MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 12: Programming Across Multiple Servers

Contents

Microsoft

Project Lead: Rich Rose
Instructional Designers: Rich Rose, Cheryl Hoople, Marilyn McGill
Instructional Software Design Engineers: Karl Dehmer, Carl Raebler, Rick Byham
Technical Lead: Karl Dehmer
Subject Matter Experts: Karl Dehmer, Carl Raebler, Rick Byham
Graphic Artist: Kirsten Larson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Copy Editor: Edward McKillop (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Support: Lori Walker (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: TestingTesting123
Classroom Automation: Lorrin Smith-Bates
Creative Director, Media/Sim Services: David Mahlmann
Web Development Lead: Lisa Pease
CD Build Specialist: Julie Challenger
Online Support: David Myka (S&T Consulting)
Localization Manager: Rick Terek
Operations Coordinator: John Williams
Manufacturing Support: Laura King; Kathy Hershey
Lead Product Manager, Release Management: Bo Galford
Lead Product Manager, Data Base: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
60 Minutes

Lab:
60 Minutes

This module provides students with an introduction to programming across multiple servers. It describes how to execute an ad hoc query on a remote data source. It then describes how to set up a linked server environment, including setting up linked servers and establishing security between servers. The module also presents how to execute linked server and pass-through queries on a linked server, execute stored procedures on a linked server, modify distributed data, and manage distributed transactions. The final section discusses how to use partitioned views to optimize performance.

In the lab, students set up a linked environment, query linked data sources, execute pass-through and ad hoc queries and remote stored procedures, and manage distributed transactions.

After completing this module, students will be able to:

- Describe distributed queries.
- Write ad hoc queries that access data that is stored in a remote Microsoft® SQL Server™ 2000 or in an OLE DB data source.
- Set up a linked server environment to access data that is stored in a remote SQL Server 2000, or in an OLE DB data source.
- Write queries that access data from a linked server.
- Execute stored procedures on a remote server or linked server.
- Manage distributed transactions.
- Use distributed transactions to modify distributed data.
- Use partitioned views to increase performance.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the Microsoft PowerPoint® file 2073A_12.ppt.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the lab.

Important The examples in this module are provided only for reference. They cannot be executed.

Module Strategy

Use the following strategy to present this module:

- Introduction to Distributed Queries

  Describe distributed queries and how they access data from multiple heterogeneous data sources that can be stored on the same or different computers. Compare the two techniques—ad hoc query and linked server query—to access an OLE DB data source from SQL Server and explain when to use each one. Point out how to specify where to process a distributed query—on the local server or on a remote server.

- Executing an Ad Hoc Query on a Remote Data Source

  Describe when and how to use the OPENROWSET function to execute an ad hoc query on a remote data source.

- Setting Up a Linked Server Environment

  Describe why setting up an environment of linked servers is beneficial. In this environment you can control where the query is executed (locally or remotely). Linked servers allow a more flexible approach to security for accessing remote data. You can choose to have users' credentials passed to the linked server, or, alternatively, to map local SQL Server accounts or Microsoft Windows® 2000 groups to accounts or groups on the linked server.

- Executing a Query on a Linked Server

  Make sure that students are aware of the four-part names that are required for querying linked data. Also, go over why certain Transact-SQL statements and actions cannot be performed on linked servers. Describe and discuss when and how to execute linked server queries and pass-through queries.

- Executing a Stored Procedure on a Linked Server

  Point out that you can execute stored procedures on linked servers.

- Managing Distributed Transactions

  Point out that students can build distributed applications by using pre-built and custom components that encapsulate business logic. Components can use the services provided by Windows 2000 Component Services.

- Modifying Data on a Linked Server

  Point out that modifications that must maintain consistency across multiple databases should be performed in a distributed transaction. Use the BEGIN DISTRIBUTED TRANSACTION statement with a matching COMMIT or ROLLBACK statement.

- Using Partitioned Views

  Point out that partitioned views can increase performance by distribution of processing across multiple servers.

Customization Information

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

Important The lab in this module is dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Classroom Setup Guide* for course 2073A, *Programming a Microsoft SQL Server 2000 Database*.

Lab Setup

The following section describes the setup requirement for the lab in this module.

Setup Requirement

The lab in this module requires the **ClassNorthwind** database to be in a state required for this lab. To prepare student computers to meet this requirement, perform one of the following actions:

- Complete the prior lab
- Execute the C:\Moc\2073A\Batches\Restore12.cmd batch file.

Warning If this course has been customized, students must execute the C:\Moc\2073A\Batches\Restore12.cmd batch file to ensure that the lab will function properly.

Lab Results

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

Overview

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

Lead-in
In this module, you will learn how to program across multiple servers by using distributed queries, distributed transactions, and partitioned views.

- Introduction to Distributed Queries
- Executing an Ad Hoc Query on a Remote Data Source
- Setting Up a Linked Server Environment
- Executing a Query on a Linked Server
- Executing a Stored Procedure on a Linked Server
- Managing Distributed Transactions
- Modifying Data on a Linked Server
- Using Partitioned Views

This module introduces programming across multiple servers. It describes how to execute an ad hoc query on a remote data source. It then describes how to set up a linked server environment, including setting up linked servers and establishing security between servers. The module also presents how to execute linked server and pass-through queries on a linked server, execute stored procedures on a linked server, modify distributed data, and manage distributed transactions. The final section discusses how to use partitioned views to optimize performance.

After completing this module, you will be able to:

- Describe distributed queries.
- Write ad hoc queries that access data that is stored in a remote Microsoft® SQL Server™ 2000 or in an OLE DB data source.
- Set up a linked server environment to access data that is stored in a remote SQL Server 2000 or in an OLE DB data source.
- Write queries that access data from a linked server.
- Execute stored procedures on a remote server or linked server.
- Manage distributed transactions.
- Use distributed transactions to modify distributed data.
- Use partitioned views to increase performance.

Introduction to Distributed Queries

Topic Objective
To introduce distributed queries.

Lead-in
Distributed queries access data from multiple heterogeneous data sources, which is stored on the local or remote computer.

- **Accessing Remote Data**
  - Ad hoc query
  - Linked server query
- **Specifying Where to Process Distributed Queries**
  - Local SQL Server
  - Remote OLE DB data source (pass-through query)
- **Verifying Connection Settings**

Distributed queries access data from multiple heterogeneous data sources stored on a local or remote computer. SQL Server supports distributed queries by using the Microsoft OLE DB Provider.

Distributed queries provide SQL Server users with access to:

- Distributed data stored on multiple computers that are running SQL Server.
- Heterogeneous data stored in various relational and non-relational data sources for which either an OLE DB provider or Open Database Connectivity (ODBC) driver exists.

Accessing Remote Data

You can use two techniques for accessing an OLE DB data source from SQL Server:

- Ad hoc query

  To access remote data when you do not expect to access a data source repeatedly over time, you can write an ad hoc query with the OPENROWSET or OPENDATASOURCE function.

- Linked server query

  To access remote data repeatedly, you can use a linked server and a four-part object name. A linked server is an OLE DB data source that is pre-registered on the local SQL Server so that when it is referenced, the local server knows where to look for the remote data and objects. Using linked servers is an efficient way to provide cross-SQL Server joins and other queries when you know in advance that certain data sources must be available.

Specifying Where to Process Distributed Queries

When you query an OLE DB data source, you can specify whether to process the query locally or on a remote server:

- Local SQL Server

  For linked servers, SQL Server processes distributed queries on the local server by default.

- Remote OLE DB data source

  You can use the OPENQUERY function with linked servers to specify that processing will occur on the remote server. This is called a pass-through query. When you use the OPENROWSET function to execute an ad hoc query on a remote data source, the query is also processed remotely.

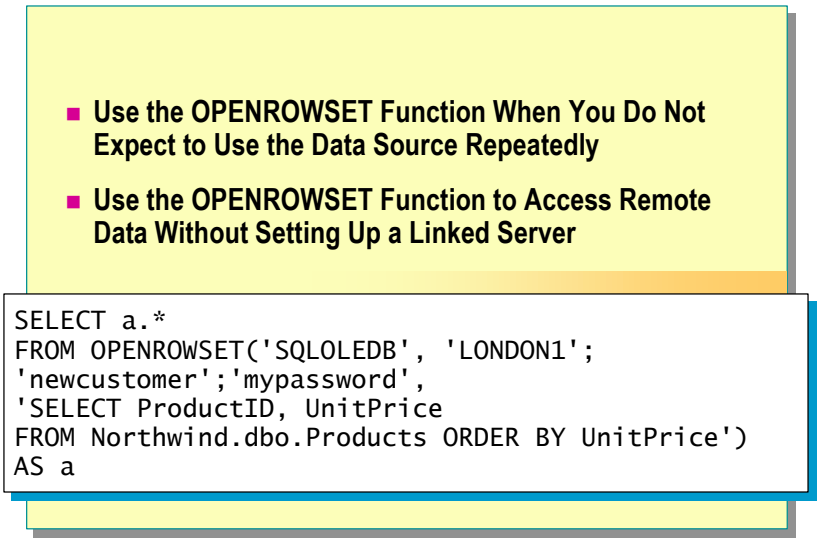
Verifying Connection Settings

In any session issuing distributed queries, the ANSI_NULLS and ANSI_WARNINGS options must be on. If you use OBDC or SQL Query Analyzer to issue distributed queries, these options are on by default. If you use the **osql** command line utility, you must explicitly set these options ON.

