MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 5: Joining Multiple Tables

Contents

Microsoft

Project Lead: Cheryl Hoople
Instructional Designer: Cheryl Hoople
Technical Lead: LeRoy Tuttle
Program Manager: LeRoy Tuttle
Graphic Artist: Kimberly Jackson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Editorial Contributor: Elizabeth Reese
Copy Editor: Bill Jones (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Tools Specialist: Julie Challenger
Production Support: Lori Walker (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: Testing Testing 123
Classroom Automation: Lorrin Smith-Bates
Creative Director, Media/Sim Services: David Mahlmann
Web Development Lead: Lisa Pease
CD Build Specialist: Julie Challenger
Online Support: David Myka (S&T Consulting)
Localization Manager: Rick Terek
Operations Coordinator: John Williams
Manufacturing Support: Laura King; Kathy Hershey
Lead Product Manager, Release Management: Bo Galford
Lead Product Manager: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
60 Minutes

Lab:
45 Minutes

This module provides students with an overview of querying multiple tables by using different types of joins, combining result sets by using the UNION operator, and creating tables by using the SELECT INTO statement.

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

- Use aliases for table names.
- Combine data from two or more tables by using joins.
- Combine multiple result sets into one result set by using the UNION operator.

Materials and Preparation

Required Materials

To teach this course, you need the following materials:

- Microsoft® PowerPoint® file 2071A_05.ppt.
- The C:\Moc\2071A\Demo\Ex_05.sql example, which contains all of the example scripts from the module, unless otherwise noted in the module.

Preparation Tasks

To prepare for this module, you should:

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

Module Strategy

Use the following strategy to present this module:

- Using Aliases for Table Names

  Point out that users can assign aliases for table names within the scope of a Transact-SQL statement. Note that using aliases for table names helps script readability and facilitates complex join logic.

- Combining Data from Multiple Tables

  Introduce the join operation and discuss in detail inner, outer, and cross joins. The examples only focus on joining two tables by using a simplified database, **joindb**, to teach these concepts.

  Explain how to join multiple tables and a table to itself. Demonstrate multiple and self-joins against the **northwind** database by using the provided script.

- Combining Multiple Result Sets

  Describe combining multiple result sets into one result set by using the UNION operator.

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*.

Module Setup

The C:\Moc\2071A\Batches\2071A_JoinDB.sql script, which adds the **joindb** database, is normally executed as part of the Classroom Setup. When you customize the course, you must ensure that this script is executed so that the examples in the module function correctly.

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 introduce the topics that this module covers.

Lead-in
In this module, you will learn about joining multiple tables.

- **Using Aliases for Table Names**
- **Combining Data from Multiple Tables**
- **Combining Multiple Result Sets**

This module provides students with an overview of querying multiple tables by using different types of joins, combining result sets by using the UNION operator, and creating tables by using the SELECT INTO statement.

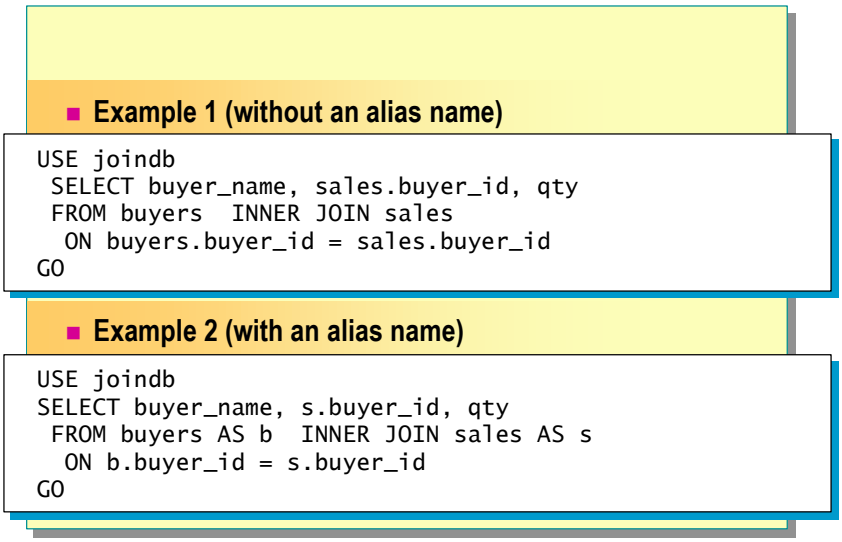
After completing this module, you will be able to:

- Use aliases for table names.
- Combine data from two or more tables by using joins.
- Combine multiple result sets into one result set by using the UNION operator.

Using Aliases for Table Names

Topic Objective
To describe how to use aliases for table names.

Lead-in
Using aliases for table names improves script readability, facilitates writing complex joins, and simplifies the maintenance of Transact-SQL.



Using aliases for table names improves script readability, facilitates writing complex joins, and simplifies the maintenance of Transact-SQL.

You can replace a long and complex fully qualified table name with a simple, abbreviated alias name when writing scripts. You use an alias name in place of the full table name.

Partial Syntax

SELECT * FROM server.database.schema.table AS table_alias

Example 1

This example displays the names of buyers, buyer ID, and the quantity sold from the **buyers** and **sales** tables. This query does not use alias names for the tables in the JOIN syntax.

```
USE joindb
SELECT buyer_name, sales.buyer_id, qty
 FROM buyers
 INNER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
GO
```

Example 2

This example displays the names of buyers, buyer ID, and the quantity sold from the **buyers** and **sales** tables. This query uses alias names for the tables in the JOIN syntax.

```
USE joindb
SELECT buyer_name, s.buyer_id, qty
 FROM buyers AS b
 INNER JOIN sales AS s
  ON b.buyer_id = s.buyer_id
GO
```

Note Sometimes complex JOIN syntax and subqueries must use aliases for table names. For example, aliases must be used when joining a table to itself.

◆ Combining Data from Multiple Tables

Topic Objective
To explain the different ways that you can combine data from two or more tables or result sets.

Lead-in
It is possible to combine data from two or more tables, even if the tables reside in different databases.

