Chapter 1

Designing Data Access

MICROSOFT EXAM OBJECTIVES COVERED IN THIS CHAPTER:

- Design appropriate data access technologies.
- Design an appropriate data access object model.
- Design a cursor strategy for a data access component.
  - Decide when to use cursors.
  - Decide how to maximize cursor performance.
  - Detect which applications are using cursors and evaluate whether to remove them.
- Design caching strategies.
  - Select ADO.NET caching.
  - Design custom caching functionality.
  - Design a refresh strategy for cached data.
Database applications allow you to use the data you have stored in your database. They provide access to and manage your data. Therefore, it is essential to select the technology for accessing your data that will benefit your enterprise most.

In this chapter, we cover the technologies that will better serve your access needs so that you can develop efficient and manageable database applications. We begin by focusing on the methods available. Some of these methods are new to SQL Server 2005; in fact, using the same technologies as you have in the past might not provide you with the tools you need to create a well-managed data access environment. Thus, we next explain how you can use the data access methods available to your advantage in building a data access layer.

The internal structure of the SQL Server 2005 database engine has changed. Knowledge of the changes in the data models from previous versions of SQL Server to SQL Server 2005 will assist you not only in choosing the correct access method for your data but also in administering SQL Server 2005. Therefore, we specifically target these new features.

Because SQL Server is a relational data system that works in a set-oriented manner, you will need to access data on a row-by-row process. In order to process row-oriented data using cursors, we discuss how to implement cursors for data access. (See Chapter 4 for further discussion of cursors and cursor strategy).

Finally, we conclude this chapter with a focus on ways to improve data access performance. We discuss designing and implementing caching strategies. Although SQL Server 2005 manages caching mechanisms automatically, you need to understand the processes involved so that your data access application will work optimally.

Determining the Right Access Method

We start with a look at the current technologies available to you for accessing data. Then we cover the appropriate scope in which to use these technologies. In this discussion, you will learn the two types of database applications and the components used within the data access system and the architecture of those components.

We use the term component to reference a part or element of an overall solution.
Designing Appropriate Data Access Technologies

As a developer you know that your application design is all about accessing data efficiently and securely. Your applications come in many shapes and sizes. However, you have two distinct types of applications: those designed for managing data and those designed for administering systems.

You can define these two types of applications as follows:

- Applications that use data access components to obtain data and store it in the database
- Administrative tools that use well-defined object models to enable you to administer your database system

You require a data access system to create the link between your data and your application. All applications, regardless of the shape or size, require the same type of data access components. Simple applications, such as a client application querying your database, might involve nothing more than a command-line utility running on the client. For example, you might use the command-line utility SQL command (SQLCMD) and connect to your database server through a network library such as SQL Native Client (SQLNCLI). For more complex business applications, such as an accounting or customer management application, your need for data access components remains the same. You might build a data access interface for your client application, but you still need a data access provider such as SQLNCLI to be able to execute queries on your database.

You can classify data access components into two types (see Figure 1.1):

**Server-side components**  These components run on a server managing requests from client computers. Server-side components can include network libraries and TSQLEndpoints. TSQLEndpoints have been introduced in SQL Server 2005 as a new concept for SQL Server connections. You can grant, revoke, and deny permissions for TSQLEndpoints. By default, all users have permissions to access an endpoint unless the permissions are denied or revoked by a member of the sysadmin group or by the endpoint owner. SQL Server Setup creates TSQLEndpoints for all supported network protocols, as well as for the dedicated administrator connection. TSQLEndpoints created by SQL Server Setup include TSQLEndpoint LocalMachine, TSQLEndpoint Named Pipes, TSQLEndpoint Default TCP, TSQLEndpoint Default (Virtual Interface Adapter) VIA, and Dedicated Admin Connection (DAC).

For more information about network libraries and protocols, search “Network Protocols and Network Libraries” in SQL Server Books Online.

**Client-side components**  These components are the initiators as well as the receivers of requests between your database application and database server. Most often you use a presentation layer user interface (UI) component on the client for this effort.
Let’s take a minute to make certain you understand how database applications are built and where the data access system components fit in. A multitiered application, which is the recommended method, consists of the following:

• The presentation tier is the interface that interacts with the end user of the application.
• The business tier is the component that enforces the business rules and the workflow definitions.
• The data access tier converts the logical representation of your data into the physical schema of the database. This tier can also control the integrity of your data during the translation.

**Understanding How Database Components Interact**

It is important you understand how the database components interact with each other to create the connection between your application and the data source. The steps involved follow and are shown in Figure 1.2:

1. Your client application implements a data access layer. This layer will provide manageability for the connection.
2. The data access layer uses a data access application programming interface (API) to interact with the remote data. An API enables your application to exploit the power of operating system. The base services functions in the API give your application access to the resources of the computer and the features of the underlying operating system, such as memory, file systems, devices, processes, and threads. Your application uses these functions to manage and monitor the resources it needs to complete its work.
3. The API uses a specific data access provider component that will interact programmatical-
    ly with the API at the remote data source.

4. The data access provider interacts with the physical network using the various systems inter-
    connection layers to communicate with the remote data source. The data access provider
    will interact with the necessary network protocols, such as transactional and security, and
    use its data serialization formats to communicate with the data source.

5. Finally, the data source translates the request. The data source responds by executing the
    requested action and returning the result set to the calling application using the same
    channels and objects through which the request was sent.

**FIGURE 1.2** Interaction of database components

---

**Understanding the Architecture of Data Access Technologies**

Now we’ve covered how the components interact, we’ll look at the architecture of data access

technologies. Understanding the architecture will enable you to choose the appropriate tech-


tology to best meet your development needs.

You have three main players in the architecture of data access technologies:

**The database** The database is your resource. Here lies your data store and the server-side

    components that enable your connectivity.

**The provider** The provider is the communicator. Providers establish the communication

    connection between the client component and the data source.

**The client component** Client components are both the askers and the receivers. Client com-

    ponents are objects that provide the capability to transform the request, reach out, and inter-

    act with a database. You can use a variety of data access technologies to create your client-side
database application. Some unmanaged technologies include ODBC (Open Database Connectivity), OLE DB, ADO (Microsoft ActiveX Data Objects), and Jet. To base your database access application on managed code, you use a .NET Framework data access provider.

As we mentioned earlier, SQL Native Client (SQLNCLI) is the technology you should be using to access the new functionalities of SQL Server 2005. When you use SQL Native Client, you have no need to use an OLE DB provider, ODBC provider, or any other layer to interface with SQL Server 2005. SQL Native Client communicates directly with the SQL Server database. Since the SQLNCLI library connects to SQL Server through the server network library, which is accessed internally from a T-SQL endpoint, you are always assured of communicating with your SQL Server using data access providers. Furthermore, since this communication process is handled automatically, your database application does not need to be aware of or involved with its implementation.

SQL Native Client was designed to provide a simplified method of gaining native data access to SQL Server using either OLE DB or ODBC. It is simplified in that it combines OLE DB and ODBC technologies into one library, and it provides a way to innovate and evolve new data access features without changing the current Microsoft Data Access Components (MDAC).

Using Legacy Technologies to Access Data

It might be appropriate for you to use or continue to use an earlier technology to access your data source. We cover some legacy technologies and discuss their capabilities, level of performance, and limitations. We will include some technologies that are probably most familiar to you: DB-Library, ODBC, OLE DB, and SQLXML.

DB-Library

Consider the following when accessing data using DB-Library:

Deprecated feature DB-Library is a client interface that originated in Sybase and is not supported by SQL Server 7.0 and later. Some legacy database applications may still be using DB-Library. However, this library, based on C code implementations, is being deprecated and will be removed from future versions of SQL Server.

No support for current versions of SQL Server DB-Library does not support features of SQL Server 7.0 and later.

ODBC

Consider the following when accessing data using ODBC:

Industry standard Open Database Connectivity is well established as an industry standard for connecting to relational database systems. Microsoft created ODBC by adapting the SQL Access Group CLI and released ODBC 1.0 in September 1992. Because ODBC dates back more than 10 years, it offers connectivity to a wider variety of data sources than other data
access APIs. Many applications requiring high connectivity, including those serving SQL Server 2000, still use C language components with native ODBC.

**Driver availability** You implement ODBC API functions through database management system (DBMS)—specific drivers. The core library, independent of the applications and DBMS systems, is the interpreter between the applications and the database drivers. It is the database driver that contains the DBMS-specific details. Thus you can write applications that use standard types and features without concern for the specifics of each DBMS that the applications may encounter. To implement these database drivers you need only know how to attach to the core library, thus making ODBC modular.