Executing an Ad Hoc Query on a Remote Data Source

Topic Objective
To describe how to execute an ad hoc query on a remote data source.

Lead-in
You can access data ad hoc from remote sources by using an OLE DB provider.



You can access data ad hoc from remote sources by using an OLE DB provider. The OPENROWSET function allows you to connect to and access data from a remote source without setting up a linked server. Use the OPENROWSET function when you do not expect to access a particular data source repeatedly over time.

Syntax

OPENROWSET('*provider_name*'
{'*data -source*'; '*user_id*' ; '*password*' | '*provider_string*'},
{[*catalog*.][*schema*.]*object* | '*query*'})

The following table describes the parameters of the OPENROWSET function.

Parameter	Description
provider_name	Unique, friendly name for the OLE DB provider corresponding to this data source.
data_source	Name of the data source as interpreted by the OLE DB provider.
user_id	User name that will be passed to the specified OLE DB provider.
password	Password to be passed to the OLE DB provider.
provider_string	OLE DB provider-specific connection string that identifies a unique data source.
catalog	Catalog or database in which the object resides.
schema	Schema or owner for an object.
object	Unique object name to act upon.
query	String containing a query to be sent to and executed by the provider. If a query is specified rather than a remote object name, the query is executed as a pass-through query.

The following table lists some common OLE DB provider names. See SQL Server Books Online for a more complete listing of OLE DB provider names for various data sources.

Product	Provider name
SQL Server	N'SQLOLEDB'
Microsoft OLE DB Provider for Access (Jet)	'Microsoft.Jet.OLEDB.4.0'
Microsoft OLE DB Provider for Oracle	'MSDAORA' *data_source* is the SQL*Net alias name for the Oracle database to be added as a linked server
OLE DB Provider for ODBC (Using *data_source* parameter)	*provider_name* is 'MSDASQL' *data_source* is 'LocalServer'
OLE DB Provider for ODBC (Using *provider_string* parameter)	*provider_name* is 'MSDASQL' *provider_string* is 'DRIVER={SQL Server} SERVER=*servername* UID=*login*;PWD=*password*;'
Data Transformation Services	DTSPackageDSO
Microsoft Directory Services	ADSDSOObject
Microsoft Indexing Service	MSIDXS

Consider the following facts and guidelines when executing queries by using the OPENROWSET function:

- You must provide catalog and schema names if the data source supports multiple catalogs and schemas (databases and object owners, in the case of SQL Server).
- The *user_id* passed to the OLE DB provider determines the permissions associated with the connection.
- The OPENROWSET function can be used in place of a table name in the FROM clause of a SELECT statement.

Example 1

This example uses the native OLE DB provider for SQL Server to access information in the **Northwind** database on the London1 SQL Server. All connection information as well as the query to be processed is contained in the arguments of the OPENROWSET function. The **newcustomer** user account is used to log in to the remote server.

```
SELECT a.*
FROM OPENROWSET('SQLOLEDB', 'LONDON1'; 'newcustomer';
'mypassword',
'SELECT ProductID, UnitPrice FROM Northwind.dbo.Products
        ORDER BY UnitPrice')
 AS a
```

Example 2

This example uses the OLE DB provider for Microsoft Access (Jet) to access the **Orders** table in the **Northwind** database on a remote Access database.

```
SELECT a.*
FROM OPENROWSET('Microsoft.Jet.OLEDB.4.0'
'C:\MSOffice\Access\Samples\Northwind.mdb';
'newcustomer'; 'mypassword',
Orders)
 AS a
```

Example 3

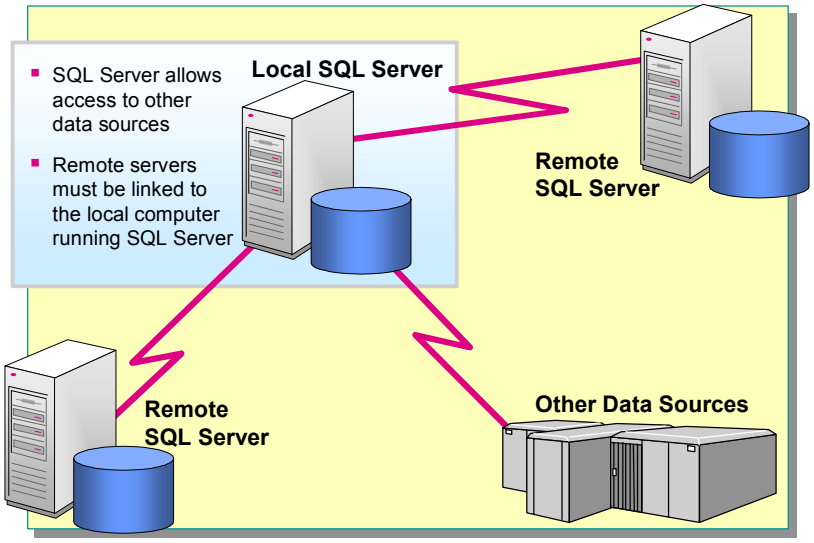
This example joins the **Orders** table in the **Northwind** database on a remote Access database with the **Customers** table in the **Northwind** database on the local SQL Server.

```
USE Northwind
SELECT cust.* ord.*
FROM  Customers as cust JOIN
OPENROWSET('Microsoft.Jet.OLEDB.4.0'
'C:\MSOffice\Access\Samples\Northwind.mdb'';
'newcustomer'; 'mypassword',
Orders)
 AS ord
On cust.customerid = ord.customerid
```

Setting Up a Linked Server Environment

Topic Objective
To describe why a distributed environment is useful and the steps to set up one.

Lead-in
To work with data from a remote SQL Server or another OLE DB data source, you must establish a linked server.



To work with data from a remote SQL Server or another OLE DB data source, you must establish a linked server. A linked server is an OLE DB data source that is pre-registered on the local SQL Server so that when it is referenced, the local server knows where to look for the remote data and objects.

Why Use Linked Servers

Linked servers are a way to enable cross-SQL Server joins and other queries when you know in advance that you want certain data sources to be available.

Using a linked server gives you the ability to submit Transact-SQL statements directly to a remote SQL Server. These actions can be performed as part of a distributed transaction. When you use linked servers, consider the following facts and guidelines:

- You can access distributed data that is stored in multiple SQL Servers and heterogeneous data that is stored in various relational and non-relational data sources.
- A source other than SQL Server can be defined as a linked server if an OLE DB provider exists for the source.
- If you regularly access information that resides on another SQL Server computer, you should define that remote server as a linked server on your local SQL Server computer.
- Information about linked servers is stored in the **sysservers** system table.

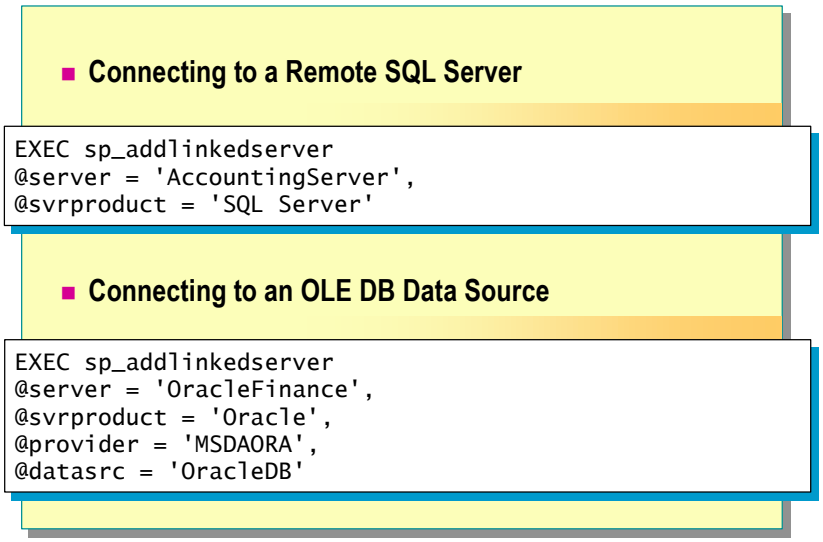
Setting Up Linked Servers

To set up linked servers, you must first establish a link to a remote data source and then establish security between the servers.

Linking to a Remote Data Source

Topic Objective
To describe how to link to a remote server or OLE DB data source.

Lead-in
To run Transact-SQL statements on a remote SQL Server or OLE DB data source, you must establish a link to the server or data source.



Delivery Tip
Demonstrate linking to a remote data source by using SQL Server Enterprise Manager.

To execute Transact-SQL statements on a remote SQL Server or OLE DB data source, you must establish a link to the server or data source.

You can establish a link to the remote SQL Server by using SQL Server Enterprise Manager or the **sp_addlinkedserver** system stored procedure. The **sp_addlinkedserver** system stored procedure defines a remote SQL Server on the local computer and specifies the OLE DB provider.