- Introduction to Joins
- Using Inner Joins
- Using Outer Joins
- Using Cross Joins
- Joining More Than Two Tables
- Joining a Table to Itself

A join is an operation that allows you to query two or more tables to produce a result set that incorporates rows and columns from each table. You join tables on columns that are common to both tables.

When you join tables, Microsoft® SQL Server™ 2000 compares the values of the specified columns row by row and then uses the comparison results to combine the qualifying values into new rows.

There are three types of joins: inner joins, outer joins, and cross joins. Additionally, you can join more than two tables by using a series of joins within a SELECT statement, or you can join a table to itself by using a self-join.

Introduction to Joins

Topic Objective
To explain how joins are implemented.

Lead-in
You join tables to produce a single result set that incorporates elements from two or more tables.

- **Selects Specific Columns from Multiple Tables**
  - JOIN keyword specifies that tables are joined and how to join them
  - ON keyword specifies join condition
- **Queries Two or More Tables to Produce a Result Set**
  - Use primary and foreign keys as join conditions
  - Use columns common to specified tables to join tables

You join tables to produce a single result set that incorporates rows and columns from two or more tables.

Partial Syntax

```
SELECT column_name [, column_name ...]
  FROM {<table_source>} [,...n]
    <join_type> ::=
    [ INNER | { { LEFT | RIGHT | FULL } [OUTER] } ]
    [ <join_hint> ]
    JOIN
    <joined_table> ::=
    <table_source> <join_type> <table_source> ON <search_condition>
    | <table_source> CROSS JOIN <table_source>
    | <joined_table>
```


Delivery Tip
Reference SQL Server Books Online to show the full SELECT statement and to highlight the joins.

Selects Specific Columns from Multiple Tables

A join allows you to select columns from multiple tables by expanding on the FROM clause of the SELECT statement. Two additional keywords are included in the FROM clause—JOIN and ON:

- The JOIN keyword specifies which tables are to be joined and how to join them.
- The ON keyword specifies which columns the tables have in common.

Queries Two or More Tables to Produce a Result Set

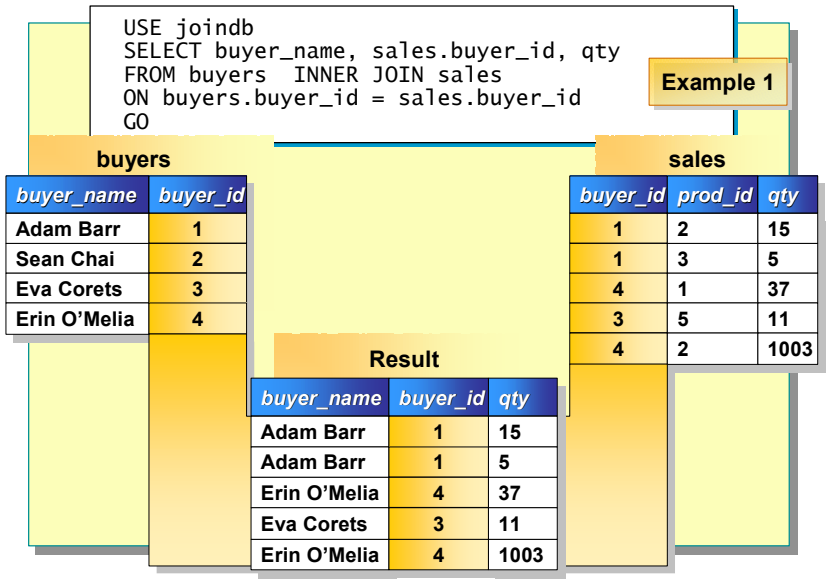
A join allows you to query two or more tables to produce a single result set. When you implement joins, consider the following facts and guidelines:

- Specify the join condition based on the primary and foreign keys.
- If a table has a composite primary key, you must reference the entire key in the ON clause when you join tables.
- Use columns common to the specified tables to join the tables. These columns should have the same or similar data types.
- Reference a table name if the column names of the joined tables are the same. Qualify each column name by using the *table_name.column_name* format.
- Limit the number of tables in a join because the more tables that you join, the longer SQL Server takes to process your query.
- You can include a series of joins within a SELECT statement.

Using Inner Joins

Topic Objective
To define and demonstrate inner joins.

Lead-in
Use inner joins to combine tables in which values in compared columns are equal.



Inner joins combine tables by comparing values in columns that are common to both tables. SQL Server returns only rows that match the join conditions.

Note The examples in this module are from the **joindb** database—a database created specifically for teaching the different types of joins. The **joindb** database is included on the Student Materials compact disc.

Delivery Tip
The examples on the slides in this module are from the **joindb** database—a database created specifically for teaching the different types of joins. The **joindb** database is included on the Student Materials compact disc.

Point out that SQL Server does not guarantee an order in the result set unless it is specified with an ORDER BY clause.

Why to Use Inner Joins

Use inner joins to obtain information from two separate tables and combine that information in one result set. When you use inner joins, consider the following facts and guidelines:

- Inner joins are the SQL Server default. You can abbreviate the INNER JOIN clause to JOIN.
- Specify the columns that you want to display in your result set by including the qualified column names in the select list.
- Include a WHERE clause to restrict the rows that are returned in the result set.
- Do not use a null value as a join condition because null values do not evaluate equally with one another.

Note SQL Server does not guarantee an order in the result set unless one is specified with an ORDER BY clause.

Example 1

This example returns the **buyer_name**, **buyer_id**, and **qty** values for the buyers who purchased products. Buyers who did not purchase any products are not included in the result set. Buyers who bought more than one product are listed for each purchase.

Delivery Tip
Point out that the **buyer_id** column from either table can be referenced in the select list.

The **buyer_id** column from either table can be specified in the select list.

```
USE joindb
SELECT buyer_name, sales.buyer_id, qty
 FROM buyers
 INNER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
GO
```

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003

(5 row(s) affected)

Example 2

This example returns the names of products and the companies that supply the products. Products without listed suppliers and suppliers without current products are not included in the result set.

```
USE northwind
SELECT productname, companyname
 FROM products
 INNER JOIN suppliers
  ON products.supplierid = suppliers.supplierid
GO
```

Result

productname	companyname
Chai	Exotic Liquids
Chang	Exotic Liquids
Aniseed Syrup	Exotic Liquids
Chef Anton's Cajun Seasoning	New Orleans Cajun Delights
.	
.	
.	

