MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 7: Modifying Data

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:
45 Minutes

This module describes how transactions work and discusses how to write INSERT, DELETE, and UPDATE statements to modify data in tables.

Lab:
60 Minutes

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

- Describe how transactions work.
- Write INSERT, DELETE, and UPDATE statements to modify data in tables.
- Describe performance considerations related to modifying data.

Materials and Preparation

Required Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2071A_07.ppt.
- The C:\Moc\2071A\Demo\Ex_07.sql example file contains all of the example scripts from the module, unless otherwise noted in the module.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials.
- Complete the lab.

Module Strategy

Use the following strategy to present this module:

- Using Transactions

  Describe how students can use transactions to modify data.

- Inserting Data

  Explain that rows can be inserted by using the DEFAULT and DEFAULT VALUES keywords to save time during data entry.

  Describe modifying data by using the INSERT…SELECT statement, as well as deleting and updating rows that are based on other tables by using subqueries.

- Deleting Data

  Discuss the use of the DELETE and TRUNCATE TABLE statements to remove rows.

- Updating Data

  Explain how to update data with the UPDATE statement.

  Compare the use of subqueries with the UPDATE statement to the use of a JOIN. Explain to students that there can be a difference in query performance.

- Performance Considerations

  Discuss the performance considerations related to modifying data.

Customization Information

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

Important The lab in this module is dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Classroom Setup Guide* for course 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 modifying data.

- **Using Transactions**
- **Inserting Data**
- **Deleting Data**
- **Updating Data**
- **Performance Considerations**

This module describes how transactions work and discusses how to write INSERT, DELETE, and UPDATE statements to modify data in tables.

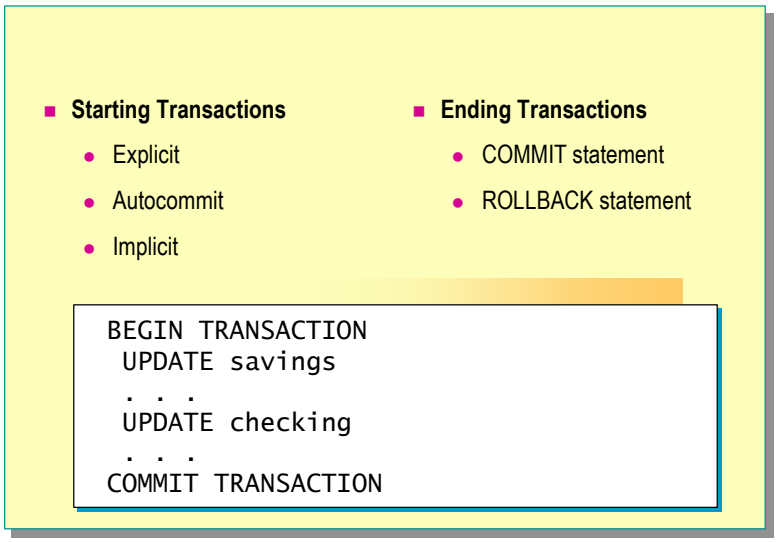
At the end of this module, you will be able to:

- Describe how transactions work.
- Write INSERT, DELETE, and UPDATE statements to modify data in tables.
- Describe performance considerations related to modifying data.

Using Transactions

Slide Objective
To introduce the topics that this section covers.

Lead-in
Transactions are used to enforce data integrity.



A *transaction* is a sequence of operations performed as a single logical unit of work. SQL programmers are responsible for starting and ending transactions at points that enforce the logical consistency of the data. The programmer must define the sequence of data modifications that leave the data in a consistent state relative to the organization's business rules.

Starting Transactions

You can start transactions in Microsoft® SQL Server™ 2000 in one of three modes—*explicit*, *autocommit*, or *implicit*.

- *Explicit* transactions start by issuing a BEGIN TRANSACTION statement.
- *Autocommit* transactions are the default for SQL Server. Each individual Transact-SQL statement is committed when it completes. You do not have to specify any statements to control transactions.
- *Implicit* transactions mode is set by an application programming interface (API) function or the Transact-SQL SET IMPLICIT_TRANSACTIONS ON statement. Using this mode, the next statement automatically starts a new transaction. When that transaction completes, the next Transact-SQL statement starts a new transaction.

The transaction mode is set on a session basis. If one session changes from one transaction mode to another, the change has no effect on the transaction mode session.

Ending Transactions

You can end transactions by using a COMMIT or ROLLBACK statement.

The COMMIT statement indicates that if a transaction is successful, SQL Server should commit it. A COMMIT statement guarantees that all of the transaction's modifications are permanently part of the database. A COMMIT statement also frees resources, such as locks, that the transaction uses.

The ROLLBACK statement cancels a transaction. It backs out all modifications made in the transaction by returning the data to the state in which it was at the start of the transaction. A ROLLBACK statement also frees resources held by the transaction. If an error occurs within a transaction, SQL Server automatically performs a ROLLBACK of the transaction in progress.

Example

This example transfers $100 from a savings account to a checking account for a customer, by using a transaction. It will undo any data changes if there is an error at any point during the transaction.

```
BEGIN TRANSACTION

  UPDATE savings
   SET balance = balance - 100
   WHERE custid = 78910

  IF @@ERROR <> 0
   BEGIN
     RAISERROR ('Error, transaction not completed!', 16, -1)
     ROLLBACK TRANSACTION
   END

  UPDATE checking
   SET balance = balance + 100
   WHERE custid = 78910

  IF @@ERROR <> 0
   BEGIN
     RAISERROR ('Error, transaction not completed!', 16, -1)
     ROLLBACK TRANSACTION
   END

COMMIT TRANSACTION
```

Inserting Data

Slide Objective
To introduce the topics that this section covers.

Lead-in
You can insert data through a transaction by specifying a set of values or inserting the results of a SELECT statement.

- Inserting a Row of Data by Values
- Using the INSERT…SELECT Statement
- Creating a Table Using the SELECT INTO Statement
- Inserting Partial Data
- Inserting Data by Using Column Defaults

You can insert data through a transaction by specifying a set of values or inserting the results of a SELECT statement. You can create a table and insert data simultaneously. You do not have to insert values into all data fields within a row.

Inserting a Row of Data by Values

Slide Objective
To show how you can add a row of values to a table by using the INSERT statement.

Lead-in
The INSERT statement adds rows to tables.

- **Must Adhere to Destination Constraints or the INSERT Transaction Fails**
- **Use a Column List to Specify Destination Columns**
- **Specify a Corresponding List of Values**

```
USE northwind
INSERT customers
      (customerid, companyname, contactname, contacttitle
      ,address, city, region, postalcode, country, phone
      ,fax)

VALUES ('PECOF', 'Pecos Coffee Company', 'Michael Dunn'
         ,'Owner', '1900 Oak Street', 'Vancouver', 'BC'
         ,'V3F 2K1', 'Canada', '(604) 555-3392'
         ,'(604) 555-7293')
GO
```

The INSERT statement adds rows to a table.

