MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 9: Introduction to Programming Objects

Contents

Microsoft

Project Lead: Cheryl Hoople
Instructional Designer: Cheryl Hoople
Technical Lead: LeRoy Tuttle
Program Manager: LeRoy Tuttle
Graphic Artist: Kimberly Jackson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Editorial Contributor: Elizabeth Reese
Copy Editor: Bill Jones (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Tools Specialist: Julie Challenger
Production Support: Lori Walker (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: Testing Testing 123
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: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
60 Minutes

Lab:
30 Minutes

This module describes how to create programming objects that enable the user to view and manipulate data while hiding the complexity of the underlying database structure. The module introduces these programming objects—views, stored procedures, triggers, and user-defined functions—and describes the advantages of using them.

At the end of this module, students will be able to:

- Display the text of a programming object.
- Describe the concept of views.
- List the advantages of using views.
- Create views.
- Describe stored procedures.
- Describe triggers.
- Describe user-defined functions.

Materials and Preparation

Required Materials

To teach this module, you will need the following materials:

- Microsoft® PowerPoint® file 2071A_09.ppt.
- The C:\Moc\2071A\Demo\Ex_09.sql example file 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.
- Complete the lab.

Module Strategy

Use the following strategy to present this module:

- Displaying the Text of a Programming Object

  Introduce the **sp_helptext** system stored procedure.

- Introduction to Views

  Introduce the concept of views. Point out that views are simply stored queries.

- Advantages of Views

  List the advantages of using views.

- Creating 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 a set of information without requiring the user to write or execute Transact-SQL statements.

  Discuss how to create, alter, and drop views. Emphasize the importance of testing the statement that creates the view prior to creation of the view itself. Cover 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.

- Introduction to Stored Procedures

  Note that this topic introduces stored procedures and describes how to use stored procedures to improve application design and performance by encapsulating business rules to process common queries and data modifications. Point out that this topic discusses what stored procedures are and the advantages of using them. Emphasize that this topic does *not* attempt to comprehensively address stored procedures.

  Introduce the elements of a stored procedure. Note that students have executed system stored procedures throughout the course, so they should be familiar with how they work. Emphasize that the primary focus of this topic is on creating stored procedures that are defined in a user's local database.

  Discuss how stored procedures are processed in order to explain why they execute faster than batches. Contrast stored procedure performance with the way that batches are processed. Highlight the advantages of stored procedures to point out why students would want to create them in their applications.

- Introduction to Triggers

  Triggers are useful tools for database implementers who want certain actions to be performed whenever data in a specific table is inserted, updated, or deleted. The goal of this section is to promote awareness of triggers and is intended only as an introductory overview.

  Define triggers. Point out that a trigger is a special type of stored procedure that is assigned to a specific table. Then discuss three key points—that triggers are invoked automatically; that they cannot be called by anything other than a trigger action to the trigger table; and that they are transactions.

- Introduction to User-defined Functions

  In addition to a number of system-defined functions that are built-in, Microsoft SQL Server™ 2000 allows users to create their own user-defined functions.

  Introduce the general syntax for creating a function, specifically CREATE FUNCTION, the function name, the input parameters, the RETURNS clause, and some of the restrictions on creating a user-defined function.

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 2071A, *Querying Microsoft SQL Server 2000 with Transact-SQL*.

Lab Setup

There are no lab setup requirements that affect replication or customization.

Lab Results

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

Overview

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

Lead-in
In this module, you will learn about programming objects.

- Displaying the Text of a Programming Object
- Introduction to Views
- Advantages of Views
- Creating Views
- Introduction to Stored Procedures
- Introduction to Triggers
- Introduction to User-defined Functions

This module describes how to create programming objects that enable the user to view and manipulate data without awareness of the complexity of the underlying database structure. The module introduces these programming objects—views, stored procedures, triggers, and user-defined functions—and describes the advantages of using them.

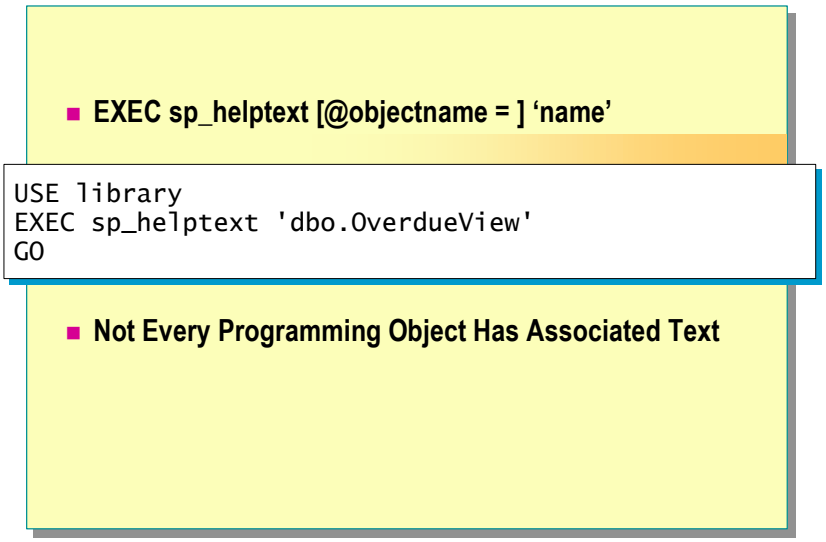
At the end of this module, you will be able to:

