MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 1: Introduction to Transact-SQL

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:
15 Minutes

Transact-SQL is a data definition, manipulation, and control language. This module provides a brief overview of Transact-SQL as a programming language. It also describes the types of Transact-SQL statements and syntax elements.

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

- Differentiate between Transact-SQL and ANSI-SQL.
- Describe the basic types of Transact-SQL.
- Describe the syntax elements of Transact-SQL.

Materials and Preparation

Required Materials

To teach this course, you need the following materials:

- Microsoft® PowerPoint® file 2071A_01.ppt.
- The C:\MOC\2071A\Demo\Ex_01.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.
- Complete the lab.
- Become familiar with the "Transact-SQL Reference" topic in Microsoft SQL Server™ Books Online.

Module Strategy

Use the following strategy to present this module:

- The Transact-SQL Programming Language

  Tell students that Transact-SQL is the programming language used within SQL Server.

  Because students are expected to be familiar with the principles of programming, this module does not cover basic programming or statement writing. Instead, it provides an overview and points out where Transact-SQL differs noticeably from the ANSI SQL-92 International Standards Organization (ISO) standard language.

- Types of Transact-SQL Statements

  This section reviews the language elements of Transact-SQL. Briefly discuss the basic types of SQL statements, such as Data Definition Language (DDL), Data Control Language (DCL), and Data Manipulation Language (DML) statements.

- Transact-SQL Syntax Elements

  Discuss the additional language elements in greater detail where needed, based on the knowledge level of the students. Discuss local and global variables, the various operators and functions, control of flow language, and comment characters. The module covers the top keywords or clauses that are commonly used; direct students to the extensive SQL Server Books Online that is available for detailed information on other keywords.

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.

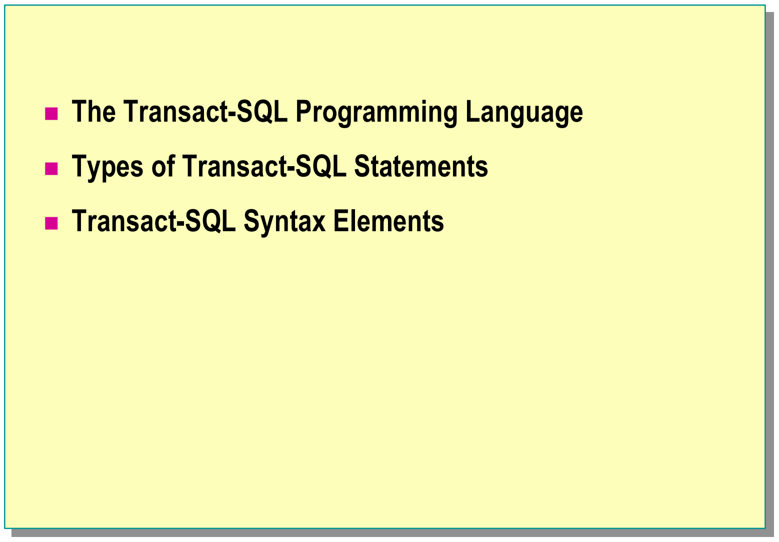
Lab Results

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 Transact-SQL.



- The Transact-SQL Programming Language
- Types of Transact-SQL Statements
- Transact-SQL Syntax Elements

Transact-SQL is a data definition, manipulation, and control language. This module provides a brief overview of Transact-SQL as a programming language. It also describes the types of Transact-SQL statements and syntax elements.

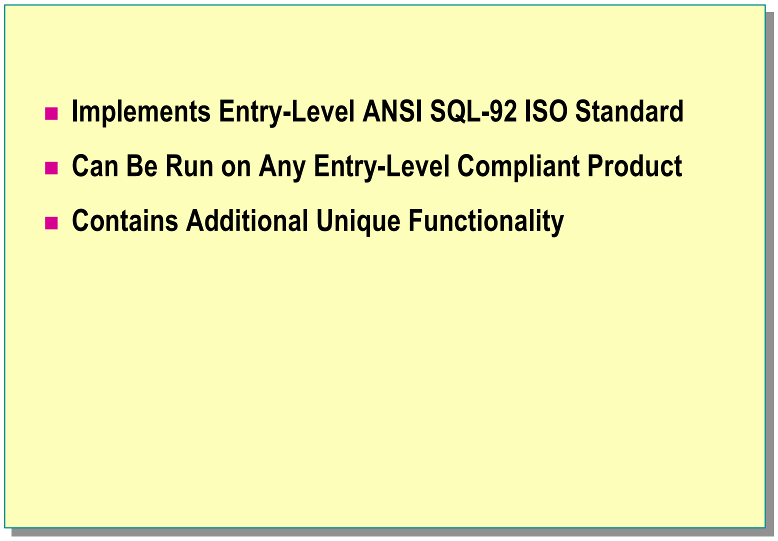
At the end of this module, you will be able to:

- Describe the Transact-SQL programming language.
- Describe the types of Transact-SQL statements.
- Describe Transact-SQL syntax elements.

The Transact-SQL Programming Language

Slide Objective
To introduce the fact that SQL Server has its own programming language.

Lead-in
Transact-SQL is a version of the SQL programming language that is used exclusively within SQL Server.



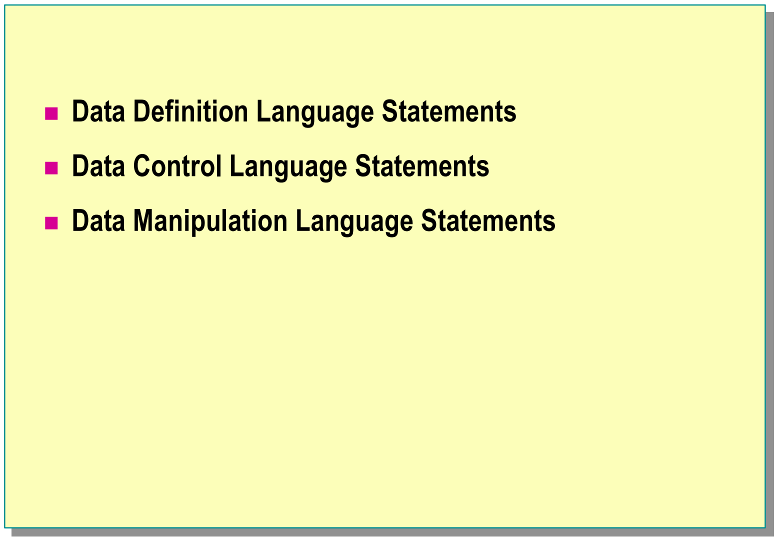
- **Implements Entry-Level ANSI SQL-92 ISO Standard**
- **Can Be Run on Any Entry-Level Compliant Product**
- **Contains Additional Unique Functionality**

The American National Standards Institute (ANSI) and the International Standards Organization (ISO) defined standards for SQL. Using Transact-SQL, Microsoft® SQL Server™ 2000 supports the entry level implementation of SQL-92, the SQL standard published by ANSI and ISO in 1992. The ANSI-SQL compliant language elements of Transact-SQL can be executed from any entry-level ANSI-SQL compliant product. Transact-SQL also contains several extensions to provide increased functionality.

◆ Types of Transact-SQL Statements

Slide Objective
To introduce the basic types of Transact-SQL statements.

Lead-in
As you write and execute Transact-SQL statements, you use some of these types of Transact-SQL statements.



- **Data Definition Language Statements**
- **Data Control Language Statements**
- **Data Manipulation Language Statements**

Delivery Tip
Point out that this course focuses on using Data Manipulation Language statements.

A *query* is a request for data stored in SQL Server. All queries present the result set of a SELECT statement to the user. A *result set* is a tabular arrangement of the data from the SELECT statement, comprising columns and rows.