Partial Syntax

INSERT [INTO]
{ *table_name* | *view_name*}
{ [**(***column_list***)**]
{ VALUES **(** { DEFAULT | NULL| *expression*}[,...*n*]**)**
| DEFAULT VALUES

Delivery Tip
Point out in the slide example that all values in the **customers** table are character values and, therefore, are enclosed in single quotation marks.

Use the INSERT statement with the VALUES clause to add rows to a table. When you insert rows, consider the following facts and guidelines:

- Must adhere to destination constraints or the INSERT transaction fails.
- Use the *column_list* to specify columns that will store each incoming value. You must enclose the *column_list* in parentheses and delimit it by commas. If you are supplying values for all columns, using the *column_list* is optional.
- Specify the data that you want to insert by using the VALUES clause. The VALUES clause is required for each column in the table or *column_list*.

  The column order and data type of new data must correspond to the table column order and data type. Many data types have an associated entry format. For example, character data and dates must be enclosed in single quotation marks.

Example

The following example adds Pecos Coffee Company as a new customer.

```
USE northwind
INSERT customers
      (customerid, companyname, contactname, contacttitle
      ,address, city, region, postalcode, country, phone
      ,fax)
VALUES ('PECOF', 'Pecos Coffee Company','Michael Dunn'
       ,'Owner', '1900 Oak Street', 'Vancouver', 'BC'
       ,'V3F 2K1', 'Canada', '(604) 555-3392'
       ,'(604) 555-7293')
GO
```

You can verify that Pecos Coffee Company has been added to the **customers** table by executing the following statement.

```
USE northwind
SELECT companyname, contactname
 FROM customers
 WHERE customerid = 'PECOF'
GO
```

Result

Companyname	contactname
Pecos Coffee Company	Michael Dunn

(1 row(s) affected)

Using the INSERT…SELECT Statement

Slide Objective
To show how to insert rows from one table into another by using nested SELECT statements.

Lead-in
You can insert rows from one table into the same or another table by using nested SELECT statements.

- All Rows That Satisfy the SELECT Statement Are Inserted
- Verify That the Table That Receives New Row Exists
- Ensure That Data Types Are Compatible
- Determine Whether Default Values Exist or Whether Null Values Are Allowed

```
USE northwind
INSERT customers
 SELECT substring(firstname, 1, 3)
         + substring (lastname, 1, 2)
       ,lastname, firstname, title, address, city
       ,region, postalcode, country, homephone, NULL
  FROM employees
GO
```

The INSERT…SELECT statement adds rows to a table by inserting the result set of a SELECT statement.

Use the INSERT…SELECT statement to add rows to an existing table from other sources. Using the INSERT…SELECT statement is more efficient than writing multiple, single-row INSERT statements. When you use the INSERT…SELECT statement, consider the following facts and guidelines:

- All rows that satisfy the SELECT statement are inserted into the outermost table of the query.
- You must verify that the table that receives the new rows exists in the database.
- You must ensure that the columns of the table that receives the new values have data types compatible with the columns of the table source.
- You must determine whether a default value exists or whether a null value is allowed for any columns that are omitted. If null values are not allowed, you must provide values for these columns.

Syntax

INSERT *table_name*
SELECT *column_list*
FROM *table_list*
WHERE *search_conditions*

Example

This example adds new customers to the **customers** table. Employees of Northwind Traders are eligible to buy company products. This query contains an INSERT statement with a SELECT clause that adds employee information to the **customers** table.

The new **customerid** column consists of the first three letters of the employee's first name, concatenated with the first two letters of the last name. The employee's last name is used as the new company name, and the first name is used as the contact name.

```
USE northwind
INSERT customers
 SELECT substring (firstname, 1, 3)
         + substring (lastname, 1, 2)
       ,lastname, firstname, title, address, city
       ,region, postalcode, country, homephone, NULL
 FROM employees
GO
```

Creating a Table Using the SELECT INTO Statement

Slide Objective
To explain the purpose and function of the SELECT INTO statement.

Lead-in
You can place the result set of any query into a new table by using the SELECT INTO statement.

- **Use to Create a Table and Insert Rows into the Table in a Single Operation**
- **Create a Local or Global Temporary Table**
- **Set the select into/bulkcopy Database Option ON in Order to Create a Permanent Table**
- **Create Column Alias or Specify Column Names in the Select List for New Table**

```
USE northwind
SELECT productname AS products
      ,unitprice AS price
      ,(unitprice * 1.1) AS tax
 INTO #pricetable
 FROM products
GO
```

You can place the result set of any query into a new table by using the SELECT INTO statement.

Use the SELECT INTO statement to populate new tables in a database with imported data. You also can use the SELECT INTO statement to break down complex problems that require a data set from various sources. If you first create a temporary table, the queries that you execute on it are simpler than those you would execute on multiple tables or databases.

When you use the SELECT INTO statement, consider the following facts and guidelines:

- You can use the SELECT INTO statement to create a table and to insert rows into the table in a single operation.

  Ensure that the table name that is specified in the SELECT INTO statement is unique. If a table exists with the same name, the SELECT INTO statement fails.

- You can create a local or global temporary table.

  Create a local temporary table by preceding the table name with a number sign (#), or create a global temporary table by preceding the table name with a double number sign (##).

  A local temporary table is visible in the current session only. A global temporary table is visible to all sessions:

  - Space for a local temporary table is reclaimed when the user ends the session.
  - Space for a global temporary table is reclaimed when the table is no longer used by any session.

- Set the **select into/bulkcopy** database option ON in order to create a permanent table.
- You must create column aliases or specify the column names of the new table in the select list.

Partial Syntax

SELECT <*select_list*>
  INTO *new_table*
  FROM {<*table_source*>}[,...*n*]
  WHERE <*search_condition*>

Example

This example creates a local temporary table based on a query made on the **products** table. Notice that you can use string and mathematical functions to manipulate the result set.