**Data source name (DSN) connection definitions** Each ODBC data source on a client has a unique data source name (DSN). An ODBC application uses an ODBC data source to connect to an instance of Microsoft SQL Server. An ODBC data source is a stored definition that records the ODBC driver to use for connections specifying the data source—the information used by the ODBC driver to connect to a source of data and driver-specific options to be used for the connection. The connection definitions can be stored either in the registry as is the case of *System* and *User DSNs* or in the file system as is the case of the *File DSN*.

You launch the ODBC Data Source Administrator by clicking Data Sources (ODBC) in Control Panel.

---

**ODBC Data Source Name (DSN) Connection Definition Locations**

Connection definitions are stored as follows:

- **System DSNs** are stored in the `HKEY_LOCAL_MACHINE` (HKLM) portion of the registry and are available to all users of the computer.
- **User DSNs** are stored in the `HKEY_USERS` (HKU) portion of the registry and are available only to the user that created them.
- **File DSNs** are stored in the file system, which makes them available for copying to other computers or to be stored on a network drive.

---

**OLE DB**

Consider the following when accessing data using OLE DB:

**Added functionality over ODBC** OLE DB was designed as a higher-level replacement for, and successor to, ODBC, extending its feature set to support a wider variety of nonrelational databases.
Chapter 1 • Designing Data Access

Object model data providers  OLE DB separates the data store from the accessing application through a set of abstractions, such as connections, recordsets, and attributes. OLE DB is conceptually divided into consumers and providers. The consumers are the applications that need access to the data, and the provider is the software component that implements the interface. This technology enables a variety of database sources to be queried in a uniform manner.

Flexibility of data sources  The OLE DB provider is suitable for database applications that access both relational and nonrelational data sources. Using the Microsoft OLE DB Provider for ODBC allows you to use a single OLE DB provider to connect to multiple ODBC data sources, including SQL Server. However, connecting to SQL Server clients with this provider entails more administrative overhead than using the native Microsoft OLE DB Provider for SQL Server.

Although using SQLNCLI allows you to develop an OLE DB consumer optimized for SQL Server databases, you can only use SQLNCLI with SQL Server. You can use the Microsoft OLE DB Provider for ODBC to access data from a number of OLE DB–compliant ODBC applications.

SQLXML

Consider the following when accessing data using SQLXML:

Updated features  SQLXML has been updated to include the new features in SQL Server 2005. For example, earlier versions of SQLXML do not have support for data types introduced in SQL Server 2005. The initial versions of SQLXML enable XML support for SQL Server 2000, bridging the gap between XML and relational data. The updated version is SQLXML 4.0. When you install SQLXML 4.0, the files installed by earlier versions of SQLXML are not removed. Therefore, you can have Dynamic-link libraries (DLLs) for several different version-distinctive installations of SQLXML on your computer. It is possible for you to run the installations side by side. SQLXML 4.0 includes both version-independent and version-dependent programmatic identifiers (PROGIDs). You should use version-dependent PROGIDs in your database applications.

Search “What’s New in SQLXML 4.0” in SQL Server Books Online for more information.

Automatic installation  You automatically install SQLXML 4.0 when you install the Microsoft SQL Server 2005 server or tools with Notification Services. At installation, the components make appropriate changes to the registry.

SQLXML 4.0 is not completely backward compatible with SQLXML 3.0. Because of some bug fixes and other functional changes, particularly the removal of SQLXML ISAPI support, you cannot use Internet Information Services (IIS) virtual directories with SQLXML 4.0. Although most applications will run with minor modifications, you must test them before putting them into production with SQLXML 4.0.
Using SQL Native Client to Access Data

SQLNCLI is a data access provider specifically designed for and included with SQL Server 2005. Actually, SQL Server Management Studio and other administrative applications use SQL Native Client to communicate with SQL Server 2005.

You might be wondering at this point what the differences between SQLNCLI and MDAC are. Microsoft Data Access Components (MDAC) is a suite of data access APIs that ship with the Windows operating system. In many cases, an older operating system can be retrofitted with a newer version of MDAC by running the redistributable installer package. MDAC includes three major API surfaces—ODBC, OLEDB, and ADO—each of which has been optimized for a different set of applications. All three major components of MDAC employ the provider model, which allows access to a variety of data sources using a virtually identical set of programmatic calls. Although it is not a part of MDAC, SQL Native Client also exposes all three major component interfaces and is fully compatible with the latest version of MDAC. SQLNCLI and MDAC can cohabitate on the same system with no known issues.

SQL Native Client New Feature Support

SQL Native Client implements many features to take advantage of new functionalities provided by SQL Server 2005. A summary of the features follows:

- SQL Native Client supports mirrored databases, which is the ability to keep a copy, or mirror, of a SQL Server 2005 database on a standby server.
- SQL Native Client supports asynchronous operations, which is the ability to return immediately without blocking on the calling thread.
- SQL Native Client supports multiple active result sets (MARS), which is the ability to execute and receive multiple result sets using a single database connection.
- SQL Native Client supports the XML data type, which is a SQL Server 2005 XML-based data type that can be used as a column type, variable type, parameter type, or function return type.
- SQL Native Client supports user-defined types (UDT), which extends the SQL type system by allowing you to store objects and custom data structures in a SQL Server 2005 database.
- SQL Native Client supports large value data types, which are large object (LOB) data types that are supported in SQL Server 2005.
- SQL Native Client supports the handling of expired passwords so that passwords can now be changed on the client without administrator involvement.
- SQL Native Client supports the enhancement to row versioning that improves database performance by avoiding reader-writer blocking scenarios.
Chapter 1 • Designing Data Access

- SQL Native Client supports consumer notification on rowset modification.
- SQL Native Client supports bulk copy operations that allow the transfer of large amounts of data into or out of a SQL Server table or view.
- SQL Native Client supports the ability to encrypt data sent to the server without validating the certificate.

So, you see you have many reasons to upgrade your data access provider to SQLNCLI when using SQL Server 2005 as your data source.

You target SQLNCLI at two application types:

- Your existing applications that connect to SQL Server through OLE DB or ODBC and that you want to take advantage of any of the new SQL Server 2005 features.
- Your applications written in a development platform other than the Microsoft .NET Framework that you want to take advantage of any of the new SQL Server 2005 features.

If you do not need to use any of the new features of SQL Server 2005, you don’t need to use the SQL Native Client OLE DB provider, just continue using your current data access provider. However, if you are enhancing an existing application and you need to use the new features of SQL Server 2005, you should use SQLNCLI.

---

SQL Native Client Connection References

When you connect to SQL Server 2005 using SQL Native Client, your applications need to reference SQLNCLI as the provider name in the connection string as you open the connection to the database server. Connection strings are lists of keywords and associated values that identify a particular connection attribute. Depending on your programming environment, you may also need to include a reference the SQLNCLI header files. Presuming your server is installed on the C: drive, the header files are located at C:\Program Files\Microsoft SQL Server\90\SDK\Include folder and are named SQLNCLI.h.

When you are connecting through ODBC, do the following:

- Include the SQLNCLI header file in your ODBC application by replacing the odbcss.h file with the following:

  `Include "SQLNCLI.h";`

- If your application is using a DSN connection, update the DSN to use the SQL Native Client driver.
- If your application uses a connection string rather than a DSN connection, update the connection string from using the SQL Server provider to using the SQL Native Client provider:

  `"Driver={SQL Native Client}; Server=(local); Database=AdventureWorks; Trusted_Connection=yes;"`
When you are connecting through OLE DB, do the following:

- Include the SQLNCLI header file in your OLE DB application by replacing the sqloledb.h file as follows:
  ```c
  Include "SQLNCLI.h";
  ```

- Update the connection string from using the SQLOLEDB provider to use the SQLNCLI provider:
  ```sql
  "Provider=SQLNCLI;Server=(local);
  Database=AdventureWorks;Integrated Security=SSPI;"
  ```

Your OLE DB applications need to reference only sqlncli.h. However, if you have an application that uses both MDAC (OLE DB) and the SQL Native Client OLE DB provider, your application can reference both sqloledb.h and sqlncli.h. However, the reference to sqloledb.h must be first.

When you are connecting through ADO, do the following:

- Update the connection string from the SQLOLEDB provider to use the SQLNCLI provider:
  ```sql
  "Provider=SQLNCLI;Server=(local);
  Database=AdventureWorks;Integrated Security=SSPI;"
  ```

If you are developing a new application, Microsoft recommends that you consider using ADO.NET and the .NET Framework Data Provider for SQL Server instead of SQL Native Client to access all the new features of SQL Server 2005. For more information about the .NET Framework Data Provider for SQL Server, please see Chapter 2 or the .NET Framework SDK documentation for ADO.NET.

### Using ADO with SQL Native Client

If you have an existing application that uses ActiveX Data Objects (ADO) and you want to take advantage of the new functionality of SQL Server 2005, you should use SQL Native Client OLE DB as the data access provider. Enhancements have been made to the SQL Native Client OLE DB provider that enable ADO to use such SQL Server 2005 features as the XML data type, user-defined types, additional functionality of the VARCHAR, NVARCHAR, and VARBINARY data types and multiple active result sets (MARS).

