MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 4: Creating Data Types and Tables

Contents

Microsoft

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:
30 Minutes

Lab:
30 Minutes

This module provides students with a description of how to create data types and tables, and generate Transact-SQL scripts containing statements that create a database and its objects.

In the lab, students create the **ClassNorthwind** database data types and tables, add and drop columns, and then generate a script to recreate all database objects that were previously created.

After completing this module, students will be able to:

- Create and drop user-defined data types.
- Create and drop user tables.
- Generate column values.
- Generate a script.

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:

- Microsoft® PowerPoint® file 2073A__04.ppt
- The C:\Moc\2073A\Demo\D04_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:

- Creating Data Types

  Describe the system-supplied data types and the guidelines for creating user-defined data types. Explain how to create and drop user-defined data types.

- Creating Tables

  Describe the process of creating, modifying, and dropping a table.

- Generating Column Values

  Discuss the **Identity** property, the NEWID function, and the **uniqueidentifier** data type so that the students will be able to define and use these capabilities to generate values automatically.

- Generating Scripts

  Describe how to generate a Transact-SQL script containing the statements that are used to create a database and its objects.

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 1

The lab in this module requires restoring the **ClassNorthwind** database to 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\Restore04.cmd batch file.

Warning If this course has been customized, students must execute the C:\Moc\2073A\Batches\Restore04.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 about creating data types and tables, and generating scripts.

- **Creating Data Types**
- **Creating Tables**
- **Generating Column Values**
- **Generating Scripts**

This module describes how to create data types and tables, and generate Transact-SQL scripts containing statements that create a database and its objects.

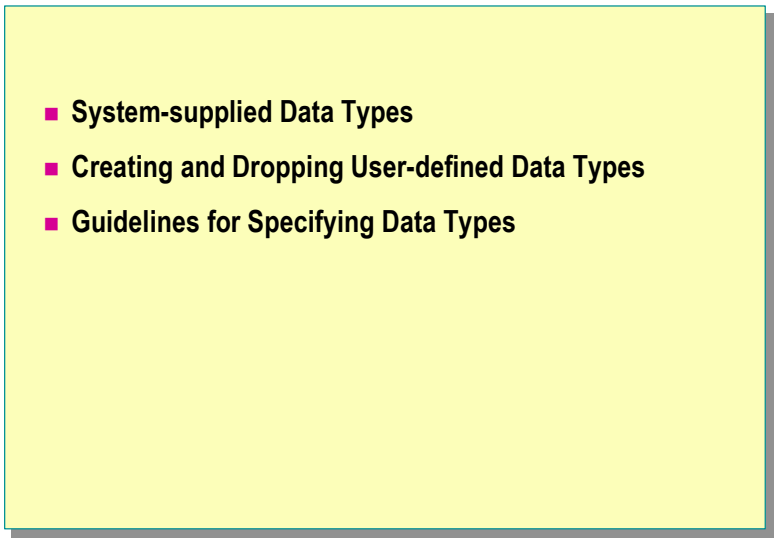
After completing this module, students will be able to:

- Create and drop user-defined data types.
- Create and drop user tables.
- Generate column values.
- Generate scripts.

Creating Data Types

Topic Objective
To provide an overview of this topic.

Lead-in
In this section, you'll learn how to create and drop user-defined data types.



Before you can create a table, you must define the data types for the table. Data types specify the type of information (characters, numbers, or dates) that a column can hold, as well as how the data is stored. Microsoft® SQL Server™ 2000 supplies various system data types. SQL Server also allows user-defined data types that are based on system data types.

System-supplied Data Types

Topic Objective
To list the SQL Server data types.

Lead-in
SQL Server provides a number of different data types.

- **Numeric**
  - Integer
  - Exact numeric
  - Approximate numeric
  - Monetary
- **Date and Time**
- **Character and Unicode Character**
- **Binary**
- **Other**

Data types define the data value allowed for each column. SQL Server provides a number of different data types. Certain common data types have several associated SQL Server data types. You should choose appropriate data types to optimize performance and conserve disk space.

Categories of System-supplied Data Types

The following table maps common types of data to SQL Server system-supplied data types. The table includes data type synonyms for ANSI compatibility.

Common data types	SQL Server system-supplied data types	ANSI synonym	Number of bytes
Integer	**int**	*integer*	4
	bigint	–	8
	smallint, **tinyint**	–	2, 1
Exact numeric	**decimal[(p[, s])]**	*dec*	2–17
	numeric[(p[, s])]	–	
Approximate numeric	**float[(n)]**	*double precision*, *float*[(*n*)] for *n=8-15*	8
	real	*float*[(*n*)] for *n=1-7*	4
Monetary	**money**, **smallmoney**	–	8, 4
Date and time	**Datetime**, **smalldatetime**	–	8 4

Delivery Tip
SQL Server can support multiple languages by storing strings in Unicode (double-byte) data type fields.

(*continued*)

Common data types	SQL Server system-supplied data types	ANSI synonym	Number of bytes
Character	**char[(n)]**	*character*[(*n*)]	0–8000
	varchar[(n)]	*char VARYING*[(*n*)]	
		character VARYING[(*n*)]	
	text	–	0–2 GB
Unicode character	**nchar[(n)]**	–	0–8000
	nvarchar[(n)]		(4000 characters)
	ntext		0–2 GB
Binary	**binary[(n)]**	–	0–8000
	varbinary[(n)]	*binary VARYING*[(*n*)]	
Image	**image**	–	0–2 GB
Global identifier	**uniqueidentifier**	–	16
Special	**bit**, **cursor**, **uniqueidentifier**	–	1, 0–8
	timestamp	**rowversion**	8
	sysname	–	256
	table	–	
	sql_variant	–	0–8016

Delivery Tip
Point out the **cursor**, **table**, and **sql_variant** data types.