Delivery Tip
Demonstrate this example by using SQL Query Analyzer.

```
USE northwind
SELECT productname AS products
      ,unitprice AS price
      ,(unitprice * 1.1) AS tax
 INTO #pricetable
 FROM products
GO
```

To view your result set, you must execute the following query.

```
USE northwind
SELECT * FROM #pricetable
GO
```

Result

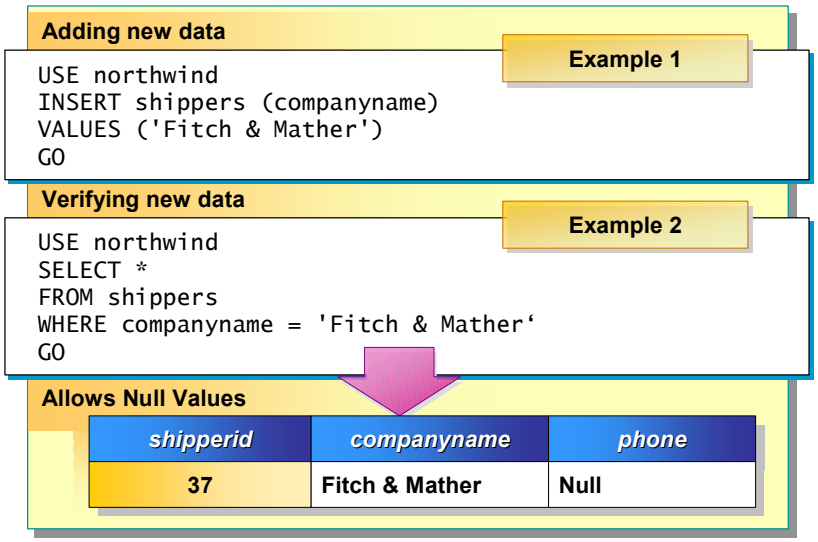
products	price	tax
Chai	18	19.8
Chang	19	20.9
Aniseed Syrup	10	11
Chef Anton's Cajun Seasoning	22	24.2
Chef Anton's Gumbo Mix	21.35	23.485
Grandma's Boysenberry Spread	27.5	30.25
Uncle Bob's Organic Dried Pears	33	36.3
Northwoods Cranberry Sauce	44	48.4
Mishi Kobe Niku	97	106.7
Ikura	31	34.1
Queso Cabrales	21	23.1
Queso Manchego La Pastora	38	41.8
Konbu	6	6.6
Tofu	23.25	25.575
Genen Shouyu	15.5	17.05
.		
.		
.		

(77 row(s) affected)

Inserting Partial Data

Slide Objective
To explain how to insert a row without supplying all of the data items.

Lead-in
If a column has a default value or accepts null values, you can leave the column out of an INSERT statement. SQL Server automatically inserts the values.



Delivery Tip
Compare Example 1 to Example 2. The DEFAULT keyword is not used in Example 1. Both examples return the same result set.

If a column has a default value or accepts null values, you can omit the column from an INSERT statement. SQL Server automatically inserts the values.

When you insert partial data, consider the following facts and guidelines:

- List only the column names for the data that you are supplying in the INSERT statement.
- Specify the columns for which you are providing a value in the *column_list*. The data in the VALUES clause corresponds to the specified columns. Unnamed columns are filled in as if they had been named and a default value had been supplied.
- Do not specify columns in the *column_list* that have an IDENTITY property or that allow default or null values.
- Enter a null value explicitly by typing Null without single quotation marks.

Example 1

This example adds the company Fitch & Mather as a new shipper in the **shippers** table. Data is not entered for columns that have an IDENTITY property or that allow default or null values. Compare this example with Example 2. Notice that the DEFAULT keyword is omitted.

```
USE northwind
INSERT shippers (companyname)
VALUES ('Fitch & Mather')
GO
```

You can verify that Fitch & Mather has been added to the **shippers** table by executing the following statement.

```
USE northwind
SELECT *
FROM shippers
WHERE companyname = 'Fitch & Mather'
GO
```

Result

shipperid	companyname	phone
37	Fitch & Mather	NULL

```
(1 row(s) affected)
```

Example 2

This example also adds Fitch & Mather as a new shipper in the **shippers** table. Notice that the DEFAULT keyword is used for columns that allow default or null values. Compare this example to Example 1.

```
USE northwind
INSERT shippers (companyname, Phone)
VALUES ('Fitch & Mather', DEFAULT)
GO
```

Result

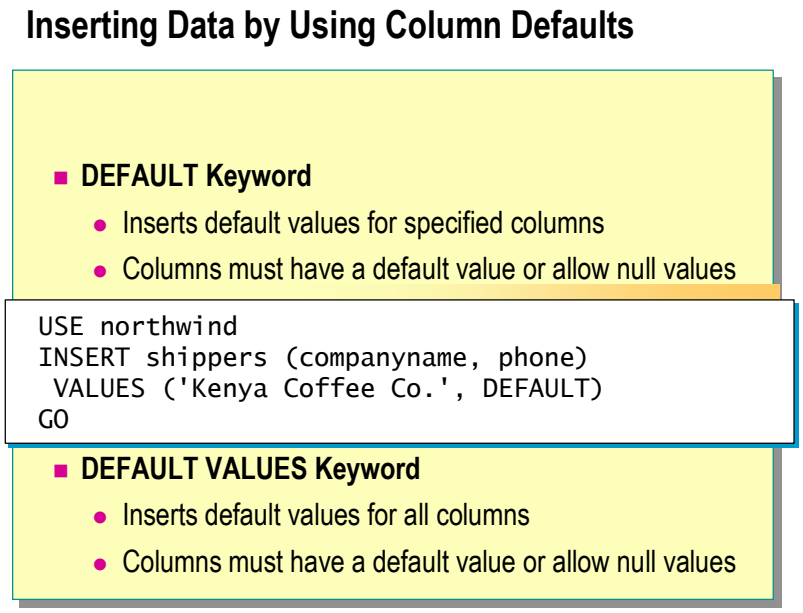
shipperid	companyname	phone
37	Fitch & Mather	NULL

```
(1 row(s) affected)
```

Inserting Data by Using Column Defaults

Slide Objective
To discuss the DEFAULT and DEFAULT VALUES keywords.

Lead-in
Use an INSERT statement with the DEFAULT keyword to insert the default value for specific columns, or use the DEFAULT VALUES keyword to insert an entire row in a table.



When you insert rows into a table, you can save time when entering values by using the DEFAULT or DEFAULT VALUES keywords with the INSERT statement.

Delivery Tip
Focus on the partial syntax and compare the DEFAULT keyword to the DEFAULT VALUES keyword in the syntax.

DEFAULT Keyword

When a table has default constraints, or when a column has a default value, use the DEFAULT keyword in the INSERT statement to have SQL Server supply the default value for you.

When you use the DEFAULT keyword, consider the following facts and guidelines:

- SQL Server inserts a null value for columns that allow null values and do not have default values.
- If you use the DEFAULT keyword, and the columns do not have default values or allow null values, the INSERT statement fails.
- You cannot use the DEFAULT keyword with a column that has the IDENTITY property (an automatically assigned, incremented value). Therefore, do not list columns with an IDENTITY property in the *column_list* or VALUES clause.
- SQL Server inserts the next appropriate value for columns that are defined with the **rowversion** data type.