To use the SQL Native Client OLE DB provider, define a new connection string keyword named DataTypeCompatibility. Then, set the DataTypeCompatibility keyword to 80 so that the new data types will map correctly to the ADO data types.
The changes to the connection string keyword are as follows:

- Provider=SQLNCLI
- DataTypeCompatibility=80

The following is an example of establishing an ADO connection string that is fully enabled to work with SQL Native Client, including the enabling of the MARS feature:

```vbnet
dim con as new adodb.connection

con.connectionstring = "provider=sqlncli;" 
& "server=(local);" 
& "database=AdventureWorks;" 
& "integrated security=sspi;" 
& "datatypecompatibility=80;" 
& "mars connection=true;"

con.open
```

**Using HTTP Endpoints and SOAP to Access Data**

You may want to access the data from SQL Server 2005 by using an application based on an Internet protocol. You could use many different tools, but two that have their functionality built into SQL Server 2005 are HTTP endpoints and Simple Object Access Protocol (SOAP).

For previous versions of SQL Server, your clients access a SQL Server database using Tabular Data Stream (TDS). TDS is a proprietary protocol that must be supported for Windows-based desktop clients. Sometimes, SQL Server clients must use Microsoft Data Access Components (MDAC). The MDAC stack is installed on the client computer that connects to SQL Server. For SQL Server, SQLXML 3.0 is a middle-tier component that supports web-based access to SQL Server, but IIS must also be used.

In SQL Server 2005, by including the use of HTTP and SOAP, native XML web services provide an alternative for other environments to access data, as shown in Figure 1.3.

**Figure 1.3** Web-based access to SQL Server
By using SOAP and HTTP, you can enable a wider range of clients to access SQL Server because you no longer need either MDAC installed at the client or SQLXML with its IIS dependency at the middle tier. These clients include web application clients that use existing client applications, such as web browsers. Furthermore, native XML web services make it easier to work with the Microsoft .NET Framework, Microsoft SOAP Toolkit, Perl, and other web development operating systems and tool sets. As you can see in our illustration, when you use HTTP or SOAP, you are making a direct call to SQL Server. Using this access method to return XML data is often more efficient than using SQLXML in the middle tier.

**Exposing SQL Server as a Web Service**

To expose a SQL Server as a web service, you must define an HTTP endpoint. An endpoint is your communication link between SQL Server and the client application. SQL Server HTTP endpoints are predefined within SQL Server to address data requests. As an object in SQL Server, the endpoint provides the necessary functions to provide SQL query responses within the security realm of SQL Server. The CREATE ENDPOINT statement is used to create both HTTP and TCP endpoints that are able to be used by Database Mirroring, Service Broker, SOAP, and T-SQL.

- The following example creates an endpoint called AWProduction, with two methods: GetProducts and UpdateProductPrice. These are the methods for which a client can send SOAP requests to the endpoint.

```sql
CREATE ENDPOINT AWProduction
STATE = STARTED
AS HTTP(
    SITE = 'localhost',
    PATH = '/AdventureWorks/Production',
    AUTHENTICATION = (INTEGRATED),
    PORTS = ( CLEAR ))
FOR SOAP(
    WEBMETHOD 'GetProducts' (name='AdventureWorks.Production.GetProducts',
        FORMAT=ROWSETS_ONLY),
    WEBMETHOD 'UpdateProductPrice' (name='AdventureWorks.Production.
        UpdateProductPrice'),
    WSDL = DEFAULT,
    DATABASE = 'AdventureWorks',
    NAMESPACE = 'http://AdventureWorks/
)
```

For the complete syntax of the CREATE ENDPOINT statement for Database Mirroring, Service Broker, SOAP, and T-SQL payloads, search “CREATE ENDPOINT (Transact-SQL)” in SQL Server Books Online.
Here are a few guidelines to follow when you are considering using Native XML web services for data access:

- Native XML web services are not recommended for the following:
  - Applications characterized by real-time highly concurrent access, with short duration transactions
  - Web farm-type scale-out
  - As a replacement for the middle tier, specifically where your application architecture has large-scale business logic demands that are better accommodated within middle-tier components
  - You should not enable basic or digest authentications unless you must impose access limitations on a database.
  - For access to sensitive data, it is preferable to adopt a Secure Sockets Layer (SSL) communication channel.
  - Do not use HTTP endpoints if you are building an intensive online transaction processing (OLTP) application or must manage large data values such as binary large objects (BLOBs)

### Writing Code That Uses HTTP Endpoints

The easiest way to write code that uses HTTP endpoints is by using a development environment such as Visual Studio 2005. In a development environment, all that you need to do is add a web reference to the endpoint. This creates a wrapper class, which contains one member with the function signature for each WEBMETHOD defined in the endpoint. With this class, you can access the endpoint in the same manner that you would with any other web service.

### Connecting SQL Server 2005 to Other Data Stores

Let’s now look at the times that you want to connect SQL Server to another data store. Your database application might use user-defined functions or store procedures to establish the connection. The queries you generate may execute wholly or partially at the remote source. For these scenarios, you must remain flexible in your approach to data access. Although OLE DB and ODBC libraries allow you to access other SQL Servers as well as other data sources, remember that you are limited by the functionality provided in the drivers themselves.

Also plan for security in your data access plan. If the remote server supports Windows Authentication, and security account delegation is available, you are able to use the same user credentials that authenticated against SQL Server to authenticate with the remote server. This delegation feature allows you to use a single credential for access to multiple data sources. Otherwise, you must create a linked server logon to map a SQL Server credential to the remote server credentials.

When your query executes against data from both remote and local sites, most often the processing for the remote data is handled at the remote server. If this process is not possible because of dialect or linked server setting difference, the remote server will return the entire data set to the local server where it will be processed with the local data. Even though local processing provides greater consistency, the cost of returning large amounts of unneeded data can be costly in terms of time and resource utilization.
Designing an Appropriate Data Access Object Model

When creating your applications for SQL Server 2005, you are programmatically administering SQL Server itself. Therefore, you need to understand the management APIs that SQL Server 2005 uses. In the next section, we analyze the changes within the management API of SQL Server so that you will be able to choose the correct object model to use for your applications.

Managing SQL Server before SQL Server 2005

In SQL Server 2000 you used the SQL Server Enterprise Manager that used the SQL Distributed Management Objects (SQL-DMO) API to administer SQL Server. The SQL-DMO API is still available with SQL Server 2005, although SQL-DMO will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work and plan to modify current applications that use this feature. In SQL Server 2000 you manage server objects using administrative tasks, evoking Data Definition Language (DDL) statements on databases and their objects and controlling services. By using SQL-DMO, your applications can administer SQL Server 2000 in the same manner that SQL Server Enterprise Manager does.

SQL-DMO objects are exposed as properties of other SQL-DMO objects. The relationship between objects is a tree-like structure that simplifies programming by using automation managers. Your objects can be referenced using the familiar dot notation used to reference properties or methods.

For example, in SQL-DMO the Database object exposes a Tables collection. Each Table object within the collection represents a single table of an instance of Microsoft SQL Server. You can obtain a SQL-DMO Table object by referencing a specific table using syntax much like the following:

Set oTable = oDatabase.Tables("Products")
Administering SQL Server 2005 Using SMO

The successor to DMO is SQL server management objects or SMO. The SMO object is a hierarchical structure of objects with the Server class being at the highest level in the SMO instance hierarchy. When you create a Server object variable, you are establishing a connection to SQL Server.

Another important top-level class with a separate hierarchy is the ManagedComputer class. You use the ManagedComputer object to view and modify SQL Server services, network settings, and alias settings. The ManagedComputer object is made available through the Windows Management Instrumentation (WMI) Provider. In addition to the Server and ManagedComputer objects, you have several utility classes that represent tasks or operations, such as Transfer, Backup, or Restore.

The SMO object model is composed of several namespaces. Each namespace represents a different area of functionality within SMO. Table 1.1 lists the SMO namespaces.

<table>
<thead>
<tr>
<th>Class</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft.SqlServer.Management.Smo</td>
<td>This namespace contains the instance classes, utility classes, and enumerations that you use to programmatically manage Microsoft SQL Server</td>
</tr>
<tr>
<td>Microsoft.SqlServer.Management.Common</td>
<td>This namespace contains the classes such as connection classes that are common to replication management objects (RMO) and SMO.</td>
</tr>
<tr>
<td>Microsoft.SqlServer.Management.Smo.Agent</td>
<td>This namespace contains classes corresponding to the SQL Server Agent</td>
</tr>
<tr>
<td>Microsoft.SqlServer.Management.Smo.Wmi</td>
<td>This namespace contains classes corresponding to the WMI Provider</td>
</tr>
<tr>
<td>Microsoft.SqlServer.Management.Smo.RegisteredServers</td>
<td>This namespace contains classes corresponding to Registered Server</td>
</tr>
</tbody>
</table>
The programming model of SMO has been improved over its predecessor DMO and the number of objects has grown to include all the new features in SQL Server. As you go through the following features, take a minute to reflect how each feature of SMO has brought about an increased functionality in SQL Server 2005.

