MICROSOFT TRAINING AND CERTIFICATION

Microsoft Official Curriculum

Module 2: Overview of Programming SQL Server

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:
45 Minutes

Lab:
30 Minutes

This module provides students with an overview of Enterprise-level application architecture and Transact-SQL as a programming language. Transact-SQL is a data definition, manipulation, and control language. Students are assumed to be familiar with ANSI-SQL and basic programming concepts, such as functions, operators, variables, and control-of-flow statements, which are covered in Microsoft® Official Curriculum (MOC) course 2071, *Querying Microsoft SQL Server 2000 with Transact-SQL*. Students will also learn the different ways to execute Transact-SQL.

In the lab, students will write basic SELECT statements, modify a script file, and use system functions.

After completing this module, students will be able to:

- Describe the concepts of enterprise-level application architecture.
- Describe the primary Microsoft SQL Server™ 2000 programming tools.
- Explain the difference between the two primary programming tools in SQL Server.
- Describe the basic elements of Transact-SQL.
- Describe the use of local variables, operators, functions, control of flow statements, and comments.
- Describe the various ways to execute Transact-SQL statements.

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_02.ppt
- The C:\Moc\2073A\Demo\D02_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.
- Complete all demonstrations.
- Practice the presentation.
- Review any relevant white papers located on the Trainer Materials compact disc.

Module Strategy

Use the following strategy to present this module:

- Designing Enterprise Application Architecture

  Point out that students can design applications by using logical layers and services. Emphasize that a physical application design depends on the choice of architecture and how business logic is distributed across application components.

- SQL Server Programming Tools

  Introduce SQL Query Analyzer. Demonstrate the basic functions of SQL Query Analyzer, pointing out that students can execute part or all of a query, execute the query into a grid, and create an execution plan.

  Introduce the **osql** command-line utility. If students ask about the **isql** utility, point out that the **osql** utility uses the Open Database Connectivity (ODBC) application programming interface (API) to connect to SQL Server, which provides greater functionality), while the **isql** utility uses the earlier DB-Library API to connect to SQL Server.

- The Transact-SQL Programming Language

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

  Because students are expected to be familiar with the principles of programming, the 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 International Standards Organization (ISO) language.

- Elements of Transact-SQL

  This section reviews the language elements of Transact-SQL. Because students should be familiar with programming fundamentals, briefly discuss the Data Definition Language (DDL), Data Manipulation Language (DML), and Data Control Language (DCL) statements and discuss SQL Server object names and guidelines for naming database objects.

- Additional Language Elements

  Discuss local and system variables, the various operators and functions, control of flow language, and comment characters. The module covers the top keywords or clauses that students commonly use; direct students to SQL Server Books Online for detailed information on other keywords.

- Ways to Execute Transact-SQL Statements

  Familiarize students with the various ways that they can execute Transact-SQL statements. These include dynamically constructing statements, submitting batches, running scripts, and executing transactions. Where possible, demonstrate these by using SQL Query Analyzer.

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

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

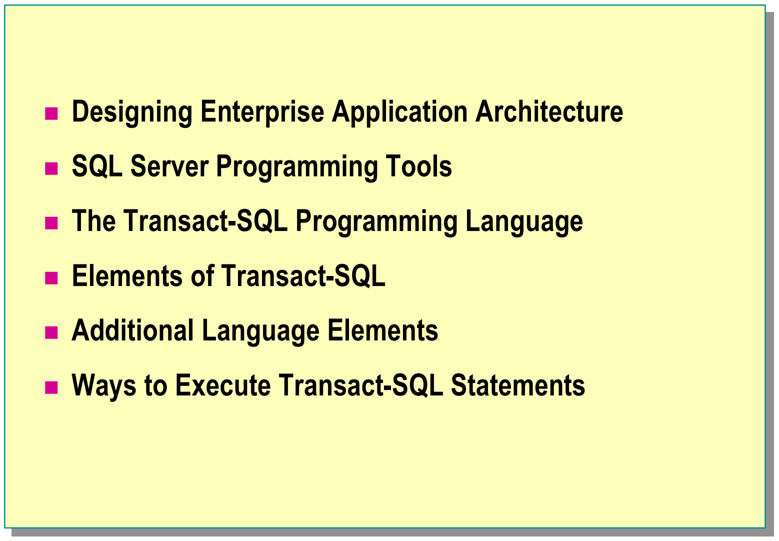
Lab Results

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

Overview

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

Lead-in
In this module, you will learn about Enterprise-level application architecture and Transact-SQL as a programming language.



- **Designing Enterprise Application Architecture**
- **SQL Server Programming Tools**
- **The Transact-SQL Programming Language**
- **Elements of Transact-SQL**
- **Additional Language Elements**
- **Ways to Execute Transact-SQL Statements**

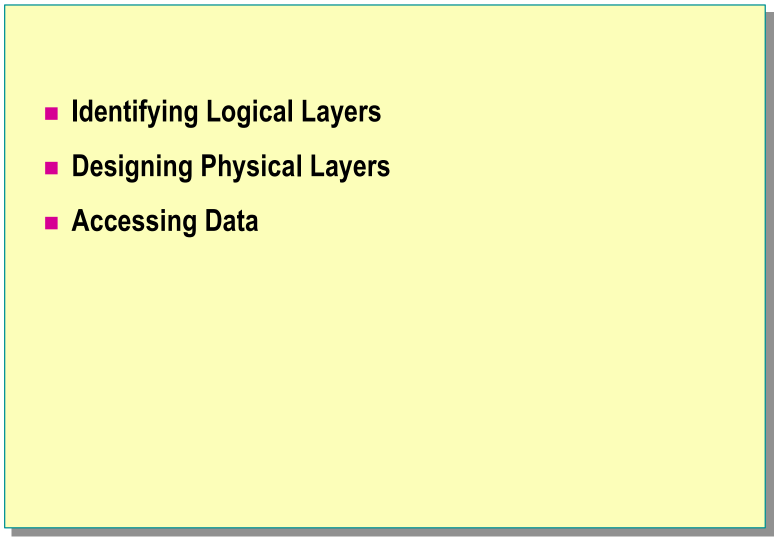
After completing this module, you will be able to:

- Describe the concepts of enterprise-level application architecture.
- Describe the primary Microsoft® SQL Server™ 2000 programming tools.
- Explain the difference between the two primary programming tools in SQL Server.
- Describe the basic elements of Transact-SQL.
- Describe the use of local variables, operators, functions, control of flow statements, and comments.
- Describe the various ways to execute Transact-SQL statements.

◆ Designing Enterprise Application Architecture

Topic Objective
To list the topics in this section.

Lead-in
In this section, you'll learn how to describe enterprise application architecture.



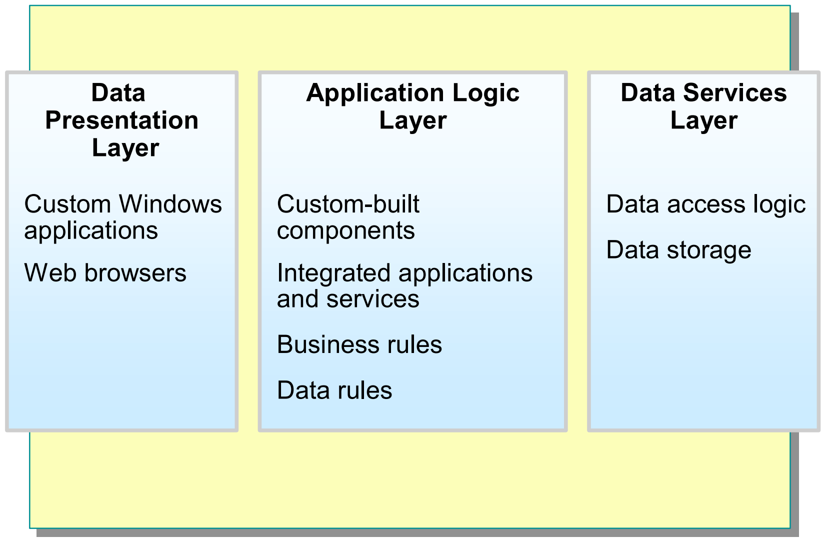
- Identifying Logical Layers
- Designing Physical Layers
- Accessing Data

SQL Server is often part of a distributed application. The design of a SQL Server implementation for an enterprise solution depends on your choice of architecture and how you intend to dsistribute logic across applications.

Identifying Logical Layers

Topic Objective
To identify logical layers in enterprise application architecture.

Lead-in
Enterprise application architecture contains logical layers.



Enterprise application architecture contains logical layers. The layers represent data presentation, application logic, and data services.

Data Presentation Layer

The data presentation layer is also referred to as user services and allows users to browse and manipulate data. The two main types of client applications are custom Microsoft Windows® applications and Web browsers. The data presentation layer uses the services that the application logic layer provides.

Application Logic Layer

This layer contains the application logic that defines rules and processes. It allows for scalability; instead of many clients directly accessing a database (with each client requiring a separate connection), clients can connect to business services that, in turn, connect to the data servers. Business services can be custom-built components or integrated applications and services, such as Web services. The application logic layer can also contain components that make use of transaction services, messaging services, or object and connection management services.

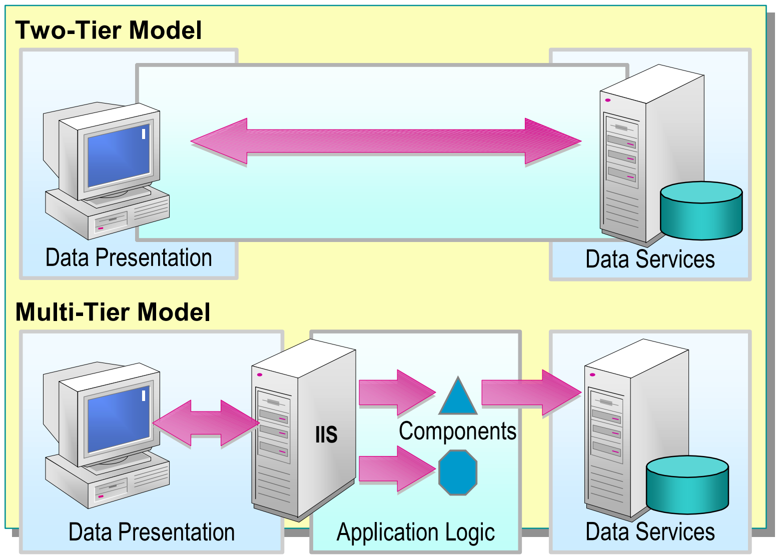
Data Services Layer

Data services include data access logic and data storage. These services can include SQL Server stored procedures to manage data traffic and integrity on the database server.