Example

This example inserts a new row for the Kenya Coffee Company without using a *column_list*. The **shippers.shipperid** column has an IDENTITY property and is not included in the column list. The **phone** column allows null values.

```
USE northwind
INSERT shippers (companyname, phone)
 VALUES ('Kenya Coffee Co.', DEFAULT)
GO
```

You can verify that Kenya Coffee Co. has been added to the **shippers** table by executing the following statement.

```
USE northwind
SELECT *
 FROM shippers
 WHERE companyname = 'Kenya Coffee Co.'
GO
```

Result

shipperid	companyname	Phone
10	Kenya Coffee Co.	NULL

```
(1 row(s) affected)
```

DEFAULT VALUES Keyword

Use the DEFAULT VALUES keyword to insert an entire row into a table. When you use the DEFAULT VALUES keyword, consider the following facts and guidelines:

- SQL Server inserts a null value for columns that allow null values and do not have a default value.
- If you use the DEFAULT VALUES keyword, and the columns do not have default values or allow null values, the INSERT statement fails.
- SQL Server inserts the next appropriate value for columns with an IDENTITY property or a **rowversion** data type.
- Use the DEFAULT VALUES keyword to generate sample data and populate tables with default values.

Deleting Data

Slide Objective
To introduce the topics that this section covers.

Lead-in
You can specify the data that you want to delete.

- Using the DELETE Statement
- Using the TRUNCATE TABLE Statement
- Deleting Rows Based on Other Tables

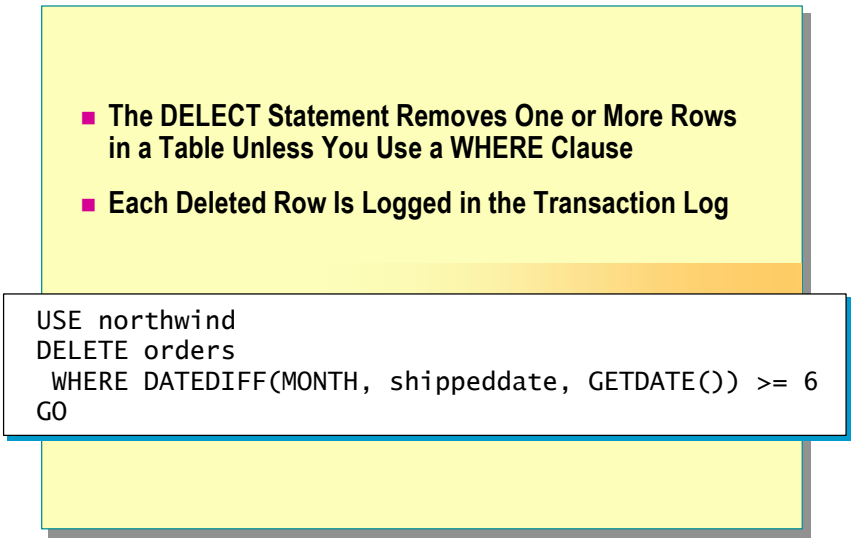
You can specify the data that you want to delete.

The DELETE statement removes one or more rows from a table or view by using a transaction. You can specify which rows SQL Server deletes by filtering on the targeted table, or by using a JOIN clause or a subquery. The TRUNCATE TABLE statement is used to remove all rows from a table without using a transaction.

Using the DELETE Statement

Slide Objective
To discuss how to delete rows from tables.

Lead-in
The DELETE statement removes rows from tables.



The DELETE statement removes rows from tables. Use the DELETE statement to remove one or more rows from a table.

Partial Syntax

DELETE [*from*] {*table_name*|*view_name*}
WHERE *search_conditions*

Example

This example deletes all order records that are equal to or greater than six months old.

```
USE northwind
DELETE orders
 WHERE DATEDIFF(MONTH, shippeddate, GETDATE()) >= 6
GO
```

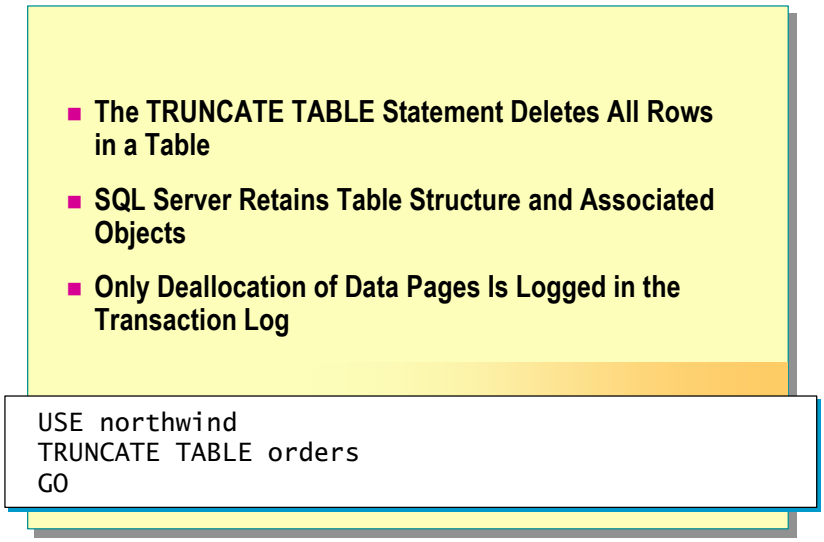
When you use the DELETE statement, consider the following facts:

- SQL Server deletes all rows from a table unless you include a WHERE clause in the DELETE statement.
- Each deleted row is logged in the transaction log.

Using the TRUNCATE TABLE Statement

Slide Objective
To describe how to use the TRUNCATE TABLE statement.

Lead-in
The TRUNCATE TABLE statement removes all data from a table.



THE TRUNCATE TABLE removes all data from a table. Use the TRUNCATE TABLE statement to perform a nonlogged deletion of all rows.

Syntax

TRUNCATE TABLE [[*database.*]*owner.*]*table_name*

Example

This example removes all data from the **orders** table.

```
USE northwind
TRUNCATE TABLE orders
GO
```

When you use the TRUNCATE TABLE statement, consider the following facts:

- SQL Server deletes all rows but retains the table structure and its associated objects.
- The TRUNCATE TABLE statement executes more quickly than the DELETE statement because SQL Server logs only the deallocation of data pages.
- If a table has an IDENTITY column, the TRUNCATE TABLE statement resets the seed value.

For Your Information
You cannot use TRUNCATE TABLE on a table referenced by a FOREIGN KEY constraint; instead, use DELETE statement without a WHERE clause.

Deleting Rows Based on Other Tables

Slide Objective
To show how to delete data from a table based on data in other tables.

Lead-in
You can use the DELETE statement with an additional FROM clause (or a subquery in the WHERE clause) to look at data in other tables and determine whether a row should be deleted.