Exact and Approximate Numeric Data Types

How you plan to use a data type should determine whether you choose an exact numeric or an approximate numeric data type.

Exact Numeric Data Types

Exact numeric data types let you specify *exactly* the scale and precision to use. For example, you can specify three digits to the right of the decimal and four to the left. A query always returns exactly what you entered. SQL Server supports two exact numeric data types for ANSI compatibility: **decimal** and **numeric**.

In general, you would use exact numeric data types for financial applications in which you want to portray the data consistently (always two decimal places) and to query on that column (for example, to find all loans with an interest rate of 8.75 percent).

Approximate Numeric Data Types

Approximate numeric data types store data as accurately as possible. For example, the fraction 1/3 is represented in a decimal system as .33333 (repeating). The number cannot be stored accurately, so an approximation is stored. SQL Server supports two approximate data types: **float** and **real**.

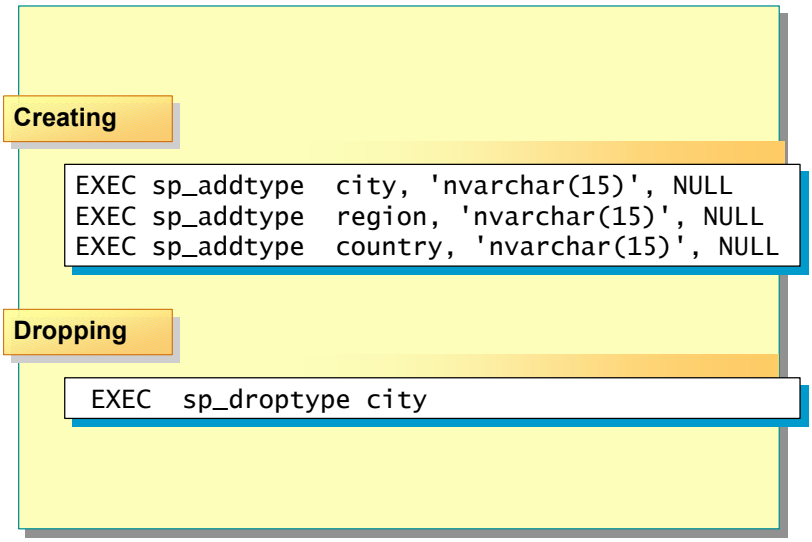
If you are rounding numbers or performing quality checks between values, you should avoid using approximate numeric data types.

Note It is best to avoid referencing columns with the **float** or **real** data type in WHERE clauses.

Creating and Dropping User-defined Data Types

Topic Objective
To explain how to add and drop user-defined data types.

Lead-in
You can create and drop user-defined data types by using SQL Server Enterprise Manager or system stored procedures.



Delivery Tip
Demonstrate creating a data type by using SQL Server Enterprise Manager.

User-defined data types are based on system-supplied data types. They allow you to refine data types further to ensure consistency when working with common data elements in different tables or databases. A user-defined data type is defined for a specific database.

Note User-defined data types that you create in the **model** database are automatically included in all databases that are subsequently created. Each user-defined data type is added as a row in the **systypes** table.

You can create and drop user-defined data types by using SQL Server Enterprise Manager or system stored procedures. Data type names must follow the rules for identifier names and must be unique to each database. Define each user-defined data type in terms of a system-supplied data type, preferably by specifying NULL or NOT NULL.

Creating a User-defined Data Type

The **sp_addtype** system stored procedure creates user-defined data types.

Syntax

sp_addtype {*type*}**,** [*system_data_type*] [, ['NULL' | 'NOT NULL']] [, '*owner_name*']

Example

The following example creates three user-defined data types.

```
EXEC  sp_addtype  city, 'nvarchar(15)', NULL
EXEC  sp_addtype  region, 'nvarchar(15)', NULL
EXEC  sp_addtype  country, 'nvarchar(15)', NULL
```

Dropping a User-Defined Data Type

The **sp_droptype** system stored procedure deletes user-defined data types from the **systypes** system table. A user-defined data type cannot be dropped if tables or other database objects reference it.

Syntax

sp_droptype {'*type'*}

Example

The following example drops a user-defined data type.

```
EXEC sp_droptype city
```

Note Execute the **sp_help** system stored procedure to retrieve a list of currently defined data types.

Guidelines for Specifying Data Types

Topic Objective
To present some guidelines for selecting data types.

Lead-in
When selecting data types, balance storage size with requirements.

- **If Column Length Varies, Use a Variable Data Type**
- **Use tinyint Appropriately**
- **For Numeric Data Types, Commonly Use decimal**
- **If Storage Is Greater Than 8000 Bytes, Use text or image**
- **Use money for Currency**
- **Do Not Use float or real as Primary Keys**

Consider the following guidelines for selecting data types and balancing storage size with requirements:

- If column length varies, use one of the variable data types. For example, if you have a list of names, you can set it to **varchar** instead of **char** (fixed).
- If you own a growing bookselling business with many locations, and you have specified the **tinyint** data type for the store identifier in the database, you will have problems when you decide to open store number 256.
- For numeric data types, the size and required level of precision helps to determine your choice. In general, use **decimal**.
- If the storage is greater than 8000 bytes, use **text** or **image**. If it is less than 8000, use **binary** or **char**. When possible, it is best to use **varchar** because it has more functionality than **text** and **image**.
- Use the **money** data type for currency.
- Do not use the approximate data types **float** and **real** as primary keys. Because the values of these data types are not precise, it is not appropriate to use them in comparisons.