Syntax

sp_addlinkedserver [**@server** =] '*server*'
[, [**@srvproduct** =] '*product_name*']
[, [**@provider** =] '*provider_name*']
[, [**@datasrc** =] '*data_source*']
[, [**@location** =] '*location*']
[, [**@provstr** =] '*provider_string*']
[, [**@catalog** =] '*catalog*']

The **sp_addserver** system stored procedure is provided for backward compatibility, but you should use **sp_addlinkedserver** instead.

The following table describes the parameters of the **sp_addlinkedserver** system stored procedure.

Parameter	Description
@server	Name of the linked server to create
@svrproduct	Product name of the OLE DB data source
@provider	The unique, friendly name for the OLE DB provider corresponding to this data source
@datasrc	Name of the data source as interpreted by the OLE DB provider
@location	Location of the database as interpreted by the OLE DB provider

(*continued*)

Parameter	Description
@provstr	The OLE DB provider-specific connection string that identifies a unique data source
@catalog	The catalog to use when making a connection to the OLE DB provider

Connecting to a Remote SQL Server

If you want to connect to a server that is running SQL Server, the only parameters that you must provide are **@srvproduct** and **@server**. You do not need to specify **@provider, @datasrc**, **@location**, **@provstr**, and **@catalog**. The SQL Server OLE DB provider (N'SQLOLEDB'**)** is automatically used.

Example 1

This example adds the **AccountingServer**, a computer running SQL Server, to the list of linked servers that are available from the local SQL Server computer.

```
EXEC sp_addlinkedserver 'AccountingServer', 'SQL Server'
```

Connecting to an OLE DB Data Source

If you want to connect to a data source other than SQL Server, you must specify a **@provider**, **@datasrc**, **@location**, **@provstr**, and **@catalog**, as well as the **@srvproduct** and **@server** parameters when creating a linked server.

Example 2

This example adds the Oracle server OracleFinance to the list of linked servers that are available from the local computer running SQL Server. This example assumes that a SQL*Net alias of 'OracleDB' has been created. This alias is used for the **@datasrc** parameter.

```
EXEC sp_addlinkedserver 'OracleFinance', 'Oracle', 'MSDAORA',
'OracleDB'
```

Establishing Linked Server Security

Topic Objective
To describe how to establish security between linked servers.

Lead-in
You may need to establish security between the local server and a remote server.

- **Local Server Must Log In to Remote Server on Behalf of User**
- **If User's Login Account Exists on Both Servers, It Can Be Used to Log In to Remote Server**
- **Map Login Accounts and Passwords Between Servers by Using sp_addlinkedsrvlogin**
- **By Using Security Account Delegation, You Can Connect to Multiple Servers with One Authentication**
- **Without Security Delegation, Map Local Login Account to Login Account on the Linked Server**

You may need to establish security between the local server and a remote server. When you establish security between local and remote SQL Servers, consider the following facts:

- When users log in to the local SQL Server and execute a distributed query, the local SQL Server logs in to the remote SQL Server on behalf of the user.
- If the user's login account and password exist on both the local and remote SQL Servers, the local SQL Server can use the credentials of the user to log in to the remote SQL Server. Establishing security in this manner is useful when both servers are using domain accounts.
- You can map login accounts and passwords between local and remote SQL Servers by using the **sp_addlinkedsrvlogin** system stored procedure. When you map a local account to a remote login account, you do not have to create a login account and password for each user on the remote SQL Server.

  For example, a user can log in to a client application that accesses a local SQL Server. The local SQL Server then accesses the linked server on behalf of the user by using one login account for all end users. The login account on the linked server to which the local login account is mapped has permission to access a specific table.
- It is possible to connect to multiple servers, and with each server change, to retain the authentication credentials of the original client. This is known as *security account delegation*. To use this delegation, all servers must be running Microsoft Windows® 2000 and using providers that employ the Security Support Provider Interface (SSPI). You must also be using the Active Directory™ directory service.

Note For more information about security account delegation, consult the Windows 2000 documentation.

- If the linked server does not support security account delegation, you must set up a local login mapping from a Windows Authenticated login account to a login account on the linked server. You must establish an account mapping to enable linked server communication.

Syntax

sp_addlinkedsrvlogin [**@rmtsrvname** =] '*rmtsrvname*'
[, [**@useself** =] '*useself*']
[, [**@locallogin** =] '*locallogin*']
[, [**@rmtuser** =] '*rmtuser*']
[, [**@rmtpassword** =] '*rmtpassword*'

The following table lists the parameters of the **sp_addlinkedsrvlogin** system stored procedure.

Parameter	Description
@rmtsrvname	Name of a linked server to which the login mapping applies.
@useself	Determines whether SQL Server login accounts use their own credentials or the values of the **@rmtuser** and **@rmtpassword** arguments to connect to the server specified by the **@rmtsrvname** argument. A value of TRUE for **@useself** is invalid for a Windows Authenticated login account.
@locallogin	An optional login account on the local server. If used, **@locallogin** must already exist on the local server. If this value is null, then all login accounts on the local SQL Server will be mapped to the account on the remote server specified by **@rmtuser**.
@rmtuser	The optional user name for connection to **@rmtsrvname** when **@useself** is FALSE.
@rmtpassword	The optional password associated with **@rmtuser**.

Example 1

In this example, a user who logs in to the local SQL Server with the **AccountWriter** login account will be able to access remote data on the AccountingServer SQL Server with the credentials of the **rmtAccountWriter** login account.

```
EXEC sp_addlinkedsrvlogin
@rmtsrvname = 'AccountingServer',
@useself = 'false',
@locallogin = 'Accountwriter',
@rmtuser = 'rmtAccountWriter',
@rmtpassword = 'financepass'
```

Example 2

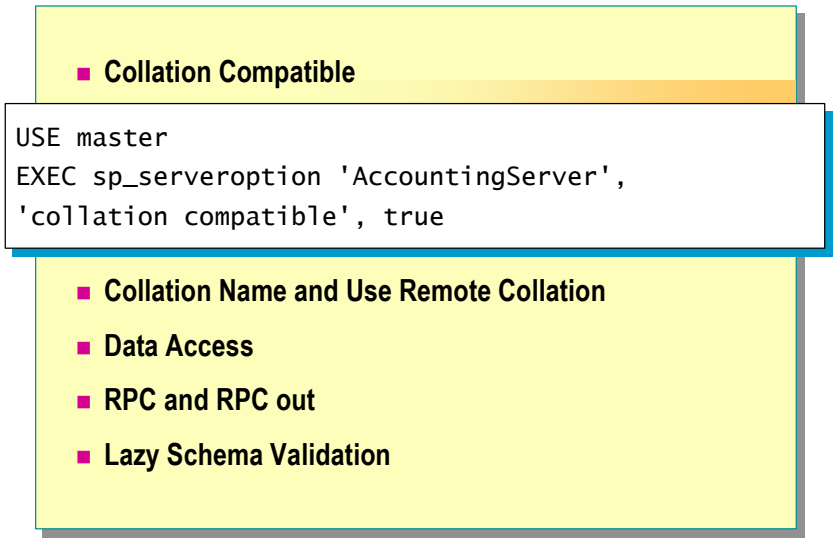
This example establishes security between a local and a linked SQL Server. Any users on the local SQL Server who access remote data on the AccountingServer linked server are logged in to the remote SQL Server with the **AccountingServer/allcustomers** user account.

```
EXEC sp_addlinkedsrvlogin
@rmtsrvname = 'AccountingServer',
@useself = 'false',
@rmtuser= 'allcustomers'
```

Configuring Linked Server Options

Topic Objective
To describe how to establish security between linked servers.

Lead-in
You may need to establish security between the local server and a remote server.



You can set options for linked servers by using the **sp_serveroption** system stored procedure. Only a member of the **sysadmin** server role can use **sp_serveroption** to set server options.

Syntax

sp_serveroption ['*server*'] [**,** '*option_name*'] [**,** '*option_value*']

The following options affect linked servers:

Collation Compatible

This option affects the performance of distributed query execution against linked servers. If this option is set to true, SQL Server assumes that all columns and character sets on the remote server are compatible with the local server-wide character set and collation. This enables SQL Server to send comparisons on character columns to the provider. If this option is not set, SQL Server must return all of the rows to the local server to evaluate comparisons on character columns.

This option should only be set if the data source that corresponds to the linked server has the same character set and sort order as the local server.

Example

This example configures the AccountingServer linked server to be collation compatible with the local SQL Server.

```
USE master
EXEC sp_serveroption 'AccountingServer',
'collation compatible', true
```

Collation Name and Use Remote Collation

The following two options are often used together.

Collation Name This option specifies the name of the collation used by the remote data source if **use remote collation** is **true** and the data source is not a SQL Server data source. The following conditions apply to this option:

- The name must be one of the collations supported by SQL Server.
- You should use this option when accessing an OLE DB data source, other than SQL Server, that has a collation that matches one of the SQL Server collations.
- The linked server must support a single collation to be used for all columns in that server.