- **Using an Additional FROM Clause**
  - First FROM clause indicates table to modify
  - Second FROM clause specifies restricting criteria for the DELETE statement
- **Specifying Conditions in the WHERE Clause**
  - Subqueries determine which rows to delete

Delivery Tip
Compare Examples 1 and 2.

Use the DELETE statement with joins or subqueries to remove rows from a table based on data stored in other tables. This is more efficient than writing multiple, single-row DELETE statements.

Using an Additional FROM Clause

In a DELETE statement, the WHERE clause references values in the table itself and is used to decide which rows to delete. If you use an additional FROM clause, you can reference other tables to make this decision. When you use the DELETE statement with an additional FROM clause, consider the following facts:

- The first FROM clause indicates the table from which the rows are deleted.
- The second FROM clause may introduce a join and acts as the restricting criteria for the DELETE statement.

Syntax

```
DELETE [FROM] {table_name | view_name}
  [FROM {<table_source>} [,...n]]
  [WHERE search_conditions ]
```

Example 1

This example uses a join operation with the DELETE statement to remove rows from the **order details** table for orders taken on 4/14/1998.

Delivery Tip
Point out the additional FROM clause in the statement.

```
USE northwind
DELETE FROM [order details]
 FROM orders AS o
 INNER JOIN [order details] AS od
  ON o.orderid = od.orderid
 WHERE orderdate = '4/14/1998'
GO
```

Specifying Conditions in the WHERE Clause

You also can use subqueries to determine which rows to delete from a table based on rows of another table. You can specify the conditions in the WHERE clause rather than using an additional FROM clause. Use a nested or correlated subquery in the WHERE clause to determine which rows to delete.

Example 2

This example removes the same rows in the **order details** table as Example 1 and shows that you can convert a join operation to a nested subquery.

```
USE northwind
DELETE FROM [order details]
 WHERE orderid IN (
                   SELECT orderid
                    FROM orders
                    WHERE orderdate = '4/14/1998'
                  )
GO
```

◆ Updating Data

Slide Objective
To introduce the topics that this section covers.

Lead-in
The UPDATE statement can change data values in single rows, groups of rows, or all rows in a table or view.

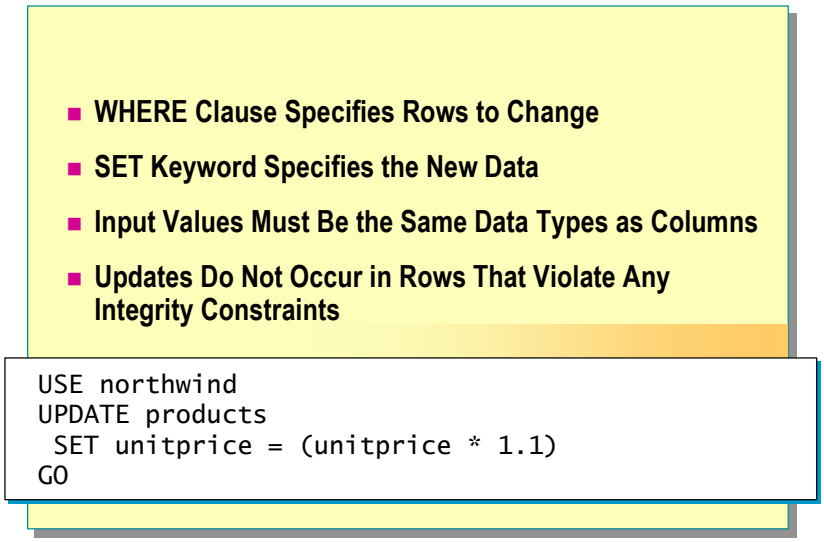
- **Updating Rows Based on Data in the Table**
- **Updating Rows Based on Other Tables**

The UPDATE statement can change data values in single rows, groups of rows, or all rows in a table or view. You can update a table based on data in the table or on data in other tables.

Updating Rows Based on Data in the Table

Slide Objective
To show how to update rows by using the UPDATE statement.

Lead-in
You can use the UPDATE statement to change data in existing rows of a table.



The UPDATE statement modifies existing data.

Partial Syntax

```
UPDATE {table_name | view_name}
  SET {   column_name = {expression | DEFAULT | NULL}   |
  @variable=expression}[,…n]
  WHERE {search_conditions}
```

Use the UPDATE statement to change single rows, groups of rows, or all of the rows in a table. When you update rows, consider the following facts and guidelines:

- Specify the rows to update with the WHERE clause.
- Specify the new values with the SET clause.
- Verify that the input values are the same as the data types that are defined for the columns.
- SQL Server does not update rows that violate any integrity constraints. The changes do not occur, and the statement is rolled back.
- You can change the data in only one table at a time.
- You can set one or more columns or variables to an expression. For example, an expression can be a calculation (like **price** * 2) or the addition of two columns.

Example

The following example adds 10 percent to the current prices of all Northwind Traders products.

```
USE northwind
UPDATE products
 SET unitprice = (unitprice * 1.1)
GO
```

Updating Rows Based on Other Tables

Slide Objective
To show how you can use joins or subqueries to update data in one table based on data in another table.

Lead-in
You can use the UPDATE statement to update rows based on other tables.

- **How the UPDATE Statement Works**
  - Never updates the same row twice
  - Requires table prefixes on ambiguous column names
- **Specifying Rows to Update Using Joins**
  - Uses the FROM clause
- **Specifying Rows to Update Using Subqueries**
  - Correlates the subquery with the updated table

Use the UPDATE statement with a FROM clause to modify a table based on values from other tables.

Using the UPDATE Statement

When you use joins and subqueries with the UPDATE statement, consider the following facts and guidelines:

- SQL Server never updates the same row twice in a single UPDATE statement. This is a built-in restriction that minimizes the amount of logging that occurs during updates.
- Use the SET keyword to introduce the list of columns or variable names to be updated. Columns referenced by the SET keyword must be unambiguous. For example, you can use a table prefix to eliminate ambiguity.

Partial Syntax

```
UPDATE {table_name | view_name}
  SET
  { column_name={expression | DEFAULT | NULL}
    |@variable=expression}[,…n]
  [FROM {   <table_source>
  ]
  [WHERE search_conditions]
```

Specifying Rows to Update Using Joins

When you use joins to update rows, use the FROM clause to specify joins in the UPDATE statement.

Example 1

This example uses a join to update the **products** table by adding $2.00 to the **unitprice** column for all products supplied by suppliers in the United States (USA).

```
UPDATE products
 SET unitprice = unitprice + 2
 FROM products
 INNER JOIN suppliers
  ON products.supplierid = suppliers.supplierid
 WHERE suppliers.country = 'USA'
GO
```

Specifying Rows to Update Using Subqueries

When you use subqueries to update rows, consider the following facts and guidelines:

- If the subquery does not return a single value, you must introduce the subquery with the IN, EXISTS, ANY or ALL keyword.
- Consider using aggregate functions with correlated subqueries, because SQL Server never updates the same row twice in a single UPDATE statement.