Creating Tables

Topic Objective
To provide an overview of this topic.

Lead-in
In this section, you'll learn how to define all of the data types for a table, create tables, create and drop columns, and generate column values.

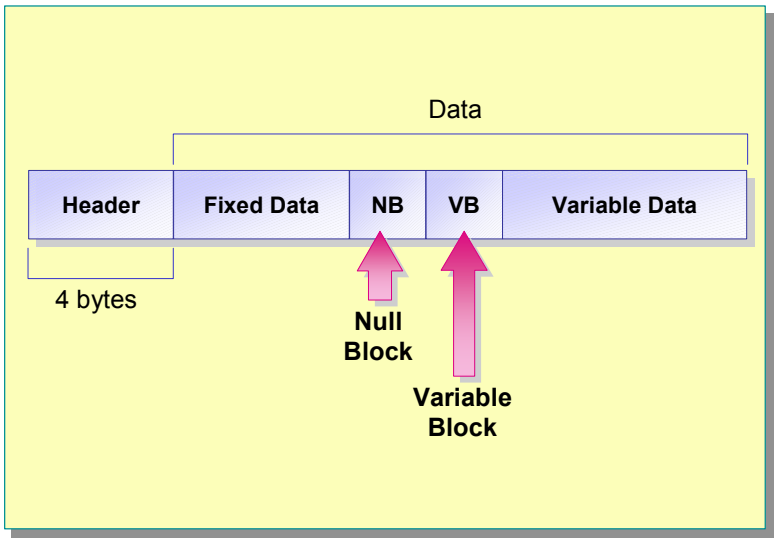
- How SQL Server Organizes Data in Rows
- How SQL Server Organizes text, ntext, and image Data
- Creating and Dropping a Table
- Adding and Dropping a Column

After you define all of the data types for your table, you can create tables, add and drop columns, and generate column values.

How SQL Server Organizes Data in Rows

Topic Objective
To describe how data is organized in rows.

Lead-in
It is important to understand the elements of the data portion of each row to accurately estimate the size of a table.



A data row consists of a row header and a data portion. It is important to understand the elements of the data portion of each row to accurately estimate the size of a table.

Row Header

The 4-byte row header contains information about the columns in the data row, such as a pointer to the location of the end of the fixed-data portion of the row, and whether variable-length columns exist in the row.

Data Portion

The data portion of a row may contain the following elements:

- *Fixed-length data*. Fixed-length data is entered into the page before variable-length data. An empty fixed-length data row takes up as much space as a populated fixed-length data row. A table with only fixed-length columns always stores the same number of rows on a page.
- *Null block*. A null block is a variable-length set of bytes. It consists of two bytes storing the number of columns followed by a null bitmap indicating whether each individual column is null. The size of a null bitmap is equal to one bit per column, rounded up to the nearest byte. One to eight columns require a 1-byte bitmap. Nine to sixteen columns require a 2-byte bitmap.

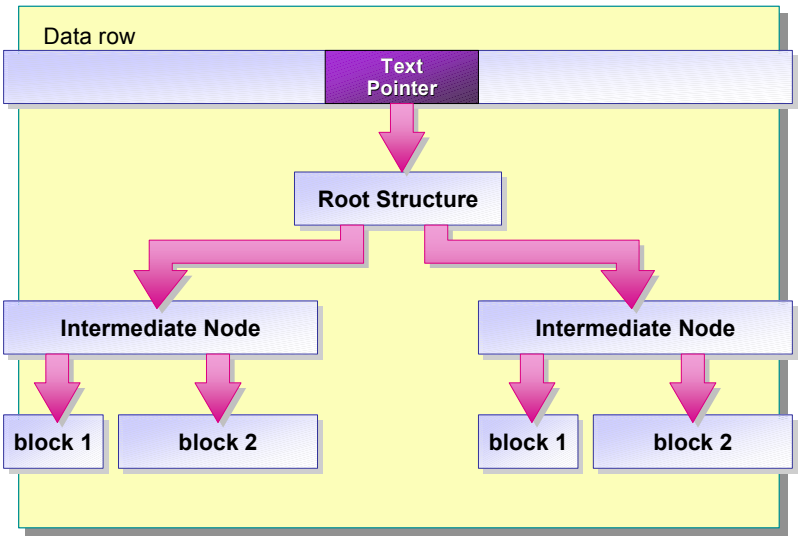
- *Variable block*. A variable block consists of two bytes that describe how many variable-length columns are present. An additional two bytes per column point to the end of each variable-length column. The variable block is omitted if there are no variable-length columns.
- *Variable-length data*. Variable-length data is entered into the page after the variable block. An empty variable-length data row takes up no space. A table with variable-length columns may have a few long rows or many short rows.

Tip When possible, keep row length compact to allow more rows to fit on a page. This reduces input/output (I/O) and improves the buffer cache hit ratio.

How SQL Server Organizes text, ntext, and image Data

Topic Objective
To demonstrate how three data types are an exception to row organization.

Lead-in
Variable-length data types are usually stored as one collection of pages, rather than in data rows.