Writing and executing Transact-SQL statements is one way to issue a query to SQL Server. As you write and execute Transact-SQL statements, you will use:

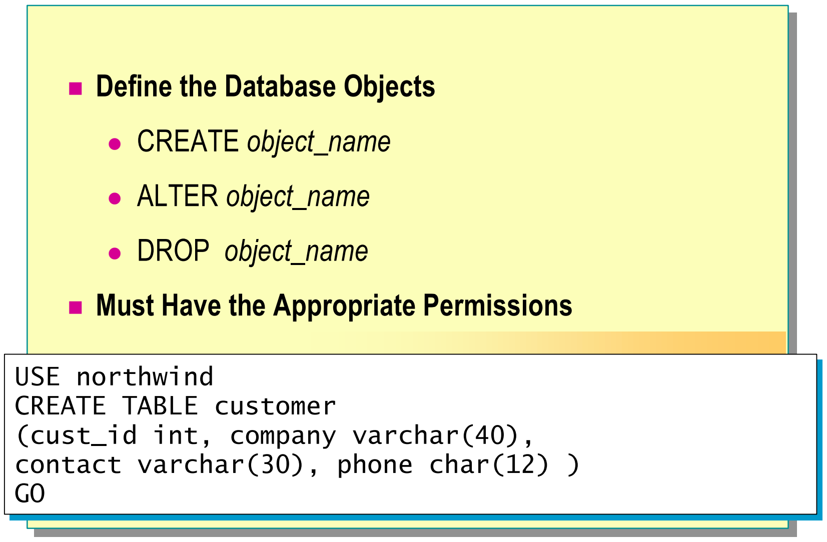
- Data Definition Language (DDL) statements, which allow you to create objects in the database.
- Data Control Language (DCL) statements, which allow you to determine who can see or modify the data.
- Data Manipulation Language (DML) statements, which allow you to query and modify the data.

Note This course focuses on using DML statements to query data in SQL Server.

Data Definition Language Statements

Slide Objective
To introduce students to DDL statements.

Lead-in
DDL statements define a database by creating databases, tables, and user-defined data types.



- **Define the Database Objects**
  - CREATE *object_name*
  - ALTER *object_name*
  - DROP *object_name*
- **Must Have the Appropriate Permissions**

```
USE northwind
CREATE TABLE customer
(cust_id int, company varchar(40),
contact varchar(30), phone char(12) )
GO
```

For Your Information
For more information on DDL statements, see course 2073A, *Programming a Microsoft SQL Server 2000 Database*.

Data Definition Language (DDL) statements define the database by creating databases, tables, and user-defined data types. You also use DDL statements to manage your database objects. Some DDL statements include:

- CREATE *object_name*
- ALTER *object_name*
- DROP *object_name*

By default, only members of the **sysadmin**, **dbcreator**, **db_owner**, or **db_ddladmin** role can execute DDL statements. In general, it is recommended that no other accounts be used to create database objects. If different users create their own objects in a database, then each object owner is required to grant the proper permissions to each user of those objects. This causes an administrative burden and should be avoided. Restricting statement permissions to these roles also avoids problems with object ownership that can occur when an object owner has been dropped from a database or when the owner of a stored procedure or view does not own the underlying tables.

Example

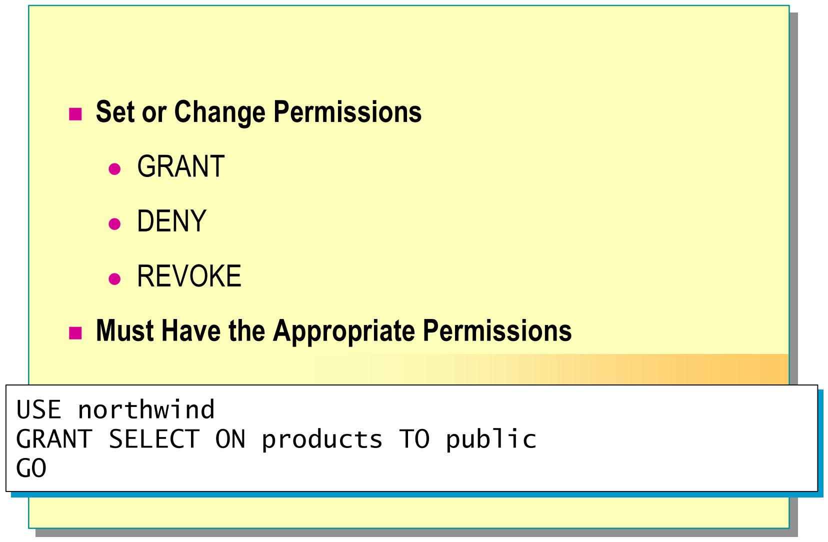
The following script creates a table called **customer** in the **Northwind** database. It includes **cust_id**, **company**, **contact**, and **phone** columns.

```
USE northwind
CREATE TABLE customer
(cust_id int, company varchar(40),contact varchar(30),
phone char(12) )
GO
```

Data Control Language Statements

Slide Objective
To introduce students to DCL statements.

Lead-in
DCL statements control access to database objects and to the ability to execute certain statements.



- **Set or Change Permissions**
  - GRANT
  - DENY
  - REVOKE
- **Must Have the Appropriate Permissions**

```
USE northwind
GRANT SELECT ON products TO public
GO
```

Delivery Tip
For more information on DCL statements, see course 2072A, *Database Administration for Microsoft SQL Server 2000*.

Data Control Language (DCL) statements are used to change the permissions associated with a database user or role. The following table describes the DCL statements.

Statement	Description
GRANT	Creates an entry in the security system that allows a user to work with data or execute certain Transact-SQL statements.
DENY	Creates an entry in the security system denying a permission from a security account and prevents the user, group, or role from inheriting the permission through its group and role memberships.
REVOKE	Removes a previously granted or denied permission.

By default, only members of the **sysadmin**, **dbcreator**, **db_owner**, or **db_securityadmin** role can execute DCL statements.

Example

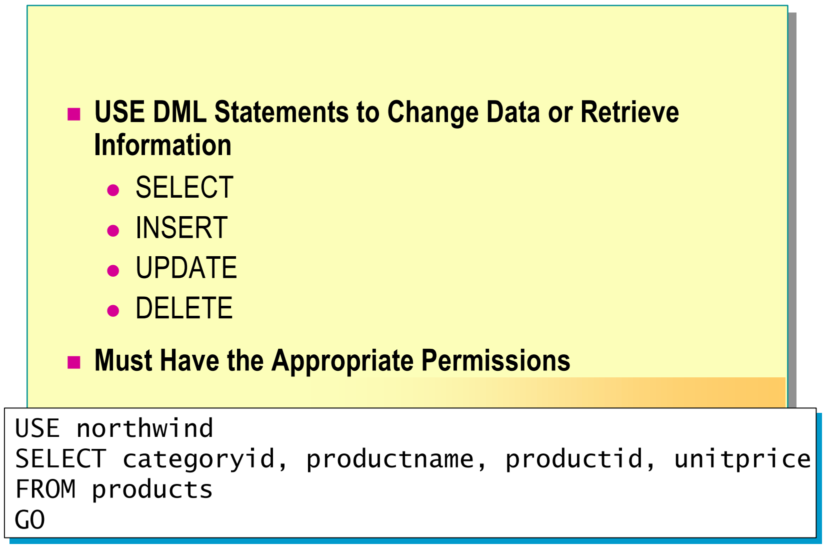
This example grants the **public** role permission to query the **products** table.

```
USE northwind
GRANT SELECT ON products TO public
GO
```

Data Manipulation Language Statements

Slide Objective
To introduce students to DML statements.

Lead-in
DML statements work with the data in the database.



- **USE DML Statements to Change Data or Retrieve Information**
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- **Must Have the Appropriate Permissions**

```
USE northwind
SELECT categoryid, productname, productid, unitprice
FROM products
GO
```

Data Manipulation Language (DML) statements work with the data in the database. By using DML statements, you can change data or retrieve information. DML statements include:

- SELECT.
- INSERT.
- UPDATE.
- DELETE.

By default, only members of the **sysadmin**, **dbcreator**, **db_owner**, or **db_datawriter** role can execute DML statements.