Example 2

This example uses a subquery to update the **products** table by adding $2.00 to the **unitprice** column for all products supplied by suppliers in the in the United States (USA). Notice that each product has only one supplier.

```
UPDATE products
 SET unitprice = unitprice + 2
 WHERE supplierid IN (
                      SELECT supplierid
                       FROM suppliers
                       WHERE country = 'USA'
                     )
GO
```


Example 3

This example updates the total sales for all orders of each product in the **products** table. Many orders for each product may exist. Because SQL Server never updates the same row twice, you must use an aggregate function with a correlated subquery to update the total number of sales-to-date of each product. If you want to execute the following example, you must add a **todatesales** column with a default value of 0 to the **products** table.

```
USE northwind
UPDATE products
 SET todatesales = (
                    SELECT SUM(quantity)
                     FROM [order details] AS od
                     WHERE products.productid = od.productid
                    )
GO
```

Performance Considerations

Slide Objective
To describe the performance considerations when using views, triggers, or stored procedures.

Lead-in
Data modifications that occur within transactions can affect the performance of SQL Server.

- **All Data Modifications Occur Within a Transaction**
- **Data Page Allocation May Occur**
- **Modifying Indexed Data Incurs Additional Overhead**
- **Indexes Can Assist Search Criteria**

Key Points
Only one transaction at a time can modify a specific row. An ongoing transaction blocks other transactions from executing until the transaction is committed or rolled back.

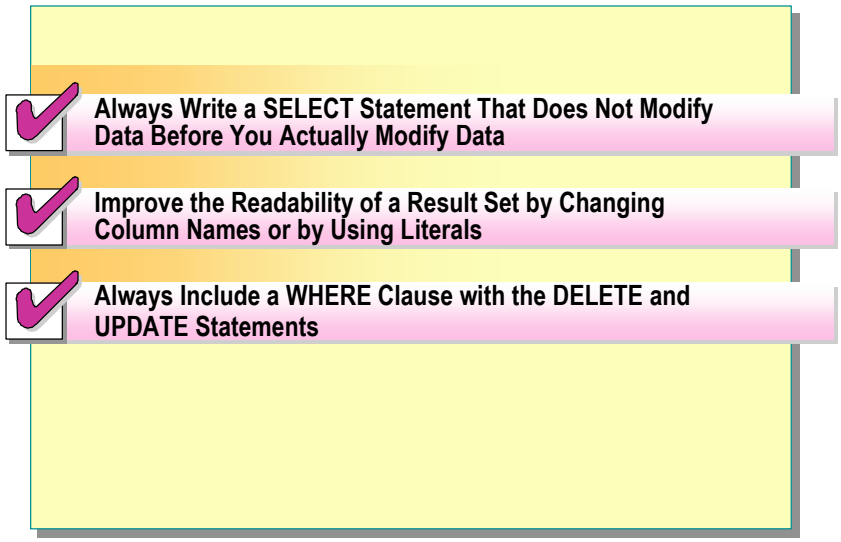
Data modifications that occur within transactions can affect the performance of SQL Server. When modifying data, remember that:

- Data locking during a single transaction can prevent other transactions and queries from running until the transaction completes.
- Modifying tables can change the way data is physically stored, leading to data page allocations that must occur within the transaction.
- When modifying data columns that are indexed, the indexes on those columns change as part of the transaction.
- Placing indexes on columns used in the WHERE clause of a data modification statement improves performance.

Recommended Practices

Slide Objective
To list the recommended practices for performing basic queries.

Lead-in
The following recommended practices should help you perform basic queries.



The following recommended practices should help you perform basic queries:

- Always write a SELECT statement that does not modify data before you actually modify data. This test verifies which rows your INSERT, UPDATE, or DELETE statement affects.
- Improve the readability of result sets by changing column names to column aliases or using literals to replace result set values. These formatting options change the presentation of the data, not the data itself.
- SQL Server deletes or updates all rows in a table unless you include a WHERE clause in the DELETE or UPDATE statements.

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

Topic	Search on
Using character strings	"LIKE comparisons"
Sorting result sets	“sort order”

Lab A: Modifying Data

Slide Objective
To introduce the lab.

Lead-in
In this lab, you will modify existing data by using INSERT, DELETE, and UPDATE statements.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Modify data in tables by using the INSERT, DELETE, and UPDATE statements.
- Insert rows into a table by using the DEFAULT and DEFAULT VALUES keywords.
- Modify data based on data in other tables.

Prerequisites

Before working on this lab, you must have:

- The script files for this lab, which are located in C:\Moc\2071A\Labfiles\L07.
- Answer files for this lab, which are located in C:\Moc\2071A\Labfiles\L07\Answers.
- The **library** database installed.

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 **library** database schema.
- Microsoft® 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: 60 minutes

Exercise 1
Using the INSERT Statement

In this exercise, you will use the INSERT statement to add rows to tables in the **library** database. Then, you will execute a query to verify that the new rows are added to the tables. C:\Moc\2071A\Labfiles\L07\Answers contains completed scripts for this exercise.

► To insert values into the item table

In this procedure, you will insert rows into the **item** table to represent a book in the library collection.

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

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

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

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

3. In the **DB** list, click **library**.

4. Insert two rows into the **item** table for title number 8, *The Cherry Orchard*. Specify the column names for which you are supplying values. Make the first item a hardback and the second item a paperback, and use the following values. Answer_InsValues1.sql is a completed script for this step.

Column name	Data
Isbn	10001 for HARDBACK; 10101 for SOFTBACK
title_no	8
Cover	HARDBACK and SOFTBACK
Loanable	Y
Translation	ENGLISH

```
USE library
INSERT item (isbn, title_no, cover, loanable, translation)
 VALUES (10001, 8, 'HARDBACK', 'Y', 'ENGLISH')
INSERT item (isbn, title_no, cover, loanable, translation)
 VALUES (10101, 8, 'SOFTBACK', 'Y', 'ENGLISH')
GO
```

5. Execute the query and verify that a single row is inserted in two different transactions.

► To insert values into the copy table

In this procedure, you will insert rows into the **copy** table to represent a book in the library collection.

1. Add a row into the **copy** table for the hardback item that you added in step 4, and use the following values. Answer_InsValues2.sql is a completed script for this step.

Column name	Data
Isbn	10001 (the ISBN number for the hardback item that you added in step 1)
copy_no	1
title_no	8 (the title number of *The Cherry Orchard*)
On_loan	N

```
USE library
INSERT copy (isbn, copy_no, title_no, on_loan)
 VALUES (10001,1,8,'N')
GO
```

2. Execute the query and verify that a single row is inserted.

► To determine the translation language of an item

In this procedure, you will write a query that returns the language that an item has been translated into.