(77 row(s) affected)

Example 3

This example returns the names of customers who placed orders after 1/1/98. Notice that a WHERE clause is used to restrict the rows that are returned in the result set.

```
USE northwind
SELECT DISTINCT companyname, orderdate
FROM orders INNER JOIN customers
ON orders.customerid = customers.customerid
WHERE orderdate > '1/1/98'
GO
```

Result

```
companyname                              orderdate
Alfreds Futterkiste                      1998-01-15 00:00:00.000
Alfreds Futterkiste                      1998-03-16 00:00:00.000
Alfreds Futterkiste                      1998-04-09 00:00:00.000
Ana Trujillo Emparedados y helados       1998-03-04 00:00:00.000
.
.
.
(264 row(s) affected)
```

Example 4

This example returns the title number of all books currently checked out and the member number of the borrower from the **copy** and **loan** tables in the **library** database. Both the **copy** and **loan** tables have a composite primary key consisting of the **isbn** and **copy_no** columns. When joining these tables, you must specify both columns as join conditions because they uniquely identify a particular copy of a book.

Delivery Tip
This example uses the **library** database, because the **northwind** database does not have two tables with composite primary keys that relate to one another.

```
USE library
SELECT copy.title_no, loan.member_no
 FROM copy
 INNER JOIN loan
  ON copy.isbn = loan.isbn
  AND copy.copy_no = loan.copy_no
 WHERE copy.on_loan = 'Y'
GO
```

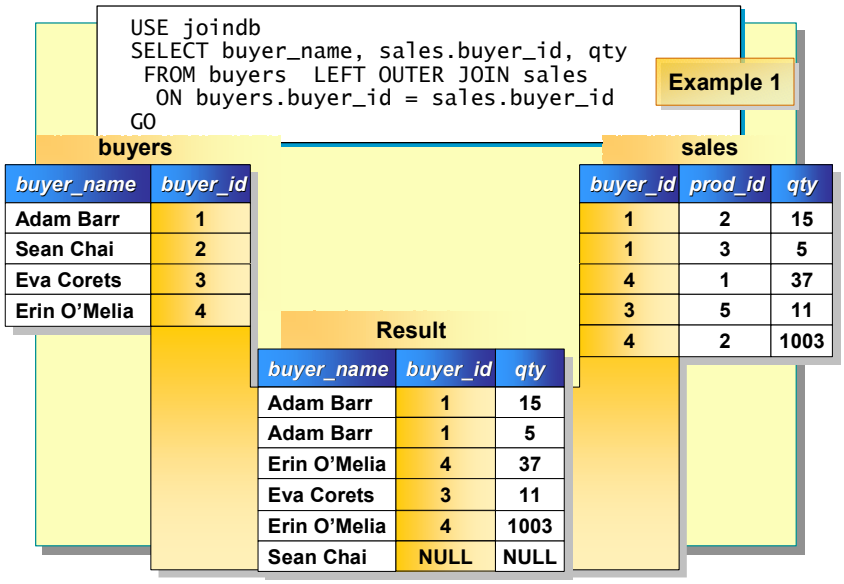
Result

```
title_no             member_no
1                    325
1                    351
2                    390
2                    416
.
.
.
(2000 row(s) affected)
```

Using Outer Joins

Topic Objective
To define outer joins and describe the three types.

Lead-in
By using left, right, or full outer joins, you can include rows that do not match your join condition in a result set.



Delivery Tip
Point out the null values on the slide for Sean Chai. Rows that do not match the join condition display NULL in the result set.

Left or right outer joins combine rows from two tables that match the join condition, plus any unmatched rows of either the left or right table as specified in the JOIN clause. Rows that do not match the join condition display NULL in the result set. You also can use full outer joins to display all rows in the joined tables, regardless of whether the tables have any matching values.

Why to Use Left or Right Outer Joins

Delivery Tip
Ask: What would you change in the slide example query to yield the same result with a RIGHT OUTER JOIN clause?

Answer: Reverse the order of the tables in the FROM clause and use the RIGHT OUTER JOIN clause.

Use left or right outer joins when you require a complete list of data that is stored in one of the joined tables in addition to the information that matches the join condition. When you use left or right outer joins, consider the following facts and guidelines:

- SQL Server returns only unique rows when you use left or right outer joins.
- Use a left outer join to display all rows from the first-named table (the table on the left of the expression). If you reverse the order in which the tables are listed in the FROM clause, the statement yields the same result as a right outer join.
- Use a right outer join to display all rows from the second-named table (the table on the right of the expression). If you reverse the order in which the tables are listed in the FROM clause, the statement yields the same result as a left outer join.

Delivery Tip
Always use the ANSI SQL-92 join syntax, with ANSI_NULLS set to ON.

- You can abbreviate the LEFT OUTER JOIN or RIGHT OUTER JOIN clause as LEFT JOIN or RIGHT JOIN.
- You can use outer joins between two tables only.

Example 1

This example returns the **buyer_name**, **buyer_id**, and **qty** values for all buyers and their purchases. Notice that the buyers who did not purchase any products are listed in the result set, but null values appear in the **buyer_id** and **qty** columns.

```
USE joindb
SELECT buyer_name, sales.buyer_id, qty
 FROM buyers
 LEFT OUTER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
GO
```

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003
Sean Chai	NULL	NULL

```
(6 row(s) affected)
```

Note The sort order of the result set can be different because the ORDER BY clause is not used in the example.