Use Remote Collation This option determines whether SQL Server will use the collation of a remote column or of a local server:

- If **true**, the collation of remote columns is used for SQL Server data sources, and the collation specified in **collation name** is used for non-SQL Server data sources.
- If **false**, distributed queries will always use the default collation of the local server. The default is **false**.

Data Access

This option enables and disables a linked server for distributed query access. You can only use this for **sysserver** entries that are added by using **sp_addlinkedserver**.

Example

This example configures the AccountingServer remote server for data access and enables its use as a linked server.

```
USE master
EXEC sp_serveroption 'AccountingServer',
'data access', true
```


RPC and RPC out

The RPC option enables remote procedure calls (RPCs) from a given server. The RPC out option enables remote procedure calls to a given server.

Lazy Schema Validation

This option determines whether the schema of remote tables will be checked. If **true**, SQL Server does not check the schema of remote tables at the beginning of the query. Deferring the schema validation can improve performance.

Getting Information About Linked Servers

Topic Objective
To describe how to gather information about linked servers.

Lead-in
In addition to using SQL Server Enterprise Manager, you can use the following stored procedures to get information about linked servers.

System stored procedure	Returns
sp_linkedservers	A list of linked servers defined on the local server
sp_catalogs	A list of catalogs and descriptions for a specific linked server
sp_indexes	Index information for the specified remote table
sp_primarykeys	The primary key columns, one row per key column, for the specified table
sp_foreignkeys	The foreign keys defined on the specified remote table
sp_tables_ex	Table information on the tables from the specified linked server
sp_columns_ex	The column information, for all columns or a specified column, for linked table

In addition to using SQL Server Enterprise Manager, you can use the following stored procedures to gather information about linked servers:

sp_linkedservers This system stored procedure returns a list of linked servers that are defined on the local server.

sp_catalogs This system stored procedure returns a list of catalogs and descriptions for a specified linked server. For remote SQL Servers, this is a list of available databases.

sp_indexes This system stored procedure returns index information for the specified remote table.

Syntax

sp_indexes {'*table_server*'} [**,** '*table_name*'] [**,** '*table_schema*'] [**,** '*table_catalog*'][**,** '*index*'] [**,** '*is_unique*']

Example

This example returns all index information from the **Employees** table of the **Northwind** database on the Cairo server.

```
USE master
EXEC sp_indexes 'CAIRO', 'Employees', 'dbo', 'Northwind',
NULL, 0
```

sp_primarykeys This system stored procedure returns the primary key columns, one row per key column, for the specified table.

sp_foreignkeys This system stored procedure returns the foreign keys that are defined on the specified remote table.

sp_tables_ex This system stored procedure is a version of **sp_tables** for use with remote data sources, and it returns table information on the tables from the specified linked server.

Syntax

sp_tables_ex {'*table_server*'} [**,** '*table_name*'] [**,** '*table_schema*'] [**,** '*table_catalog*'] [**,** '*table_type*']

sp_columns_ex This is a version of **sp_columns** for linked servers. This system stored procedure returns the column information, for all columns or a specified column, for the given linked server table. If a column is specified, only information for that particular column is returned.

Syntax

sp_columns_ex {'*table_server*'} [**,** '*table_name*'] [**,** '*table_schema*'] [**,** '*table_catalog*'] [**,** '*column*']

Executing a Query on a Linked Server

Topic Objective
To introduce how to execute a query on a linked server.

Lead-in
SQL Server can process a distributed query locally on the local server, or remotely on a linked server.

- Working with Linked Servers
- Executing Linked Server Queries
- Executing Pass-Through Queries

SQL Server can process a distributed query locally on the local server, or remotely on a linked server.

Working with Linked Servers

Topic Objective
To describe how to access data from tables on a linked server.

Lead-in
In a query, you can include multiple data sources.

- **How SQL Server Optimizes Remote Queries**
- **Referring to Objects on Linked Servers**
- **Allowed Transact-SQL Statements**
  - SELECT, INSERT, UPDATE, DELETE
- **Disallowed Transact-SQL Statements**
  - CREATE, ALTER, DROP
  - ORDER BY on remote tables containing large objects
  - READTEXT, WRITETEXT, UPDATETEXT

Distributed queries access data from multiple data sources, such as OLE DB providers and other SQL Servers.

How SQL Server Optimizes Remote Queries

SQL Server attempts to delegate distributed query evaluation to the OLE DB providers. SQL Server extracts from the original distributed query those syntactical elements that access only the remote tables in the provider's data source, and then it executes this reduced query against the provider. This process reduces the number of rows returned from the provider and allows the provider to use its indexes to evaluate the query.

Referring to Objects on Linked Servers

When you perform distributed queries, you must refer to the linked objects with four-part names in the following format:
linked-server-name.catalog-name.schema-name.object-name

The following table describes these parameters.

Parameter	Description
linked-server-name	Is the network-wide name of a linked server
catalog-name	Corresponds to a database
schema-name	Is the collection of objects that are owned by a particular user and corresponds to the object owner
object-name	Refers to the table that you want to access

For example, to refer to the **Orders** table that is owned by the **database owner** (**dbo**) role in the **Northwind** database on the linked server, Corpserver, use the four-part name **corpserver.Northwind.dbo.Orders** in your query.

Allowed Transact-SQL Statements

When you use a linked SQL Server, you can execute the following Transact-SQL statements on linked data:

- SELECT statement with a WHERE clause or a JOIN clause
- INSERT, UPDATE, and DELETE statements

Disallowed Transact-SQL Statements

Delivery Tip
Point out that this is only a partial list.

When you use a linked SQL Server, you cannot:

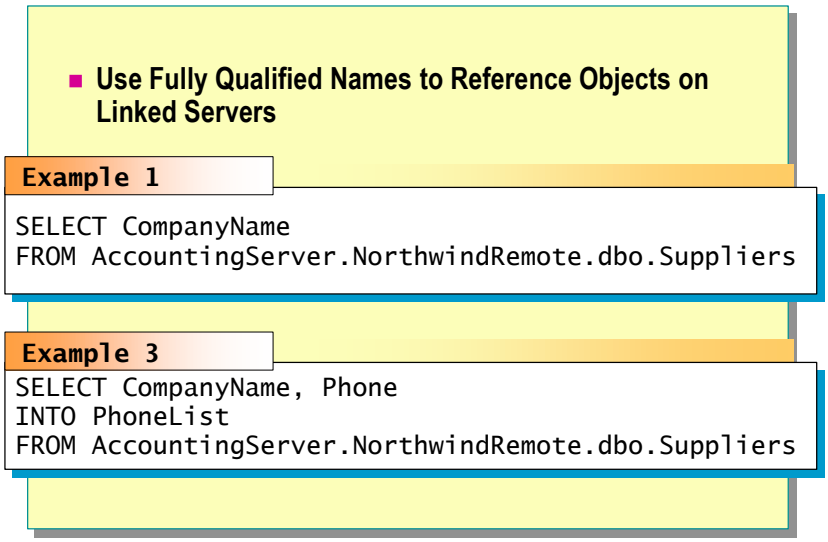
- Use the CREATE, ALTER, or DROP statements on linked servers.

  Therefore, you cannot execute a CREATE TABLE statement that contains a SELECT INTO statement. However, you can use linked data as the source for tables that are created on the local server with the SELECT INTO statement.
- Include an ORDER BY clause in a SELECT statement if a large object column from a linked table is in the select list of the SELECT statement.
- Use the READTEXT, WRITETEXT, and UPDATETEXT statements.

Executing Linked Server Queries

Topic Objective
To describe how to query a linked server.

Lead-in
When you query a linked server, you should reference objects by using the fully qualified four-part object name.



When you query a linked server, you should reference objects by using the fully qualified four-part object name.

Example 1

This example retrieves the company names from the **Suppliers** table in the **NorthwindRemote** database on the AccountingServer linked server.

```
SELECT CompanyName
FROM AccountingServer.NorthwindRemote.dbo.Suppliers
```

Example 2

This example joins the **Suppliers** table in the **NorthwindRemote** database on a linked server to the **Products** table on the local SQL Server.

```
SELECT ProductName, CompanyName
FROM Products p JOIN
AccountingServer.NorthwindRemote.dbo.Suppliers
ON p.supplierid = s.supplierid
```

Example 3

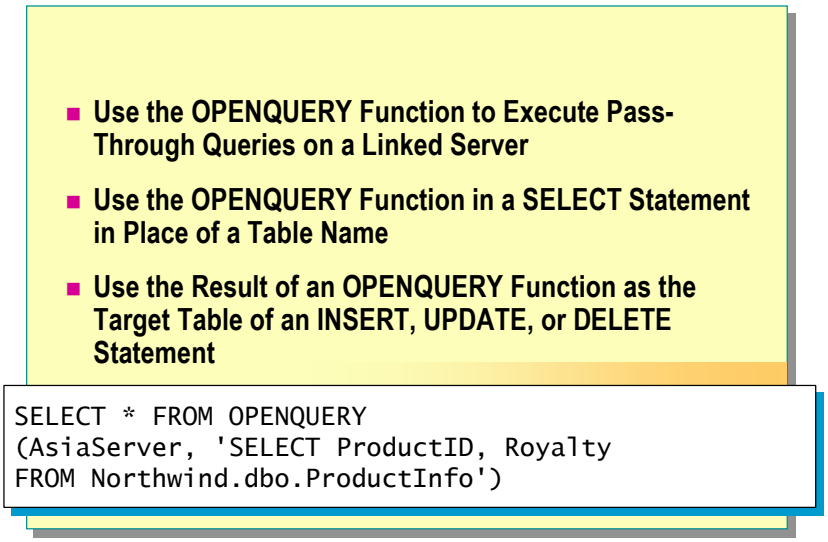
This example uses a SELECT INTO statement to create and transfer data from a table on a linked SQL Server to a permanent table on the local SQL Server. You must set the SELECT INTO/BULK COPY database option if you want to execute this example.

```
SELECT CompanyName, Phone
INTO PhoneList
FROM AccountingServer.NorthwindRemote.dbo.Suppliers
```

Executing Pass-Through Queries

Topic Objective
To describe how to execute pass-through queries on a linked server.