Variable-length data types can be stored as one collection of pages or in data rows. They are the:

- **text** data type, which can hold 2,147,483,647 characters. The non-Unicode **text** data type cannot be used for variables or parameters in stored procedures.
- **ntext** data type, which can hold a maximum of 2^{30} -1 (1,073,741,823) characters or 2^{31} -1 bytes, which is 2,147,483,647 bytes of variable-length Unicode data. The SQL-92 synonym for **ntext** is national text.
- **image** data type, which can hold from 0 through 2,147,483,647 bytes of binary data.

Because **text**, **ntext**, and **image** data types are usually large, SQL Server stores them outside of rows. A 16-byte pointer in the data row points to a root structure that holds the data. The text root structure forms the root node of the B-Tree, which points to the data blocks. If there are more than 32 kilobytes (KB) of data, intermediate nodes in the B-Tree are added between the root node and the blocks of data. This permits quick B-Tree navigation starting in the middle of a string.

With small- to medium-sized **text**, **ntext**, and **image** content, SQL Server provides the option to store values in the data row rather than in a separate B-Tree structure. You can specify this **text in row** option. You can also set the option limit; the range is 24 through 7,000 bytes.

You can enable the **text in row** option for a table by using the **sp_tableoption** system stored procedure.

Example

This example sets the **sp_tableoption** system stored procedure **text in row** option ON and specifies that up to 1000 **text**, **ntext**, or **image** characters will be stored in the data page.

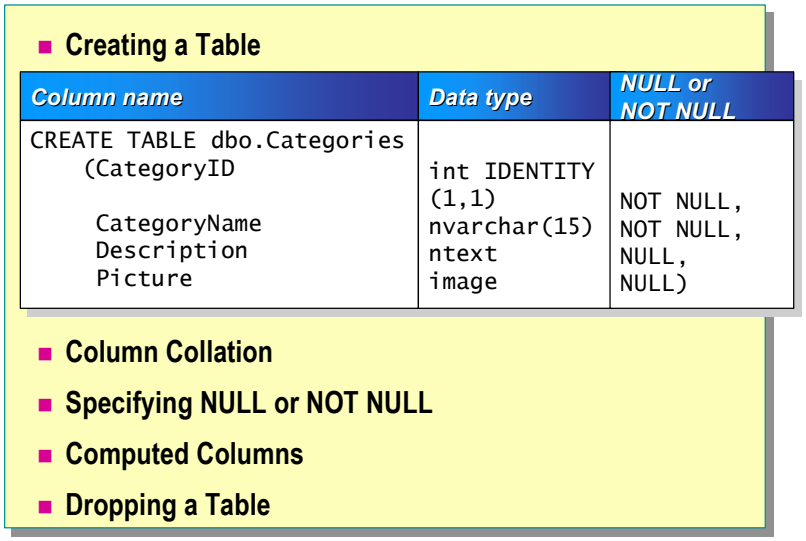
```
EXEC sp_tableoption N'Employees', 'text in row', '1000'
```

Note If you specify ON, but do not specify a value, the default is 256 bytes. The default value ensures that small values and text pointers can be stored in the data rows.

Creating and Dropping a Table

Topic Objective
To explain how to create and drop tables.

Lead-in
When you create a table, you must specify the table name, column names, and data types.



Delivery Tip
Demonstrate creating a table by using SQL Server Enterprise Manager.

When you create a table, you must specify the table name, column names, and column data types. Column names must be unique to a specific table, but you can use the same column name in different tables within the same database. You must specify a data type for each column.

Creating a Table

Consider the following facts when you create tables in SQL Server. You can have up to:

- Two billion tables per database.
- 1,024 columns per table.
- 8060 bytes per row (this approximate maximum length does not apply to **image**, **text**, and **ntext** data types).

Column Collation

SQL Server supports storing objects with different collations in the same database. Separate SQL Server collations can be specified at the column-level, so that each column in a table can be assigned a different collation.

Specifying NULL or NOT NULL

You can specify in the table definition whether to allow null values in each column. If you do not specify NULL or NOT NULL, SQL Server provides the NULL or NOT NULL characteristic, based on the session- or database-level default. However, these defaults can change, so do not rely on them. NOT NULL is the SQL Server default.

Partial Syntax

CREATE TABLE *table_name*
column_name data type [COLLATE<*collation_name*>]
[NULL | NOT NULL]
| *column_name* AS *computed_column_expression*
[,….*n*]

Example

The following example creates the **dbo.CategoriesNew** table, specifying the columns of the table, a data type for each column, and whether that column allows null values.

```
CREATE TABLE dbo.CategoriesNew
   (CategoryID     int IDENTITY
                   (1, 1)           NOT NULL,
   CategoryName    nvarchar(15)     NOT NULL,
   Description     ntext            NULL,
   Picture         image            NULL)
```

Note To view table properties, right-click a table in SQL Server Enterprise Manager, or execute the **sp_help** system stored procedure and then scroll to the right.

Computed Columns

A *computed column* is a virtual column that is not physically stored in the table. SQL Server uses a formula that you create to calculate this column value by using other columns in the same table. Using a computed column name in a query can simplify the query syntax.

Dropping a Table

Dropping a table removes the table definition and all data, as well as the permission specifications for that table.

Before you can drop a table, you should remove any dependencies between the table and other objects. To view existing dependencies, execute the **sp_depends** system stored procedure.

