CHAPTER

Database Architecture and Storage Engine Features

IN THIS CHAPTER:
New Hardware Support
SQL Server Engine
Security
Building on the same essential architecture that was established with the SQL Server 7 release, Microsoft has continued to make evolutionary improvements in the SQL Server engine. Following Moore’s Law, hardware has continued to double in capacity every 18 months, while at the same time hardware costs have continued to decline. However, while hardware has been getting cheaper, people costs have risen. In the SQL Server 2005 release, Microsoft has made enhancements that enable SQL Server to take advantage of the newest generations of high-performance hardware, providing it with the capability to scale to the utmost peak of the enterprise. At the same time, the company has added features that make SQL Server easier to manage and also provided new capabilities that enable you to get more value out of SQL Server for your organization. In this chapter, I’ll introduce you to some of the most important architectural improvements that Microsoft has made to SQL Server 2005, with the goal of helping you to see how these features can improve your productivity and the reliability of your database environment.

New Hardware Support

One of the key new features that enables SQL Server 2005 as a database engine to achieve the same performance levels as the biggest UNIX databases is its improved hardware support. Memory has always been one of the most critical factors that contribute to database performance, and the large UNIX databases that topped the non-clustered TPC-C performance rankings all attained that level of performance using a 64-bit hardware and operating system platform. When a comparable 64-bit platform became available for SQL Server in the form of the 64-bit Intel Itanium2 processor and Windows Server 2003’s native 64-bit support, SQL Server immediately jumped to the top of TPC-C benchmarks. SQL Server 2005 inherits that native 64-bit support from Windows Server 2003 along with the capability to support Non-Uniform Memory Architecture (NUMA). Building on the Windows Server 2003 platform, SQL Server 2005 provides 64-bit support for both the Itanium IA-64 architecture and the AMD x64 architecture.

Native 64-Bit Support

Opteron and Athlon 64 processors as well as Intel’s Xeon with Intel Extended Memory 64 Technology (EMT64). SQL Server 2005 fully supports both 32-bit and 64-bit hardware platforms for all of its major services, including the SQL Server Engine, Analysis Services, SQL Agent, and Reporting Services. The real advantage in moving a database to the 64-bit platform isn’t faster processing power. Rather, the real advantage lies in the vastly increased addressable memory. Figure 1-1 shows a comparison of the maximum addressable memory for the 32-bit SQL Server and the 64-bit version of SQL Server 2005.

The native 32-bit architecture is limited to a maximum of 4GB of addressable memory. Under Windows, this 4GB limit is divided evenly between the operating system and the applications. In other words, 2GB is reserved for the Windows operating system, leaving the remaining 2GB for applications. Using the Advanced Windowing Extensions (AWE) support found in the 32-bit version of Windows, the 32-bit version of SQL Server can address a maximum of 32GB of RAM. While this increase is substantial, there was still the overhead of paging to get to the appropriate memory pages. The native 64-bit implementation virtually eliminates the memory constraint by raising the maximum addressable memory to 32TB. Currently no production system supports anywhere near this amount of physical RAM. The maximum amount of physical RAM on which SQL Server 2005 has been tested at the time of release is 512GB. With the upcoming release of the next service pack for Windows Server 2003, the maximum supported memory is expected to jump to 1TB.
NUMA Support

Another new feature that leverages the Windows Server 2003 operating system is Non-Uniform Memory Architecture (NUMA) support. NUMA is a system architecture used by some system manufacturers such as IBM and Unisys that manages CPU and memory usage in multiprocessor systems more efficiently than the typical SMP architecture. In standard SMP systems, as the speed and number of processors increase, the competition between the processors for access to memory creates bus contention. This injects delays in the processing and hinders the capability of a system to scale. The result of this is that SMP systems don’t scale linearly as the number of processors increases. The NUMA architecture is designed to solve this problem found in SMP systems, in which the processors are able to access the data in RAM faster than the RAM and system bus can provide it. The NUMA architecture groups the CPU and memory into local pods of multiple processors. These pods are connected to each other via an external bus that transports cross-pod data traffic. This pod arrangement addresses the contention issue by limiting the number of CPUs competing for access to memory. To realize the maximum benefits from this architecture, both the operating system and the applications must be designed to minimize cross-pod data traffic and maximize the faster intra-pod memory access. If the operating system and the application are designed correctly, the NUMA architecture enables nearly linear scaling as more processors are added. Windows Server 2003 and SQL Server 2005 incorporate architecture improvements to increase the degree to which threads and the memory they use are located in the same pod.