Designing Physical Layers

Topic Objective
To illustrate ways to incorporate logical layers in physical models.

Lead-in
You can physically place logical layers in a distributed environment in a variety of ways.



You can physically place logical layers in a distributed environment in a variety of ways. Although all logical layers can exist on one computer, it is typical to distribute the logical layers in a *two-tier* or *multi-tier model*. This allows you to implement logic, business rules, and processing where they are most effective.

Using a Two-Tier Model

If you use this model, you can locate the presentation and application logic on the client and the data services on a server. Alternatively, you can locate the application logic in stored procedures on the server. You can also have a mixed solution in which the application logic is divided between the client and the server.

Two-tier designs are less common than multi-tier designs, due to the growing popularity of Internet applications. They are not as scalable and may not be as easy to maintain as multi-tier designs are.

Using a Multi-Tier Model

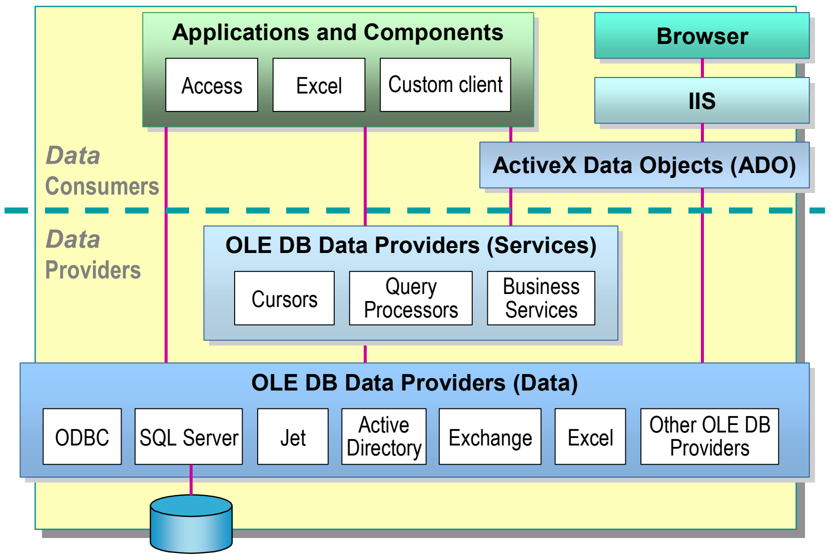
The multi-tier model, also known as three-tier or *n*-tier, allows you to distribute logic across applications. Business rules can be separate from the client or the database. When this model is applied to the Internet, you can divide presentation services between a browser client and a Microsoft Internet Information Services (IIS) Web server; the Web server formats the Web pages that the browser displays.

The multi-tier model is scalable for large client bases and many applications, and you can spread the workload among many computers. A multi-tier model is easy to manage because you can isolate a change to one business rule without affecting others. Also, an update to an Active Server Page (ASP) on a Web server automatically updates all clients.

Accessing Data

Topic Objective
To illustrate the various ways to access data.

Lead-in
You can access enterprise data by using pre-built clients or custom clients.



Microsoft technologies allow you to access enterprise data by using a wide range of pre-built clients or custom clients that use a data access-programming interface.

Using Pre-Built Clients

You can use pre-built client applications to access data on SQL Server. The data retrieval logic is part of the client application.

Microsoft Office 2000 includes Microsoft Access and Microsoft Excel. When part of a multi-tier solution, you use these applications primarily for presentation services. However, you can also use them for application logic and data services. These applications allow users to browse server-side data and perform ad hoc queries. You can use them to retrieve SQL Server data or as a client in a multi-tier design. You can also use Office 2000 as a development environment for building data access applications.

Access and Excel are examples of pre-built clients that offer a range of functionality. You can also use pre-built clients that only offer presentation services, such as a browser that communicates with IIS.

Building Custom Clients

You can build custom clients by using a data access programming interface and a development environment, such as Microsoft Visual Studio® version 6.0 Enterprise Edition.

Providing Universal Data Access

Custom clients may need to access many different data sources in the enterprise. Microsoft Data Access Components (MDAC) is an interface that allows communication with different data sources. You can use the following MDAC components to facilitate communication:

- *OLE DB*. A set of Component Services interfaces that provides uniform access to data stored in diverse information sources. OLE DB enables you to access relational and nonrelational data sources.
- *Microsoft ActiveX® Data Objects (ADO).* An easy-to-use application programming interface (API) to any OLE DB data provider. You can use ADO in a broad range of data access application scenarios. OLE DB and ADO allow you to create data components that use the integrated services provided by Component Services.

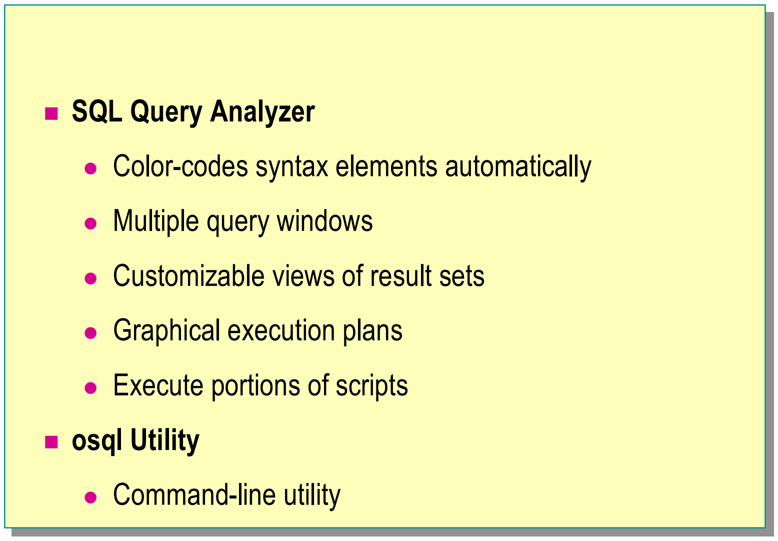
  ADO allows you to:

  - Open and maintain connections.
  - Create ad hoc queries.
  - Execute stored procedures on SQL Server.
  - Retrieve results and use cursors.
  - Cache query results on the client.
  - Update rows in the database.
  - Close connections.

SQL Server Programming Tools

Topic Objective
To introduce the programming tools in SQL Server.

Lead-in
SQL Server has many programming tools. Those you will use most often include SQL Query Analyzer and the **osql** utility.



- **SQL Query Analyzer**
  - Color-codes syntax elements automatically
  - Multiple query windows
  - Customizable views of result sets
  - Graphical execution plans
  - Execute portions of scripts
- **osql Utility**
  - Command-line utility

SQL Server 2000 offers several programming tools, including SQL Query Analyzer and the **osql** utility. SQL Query Analyzer is a Windows-based application, and **osql** is a utility that you can run from a command prompt.

SQL Query Analyzer

You can use SQL Query Analyzer to view query statements and results at the same time. You also can use it for writing, modifying, and saving Transact-SQL scripts.

SQL Query Analyzer provides the following features:

- Customized marking of syntax elements. As you write a query, SQL Query Analyzer highlights keywords, character strings, and other language elements; you can customize how they appear.
- Multiple query windows, each with its own connection.
- Customizable views of result sets. You can view results in default result set form or in a grid so that you can manipulate them as you would a table.
- Graphical execution plans that describe how SQL Server executes the query. You can view the optimized plan of execution and verify your syntax.
- The ability to execute portions of a script. You can select portions of a script, and SQL Server will execute only those portions.

For Your Information
The **isql** utility is not included in this course because it uses DB-Library to communicate with the server and does not support Unicode data types.

osql Utility

The **osql** utility allows you to write Transact-SQL statements, system procedures, and script files. It uses Open Database Connectivity (ODBC) to communicate with the server. You start the utility directly from the operating system with the case-sensitive arguments listed below. Once started, **osql** accepts Transact-SQL statements and sends them to SQL Server interactively. **Osql** formats and displays the results on the screen. Use the QUIT or EXIT commands to exit from **osql**.

Syntax

osql -U *login_id* [**-e**] [**-E**] [**-p**] [**-n**] [**-d** *db_name*] [**-q "***query***"**] [**-Q "***query***"**]
[**-c** *cmd_end*] [**-h** *headers*] [**-w** *column_width*] [**-s** *col_separator*]
[**-t** *time_out*] [**-m** *error_level*] [**-L**] [**-?**] [**-r** {**0** | **1**}]
[**-H** *wksta_name*] [**-P** *password*] [**-R**]
[**-S** *server_name*] [**-i** *input_file*] [**-o** *output_file*] [**-a** *packet_size*]
[**-b**] [**-O**] [**-l** *time_out*]

Note Parameters in **osql** statements are case sensitive.

Delivery Tip
Mention that arguments can be preceded by either the dash (-) or forward slash (/) character.

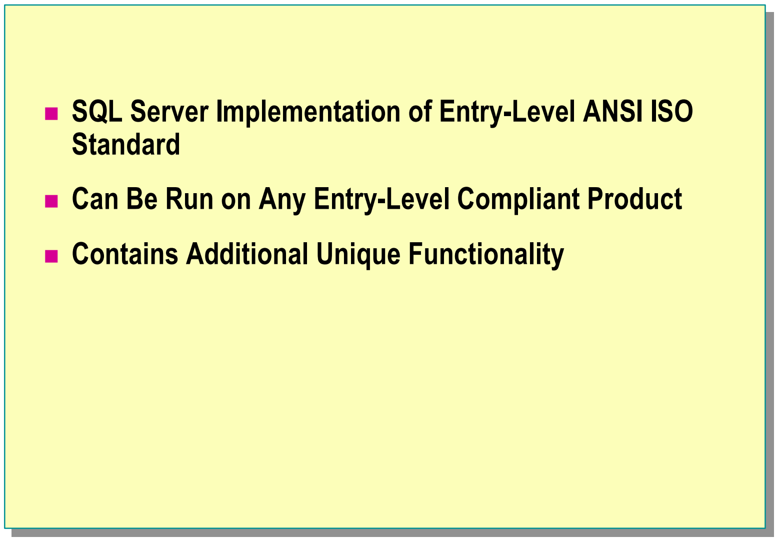
The following table describes the most commonly used arguments.