You have many new features provided with SMO including the following:

- SMO gives you improved performance. Objects are loaded only when you specifically reference them. When you create an object, its properties are only partially loaded. The remaining objects are loaded when they are referenced directly.
- SMO provides capture and batched execution of Transact-SQL statements. Your network performance is improved because statements can be captured and sent as a batch.
- SMO enables you to manage your SQL Server services through the WMI Provider. SMO gives you the capability to programmatically start, pause, and stop all services on your server.
- SMO enables you to recreate any SQL Server object as well as its relationships to other objects through generated Transact-SQL scripts.
- SMO enables data transfer flexibility through its scripting functionality and use with Integration Services.
- SMO uses Uniform Resource Names (URNs).
- SMO’s event handling allows you to insert code that is executed when a specific event occurs.
- SMO’s exception processing allows you to identify errors specific to SMO.
- SMO gives you better control over your database objects and their security. Database objects are now managed objects and as such are controlled by their predetermined properties and methods.

Next we will look at how SMO represents some objects or properties in a number of features and components new to SQL Server 2005. These include the following:

- SMO provides support for partitioning tables and indexes on a partition scheme.
- SMO provides support for HTTP endpoints.
SMO provides support for new concurrency controls through snapshot isolation and row level versioning.

SMO provides support for the storage of XML data.

SMO provides support for creating read-only copies of databases.

SMO represents support for Service Broker communication.

SMO supports database object Synonyms.

SMO supports Database Mail.

SMO supports server registration and connection.

SMO supports SQL Server Profiler trace and replay events.

SMO supports Notification Services.

SMO supports security features in SQL Server 2005 through certificates and keys.

SMO supports DDL (Data Definition Language) triggers.

We cover how to use management object applications with the SMO object model in Chapter 2.

Administering SQL Server 2005 Replication Using RMO

You use replication management objects (RMO) to program SQL Server replication. SQL Server 2005 replication technologies have been enhanced in many areas. RMO supports all the new replication functionalities. The RMO object model extends and supersedes the replication-specific objects that you find in the DMO object model. You can use RMO with SQL Server version 7.0 and newer.


RMO allows you programmatic control of the following types of replication tasks:

**Configuring replication** You can use RMO to configure publishing and distribution, as well as to create, delete, and modify publications, articles, and subscriptions.

**Synchronizing subscriptions** You can use RMO to synchronize your subscriptions.

**Maintaining a replication topology** You can use RMO to perform replication maintenance tasks, such as managing partitions in a merge publication and validating subscriptions.

**Monitoring a replication topology** You can use RMO to programmatically monitor your replication topology.
Deciding When to Implement Cursors

We cover how to use management object applications with the RMO object model in Chapter 2.

Utilizing Patterns and Practices

Microsoft has done extensive work in the development of data access technologies. The Microsoft Patterns and Practices team provides application blocks that they have designed and developed. Application blocks contain tested code for a variety of technologies. You can customize these reusable code templates for your own applications, thus minimizing your need to write code.

You can find the Data Access Application Block link at the Microsoft Patterns and Practices team website at http://msdn.microsoft.com/practices.

Deciding When to Implement Cursors

Your SQL Server database is a relational system and hence operates in a set-oriented manner. However, many applications work with specific objects stored within the individual rows of the tables in a database. Your application needs to access the database one row at a time, which takes place by using row-by-row processing. SQL Server supports using cursors for this row-by-row processing.

As a developer or a database administrator, you need to understand cursors and their use. We discuss their implementation in the following sections and their use in Chapter 4.

Deciding When to Use Cursors

Here are a few reasons you might use a cursor strategy in your database application:

- You decide to access data by looping through the data row by row; cursors are a natural programming technique to use.
- You do not think another type of implementation other than using cursors is optimal.
- You find it too painful programmatically to solve the data access problem without using cursors.

For when not to use cursors, see Chapter 4.
Designing a Cursor Strategy for a Data Access Component

Microsoft SQL Server 2005 supports three cursor implementations:

**Transact-SQL cursors**  These cursors are based on the DECLARE CURSOR syntax. They are used mainly in Transact-SQL scripts, stored procedures, and triggers. They are implemented on the server and are managed by Transact-SQL statements sent from the client to the server.

**Application programming interface (API) server cursors**  These cursors support the API cursor functions in OLE DB and ODBC. API server cursors are implemented on the server. Each time a client application calls an API cursor function, the SQL Native Client OLE DB provider or ODBC driver transmits the request to the server for action against the API server cursor.

**Client cursors**  These cursors are implemented internally by the SQL Native Client ODBC driver and by the DLL that implements the ADO API. Client cursors are implemented by caching all the result set rows on the client. Each time a client application calls an API cursor function, the SQL Native Client ODBC driver or the ADO DLL performs the cursor operation on the result set rows cached on the client.

Because Transact-SQL cursors and API server cursors are implemented on the server, they are referred to collectively as **server cursors**.

Deciding How to Maximize Cursor Performance

You should be careful to not mix the use of these various types of cursors. If you execute a DECLARE CURSOR and OPEN statement from an application, you must first set the API cursor attributes to their defaults. If you set API cursor attributes to something other than their defaults and then execute a DECLARE CURSOR and OPEN statement, you are asking SQL Server to map an API cursor over a Transact-SQL cursor.

Using Server Cursors

A potential drawback of server cursors is that they currently do not support all Transact-SQL statements. Server cursors do not support Transact-SQL statements that generate multiple result sets; therefore, they cannot be used when the application executes a stored procedure or a batch that contains more than one SELECT statement. Server cursors also do not support SQL statements containing the keywords COMPUTE, COMPUTE BY, FOR BROWSE, or INTO.

However, using server cursors instead of client cursors does have some advantages:

**Performance**  If you are going to access a portion of the data in the cursor, using server cursors provides optimal performance since only fetched data is sent over the network. In this instance, a client cursor would be inappropriate since it would cache the entire result set on the client. Therefore, if you need to access just part of the query results from the cursor, you should use a server cursor so the processing will be done on the server and the network traffic will be minimized.
Additional cursor types  If the SQL Native Client ODBC driver used only client cursors, it could support only forward-only and static cursors. By using API server cursors, the driver can also support keyset-driven and dynamic cursors. SQL Server also supports the full range of cursor concurrency attributes only through server cursors. Client cursors are limited in the functionality they support.

More accurate positioned updates  Server cursors directly support positioned operations, such as the ODBC SQLSetPos function or UPDATE and DELETE statements with the WHERE CURRENT OF clause. However, client cursors simulate positioned cursor updates by generating a Transact-SQL searched UPDATE statement. This action may lead to inadvertent updates if more than one row matches the WHERE clause conditions of the UPDATE statement.

Memory usage  If you use server cursors, all cursor maintenance is done at the server. The client is spared the possibility of caching large amounts of data or of maintaining information about the cursor position.

Multiple active statements  When using server cursors, no results are left outstanding on the connection between cursor operations. This allows you to have multiple cursor-based statements active at the same time.

Now that you have had an introduction to server cursors, you need to keep in mind that the operation of all server cursors, except static or insensitive cursors, depends on the schema of the underlying tables. If you change the schema of a table after a cursor has been declared on it, you will get an error on any subsequent operation on that cursor.

Implementing Client-Side Cursors

Your implementation of client-side cursors depends on the data access provider you choose. In developing data access, you usually choose providers based on factors such as database server support, transaction support, security features, performance, and technical support. You should also consider client-side features such as connection pooling and client-side cursors.

As you saw earlier in this chapter, each data access provider supports a definite feature set. Some of the data access providers, such as OLE DB and ODBC, were designed to work with multiple data sources. Other providers, such as SqlClient and the SQL Native Client, were designed to work with a single specific data source providing native support for that data source.

When OLE DB, ODBC, and ADO technologies are implemented, a cursor is implicitly opened over the result set returned by a T-SQL statement. However, you can change this behavior by modifying the properties of the object that executes the T-SQL statement.
Let’s take another look at the client data access libraries, this time encapsulating the features of client-side cursors they support.

These are the features of client-side cursors that OLE DB supports:

- When you use OLE DB, the term *rowset* means a combination of the result set and its associated cursor behaviors.
- Native OLE DB does not support client-side cursors. However, you can add Microsoft Cursor Services for OLE DB to provide client-side cursors.