Support for Hyper-Threading

SQL Server 2005 can also take advantage of hyper-threading, thanks in part to the new support for hyper-threading that Microsoft has added to Windows Server 2003. Hyper-threading is a CPU technology developed by Intel that creates two logical processors for each physical processor in a system. Each logical processor is capable of simultaneously executing separate threads. The goal of hyper-threading is to provide better resource consumption for multithreaded applications or multiple applications running on a single machine. While hyper-threading provides the potential of increasing the throughput on a server, the logical processors do compete for system resources such as the data in the processor’s cache. This competition for resources prevents a hyper-threaded CPU from performing as well as a comparable system that is using two physical CPUs.

SQL Server 2005 gets two main benefits from Windows Server 2003’s support for hyper-threading. First, unlike Windows 2000, Windows Server 2003 counts only
physical processors for licensing purposes. Thus, a single processor using hyper-threading is licensed as a single processor rather than a dual processor, as it would be under Windows 2000. Next, Windows Server 2003 provides improved thread scheduling for hyper-threaded systems. These changes result in better performance for multithreaded applications like SQL Server.

SQL Server Engine

While the first section of this chapter looked at the bigger picture of new hardware support provided in SQL Server 2005, this next section will drill down into some the most important improvements that Microsoft has made to the SQL Server engine itself.

.NET Framework Integration

The most significant SQL Server engine enhancement in SQL Server 2005 is the integration of the Microsoft .NET Framework. The integration of the .NET Framework with SQL Server extends the capability of SQL Server 2005 by enabling the development of stored procedures, user-defined functions, triggers, aggregates, and user-defined types using any of the .NET languages, such as VB.NET, C#, or J#. The integration of the .NET CLR with SQL Server 2005 is more than just skin deep, as the SQL Server database engine hosts the CLR in-process. You can learn more about the integration of the .NET Framework with SQL Server 2005 in Chapter 4.

Enhanced Multiple Instance Support

Another important enhancement found in the Enterprise Edition of SQL Server 2005 is support for up to 50 instances. This is up from the maximum of 16 instances that was supported in SQL Server 2000. This is a particularly significant improvement for hosting vendors who lease out multiple SQL Server services as part of their web services offerings.

New Data Types

SQL Server 2005 also supports several new data types. While the integration of the .NET Framework enables support for user-defined types, SQL Server 2005 also provides a couple of other new native data types: the varbinary(max) and XML data types. The varbinary(max) data type provides a new method for using LOBs with SQL Server. Unlike the older Image and Text data types, the new varbinary(max)
data type can be used as a variable, and programmatically it can be treated like the smaller data types, allowing for easier and more consistent usage scenarios.

The new XML data type, based on the varbinary(max) data type, enables you to store XML documents in the database. However, unlike the varbinary(max) data type, which is essentially data agnostic, the new XML data type is designed expressly for XML data and supports schema verification of XML documents. You can learn more about the new data types supported by SQL Server 2005 in Chapter 7.

Database Snapshots and Mirroring

Database Mirroring protects against database failure by giving SQL Server 2005 an instant database standby capability. Database Mirroring is a database-level availability technology that works with all of the standard hardware supported by SQL Server. There’s no need for any shared storage between the primary server and the mirrored server, and there are no distance limitations. Database Mirroring works by sending transaction logs between the primary server and the mirroring server, basically making the new Database Mirroring feature a real-time log shipping application. Database Mirroring can be set up with a single database, or it can be set up for multiple databases.

Database Snapshots provide a read-only snapshot of a database at a specific point in time. Database Snapshots are best suited for creating copies of a database for reporting or for creating backup copies of a database that you can use to roll back a production database to a prior state. Database Snapshots can be combined with Database Mirroring to create a reporting server based on the data that’s on the mirrored server. The data on the mirrored server can’t be accessed directly because the mirroring server is always in recovery mode. However, you can create a Database Snapshot on the mirrored database and then access the database view for reporting. You can learn more about database views and mirroring in Chapter 3.