Example 2

This example displays all customers with order dates. By using a left outer join, you retrieve one row for each customer and additional rows if the customer has placed multiple orders. NULL in the **orderdate** column is returned in the result set for customers who have not placed an order. Notice the NULL entries for customers FISSA and Paris Spécialités.

```
USE northwind
SELECT companyname, customers.customerid, orderdate
 FROM customers
 LEFT OUTER JOIN orders
  ON customers.customerid = orders.customerid
GO
```

Result

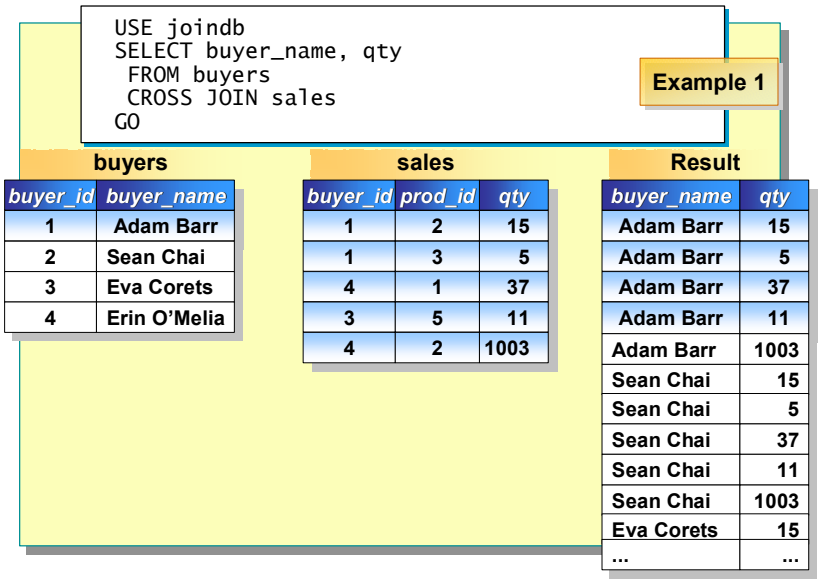
companyname	customerid	orderdate
Vins et alcools Chevalier	VINIT	1996-07-04 00:00.0
Toms Spezialitaten	TOMSP	1996-07-05 00:00.0
Hanari Carnes	HANAR	1996-07-08 00:00.0
Victuailles en stock	VICTE	1996-07-08 00:00.0
.		
.		
.		
FISSA Fabrica Inter. Salichichas S.A.	FISSA	NULL
Paris specialities	PARIS	NULL

```
(832 row(s) affected)
```

Using Cross Joins

Topic Objective
To show how cross joins work and describe the result set.

Lead-in
Use cross joins to display all possible row combinations of the selected columns in the joined tables.



Delivery Tip
Point out that the ON keyword and associated column list is not used in the SELECT statement because cross joins return all possible row combinations from each specified table.

A common column is not required to use cross joins.

Cross joins display every combination of all rows in the joined tables. A common column is not required to use cross joins.

Why to Use Cross Joins

While cross joins are rarely used on a normalized database, you can use them to generate test data for a database or lists of all possible combinations for checklists or business templates.

When you use cross joins, SQL Server produces a Cartesian product in which the number of rows in the result set is equal to the number of rows in the first table, multiplied by the number of rows in the second table. For example, if there are 8 rows in one table and 9 rows in the other table, SQL Server returns a total of 72 rows.

Example 1

This example lists all possible combinations of the values in the **buyers.buyer_name** and **sales.qty** columns.

```
USE joindb
SELECT buyer_name, qty
 FROM buyers
 CROSS JOIN sales
GO
```

Result

```
buyer_name              qty
Adam Barr               15
Adam Barr               5
Adam Barr               37
Adam Barr               11
Adam Barr               1003
Sean Chai               15
Sean Chai               5
.
.
.
(20 row(s) affected)
```

Example 2

This example displays a cross join between the **shippers** and **suppliers** tables that is useful for listing all of the possible ways that suppliers can ship their products.

Delivery Tip
Execute this query and explain that this example is useful for listing all of the possible ways that suppliers can ship their products.

The use of a cross join displays all possible row combinations between these two tables. The **shippers** table has 3 rows, and the **suppliers** table has 29 rows. The result set contains 87 rows.

```
USE northwind
SELECT suppliers.companyname, shippers.companyname
 FROM suppliers
 CROSS JOIN shippers
GO
```

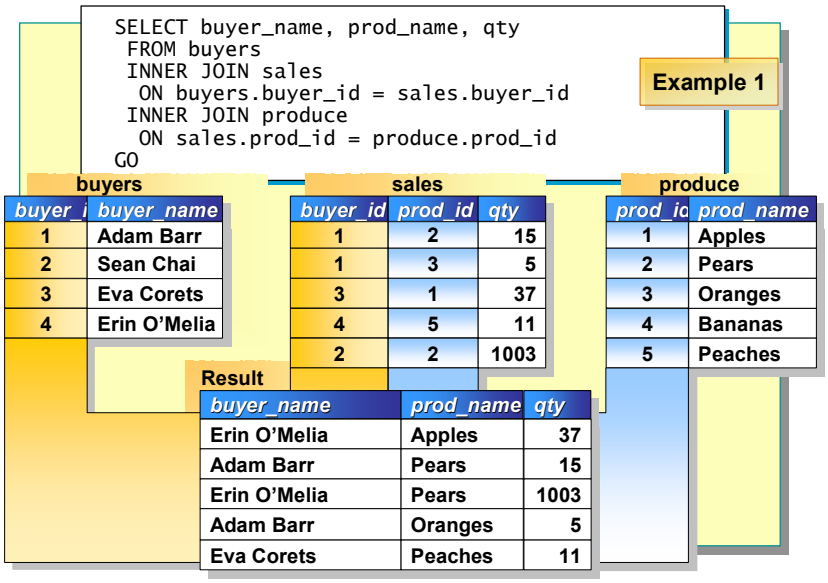
Result

```
companyname                          companyname
Aux joyeux ecclésiastiques           Speedy Express
Bigfoot Breweries                    Speedy Express
Cooperativa de Quesos 'Las Cabras'   Speedy Express
Escargots Nouveaux                   Speedy Express
.
.
.
Aux joyeux ecclésiastiques           United Package
Bigfoot Breweries                    United Package
Cooperativa de Quesos 'Las Cabras'   United Package
Escargots Nouveaux                   United Package
.
.
.
Aux joyeux ecclésiastiques           Federal Shipping
Bigfoot Breweries                    Federal Shipping
Cooperativa de Quesos 'Las Cabras'   Federal Shipping
Escargots Nouveaux                   Federal Shipping
.
.
.
(87 row(s) affected)
```

Joining More Than Two Tables

Topic Objective
To explain how to join more than two tables.