Argument	Description
-U *login_id*	Is the user login ID. Login IDs are case sensitive. If neither the **-U** or **-P** option is used, SQL Server uses the currently logged in user account and will not prompt for a password.
-E	Uses a trusted connection instead of requesting a password.
-?	Displays the syntax summary of **osql** switches.
-P *password*	Is a user-specified password. If the **-P** option is not used, **osql** prompts for a password. If the **-P** option is used at the end of the command prompt without any password, **osql** uses the default password (NULL). Passwords are case sensitive. If neither the **-U** or **-P** option is used, SQL Server uses the currently logged in user account and will not prompt for a password.
-S *server_name*	Specifies the SQL Server to which to connect. *server_name* is the name of the server computer on the network. This option is required if you execute **osql** from a remote computer on the network.
-i *input_file*	Identifies the file that contains a batch of Transact-SQL statements or stored procedures. You can use the less than (<) symbol instead of **-i**.
-o *output_file*	Identifies the file that receives output from **osql**. You can use the greater than (>) symbol in place of **-o**. If the input file is Unicode, the output file will be Unicode if you specify **-o**. If the input file is not Unicode, the output file is OEM.
-b	Specifies that **osql** exits and returns a Microsoft MS-DOS® ERRORLEVEL value when an error occurs. The value returned to the DOS ERRORLEVEL variable is 1 when the SQL Server error message has a severity of 10 or greater; otherwise, the value returned is 0. MS-DOS batch files can test the value of DOS ERRORLEVEL and handle the error appropriately.

The Transact-SQL Programming Language

Topic Objective
To introduce the SQL Server programming language.

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



- **SQL Server Implementation of Entry-Level ANSI ISO Standard**
- **Can Be Run on Any Entry-Level Compliant Product**
- **Contains Additional Unique Functionality**

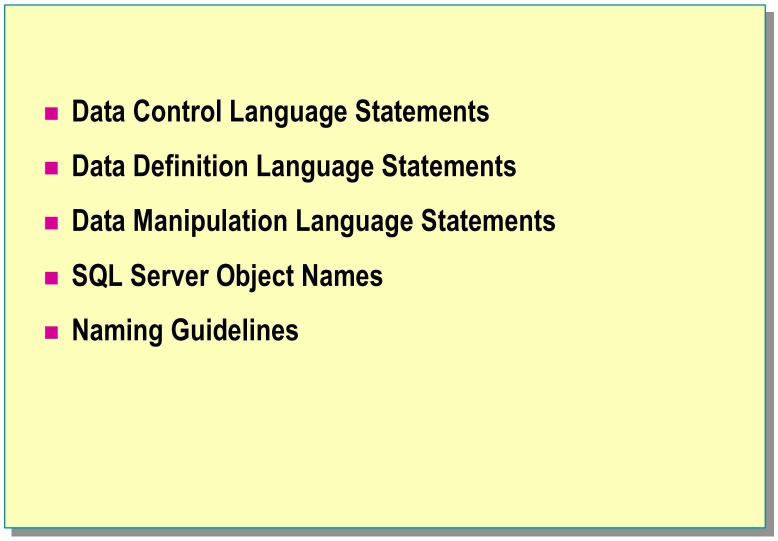
Transact-SQL is the SQL Server implementation of the entry-level ANSI-SQL International Standards Organization (ISO) standard. The ANSI-SQL compliant language elements of Transact-SQL can be executed from any entry-level ANSI-SQL compliant product. Transact-SQL also contains additional language elements that are unique to it.

Important It is recommended that you write scripts that include only ANSI-SQL standard statements to increase the compatibility and portability of your database.

◆ Elements of Transact-SQL

Topic Objective
To introduce the basic elements of the Transact-SQL programming language.

Lead-in
As you write and execute Transact-SQL statements, you'll use some of these elements of the language.



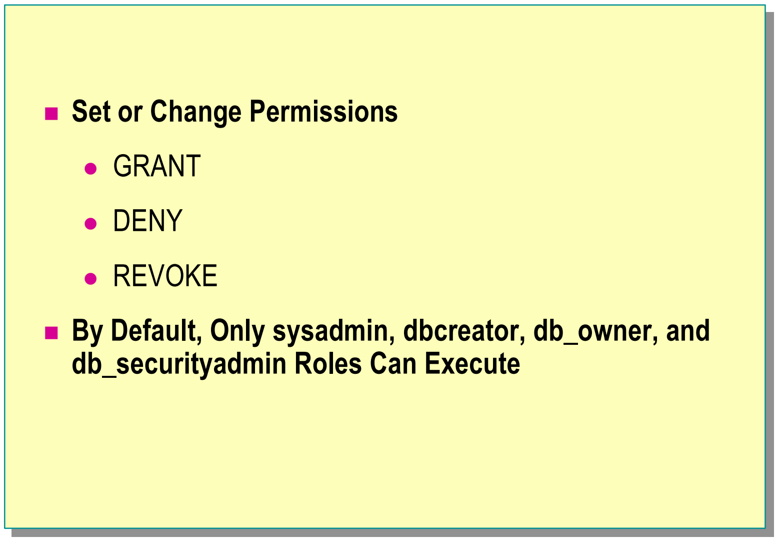
- **Data Control Language Statements**
- **Data Definition Language Statements**
- **Data Manipulation Language Statements**
- **SQL Server Object Names**
- **Naming Guidelines**

As you write and execute Transact-SQL statements, you will use different languages statements, which are used to determine who can see or modify the data, create objects in the database, and query and modify the data. You should follow the rules for naming SQL Server objects, and become familiar with the naming guidelines for database objects.

Data Control Language Statements

Topic 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
- **By Default, Only sysadmin, dbcreator, db_owner, and db_securityadmin Roles Can Execute**

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

You use Data Control Language (DCL) statements 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 that denies 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

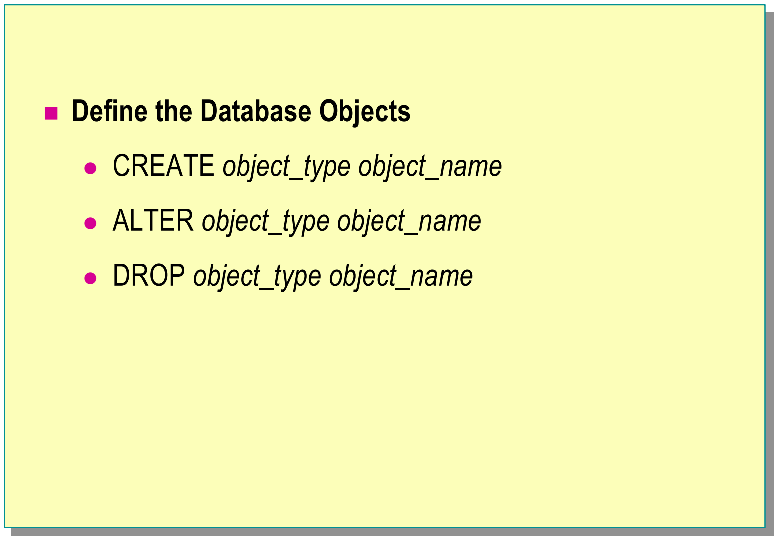
This example grants the **public** role permission to query the **Products** table.

```
USE Northwind
GRANT SELECT ON Products TO public
```

Data Definition Language Statements

Topic 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_type object_name*
  - ALTER *object_type object_name*
  - DROP *object_type object_name*

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_type object_name*.
- ALTER *object_type object_name*.
- DROP *object_type 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 allowed to create database objects. If users create their own objects in databases, 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.

If multiple user accounts create objects, the **sysadmin** and **db_owner** roles can use the SETUSER function to impersonate other users or the **sp_changeobjectowner** system stored procedure to change the owner of an object.

Example

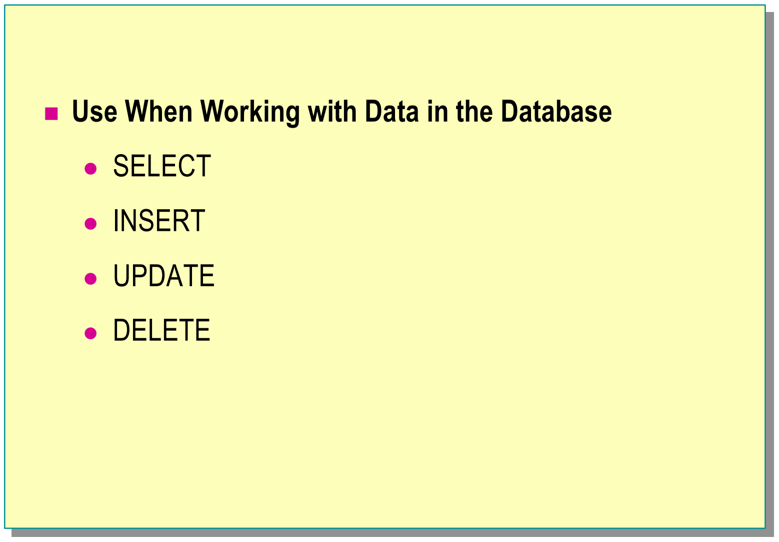
The following script creates a table called **Client** in the **ClassNorthwind** database. It includes **CustomerID**, **Company**, **Contact**, and **Phone** columns.

```
USE ClassNorthwind
CREATE TABLE Client
(CustomerID int, Company varchar(40),Contact varchar(30),
Phone char(12) )
```

Data Manipulation Language Statements

Topic Objective
To introduce students to DML statements.

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



- **Use When Working with Data in the Database**
  - SELECT
  - INSERT
  - UPDATE
  - DELETE

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**, **db_datawriter**, and **db_datareader** roles can execute DML statements.

Example

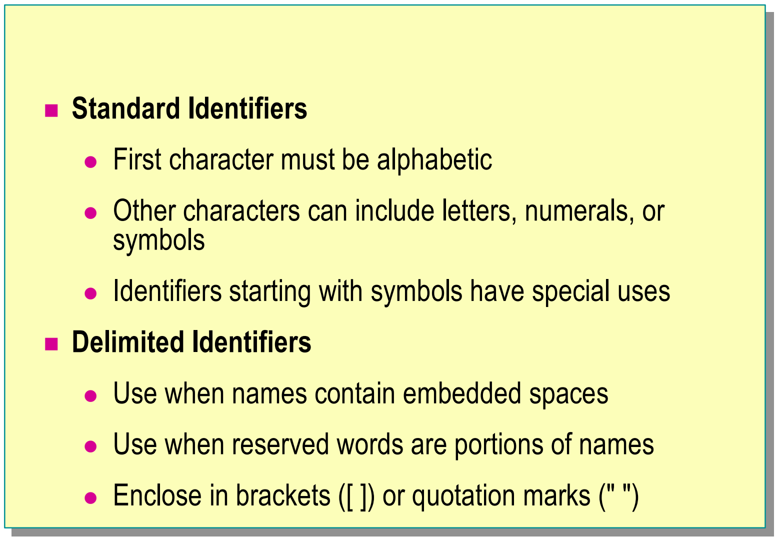
This example retrieves the category ID, product name, product ID, and unit price of the products in the **Northwind** database.

```
SELECT CategoryID, ProductName, CategoryID, ProductID,
UnitPrice
FROM Northwind..Products
```

SQL Server Object Names

Topic 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 by 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. You should observe the following rules for using identifiers:

- 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 at sign (@)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 sign (#) or double-number sign (##), because SQL Server gives temporary objects an internal numeric suffix.