- Display the text of a programming object.
- Describe the concept of views.
- List the advantages of using views.
- Create views.
- Describe stored procedures.
- Describe triggers.
- Describe user-defined functions.

Displaying the Text of a Programming Object

Slide Objective
To describe how to display the text associated with a programming object.

Lead-in
You can display the text definition of a programming object with a special system stored procedure.



You can use system stored procedures to perform many administrative and informational activities in Microsoft® *SQL Server*™ 2000. For example, you can use the **sp_helptext** system stored procedure to retrieve the text associated with a programming object.

Syntax

```
EXEC sp_helptext [ @objname = ] 'name'
```

The *parameter* is the name of the object in the current database for which *SQL Server* will display the text of the definition information.

Delivery Tip
Throughout this module, always use **sp_helptext** to display the definition of programming objects.

The **sp_helptext** system stored procedure prints out the text used to create an object in multiple rows, each with 255 characters of the Transact-SQL definition. The definition resides in the text in the **syscomments** table of the current database only.

Example

This example returns the text that defines the **dbo.OverdueView** view.

```
USE library
EXEC sp_helptext 'dbo.OverdueView'
GO
```

Result

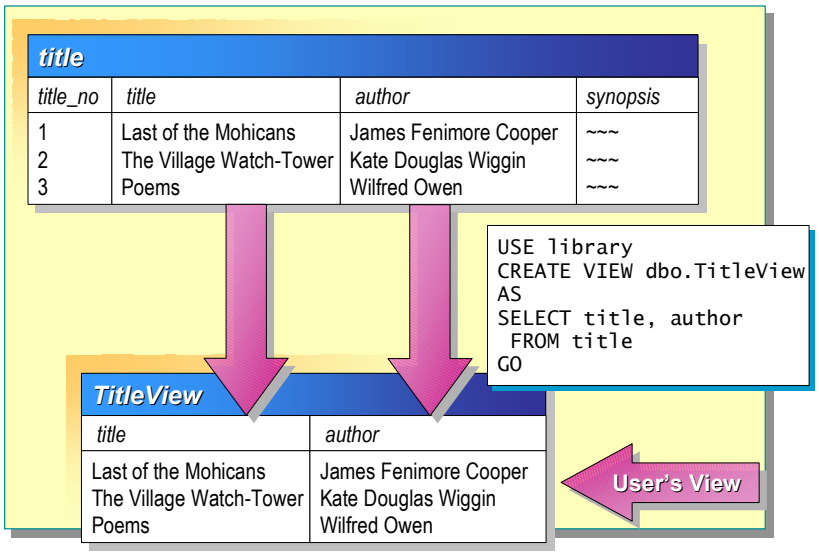
```
Text
------------------------------------------------------------------

/*
OverdueView: Queries OnloanView. (3 table join.)
Lists the member, title, and loan information of a copy on
loan that is overdue.
*/

CREATE VIEW dbo.OverdueView
AS
SELECT *
FROM OnloanView
WHERE OnloanView.due_date < GETDATE()
```

Tip Use **EXEC sp_helptext** to verify the definition of newly created programming objects.

Introduction to Views

Slide 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.



Delivery Tip
Remind students that other modules in this course cover how to write ad-hoc queries.

Point out that this module describes how to store queries as objects (views, stored procedures, and triggers) in the database.

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

A view can be thought of as either a virtual table or a stored query. The data accessible through a view is not stored in the database as a distinct object. What is stored in the database is a SELECT statement. The result set of the SELECT statement forms the virtual table returned by the view. You can use this virtual table by referencing the view name in Transact-SQL statements the same way that you reference a table.

You can use a view to do any or all of these functions:

- Restrict a user to specific rows in a table.

  For example, allow an employee to see only the rows recording his or her work in a labor-tracking table.

- Restrict a user to specific columns.

  For example, allow employees who do not work in payroll to see the name, office, work phone, and department columns in an employee table, but do not allow them to see any columns with salary information or personal information.

- Join columns from multiple tables so that they look like a single table.

- Aggregate information instead of supplying details.

  For example, present the sum of a column, or the maximum or minimum value from a column.