Example

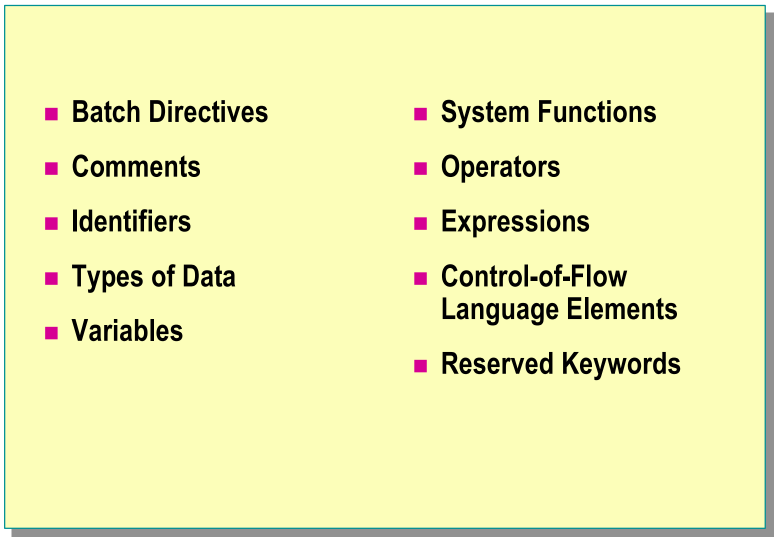
This example retrieves the category ID, product name, product ID, and unit price of the products in the **Northwind** database.

```
USE northwind
SELECT categoryid, productname, productid, unitprice
FROM products
GO
```

◆ Transact-SQL Syntax Elements

Slide Objective
To introduce various syntax elements of Transact-SQL.

Lead-in
DML statements are constructed by using a number of Transact-SQL syntax elements.



- Batch Directives
- Comments
- Identifiers
- Types of Data
- Variables
- System Functions
- Operators
- Expressions
- Control-of-Flow Language Elements
- Reserved Keywords

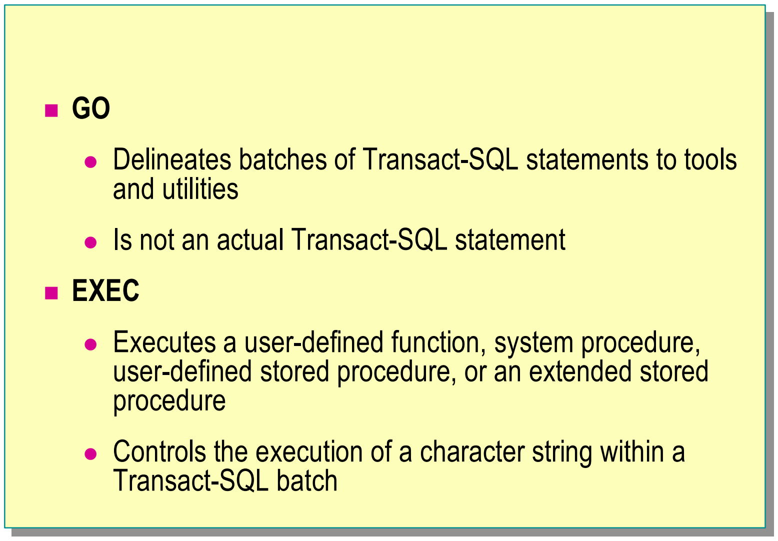
DML statements are constructed by using a number of Transact-SQL syntax elements. These include:

- Batch directives
- Comments
- Identifiers
- Types of data
- Variables
- System functions
- Operators
- Expressions
- Control-of-flow language elements
- Reserved keywords

Batch Directives

Slide Objective
To introduce how to execute batches of Transact-SQL.

Lead-in
Whatever tool you use, you need a way to tell it how to process your Transact-SQL code.



- **GO**
  - Delineates batches of Transact-SQL statements to tools and utilities
  - Is not an actual Transact-SQL statement
- **EXEC**
  - Executes a user-defined function, system procedure, user-defined stored procedure, or an extended stored procedure
  - Controls the execution of a character string within a Transact-SQL batch

SQL Server processes single or multiple Transact-SQL statements in batches. A batch directive instructs SQL Server to parse and execute all of the instructions within the batch. There are two basic methods for handing off batches to SQL Server.

GO

SQL Server utilities interpret GO as a signal to send the current batch of Transact-SQL statements to SQL Server. A GO command delineates batches of Transact-SQL statements to tools and utilities. A GO command is not an actual Transact-SQL statement.

When using GO, consider these facts:

- The current batch of statements is composed of all statements entered since the last GO, or since the start of the ad hoc session (or script, if this is the first GO).
- A Transact-SQL statement cannot occupy the same line as a GO command, although the line can contain comments.
- Users must follow the rules for batches.

  For example, some Data Definition Language statements must be executed separately from other Transact-SQL statements by separating the statements with a GO command.

  The scope of local (user-defined) variables is limited to a batch, and cannot be referenced after a GO command.

Note GO is not an actual Transact-SQL statement; it is used to delineate batches to tools and utilities.

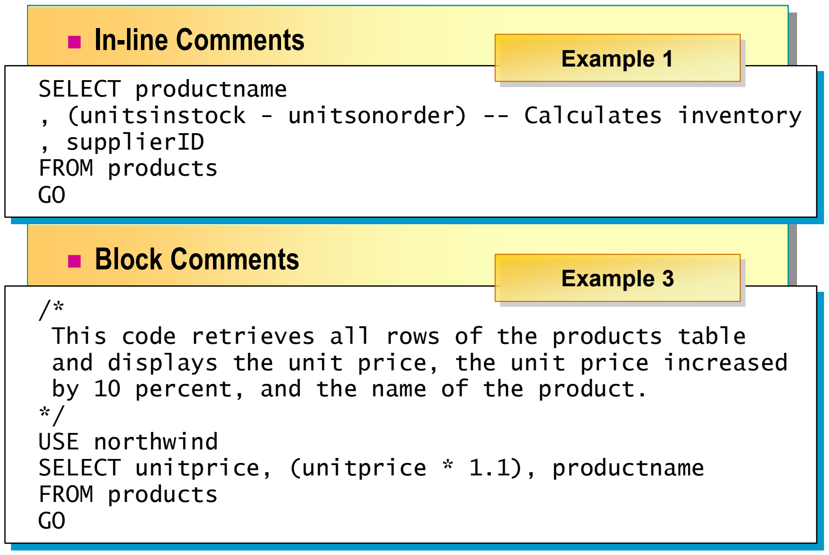
EXEC

The EXEC directive is used to execute a user-defined function, system procedure, user-defined stored procedure, or an extended stored procedure; it can also control the execution of a character string within a Transact-SQL batch. Parameters can be passed as arguments, and a return status can be assigned.

Comments

Slide Objective
To introduce students to the basic use of comments.

Lead-in
Comments are non-executing strings that you can place in statements to annotate the statement or to disable a portion of it during testing.


■ In-line Comments

Example 1

```
SELECT productname
, (unitsinstock - unitsonorder) -- Calculates inventory
, supplierID
FROM products
GO
```

■ Block Comments

Example 3

```
/*
 This code retrieves all rows of the products table
 and displays the unit price, the unit price increased
 by 10 percent, and the name of the product.
*/
USE northwind
SELECT unitprice, (unitprice * 1.1), productname
FROM products
GO
```

Delivery Tip
In-line comments at the start of the line can aid in troubleshooting scripts.

Comments are non-executing strings of text placed in statements to describe the action that the statement is performing or to disable one or more lines of the statement. Comments can be used in one of two ways—in line with a statement or as a block.

In-line Comments

You can create in-line comments using two hyphens (--) to set a comment apart from a statement. Transact-SQL ignores text to the right of the comment characters. This commenting character can also be used to disable lines of a statement.

Example 1

This example uses an in-line comment to explain what a calculation is doing.

```
USE northwind
SELECT productname
  , (unitsinstock - unitsonorder) -- Calculates inventory
  , supplierid
FROM products
GO
```

Example 2

This example uses an in-line comment to prevent the execution of a section of a statement.

```
USE northwind
SELECT productname
  , (unitsinstock - unitsonorder) -- Calculates inventory
-- , supplierid
FROM products
GO
```

Block Comments

You can create multiple line blocks of comments by placing one comment character (/*) at the start of the comment text, typing your comments, and then concluding the comment with a closing comment character (*/).

Use this character designator to create one or more lines of comments or comment headers—descriptive text that documents the statements that follow it. Comment headers often include the author's name, creation and last modification dates of the script, version information, and a description of the action that the statement performs.

Example 3

This example shows a comment header that spans several lines.

```
/*
 This code retrieves all rows of the products table
 and displays the unit price, the unit price increased
 by 10 percent, and the name of the product.
*/
USE northwind
SELECT unitprice, (unitprice * 1.1), productname
 FROM  products
GO
```

Note You should place comments throughout a script to describe the actions that the statements are performing. This is especially important if others must review or implement the script.