Delimited Identifiers

If an identifier complies with all of the rules for the format of identifiers, you can use it 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.

You can use delimited identifiers in the following situations:

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

You must enclose delimited identifiers in brackets or quotation marks when you use them in Transact-SQL statements.

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

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

  Note You can always use bracketed delimiters, regardless of the status of the SET QUOTED_IDENTIFIER option.

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

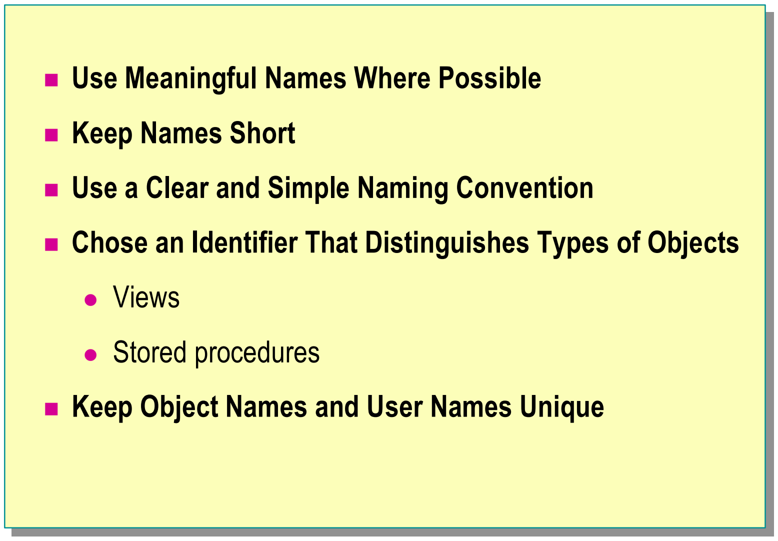
  ```
  SELECT * FROM "Blanks in Table Name"
  ```

  You can use quoted identifiers only if the SET QUOTED_IDENTIFIER option is on.

Naming Guidelines

Topic Objective
To introduce suggested naming guidelines.

Lead-in
Guidelines for naming database objects are important for identifying the type of object and to promote ease in troubleshooting or debugging. When naming database objects, you should…



- **Use Meaningful Names Where Possible**
- **Keep Names Short**
- **Use a Clear and Simple Naming Convention**
- **Chose an Identifier That Distinguishes Types of Objects**
  - Views
  - Stored procedures
- **Keep Object Names and User Names Unique**

Guidelines for naming database objects are important for identifying the type of object and to promote ease in troubleshooting or debugging. When naming database objects, you should:

- Use meaningful names where possible.

  For example, for a column that contains the name of customers, you could name the column **Chr_Name_Of_Customer**. A prefix of **Chr** in the column name denotes a **character** data type.

- Keep names short.

  For example, although the column name **Chr_Name_Of_Customer** is meaningful, you could shorten the column name to **Name** or **Chr_Name**.

- Use a clear and simple naming convention.

  Decide what works best for your situation, and be consistent. Avoid naming conventions that are too complex, because they can become difficult to remember. For example, you can remove vowels if an object name must resemble a keyword (such as a backup stored procedure named **Bckup**).

- Chose an identifier that distinguishes the type of object, especially when using views and stored procedures.

  System administrators often mistake views for tables, an oversight that can cause unexpected problems. For example, if you create a view that joins two tables, you could name that view, **SoldView**.

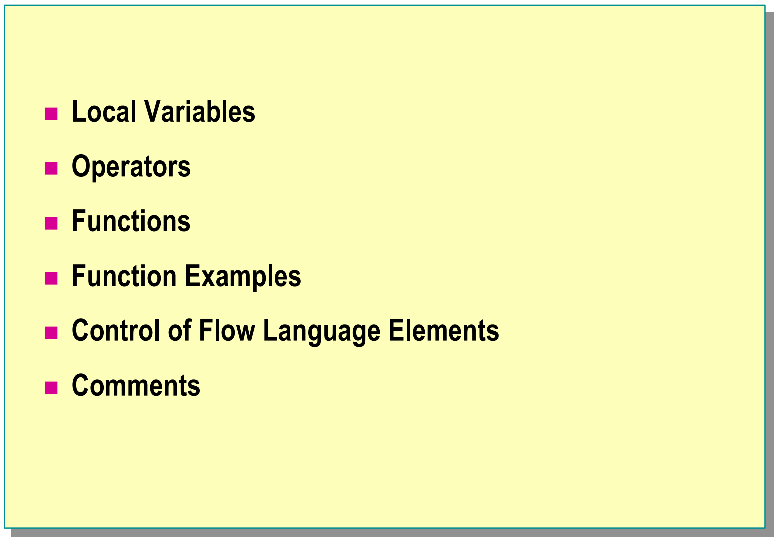
- Keep object names and user names unique.

  For example, avoid creating a **Sales** table and a **sales** role in the same database.

◆ Additional Language Elements

Topic Objective
To present an overview of the additional elements of the Transact-SQL language.

Lead-in
Some additional elements of the Transact-SQL language include…



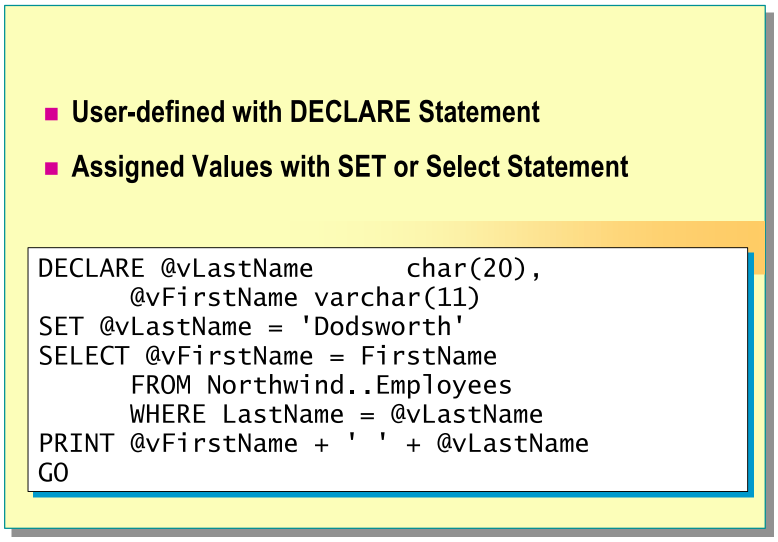
- **Local Variables**
- **Operators**
- **Functions**
- **Function Examples**
- **Control of Flow Language Elements**
- **Comments**

Some additional elements of the Transact-SQL language include local variables, operators, functions, control of flow statements, and comments.

Local Variables

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

Lead-in
You declare local variables in the body of a batch or procedure by using the DECLARE statement, and you give them values by using a SET or SELECT statement.



- **User-defined with DECLARE Statement**
- **Assigned Values with SET or Select Statement**

```
DECLARE @vLastName      char(20),
      @vFirstName varchar(11)
SET @vLastName = 'Dodsworth'
SELECT @vFirstName = FirstName
      FROM Northwind..Employees
      WHERE LastName = @vLastName
PRINT @vFirstName + ' ' + @vLastName
GO
```

Delivery Tip
Demonstrate this by using SQL Query Analyzer.

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

You define a local variable in a DECLARE statement and then assign it an initial value with either the SET or SELECT statement. Use the SET statement when the desired value is known. Use the SELECT statement when you must look up the desired value in a table. After you establish the value of the variable, you can use it in the statement, batch, or procedure in which it was declared. A batch is a set of Transact-SQL statements that are submitted together and executed as a group. A local variable is shown with one at sign (@) preceding its name.

Syntax

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

SET *@local_variable_name* = *expression*

Example

The following example declares two variables. It uses the SET statement to establish the value of the @vLastName variable and the SELECT statement to look up the value of the @vFirstName variable. It then prints both variables.

```
DECLARE @vLastName    char(20),
        @vFirstName   varchar(11)
SET @vLastName = 'Dodsworth'
SELECT @vFirstName = FirstName
   FROM Northwind..Employees
   WHERE LastName = @vLastName
PRINT @vFirstName + ' ' + @vLastNameGO
```

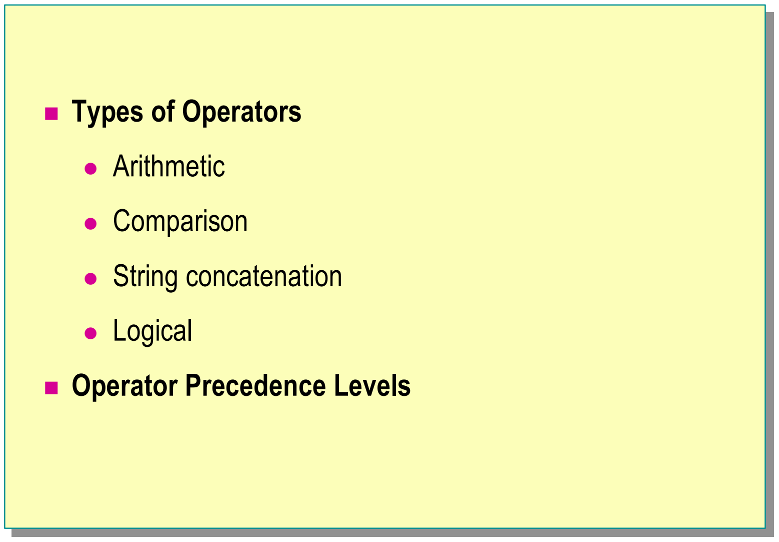
Result

Anne Dodsworth

Operators

Topic 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. You can combine them and use them in search conditions. When you combine them, the order in which SQL Server processes the operators 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.

Arithmetic

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

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. You can make comparisons between variables, columns, and expressions of similar type. The following table defines the comparison operators in Transact-SQL.

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. String functions handle all other string manipulation.

Logical

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

Delivery Tip
Mention that the precedence level of logical operators is 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. The following table shows the precedence level of operators (levels go from highest to lowest).