Lead-in
When querying a linked server, you can specify that SQL Server perform a pass-through query.



When querying a linked server, you can specify that SQL Server perform a pass-through query. Use the OPENQUERY function to execute pass-through queries on a linked server.

Syntax

OPENQUERY (*linked_server*, '*query*')

Consider the following facts when performing pass-through queries with the OPENQUERY function:

- You can use the result of the OPENQUERY function with a SELECT statement in the place of a table name.
- You can use the result of the OPENQUERY function as the target table of an INSERT, UPDATE, or DELETE statement if the OLE DB provider for the data source supports these actions.

Example 1

In this example, the OPENQUERY function is used to process a SELECT statement on the AsiaServer linked server and return the results to the local SQL Server. Assume that AsiaServer has already been established as a linked server and that security has been set up.

```
SELECT * FROM OPENQUERY(AsiaServer, 'SELECT ProductID, Royalty
FROM Northwind.dbo.ProductInfo')
```

Example 2

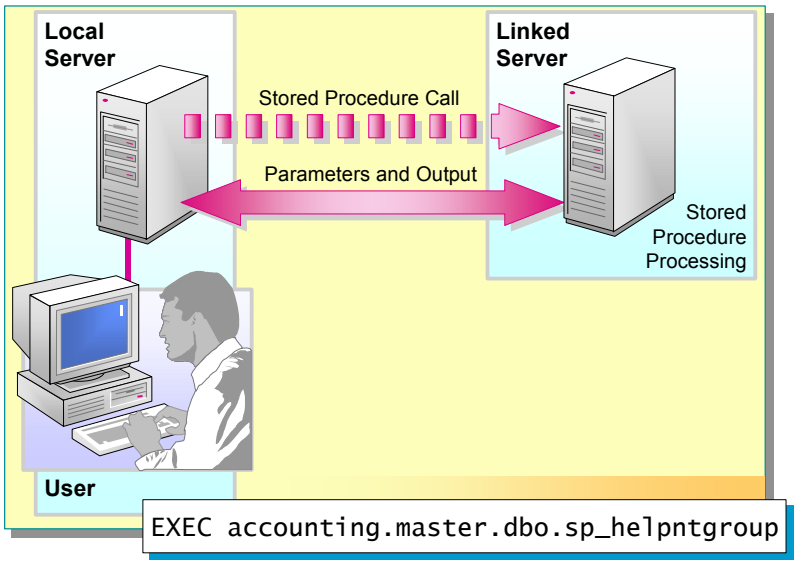
In this example, the OPENQUERY function is used to delete discontinued products from the **Northwind.Products** table on the AsiaServer linked server. All processing of the DELETE statement occurs on the AsiaServer linked server.

```
DELETE FROM OPENQUERY(AsiaServer, 'Northwind.dbo.Products')
WHERE Discontinued = 1
```

Executing a Stored Procedure on a Linked Server

Topic Objective
To illustrate how a stored procedure can be executed remotely.

Lead-in
It is possible to execute a stored procedure on a linked server.



Execution of stored procedures on a linked server allows a client that is connected to one SQL Server to execute a stored procedure on another SQL Server without establishing a client connection to that server.

- The server to which the client is connected accepts the client request and sends the request to the linked server. The EXECUTE statement must contain the name of the linked server as part of its syntax.
- The linked server processes the request and returns any results to the original server, which in turn passes those results to the client.
- Applications on either the client or server can initiate linked stored procedure requests.

Syntax

EXECUTE *servername.dbname.owner. procedure_name*

Example

The following batch executes the **sp_helpntgroup** system stored procedure on the Accounting remote server. The system stored procedure lists the Windows 2000 groups and specifies the databases to which they have access.

```
EXEC accounting.master.dbo.sp_helpntgroup
```

Managing Distributed Transactions

Topic Objective
To describe how to manage distributed transactions.

Lead-in
Distributed transactions coordinate activity on multiple resources as a single unit of work.

- Managing Distributed Transactions by Using MS DTC
- Managing Distributed Transactions by Using Component Services

Distributed transactions coordinate activity on multiple resources as a single unit of work. SQL Server supports distributed transactions, allowing users to update multiple SQL Server databases and other sources of data. You can also use Windows 2000 Component Services to coordinate distributed transactions among components.

Managing Distributed Transactions by Using MS DTC

The Microsoft Distributed Transaction Coordinator (MS DTC) coordinates commitment of a distributed transaction across all servers that participate in the transaction. These servers can include SQL Server in addition to middle-tier components.

You can use MS DTC from a SQL Server stored procedure to coordinate transactions across multiple computers running SQL Server or between a SQL Server and linked servers.

You can add remote computers running SQL Server to a distributed transaction. A stored procedure issues a BEGIN DISTRIBUTED TRANSACTION statement, and then it either makes a remote stored procedure call referencing a remote server or executes a distributed query referencing a remote or linked server.

Managing Distributed Transactions by Using Component Services

Use Component Services to deploy and manage distributed transactions. The underlying mechanism is MS DTC. Components in the middle tier can participate in a distributed transaction.

Modifying Data on a Linked Server

Topic Objective
To introduce distributed transactions.

Lead-in
When you want to modify data on a linked server, you can perform a distributed transaction by executing the BEGIN DISTRIBUTED TRANSACTION statement.

- **Distribute Transactions by:**
  - Executing BEGIN DISTRIBUTED TRANSACTION

    -OR-
  - Calling API functions from a client
- **Consider These Facts:**
  - BEGIN DISTRIBUTED TRANSACTION statements cannot be nested
  - ROLLBACK TRANSACTION rolls back entire transaction
  - Savepoints are not supported
  - Set the XACT_ABORT session option

When you want to modify data on a linked server, you must perform a distributed transaction. You can execute a BEGIN DISTRIBUTED TRANSACTION statement or reference the API functions in a client application.

Syntax

BEGIN DISTRIBUTED TRANSACTION [*transaction_name*]

Example

The following example uses a distributed transaction to transfer funds between two bank accounts stored on different servers. A stored procedure named **withdraw** on the local server is used to withdraw funds from a savings account, and a stored procedure named **deposit** on a linked server is used to deposit funds to a checking account. One hundred dollars is withdrawn from account number 1234 on the local server and deposited in the corresponding checking account on the Centralserver linked server. Both the local and linked databases commit or roll back the transaction.

```
SET XACT_ABORT ON
BEGIN DISTRIBUTED TRANSACTION
  EXEC Savingsdb.dbo.withdraw 1234, 100
  EXEC Centralserver.Checkingdb.dbo.deposit 1234, 100
COMMIT TRAN
```

Consider the following facts when you work with distributed transactions:

- BEGIN DISTRIBUTED TRANSACTION statements cannot be nested. SQL Server detects such calls, rejects them, and reports an error.
- A ROLLBACK TRANSACTION statement rolls back the entire distributed transaction.
- Savepoints are not supported. If SQL Server rolls back a distributed transaction, the entire transaction is rolled back to the beginning of the distributed transaction, regardless of any savepoints.
- You must set the XACT_ABORT session option when performing distributed transactions among linked servers. If a Transact-SQL statement fails when the XACT_ABORT session option is set, the entire transaction is rolled back. If this option is not set, only the statement that failed is rolled back, and transaction processing continues.

Using Partitioned Views

Topic Objective
To introduce how partitioned views can increase performance.

Lead-in
Partitioned views can increase performance by distributing processing across multiple servers.

- The Need for Partitioned Views
- How Partitioned Views Work
- Implementing Distributed Partitioned Views
- Considerations for Partitioning Data

Partitioned views can increase performance by distributing what is processed across multiple servers.

The Need for Partitioned Views

Topic Objective
To describe the need for distributed partitioned views.

Lead-in
When workloads increase, it is important to be able to easily add resources.

- **Scalability**
  - Add more hardware to a single server
  - Divide workload and database across multiple independent computers
- **Benefits of Partitioned Views**
  - Results of separate tables can appear as one table
  - Data location is transparent to the application
  - Database is programmed as a single entity

When workloads increase, it is important to be able to easily add resources.