Example

This example creates the **titleview** view in the **library** database. The view displays two columns in the **title** table.

```
USE library
CREATE VIEW dbo.TitleView
AS
SELECT title, author
 FROM title
GO
```

Query

```
SELECT * from TitleView
GO
```

Result

Title	author
Last of the Mohicans	James Fenimore Cooper
The Village Watch-Tower	Kate Douglas Wiggin
Self Help; Conduct & Perseverance	Samuel Smiles
.	
.	
.	

(50 row(s) affected)

Advantages of Views

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

Lead-in
Views offer several advantages.

- **Focus the Data for Users**
  - Focus on important or appropriate data only
  - Limit access to sensitive data
- **Mask Database Complexity**
  - Hide complex database design
  - Simplify complex queries, including distributed queries to heterogeneous data
- **Simplify Management of User Permissions**
- **Organize Data for Export to Other Applications**

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 and conceals other data. Data that is unnecessary, sensitive, or inappropriate can be left out of a view. Users can manipulate the display of data in a view, similar to a table. In addition, with the proper permissions and a few restrictions, users can modify the data that a view produces.

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.

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.

Creating Views

Slide Objective
To introduce the topics that this section covers.

Lead-in
This section describes how to create views.

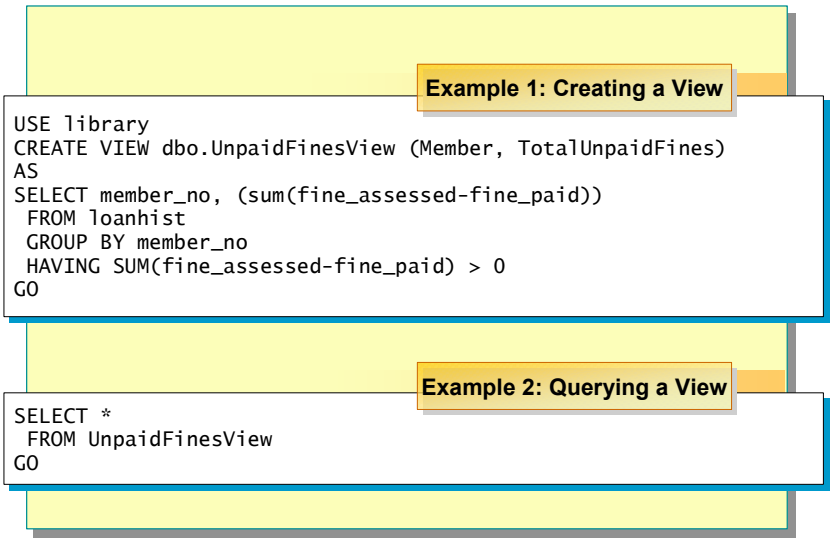
- Defining Views
- Restrictions on Creating Views
- Example: Viewing Information from Multiple Tables

This section describes how to create views and discusses restrictions to consider when creating views. It also provides an example of how to view information from two or more joined tables in one central location.

Defining Views

Slide Objective
To describe how to define views.

Lead-in
When you create a view, SQL Server verifies the existence of objects that are referenced in the view definition.



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

When you create a view, SQL Server 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]
  AS
    select_statement

  [WITH CHECK OPTION]
```

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.

The contents of a view are specified with 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 arithmetic expression, built-in function, or constant.
- Any columns in tables that will be joined share the same name.

Delivery Tip
Column names can be specified 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 in order 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.

Example 1

Here is an example of a view that creates a column (**TotalUnpaidFines**) that contains the values that are calculated by subtracting the value of the **fine_paid** column from the **fine_assessed** column.

```
USE library
CREATE VIEW dbo.UnpaidFinesView (Member, TotalUnpaidFines)
AS
SELECT member_no, (sum(fine_assessed-fine_paid))
 FROM loanhist
 GROUP BY member_no
 HAVING SUM(fine_assessed-fine_paid) > 0
GO
```

Example 2

This example queries the view to see the results.

```
SELECT *
 FROM UnpaidFinesView
GO
```

Result

Member	TotalUnpaidFines
7744	83.2

```
(1 row(s) affected)