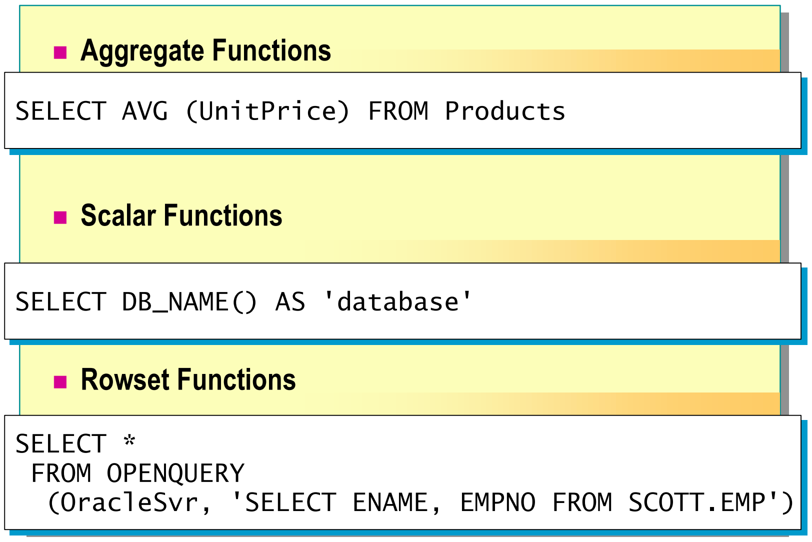
Type	Operator	Symbol
Grouping	Primary grouping	( )
Arithmetic	Multiplicative	* / %
Arithmetic	Additive	- +
Other	String concatenation	+
Logical	NOT	NOT
Logical	AND	AND
Logical	OR	OR

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.

Functions

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

Lead-in
Transact-SQL provides many functions that return information, including…



- **Aggregate Functions**

```
SELECT AVG (UnitPrice) FROM Products
```

- **Scalar Functions**

```
SELECT DB_NAME() AS 'database'
```

- **Rowset Functions**

```
SELECT *
 FROM OPENQUERY
  (OracleSvr, 'SELECT ENAME, EMPNO FROM SCOTT.EMP')
```

Delivery Tip
Mention that students can use 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, scalar, and rowset.

Aggregate 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.

```
SELECT AVG(UnitPrice) FROM Products
```

Result

```
Products
28.8663

(1 row(s) affected)
```

Scalar Functions

Scalar functions operate on a single value and then return a single value. You can use these functions wherever an expression is valid. You can group scalar functions into the categories in the following table.

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 then 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.

```
SELECT DB_NAME() AS 'database'
```

Result

```
Database
Northwind

(1 row(s) affected)
```

Rowset Functions

Rowset functions can be used like table references in a Transact-SQL statement.

Example 3

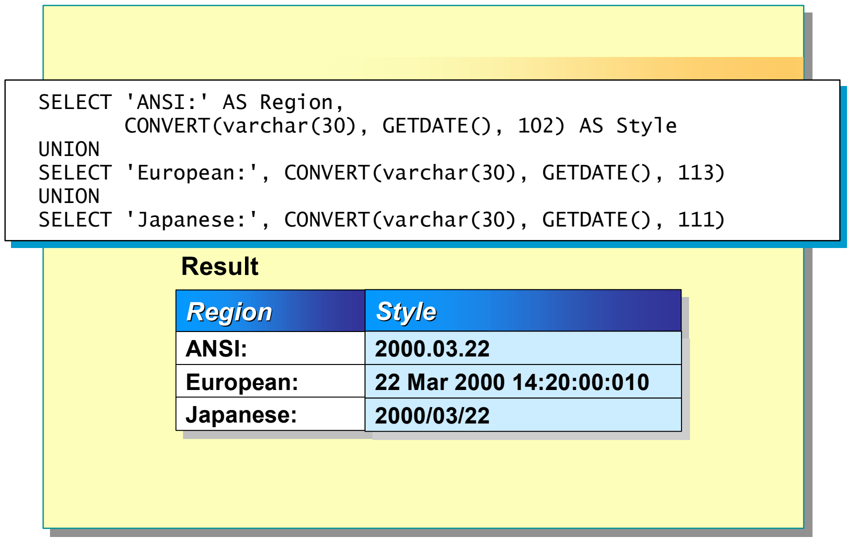
This example performs a distributed query to retrieve information from the **EMP** table.

```
SELECT *
FROM OPENQUERY(OracleSvr, 'SELECT ENAME, EMPNO FROM
SCOTT.EMP')
```

Function Examples

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

Lead-in
You commonly use functions to convert date data from the format of one country to that of another.



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

Result

Region	*Style*
ANSI:	2000.03.22
European:	22 Mar 2000 14:20:00:010
Japanese:	2000/03/22

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

You commonly use functions 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:' AS Region,
   CONVERT (varchar(30), GETDATE(), 102) AS Style
UNION
SELECT 'European:', CONVERT(varchar(30), GETDATE(), 113)
UNION
SELECT 'Japanese:', CONVERT(varchar(30), GETDATE(), 111)
```

Result

```
Region      Style
ANSI:       2000.03.22
European:   22 Mar 2000 14:20:00:010
Japanese:   2000/03/22
```

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 and has no effect on the display of date values.

```
SET DATEFORMAT dmy
GO
DECLARE @vdate datetime
SET @vdate = '29/11/00'
SELECT @vdate
```

Result

```
2000-11-29 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 Northwind
SELECT user_name(), app_name()
```

Result

```
dbo          MS SQL Query Analyzer

(1 row(s) affected)
```

Example 4

This example determines whether the **FirstName** column in the **Employees** table of the **Northwind** 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 retrieve the **object id** of the **Employees** table.

```
USE Northwind
SELECT COLUMNPROPERTY(OBJECT_ID('Employees'), 'FirstName',
   'AllowsNull')
```

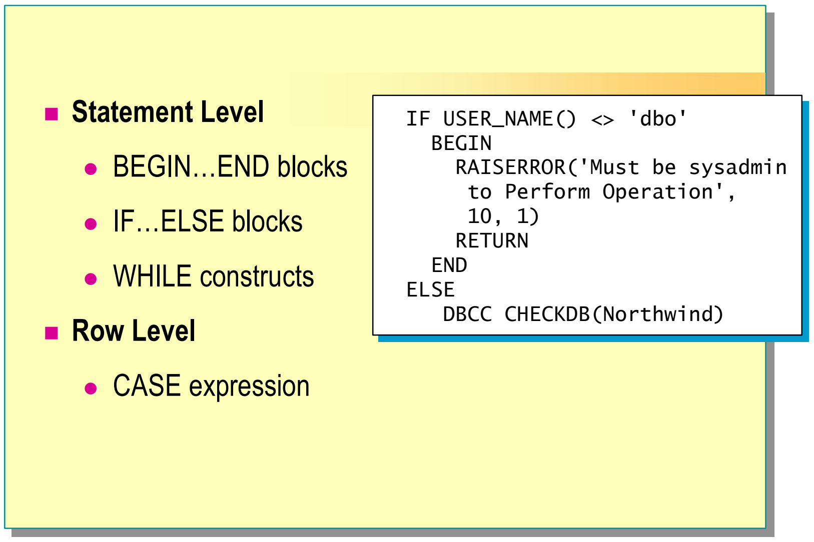
Result

```
0

(1 row(s) affected)
```

Control of Flow Language Elements

Topic Objective
To introduce students to the Transact-SQL language elements that control the processing of statements.

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



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

```
IF USER_NAME() <> 'dbo'
  BEGIN
    RAISERROR('Must be sysadmin
     to Perform Operation',
     10, 1)
    RETURN
  END
ELSE
   DBCC CHECKDB(Northwind)
```

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

Statement Level

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

BEGIN…END Blocks These elements enclose a series of Transact-SQL statements so that SQL Server treats them as a unit.

IF…ELSE Blocks These elements specify that SQL Server should execute the first alternative if a 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.

Example 1

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

```
USE Northwind
IF EXISTS (SELECT OrderID FROM Orders
          WHERE CustomerID = 'Frank')
      PRINT '*** Customer cannot be deleted ***'
ELSE
  BEGIN
      DELETE Customers WHERE CustomerID = 'Frank'
      PRINT '*** Customer deleted ***'
  END
```

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

The following example reviews the inventory status of products in the **Products** table and returns messages based on the quantities available and quantities back ordered, and whether the product has been discontinued.

```
SELECT ProductID, 'Product Inventory Status' =
  CASE
  WHEN (UnitsInStock < UnitsOnOrder AND Discontinued = 0)
      THEN 'Negative Inventory - Order Now!'
  WHEN ((UnitsInStock-UnitsOnOrder) < ReorderLevel AND
         Discontinued = 0)
      THEN 'Reorder level reached- Place Order'
  WHEN (Discontinued = 1) THEN '***Discontinued***'
  ELSE 'In Stock'
   END
FROM Northwind..Products
```

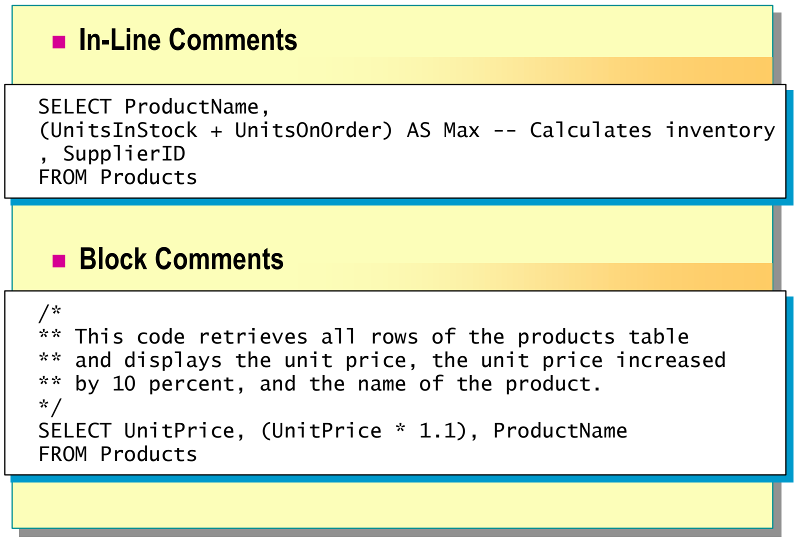
Result

```
ProductID     Product Inventory Status
1             In Stock
2             Negative Inventory - Order Now!
3             Negative Inventory - Order Now!
4             In Stock
5             ***Discontinued***
6             In Stock
7             In Stock
8             In Stock
9             ***Discontinued***
10            In Stock
11            Negative Inventory - Order Now!
12            In Stock
13            Reorder level reached - Place Order
.
.
.
(77 row(s) affected)
```

Comments

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

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



- **In-Line Comments**

```
SELECT ProductName,
(UnitsInStock + UnitsOnOrder) AS Max -- Calculates inventory
, SupplierID
FROM Products
```

- **Block Comments**

```
/*
** 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.
*/
SELECT UnitPrice, (UnitPrice * 1.1), ProductName
FROM Products
```

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. You can use comments in one of two ways—in line with a statement, or as a block.

In-Line Comments

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

Example 1

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

```
SELECT ProductName
,(UnitsInStock + UnitsOnOrder) AS Max -- Calculates inventory
  , SupplierID
FROM Products
```

Example 2

This example uses a second set of in-line comments, as represented by the second set of hyphens (--), to prevent the execution of a section (SupplierID) of a statement.