1. Write a query that returns the translation of one of the items that you inserted in step 4 from the **item** table. Answer_Translate.sql is a completed script for this step.

```
USE library
SELECT translation
 FROM item
 WHERE isbn = 10001
GO
```

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

Exercise 2
Using the INSERT Statement with the DEFAULT Keyword

In this exercise, you will use the INSERT statement with the DEFAULT keyword to add two rows to the **title** table and to provide partial data for columns that allow null values or have default values. C:\Moc\2071A\Labfiles\L07\Answers contains completed scripts for this exercise.

► To determine which columns allow null values

In this procedure, you will determine which columns allow null values.

1. Execute the **sp_help** system stored procedure to determine which columns in the **title** table allow null values. You do not have to supply values for columns that allow null values or have default values or have the IDENTITY property. Answer_WhichNull.sql is a completed script for this step.

```
USE library
EXEC sp_help title
GO
```

2. Review the second results returned to determine which columns allow null values.

► To insert values into the title table

In this procedure, you will insert values into the **title** table.

1. Insert a row into the **title** table for the book, *The Art of Lawn Tennis*, by William T. Tilden. Use the DEFAULT keyword for columns that allow null values or that have default values. Do not supply a value for the **title_no** column because this column has the IDENTITY property. Answer_InsDefault1.sql is a completed script for this step.

```
USE library
INSERT title (title, author, synopsis)
 VALUES ('The Art of Lawn Tennis', 'William T. Tilden'
        ,DEFAULT )
GO
```

2. Execute the query and verify that a single row is inserted.

► To determine the last identity value used

In this procedure, you will determine the last identity value used.

1. Write query to determine the **title_no** of the title that you added in step 1 of the previous procedure. Answer_Identity.sql is a completed script for this step.

```
USE library
SELECT @@identity
GO
```

2. Execute the query and make note of the value that is returned.

► To retrieve the last row inserted into the title table

In this procedure, you will retrieve the last row inserted into the **title** table.

1. Write a query to verify that the new title was added to the **title** table. Use the value that you obtained in step 1 of the previous procedure for the **title_no** column. Answer_LastRow.sql is a completed script for this step.

```
USE library
SELECT *
 FROM title
 WHERE title_no = @@identity
GO
```

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

► To insert more values into the title table

In this procedure, you will insert more values into the **title** table.

1. Insert a row into the **title** table for the title, *Riders of the Purple Sage*, by Zane Grey. Specify a *column_list* and values for columns that do not allow null values or have default values. Answer_InsValues3.sql is a completed script for this step.

```
USE library
INSERT title (title, author)
 VALUES ('Riders of the Purple Sage', 'Zane Grey')
GO
```

2. Execute the query and verify that a single row is inserted.

► To verify that values were inserted into the title table

In this procedure, you will verify that values were inserted into the **title** table.

1. Write and execute a query to verify that the new member was added to the **member** table. Answer_ChkValues3.sql is a completed script for this step.

```
USE library
SELECT *
 FROM title
 WHERE title = 'Riders of the Purple Sage'
GO
```

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

Exercise 3
Using the INSERT Statement with the DEFAULT VALUES Keyword

In this exercise, you will use the INSERT statement with the DEFAULT VALUES keyword to add rows to a table without providing values. First you will create and work with a sample table in the **library** database. C:\Moc\2071A\Labfiles\L07\Answers contains completed scripts for this exercise.

► To create the sample1 table

In this procedure, you will create a new table in the **library** database that allows null values and that specifies default values for some columns.

1. Execute the C:\Moc\2071A\Labfiles\L07\ MakeSample1.sql script to create a new table called **sample1** in the **library** database with the following characteristics.

Column name	Datatype	IDENTITY property?	Allows NULL?
Cust_id	**Int**	Yes (100,5)	No
Name	**char(10)**	No	Yes

```
USE LIBRARY
CREATE TABLE sample1 (
     Cust_id    int        NOT NULL  IDENTITY(100,5)
    ,Name       char(10)   NULL
    )
GO
```

2. Execute the query to verify that it creates the **sample1** table.

► To insert a row of default values into the sample1 table

In this procedure, you will insert a row into the **sample1** table by using the DEFAULT VALUES keyword. Then, you will write and execute a query to verify that the new row was added to the table. Answer_InsDefault2.sql is a completed script for this procedure.

1. Write a query that would insert a new row into the **sample1** table without specifying the column names. Use the DEFAULT VALUES keyword with the INSERT statement.

```
USE LIBRARY
INSERT sample1
 DEFAULT VALUES
GO
```

2. Execute the query and verify that a single row is inserted.

► To verify that values that were inserted into the sample1 table

In this procedure, you will verify that values that were inserted into the **sample1** table.

1. Write a query to verify that the new row was added to the **sample1** table. Answer_ChkDefault2.sql is a completed script for this procedure.

```
USE LIBRARY
SELECT *
 FROM sample1
GO
```

2. Execute the query and compare the results to the default values defined for the table.

Result

Your result should look similar to the following result set.

cust_id	name
100	NULL

```
(1 row(s) affected)
```

Exercise 4
Using the DELETE Statement

In this exercise, you will use the DELETE statement to remove a book with an ISBN of 10101 and a title number of 8 from the **item** table in the **library** database. C:\Moc\2071A\Labfiles\L07\Answers contains completed scripts for this exercise.

► To retrieve a row of data that you intend on deleting from the item table

In this procedure, you will retrieve a row of data that you want to delete from the **item** table. Answer_SelDelete1.sql is a completed script for this procedure.

1. Write a query that returns the row from the **item** table that represents a paperback copy (**isbn** 10101) of *The Cherry Orchard* (**title_no** 8).

```
USE library
SELECT *
 FROM item
 WHERE isbn = 10101
  AND  title_no = 8
GO
```

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

► To delete a specific row of data from the item table

In this procedure, you will delete a specific row of data from the **item** table. Answer_Delete1.sql is a completed script for this procedure.

1. Modify the query from step 1 in the previous procedure so that it deletes the row from the **item** table that represents a paperback copy (**isbn** 10101) of *The Cherry Orchard* (**title_no** 8).

```
USE library
DELETE FROM item
 WHERE isbn = 10101
  AND  title_no = 8
GO
```

2. Execute the query and confirm that it one row is deleted from the **item** table.

Exercise 5
Using the UPDATE Statement

In this exercise you will modify the last name of member number 507 in the **member** table of the **library** database. C:\Moc\2071A\Labfiles\L07\Answers contains completed scripts for this exercise.

► To retrieve a row of data that you intend on updating from the member table

In this procedure, you will retrieve a row of data that you want to update from the **member** table. Answer_SelUpdate1.sql is a completed script for this procedure.