Warning: Null value eliminated from aggregate
```

Restrictions on Creating Views

Slide Objective
To describe restrictions when creating views.

Lead-in
When you define views, consider these restrictions.

- **Can Reference a Maximum of 1024 Columns**
- **Cannot Include COMPUTE or COMPUTE BY clauses**
- **Cannot Include ORDER BY Clause, Unless Used in Conjunction with a TOP Clause**
- **Cannot Include the INTO Keyword**
- **Cannot Reference a Temporary Table**
- **Must Be Expressed as a Single Transact-SQL Batch**

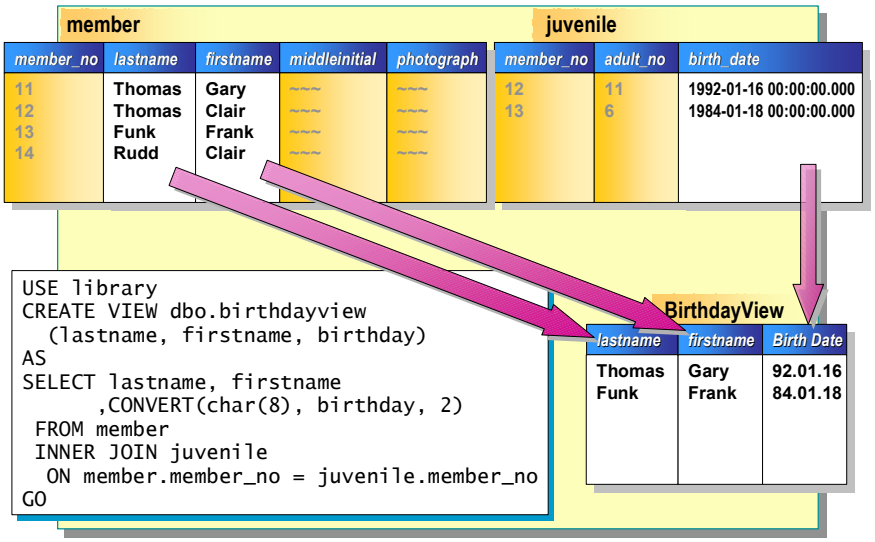
When you create views, consider the following restrictions:

- Views cannot reference more than 1024 columns.
- The CREATE VIEW statement cannot include the COMPUTE or COMPUTE BY clauses.
- The CREATE VIEW statement cannot include the ORDER BY clause, unless used with a TOP clause in the SELECT statement.
- The CREATE VIEW statement cannot include the INTO keyword.
- Views cannot reference temporary tables.
- The CREATE VIEW statement cannot be combined with other Transact-SQL statements in a single batch.

Example: Viewing Information from Multiple Tables

Slide Objective
To give an example of viewing information from multiple tables.

Lead-in
You often create views to view information from two or more joined tables in one central location.



You often create views to view information from two or more joined tables in one central location.

Example

In this example, **birthdayview** that joins the **member** and **juvenile** tables is created.

Delivery Tip
The CONVERT function in the query changes the **birth_date** column, which is defined with the **datetime** data type, to the ANSI format.

```
USE library
CREATE VIEW dbo.birthdayview
  (lastname, firstname, birthday)
AS
SELECT lastname, firstname
      ,CONVERT(char(8), birthday, 2)
 FROM member
 INNER JOIN juvenile
  ON member.member_no = juvenile.member_no
GO
```

If we query the view to determine whether there are any null values in the **birthday** column, it correctly returns no rows.

```
SELECT *
 FROM birthdayview
 WHERE birthday is null
GO
```

Result

```
lastname     firstname   birthday

(0 row(s) affected)
```

Introduction to Stored Procedures

Slide Objective
To list the topics that this section covers.

Lead-in
This section introduces stored procedures and lists some of the advantages of using stored procedures.

- Defining Stored Procedures
- Advantages of Using Stored Procedures

Key Points
Emphasize that this topic does *not* attempt to comprehensively address stored procedures. Refer students to SQL Server Books Online for more information.

This section introduces stored procedures and lists some of the advantages of using stored procedures.

Defining Stored Procedures

Slide Objective
To define stored procedures.

Lead-in
A stored procedure is a named collection of precompiledTransact-SQL statements that is stored on the server.

- **A Stored Procedure Is a Precompiled Collection of Transact-SQL Statements**
- **A Stored Procedure Encapsulates Repetitive Tasks**
- **Stored Procedures Can:**
  - Contain statements that perform operations
  - Accept input parameters
  - Return status value to indicate success or failure
  - Return multiple output parameters

A *stored procedure* is a named collection of precompiled Transact-SQL statements that is stored on the server. Using a stored procedure is a method of encapsulating repetitive tasks that executes efficiently. Stored procedures support user-declared variables, control-of-flow execution, and other advanced programming features.

Stored procedures in SQL Server are similar to procedures in other programming languages in that they can:

- Contain statements that perform operations in the database, including the ability to call other stored procedures.
- Accept input parameters.
- Return a status value to a calling stored procedure or batch to indicate success or failure (and the reason for failure).
- Return multiple values to the calling stored procedure or batch in the form of output parameters.

Example

This example shows the creation of a simple stored procedure with a complex SELECT statement. This stored procedure returns all authors (first and last names supplied), their titles, and their publishers from a four-table join in the **pubs** database. This stored procedure does not use any parameters.