```
SELECT ProductName
,(UnitsInStock + UnitsOnOrder) AS Max -- Calculates inventory
-- , SupplierID
FROM Products
```

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.

Note You cannot place the GO statement inside of block comments.

Example 3

This example shows a comment header that spans several lines. The two asterisks (**) preceding each line improve readability.

```
/*
** 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.
*/
SELECT UnitPrice, (UnitPrice * 1.1), ProductName
FROM Products
```

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

Example 4

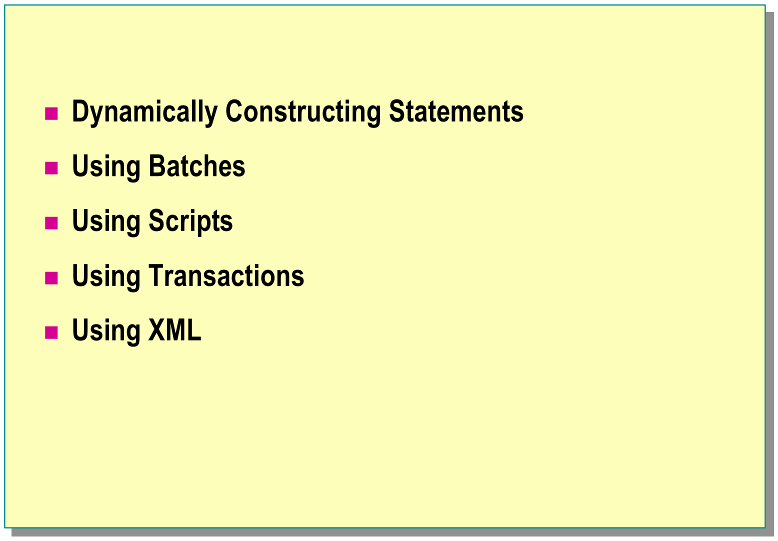
This section of a script is commented to prevent it from executing. This can be helpful when debugging or troubleshooting a script file.

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

◆ Ways to Execute Transact-SQL Statements

Topic Objective
To provide an overview of the ways to execute Transact-SQL statements.

Lead-in
You can execute Transact-SQL statements in a variety of ways.



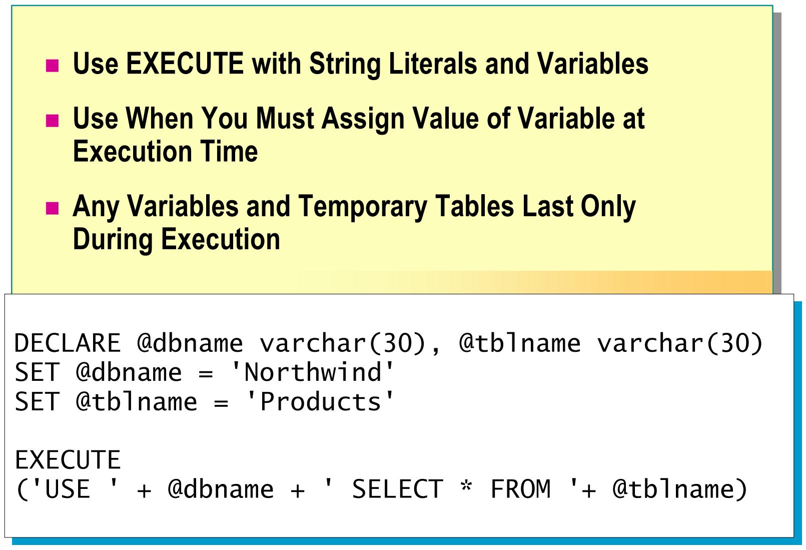
- **Dynamically Constructing Statements**
- **Using Batches**
- **Using Scripts**
- **Using Transactions**
- **Using XML**

You can execute Transact-SQL statements by dynamically constructing statements, and by using batches, scripts, and transactions. You can also use Extensible Markup Language (XML) to present data to Web pages.

Dynamically Constructing Statements

Topic Objective
To introduce students to the dynamic execution of statements.

Lead-in
You can build statements dynamically so that they are constructed at the same time that SQL Server executes a script.



- **Use EXECUTE with String Literals and Variables**
- **Use When You Must Assign Value of Variable at Execution Time**
- **Any Variables and Temporary Tables Last Only During Execution**

```
DECLARE @dbname varchar(30), @tblname varchar(30)
SET @dbname = 'Northwind'
SET @tblname = 'Products'

EXECUTE
('USE ' + @dbname + ' SELECT * FROM '+ @tblname)
```

You can build statements dynamically so that they are constructed at the same time that SQL Server executes a script.

To build a statement dynamically, use the EXECUTE statement with a series of string literals and variables that are resolved at execution time.

Dynamically constructed statements are useful when you want SQL Server to assign the value of the variable when it executes the statement. For example, you can create a dynamic statement that performs the same action on a series of database objects.

Syntax

EXECUTE (**{***@str_var* | '*tsql_string*'} + [{*@str_var* | '*tsql_string*'}...])}

You set options dynamically, and variables and temporary tables that you create dynamically last only as long as it takes for SQL Server to execute the statement.

Consider the following facts about the EXECUTE statement:

- The EXECUTE statement executes statements composed of character strings in a Transact-SQL batch. Because these are string literals, be sure that you add spaces in the appropriate places to ensure proper concatenation.
- The EXECUTE statement can include a string literal, a string local variable, or a concatenation of both.
- All items in the EXECUTE string must consist of character data; you must convert all numeric data before you use the EXECUTE statement.
- You cannot use functions to build the string for execution.
- You can create any valid Transact-SQL statements dynamically, including functions.
- You can nest EXECUTE statements.

Example 1

This example demonstrates how you can use a dynamically executed statement to specify a database context other than the one you are currently in, and then use it to select all of the columns and rows from a specified table. In this example, the change of the database context to the **Northwind** database lasts only for the duration of the query. The current database context is unchanged.

By using a stored procedure, the user could pass the database and table information into the statement as parameters, and then query a specific table in a database.

```
DECLARE @dbname varchar(30), @tablename varchar(30)
SET @dbname = 'Northwind'
SET @tablename = 'Products'

EXECUTE
  ('USE ' + @dbname +
   ' SELECT ProductName FROM ' + @tablename)
```

Result

```
ProductName
---------------------------
Chai
Chang
Aniseed Syrup
```

Example 2

This example demonstrates how you can use a dynamically executed statement to change a database option for the duration of the statement. The following statement does not return a count of the number of rows affected.

```
EXECUTE ('SET NOCOUNT ON '+ 'SELECT LastName, ReportsTo
   FROM Employees WHERE ReportsTo IS NULL')
```

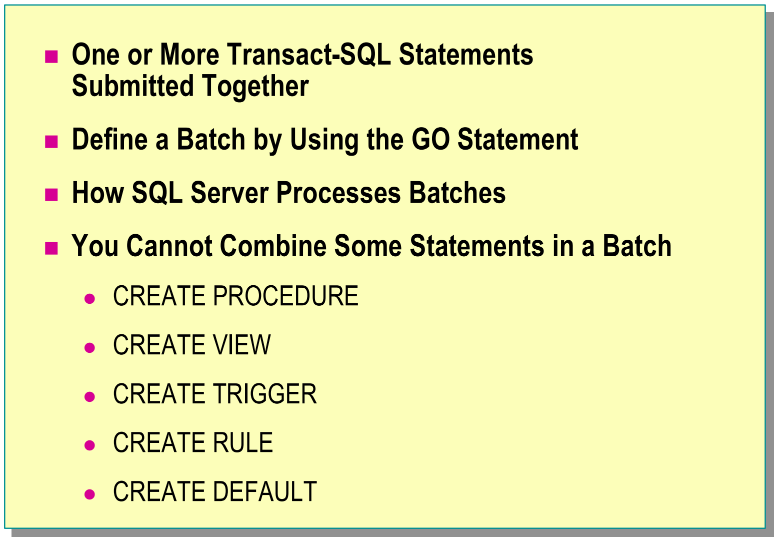
Result

```
LastName    ReportsTo
---------------------------
Fuller      NULL
```

Using Batches

Topic Objective
To introduce students to using batches.

Lead-in
A batch is a set of Transact-SQL statements that are submitted together and executed as a group.



- **One or More Transact-SQL Statements Submitted Together**
- **Define a Batch by Using the GO Statement**
- **How SQL Server Processes Batches**
- **You Cannot Combine Some Statements in a Batch**
  - CREATE PROCEDURE
  - CREATE VIEW
  - CREATE TRIGGER
  - CREATE RULE
  - CREATE DEFAULT

You can also submit one or more statements in a batch.

One or More Transact-SQL Statements Submitted Together

Batches can be run interactively or as part of a script. A script can include more than one batch of Transact-SQL statements.

Define a Batch by Using the GO Statement

Use a GO statement to signal the end of a batch. GO is not a universally accepted Transact-SQL statement; only SQL Query Analyzer and the **osql** utility accept it. Applications based on the ODBC or OLE DB APIs generate a syntax error if they attempt to execute a GO statement.

How SQL Server Processes Batches

SQL Server optimizes, compiles, and executes the statements in a batch together. However, the statements do not necessarily execute as a recoverable unit of work.

The scope of user-defined variables is limited to a batch, so a variable cannot be referenced after a GO statement.

Note If a syntax error exists in a batch, none of the statements in that batch executes. Execution begins with the next batch.

You Cannot Combine Some Statements in a Batch

SQL Server must execute certain object creation statements in their own batches in a script, because of the way that the objects are defined. Each of the following statements is defined by including an object definition header followed by the AS keyword (indicating that one or more statements follow). The object definitions are delimited by the GO statement; SQL Server recognizes the end of the object definition when it reaches the GO statement:

- CREATE PROCEDURE
- CREATE VIEW
- CREATE TRIGGER
- CREATE RULE
- CREATE DEFAULT

Example 1

If you want to use more than one of the non-combinable statements, you must submit multiple batches, as the following script indicates.