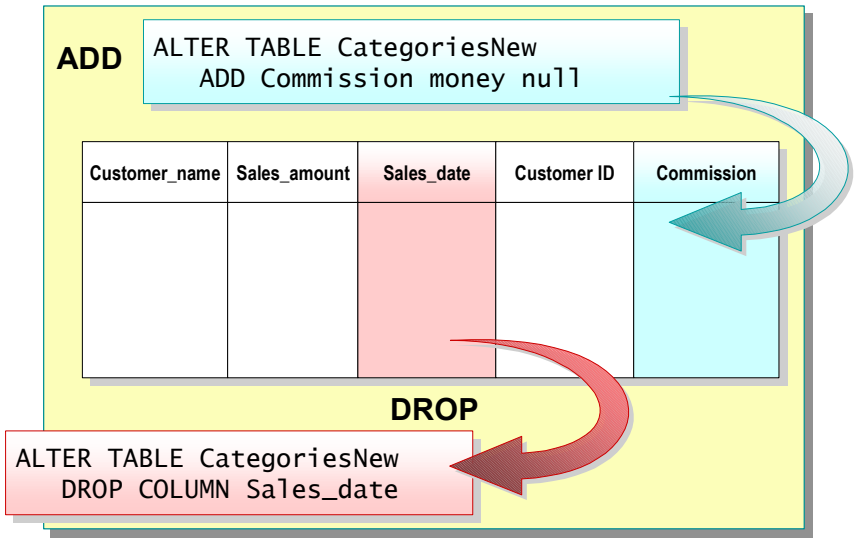
Syntax

DROP TABLE *table_name* [,…*n*]

Adding and Dropping a Column

Topic Objective
To show how to add and drop columns.

Lead-in
As requirements change, you may have to modify tables by adding or dropping columns.



Adding and dropping columns are two ways to modify tables.

Partial Syntax

```
ALTER TABLE table
    {| [ALTER COLUMN column_name ]
        |{  ADD
                {   <column_definition> ::=
                        column_name data_type
                        {   [NULL | NOT NULL]
            | DROP column column_name} [,…n]
```

Adding a Column

The type of information that you specify when you add a column is similar to that which you supply when you create a table.

Example

This example adds a column that allows null values.

```
ALTER TABLE CategoriesNew
 ADD Commission money null
```

Dropping a Column

Dropped columns are unrecoverable. Therefore, be certain that you want to remove a column before doing so.

Example

This example drops a column from a table.

```
ALTER TABLE CategoriesNew
 DROP COLUMN Sales_date
```

Note All indexes and constraints that are based on a column must be removed before you drop the column.

◆ Generating Column Values

Topic Objective
To list the topics in this section.

Lead-in
This section describes how to generate column values.

- **Using the Identity Property**
- **Using the NEWID Function and the uniqueidentifier Data Type**

Several features allow you to generate column values: the **Identity** property, the NEWID function, and the **uniqueidentifier** data type.

Using the Identity Property

Topic Objective
To explain how to use the **Identity** property.

Lead-in
An identity column is often used for primary key values.

- **Requirements for Using the Identity Property**
  - Only one identity column is allowed per table
  - Use with **integer**, **numeric**, and **decimal** data types
- **Retrieving Information About the Identity Property**
  - Use IDENT_SEED and IDENT_INCR for definition information
  - Use **@@identity** to determine most recent value
- **Managing the Identity Property**

Delivery Tip
If you do not use auto-incrementing, you must query to find the next value.

You can use the **Identity** property to create columns (referred to as identity columns) that contain system-generated sequential values identifying each row inserted into a table. An identity column is often used for primary key values.

Having SQL Server automatically provide key values can reduce costs and improve performance. It simplifies programming, keeps primary key values short, and reduces user-transaction bottlenecks.

Partial Syntax

```
CREATE TABLE table
  (column_name data_type
   [ IDENTITY [(seed, increment)]] NOT NULL )
```

Consider the following requirements for using the **Identity** property:

- Only one identity column is allowed per table.
- It must be used with **integer** (**int**, **bigint**, **smallint**, or **tinyint**), **numeric**, or **decimal** data types. The **numeric** and **decimal** data types must be specified with a scale of 0.
- It cannot be updated.
- You can use the IDENTITYCOL keyword in place of the column name in a query. This allows you to reference the column in the table having the **Identity** property without having to know the column name.
- It does not allow null values.

When you determine what data type to use when defining the column, by using the **Identity** property, try to estimate the number of rows that the table will contain.

You can retrieve information about the **Identity** property in several ways:

- Two system functions return information about an identity column definition: IDENT_SEED (returns the seed, or starting, value) and IDENT_INCR (returns the increment value).
- You can retrieve data from identity columns using the **@@identity** global variable, which determines the value of the last row inserted into an identity column during a session.
- SCOPE_IDENTITY returns the last IDENTITY value inserted into an indentity column in the same scope. A scope is a stored procedure, trigger, function, or batch.
- IDENT_CURRENT returns the last identity value generated for a specified table in any session and any scope.

You can manage the **Identity** property in several ways:

- You can allow explicit values to be inserted into the identity column of a table by setting the IDENTITY_INSERT option ON. When IDENTITY_INSERT is ON, INSERT statements must supply a value.
- To check and possibly correct the current identity value for a table, you can use the DBCC CHECKIDENT statement. DBCC CHECKIDENT allows you to compare the current identity value with the maximum value in the identity column.

Note The **Identity** property does not enforce uniqueness. To enforce uniqueness, create a unique index.

