create views based on views. They can hide complexities and could be the source of performance problems.

Additional information on the following topics is available in SQL Server Books Online.

Topic	Search on
CREATE VIEW	“create view”
ALTER VIEW	“alter view”
DROP VIEW	“drop view”
Broken ownership chains	“ownership chain”
Generating SQL scripts	“documenting and scripting databases”

Lab A: Implementing Views

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will create, test, and modify views.

**********ILLEGAL FOR NON-TRAINER USE**********

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Create a view using a wizard.
- Alter a view to encrypt its definition.
- Alter a view by using WITH CHECK OPTION.
- Use information schema to obtain information about views.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2073A\Labfiles\L08.
- Answer files for this lab, which are located in C:\Moc\2073A\Labfiles\L08\Answers.

Lab Setup

To complete this lab, you must have either:

- Completed the prior lab, or
- Executed the C:\Moc\2073A\Batches\Restore08.cmd batch file.

  This command file restores the **ClassNorthwind** database to a state required for this lab.

For More Information

If you require help with executing files, search SQL Query Analyzer Help for "Execute a query".

Other resources that you can use include:

- The **Northwind** database schema.
- Microsoft SQL Server Books Online.

Scenario

The organization of the classroom is meant to simulate that of a worldwide trading firm named Northwind Traders. Its fictitious domain name is nwtraders.msft. The primary DNS server for nwtraders.msft is the instructor computer, which has an Internet Protocol (IP) address of 192.168.*x*.200 (where *x* is the assigned classroom number). The name of the instructor computer is London.

The following table provides the user name, computer name, and IP address for each student computer in the fictitious **nwtraders.msft** domain. Find the user name for your computer, and make a note of it.

User name	Computer name	IP address
SQLAdmin1	Vancouver	192.168.*x*.1
SQLAdmin2	Denver	192.168.*x*.2
SQLAdmin3	Perth	192.168.*x*.3
SQLAdmin4	Brisbane	192.168.*x*.4
SQLAdmin5	Lisbon	192.168.*x*.5
SQLAdmin6	Bonn	192.168.*x*.6
SQLAdmin7	Lima	192.168.*x*.7
SQLAdmin8	Santiago	192.168.*x*.8
SQLAdmin9	Bangalore	192.168.*x*.9
SQLAdmin10	Singapore	192.168.*x*.10
SQLAdmin11	Casablanca	192.168.*x*.11
SQLAdmin12	Tunis	192.168.*x*.12
SQLAdmin13	Acapulco	192.168.*x*.13
SQLAdmin14	Miami	192.168.*x*.14
SQLAdmin15	Auckland	192.168.*x*.15
SQLAdmin16	Suva	192.168.*x*.16
SQLAdmin17	Stockholm	192.168.*x*.17
SQLAdmin18	Moscow	192.168.*x*.18
SQLAdmin19	Caracas	192.168.*x*.19
SQLAdmin20	Montevideo	192.168.*x*.20
SQLAdmin21	Manila	192.168.*x*.21
SQLAdmin22	Tokyo	192.168.*x*.22
SQLAdmin23	Khartoum	192.168.*x*.23
SQLAdmin24	Nairobi	192.168.*x*.24

Estimated time to complete this lab: 30 minutes

Exercise 1
Creating and Testing Views

In this exercise, you will create views to manage daily requests on the **ClassNorthwind** database. You will use the Create View Wizard and execute a script that creates several views. Finally, you will query the views to verify that you received the expected results.

► To use the Create View Wizard

In this procedure, you will use the Create View Wizard to quickly create a view.

1. Log on to the **NWTraders** classroom domain by using the information in the following table.

Option	Value
User name	**SQLAdmin*x*** (where *x* corresponds to your computer name as designated in the **nwtraders.msft** classroom domain)
Password	**password**

2. Open SQL Server Enterprise Manager.
3. In the console tree, click your server.
4. On the **Tools** menu, click **Wizards**.
5. Expand Database, and then double-click **Create View Wizard**.
6. Use the information in the following table to create a view that lists the products from a particular supplier.