Example 4

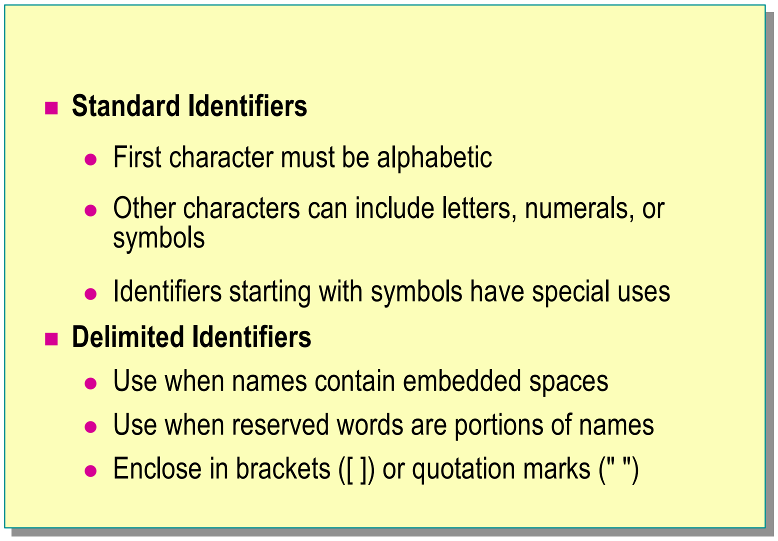
This section of a script is commented to prevent it from executing. This can be helpful when debugging or troubleshooting a script file.

```
/*
DECLARE @v1 int
SET @v1 = 0
WHILE @v1 < 100
   BEGIN
    SELECT @v1 = (@v1 + 1)
    SELECT @v1
   END
*/
```

◆ Identifiers

Slide Objective
To introduce the rules for naming SQL Server objects (rules for identifiers).

Lead-in
SQL Server provides a series of standard naming rules for object identifiers and a method of using delimiters for identifiers that are not standard.



- **Standard Identifiers**
  - First character must be alphabetic
  - Other characters can include letters, numerals, or symbols
  - Identifiers starting with symbols have special uses
- **Delimited Identifiers**
  - Use when names contain embedded spaces
  - Use when reserved words are portions of names
  - Enclose in brackets ([ ]) or quotation marks (" ")

Delivery Tip
Demonstrate how SQL Query Analyzer automatically color-codes the statement elements to show reserved words, strings, and so on.

SQL Server provides a series of standard naming rules for object identifiers and a method of using delimiters for identifiers that are not standard. It is recommended that you name objects using the standard identifier characters if possible.

Standard Identifiers

Standard identifiers can contain from one to 128 characters, including letters, symbols (_, @, or #), and numbers. No embedded spaces are allowed in standard identifiers. Rules for using identifiers include:

- The first character must be an alphabetic character of a-z or A-Z.
- After the first character, identifiers can include letters, numerals, or the @, $, #, or _ symbol.
- Identifier names starting with a symbol have special uses:
  - An identifier beginning with the @ symbol denotes a local variable or parameter.
  - An identifier beginning with a number sign (#) denotes a temporary table or procedure.
  - An identifier beginning with a double number sign (##) denotes a global temporary object.

Note Names for temporary objects should not exceed 116 characters, including the number (#) or double number (##) sign, because SQL Server gives temporary objects an internal numeric suffix.

For Your Information
The ability of object names to contain spaces is new to this version of SQL Server.

Delimited Identifiers

If an identifier complies with all of the rules for the format of identifiers, it can be used with or without delimiters. If an identifier does not comply with one or more of the rules for the format of identifiers, it must always be delimited.

Delimited identifiers can be used in the following situations:

- When names contain embedded spaces
- When reserved words are used for object names or portions of object names

Delimited identifiers must be enclosed in brackets or double quotation marks when they are used in Transact-SQL statements.

- Bracketed identifiers are delimited by square brackets ([ ]):

  ```
  SELECT * FROM [Blanks In Table Name]
  ```

  Note Bracketed delimiters can always be used, regardless of the status of the SET QUOTED_IDENTIFIER option.

- Quoted identifiers are delimited by double quotation marks (""):

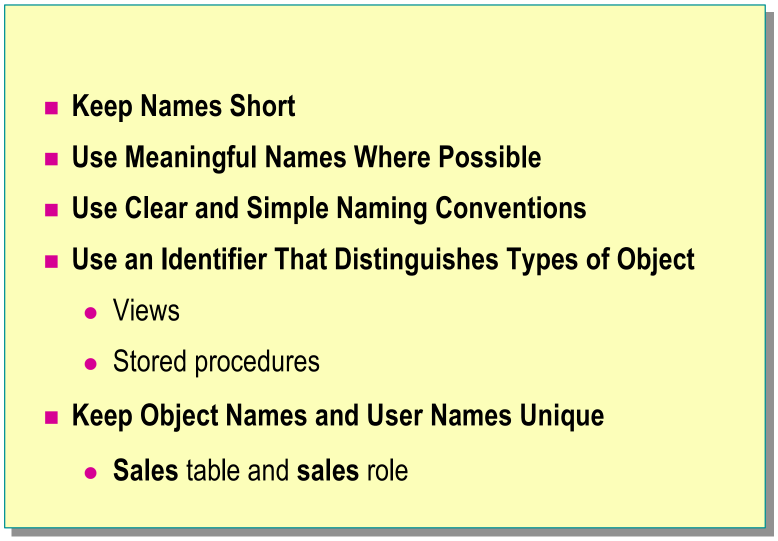
  ```
  SELECT * FROM "Blanks in Table Name"
  ```

  Quoted identifiers can only be used if the SET QUOTED_IDENTIFIER option is on.

Naming Guidelines for Identifiers

Slide Objective
To introduce suggested naming guidelines.

Lead-in
Keep database object names short when naming them.



- **Keep Names Short**
- **Use Meaningful Names Where Possible**
- **Use Clear and Simple Naming Conventions**
- **Use an Identifier That Distinguishes Types of Object**
  - Views
  - Stored procedures
- **Keep Object Names and User Names Unique**
  - **Sales** table and **sales** role

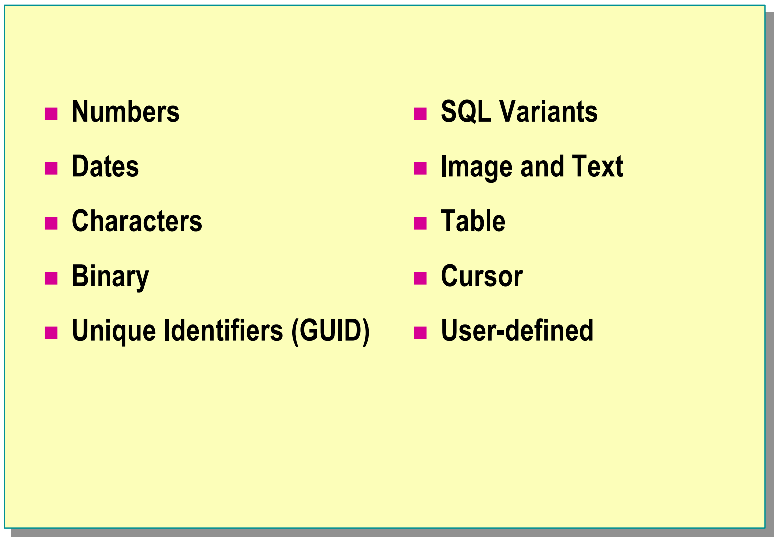
When naming database objects, you should:

- Keep names short.
- Use meaningful names where possible.
- Use a clear and simple naming convention. Decide what works best for your situation and be consistent. Try not to make naming conventions too complex, because they can become difficult to track or understand. For example, you can remove vowels if an object name must resemble a keyword (such as a backup stored procedure named **bckup**).
- Use an identifier that distinguishes the type of object, especially for views and stored procedures. System administrators often mistake views for tables, an oversight that can cause unexpected problems.
- Keep object names and user names unique. For example, avoid creating a **sales** table and a **sales** role within the same database.

Types of Data

Slide Objective
To introduce Transact-SQL base data types.