Example

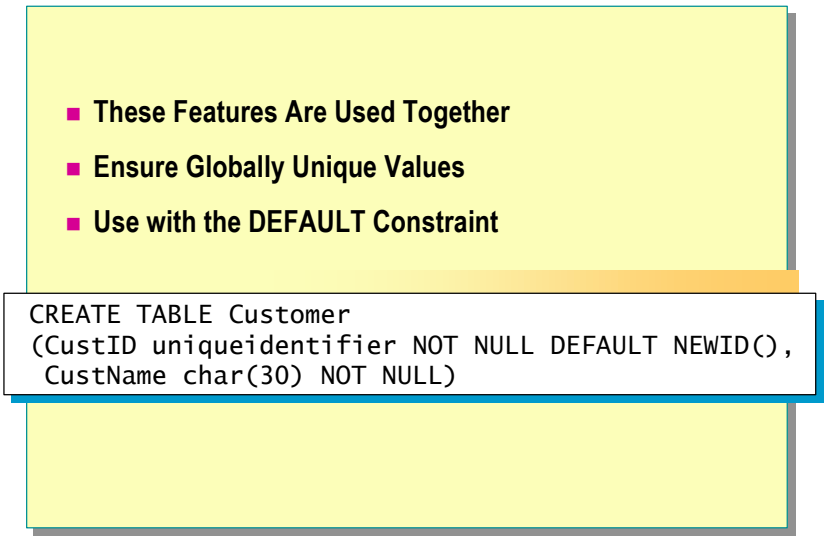
This example creates a table with two columns, **StudentId** and **Name**. The **Identity** property is used to increment the value automatically in each row added to the **StudentId** column. The seed is set to 100, and the increment value is 5. The values in the column would be 100, 105, 110, 115, and so on. Using 5 as an increment value allows you to insert records between the values at a later time.

```
CREATE TABLE Class
  (StudentID int  IDENTITY(100, 5) NOT NULL,
  Name varchar(16))
```

Using the NEWID Function and the uniqueidentifier Data Type

Topic Objective
To describe how to use these features.

Lead-in
The **uniqueidentifier** data type and the NEWID function are two features that are used together.



The **uniqueidentifier** data type and the NEWID function are two features that are used together. Use these features when data is collated from many tables into a larger table and when uniqueness among all records must be maintained:

- The **uniqueidentifier** data type stores a unique identification number as a 16-byte binary string. This data type is used for storing a globally unique identifier (GUID).
- The NEWID function creates a unique identifier number that can store a GUID by using the **uniqueidentifier** data type.
- The **uniqueidentifier** data type does not automatically generate new IDs for inserted rows the way the **Identity** property does. To get new **uniqueidentifier** values, you must define a table with a DEFAULT constraint that specifies the NEWID function. When you use an INSERT statement, you must also specify the NEWID function.

Example

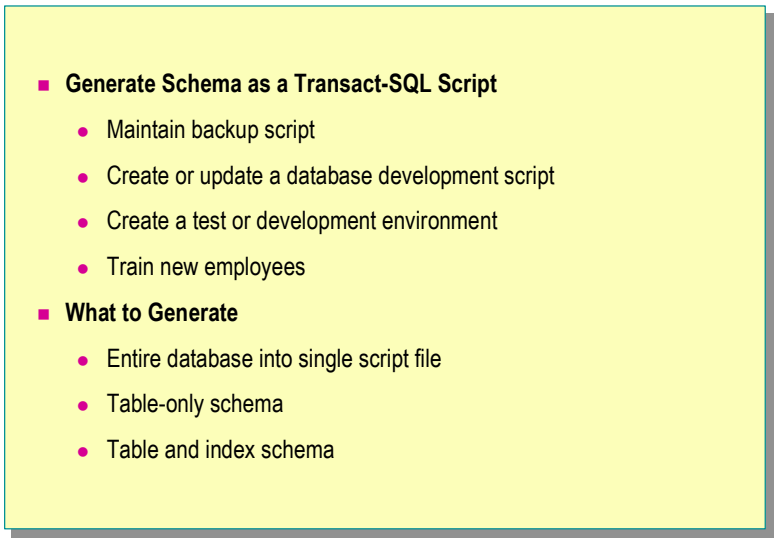
In this example, the **Customer** table customer ID column is created with a **uniqueidentifier** data type, with a default value generated by the NEWID function. A unique value for the **CustID** column will be generated for each new and existing row.

```
CREATE TABLE Customer
(CustID uniqueidentifier NOT NULL DEFAULT NEWID(),
  CustName char(30) NOT NULL)
```

Generating Scripts

Topic Objective
To describe how to generate a script file.

Lead-in
When you create objects in a database, it is important to save all object definitions in a script file.



Delivery Tip
Demonstrate generating a script using the **Northwind** database.

When you create objects in a database, it is important to save all object definitions in a script file.

Generate Schema as a Transact-SQL Script

You can use SQL Server Enterprise Manager to document an existing database structure (schema) by generating it as one or more Transact-SQL scripts. These Transact-SQL scripts contain descriptions of the statements that were used to create a database and its objects.

Schema generated as Transact-SQL scripts can be used to:

- Maintain a backup script that allows a user to recreate all users, groups, logins, and permissions.
- Create or update a database development script.
- Create a test or development environment from an existing schema.
- Train newly hired employees.