1. Write a query that retrieves the last name of member number 507 in the **member** table.

```
USE library
SELECT *
 FROM member
 WHERE member_no = 507
GO
```

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

► To update a specific row of data from the member table

In this procedure, you will update a specific row of data from the **member** table. Answer_Update1.sql is a completed script for this procedure.

1. Write a query that changes the last name of member number 507 in the **member** table to a different one of your choice.

```
USE library
UPDATE member
 SET lastname = 'BENSON'
 WHERE member_no = 507
GO
```

2. Execute the query and confirm that it updates one row in the **member** table.

Exercise 6
Modifying Tables Based on Data in Other Tables

In this exercise, you will write queries that insert values from one or more tables in the database into an existing table, and you will delete or update rows in a table based on criteria in other tables.
C:\Moc\2071A\Labfiles\L07\Answers contains completed scripts for this exercise.

► To add a new juvenile member to the database

In this procedure, you will add a new juvenile member to the **library** database.

1. Review and execute the C:\Moc\2071A\Labfiles\L07\AddJuvenile.sql script file to add a new juvenile member to the **library** database.

   Because the process of adding a new juvenile member requires two INSERT statements, this action is treated as a transaction. The SET IDENTITY_INSERT statement is used to supply a specific value for the **member.member_no** column rather than using the value that the IDENTITY property supplied.

2. Execute the query and verify that a single row was inserted into each of two tables.

► To determine which records should be moved from the juvenile table to the adult table

In this procedure, you will retrieve data from the **adult** and **item** tables for all juvenile members over age 18. Answer_SelNewAdult.sql is a completed script for this procedure.

1. Write a SELECT statement that returns the **member_no** column from the **juvenile** table and the **street**, **city**, **state**, **zip**, and **phone_no** columns from the **adult** table. Also include in the query today's date plus one year by using the following expression:

   ```
   DATEADD( YY, 1, GETDATE() )
   ```

   This last column will be used later to provide a value for the **adult.expr_date** column. This SELECT statement joins the **juvenile** table with the **member** table, such that **juvenile.adult_member_no** = **adult.member_no**.

   Include a WHERE clause to limit the rows that are added to those members in the **juvenile** table who are over age 18 by using the DATEADD function in an expression. Search Books Online for "DATEADD" if you need further assistance.

   ```
   USE library
    SELECT ju.member_no, ad.street, ad.city, ad.state
          ,ad.zip, ad.phone_no, DATEADD( YY, 1, GETDATE() )
    FROM juvenile AS ju
    INNER JOIN adult AS ad
     ON ju.adult_member_no = ad.member_no
    WHERE ( DATEADD(YY, 18, ju.birth_date) < GETDATE() )
   GO
   ```

2. Execute the query to verify that it returns the desired results. Note the **member_no** values that are returned.

► To insert new rows into the adult table from the juvenile table

1. Write an INSERT statement that incorporates the SELECT statement that you created in step 1 of the previous procedure in order to add rows to the **adult** table. Answer_InsNewAdult.sql is a completed script for this procedure.

```
USE library
INSERT adult( member_no, street, city, state
             ,zip, phone_no, expr_date )
 SELECT ju.member_no, ad.street, ad.city, ad.state
       ,ad.zip, ad.phone_no, DATEADD( YY, 1, GETDATE() )
  FROM juvenile AS ju
  INNER JOIN adult AS ad
   ON ju.adult_member_no = ad.member_no
  WHERE ( DATEADD(YY, 18, ju.birth_date) < GETDATE() )
GO
```

2. Execute the query and verify that one row is inserted.

► To verify that a certain juvenile record was added to the adult table

1. Write a query to verify that juvenile member number 16101 was added to the **adult** table. Answer_ChkNewAdult.sql is a completed script for this procedure.

```
USE library
SELECT *
 FROM adult
 WHERE member_no = 16101
GO
```

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

Result

Your result will look similar to the following partial result set.

```
member_no   name              expr_date
----------- ----------------- ------------------------
16101       Walters, B. L.    Feb 7 1998 2:58PM

(1 row(s) affected)
```


► To determine which rows in the juvenile table should be removed

In this procedure, you will create a query that deletes rows from the **juvenile** table that have matching rows in the **adult** table. After juvenile members are converted to adult members, those members must be deleted from the **juvenile** table. Answer_SelOldJuvenile.sql is a completed script for this procedure.

1. Write a SELECT statement that joins the **adult** and **juvenile** tables so that **juvenile.member_no** = **adult.member_no**.

```
USE library
SELECT *
 FROM juvenile
 INNER JOIN adult
  ON juvenile.member_no = adult.member_no
GO
```

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

► **To delete rows in the juvenile table that have matches in the adult table**

1. Write a DELETE statement that uses the SELECT statement that you created in step 1 of the previous procedure to delete these rows from the **juvenile** table. Answer_DelOldJuvenile.sql is a completed script for this procedure.

```
USE library
DELETE juvenile
 FROM juvenile
 INNER JOIN adult
  ON juvenile.member_no = adult.member_no
GO
```

2. Execute the query and verify that one row is deleted.

► **To verify that a certain records were removed from the juvenile table**

1. Write a SELECT statement to verify that member number 16101 was removed from the **juvenile** table. Answer_ChkOldJuvenile.sql is an example of this query.

```
USE library
SELECT *
 FROM juvenile
 WHERE member_no = 16101
GO
```

2. Execute the query and verify that no records are returned.

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.

- Using Transactions
- Inserting Data
- Deleting Data
- Updating Data
- Performance Considerations

Use this scenario to answer these questions and review module topics.

Ask students whether they need clarification on any topic.

You are the database administrator for a health care plan. The **physicians** table was created by using the following statement:

```
CREATE TABLE dbo.physicians (
    physician_no int IDENTITY (100, 2) NOT NULL
   ,f_name varchar (25) NOT NULL
   ,l_name varchar (25) NOT NULL
   ,street varchar (50) NULL
   ,city varchar (255) NULL
   ,state varchar (255) NULL
   ,postal_code varchar (7) NULL
   ,co_pay money NOT NULL CONSTRAINT phys_co_pay DEFAULT (10)
   )
GO
```


1. What is the minimum number of column values that you must supply to add a new row to the table?

   You must supply data for at least two columns. At a minimum, the INSERT statement will contain values for f_name and l_name. All other columns allow null values or have defaults generated for them.

2. The participating physicians have increased their costs of services. How can you increase the **co_pay** value for all doctors by 12 percent?

   Use an UPDATE statement of the following type:

   ```
   UPDATE physicians SET co_pay = (co_pay + co_pay * .12)
   ```

   _____

   _____

3. How can you remove all rows from the **physicians** table?

   Use a DELETE statement or a TRUNCATE TABLE statement.

   _____

   _____