Lead-in
Data types constrain the type of values that you can store in a database.



- Numbers
- Dates
- Characters
- Binary
- Unique Identifiers (GUID)
- SQL Variants
- Image and Text
- Table
- Cursor
- User-defined

Delivery Tip
Relatively simple and common data types are in the left column of the slide, and more complex data types are in the right column.

Data types constrain the type of values that you can store in a database. Data types are attributes that specify what type of information can be stored in a column, parameter, or variable. Most Transact-SQL statements do not explicitly reference data types, but the results of most statements are influenced by the interactions between the data types of the objects referenced in the statement.

SQL Server provides system-supplied (base) data types, but you also can create data types. Examples of base data types include:

Numbers

This type of data represents numeric values and includes integers such as **int**, **tinyint**, **smallint**, and **bigint**. It also includes precise decimal values such as **numeric**, **decimal**, **money**, and **smallmoney**. It includes floating point values such as **float** and **real**.

Dates

This type of data represents dates or spans of time. The two date data types are **datetime**, which has a precision of .333 milliseconds, and **smalldatetime**, which has a precision of 1-minute intervals.

Characters

This type of data is used to represent character data or strings and includes fixed-width character string data types such as **char** and **nchar**, as well as variable-length string data types such as **varchar** and **nvarchar**.

For Your Information
The **rowversion** data type is the SQL Server 2000 alias for the **timestamp** data type. The **rowversion** data type has the same functionality as **timestamp**. The definition of **timestamp** in a future release of SQL Server will be modified to align with the SQL-99 definition of **timestamp**.

Binary

This type of data is very similar to character data types in terms of storage and structure, except that the contents of the data are treated as a series of byte values. Binary data types include **binary** and **varbinary**. A data type of **bit** indicates a single bit value of zero or one. A **rowversion** data type indicates a special 8-byte binary value that is unique within a database.

Unique Identifiers

This special type of data is a **uniqueidentifier** that represents a globally unique identifier (GUID), which is a 16-byte hexadecimal value that should always be unique.

SQL Variants

This type of data type can represent values of various SQL Server-supported data types, with the exception of **text**, **ntext**, **rowversion**, and other **sql_variant** values.

Image and Text

These types of data are binary large object (BLOB) structures that represent fixed- and variable-length data types for storing large non-Unicode and Unicode character and binary data, such as **image**, **text**, and **ntext**.

Tables

This type of data represents a table structure. It is possible to store a table within a field in SQL Server 2000.

Cursors

This type of data is used for programming within stored procedures and with low-level client interfaces. The cursor data type is never used as part of a DDL statement.

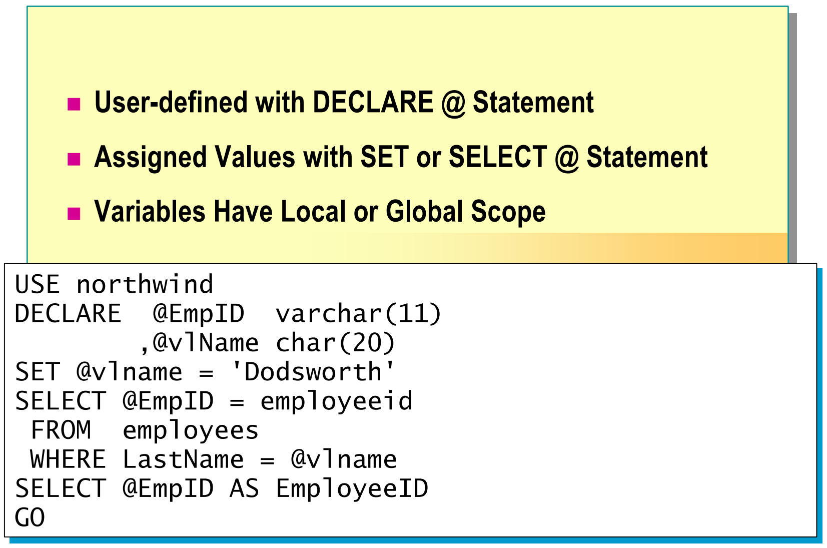
User-defined Data Types

This data type is created by the database administrator and is based on system data types. Use user-defined data types when several tables must store the same type of data in a column and you must ensure that the columns have exactly the same data type, length, and nullability.

Variables

Slide Objective
To define a variable and discuss how to use it.

Lead-in
Local variables are declared in the body of a batch or procedure by using the DECLARE statement, and they are given values with a SELECT statement.



- User-defined with DECLARE @ Statement
- Assigned Values with SET or SELECT @ Statement
- Variables Have Local or Global Scope

```
USE northwind
DECLARE  @EmpID  varchar(11)
        ,@vlName char(20)
SET @vlname = 'Dodsworth'
SELECT @EmpID = employeeid
 FROM  employees
 WHERE LastName = @vlname
SELECT @EmpID AS EmployeeID
GO
```

Variables are language elements with assigned values. You can use local variables in Transact-SQL.

A local variable is user-defined in a DECLARE statement, assigned an initial value in a SET or SELECT statement, and then used within the statement, batch, or procedure in which it was declared. A local variable is shown with one @ symbol preceding its name; a global variable is shown with two @ symbols preceding its name.

Note Local variables last only for the duration of a batch, whereas global variables last for the duration of a session.

Syntax

DECLARE {@local_variable data_type} [,...*n*]

SET *@local_variable_name = expression*

Example

This example creates the **@EmpID** and **@vlname** local variables, assigns a value to **@vlname**, and then assigns a value to **@EmpID** by querying the **Northwind** database to select the record containing the value of the **@vlname** local variable.

```
USE northwind
DECLARE  @EmpID  varchar(11)
        ,@vlName char(20)
SET @vlname = 'Dodsworth'
SELECT @EmpID = employeeid
 FROM  employees
 WHERE LastName = @vlname
SELECT @EmpID AS EmployeeID
GO
```

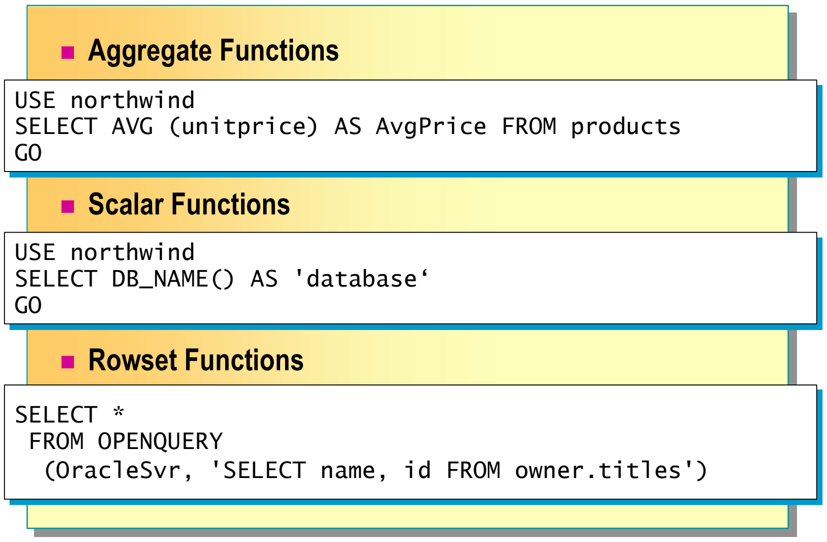
Result

```
EmployeeID
9
(1 row(s) affected)
```

◆ System Functions

Slide Objective
To provide an overview of the functions available in SQL Server.

Lead-in
Transact-SQL provides many system functions that return information.



- **Aggregate Functions**

```
USE northwind
SELECT AVG (unitprice) AS AvgPrice FROM products
GO
```

- **Scalar Functions**

```
USE northwind
SELECT DB_NAME() AS 'database‘
GO
```

- **Rowset Functions**

```
SELECT *
 FROM OPENQUERY
  (OracleSvr, 'SELECT name, id FROM owner.titles')
```

You can use system functions anywhere that an expression is allowed in a SELECT statement. Transact-SQL provides many functions that return information.

Functions take input parameters and return values that can be used in expressions. The Transact-SQL programming language provides three types of functions:

Aggregate Functions Operate on a collection of values but return a single, summarizing value.