Option	Value
Select a database	**ClassNorthwind**
Select tables	**Products**
Select columns	**ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, Discontinued**
Define restriction	`WHERE SupplierID = 14`
Name the view	**FormaggiProductsView**

7. Query the view to ensure that you received the expected result set.

► To create views from a script

In this procedure, you will execute a script to create views.

1. Open SQL Query Analyzer and, if requested, log in to the (local) server with Microsoft Windows® Authentication.

   You have permission to log in to and administer SQL Server because you are logged as **SQLAdmin*x***, which is a member of the Microsoft Windows 2000 local group, Administrators. All members of this group are automatically mapped to the SQL Server **sysadmin** role.

2. Open, review, and execute Labfiles\L08\CreaView.sql to create the following views.

View name	View description
FormaggiProductsView	All products from supplier ID of 14
Customer and Suppliers by City	All customers and all suppliers listed (UNION)
Current Product List	All products that are not discontinued
Orders Qry	All customers orders and order information
Products Above Average Price	All products that are priced above the average product unit price
Products by Category	All products listed by category
Invoices	All invoice information
Order Details Extended	All order details extended price information
Sales by Category	All sales for 1997 listed by category

3. Open a new query window and execute the **sp_depends** system stored procedure. List the tables on which the **Order Details Extended** view depends, as well as objects that depend on **Order Details Extended**.

   Tables: Order details and Products. View: Sales by Category.

4. Switch to SQL Server Enterprise Manager to determine the dependencies on the **Orders** table.

5. In the console tree, expand the **ClassNorthwind** database, and then click **Tables**.

6. In the details pane, right-click the **Orders** table; on the shortcut menu, click **All Tasks**, and then click **Display Dependencies**.

   What objects are dependent on the **Orders** table?

   Tables: Order Details. Views: Invoices, Orders Qry, Product Sales for 1997, Quarterly Orders, Sales by Category, and Sales Totals by Amount.

Exercise 2
Encrypting a View Definition

In this exercise, you will alter a view to encrypt its definition so that it will be invisible.

► **To alter and encrypt the Sales by Category view**

In this procedure, you will use SQL Server Enterprise Manager to display the script that created the **Sales by Category** view. Then, you will alter the view to encrypt the script. L08\Answers\EncryptView.sql is a completed script for this procedure.

1. Open SQL Server Books Online to the topic "How to generate a script (Enterprise Manager)."
2. Use the procedure in SQL Server Books Online to generate a script for the **Sales by Category** view.
3. Save your script as SaleByCatView.sql

Note If you display the properties of the view or select preview when you generate the script, you can copy and paste the view definition into a query window for modification.

4. In a query window, revise the script to alter **Sales by Category** so that it is created by using the WITH ENCRYPTION option.
5. Execute the modified script to alter **Sales by Category**.
6. Save your revised script as SaleByCatView.sql

► To test that the statements have been encrypted

In this procedure, you will use the **sp_helptext** system stored procedure and SQL Server Enterprise Manager to observe the effect of using the encryption option.

1. Execute the **sp_helptext** system stored procedure that displays the script that created **Sales by Category**.

   The Results window will display the following statement: "The object's comments have been encrypted."

2. In SQL Server Enterprise Manager, in the details pane, right-click **Sales by Category**, and then click **Properties**.

   Can you see the CREATE VIEW syntax that was used to create **Sales by Category**?

   No. The syntax is encrypted.

   _____

3. How would you decrypt the **Sales by Category** view?

   By altering the view and including the original syntax, without including the WITH ENCRYPTION option. This assumes that you have previously saved the script with the original syntax.

   _____

Exercise 3
Modifying Data Through Views

In this exercise, you will alter **FormaggiProductsView** to include WITH CHECK OPTION so that data modifications can only be made that adhere to the view definition.

► To alter FormaggiProductsView to enable WITH CHECK OPTION

In this procedure, you will alter **FormaggiProductsView** to enable WITH CHECK OPTION. L08\Answers\Supplier14.sql is a completed script for this procedure.

1. Generate a script for the **FormaggiProductsView** view.
2. Modify your script to enable WITH CHECK OPTION.
3. Execute the script and save the file with your modifications.