```
CREATE DATABASE ...
CREATE TABLE ...
GO

CREATE VIEW1 ...
GO
CREATE VIEW2 ...
GO
```

Example 2

The following example is a batch that fails. To execute it correctly, insert a GO statement before each CREATE TRIGGER statement.

```
CREATE DATABASE ...
CREATE TABLE ...
CREATE TRIGGER ...
CREATE TRIGGER ...
GO
```

Example 3

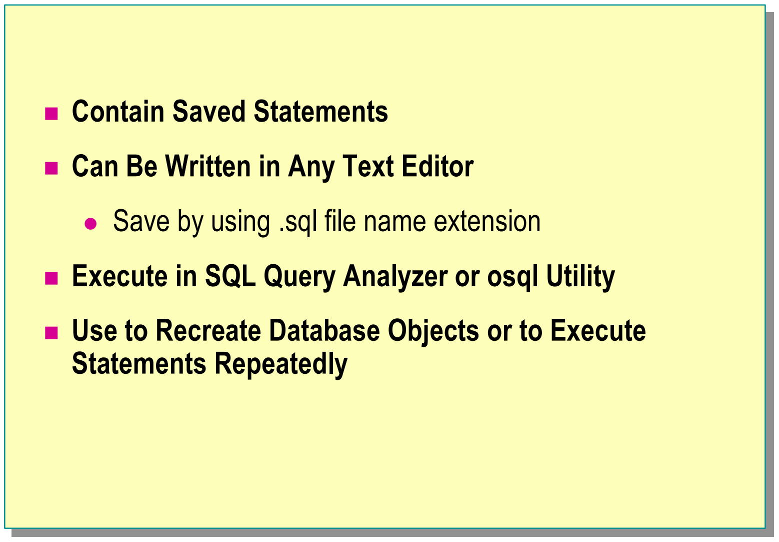
The following example shows how to group the statements of Example 2 so that they execute correctly.

```
CREATE DATABASE ...
CREATE TABLE ...
GO

CREATE TRIGGER ...
GO

CREATE TRIGGER ...
GO
```

Using Scripts

Topic Objective
To introduce using scripts to execute Transact-SQL statements.

Lead-in
Scripts are one of the most common ways to execute Transact-SQL statements.



- **Contain Saved Statements**
- **Can Be Written in Any Text Editor**
  - Save by using .sql file name extension
- **Execute in SQL Query Analyzer or osql Utility**
- **Use to Recreate Database Objects or to Execute Statements Repeatedly**

Scripts are one of the most common ways to execute Transact-SQL statements. A script is one or more Transact-SQL statements that are saved as a file.

You can write and save scripts in SQL Query Analyzer or in any text editor, such as Notepad. Save the script file by using the .sql file name extension.

You can open and execute the script file in SQL Query Analyzer or the **osql** utility (or another query tool).

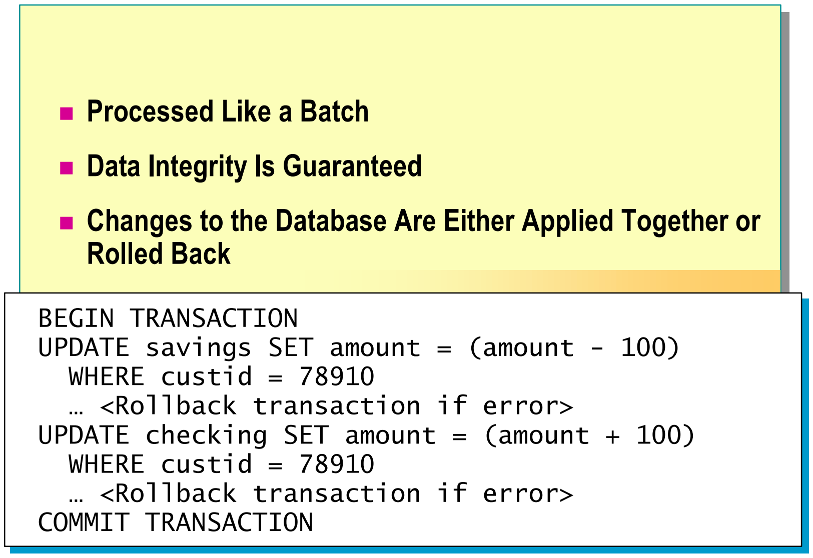
Saved scripts are very useful when recreating databases or data objects, or when you must use a set of statements repeatedly.

Format Transact-SQL statements to be legible to others. Use indenting to indicate levels of relationships.

Using Transactions

Topic Objective
To introduce students to transactions.

Lead-in
Transactions are processed like batches but with a few key differences…



- **Processed Like a Batch**
- **Data Integrity Is Guaranteed**
- **Changes to the Database Are Either Applied Together or Rolled Back**

```
BEGIN TRANSACTION
UPDATE savings SET amount = (amount - 100)
  WHERE custid = 78910
  … <Rollback transaction if error>
UPDATE checking SET amount = (amount + 100)
  WHERE custid = 78910
  … <Rollback transaction if error>
COMMIT TRANSACTION
```

Delivery Tip
The **Savings** and **Checking** tables used in the examples in this topic are fictitious.

Transactions, like batches, are groups of statements that are submitted as a set. However, SQL Server handles transactions as a single unit of work, and the transaction succeeds or fails as a whole. This process maintains data integrity. Transactions can span multiple batches.

Preface a transaction with a BEGIN TRANSACTION statement, and terminate it with a COMMIT TRANSACTION or ROLLBACK TRANSACTION statement.

When a transaction is committed, SQL Server makes the changes to that transaction permanent. When a transaction is rolled back, SQL Server returns any rows affected by the transaction to their pretransaction states.

Partial Syntax

BEGIN TRANSACTION

COMMIT / ROLLBACK TRANSACTION

Example

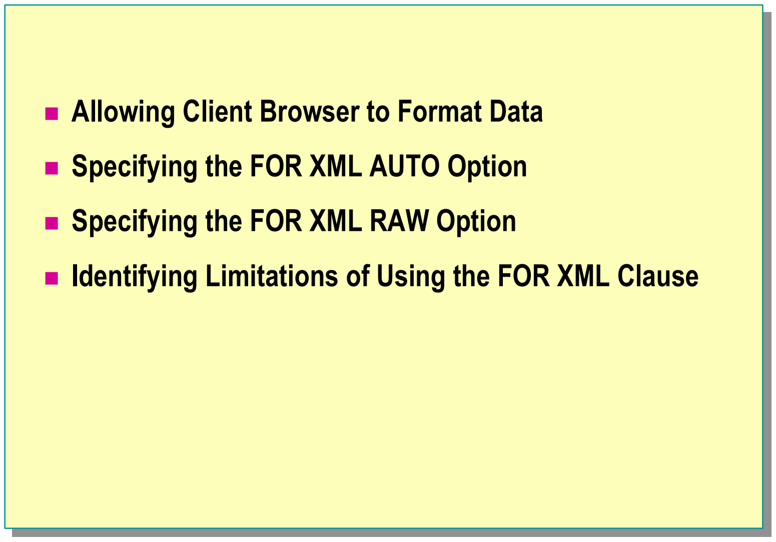
In the following example, $100 is debited from the savings account of customer number 78910, and $100 is credited to the customer's checking account. The customer transferred $100 from savings to checking.

```
BEGIN TRANSACTION
UPDATE savings
   SET balance = (amount - 100)
   WHERE custid = 78910
IF @@ERROR <> 0
   BEGIN
      RAISERROR ('Transaction not completed due to
                  savings account problem.', 16, -1)
      ROLLBACK TRANSACTION
   END
UPDATE checking
   SET balance = (amount + 100)
   WHERE custid = 78910
IF @@ERROR <> 0
   BEGIN
      RAISERROR ('Transaction not completed due to
                  checking account problem.', 16, -1)
      ROLLBACK TRANSACTION
   END
COMMIT TRANSACTION
```

Using XML

Slide Objective
To introduce XML and how a query can return data formatted as well-formed XML.

Lead-in
XML is a programming language that Web developers can use to present data from a SQL Server database to Web pages.



- **Allowing Client Browser to Format Data**
- **Specifying the FOR XML AUTO Option**
- **Specifying the FOR XML RAW Option**
- **Identifying Limitations of Using the FOR XML Clause**

XML is a programming language that Web developers can use to present data from a SQL Server database to Web pages.

Allowing Client Browser to Format Data

When using the FOR XML clause in the SELECT statement, SQL Server:

- Returns the results of a query as a character string.
- Returns the attributes of the data, such as column and table names, as tags. A client browser can then use these tags to format the returned data.

Specifying the FOR XML AUTO Option

You can specify the FOR XML AUTO option to return query results in a standardized format.

Each table in the FROM clause for which at least one column is listed in the SELECT clause is represented as an XML element. An element includes both data and attributes that describe the data.

Example 1

This example selects three columns from two joined tables. Notice that the results combine all of the columns into a single text string.

```
SELECT Orders.OrderID, Shippers.CompanyName, Orders.CustomerID
FROM Orders JOIN Shippers
ON Orders.shipvia = Shippers.ShipperID
WHERE OrderID < 10250
FOR XML AUTO
```

Result

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------
<Orders OrderID="10248" CustomerID="VINET">
  <Shippers CompanyName="Federal Shipping"/>
</Orders>
<Orders OrderID="10249" CustomerID="TOMSP">
  <Shippers CompanyName="Speedy Express"/>
</Orders>
```

Note SQL Server reorders the result set to group columns by table name.

Specifying the FOR XML RAW Option

In some cases, Web developers do not want the automatic formatting. You can specify the RAW option to transform each row in the result set into an XML element with a generic identifier row as the element tag.