These are the features of client-side cursors that ODBC supports:

- When you use ODBC, the terms *result set* and *cursor* are interchangeable because a cursor is automatically mapped to a result set.
- ODBC implements client cursors through the ODBC Cursor Library.
- ODBC enables multiple active statements on a connection if used in conjunction with SQLNCLI.
- ODBC supports read-only and updatable cursor types.
- ODBC supports forward-only and scrollable cursor navigation.
- Using ODBC data access providers, you are able to configure and specify the cursor type, concurrency, and rowset size.

These are the features of client-side cursors that ADO supports:

- When you use ADO, the term *recordset* is a combination of a result set and its associated cursor behaviors.
- ADO supports only static read-only cursor types.
- ADO supports forward-only and scrollable cursor navigation.
- ADO supports asynchronous retrieval of results from the database server.

These are the features of client-side cursors that ADO.NET/SqlClient supports:

- ADO.NET implements classes that contain a separation between the result set (DataSet class) and the cursor (SqlDataReader and TableDataReader classes).
- ADO.NET supports only read-only, forward-only cursors.
- ADO.NET enables you to use multiple active statements on a connection.
- ADO.NET supports asynchronous retrieval of results from the database server.

## Detecting Which Applications Are Using Cursors and Evaluating Whether to Remove Them

In SQL Server 2005, you have many options available to you for tracking the status and usefulness of cursors.
Using SQL Server Profiler

SQL Server Profiler enables you to analyze your application’s cursor strategy. When you run a trace, you can capture the following cursor events by their start time, application name, user, client process ID, and server process ID:

**CursorOpen**  This event indicates when a cursor has been opened by a Transact-SQL statement.

**CursorExecute**  This event occurs when the Microsoft SQL Server 2005 database engine creates and populates a cursor from the execution plan created by a cursor prepare event.

**CursorImplicitConversion**  This event occurs when the Microsoft SQL Server 2005 database engine executes a Transact-SQL statement that is not supported by server cursors of the type requested. SQL Server database engine returns an error that indicates the cursor type has changed.

**CursorClose**  This event occurs when the Microsoft SQL Server 2005 database engine closes and deallocates a cursor.

**CursorPrepare**  This event occurs when the Microsoft SQL Server 2005 database engine compiles a SELECT statement associated with a cursor into an execution plan but does not create the cursor.

**CursorUnprepare**  This event occurs when the Microsoft SQL Server 2005 database engine discards an execution plan

**CursorRecompile**  This event occurs when the Microsoft SQL Server 2005 database engine recompiles a Transact-SQL cursor due to a schema change.

**Using Performance: System Monitor**

You are able to gather real-time metrics with System Monitor. You can use these metrics for a direct analysis of your cursor strategies or as a baseline to measure changes in cursor performance. Table 1.2 describes the SQL Server Cursor Manager by Type counters.

**Table 1.2**  Cursor Manager Counters in System Monitor

<table>
<thead>
<tr>
<th>Cursor Manager by Type Counters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active cursors</td>
<td>Number of active cursors.</td>
</tr>
<tr>
<td>Cache Hit Ratio</td>
<td>Ratio between cache hits and lookups.</td>
</tr>
<tr>
<td>Cached Cursor Counts</td>
<td>Number of cursors of a given type in the cache.</td>
</tr>
<tr>
<td>Cursor Cache Use Count/sec</td>
<td>Times each type of cached cursor has been used.</td>
</tr>
<tr>
<td>Cursor memory usage</td>
<td>Amount of memory consumed by cursors in kilobytes (KB).</td>
</tr>
</tbody>
</table>
Chapter 1 • Designing Data Access

**TABLE 1.2** Cursor Manager Counters in System Monitor *(continued)*

<table>
<thead>
<tr>
<th>Cursor Manager by Type Counters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cursor Requests/sec</td>
<td>Number of SQL cursor requests received by server.</td>
</tr>
<tr>
<td>Cursor worktable usage</td>
<td>Number of worktables used by cursors.</td>
</tr>
<tr>
<td>Number of active cursor plans</td>
<td>Number of cursor plans.</td>
</tr>
</tbody>
</table>

**Using Cursor Metadata**

You can use either of the following ways to get metadata describing a server cursor:

- Applications using API server cursors with a database API such as ADO, OLE DB, or ODBC typically use the cursor functionality of the API to get information about the state of the cursor.
- Transact-SQL scripts, stored procedures, and triggers can use the Transact-SQL functions and system stored procedures to get information about a Transact-SQL cursor.

**Using Stored Procedures**

Several system stored procedures report the characteristics of a server cursor:

- `sp_describe_cursor` You use this system stored procedure to return a cursor describing its global attributes, such as its scope, name, type, status, and the number of rows.
- `sp_describe_cursor_columns` You use this system stored procedure to return a cursor describing the attributes of each column, such as the column’s name, position, size, and data type.
- `sp_describe_cursor_tables` You use this system stored procedure to return a cursor describing the objects or base tables it references.
- `sp_cursor_list` You use this system stored procedure to return a listing of all the currently visible cursors for the connection.

**Using System Functions**

Finally, several system functions available report server cursor status information:

- `CURSOR_STATUS` You use this system function to indicate whether a cursor is open or closed, or whether a cursor variable is currently associated with a cursor.
- `@@FETCH_STATUS` You use this system function to indicate the success or failure of the last fetch operation performed for the connection.
- `@@CURSOR_ROWS` You use this system function to report the number of rows populated in the last cursor opened for the connection.
You should be very careful in how you use the analysis from stored procedures and functions. Their status information is often affected by other operations and other cursors. Their results may also be fleeting as in the results from the @@FETCH_STATUS function, which changes every time a FETCH statement is issued against any cursor open for the connection.

**Making the Cursored Choice**

When should you use cursors? Some developers would say never and manage to stay away from them whenever feasible (we believe we’ve heard them say, “Stay away from them like they’re the plague!”). Let’s take a moment to recap what we have seen thus far on the subject so you can make a well-educated decision for your strategy.

Using a cursor is less efficient than using a default result set. When you use a default result set you send only one packet from the client to the server—the packet containing the statement to execute. When you use a server cursor, each FETCH statement must be sent from the client to the server, where it must be parsed and compiled into an execution plan. It is easy to see the performance difference.

Recall that if a Transact-SQL statement will return a relatively small result set that can be cached in the memory available to the client application, and you know previous to execution that you must retrieve the entire result set, use a default result set. And finally, you should use server cursors only when cursor operations are necessary to support the functionality of the application; it is seldom “just a neat thing” to use cursors.

**Planning Caching**

As we complete our discussion of data access strategies, we must focus our attention on one last piece of the access structure—caching. Caching is the process of creating a persisted copy of data and objects local to the user.

Here are some questions to consider:

- Do you need to repeatedly access static data or data that changes rarely?
- Do you wish to minimize your expenses in data access retrieval?
- Does your data need to always be available, even when the source, such as a server, is not available?

Answering yes to each of the previous questions is a valid reason to implement a caching strategy.

**Designing Caching Strategies**

Although SQL Server 2005 provides many internal caching mechanisms, you can apply several available caching techniques to make your applications more robust. You can implement these
caching methodologies at each of the layers of your application to improve performance and scalability as well as to optimize system resources. Furthermore, network bandwidth, physical I/O, and memory can also profit from caching strategies.

**Tip**
Most often we think of caching in terms of data, but caching frequently used objects is often beneficial to performance.

You usually create database applications in layers. The layering process allows you to implement a variety of logic and scalability methodologies into your application. You can use numerous caching techniques at each layer. You can enhance each layer’s functionality by applying the correct caching techniques. Next we examine the considerations you need to take into account at various layers of your application.

### Caching Guidelines for Working with the Database Server Layer

Some guidelines for the database server layer follow:

**Don’t be greedy.** Read only the data you need when you query tables. When you read excessive amounts of data, existing data currently in the buffer cache will be forced out. The end result is that the data in cache could be useless to you and the data you need is gone.

**Avoid recompiling stored procedures.** When you compile stored procedures, the query plan is put into the procedure cache. One of the main reasons for creating stored procedures is to be able to reuse the query plan for faster execution on subsequent calls of the procedure. If you recompile the stored procedure, the cache is invalidated and a new query plan must be generated. This action can be very costly in time and resources.

**Monitor the buffer cache.** A variety of performance counters have been designed to assist you in monitoring the cache. One important counter is the SQL Server Buffer Manager: Page Life Expectancy counter. This counter states the number of seconds a page will stay in the buffer pool without references. In short, this is the life span of a page in the buffer cache. You want this number to be high. The higher the number, the longer data will reside in memory. This means that you will need fewer physical I/O operations to retrieve data.

**Denormalization is not always a bad word.** If your queries use complex joins or aggregations, you should consider denormalizing the data. If you are creating summary reports, denormalizing the underlying data will increase the performance of the query since it will need to read fewer pages of data. The fewer the pages, the fewer I/O operations will need to be performed, the faster the query.