Example 1

This example determines the average of the **unitprice** column for all products in the **products** table.

```
USE northwind
SELECT AVG(unitprice) AS AvgPrice
 FROM products
GO
```

Result

```
AvgPrice
28.8663

(1 row(s) affected)
```

Scalar Functions Operate on a single value and then return a single value. These functions can be used wherever an expression is valid. Scalar functions can be grouped into the following categories.

Function category	Description
Configuration	Returns information about the current configuration
Cursor	Returns information about cursors
Date and Time	Performs an operation on a date and time input value and returns a string, numeric, or date and time value
Mathematical	Performs a calculation based on input values provided as parameters to the function and returns a numeric value
Metadata	Returns information about the database and database objects
Security	Returns information about users and roles
String	Performs an operation on a string (**char** or **varchar**) input value and returns a string or numeric value
System	Performs operations and returns information about values, objects, and settings in SQL Server
System Statistical	Returns statistical information about the system
Text and Image	Performs an operation on a text or image input value or column, and returns information about the value

Example 2

This metadata function example returns the name of the database currently in use.

```
USE northwind
SELECT DB_NAME() AS 'database'
GO
```

Result

```
database
Northwind

(1 row(s) affected)
```

Rowset Functions Can be used like table references in a Transact-SQL statement.

Example 3

This example performs a distributed query to retrieve information from the **titles** table.

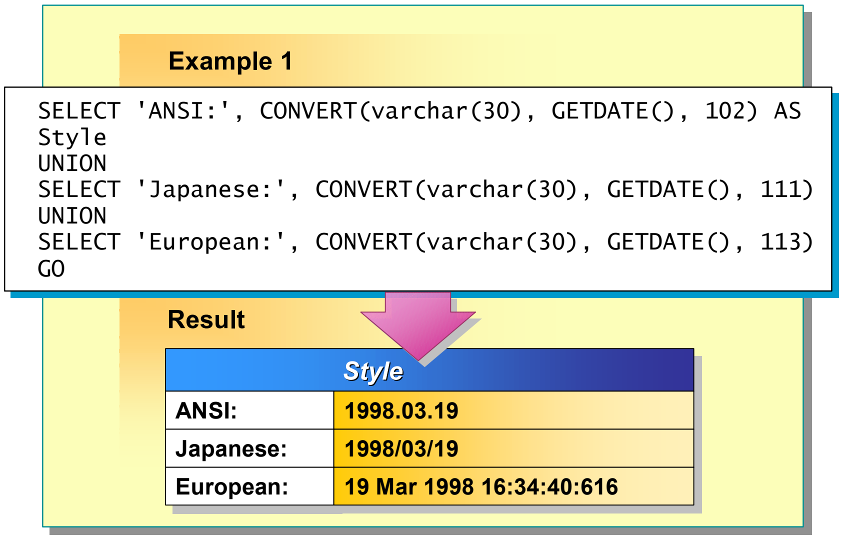
Delivery Tip
This example will not execute properly without access to an Oracle server.

```
SELECT *
FROM OPENQUERY(OracleSvr, 'SELECT name, id FROM owner.titles')
GO
```

System Function Examples

Slide Objective
To demonstrate some of the common uses of functions.

Lead-in
System functions are commonly used to convert date data from the format of one country to that of another.



Example 1

```
SELECT 'ANSI:', CONVERT(varchar(30), GETDATE(), 102) AS
Style
UNION
SELECT 'Japanese:', CONVERT(varchar(30), GETDATE(), 111)
UNION
SELECT 'European:', CONVERT(varchar(30), GETDATE(), 113)
GO
```

Result

	Style
ANSI:	1998.03.19
Japanese:	1998/03/19
European:	19 Mar 1998 16:34:40:616

Delivery Tip
Go to the "CONVERT" topic in SQL Server Books Online and point out the style options.

System functions are commonly used when converting date data from the format of one country to that of another.

Note To change date formats, you should use the CONVERT function with the style option to determine the date format that will be returned.

Example 1

This example demonstrates how you can convert dates to different styles.

```
SELECT 'ANSI:', CONVERT (varchar(30), GETDATE(), 102) AS Style
UNION
SELECT 'Japanese:', CONVERT(varchar(30), GETDATE(), 111)
UNION
SELECT 'European:', CONVERT(varchar(30), GETDATE(), 113)
GO
```

Result

```
             Style
European:    20 Nov 1998 16:44:12:857
Japanese:    11/20/98
ANSI:        1998.11.20
```

Example 2

This example uses the DATEFORMAT option of the SET statement to format dates for the duration of a connection. This setting is used only in the interpretation of character strings as they are converted to date values. It has no effect on the display of date values.

```
SET DATEFORMAT dmy
GO
DECLARE @vdate datetime
SET @vdate = '29/11/98'
SELECT @vdate
GO
```

Result

```
1998-11-29 00:00:00.000

(1 row(s) affected)
```

Example 3

This example returns the current user name and the application that the user is using for the current session or connection. The user in this example is a member of the **sysadmin** role.

```
USE library
SELECT user_name(), app_name()
GO
```

Result

```
dbo          MS SQL Query Analyzer

(1 row(s) affected)
```

Example 4

This example determines whether the **firstname** column in the **member** table of the **library** database allows null values.

A result of zero (false) means that null values are not allowed, and a result of one (true) means that null values are allowed. Notice that the OBJECT_ID function is embedded in the COLUMNPROPERTY function. This allows you to get the **object id** of the **member** table.

```
USE library
SELECT COLUMNPROPERTY(OBJECT_ID('member'), 'firstname',
'AllowsNull')
GO
```

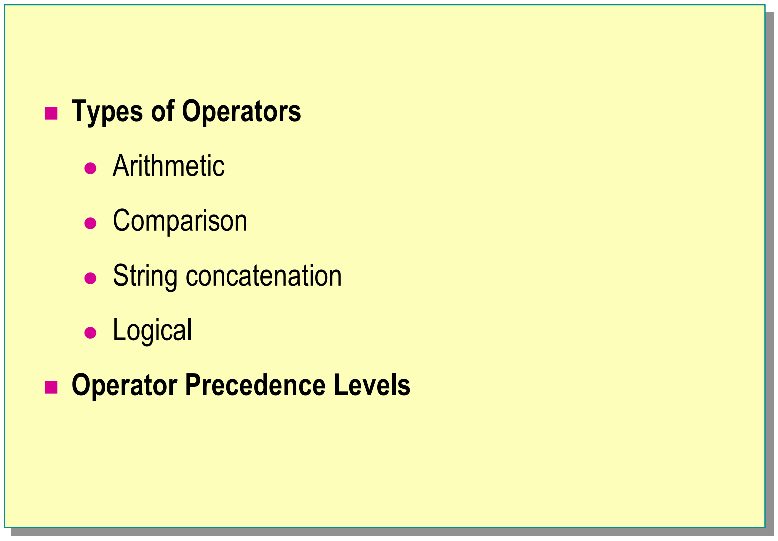
Result

```
0

(1 row(s) affected)
```

Operators

Slide Objective
To show how you can use operators to manipulate result sets.

Lead-in
You can use operators to perform computations or compare values.



- **Types of Operators**
  - Arithmetic
  - Comparison
  - String concatenation
  - Logical
- **Operator Precedence Levels**

Operators are symbols that perform mathematical computations, string concatenations, and comparisons between columns, constants, and variables. They can be combined and used in search conditions. When you combine them, the order in which the operators are processed is based on a predefined precedence.

Partial Syntax

{*constant* | *column_name* | *function* | (*subquery*)}
 [{*arithmetic_operator* | *string_operator* |
  AND | OR | NOT}
  {*constant* | *column_name* | *function* | (*subquery*)}…]

Types of Operators

SQL Server supports four types of operators: arithmetic, comparison, string concatenation, and logical.

Delivery Tip
Transact-SQL also supports bitwise operators. They are omitted here because they are used so seldom.

Arithmetic

Arithmetic operators perform computations with numeric columns or constants. Transact-SQL supports multiplicative operators, including multiplication (*), division (/), and modulo (%)—the integer remainder after integer division—and the addition (+) and subtraction (-) additive operators.

Comparison

Comparison operators compare two expressions. Comparisons can be made between variables, columns, and expressions of similar type. Comparison operators include those in the following table.