What to Generate

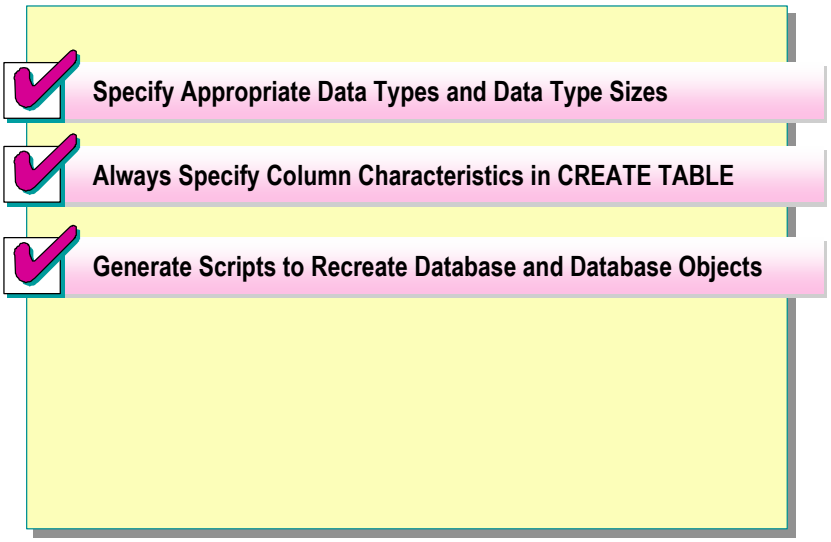
You can generate:

- An entire database into a single script file.
- Table-only schema for one, some, or all tables in a database into one or more script files.
- Table and index schema into one script file, stored procedures into another script file, and defaults and rules into yet another script file.

Recommended Practices

Topic Objective
To present recommended practices for creating data types and tables.

Lead-in
The following are recommended practices for creating data types and tables.



The following recommended practices will help you create data types and tables:

- Specify appropriate data types and data type sizes.
- When writing a CREATE TABLE statement, always specify column characteristics.
- After you create a database and database objects, generate a script that allows you to recreate the database and its objects.

Lab A: Creating Data Types and Tables

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will create user-defined data types, create tables, add and drop columns, and generate scripts.

Objectives

After completing this lab, you will be able to:

- Create user-defined data types.
- Create tables.
- Add and drop columns.
- Generate Transact-SQL scripts from a database.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2073A\Labfiles\L04.
- Answer files for this lab, which are located in C:\Moc\2073A\Labfiles\L04\Answers.

Lab Setup

To complete this lab, you must have either:

- Completed the prior lab, or
- Executed the C:\Moc\2073A\Batches\Restore04.cmd batch file.

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

For More Information

If you require help in 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 User-defined Data Types

In this exercise, you will create user-defined data types for the **ClassNorthwind** database.

► To execute a script that creates a user-defined data type

In this procedure, you will execute a script to create a user-defined data type in the **ClassNorthwind** database.

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 Microsoft 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 **ClassNorthwind**.
4. Open and review the Creatyp1.sql script file in C:\Moc\2073A\Labfiles\L04.

   This script creates a new data type named **postalcode** that contains up to 10 bytes of character data and that may be NULL.

5. Execute Creatyp1.sql.
6. Open, review and execute the Vertype.sql script file in C:\Moc\2073A\Labfiles\L04. Verify that the data type was created.

► To create user-defined data types

In this procedure, you will write and execute statements that create user-defined data types in the **ClassNorthwind** database.

A complete Transact-SQL script for this procedure is located in C:\MOC\2073A\Labfiles\L04\Answers\Creatyp2.sql.

1. Verify that you are using the **ClassNorthwind** database.
2. Write and execute statements to create the user-defined data types described in the following table.

Data type	Description of data
City	Up to 15 bytes of character data that may be NULL
Region	Up to 15 bytes of character data that may be NULL
Country	Up to 15 bytes of character data that may be NULL

```
EXEC sp_addtype city, 'nvarchar(15)', NULL
EXEC sp_addtype region, 'nvarchar(15)', NULL
EXEC sp_addtype country, 'varchar(15)', NULL
```

3. Open and execute Vertype.sql to verify that the data types were created.

Exercise 2
Creating Tables in the ClassNorthwind Database

In this exercise, you will create all of the tables for the **ClassNorthwind** database.

► To execute a script that creates a table

In this procedure, you will execute a script that creates the **Employees** table in the **ClassNorthwind** database.

1. Open and review the Creatab1.sql script file in C:\Moc\2073A\Labfiles\L04.

   This script creates the **Employees** table in the **ClassNorthwind** database.

2. Execute Creatab1.sql.

► To create a table by using statements

In this procedure, you will write and execute a statement that creates the **Suppliers** table in the **ClassNorthwind** database.

A complete Transact-SQL script for this procedure is located in C:\Moc\2073A\Labfiles\L04\Answers\ Creatab2.sql.

1. Verify that you are using the **ClassNorthwind** database.
2. Write and execute a statement to create the **Suppliers** table, defining the following column names with their respective data types.

Ensure that the **SupplierID** and **CompanyName** columns do not allow null values, and that all other columns do allow null values.

Column name	Data type	Allows NULLs	Identity property
SupplierID	**int**	No	Seed = 1 Increment = 1
CompanyName	**nvarchar (40)**	No	No
ContactName	**nvarchar (30)**	Yes	No
ContactTitle	**nvarchar (30)**	Yes	No
Address	**nvarchar (60)**	Yes	No
City	**city**	Yes	No
Region	**region**	Yes	No
PostalCode	**postalcode**	Yes	No
Country	**country**	Yes	No
Phone	**nvarchar (24)**	Yes	No
Fax	**nvarchar (24)**	Yes	No
HomePage	**ntext**	Yes	No

```
CREATE TABLE Suppliers
   (SupplierID     int     IDENTITY(1,1)     NOT NULL,
   CompanyName     nvarchar (40)             NOT NULL,
   ContactName     nvarchar (30)             NULL,
   ContactTitle    nvarchar (30)             NULL
   Address         nvarchar (60)             NULL
   City            city                      NULL
   Region          region                    NULL
   PostalCode      postalcode                NULL
   Country         country                   NULL
   Phone           nvarchar (24)             NULL
   Fax             nvarchar (24)             NULL
   HomePage        ntext                     NULL)
```

Which data types are user-defined?