**Warning**
You should not denormalize data that is updated often. The process of keeping the denormalized data in synchronization with the underlying normalized data might not make it worth the effort.
Caching Guidelines for Working with the Application (Middle) Layer

Some guidelines for the application layer follow:

**Avoid caching data that is updatable by other processes.** The data in the cache becomes invalid when it is updated by another application or process. To overcome this problem you might need to implement logic to resolve the update issues; this logic might cause as greater drain on performance.

**Use Microsoft ADO.NET connection pooling.** Connection pooling reduces the cost of opening and closing database connections. ADO.NET connection pooling is enabled by default, allowing you to reuse the database connection for subsequent data requests. This process can significantly improve the efficiency of your data access. Connection pooling requires that the pooled connections maintain the same parameters, including the security context.

**Think of creating a cache just for your business objects.** A specialized cache will enable a more efficient reuse of these business objects.

**Refrain from caching data that is sensitive in shared memory.** You tempt a security threat to this sensitive data by caching it in a shared memory location.

Caching Guidelines for Working with the Web Services Layer

Some guidelines for the web services layer follow:

**Cache partial results.** When you cache intermediate results you are decreasing the calls to the application tier. This action results in a faster response time by reducing the workload of the web service.

**Be aware of outdated data.** Any caching that is done at a middle-tier layer must be mindful of data changes at either the database or the application tier. You need to plan your caching strategy carefully so that you do not cause your data to be invalid.

Caching Guidelines for Working with the Front-end Application Layer

Some guidelines for the front-end application layer follow:

**Cache same-session data for reuse.** When you reuse data, even if the format at the client has changed, you should cache the data. You will use fewer system and network resources. You should design your application to cache the data at the application tier.

**Cache resource intensive objects with static data to disk.** For objects that are I/O and resource intensive, consider caching the objects to disk on the application’s operating system. If your data is not changing often, caching provides a means to quickly retrieve the objects and their interface components.

**Use time stamps to map disk-cached objects to their application retrieval time.** When you cache objects to disk, you can add a time stamp or a checksum to signify the currency of the data. Your application can use this information to retrieve data from the database based on this metric as opposed to retrieving the entire data set.
Cache dynamic user interface components in memory. Although you usually store interface components within the databases, you might consider caching them in memory. This action will save round-trips between the application and the database each time the user selects a new option.

Selecting ADO.NET Caching

ADO.NET is a cache-friendly database connection library that is designed for data reuse and database disconnection. ADO.NET uses a pooling technology to avoid repeatedly creating and disposing database connections when the database application is busy. This disconnected approach to data access provides better scalability for your application. ADO.NET leverages the ability of XML to provide disconnected access to data. ADO.NET was designed hand in hand with the XML classes in the .NET Framework. They are both components of the same architecture. Figure 1.4 shows the ADO.NET architecture.

Choosing a DataSet or a DataReader

The **DataSet** object is central to supporting disconnected, distributed data scenarios with ADO.NET. The **DataSet** is a memory-resident representation of data that provides a consistent relational programming model regardless of the data source. It is explicitly designed for data access independent of any data source. Therefore, you can use the **DataSet** with multiple data sources as well as a variety of data sources. **DataSets** do not require an active database connection to maintain a set of data in memory. The **DataSet** also represents a complete set of data. It includes the data as well as any related data, constraints, and relationships among the data. Figure 1.5 shows the **DataSet** object model.

**FIGURE 1.4** ADO.NET architecture
If you do not require the functionality provided by the DataSet, you can improve the performance of your application by using the **DataReader** to return your data in a forward-only, read-only fashion. By using the DataReader you can boost performance since you will be saving memory that would be consumed by the DataSet.

You need to consider the type of functionality required by your application when deciding between using a DataReader or a DataSet. You can use the DataSet to do the following:

- To cache data locally in your application so that you can manipulate it. On the other hand, if you need merely to read the results of a query, the DataReader is your better choice.
- To access remote data between tiers or from an XML web service.
- To interact with data dynamically. For example, you can use the DataSet to combine data from multiple sources or bind to a Windows Forms control.
- To execute extensive processing of data without requiring an open connection to the data source. This enables other clients and resources to use the connection.

**Using Connection Settings**

ADO.NET uses the ConnectionSettings class to represent the settings used by a connection to an instance of a server, such as Microsoft SQL Server. The ServerConnection class inherits the ConnectionSettings class.

Table 1.3 lists the properties exposed by the ConnectionSettings type.
### TABLE 1.3 ADO.NET ConnectionSettings Members’ Public Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationName</td>
<td>This property gets or sets the application’s name during its connection with the server.</td>
</tr>
<tr>
<td>ConnectAsUser</td>
<td>This property gets or sets the Boolean value indicating whether impersonation is being used when the connection is established.</td>
</tr>
<tr>
<td>ConnectAsUserName</td>
<td>This property gets or sets the username when impersonation is used for the connection.</td>
</tr>
<tr>
<td>ConnectAsUserPassword</td>
<td>This property gets or sets the user password when impersonation is used for the connection.</td>
</tr>
<tr>
<td>ConnectionString</td>
<td>This property gets or sets the data source–specific connection string used to establish the connection.</td>
</tr>
<tr>
<td>ConnectTimeout</td>
<td>This property gets or sets the time-out period in seconds of the connection.</td>
</tr>
<tr>
<td>DatabaseName</td>
<td>This property gets or sets the connection database.</td>
</tr>
<tr>
<td>EncryptConnection</td>
<td>This property gets or sets the Boolean value indicating whether the connection is encrypted.</td>
</tr>
<tr>
<td>Login</td>
<td>This property gets or sets the instance logon required to establish the server connection.</td>
</tr>
<tr>
<td>LoginSecure</td>
<td>This property gets or sets the Boolean value indicating a trusted or SQL Server authentication connection is established with the server.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>This property gets or sets a number indicating the maximum number of connections a connection pool accepts when establishing a connection with a server.</td>
</tr>
<tr>
<td>MinPoolSize</td>
<td>This property gets or sets a number indicating the minimum number of connections a connection pool accepts when establishing a connection with a server.</td>
</tr>
<tr>
<td>MultipleActiveResultSets</td>
<td>This property gets or sets the Boolean value indicating the permission of more than one active result set.</td>
</tr>
<tr>
<td>NetworkProtocol</td>
<td>This property gets or sets the server network protocol used to establish the connection.</td>
</tr>
</tbody>
</table>
As you have seen, ADO.NET provides reuse and disconnection functionality as you work with database connections. As noted above, connection pooling technology is also appropriate to avoid continually creating and disposing of database connections in highly active database application scenarios.

### Table 1.3 ADO.NET ConnectionSettings Members’ Public Properties (continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonPooledConnection</td>
<td>This property gets or sets the Boolean value indicating the pooling state of the connection.</td>
</tr>
<tr>
<td>PacketSize</td>
<td>This property gets or sets the network packets in bytes of data sent between the client and the server.</td>
</tr>
<tr>
<td>Password</td>
<td>This property gets or sets the password used with the logon to establish a SQL Server Authenticated connection.</td>
</tr>
<tr>
<td>PooledConnectionLifetime</td>
<td>This property gets or sets the pooled connection lifetime.</td>
</tr>
<tr>
<td>SecurePassword</td>
<td>This property gets or sets the secure password used with the logon to establish a SQL Server Authenticated connection.</td>
</tr>
<tr>
<td>ServerInstance</td>
<td>This property gets or sets the server name.</td>
</tr>
<tr>
<td>WorkstationId</td>
<td>This property gets or sets the workstation’s unique ID value.</td>
</tr>
</tbody>
</table>

As you have seen, ADO.NET provides reuse and disconnection functionality as you work with database connections. As noted above, connection pooling technology is also appropriate to avoid continually creating and disposing of database connections in highly active database application scenarios.

### Using Connection Pooling

Database connection pooling allows an application to reuse an existing connection from a pool instead of repeatedly establishing a new connection with the database. This technique can significantly increase the scalability of an application, because a limited number of database connections can serve a much larger number of clients. This technique also improves performance, because the significant time required establishing a new connection can be avoided.

Data access technologies such as ODBC and OLE DB provide forms of connection pooling, which are configurable to varying degrees. Both approaches are largely transparent to the database client application. OLE DB connection pooling is often referred to as session or resource pooling.

### Designing Custom Caching Functionality

As we mentioned earlier, the main cache memory pools of SQL Server 2005 are the procedure cache and the buffer cache. As a database developer, you are able to enhance your data access by controlling the contents of the caches. You want your applications to be scalable and to
perform well. You want to make certain that query plan recompiles are avoided and that you do not read large amounts of data unnecessarily. You need to monitor and strive to maximize cache reuse.