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

String Concatenation

The string concatenation operator (+) concatenates string values. All other string manipulation is handled through string functions. The empty string is never evaluated as a null value.

Logical

The logical operators AND, OR, and NOT connect search conditions in WHERE clauses.

Delivery Tip
Point out that the precedence levels of logical operators in SQL Server are different from other programming languages.

Operator Precedence Levels

If you use multiple operators (logical or arithmetic) to combine expressions, SQL Server processes the operators in order of their precedence, which may affect the resulting value. Operators have the following precedence levels (from highest to lowest).

Type	Operator	Symbol
Grouping	Primary grouping	( )
Arithmetic	Multiplicative	* / %
Arithmetic	Additive	- +
Other	String concatenation	+
Logical	NOT	NOT
Logical	AND	AND
Logical	OR	OR

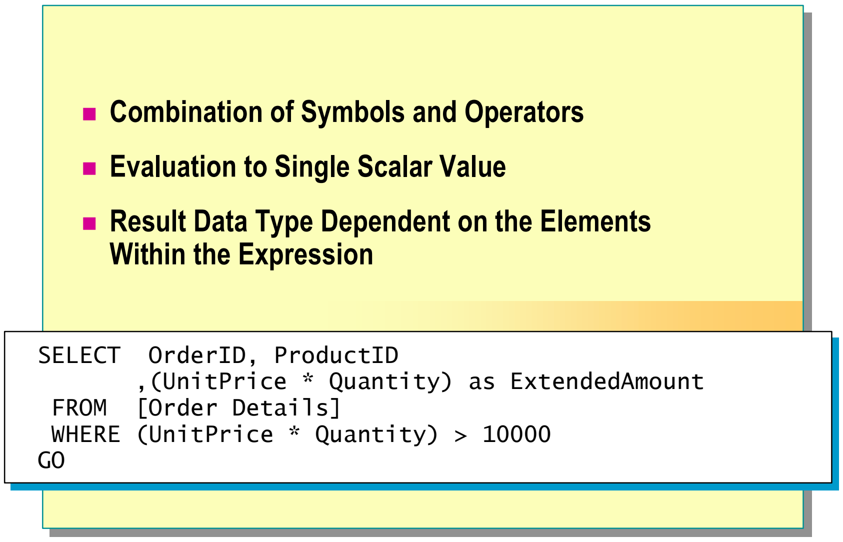
SQL Server handles the most deeply nested expression first. In addition, if all arithmetic operators in an expression share the same level of precedence, the order is from left to right.

Note The precedence levels of logical operators in SQL Server are different from other programming languages.

Expressions

Slide Objective
To introduce students to expressions.

Lead-in
Expressions are a combination of symbols and operators that evaluate to a single data value.



- **Combination of Symbols and Operators**
- **Evaluation to Single Scalar Value**
- **Result Data Type Dependent on the Elements Within the Expression**

```
SELECT  OrderID, ProductID
       ,(UnitPrice * Quantity) as ExtendedAmount
 FROM  [Order Details]
 WHERE (UnitPrice * Quantity) > 10000
GO
```

Expressions are a combination of symbols and operators that evaluate to a single data value. They can be simple—such as a constant, variable, column or scalar value—or complex expressions created by connecting one or more simple expressions with operators.

The data type of the result depends on the elements within the expression. Implicit data type conversions are frequently done on elements within the expression during evaluation.

Example

The following example calculates the extended amount of a product on an order by multiplying the unit price by the quantity ordered, and then filters the results so that the only rows returned are those orders with products that have an extended amount greater than $10,000.

```
SELECT  OrderID, ProductID
       ,(UnitPrice * Quantity) as ExtendedAmount
 FROM  [Order Details]
 WHERE (UnitPrice * Quantity) > 10000
GO
```

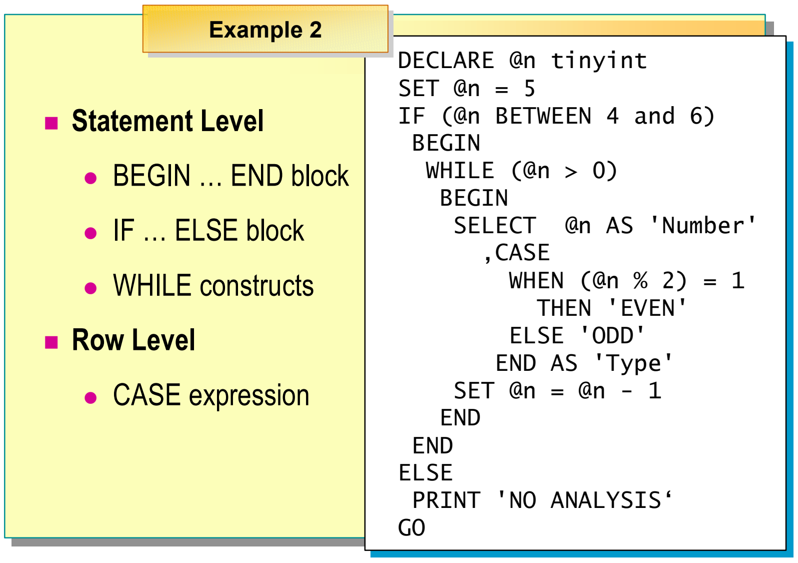
Result

```
OrderID           ProductID      ExtendedAmount
10353             38             10540.0000
10417             38             10540.0000
10424             38             10329.2000
10865             38             15810.0000
10889             38             10540.0000
10981             38             15810.0000

(6 row(s) affected)
```

Control-of-Flow Language Elements

Slide Objective
To introduce the Transact-SQL language elements that control the processing of statements.

Lead-in
Transact-SQL supports language that controls the flow of logic within statements.



Example 2

- **Statement Level**
  - BEGIN … END block
  - IF … ELSE block
  - WHILE constructs
- **Row Level**
  - CASE expression

```
DECLARE @n tinyint
SET @n = 5
IF (@n BETWEEN 4 and 6)
 BEGIN
  WHILE (@n > 0)
   BEGIN
    SELECT  @n AS 'Number'
      ,CASE
        WHEN (@n % 2) = 1
          THEN 'EVEN'
        ELSE 'ODD'
       END AS 'Type'
    SET @n = @n - 1
   END
 END
ELSE
 PRINT 'NO ANALYSIS‘
GO
```

Transact-SQL contains several language elements that control the flow of logic within a statement. It also contains the CASE expression that allows you to use conditional logic on a single row at a time within a SELECT or UPDATE statement.

Statement Level

The following language elements enable you to control the flow of logic within a script:

BEGIN … END Blocks These elements enclose a series of Transact-SQL statements so that they are treated as a unit.

IF … ELSE Blocks These elements specify that SQL Server should execute the first alternative if the certain condition is true. Otherwise, SQL Server should execute the second alternative.

WHILE Constructs These elements execute a statement repeatedly as long as the specified condition is true. BREAK and CONTINUE statements control the operation of the statements inside a WHILE loop.

Tip Indent Transact-SQL statements within a control-of-flow block to improve readability.

Example 1

This example determines whether a customer has any orders before deleting the customer from the customer list.

```
USE northwind
IF EXISTS (SELECT * FROM orders
           WHERE customerid = 'frank')
  PRINT '*** Customer cannot be deleted ***'
ELSE
  BEGIN
    DELETE customers WHERE customerid = 'frank'
    PRINT '*** Customer deleted ***'
  END
GO
```

Row Level

A CASE expression lists predicates, assigns a value for each, and then tests each one. If the expression returns a true value, the CASE expression returns the value in the WHEN clause. If the expression is false, and you have specified an ELSE clause, SQL Server returns the value in the ELSE clause. You can use a CASE expression anywhere that you use an expression.

Syntax

CASE *expression*
    {WHEN *expression* THEN *result*} [,…*n*]
  [ELSE *result*]
  END

Example 2

The following example declares a local variable, checks to see whether it equals 4, 5, or 6, and if it does, counts through a WHILE loop that determines whether the current value is an odd or even number.