```
USE pubs
CREATE PROCEDURE au_info_all
AS
SELECT au_lname, au_fname, title, pub_name
 FROM authors AS a
 INNER JOIN titleauthor AS ta  ON a.au_id = ta.au_id
 INNER JOIN titles AS t  ON t.title_id = ta.title_id
 INNER JOIN publishers AS p  ON t.pub_id = p.pub_id
GO
```

Advantages of Using Stored Procedures

Slide Objective
To show the advantages of stored procedures.

Lead-in
Stored procedures offer numerous advantages.

- Share Application Logic
- Shield Database Schema Details
- Provide Security Mechanisms
- Improve Performance
- Reduce Network Traffic

Stored procedures offer numerous advantages. Stored procedures significantly reduce resource and time requirements for execution. They can:

- Share application logic with other applications, thereby ensuring consistent data access and modification.

  Stored procedures can encapsulate business functionality. Business rules or policies encapsulated in stored procedures can be changed in a single location. All clients can use the same stored procedures to ensure consistent data access and modification.
- Shield users from exposure to the details of the tables in the database. If a set of stored procedures supports all of the business functions that users must perform, users never have to access the tables directly.
- Provide security mechanisms. Users can be granted permission to execute a stored procedure even if they do not have permission to access the tables or views that are referred to in the stored procedure.
- Improve performance. Stored procedures implement many tasks as a series of Transact-SQL statements. Conditional logic can be applied to the results of the first Transact-SQL statements to determine which subsequent Transact-SQL statements will be executed. All of these Transact-SQL statements and conditional logic become part of a single execution plan on the server.

Network traffic is reduced because fewer packets are required to send requests.

- Reduce network traffic. Rather than sending hundreds of Transact-SQL statements over the network, users can perform a complex operation by sending a single statement, which reduces the number of requests that pass between the client and server.

Introduction to Triggers

Slide Objective
To introduce the concept of a trigger.

Lead-in
A trigger is a special type of stored procedure.

- **A Trigger Is a Special Type of Stored Procedure**
- **A Trigger Is:**
  - Associated with a table
  - Invoked automatically
  - Not called directly
  - Treated as part of the transaction that fired it

Key Points
Emphasize that this topic does *not* attempt to comprehensively address triggers. Refer students to SQL Server Books Online for more information.

A *trigger* is a special type of stored procedure that executes whenever an attempt is made to modify data in a table that the trigger protects.

Triggers are best used to maintain low-level data integrity, *not* to return query results. The primary benefit of triggers is that they can contain complex processing logic. A trigger is:

Associated with a Table Triggers are defined on a specific table, which is referred to as the *trigger table*.

Invoked Automatically When an attempt is made to insert, update, or delete data in a table and a trigger for that particular action has been defined on the table, the trigger executes automatically. It cannot be circumvented.

Not Called Directly Unlike standard system stored procedures, triggers cannot be called directly and do not pass or accept parameters.

Treated as Part of the Transaction That Fired It The trigger and the statement that fires it are treated as a single transaction that can be rolled back from anywhere within the trigger. The statement that invokes the trigger is considered the beginning of an implicit transaction, unless an explicit BEGIN TRANSACTION statement is included. The user that invoked the trigger must also have permission to perform all of the statements on all of the tables. If the trigger fails, then the transaction that called it also fails.