► To update the title table through FormaggiProductsView

In this procedure, you will update the **Products** table with data that is out of the range of **FormaggiProductsView**. You then will observe the results.

1. Write an UPDATE statement to change the products listed in **FormaggiProductsView** from supplier 14 to supplier 12, where product ID equals 31.

   ```
   UPDATE dbo.FormaggiProductsView
   SET SupplierID = 12 WHERE ProductID = 31
   ```

2. Execute the UPDATE statement.

   What was the result?

   Error Message 550: The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.

   _____

   _____

Exercise 4
Locating View Definitions

In this exercise, you will query the information schema views to obtain information about the views that you have created in the **ClassNorthwind** database. L08\Answers\Schema.sql is a completed script for this exercise.

► To display information about views

In this procedure, you will query the information schema views to display details about views in the **ClassNorthwind** database.

1. Verify that you are using the **ClassNorthwind** database.
2. Query **INFORMATION_SCHEMA.VIEWS** to display all views and their definitions.

   ```
   SELECT *
   FROM INFORMATION_SCHEMA.VIEWS
   ```

   What information was displayed about **FormaggiProductsView** and **Products Above Average Price**?

   FormaggiProductsView displays CASCADE in the check_option column. The Sales by Category view definition is encrypted in the view_definition column.

   _____

   _____

3. Which information schema view displays a list of table and view names?

   INFORMATION_SCHEMA.TABLES

   _____

   _____

4. Query **INFORMATION_SCHEMA.VIEW_COLUMN_USAGE** to display a list of columns that are referenced in the **Invoices** view.

   ```
   SELECT *
   FROM INFORMATION_SCHEMA.VIEW_COLUMN_USAGE
   WHERE view_name = 'Invoices'
   ```

5. Query **INFORMATION_SCHEMA.VIEW_TABLE_USAGE** to display a list of tables that are referenced in the **Sales by Category** view.

```
SELECT *
FROM INFORMATION_SCHEMA.VIEW_TABLE_USAGE
WHERE view_name = 'Sales by Category '
```

What tables were listed?

Tables: Categories, Orders, and Products. Views: Order Details Extended.

Why were you able to see the tables that are referenced in **Sales by Category** when this view is encrypted?

The WITH ENCRYPTION option only encrypts the view definition in the syscomments system table. Tables or views that are referenced in a view are listed in the sysobjects system table.

Which system tables, system functions, or system stored procedures could also have been used to display information about views?

Sysobjects, sysdepends, syscomments, syscolumns, OBJECTPROPERTY, OBJECT_ID, OBJECT_NAME, sp_helptext, sp_help, and sp_depends.

Review

Topic Objective
To reinforce module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- Introduction to Views
- Advantages of Views
- Defining Views
- Modifying Data Through Views
- Optimizing Performance by Using Views

**********ILLEGAL FOR NON-TRAINER USE**********

1. What are the benefits of views?

   Users focus only on data that they need; user manipulation of data is simplified; database and query complexity is hidden from users, allowing users to see friendly names. Views provide a security mechanism by allowing users access to data only in views. Creating indexes on views and partitioning data by using views can optimize performance.

2. You have developed a query that joins the **Customer**, **Orders**, and **Order Details** tables to list the details of each customer order, such as the quantity of an item and the date that delivery is required. When customers change an existing order, employees need to be able to update the **Orders** and **Order Details** tables. How would you accomplish this task without granting permission on the underlying tables?

   Create a view on your query named OrderDetailsView. Grant update permissions on the RequiredDate and Quantity columns in the view. This ensures that employees can update these columns only in the Orders and Order Details tables.

3. What is the benefit of using the WITH CHECK OPTION in your view definition?

   The option forces modification statements that are executed against the view to adhere to the criteria that are set within the SELECT statement that defines the view.

4. What are some considerations to remember when you use views?

   Objects that are referenced in a view are verified when a view is created; views can be altered so that the permissions that are assigned to the view are maintained. Dropping or altering an underlying table affects the view; if the owner of a view is not the dbo user, the user name must be specified as part of the view name. The same owner should own all objects upon which a view depends in order to avoid broken ownership chains. Hidden complexity can make it difficult to determine the source of performance problems.