Native HTTP Support

One of the other significant improvements to the SQL Server database engine is the addition of native HTTP support to the engine itself. The capability of SQL Server to process incoming HTTP requests enables SQL Server to provide SQL statement execution and stored procedure invocation via the SOAP protocol. This means that SQL Server 2005 is able to process incoming web service requests without the presence of IIS or another web server. The new HTTP support gives SQL Server native HTTP listening capabilities, including the capability to support HTTP endpoints specifying the URL, port, and requests that will be supported. SQL Server is also able to publish web services as Web Services Description Language
(WSDL) for endpoints. SQL Server’s HTTP support is standards-compliant, supporting
SOAP 1.0 and 1.2, as well as WSDL 1.1. The new native HTTP support feature also
supports both Windows and SQL Server authentication, as well as SSL. To enable
this new feature to have greater compatibility with middle-tier programming, stored
procedures can return result sets as an ADO.NET DataSet.

Server Events and DDL Triggers

SQL Server 2005’s new Server Event and DDL Triggers features enable you to
programmatically respond to changes in the system. While both of these new
features can accomplish similar functions, they are implemented quite differently.
Like standard DML triggers, DDL triggers are synchronous events that execute
stored procedures. You learn more about DDL triggers in Chapter 4.

In contrast, server events are asynchronous. In the server event model, the server
posts an event to a SQL Broker Service, and then a consumer can independently
retrieve that event. The event itself is recorded as XML data. There is no way to roll
back an event, and an event can be ignored if no consumer retrieves it. When the
event occurs, the system event is fired, which can notify you of the event or optionally
execute a code routine. The following example illustrates the syntax used to set up
an event notification:

CREATE EVENT NOTIFICATION MyDDLEvents
ON SERVER FOR DDL_STATEMENTS TO SERVICE MyDDL_log

This example creates a new event, names the event notification MyDDLEvents, and
attaches the event to the DDL statement. The TO SERVICE clause specifies that the
SQL Broker Service named MyDDL_log will be the recipient of the events. You can
find out more about the SQL Service Broker in Chapter 6.

Database Data File Enhancements

SQL Server 2005 now supports the ability to change the path of a database’s data
and log files using the ALTER DATABASE command. SQL Server 2000 provided
the ability to move the files for the tempdb database, but this wasn’t allowed for any
other database. As you might expect, SQL Server 2005 supports moving the files
only as an offline operation. The following example illustrates the new ALTER
DATABASE statements syntax:

ALTER DATABASE <database_name>
MODIFY FILE(name=<'data_file_name'>, filename=<'new path'>)
Data Partitioning

Another new enhancement that’s found in SQL Server 2005 is the ability to perform data partitioning. *Data partitioning* allows you to break a single database object such as a table or an index into multiple pieces. The new data partitioning feature facilitates the management of very large tables and indexes. The partitioning is transparent to the applications, which see only the database object itself and are unaware of multipart underlying storage, which is managed by SQL Server. Partitions can be created and dropped without affecting the availability of the database object itself. Essentially, partitioning enables you to split the underlying data store into multiple objects while still presenting a unified view of the object and all its partitions to an application. Figure 1-2 presents a basic overview of partitioning.

SQL Server 2005 supports data partitioning for tables, indexes, and indexed views. The row is the basic unit of partitioning. The partitions can be created according to values found in the columns in a row. This is known as *horizontal partitioning*. For instance, a table might be partitioned by date, where a different partition is created for each year. This type of partitioning by date enables you to perform a sliding date window type of processing, where you can drop the partition containing data from last year and not affect access to data contained in the current year’s partition.

Data partitioning provides a couple of important benefits for very large databases (VLDBs). Data partitions can facilitate data management, enabling you to selectively back up only specified partitions. For example, in the case of a large table that is partitioned by date you may want to back up only the current year, not last year’s partition. Another advantage is that in multiprocessor systems, you can devote a CPU to processing its own partition for improved throughput.

There are two basic steps to implementing data partitioning. First, you need to determine exactly how you want to partition a given object. Second, you need to assign each partition to a physical storage location. The different partitions can all be assigned to a single filegroup or different partitions can be mapped to multiple filegroups.