Scalability

Use scalability to increase the resources of a computer to meet increasing workloads over time. You can achieve scalability by either adding more hardware to a single server or adding multiple independent computers that divide the database. Partitioning the workload across an array is especially suited for e-commerce applications wherein enormous growth will occur.

Benefits of Partitioned Views

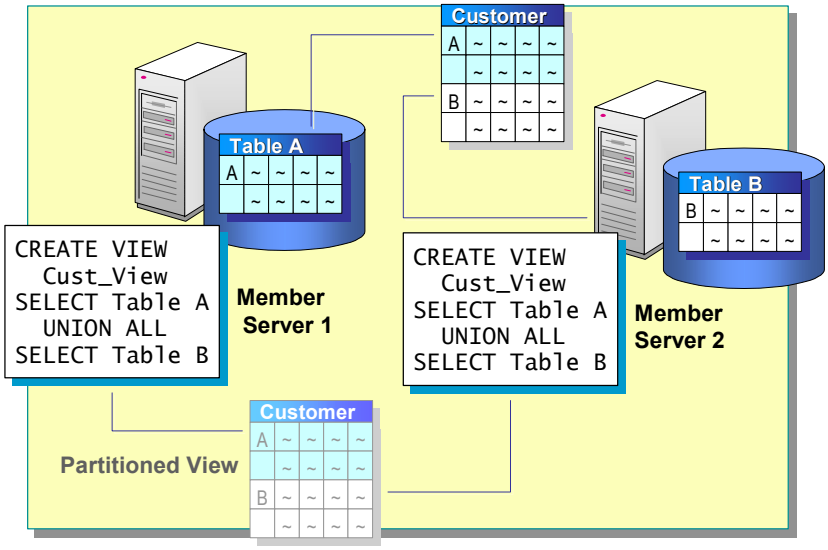
You can use views to partition data across multiple databases or instances of SQL Server. The benefits of using partitioned views are as follows:

- The results of separate tables can be combined into one result set that appears to the user as a single table called a *partitioned view*.
- The location of the data is transparent to the application.
- The database is programmed as a single entity.

How Partitioned Views Work

Topic Objective
To describe how partitioned views work.

Lead-in
Partitioned views allow the data in a large table to be horizontally partitioned into smaller tables.



Partitioned views allow the data in a large table to be horizontally partitioned into smaller *member tables*. Each member table has the same format as the original table, but only part of the rows. A server containing a member table is called a *member server*. Each member server contains one member table and a distributed partitioned view.

An application that references the partitioned view on any of the servers gets the same results as would be obtained if a complete copy of the original table were present on each server.

Local and Distributed Partitioned Views

You can implement partitioned views locally on a single server or in a distributed environment on multiple servers. *Local partitioned views* reference member tables on one server. *Distributed partitioned views* reference member tables on multiple servers. You will typically use distributed partitioned views.

In the illustration, the **Customer** table is partitioned by region. Region A is on member server 1, and Region B is on member server 2. A view is created on each server that makes it possible to view the partitioned data as if it were in one table. This view will appear as a virtual table rendition of the original table.

Features Necessary to Implement Partitioned Views

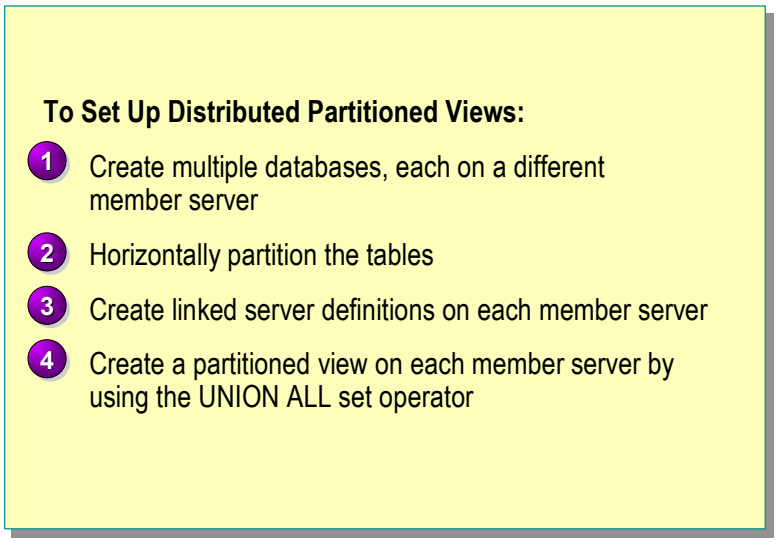
Certain SQL Server features are necessary to implement partitioned views. These features appear in the following table.

Feature	Benefit
Views	Allow user to see all of the partitioned tables as one table
CHECK constraints	Define and enforce the integrity of partitions
Distributed queries	Query and update partitioned data
INSTEAD OF triggers	Manage updates to views

Implementing Distributed Partitioned Views

Topic Objective
To list the steps necessary to implement distributed partitioned views.

Lead-in
Setting up distributed partitioned views requires four steps.



Setting up distributed partitioned views requires four steps.

1. Create multiple databases, each on a different member server running an instance of SQL Server.
2. Horizontally partition the tables by creating tables on each member server.
3. Create linked server definitions on each member server. The linked server definition will be used to send distributed queries to each member server.
4. Create a partitioned view on each member server by using the UNION ALL set operator to combine all of the rows from each member server table.

   Each view should have the same name. This allows queries referencing the distributed partitioned view name to run on any of the member servers.

Considerations for Partitioning Data

Topic Objective
To describe some of the considerations for partitioning data.

Lead-in
It is essential to have a good database design to partition data.

- **Design Considerations**
  - Partition data to keep related data on the same server
  - Minimize need to access data on other member servers
  - Place complete records on the same member server
  - Select the appropriate column to define the partition
- **Ways to Partition**
- **Rules For Partitioning**

Partitioning works well if the tables in the database are naturally divisible into similar partitions where most of the rows accessed by any SQL statement can be found on the same member server.

Design Considerations

Design considerations for partitioning include the following:

- Partition the individual tables in the original database so that most related data is placed together on a member server.
- Minimize any requirements for data on other member servers. Distributed queries should only be needed for 20 percent, or less, of the data.
- Place complete records on the same member server. A partition should allow all rows to be placed on the same member server as all their referencing foreign key rows.
- To evenly distribute the workload, you should define the partition on the column that most evenly distributes the data among the partitioned tables.

  For example, a primary key may be the best method to partition, wherein you specify a range of data for each table. In some cases it may be more beneficial to partition by a non-primary key column such as region.

Ways to Partition

You can use different methods of distributing data in various tables across all the member databases. You should consider:

- Partitioning some tables.
- Making complete copies of other tables in each member database.
- Leaving some tables intact on the original server.

Rules for Partitioning

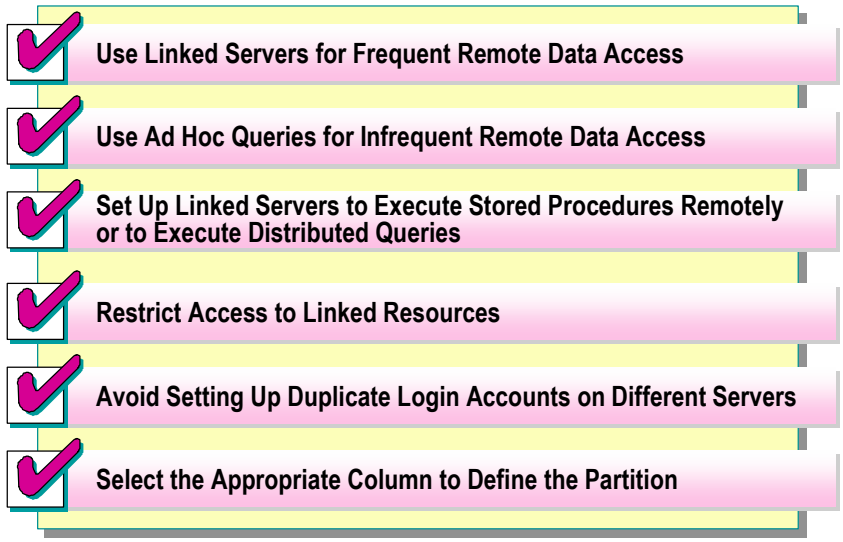
Some rules for partitioning are:

- Tables must have the same format as the original table. Tables must include the same number of columns, which must have the same attributes.
- Partition ranges cannot overlap.
- You must enforce the partitioned range of values on each member table through the use of a CHECK constraint.

Recommended Practices

Topic Objective
To summarize tips and techniques for optimizing programming multiple servers.

Lead-in
Use these recommended practices to optimize the process of programming multiple servers.



The following recommended practices should help you when you program multiple servers:

- Use linked servers when you expect to access remote data on a regular basis.
- Use ad hoc queries when you do not expect to access a data source repeatedly over time.
- Set up a linked server environment in which to execute either stored procedures remotely or to execute distributed queries.
- Restrict access to linked resources by using application roles or local accounts that are mapped to accounts on the linked server.
- Avoid setting up duplicate login accounts on different servers. Map user accounts on a server to a single account (with appropriate permissions) on the linked server to access data on the linked server.
- Select the appropriate column to define the partition.