Lead-in
Until now, we've looked at joining only two tables. It is possible, however, to join more than two tables.



Delivery Tip
The first join is between the **buyers** and **sales** tables. The second join is between the **sales** and **produce** tables.

Emphasize that any table referenced in a join operation can be joined to another table by a common column.

It is possible to join any number of tables. Any table that is referenced in a join operation can be joined to another table by a common column.

Why to Join More Than Two Tables

Use multiple joins to obtain related information from multiple tables. When you join more than two tables, consider the following facts and guidelines:

- You must have one or more tables with foreign key relationships to each of the tables that you want to join.
- You must have a JOIN clause for each column that is part of a composite key.
- Include a WHERE clause to limit the number of rows that are returned.

Example 1

This example returns the **buyer_name**, **prod_name**, and **qty** columns from the **buyers**, **sales** and **produce** tables. The **buyer_id** column is common to both the **buyers** and **sales** tables and is used to join these two tables. The **prod_id** column is common to both the **sales** and **produce** tables and is used to join the **produce** table to the result of the join between **buyers** and **sales**.

```
USE joindb
SELECT buyer_name, prod_name, qty
 FROM buyers
 INNER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
 INNER JOIN produce
  ON sales.prod_id = produce.prod_id
GO
```

Result

buyer_name	prod_name	qty
Erin O'Melia	Apples	37
Adam Barr	Pears	15
Erin O'Melia	Pears	1003
Adam Barr	Oranges	5
Eva Corets	Peaches	11

```
(5 row(s) affected)
```

Example 2

This example displays information from the **orders** and **products** tables by using the **order details** table as a link. For example, if you want a list of products that are ordered each day, you need information from both the **orders** and **products** tables. An order can consist of many products, and a product can have many orders.

To retrieve information from both the **orders** and **products** tables, you can use an inner join through the **order details** table. Even though you are not retrieving any columns from the **order details** table, you must include this table as part of the inner join in order to relate the **orders** table to the **products** table. In this example, the **orderid** column is common to both the **orders** and **order details** tables, and the **productid** column is common to both the **order details** and **products** tables.

```
USE northwind
SELECT orderdate, productname
 FROM orders AS O
 INNER JOIN [order details] AS OD
  ON O.orderid = OD.orderid
 INNER JOIN products AS P
  ON OD.productid = P.productid
 WHERE orderdate = '7/8/96'
```

Result

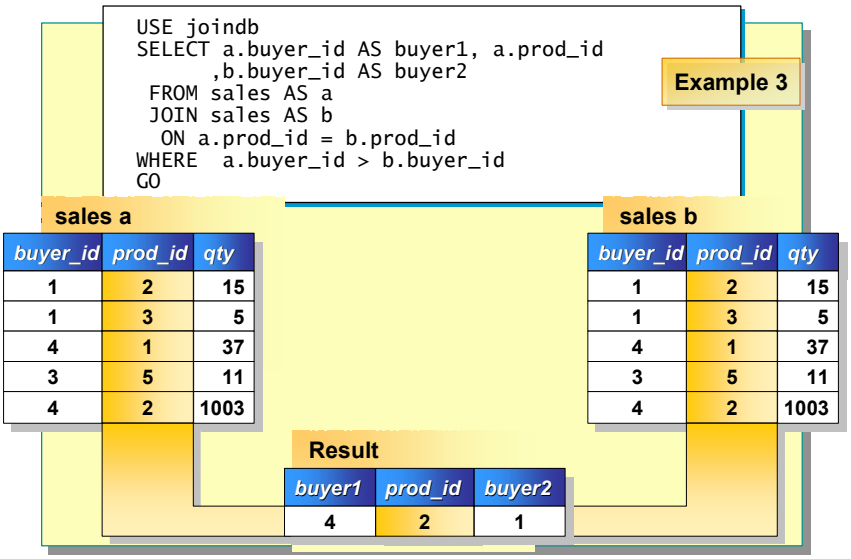
orderdate	productname
1996-07-08	Jack's New England Clam Chowder
1996-07-08	Manjimup Dried Apples
1996-07-08	Louisiana Fiery Hot Pepper Sauce
1996-07-08	Gustaf's Knakebrod
1996-07-08	Ravioli Angelo
1996-07-08	Louisiana Fiery Hot Pepper Sauce

```
(6 row(s) affected)
```

Joining a Table to Itself

Topic Objective
To explain a self-join.

Lead-in
Although joins are used most commonly to combine multiple tables, you can use a self-join to join a table to itself as well.



Delivery Tip
The slide example shows the desired result when a table is joined to itself. Use the series of examples in the student workbook to teach joining a table to itself.

If you want to find rows that have values in common with other rows in the same table, you can use a self-join to join a table to another instance of itself.

Why to Use Self-Joins

While self-joins rarely are used on a normalized database, you can use them to reduce the number of queries that you execute when you compare values of different columns of the same table. When you use self-joins, consider the following guidelines:

- You must specify table aliases to reference two copies of the table. Remember that table aliases are different from column aliases. Table aliases are designated as the table name followed by the alias.
- When you create self-joins, each row matches itself and pairs are repeated, resulting in duplicate rows. Use a WHERE clause to eliminate these duplicate rows.

Example 1

This example displays a list of all buyers who purchased the same products. Notice that the first and third rows of the result set are rows where **buyer1** matches itself. The fourth and seventh rows are rows where **buyer4** matches itself. The second and sixth rows are rows that mirror one another.

Delivery Tip
Point out the duplicates where the rows match themselves (Rows 1, 3, 4, and 7). Use a WHERE clause with the not equal to (<>) operator to eliminate this type of duplicate.

```
USE joindb
SELECT a.buyer_id AS buyer1, a.prod_id, b.buyer_id AS buyer2
 FROM sales AS a
 INNER JOIN sales AS b
  ON a.prod_id = b.prod_id
GO
```

Result

buyer1	prod_id	buyer2
1	2	1
4	2	1
1	3	1
4	1	4
3	5	3
1	2	4
4	2	4

(7 row(s) affected)

Example 2

This example displays a list of buyers who all purchased the same products, but it eliminates duplicate rows, such as **buyer1** matching itself and **buyer4** matching itself.