Delivery Tip
Point out that the block indentation that Example 2 uses improves readability.

```
DECLARE  @n tinyint
SET   @n = 5
IF (@n BETWEEN 4 and 6)
 BEGIN
  WHILE (@n > 0)
   BEGIN
    SELECT  @n AS 'Number'
      ,CASE
        WHEN (@n % 2) = 1
          THEN 'EVEN'
        ELSE 'ODD'
       END AS 'Type'
    SET @n = @n - 1
   END
 END
ELSE
 PRINT 'NO ANALYSIS'
GO
```

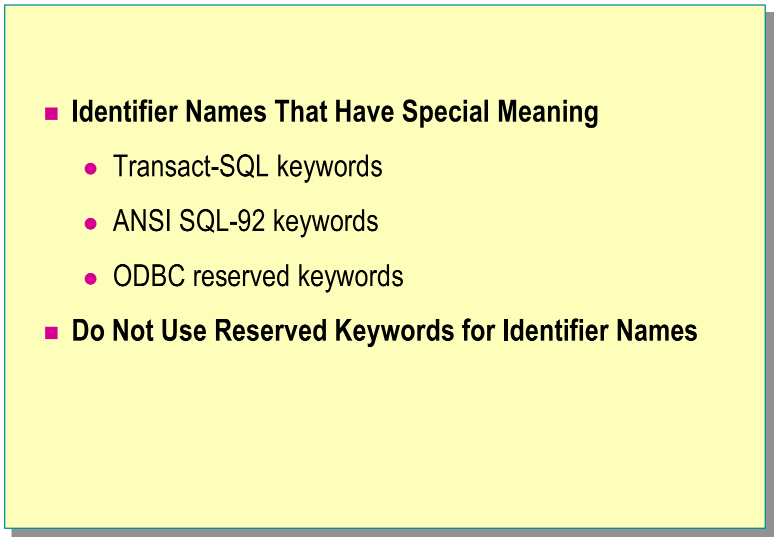
Result

```
Number                  Type
5                       EVEN

(1 row(s) affected)

Number                  Type
4                       ODD

(1 row(s) affected)

Number                  Type
3                       EVEN

(1 row(s) affected)

Number                  Type
2                       ODD

(1 row(s) affected)

Number                  Type
1                       EVEN

(1 row(s) affected)
```

Reserved Keywords

Slide Objective
To acquaint students with the concept of reserved keywords.

Lead-in
SQL Server reserves certain keywords for its own use; you should avoid using them for identifier names.



- **Identifier Names That Have Special Meaning**
  - Transact-SQL keywords
  - ANSI SQL-92 keywords
  - ODBC reserved keywords
- **Do Not Use Reserved Keywords for Identifier Names**

Delivery Tip
Display the extensive list of keywords from SQL Server Books Online.

SQL Server reserves certain keywords for its exclusive use. For example, using either the DUMP or BACKUP keyword in an **osql** or SQL Query Analyzer session tells SQL Server to make a backup copy of all or part of a database, or a backup copy of the log.

You cannot include reserved keywords in a Transact-SQL statement in any location except that defined by SQL Server. You should avoid naming an object with a reserved keyword. If an object name matches a keyword, whenever you refer to the object you must enclose it within delimiting identifiers, such as quotation marks or brackets [ ].

The system and database administrator roles, or the database creator, is usually responsible for checking for reserved keywords in Transact-SQL statements and database names.

Warning It is possible to construct syntactically correct Transact-SQL statements that may parse successfully and compile, but that still return a run-time error during execution. As a best practice, do not use reserved keywords.

Lab A: Using SQL Server Books Online

Slide Objective
To introduce the lab.

Lead-in
In this lab, you use SQL Server Books Online to retrieve information about SQL Server.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- View the contents, use the index, and search for information in Microsoft® SQL Server™ Books Online, as well as save the location of information on the **Favorites** tab.

Prerequisites

None.

Lab Setup

None.

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.
- SQL Server Books Online.

Scenario

The organization of the classroom is meant to simulate 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 the 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: 15 minutes

Exercise 1
Using SQL Server Books Online

In this exercise, you will use SQL Server Books Online to retrieve information on SQL Server.

► **To view the contents of Getting Started in SQL Server Books Online**

In this procedure, you will view the contents of SQL Server Books Online and familiarize yourself with conventions used in the documentation.

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. On the taskbar, click the **Start** button, point to **Programs**, point to **Microsoft SQL Server**, and then click **Books Online**.

   Note that you can also access SQL Server Books Online directly from the SQL Server 2000 compact disc. Insert the SQL Server 2000 compact disc into the CD-ROM drive, and when the **Microsoft SQL Server** dialog box appears, click **Browse Books Online**.
3. In the console tree, review the organization of SQL Server Books Online.
4. On the **Contents** tab, in the **Active Subset** list, click **Entire Collection**, and then review the contents of **Getting Started**.
5. In the console tree, expand **Getting Started with SQL Server Books Online**, and then click **Documentation Conventions**. Review the information in the details pane.
6. In the console tree, expand **Using SQL Server Books Online**, and then click **Finding a Topic**. Review the information in the details pane.
7. In the console tree, expand **Finding a Topic**, and then click **Using the Search tab**. Review the information in the details pane.

► **To use the SQL Server Books Online index to obtain information on the Northwind sample database**

In this procedure, you will use the SQL Server Books Online index to view information on the **Northwind** sample database quickly.

1. Click the **Index** tab, and then type **Northwind**
2. Double-click **Northwind sample database**.
3. In the **Topics Found** dialog box, double-click **Northwind sample database**. Review the information in the details pane.
4. Click the **Favorites** tab, and then click **Add**.
5. Click the **Contents** tab, and then in the console tree, expand **Northwind sample database** and notice the available topics.

► **To search SQL Server Books Online for a word or phrase**

In this procedure, you will use SQL Server Books Online to search for information about the architecture of SQL Server.

1. On the **Search** tab, select the **Match similar words** check box; clear the **Search titles only** check box.
2. Click the **Search** tab, type **sql NEAR architecture** and then click **List Topics**.

   Notice the number of topics that are found.
3. On the **Search** tab, clear the **Match similar words** check box, select the **Search titles only** check box, and then click **List Topics**.

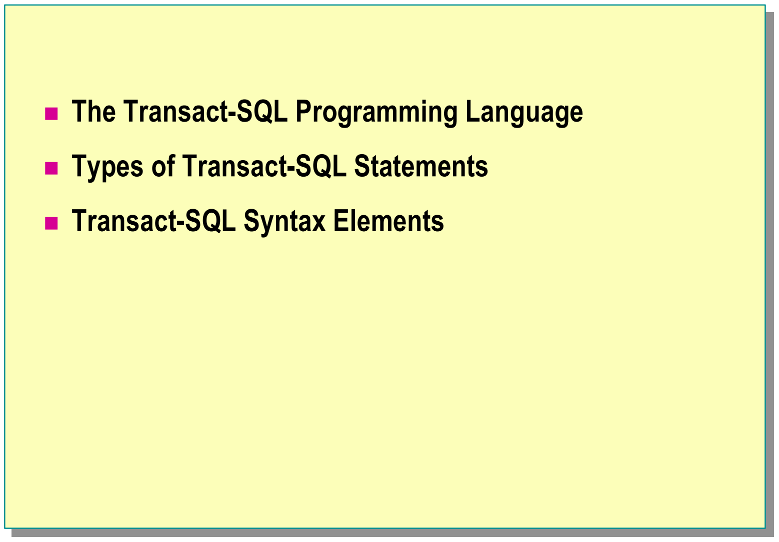
   Notice that only two topics are found.
4. Double-click **Fundamentals of SQL Server Architecture**.
5. Click the details pane, and then press CTRL+F.
6. In the **Find** box, type **oltp** and then click **Find Next**.

   Notice that the search finds the first instance of oltp.
7. Close SQL Server Books Online.

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.



- **The Transact-SQL Programming Language**
- **Types of Transact-SQL Statements**
- **Transact-SQL Syntax Elements**

Use these questions to review module topics.

Ask students whether they have any questions.

1. Describe the basic types of Transact-SQL statements and their uses.

   Data Definition Language (DDL) statements, which allow you to create objects in the database.

   Data Control Language (DCL) statements, which allow you to determine who can see or modify the data.

   Data Manipulation Language (DML) statements, which allow you to query and modify the data.

2. How does Transact-SQL relate to the ANSI SQL-92 specification?

   Transact-SQL implements the Entry-Level ANSI SQL-92 Specification, and has additional functionality exposed through SQL Server specific extensions.