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

Topic	Search on
OLE DB providers	"OLE DB providers tested with SQL Server"
Linked servers	**sp_addlinkedserver**
Linked server security	**sp_addlinkedsrvlogin**
Pass-through queries	OPENQUERY
Ad hoc queries	OPENROWSET
Allowed and disallowed Transact-SQL statements and actions	"accessing external data using distributed queries"
Distributed query restrictions	"external data and Transact-SQL"

Lab A: Using Distributed Data

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will set up linked servers and query remote data.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Set up a linked server and establish security.
- Query data on a linked server.
- Import data from a linked server.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2073A\Labfiles\L12.
- Answer files for this lab, which are located in C:\Moc\2073A\Labfiles\L12\Answers.

Lab Setup

To complete this lab, you must have either:

- Completed the prior lab, or
- Executed the C:\Moc\2073A\Batches\Restore12.cmd batch file.

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

For More Information

If you require help with executing files, search SQL Query Analyzer Help for "Execute a query".

Other resources that you can use include:

- The **Northwind** database schema.
- Microsoft SQL Server Books Online.

Scenario

The organization of the classroom is meant to simulate that of a worldwide trading firm named Northwind Traders. Its fictitious domain name is nwtraders.msft. The primary DNS server for nwtraders.msft is the instructor computer, which has an Internet Protocol (IP) address of 192.168.*x*.200 (where *x* is the assigned classroom number). The name of the instructor computer is London.

The following table provides the user name, computer name, and IP address for each student computer in the fictitious **nwtraders.msft** domain. Find the user name for your computer, and make a note of it.

User name	Computer name	IP address
SQLAdmin1	Vancouver	192.168.*x*.1
SQLAdmin2	Denver	192.168.*x*.2
SQLAdmin3	Perth	192.168.*x*.3
SQLAdmin4	Brisbane	192.168.*x*.4
SQLAdmin5	Lisbon	192.168.*x*.5
SQLAdmin6	Bonn	192.168.*x*.6
SQLAdmin7	Lima	192.168.*x*.7
SQLAdmin8	Santiago	192.168.*x*.8
SQLAdmin9	Bangalore	192.168.*x*.9
SQLAdmin10	Singapore	192.168.*x*.10
SQLAdmin11	Casablanca	192.168.*x*.11
SQLAdmin12	Tunis	192.168.*x*.12
SQLAdmin13	Acapulco	192.168.*x*.13
SQLAdmin14	Miami	192.168.*x*.14
SQLAdmin15	Auckland	192.168.*x*.15
SQLAdmin16	Suva	192.168.*x*.16
SQLAdmin17	Stockholm	192.168.*x*.17
SQLAdmin18	Moscow	192.168.*x*.18
SQLAdmin19	Caracas	192.168.*x*.19
SQLAdmin20	Montevideo	192.168.*x*.20
SQLAdmin21	Manila	192.168.*x*.21
SQLAdmin22	Tokyo	192.168.*x*.22
SQLAdmin23	Khartoum	192.168.*x*.23
SQLAdmin24	Nairobi	192.168.*x*.24

Estimated time to complete this lab: 60 minutes

Exercise 1
Setting Up Linked Servers

In this exercise, you will work with a partner. You will set up your local SQL Server as a remote data source for your partner. You will also set up a link to your partner's SQL Server and manage security for remote data access.

► To create and populate the ProductInfo table on your local server

In this procedure, you will create a new table in the **ClassNorthwind** database. Your partner will use this table.

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.

   Leave this query window open for the remainder of the exercise.

3. Open, review, and execute the Labfiles\L12\RemoteTbl.sql script. This file creates the **ProductInfo** table in the local **ClassNorthwind** database.

► To create login accounts for local and remote users

In this procedure, you will create two new login accounts: one for your partner to use when your partner links to your SQL Server and one for you to use when you access linked servers.

1. Open SQL Server Enterprise Manager.
2. Open the SQL Server Books Online topics "How to add a SQL Server login (Enterprise Manager)" and "How to grant a SQL Server login access to a database (Enterprise Manager)."
3. Use the procedures in SQL Server Books Online and the information in the following table to create a new login account that remote servers that access information on the local computer will use.

Option	Value
Login name	**NWInfoRemote**
Authentication	SQL Server
Password	**nwpassrmt**
Default database	**ClassNorthwind**
Database access	**ClassNorthwind**
Permit in database role	**public**

4. Create another login account with the following characteristics. This local account will be mapped to the **NWInfoRemote** login account on your partner's SQL Server.

Option	Value
Login name	**NWInfoLocal**
Authentication	SQL Server
Password	**nwpasslocal**
Default database	**ClassNorthwind**
Database access	**ClassNorthwind**
Permit in database role	**public**

► To grant SELECT and INSERT permissions on the ProductInfo table

In this procedure, you will grant SELECT and INSERT permissions on the **ProductInfo** table on your local computer running SQL Server to the **NWInfoRemote** login account. This allows remote users to read and insert data into the **ProductInfo** table.

1. Open the SQL Server Books Online topic "How to grant permissions on multiple objects to a user, group, or role (Enterprise Manager)."
2. Use the procedure in SQL Server Books Online to grant SELECT and INSERT permissions on the **ClassNorthwind.dbo.ProductInfo** table to the **NWInfoRemote** login account.

► To add the NWInfoLocal login account to the db_datareader role in ClassNorthwind

In this procedure, you will add the **NWInfoLocal** login account to the **db_datareader** role for the **ClassNorthwind** database. The **db_datareader** role has SELECT permission on all user tables in the database. This role will be used to access the local copy of **ClassNorthwind**.

1. Open the SQL Server Books Online topic, "How to add a member to a SQL Server database role (Enterprise Manager)."
2. Use the procedure in SQL Server Books Online to add the **NWInfoLocal** login account to the **db_datareader** role in the **ClassNorthwind** database.

► To set up a linked server

In this procedure, you will register your partner's computer as a linked server on your local server.

1. Switch to SQL Query Analyzer.
2. Open the Labfiles\L12\MakeLink.sql script.
3. Modify the script, substituting the name of your partner's SQL Server for ***servername***.
4. Execute the script.

► To establish security between your local computer and the linked SQL Server

In this procedure, you will map a login account on your local computer, running SQL Server, to a login account on your partner's computer running SQL Server.

1. Open the Labfiles\L12\MapToLnk.sql script by using SQL Query Analyzer.
2. Modify the script, substituting the name of your partner's SQL Server for ***servername***.
3. Execute the script.
4. Close SQL Query Analyzer.

Exercise 2
Querying Remote Data

In this exercise, you will write and execute queries that access data on the linked server that you set up in the previous exercise. Both you and your partner must complete Exercise 1 before you start this exercise.

► To access remote data on the linked server

- In this procedure, you will log in to your local SQL Server with security credentials for a local application. You will then write a simple query that returns data from the **ProductInfo** table in the **ClassNorthwind** database on your partner's computer. \L12\Answers\LnkSelect.sql is a completed script for this procedure.

► To access remote data by using a pass-through query

Because each linked server is using SQL Server Authentication, each server needs to be configured to allow both SQL Server and Windows Authentication.

1. Using SQL Server Enterprise Manager, expand the server group.
2. Right-click your server, and then click **Properties**.
3. Click the **Security** tab.
4. Under **Authentication**, click **SQL Server** and **Windows**.
5. Stop and restart SQL Server.

► To log on to your server by using SQL Server Authentication

1. On the **Start** menu, point to **Programs**, point to **Microsoft SQL Server**, and then click **Query Analyzer**. Log in to the (local) server with SQL Server Authentication. Connect as **NWInfoLocal** with a password of **nwpasslocal**—you will use this connection for all procedures in this exercise.
2. Write and execute a Transact-SQL statement that returns all columns from the **Northwind.dbo.ProductInfo** table on the linked server (your partner's computer).

```
SELECT * FROM <servername>.ClassNorthwind.dbo.ProductInfo
```


► To access remote data by using a pass-through query

In this procedure, you will write a query that uses the OPENQUERY function to return data from the **ProductInfo** table in the **ClassNorthwind** database on your partner's computer. The query will be processed remotely as a pass-through query. \L12\Answers\PassThru.sql is a completed script for this procedure.

- Write and execute a pass-through query that returns the **ProductID** and **Royalty** columns from the **Northwind.dbo.ProductInfo** table on the linked server (your partner's computer). Use the OPENQUERY function in the FROM clause of the SELECT statement.

```
SELECT * FROM
    OPENQUERY(<servername>,
            'SELECT ProductID, Royalty
             FROM ClassNorthwind.dbo. ProductInfo')
```

► To join local and remote tables

In this procedure, you will write a query that joins the local copy of **ClassNorthwind.dbo.Products** to the **ClassNorthwind.dbo.ProductInfo** table on your partner's computer. \L12\Answers\LnkJoin.sql is a completed script for this procedure.