Some guidelines for maximizing cache utilization follow:

Avoid reading columns you are not using. You should always avoid returning data you don’t need. The extra data will be unnecessarily forced into memory, making memory unavailable for other tasks. Especially be careful that you do not return large and variable-width columns, such as VARCHAR and TEXT columns, that are not needed. These columns may reside outside the data page. Needless reading these large pieces of data can create an excessive burden on your cache.

Read only rows that need to be processed. We have mentioned this before. When you read excessive amounts of data, existing data currently in the buffer cache will be forced out. The end result is that the data in the cache is useless to you and the data you need is gone.

Avoid using nonparameterized SQL. Nonparameterized SQL may create new rather than reusable query plans.

Monitor procedures and attempt to avoid their recompilation. When your procedures recompile, you are not only creating extra resources to be applied but you are increasing the duration of the query process.

Use XML and relation data appropriately. If you need to read a single scalar value, you should most likely be using relational data. If you were to read the same from an XML column, the entire column would need to be loaded into memory prior to the read.

Use multiple SQL Server instances to isolate memory. You should use this process to adjust the memory on shared resources. Creating a new instance will isolate memory and may increase process utilization. However, other overhead also must be considered when creating a new instance, so weigh your options wisely.

Your caching scheme must include a plan to handle outdated data. If you do not update the cache when you have changes to your data, the data you retrieve might be inconsistent and yield inaccurate results. You must always give your users trustworthy data.

Designing a Refresh Strategy for Cached Data

As you have seen, caching is a broad topic. There are no concrete answers to what are the best techniques to employ for your caching implementation. However, you have tools available to assist you in monitoring your cache utilization. These tools also provide a basis for a refresh strategy that you can implement.
Using SQL Server Profiler

SQL Server Profiler provides metrics you can use to capture how the procedure cache is used in your queries. Here are a few to use in your trace:

- **SP:CacheHit**: Indicates when a procedure is found in the cache.
- **SP:CacheInsert**: Indicates when an item is inserted into the procedure cache.
- **SP:CacheMiss**: Indicates when a stored procedure is not found in the procedure cache.
- **SP:CacheRemove**: Indicates when an item is removed from the procedure cache.
- **SQL:StmtRecompile**: Occurs when a statement in a batch (including stored procedures) recompiles. This event should be used in SQL Server 2005 instead of **SP:Recompile** to trace statement-level recompilations.

Using Performance Monitor

You can use **Performance Monitor** to show current buffer cache activity. Here are two BufferManager counters that you will find useful:

- **SQLServer:BufferManager Buffer cache hit ratio**: Percentage of pages that were found in the buffer pool without having to incur a read from disk.
- **SQLServer:BufferManager Database pages**: Number of pages in the buffer pool with database content.

Using Dynamic Management Views and Functions

You can use **dynamic management views and functions** to return server state information that can be used to monitor the health of a server instance, diagnose problems, and tune performance. There are two types of dynamic management views and functions:

- Server-scoped dynamic management views and functions. These require VIEW SERVER STATE permission on the server.
- Database-scoped dynamic management views and functions. These require VIEW DATABASE STATE permission on the database.

Some dynamic management views and functions useful in returning caching information follow:

- **sys.dm_exec_cached_plans**: Returns information about the query execution plans that are cached by SQL Server for faster query execution.
- **sys.dm_exec_query_plan**: Returns the Showplan in XML format for a Transact-SQL batch whose query execution plan resides in the plan cache.
- **sys.dm_exec_query_stats**: Returns aggregate performance statistics for cached query plans. The view contains one row per query plan and the lifetime of the row is tied to the plan itself. When a plan is removed from the cache, the corresponding row is eliminated from this view.
- **sys.dm_exec_requests**: Returns information about each request that is executing within SQL Server.
Chapter 1 • Designing Data Access

Dynamic Management Views and Functions are new to SQL Server 2005. For more information on dynamic management views and functions and their use, search “Dynamic Management Views and Functions” in SQL Server Books Online.

Using Query Notifications

You have a new functionality available that allows an application to request a notification from SQL Server when the results of a query change. This new functionality is query notifications. Query notifications allow you to design applications that query the database only when there is a change to information that your application has previously retrieved.

Your applications can take advantage of query notifications to reduce round-trips to the database. In lieu of writing code that periodically reexecutes a query to maintain current results, you can now design applications that are automatically notified when the results for the query may be out of date. With query notifications, your application issues a command that contains a query and a request for notification. Your application then caches the results of the dynamic content generated from the query results. When your application receives the query notification, your application clears the cached content.

The notification messages are delivered by Service Broker. Therefore, Service Broker must be active in the database where the application requests the subscription. Please note though that query notifications functionality does not require or use Notification Services. Query notifications is independent of event notifications.

For more about query notifications, search “Working with Query Notifications” in SQL Server Books Online.

Summary

In this chapter, you started your journey with data access, a topic that lays the foundation for you as you program SQL Server 2005. You were taken through the paths of some of the older technologies that are still used. We evaluated those technologies expressly built for SQL Server 2005 data access. We discussed the various technologies and when to best apply them. At this point, you should certainly be excited at what is now available to you.

Since your SQL Server database is a relational system, we then turned to the topic of row-by-row access of the data, using cursors. You learned when to use cursors and how to use cursors to maximize performance. You further learned how to use tools so you can detect which applications are using cursors. We will continue our discussion on using cursors in Chapter 4.
Finally, we focused on optimizing your system resources by caching both data and objects in the appropriate layers. You learned how to reduce your resource utilization and gain system performance in data access by implementing a caching strategy. Furthermore, you also learned you can optimize resources such as memory, physical I/O, and network bandwidth by using caching methodologies.

Exam Essentials

Before you take the exam, you need to have a good understanding of the following:

Understand how to design data access technologies. You need to be familiar with the types of data access applications, data access components and their interaction, and the architecture of data access technologies.

Be able to determine an appropriate data access method. You need to be familiar with legacy and present-day access technologies for SQL Server 2005 data access. You need to know how and why to use SQL Native Client to access data. You need to understand the data access features of HTTP endpoints and SOAP. Furthermore, you need to be familiar with the data access models of DMO, SMO, and RMO as SQL Server management APIs.

Be able to decide when to use cursors. You need to be familiar with the types of cursor implementations in SQL Server 2005. You need to know the differences in implementing server and client cursors. You need to know the data access libraries and the cursor strategies they support.

Know how to detect cursor use. You need to be familiar with the tools used to track cursors and how to evaluate their metrics.

Be able to design a caching strategy. You need to understand and be able to apply the techniques for caching at the various layers of your applications.

Understand how to evaluate caching and designing a refresh strategy. You need to be familiar with the tools used to track caching and how to use them effectively.
Review Questions

1. Which of the following are database applications? (Choose all that apply.)
   A. Microsoft Windows Explorer
   B. Microsoft SQL Server Management Studio
   C. Microsoft Office Access
   D. Customer Relationship Management (CRM) software

2. Which of the following is a new technology library implemented by SQL Server 2005 to provide full access to all SQL Server databases including SQL Server 2005?
   A. SQLXML
   B. SQLNCLI
   C. SQLCMD
   D. SSMS

3. Which of the following are server-side components for data access in SQL Server 2005?
   A. Network libraries
   B. ODBC
   C. T-SQL endpoints
   D. OLE DB
   E. SQLNCLI

4. You are the database developer for your company. You have upgraded your servers to SQL Server 2005. You have created a new version of your SalesFocus application. You need to allow the new version to run from your remote offices. All new functionalities of SQL Server 2005 must be available through this connection. What should you do?
   A. Use ODBC to connect to SQL Server.
   B. Use SQLNCLI to connect to SQL Server.
   C. Create a web service on another server. Connect this server to SQL Server using an HTTP endpoint. Have the clients at the remote offices connect to the web service.
   D. Use SQLXML 4.0 to connect to SQL Server.
   E. Use OLE DB to connect to SQL Server.

5. You have upgraded your databases to SQL Server 2005. Now you want to update the connections in your SalesFocus application that is currently accessing SQL Server using OLE DB programmatically in the application. You need to update your code so that you can connect using SQLNCLI. What should you do? (Choose two.)
   A. Replace the OLE DB header file with the SQLNCLI header file.
   B. Add the SQLNCLI header file to the OLE DB header file.
   C. Update the DSN to use the SQL Native Client.
   D. Change the connection string to use the SQLNCLI provider.
6. Which components in the data access technology architecture act as a data transformer and local data depository?
   A. Providers
   B. Databases
   C. Client components
   D. Server endpoints

7. You are the database developer for your company. Your manager has told you to make certain that the DSN for your TaskForce ODBC application connects such that it is available to all users of each connecting computer. What should you do?
   A. You check the ODBC administrator in control panel to make certain the User DSN is being used for the TaskForce application.
   B. You check the ODBC administrator in control panel to make certain the System DSN is being used for the TaskForce application.
   C. You check the ODBC administrator in control panel to make certain the File DSN is being used for the TaskForce application.
   D. You check the HKU hive of the registry to make certain the User DSN is being used for the TaskForce application.