City, region, postalcode, and country.

_____

_____

► To create a table by using SQL Server Enterprise Manager

In this procedure, you will use SQL Server Enterprise Manager to create the **Customers** table in the **ClassNorthwind** database.

1. Open SQL Server Enterprise Manager.
2. Expand Microsoft SQL Servers, expand SQL Server Group, expand your server, and then expand Databases.
3. Right-click **ClassNorthwind**, point to **New**, and then click **Table**.
4. Use the information in the following table to create the **Customers** table, defining the column names with their respective data types.

   Make sure that the **CustomerID** and **CompanyName** columns do not allow null values. Make sure that all other columns allow null values.

Column name	Data type	Allows NULLs?
CustomerID	**nchar (5)**	No
CompanyName	**nvarchar (40)**	No
ContactName	**nvarchar (40)**	Yes
ContactTitle	**nvarchar (30)**	Yes
Address	**nvarchar (60)**	Yes
City	**city**	Yes
Region	**region**	Yes
PostalCode	**postalcode**	Yes
Country	**country**	Yes
Phone	**nvarchar (24)**	Yes
Fax	**nvarchar (24)**	Yes

5. Save the new table as **Customers** and close the New Table window.

► To execute a script that creates all tables in the ClassNorthwind database

In this procedure, you will execute a script that creates all of the tables in the **ClassNorthwind** database.

1. Switch to SQL Query Analyzer.
2. Open, review and execute the Creatab3.sql script file in C:\Moc\2073A\Labfiles\L04.

   This script creates all of the tables in the **ClassNorthwind** database. It first drops all previously created tables.

Exercise 3
Adding and Dropping Columns

In this exercise, you will use the ALTER TABLE statement to add columns to a table and drop columns from a table in the **ClassNorthwind** database.

► To add a column to a table

In this procedure, you will write and execute a statement that adds a column to the **Employees** table in the **ClassNorthwind** database.

A complete Transact-SQL script for this procedure is located in C:\Moc\2073A\Labfiles\L04\Answers\Addcol.sql.

1. Verify that you are using the **ClassNorthwind** database.
2. Write and execute a statement to add a column to the **Employees** table. The column should be named **Age**, use the **tinyint** data type, and allow null values.

```
ALTER TABLE Employees
 ADD Age tinyint NULL
GO
```

3. Execute the **sp_help** system stored procedure on the **Employees** table to verify that the **Age** column was defined as you specified. Notice that the **Age** column appears as the final column in the table.

```
EXEC sp_help Employees
GO
```

► To drop a column from a table

After discussing client requirements with your users, you have determined that the **Age** column is not needed after all. In this procedure, you will write and execute a statement that drops the **Age** column from the **Employees** table in the **ClassNorthwind** database.

A complete Transact-SQL script for this procedure is located in C:\Moc\2073A\Labfiles\L04\Answers\Dropcol.sql.

1. Verify that you are using the **ClassNorthwind** database.
2. Write and execute a statement that drops the **Age** column from the **Employees** table.

```
ALTER TABLE Employees
 DROP COLUMN Age
GO
```

3. Execute the **sp_help** system stored procedure on the **Employees** table to verify that the **Age** column was dropped.

```
EXEC sp_help Employees
GO
```

Exercise 4
Generating Transact-SQL Scripts

In this exercise, you will use SQL Server Enterprise Manager to generate a Transact-SQL script of objects that you have created in the **ClassNorthwind** database.

► To generate scripts to recreate objects

In this procedure, you will use SQL Server Enterprise Manager to generate a Transact-SQL script that allows you to recreate all objects that you have created in the **ClassNorthwind** database.

1. Switch to SQL Server Enterprise Manager.
2. Expand Microsoft SQL Servers, expand SQL Server Group, expand your server, and then expand Databases.
3. Right-click **ClassNorthwind**, point to **All Tasks**, and then click **Generate SQL Script**.
4. On the **General** tab, click **Show All**.
5. Select the **All tables** and the **All user-defined data types** check boxes, and then click **OK**.
6. Save this script file as L04.sql.
7. Open and review the generated script.

Exercise 5
Loading the ClassNorthwind Database with Data

In this exercise, you will run a script to populate the **ClassNorthwind** database with data.

► To load data into the ClassNorthwind database

In this procedure, you will execute scripts that will generate and load sample data into the **ClassNorthwind** database.

1. Switch to SQL Query Analyzer.
2. Open and review the **Loaddata.sql** script file in C:\Moc\2073A\Labfiles\L04
3. Execute **Loaddata.sql**. This may take a minute or two to complete. Review the output in the results pane.

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.

- Creating Data Types
- Creating Tables
- Generating Column Values
- Generating Scripts

1. Can a user-defined data type in one database be used in another database on the same SQL Server?

   No. User-defined data types are limited to a single database. You can create a matching data type in another database or you can create user-defined data types in the model database.

2. You are designing a database that stores information about millions of different products. You want to minimize the storage space that you use to store the product information. Each product has a description line in the **products** table. Occasionally, a product description will require up to 200 characters, but most product descriptions will only require 50 characters. What data type would you use?

   Use a varchar(200) data type to keep the rows compact while at the same time allowing for the occasional product description that will require up to 200 bytes.

3. You need to run a script that was created using SQL Server Enterprise Manager. How do you do this?

   Open and run the script by using SQL Query Analyzer or osql.