![Data Partitioning](image)

**Figure 1-2** Data partitioning
The following example shows the syntax for creating a simple partition function and scheme that will partition a table using a Range partition:

```sql
CREATE PARTITION FUNCTION MyPF
(int) AS RANGE LEFT FOR VALUES (50, 100)
GO
CREATE PARTITION SCHEME MyPS
AS PARTITION MyPF TO (FileGroup1)
GO
CREATE TABLE MyTable (col1 int, col2 varchar(50))
ON MyPS(col1)
GO
```

The first line creates a partition function named MyPF. The (int) shows that the partitioning will be performed on a column that’s defined using the int data type. The keyword RANGE specifies that Range partitioning will be used. The LEFT keyword controls which partition will receive borderline values. The value of LEFT indicates that any row that has a value that matches the partition boundary will be moved to the partition immediately to the left. The VALUES clause is used to define the boundary points of the partitions. It’s important to note that these values are boundary points and not the partitions themselves. This will actually result in the creation of three partitions: the first will contain negative values to 50; the second partition will contain the values 51–100; the third partition will contain all values of 101 and over.

The second line creates a partition scheme named MyPS. The AS PARTITION clause is used to specify the partition function that will be used by this scheme. This example uses the MyPF partition function. The TO clause identifies the filegroup or filegroups that will store the partitions. This example uses a single filegroup, named FileGroup1.

Next, the partition scheme needs to be attached to the table that will be partitioned. This example shows the extended CREATE TABLE syntax that enables the table to be partitioned. The first part of the CREATE TABLE statement is unchanged. It specifies the table name, MyTable in this example, and the table’s columns. This simple table uses two columns, named col1 and col2. The new ON keyword is then used to specify the partition scheme that will be used. This example uses the MyPS partition scheme that was just created. And the column that contains the partition’s key data is supplied in parentheses. This example uses the column col1 for the partitioning key. This column is an int data type, which must match the data type specified in the partition function.

There are a few restrictions on the types of columns that can be used for the partitioning key. These restrictions are very similar to the limitation of columns that can be used in an index. The text, ntext, and image data types cannot be used.
Likewise, timestamp columns are also restricted. Only native “T-SQL” data types can be used. You can’t use a user-defined type as a partitioning key. However, you can use the new varchar(max) data type. There is also a limitation of 1000 partitions per table, and all partitions must exist on a single node.

**Index Enhancements**

There are many new enhancements to indexes in SQL Server 2005. First, rebuilding a clustered index no longer forces all of the non-clustered index to be rebuilt. In SQL Server 2000, when you rebuilt a clustered index all of the related non-clustered indexes were rebuilt as well. That’s no longer the case, as SQL Server 2005 keeps the non-clustered indexes intact during the rebuild of the clustered index.

Next, there’s a new included columns feature that enables you to add non-key columns to an index. This new feature enables more queries to be covered by the index, thereby enhancing the performance of the queries by minimizing the need for the SQL Server engine to go to the underlying table to complete the query. Instead, the engine can satisfy the query requirements by using just the data in the covering index. One of the really nice aspects of the new included columns feature is the fact that the included columns that are not part of the key are not included in the maximum size of the index, which is still 900 bytes.

Another new index enhancement that Microsoft added to SQL Server 2005 is the ability to disable an index. Disabling an index stops that index from being maintained by the SQL Server engine and also prevents the index from being used. When an index is disabled, SQL Server deallocates the storage space used by the index but keeps the index’s metadata. Before a disabled index can be enabled again, it needs to be rebuilt using the ALTER INDEX command.

**Online Index Operations**

Prior versions of SQL Server didn’t allow any access to an index while that index was being rebuilt. You needed to wait until the rebuild process completed until the table could be updated again. SQL Server 2005’s new online index operations feature enables applications to access the index as well as perform update, insert, and delete operations on a table while the index rebuilding operation is running. You can find more information about SQL Server 2005’s online index operations in Chapter 3.

**System Catalog and Metadata Enhancements**

In SQL Server 2000 and earlier versions, the system catalog and metadata were stored as part of every database in the master database. With SQL Server 2005, this has changed
and the metadata now resides in the resource database, which the system stores as a sys object. SQL Server 2005 no longer allows any direct access to system tables. This change has enabled better security and faster system upgrades by consolidating the system’s metadata. The catalog metadata is secured using row-level filters. You can learn more about SQL Server 2005’s row-level security in the later section “Security” in this chapter.