8. You are the database developer for your company. You need to create objects to expose SQL Server as a web service. Which of the following can you use?
   A. SSL communication channel
   B. HTTP endpoint
   C. IIS virtual directory
   D. TCP port

9. You are the database developer for your company. You have been told to develop a native XML web services strategy for data access. Which of the following guidelines are not recommended for XML web services? (Choose all that apply.)
   A. An intensive online transaction processing application.
   B. Applications that manage large values.
   C. Applications using short duration transactions with highly concurrent access.
   D. Applications that manage static data.

10. You are the database developer for your company. You want to process remote information prior to combining it with your local data sources. What type of provider should you use?
    A. An in-process data provider
    B. An ODBC data provider
    C. An OLE DB data provider
    D. An External Access data provider
11. You are the database developer for your company. You have been asked to manage your SQL Server 2005 server programmatically. What API should you use?
   - A. SMO
   - B. DMO
   - C. RMO
   - D. ADO

12. You are the database developer for your company. You have decided to use a cursor strategy in your SQL Server 2005 application. What cursor implementations are you able to use for SQL Server 2005? (Choose all that apply.)
   - A. API server cursors
   - B. Client cursors
   - C. Transact-SQL cursors
   - D. Web services cursors

13. You are the database developer for your company. You are trying to decide whether to implement server cursors or client cursors in your SQL Server 2005 application. Which of the following are advantages of using server cursors? (Choose all that apply.)
   - A. Performance
   - B. Memory usage
   - C. Multiple active statements
   - D. Choice of cursor types

14. You are the database developer for your company. You have decided to use the ODBC SQL Native Client ODBC driver with your SQL Server 2005 application. Which of the following is not a cursor type supported by the ODBC driver?
   - A. Static cursors
   - B. Dynamic cursors
   - C. Forward-only cursors
   - D. Backward compatible cursors
   - E. Keyset-driven cursors

15. You are the database developer for your company. You want to gather real-time metrics on the number of active cursors in your SQL Server 2005 server. Which tool should you use?
   - A. System Monitor
   - B. SQL Server Profiler
   - C. Dynamic Management Views
   - D. System stored procedures
16. You are the database developer for your company. You need to find the number of rows populated in the last cursor opened on the connection to your SQL Server 2005 server. Which function should you use?
   A. @@CURSOR_ROWS
   B. CURSOR_STATUS
   C. @@FETCH_STATUS
   D. @@ROWCOUNT

17. You are the database developer for your company. You want to enhance your data access by controlling the contents of the caches used by SQL Server 2005. What are the two cache pools you should be controlling?
   A. Procedure cache
   B. Disk cache
   C. Buffer cache
   D. I/O cache

18. You are the database developer for your company. You want to maximize your cache utilization. Which of the following are guidelines to follow? (Choose all that apply.)
   A. Read all rows from a table, even if all are not needed.
   B. Avoid recompiling procedures.
   C. Read only rows that need to be processed.
   D. Read all columns in a table.
   E. Read only columns that need to be processed.

19. You are the database developer for your company. Which dynamic management view or function would you use to return information about the query execution plans that are cached by SQL Server?
   A. sys.dm_exec_query_stats
   B. sys.dm_exec_cached_plans
   C. sys.dm_exec_query_plan
   D. sys.dm_exec_requests

20. You are the database developer for your company. You want to use a functionality of SQL Server 2005 that will allow you to design your application so that it will inform you when there is a change to information that your application has previously retrieved. What is this functionality?
   A. Event notification
   B. Query notifications
   C. Execution plan notification
   D. Cache notification
Chapter 1 • Designing Data Access

Answers to Review Questions

1. A, B, D. A database application provides a user interface to either manage or provide access to a database system. You use Windows Explorer to manage the data in files and directories in the Windows file system. Files and directories are considered data. Thus, the Windows file system is indeed a database management system, and Windows Explorer is a database application. You use SQL Server Management Studio to manage SQL Server database objects either through a graphical or programming environment. You use CRM to enable your organization to manage its customer database through the introduction of reliable systems, processes and procedures. You use Microsoft Office Access as a client application that accesses data in a file through a Jet interface. Access is a client application but does not provide access to a database system and therefore is not a database application.

2. B. SQL Native Client (SQLNCLI) was designed to provide a simplified method of gaining native data access to SQL Server using either OLE DB or ODBC. It is simplified in that it combines OLE DB and ODBC technologies into one library, and it provides a way to innovate and evolve new data access features used in SQL Server 2005. SQLXML is being replaced by the functionality within the HTTP endpoints and new XML support in SQL Server 2005. SQLCMD is the new command-line utility in SQL Server 2005. SSMS stands for SQL Server Management Studio.

3. A, C. Server-side components run on the server managing requests from client computers. Server-side components can include network libraries and TSQL endpoints. TSQL endpoints have been introduced in SQL Server 2005 as a new concept for SQL Server connections. ODBC, OLE DB, and SQLNCLI are client-side data access technologies.

4. B. You should use SQLNCLI to connect to SQL Server 2005. SQLNCLI provides a way to innovate and evolve the new data access features used in SQL Server 2005. Although ODBC and OLE DB can be used to connect to SQL Server 2005, new functionalities will not be available. There is no mention that your application uses XML, so SQLXML 4.0 is not appropriate. Likewise creating the web service is overkill and will not provide the full functionality you are seeking.

5. A, D. You must replace the header file for OLE DB with that of SQLNCLI. You should not add the new header file. You must also change the connection string to the new provider, SQLNCLI. OLE DB does not use DSNs.

6. C. Client components enable your application to connect and interact with your data source. As a part of this process, the client components need to transform data and collect it locally in cache. Databases are your data sources. Providers supply the communication between the client and database server. Server endpoints are not a component in the data access technology architecture.

7. B. To allow all users of the computer to have access to the DSN connection, you must make certain that the DSN is a system DSN. You can check this in the ODBC administrator in the control panel of the client computer or in the HKLM portion of the registry of that computer. References to the User DSN or File DSN are incorrect.

8. B. To expose SQL Server as a web service, you need to create an HTTP endpoint. The endpoint creates your communication link between SQL Server and the client application.
9. A, B, C. Native XML web services is not recommended for applications characterized by real-time highly concurrent access, with short duration transactions. Native XML is also not recommended for web farm-type scale-out or as a replacement for the middle tier, specifically where your application architecture has large-scale business logic demands that are better accommodated within middle-tier components. Native XML does provide fast access to data in applications that manage static data.

10. A. You would use managed code to connect to an external database in this situation. It might be useful to write a SQL CLR stored procedure with permission set to EXTERNAL_ACCESS. By using the SQL CLR procedure, you are able to define your connections to the external data source using ADO.NET, then transform and process the data. Finally, you are able to merge it with your local data.

11. A. You should use SMO. SMO, the successor to DMO, is the API that is used for SQL Server 2005. SQL server management objects, or SMO, is a programming model that encompasses the new feature of all objects created with SQL Server 2005. RMO, replication management objects, allows you to programmatically control replication tasks.

12. A, B, C. API server cursors and Transact-SQL cursors are implemented on the server. They are often referenced collectively as server cursors. Client cursors are implemented internally by the SQL Native Client driver. Web services does not use cursors.

13. A, B, C, D. Server cursors provide optimal performance if you are accessing only a portion of the data in the cursor. Server cursors reduce caching at the client. Server cursors fully handle multiple active statements leaving no results outstanding. SQL Native Client supports the full range of cursor types, whereas client cursors are restricted to the functionality they support. These are “clinically” listed advantages, but you need to always know your data, its use, and its environment.

14. D. ODBC defines four cursor types supported by Microsoft SQL Server and the SQL Native Client ODBC driver: static cursors, dynamic cursors, forward-only cursors, and keyset-driven cursors. Backward compatible is not a cursor type.

15. A. You would use the Cursor Manager Active cursors object in System Monitor. System Monitor allows you to gather real-time metrics.

16. A. @@CURSOR_ROWS returns the number of qualifying rows currently in the last cursor opened on the connection. To improve performance and since Microsoft SQL Server can populate large keyset and static cursors asynchronously, @@CURSOR_ROWS can be called to determine that the number of the rows that qualify for a cursor to be retrieved.

17. A, C. SQL Server uses two main cache memory pools, the procedure cache and the buffer cache.

18. B, C, E. You should avoid reading anything you are not processing. You should also monitor your procedures and attempt to avoid their recompiling whenever possible.

19. B. The sys.dm_exec_cached_plans returns information regarding the query execution plans cached by SQL Server.

20. B. Query notifications enables your application to request a notification from SQL Server when the results of a query change. It uses the Service Broker service to allow you to design applications that are automatically notified when the results for the query may be out of date.