◆ Introduction to User-defined Functions

Slide Objective
To introduce the topics that this section covers.

Lead-in
This section provides an overview of user-defined functions and explains why and how to use them.

- **What Is a User-defined Function?**
- **Creating a User-defined Function**

Key Points
Emphasize that this topic does *not* attempt to comprehensively address user-defined functions. Refer students to SQL Server Books Online for more information.

Functions are subroutines made up of one or more Transact-SQL statements that you can use to encapsulate code for reuse. In addition to a number of system-defined functions that are built-in, SQL Server allows users to create their own user-defined functions.

What Is a User-defined Function?

Slide Objective
To introduce the concept of a user-defined function and to state the advantages of using one.

Lead-in
There are three types of user-defined functions.

- **Scalar Functions**
  - Similar to a built-in function
  - Returns a single data value built by a series of statements
- **Multi-Statement Table-valued Functions**
  - Content like a stored procedure
  - Referenced like a view
- **In-line Table-valued Functions**
  - Similar to a view with parameters
  - Returns a table as the result of single SELECT statement

With SQL Server, you can design your own functions to supplement and extend the system-supplied (built-in) functions. You can use user-defined functions as part of a Transact-SQL query, in the same way that you use system-supplied functions.

A user-defined function takes zero or more input parameters and returns either a scalar value or a table. Input parameters can be any data type except **timestamp**, **cursor**, or **table**. User-defined functions do not support output parameters.

SQL Server 2000 supports three types of user-defined functions:

Scalar Functions

This type of user-defined function returns a single data value of the type defined in a RETURNS clause. The body of the function, defined in a BEGIN-END block, contains the series of Transact-SQL statements that return the value. The return type can be any data type except **text**, **ntext**, **image**, **cursor**, or **timestamp**.

Multi-Statement Table-valued Functions

This type of user-defined function returns a table built by one or more Transact-SQL statements. The function body is defined in a BEGIN-END block and is similar to a stored procedure. Unlike a stored procedure, though, a multi-statement table-valued function can be referenced in the FROM clause of a SELECT statement as if it were a view.

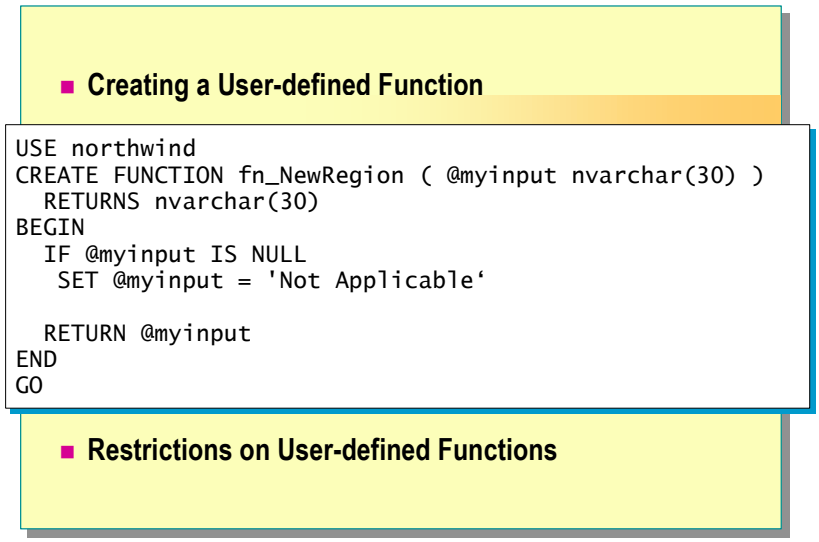
In-line Table-valued Functions

This type of user-defined function returns a table that is the result of a single SELECT statement. An in-line table-valued function provides a representation of data similar to a view. This type of function offers more flexibility than views in the use of parameters and extends the features of indexed views.

Creating a User-defined Function

Slide Objective
To describe the CREATE FUNCTION statement.

Lead-in
You create a user-defined function in much the same way that you create a view or stored procedure.



You create a user-defined function in much the same way that you create a view or stored procedure.

Creating a User-defined Function

You create user-defined functions by using the CREATE FUNCTION statement. Each fully qualified user-defined function name (database_name.owner_name.function_name) must be unique. The statement specifies the input parameters with their data types, the processing instructions, and the value returned with each data type.

Syntax