Delivery Tip
Point out that the example will not eliminate duplicate rows that are mirror images of one another.

Compare the result sets of Examples 1 and 2. Notice that by using a WHERE clause with the not equal to (<>) operator, the duplicate rows are eliminated. However, duplicate rows that are mirror images of one another are still returned in the result set.

```
USE joindb
SELECT a.buyer_id AS buyer1, a.prod_id, b.buyer_id AS buyer2
 FROM sales AS a
 INNER JOIN sales AS b
  ON a.prod_id = b.prod_id
 WHERE a.buyer_id <> b.buyer_id
GO
```

Result

buyer1	prod_id	buyer2
4	2	1
1	2	4

(2 row(s) affected)

Example 3

This example displays a list of buyers who all purchased the same products.

Delivery Tip
Point out that using the WHERE clause with the greater than (>) or less than (<) operator eliminates the duplicates in Example 2.

Notice that when the WHERE clause includes the greater than (>) operator, all duplicate rows are eliminated.

```
USE joindb
SELECT a.buyer_id AS buyer1, a.prod_id, b.buyer_id AS buyer2
 FROM sales AS a
 INNER JOIN sales AS b
  ON a.prod_id = b.prod_id
 WHERE a.buyer_id > b. buyer_id
GO
```

Result

buyer1	prod_id	buyer2
4	2	1

(1 row(s) affected)

Example 4

This example displays pairs of employees who have the same job title. When the WHERE clause *includes* the less than (<) operator, rows that match themselves and duplicate rows are eliminated.

```
USE northwind
SELECT a.employeeid, LEFT(a.lastname,10) AS name
      ,LEFT(a.title,10) AS title
      ,b.employeeid, LEFT(b.lastname,10) AS name
      ,LEFT(b.title,10) AS title
 FROM employees AS a
 INNER JOIN employees AS b
  ON a.title = b.title
 WHERE a.employeeid < b.employeeid
GO
```

Result

employeeid	Name	title	employeeid	name	title
1	Davolio	Sales Repr	3	Leverling	Sales Repr
1	Davolio	Sales Repr	4	Peacock	Sales Repr
1	Davolio	Sales Repr	6	Suyama	Sales Repr
1	Davolio	Sales Repr	7	King	Sales Repr
1	Davolio	Sales Repr	9	Dodsworth	Sales Repr
3	Leverling	Sales Repr	4	Peacock	Sales Repr
3	Leverling	Sales Repr	6	Suyama	Sales Repr
3	Leverling	Sales Repr	7	King	Sales Repr
3	Leverling	Sales Repr	9	Dodsworth	Sales Repr
4	Peacock	Sales Repr	6	Suyama	Sales Repr
4	Peacock	Sales Repr	7	King	Sales Repr
4	Peacock	Sales Repr	9	Dodsworth	Sales Repr
6	Suyama	Sales Repr	7	King	Sales Repr
6	Suyama	Sales Repr	9	Dodsworth	Sales Repr
7	King	Sales Repr	9	Dodsworth	Sales Repr

15 row(s) affected)

Combining Multiple Result Sets

Topic Objective
To explain the purpose and function of the UNION operator.

Lead-in
You can combine the results of two or more SELECT statements into a single result set by using the UNION operator.

- **Use the UNION Operator to Create a Single Result Set from Multiple Queries**
- **Each Query Must Have:**
  - Similar data types
  - Same number of columns
  - Same column order in select list

```
USE northwind
SELECT  (firstname + ' ' + lastname) AS name
       ,city, postalcode
 FROM employees
UNION
SELECT companyname, city, postalcode
 FROM customers
GO
```

The UNION operator combines the result of two or more SELECT statements into a single result set.

Use the UNION operator when the data that you want to retrieve resides in different locations and cannot be accessed with a single query. When you use the UNION operator, consider the following facts and guidelines:

- SQL Server requires that the referenced tables have similar data types, the same number of columns, and the same column order in the select list of each query.
- SQL Server removes duplicate rows in the result set. However, if you use the ALL option, all rows (including duplicates) are included in the result set.
- You must specify the column names in the first SELECT statement. Therefore, if you want to define new column headings for the result set, you must create the column aliases in the first SELECT statement.
- If you want the entire result set to be returned in a specific order, you must specify a sort order by including an ORDER BY clause within the UNION statement. Otherwise, the result set may not be returned in the order that you want.
- You may experience better performance if you break a complex query into multiple SELECT statements and then use the UNION operator to combine them.

Key Point
When you use the UNION operator, the referenced tables must have similar data types, the same number of columns, and the same column order in the select list of each query.

Syntax

select_statement UNION [ALL] *select_statement*

Example

This example combines two result sets. The first result set returns the name, city, and postal code of each customer from the **customers** table. The second result set returns the name, city, and postal code of each employee from the **employees** table. When you use the UNION operator to combine these result sets, notice that the column alias from the first select list is returned.