Example 2

Compare the result from this example with that of Example 1. This example returns the same data, but the formatting is more generic. Notice that the tables are not named, and the columns are not grouped by table name.

```
SELECT Orders.OrderID, Shippers.CompanyName, Orders.CustomerID
FROM Orders JOIN Shippers
ON Orders.shipvia = Shippers.ShipperID
WHERE OrderID < 10250
FOR XML RAW
```

Result

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
----------------------------------------
<row OrderID="10248"
  CompanyName="Federal Shipping"
  CustomerID="VINET"/>
<row OrderID="10249"
  CompanyName="Speedy Express"
  CustomerID="TOMSP"/>
```

Identifying Limitations of Using the FOR XML Clause

A SELECT statement that contains the FOR XML clause reformats the output for the SQL Server client. Because of these changes, you cannot use a query output in XML format as an input for further SQL Server processing.

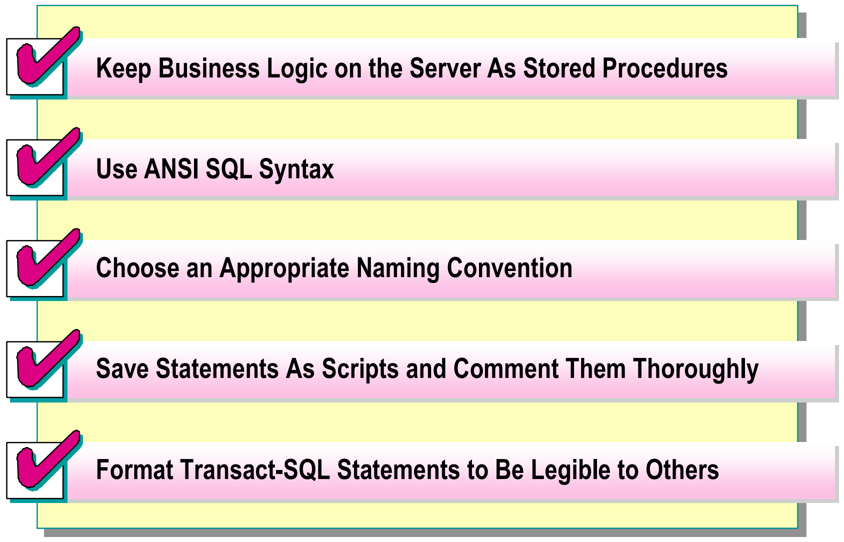
You cannot use XML formatted output in:

- A nested SELECT statement.
- A SELECT INTO statement.
- A COMPUTE BY clause.
- Stored procedures that are called in an INSERT statement.
- A view definition or a user-defined function that returns a rowset.

Recommended Practices

Topic Objective
To present recommended practices for programming with Transact-SQL.

Lead-in
The following are recommended practices for programming with Transact-SQL.



- ✓ Keep Business Logic on the Server As Stored Procedures
- ✓ Use ANSI SQL Syntax
- ✓ Choose an Appropriate Naming Convention
- ✓ Save Statements As Scripts and Comment Them Thoroughly
- ✓ Format Transact-SQL Statements to Be Legible to Others

The following recommended practices should help you to create clean scripts in Transact-SQL:

- Keep business logic on the server as stored procedures.
- Use ANSI SQL syntax when possible to ensure that your scripts are as compatible and portable as possible.
- Choose an appropriate naming convention, and name items consistently.
- Save statements as scripts, and comment them thoroughly.
- Format Transact-SQL statements to be legible to others. Use indenting to indicate levels of relationships.

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

Topic	Search on
Transact-SQL variables	variables
Functions	functions
Transact-SQL tips	transact-sql
Transact-SQL conventions	transact-sql
SQL syntax recommendations	"sql syntax"
Preparing statements	"preparing statements"
osql utility	osql
Reserved keywords	keywords
Ad hoc batch caching	"OLE DB", ODBC
Using XML	"SELECT (T-SQL)"

Lab A: Overview of Transact-SQL

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will write basic SELECT statements, modify a script file, and use system functions.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Write basic SELECT statements that return ordered and limited result sets.
- Modify and execute a script.
- Execute a script by using the **osql** utility.
- Use system functions to retrieve system information.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2073A\Labfiles\L02.
- Answer files for this lab, which are located in C:\Moc\2073A\Labfiles\L02\Answers.

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
Writing Basic SELECT Statements

In this exercise, you will write various statements that return rows from the **Products** table in the **Northwind** database.

► To write a SELECT statement that returns ordered data

In this procedure, you will write a statement that returns all of the rows and columns from the **Products** table and sorts the results in ascending order by the **ProductName** column. C:\Moc\2073A\Labfiles\L02\Answers\Basica.sql is a completed script for this procedure.

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 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 **Northwind**.
4. Write a SELECT statement that returns all of the rows and columns from the **Products** table and sorts the results in ascending order by the **ProductName** column.

   You can execute the **sp_help** system stored procedure on the **Products** table to find the correct column names.

```
SELECT * FROM Products ORDER BY ProductName
```

5. On the toolbar, click **Execute mode**, and then click **Results in grid**.
6. Execute the statement again.

► **To write a SELECT statement that returns limited data**

In this procedure, you will write a statement that retrieves products from a specific category.

- Write a SELECT statement that retrieves all products in category (**CategoryID**) 4 from the **Products** table.

  You can execute the **sp_help** system stored procedure on the **Products** table to find the correct column names.

  ```
  SELECT * FROM Products WHERE CategoryID = 4
  ```

Tip For more information about the SELECT statement (as well as any Transact-SQL statement and system table), select the SELECT keyword in the query window, and then press SHIFT+F1 to open SQL Server Books Online. Double-click **SELECT: clauses**.

Exercise 2
Modifying a Script File

In this exercise, you will modify, save, and execute a simple script file.

► **To modify a script file**

In this procedure, you will execute a script that contains errors. By using the error information that SQL Server returns, you will make changes to the script so that it executes correctly. Then, you will save and execute the script.

1. Open C:\Moc\2073A\Labfiles\L02\Sample_Script.sql, review it, and then execute it.

   You will receive errors when you run this file. These errors are intentional. C:\Moc\2073A\Labfiles\L02\Answers\Sample_Script.sql is a completed script for this procedure.

2. Place comments around the script name and description so that they do not execute.

```
/*
**           Sample_Script.sql
**
**  This script creates the Sample1 table and the
**  Sample_View view. After the objects are created
**  five rows are inserted into the Sample1 table
**  and then queried.
**  This script should be run in the Northwind
**  database.
**
*/
```

3. Add a statement that specifies that the script be executed in the context of the **Northwind** database.

```
USE Northwind
```

4. Include end of batch markers (GO statements) in the proper areas of the script. Only two additional batch markers are necessary.

```
GO
CREATE VIEW Sample_View
  AS
  SELECT cust_no, lname FROM Sample1
GO
```

5. Save the script, and then execute it.

► **To execute a script file by using osql**

In this procedure, you will execute a script file by using the **osql** utility.

1. Open a command prompt window.
2. Type the following command to execute C:\Moc\2073A\Labfiles\L02\Sample_Script2.sql. Make sure that the path is correct.

   ```
   osql /Usa /P /i
   "c:\moc\2073A\labfiles\L02\Sample_Script2.sql"
   ```

   Note Write this command in one line.

Exercise 3
Using System Functions

In this exercise, you will gather system information by using system functions.

► To determine the server process ID

In this procedure, you will observe current server activity and determine the activity that your session is generating.

1. Execute the **sp_who** system stored procedure.

   SQL Server displays all activity that is occurring on the server.
2. To determine which activity is yours, execute the following statement:

   ```
   SELECT @@spid
   ```

   SQL Server returns the server process ID (spid) number of your process in the results.
3. Execute the **sp_who** system stored procedure again, using your spid number as an additional parameter. (In the following statement, *n* represents your spid number.)

   ```
   EXEC sp_who n
   ```

   SQL Server displays the activity related to your spid.

► To retrieve environmental information

In this procedure, you will determine the version of SQL Server that you are running, and you will retrieve connection, database context, and server information. You will perform these tasks by using system functions.

1. Execute the following statement:

   ```
   SELECT @@version
   ```

2. Execute the following statement:

   ```
   SELECT USER_NAME(), DB_NAME(), @@servername
   ```

► **To retrieve metadata**

In this procedure, you will execute several queries to return the metadata from specific database objects by using information schema views. Remember that **INFORMATION_SCHEMA** is a predefined database user who is the owner of the information schema views.

1. Execute the following statement to return a list of all of the user-defined tables in a database:

   ```
   USE Northwind
   SELECT * FROM INFORMATION_SCHEMA.TABLES
      WHERE TABLE_TYPE = 'BASE TABLE'
   ```

2. Execute the following statement to return the primary key and foreign key columns for the **Orders** table:

   ```
   SELECT * FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
      WHERE TABLE_NAME = 'Orders'
   ```

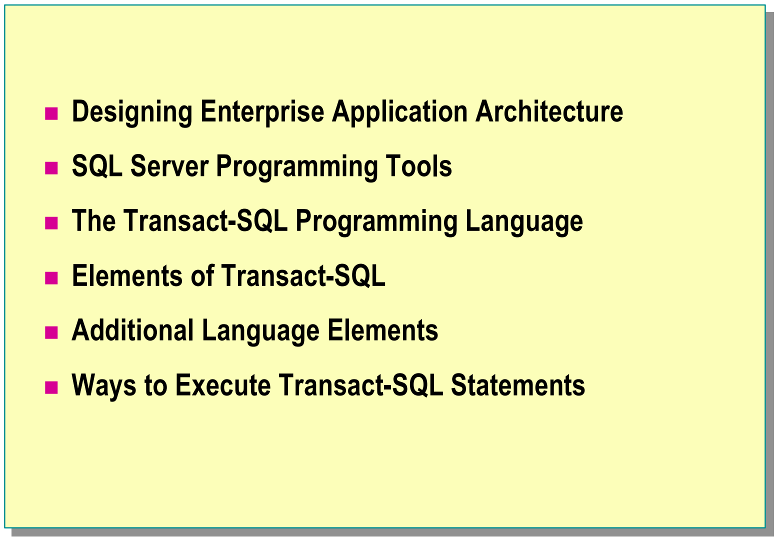
   What column has a primary key defined on it?

   OrderID.

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.



- **Designing Enterprise Application Architecture**
- **SQL Server Programming Tools**
- **The Transact-SQL Programming Language**
- **Elements of Transact-SQL**
- **Additional Language Elements**
- **Ways to Execute Transact-SQL Statements**

1. You are designing a multi-tier application with a Web interface. This application must update a database table frequently. How and where should you implement the logic to perform the update?

   You will probably achieve the best performance by creating a stored procedure on the SQL Server to perform the update. You call this stored procedure from a middle-tier component.

2. Explain the difference between a batch and a script.

   A batch is a set of Transact-SQL statements submitted together, checked for syntax together, and executed as a group. A script is a group of Transact-SQL statements saved as a file.

3. What advantage does a transaction have over a batch or script?

   A transaction executes as a single unit of work. If a transaction fails, it can be rolled back as a unit, leaving data in a consistent state.

4. If you want to include conditional logic in a script, what type of language element would you use? Give as many examples of the language element keywords as you can.

   Control of flow keywords. Examples include BEGIN...END, IF...ELSE, RETURN, WHILE, BREAK, and CONTINUE.