```
CREATE FUNCTION [ owner_name. ] function_name
  ( [ { @parameter_name scalar_parameter_data_type [ = default ] } [ ,...n ] ] )
  RETURNS scalar_return_data_type
  [ WITH < function_option> [,...n] ]
  [ AS ]
  BEGIN
  function_body
  RETURN scalar_expression
  END
```

Example

This example creates a user-defined function to replace a null value with the words Not Applicable.

```
USE northwind
CREATE FUNCTION fn_NewRegion ( @myinput nvarchar(30) )
  RETURNS nvarchar(30)
BEGIN
  IF @myinput IS NULL
   SET @myinput = 'Not Applicable'

  RETURN @myinput
END
GO
```

When referencing a scalar user-defined function, specify both the function owner and the function name in two-part syntax.

```
SELECT LastName, City
      ,dbo.fn_NewRegion(Region) AS Region
      ,Country
 FROM Employees
GO
```

Result

LastName	City	Region	Country
Davolio	Seattle	WA	USA
Fuller	Tacoma	WA	USA
Leverling	Kirkland	WA	USA
Peacock	Redmond	WA	USA
Buchanan	London	Not Applicable	UK
Suyama	London	Not Applicable	UK
King	London	Not Applicable	UK
Callahan	Seattle	WA	USA
Dodsworth	London	Not Applicable	UK

Restrictions on User-defined Functions

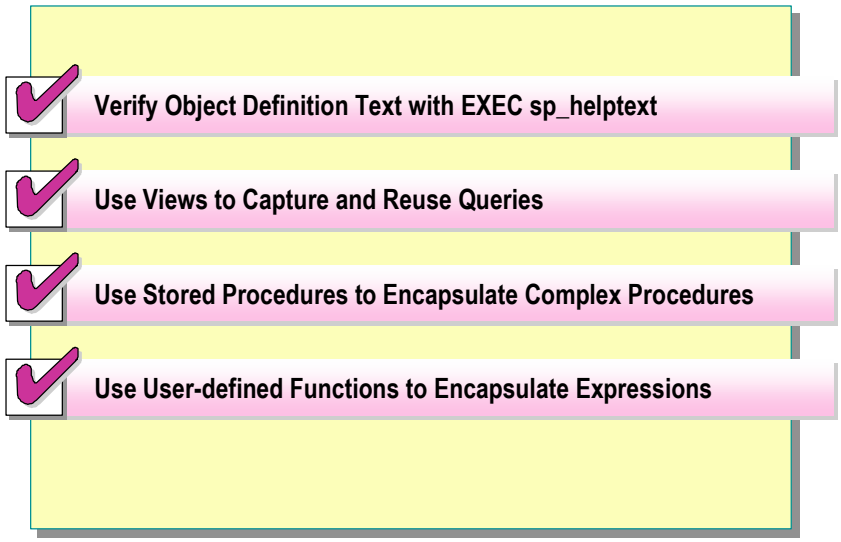
Nondeterministic functions are functions such as GETDATE() that could return different result values each time that they are called with the same set of input values. Built-in nondeterministic functions are not allowed in the body of user-defined functions. These built-in functions from other categories are always nondeterministic:

@@ERROR	FORMATMESSAGE	IDENTITY	USER_NAME
@@IDENTITY	GETANSINULL	NEWID	@@ERROR
@@ROWCOUNT	GETDATE	PERMISSIONS	@@IDENTITY
@@TRANCOUNT	GetUTCDate	SESSION_USER	@@ROWCOUNT
APP_NAME	HOST_ID	STATS_DATE	@@TRANCOUNT
CURRENT_TIMESTAMP	HOST_NAME	SYSTEM_USER	
CURRENT_USER	IDENT_INCR	TEXTPTR	
DATENAME	IDENT_SEED	TEXTVALID	

Recommended Practices

Slide Objective
To present recommended practices for programming objects.

Lead-in
The following are recommended practices for using programming objects.



The following recommended practices should help you use programming objects:

- Verify object definitions by displaying the text associated with the object by using the **EXEC sp_helptext** system stored procedure.
- Use views to capture and reuse common queries for consistency and efficiency.
- Use stored procedures to encapsulate complex multi-statement procedures.
- Use user-defined functions to encapsulate common expressions.

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

Topic	Search on
Using stored procedures	"effects of stored procedures on application performance"
Using views	"comparison of queries and views"
Using triggers	"enforcing business rules with triggers"

Lab A: Working with Views

Slide Objective
To introduce the lab.

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

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Generate a view by using a SQL Query Analyzer template.
- Alter a view by using the Object Browser in SQL Query Analyzer.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2071A\Labfiles\L09.
- Answer files for this lab, which are located in C:\Moc\2071A\Labfiles\L09\Answers.
- The **library** database installed.

For More Information

If you require help, search SQL Query Analyzer Help for "Using Templates in SQL Query Analyzer."

Other resources that you can use include:

- The **library** 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
Generating a View by Using a SQL Query Analyzer Template

In this exercise, you will use a SQL Query Analyzer template to create a view and assign values to the parameters in the view. C:\Moc\2071A\Labfiles\L09\Answers contains completed scripts for this exercise.

► To create a Transact-SQL statement from a SQL Query Analyzer template

In this procedure, you will create a Transact-SQL statement by using a SQL Query Analyzer template. Answer_Template1.sql is a completed script for this step.

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 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 Windows 2000 local group, Administrators. All members of this group are automatically mapped to the SQL Server **sysadmin** role.

3. In the **DB** list, click **library**.
4. On the **Edit** menu, click **Insert Template**.
5. In the **Insert Template** dialog box, double-click **Create View**, and then open Create View Basic Template.tql.
6. Review the contents of the file in the edit pane.

   How does the generated script differ from standard Transact-SQL?

   The generated script has template parameters in place of expected identifier names, in the format of <*parameter_name*, *data_type*, *value*>.

► To replace template parameters

In this procedure, you will replace template parameters. Answer_Template2.sql is a completed script for this step.

1. On the **Edit** menu, click **Replace Template Parameters**.
2. In the **Replace Template Parameters** dialog box, in the **Value** column, type **AustenTitlesView** to change the **view_name** parameter.
3. In the **Replace Template Parameters** dialog box, in the **Value** column, type the following statement to change the **view_name** parameter:

   ```
   SELECT * FROM title WHERE author = 'Jane Austen'
   ```

4. On the **Edit** menu, click **Replace All** to replace each of the occurring parameters with the newly assigned values.

► To validate the syntax and create the view

In this procedure, you will validate and execute Transact-SQL statements to create the view. Answer_View1.sql is a completed script for this step.

1. On the **Query** menu, click **Parse**.

   This verifies that the syntax of the newly constructed Transact-SQL script is valid.
2. In the edit pane, create the view by executing the script.
3. On the toolbar, click the **New Query** button to open a new Query window.

   This opens a new connection to SQL Server.
4. Write a query that uses an asterisk in the SELECT statement to retrieve all of the data in the **AustenTitlesView** view.

   ```
   USE library
   SELECT *
    FROM AustenTitlesView
   GO
   ```

5. Execute the query to verify that it returns the desired results.

   What columns does querying the view return?