Delivery Tip
Demonstrate this example by using SQL Query Analyzer.

```
USE northwind
SELECT (firstname + ' ' + lastname) AS name, city, postalcode
 FROM employees
UNION
SELECT companyname, city, postalcode
 FROM customers
GO
```

Result

name	city	postalcode
Alfreds Futterkiste	Berlin	12209
Ana Trujillo Emparedados y helados	México D.F.	05021
Antonio Moreno Taquería	México D.F.	05023
Around the Horn	London	WA1 1DP
B's Beverages	London	EC2 5NT
.		
.		
.		
Andrew Fuller	Tacoma	98401
Robert King	London	RG 19SP
Janet Leverling	Kirkland	98033
Anne Dodsworth	London	WG2 7LT

(100 row(s) affected)

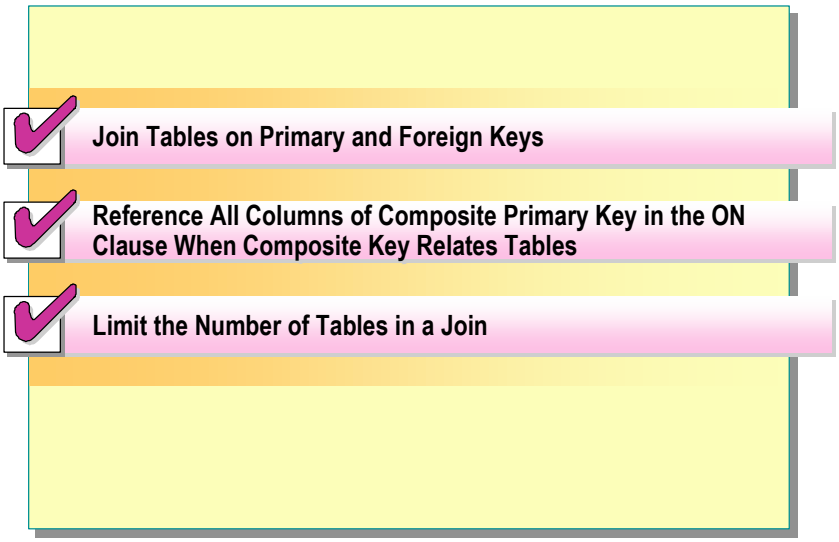
Delivery Tip
Ask: In the result set, why are the customers listed before the employees, as the syntax would imply?

Answer: SQL Server does not guarantee a particular order unless one is specified with an ORDER BY clause.

Recommended Practices

Topic Objective
To list the recommended practices for data retrieval and modification.

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



The following recommended practices should help you perform queries:

- Join tables on primary and foreign keys.
- Reference all columns of a composite primary key in the ON clause when a composite key relates tables.
- Limit the number of tables in a join because the more tables that you join, the longer SQL Server takes to process your query.

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

Topic	Search on
Working with joins	"join fundamentals"
	"using multiple tables"

Lab A: Querying Multiple Tables

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will you will perform different types of joins to combine data from multiple tables.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Join tables by using different join types.
- Combine result sets by using the UNION operator.

Prerequisites

Before working on this lab, you must have:

- Answer files for this lab, which are located in C:\Moc\2071A\Labfiles\L05\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: 45 minutes

Exercise 1
Joining Tables

In this exercise, you will write and execute queries that join tables in the **library** database. C:\Moc\2071A\Labfiles\L05\Answers contains completed scripts for this exercise.

► To create a mailing list by using a join

In this procedure, you will create a mailing list of library members that includes the members' full names and complete address information. Answer_Mailing.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 Microsoft Windows® Authentication.

   You have permission to log in to and administer SQL Server 2000 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. Write a query on the **member** and **adult** tables that returns the **firstname**, **middleinitial**, **lastname**, **street**, **city**, **state**, and **zip** values. Concatenate the **firstname**, **middleinitial** and **lastname** columns into one string and alias the column as **name**.

```
USE library
SELECT firstname +' '+ middleinitial +' '+ lastname AS name
      ,street, city, state, zip
 FROM member
 INNER JOIN adult
  ON member.member_no = adult.member_no
GO
```

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

Result

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

```
name                 street           city          state   zip
Amy A Anderson       Bowery Estates   Montgomery    AL      36100
Brian A Anderson     Dogwood Drive    Sacramento    CA      94203
Daniel A Anderson    Fir Street       Washington    DC      20510-0001
Eva A Anderson       The Highlands    Atlanta       GA      30026
Gary A Anderson      James Road       Springfield   IL      62700
.
.
.
5000 row(s) affected
```

► To join several tables and order the results

In this procedure, you will write and execute a query on the **title**, **item**, and **copy** tables that returns the **isbn**, **copy_no**, **on_loan**, **title**, **translation**, and **cover**, and values for rows in the **copy** table with an ISBN of 1 (one), 500 (five hundred), or 1000 (one thousand). Order the results by the **isbn** column.

Answer_Serveral.sql is a completed script for this procedure.

1. Write the select list of the query. Qualify the name of each column with a table alias of at least two characters (for example, **ti.title_no** for **title.title_no**).
2. Write a FROM clause that creates a inner join between the **title** and **copy** tables on the **title_no** columns. Set up the table aliases in the FROM clause that you used in the select list.
3. Add a second INNER JOIN clause to create a join between the **item** and **copy** tables on the **isbn** columns.
4. Compose a WHERE clause to restrict the rows that are retrieved from the **copy** table to those with an ISBN of 1 (one), 500 (five hundred), or 1000 (one thousand).
5. Write the ORDER BY clause to sort the result by ISBN.
6. Execute the script.

```
USE library
SELECT co.isbn, co.copy_no, co.on_loan
      ,ti.title, it.translation, it.cover
 FROM copy co
 INNER JOIN title AS ti
  ON co.title_no = ti.title_no
 INNER JOIN item AS it
  ON co.isbn = it.isbn
 WHERE co.isbn IN ( 1, 500, 1000)
 ORDER BY co.isbn
GO
```

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

Result

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

isbn	copy_no	on_loan	title	translation	cover
1	1	N	Last of the Mohicans	ARABIC	HARDBACK
1	2	N	Last of the Mohicans	ARABIC	HARDBACK
1	3	N	Last of the Mohicans	ARABIC	HARDBACK
1	4	N	Last of the Mohicans	ARABIC	HARDBACK

```
.
.
.
(30 row(s) affected)
```

► To join multiple tables by using an outer join

In this procedure, you will write and execute a query to retrieve the member's full name and **member_no** from the **member** table and the **isbn** and **log_date** values from the **reservation** table for member numbers 250, 341, and 1675. Order the results by **member_no**. You should show information for these members, even if they have no books on reserve.

Answer_LeftOuter.sql is a completed script for this procedure.