- Write and execute a Transact-SQL statement that joins the local copy of **ClassNorthwind.dbo.Products** to the **ClassNorthwind.dbo.ProductInfo** table on the linked server. Join the tables on the **ProductID** column. Include the **ProductName**, **ProductID**, and **ImportTax** columns, and then order the results by **ProductName**.

```
USE ClassNorthwind

SELECT ProductName, Products.ProductID, Royalty, ImportTax
FROM Products JOIN
<servername>.ClassNorthwind.dbo.ProductInfo CNWR
    ON Products.ProductID = CNWR.ProductID
```

► To add the NWInfoLocal login account to the database owner role

In this procedure, you will add the **NWInfoLocal** login account to the **db_owner** role for the **ClassNorthwind** database. As a member of the **db_owner** role, the **NWInfoLocal** login account will have the ability to create new tables in the **ClassNorthwind** database.

1. Open the SQL Server Books Online topic, "How to add a member to a SQL Server database role (Enterprise Manager)."
2. Use the procedure in SQL Server Books Online to add the **NWInfoLocal** login account to the **db_owner** role for the **ClassNorthwind** database.

► To import data from a linked server and create a new local table

In this procedure, you will create a new table on your local server and populate it with the results of a query on your linked server. \L12\Answers\LnkImport.sql is a completed script for this procedure.

1. Switch to SQL Query Analyzer.
2. Write and execute a Transact-SQL statement that returns the **ProductID** and **Royalty** columns from the **ProductInfo** table on the linked server.
3. Modify the statement to create a new, local, and permanent table named **LocalProdInfo** that contains the results of the query.
4. Examine the contents of **LocalProdInfo**.

```
USE ClassNorthwind

SELECT ProductID,Royalty
FROM <servername>.ClassNorthwind.dbo.ProductInfo

SELECT ProductID,Royalty
INTO LocalProdInfo
FROM <servername>.ClassNorthwind.dbo.ProductInfo

SELECT * FROM LocalProdInfo
```

► To execute a stored procedure on a linked server

In this procedure, you will execute the **Sales by Year** stored procedure on the linked server. \L12\Answers \RmtProc.sql is a completed script for this procedure.

1. Write and execute a Transact-SQL statement that executes the **Sales by Year** stored procedure in the **ClassNorthwind** database on the linked server by using a fully qualified four-part name. Use the information in the following table to write this query.

Parameter	Value
Start_date	'1996'
End_date	'1997'

2. Close all open query windows, and then quit SQL Query Analyzer.

```
EXECUTE <servername>.ClassNorthwind.dbo.[Sales by Year]
'1996', '1997'
```

► To access remote data with an ad hoc query

In this procedure, you will write an ad hoc query that returns all columns from the **ClassNorthwind.dbo.ProductInfo** table on the remote server. Use the OPENROWSET function to connect to the remote data source. \L12\Answers\Adhoc.sql is a completed script for this procedure.

1. Open SQL Query Analyzer and, if prompted, log in to the (local) server with Windows Authentication.

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

   Because you are not using the **NWInfoLocal** user account, your partner's server cannot be used as a linked server directly from this connection.

2. Write and execute a Transact-SQL statement that returns the **ProductID** and **Royalty** columns from the **ProductInfo** table on your partner's computer. Use the OPENROWSET function to provide connection information. Use the information in the following table to write this query.

Parameter	Value
provider_name	'SQLOLEDB'
data_source	<Servername> (your partner's computer name)
User_id	**'nwinforemote'**
Password	**'nwpassrmt'**

```
SELECT * FROM
OPENROWSET('SQLOLEDB', '<servername>'; 'NWInfoRemote';
'nwpassrmt',
'SELECT ProductID, Royalty FROM
     ClassNorthwind.dbo.ProductInfo')
```

3. Close all connections to SQL Query Analyzer.

If Time Permits
Managing Distributed Transactions

In this exercise, you will create a stored procedure to use in distributed transactions. You will confirm that the MS DTC service has been started. You will write and execute Transact-SQL statements inside a distributed transaction to ensure tight data consistency between two SQL Servers.

► To create a stored procedure to use in distributed transactions on your local computer

In this procedure, you will log in to your local SQL Server as a member of the **sysadmin** role. You will review and execute a script that creates the **DeleteProductInfo** stored procedure and grants EXECUTE permission to the **NWInfoRemote** login account. Later, your partner will use this stored procedure to delete rows from the **ProductInfo** table on your local computer.

Important You and your partner must complete the preceding procedure before continuing with this exercise.

---

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

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

2. Open, review, and execute the Labfiles\L12\RemoteSP.sql script. This file creates the **DeleteProductInfo** stored procedure in the local **ClassNorthwind** database and grants EXECUTE permission to the NWInfoRemote user account. Your partner will use this stored procedure to delete rows from the **ProductInfo** table on your local computer.

3. Close SQL Query Analyzer.

► To start the MS DTC service

In this procedure, you will start the MS DTC service.

1. Open the SQL Server Service Manager.
2. Start the MS DTC service on your local computer if it is not running.

► To perform a distributed transaction between the local computer and a linked server

Suppose that ClassNorthwind Traders has two copies of its database, one at each of its warehouses. Any changes to one copy should be made to the other copy.

In this procedure, you will insert data into your local **ProductInfo** table as well as the **ProductInfo** table on your partner's computer as a single transaction. \L12\Answers \DistIns.sql is a completed script for this procedure.

1. On the **Start** menu, point to **Programs**, point to **Microsoft SQL Server**, and then click **Query Analyzer**. Log in to the (local) server with SQL Server Authentication. Connect as **NWInfoLocal** with a password of **nwpasslocal**
2. Write a Transact-SQL statement that inserts a new row into the **ProductInfo** table on your local computer. Select one row in the following table. Your partner should select the remaining row.

Productid	Royalty	Import tax
55	3	.09
55	3	.09

3. Write a Transact-SQL statement that inserts the row that you selected into the **ClassNorthwind.dbo.ProductInfo** table on the linked server (your partner's computer). Use a fully qualified four-part name.
4. Enclose the two statements in a BEGIN DISTRIBUTED TRANSACTION and COMMIT TRANSACTION block and set the XACT_ABORT session option.
5. Execute the transaction.

```
USE ClassNorthwind
SET XACT_ABORT ON
GO

BEGIN DISTRIBUTED TRANSACTION
   -- Insert to the liked server.
   INSERT INTO <servername>.ClassNorthwind.dbo.ProductInfo
    VALUES (55, 3, .09)
   INSERT INTO ClassNorthwind.dbo.ProductInfo
    VALUES (55, 3, .09)
COMMIT TRANSACTION
```

6. Write and execute queries to confirm that the added row appears in both tables.

```
SELECT * FROM <servername>.ClassNorthwind.dbo.ProductInfo
SELECT * FROM ClassNorthwind.dbo.ProductInfo
```

► To manage distributed data by using a stored procedure

In this procedure, you will examine and execute a script that deletes the rows previously added to the **ClassNorthwind.dbo.ProductInfo** tables on the local and linked servers.

1. Open and review the Labfiles\L12\DistDel.sql script by using SQL Query Analyzer. This script deletes the row that you added to the local **ProductInfo** table by using a Transact-SQL statement. A stored procedure deletes the row that you added to the **ProductInfo** table on the linked server. These actions are done as a transaction.
2. Modify the script so that your partner's computer serves as the linked server.
3. Modify the script to delete the **Product_id** that you added in the previous procedure.
4. Execute the modified Labfiles\L12\DistDel.sql script.
5. Examine the output to confirm that the rows were deleted.

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.

- Introduction to Distributed Queries
- Executing an Ad Hoc Query on a Remote Data Source
- Setting Up a Linked Server Environment
- Executing a Query on a Linked Server
- Executing a Stored Procedure on a Linked Server
- Managing Distributed Transactions
- Modifying Data on a Linked Server
- Using Partitioned Views

You own a mail order business with two warehouse locations, one in the United States and one in Asia. Each warehouse has a SQL Server that hosts a location-specific copy of the inventory database. The servers, named USsales and Asiasales, are connected by a WAN. Your business requires that these databases remain current and synchronized at all times.

1. What method of data distribution should you use to propagate changes from one server to the other?

   You should use distributed transactions to update both databases, because tight consistency is required. Do not use replication, because these databases must remain synchronized at all times.

2. Your office is based in the United States. Each morning, you generate a report that shows the number of units in stock for the ten most frequently sold items from each warehouse. How would you generate this report?

   You would register the Asiasales server as a linked server on the USsales computer and then write and execute queries on the USsales server, using fully qualified four-part names to access information on the linked server.

3. Because you often transfer inventory between warehouses, you must update the databases at both locations so that the number of stocked units is always current. How would you do this?

   You would write distributed queries or stored procedures to subtract and add inventory to each database. You then would write a query that encloses these queries, or calls to stored procedures, in a BEGIN DISTRIBUTED TRANSACTION and COMMIT TRANSACTION block. Using distributed transactions ensures that the databases are synchronized.

4. The Asiasales server also hosts a Microsoft Access database that contains sales summary information. Occasionally, you want to access this data to produce a report that includes the data from the USsales inventory database. How would you combine data from both sources in your report?

   You would write a query that accesses data in the Access database by using the OPENROWSET function. To generate the report, you would join tables in the Access database with tables in the local SQL Server database.