   The columns are title_no, title, author, and synopsis.

   _____

   _____

   What are the advantages of using SQL Query Analyzer templates for generating Transact-SQL scripts?

   Using SQL Query Analyzer templates for generating Transact-SQL scripts reduces repetitive typing and errors and fosters consistency through reuse of standard code and syntax.

   _____

   _____

Exercise 2
Altering a View by Using the Object Browser in SQL Query Analyzer

In this exercise, you will use the Object Browser in SQL Query Analyzer to script and edit Transact-SQL statements that alter the view that you created in exercise 1. C:\Moc\2071A\Labfiles\L09\Answers contains completed scripts for this exercise.

► To display the Object Browser

- On the **Tools** menu, select **Object Browser**, and then click **Show/Hide**.

  This displays the Object Browser pane.

► To locate the dbo.AustenTitlesView object

In this procedure, you will locate the **dbo.AustenTitlesView** object.

1. In the Object Browser pane, expand **library**.
2. Expand **Views**, and then expand the **dbo.AustenTitlesView** object.

► To generate a script to alter the view

In this procedure, you will generate a script that alters a view. Answer_View2.sql is a completed script for this step.

1. Right-click **dbo.AustenTitlesView**.
2. Select **Script Object to New Window as**.
3. Click **Alter**.

   This creates a series of Transact-SQL statements that alter the **dbo.AustenTitlesView** object, which was created in a new edit pane.

   What other types of scripts can you create that relate to this view object?

   You can create scripts that act on the view object that create, alter, and drop. You can also create scripts that perform operations by using the object, such as SELECT, INSERT, UPDATE, and DELETE.

► To copy a column list from the Object Browser

In this procedure, you will copy a column list from the Object Browser. Answer_View3.sql is a completed script for this step.

1. On the **Query** menu, click **Parse**.

   This verifies that the syntax of the newly constructed Transact-SQL script is valid.

2. Expand **dbo.AustenTitlesView**, and then expand **Columns**.
3. Drag the **Columns** folder from the Object Browser pane to the edit pane, and then place the folder in the SELECT statement of the view definition after the asterisk.
4. Delete the asterisk in the SELECT statement.

► To copy an object name from the Object Browser

In this procedure, you will copy an object name from the Object Browser. Answer_View4.sql is a completed script for this step.

1. In the edit pane, delete the **AustenTitlesView** view name that immediately follows the ALTER VIEW statement.
2. Drag the **dbo.AustenTitlesView** object from the Object Browser pane to the edit pane, and then place the object after the ALTER VIEW statement.

► To validate the syntax and alter the view

In this procedure, you will validate the syntax and alter the view. Answer_View5.sql is a completed script for this step.

1. On the **Query** menu, click **Parse**.

   This verifies that the syntax of the newly constructed Transact-SQL script is valid.
2. Execute the script in the edit pane to alter the view.
3. On the toolbar, click the **New Query** button to open a new Query window.

   This opens a new connection to SQL Server.
4. Write a SELECT statement to retrieve all of the data in the **dbo.AustenTitlesView** view.

   ```
   USE library
   SELECT title_no, title, author, synopsis
    FROM dbo.AustenTitlesView
   GO
   ```
5. Execute the query to verify that it returns the desired results.

   Why should you enumerate all columns when constructing Transact-SQL statements?

   You should always explicitly reference column names to ensure their intended use and relative order. The SELECT * syntax retrieves columns in the order in which they are defined. This order may differ from table to table, or may subsequently change.

Review

Slide Objective
To reinforce module objectives by reviewing key points.

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

- Displaying the Text of a Programming Object
- Introduction to Views
- Advantages of Views
- Creating Views
- Introduction to Stored Procedures
- Introduction to Triggers
- Introduction to User-defined Functions

Use these questions to review module topics.

Ask students whether they have any questions before continuing.

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; and views provide a security mechanism by allowing users access to data in views only.

2. What are the benefits of stored procedures?

   Encapsulation of shared application logic, improved performance, better security management, and reduced network traffic.

3. Why would you use a view instead of a stored procedure to encapsulate a query?

   When you use a view to encapsulate a query, you can readily reuse the view as part of a SELECT statement when writing another query.

4. You have developed a query that joins the **member**, **title**, and **loanhist** tables to list the fines that are assessed, paid, and waived for each member and the title of each overdue book and to calculate the number of days that each book is overdue. Other developers are interested in leveraging the work that you have done to build their own queries. How can you best accomplish this?

   Create a view on your query. Have the other developers write their queries or views to execute against your view. This ensures that all queries return consistent results and that the work and business knowledge encapsulated in the original view does not have to be recreated.

5. Describe the three types of user-defined functions.

   Scalar functions are similar to built-in functions.

   Multi-statement table-valued functions are similar to stored procedures.

   In-line table-valued functions are similar to views.

6. How is a trigger different from a stored procedure?

   Triggers are a special type of stored procedure that is attached to a table and executed automatically whenever an attempt is made to modify data in the table. Triggers cannot be called directly and do not accept parameters.