1. Write the select list of the query:
   a. Create the **name** column by concatenating the **lastname**, **firstname**, and **middleinitial** for each member.
   b. Create the **date** column by converting the **log_date** to the **char(8)** data type.
2. Write a FROM clause that creates an left outer join between the **member** and **reservation** tables on the **member_no** columns.
3. Compose a WHERE clause to retrieve member numbers 250, 341, and 1675 from the **member** table.
4. Write the ORDER BY clause to sort the result by the member numbers.

```
USE library
SELECT me.member_no
      ,me.lastname + ', ' + me.firstname + ' '
        + me.middleinitial AS name
      ,re.isbn
      ,CONVERT(char(8),re.log_date,1) AS date
 FROM member AS me
 LEFT OUTER JOIN reservation AS re
  ON me.member_no = re.member_no
 WHERE me.member_no IN ( 250, 341, 1675 )
 ORDER BY me.member_no
GO
```

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

   Which members have no books on reserve?

   250 and 1675

Result

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

member_no	Name	isbn	Date
250	Hightower, Michael A	NULL	NULL
341	Martin, Brian A	43	11/21/98
341	Martin, Brian A	330	11/21/98
341	Martin, Brian A	617	11/21/98
341	Martin, Brian A	904	11/21/98
1675	LaBrie, Joshua B	NULL	NULL

6 row(s) affected)

Exercise 2
Using the UNION Operator to Combine Result Sets

In this exercise, you will produce a single result set by using the UNION operator to concatenate the results of two similar SELECT statements. C:\Moc\2071A\Labfiles\L05\Answers contains completed scripts for this exercise.

► To determine which members living in Arizona have more than two children with library cards

In this procedure, you will determine which members living in Arizona have more than two children with library cards. Answer_Union1.sql is a completed script for this procedure.

1. Write a SELECT statement that returns **member_no** and the number of juvenile records that each member has in a calculated field called **numkids**. Only return records for library members living in Arizona that have more than two kids.

```
USE library
SELECT a.member_no
      ,count(*) AS numkids
 FROM juvenile AS j
 INNER JOIN adult AS a
  ON  j.adult_member_no = a.member_no
 WHERE a.state = 'AZ'
 GROUP BY a.member_no
 HAVING COUNT(*) > 2
GO
```

2. Execute the query to verify that it returns the desired results. Note how many rows are returned.
3. Do not erase the query.

► To determine which members living in California have more than three children with library cards

In this procedure, you will determine which members living in California have more than three children with library cards. Answer_Union2.sql is a completed script for this procedure.

1. Press CTRL+N, and create a new query window.
2. Copy the query from the first procedure of this exercise and paste it into the new query window.

3. Modify the query in step 2 such so that it only returns records for library members living in California that have more than three children with library cards.

```
USE library
SELECT a.member_no
      ,count(*) AS numkids
 FROM juvenile AS j
 INNER JOIN adult AS a
  ON  j.adult_member_no = a.member_no
 WHERE a.state = 'CA'
 GROUP BY a.member_no
 HAVING COUNT(*) > 3
GO
```

4. Execute the query to verify that it returns the desired results. Note how many rows are returned.

► To combine the result sets of separate queries

In this procedure, you will combine the result sets of separate queries. Answer_Union3.sql is a completed script for this procedure.

1. Press CTRL+N, and create a new query window.
2. Copy the query from the first procedure of this exercise and paste it into the new window.
3. Add the UNION statement on a new line at the end of the query.

4. Copy the query from the second procedure of this exercise and paste it into the new window on the line following the UNION statement that you add in step 3 in this procedure.

```
USE library
SELECT a.member_no
      ,count(*) AS numkids
 FROM juvenile AS j
 INNER JOIN adult AS a
  ON  j.adult_member_no = a.member_no
 WHERE a.state = 'AZ'
 GROUP BY a.member_no
 HAVING COUNT(*) > 2
UNION
SELECT a.member_no
      ,count(*) AS numkids
 FROM juvenile AS j
 INNER JOIN adult AS a
  ON  j.adult_member_no = a.member_no
 WHERE a.state = 'CA'
 GROUP BY a.member_no
 HAVING COUNT(*) > 3
GO
```

5. Execute the query to verify that it returns the desired results. Note how many rows are returned.

   How does the number of rows that this query returns compare to the number of rows that the queries in the first two procedures return?

   The UNION statement combines the result sets of the first two queries into a single records set. The number of rows that the last procedure returns should equal the sum of the rows returned by the first two procedures.

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.

- Using Aliases for Table Names
- Combining Data from Multiple Tables
- Combining Multiple Result Sets

The Duluth Mutual Life health care organization has a database that tracks information about doctors and their patients. The database includes the following tables.

Doctor table

Column	Data type and constraints
doc_id	char(9), PRIMARY KEY
fname	char(20)
lname	char(25)
specialty	char(25)
phone	char(10)

Patient table

Column	Data type and constraints
pat_id	char(9), PRIMARY KEY
fname	char(20)
lname	char(25)
insurance_company	char(25)
phone	char(10)

Casefile table

Column	Data type and constraints
admission_date	datetime, PRIMARY KEY (composite)
pat_id	char(9), PRIMARY KEY (composite), FOREIGN KEY to patient.pat_id
doc_id	char(9), FOREIGN KEY to doctor.doc_id
diagnosis	varchar(150)

Based on this table structure, answer the following questions.

1. How can you generate a list of patient names and hospital admission dates?

   Join the patient table to the casefile table on the pat_id column.

2. How can you generate a list of patient names for a particular doctor?

   You must join all three tables. The relationship between doctor and patient is a many-to-many relationship. Even though you only want information from the doctor and patient tables, you must also use the casefile table, because this table relates doctor to patient. Join the doctor table to the casefile table on doc_id and then join the patient table to the casefile table on pat_id. Use a WHERE clause to limit the results for a particular doctor.

3. How can you produce a list of pairs of doctors who have the same specialty?

   Join the doctor table to itself. Join the two copies of the table on the specialty column. Restrict the results to rows where doc_id does not match. Be sure to eliminate mirror image pairs by using a greater than (>) operator in the WHERE clause.

4. How can you produce a single list of names and phone numbers for both doctors and patients?

   Write a query that retrieves name and phone number information from the doctor table. Write a second query that retrieves similar information from the patient table. Use the UNION operator to combine the queries.