The new metadata is completely backward compatible as long as you haven’t used the undocumented system tables that Microsoft has repeatedly warned everyone not to use. The systems metadata in SQL Server 2005 is exposed through a set of catalog views. Catalog views, as well as ANSI INFORMATION_SCHEMA views, Property functions, and Built-in functions, replace the need to use system tables like you may have done in SQL Server 2000. In all, there are over 250 new catalog views in SQL Server 2005, and they can be viewed from the sys schema of every user database. You can find the new system views by using the Microsoft SQL Server Manager Studio to open the Object Browser and then navigating to the Databases | <database> | Views | System Views node. You can also open a new query window and enter the following query:

```
select * from sys.system_views
```

### Multiple Active Results Sets (MARS)

Previous versions of SQL Server were limited to one active result set per connection. SQL Server 2005 is now capable of supporting multiple active result sets on a single connection. This new feature enables you to open a single connection to the database, execute a query and process some results, and then later begin another query and process its results. Your applications can freely go back and forth between the multiple open results sets. Examples showing how you use the new MARS feature are presented in Chapter 4.

### Bulk Data Loading

SQL Server 2005 provides some great improvements as well as performance increases in bulk data loading. The bulk data loading process now uses an XML-based format file that provides all of the functionality found in previous versions of the Bulk Copy Program’s (BCP) format file and more. Plus, the XML format makes the BCP format file easier to read and understand. For backward compatibility with existing applications, the old BCP format file can still be used.

SQL Server 2005’s bulk data loading process now supports logging of bad rows. This enables the bulk data loading process to continue even if invalid rows or data
are encountered. Incorrectly formatted rows are written to an error file along with a
description of the error condition. Rows that violate constraints are redirected to an
error table along with their specific error condition.

**Full-Text Search**

Support for Full-Text search has also been enhanced in SQL Server 2005. Earlier
versions of SQL Server required the use of stored procedures to create Full-Text
search catalogs. With SQL Server 2005, several new DDL statements have been
introduced to enable you to work with SQL Server’s Full-Text search features. For
instance, two of the new T-SQL Full-Text search DDL statements are: CREATE
FULLTEXT CATALOG and CREATE FULLTEXT INDEX.

Other enhancements to Full-Text search in SQL Server 2005 include the ability to
back up and restore Full-Text search catalogs and indexes along with your database
data. Likewise, Full-Text catalogs and indexes can be attached and detached with
their corresponding databases. Another interesting enhancement in SQL Server
2005’s Full-Text search support is the ability to use a thesaurus to find synonyms
of search words.

**T-SQL Query Processor Enhancements**

There are several enhancements to the query processor in SQL Server 2005,
including Common Tables Expressions (CTE), an enhanced TOP clause, an
enhanced WAITFOR statement, and a new OUTPUT clause for DML statements.
Examples of using these enhancements in T-SQL are presented in Chapter 4.

**Security**

Security has been a big push for Microsoft ever since the company kicked off its
Trustworthy Computing Initiative. The goal of Microsoft’s Trustworthy Computing
Initiative is to make all Microsoft products more secure and more reliable. As you
would expect, being a part of the Trustworthy Computing Initiative, SQL Server
2005 is the recipient of a number of very significant security enhancements.
Microsoft’s security push for SQL Server is focused on making the product more
secure and more robust from its design through its deployment. When designing the
security enhancements for SQL Server 2005, Microsoft followed some basic security
tenets. First, it wanted to make the system secure right out of the box by gearing all
of the default installation settings toward creating a secure environment. While it left
options open for users to select less secure settings, selecting these options required
deliberate choices. Next, Microsoft followed the principle of least privileges in their system design. The system is designed so that an individual should have only the require privileges to perform a given function and no more. Finally, Microsoft wanted to reduce the potential exposure surface area by providing the ability to install only those components that are needed.

All of the new security features found in SQL Server 2005 were deeply influenced by the things Microsoft discovered during its security push in early 2002 and were carried into the SQL Server 2005 design and implementation. Some of the core security feature enhancements found in SQL Server 2005 that you’ll read about in this section include the separation of users from schemas, the new stored procedure execution context, more granular control of permissions, new password policy enforcement, changes to row-level security, and enhanced security for catalogs.

User-Schema Separation

The most significant new security-related change found in SQL Server 2005 is user-schema separation. A user, or perhaps more accurately, a principal, is any entity against whom database objects are secured. A principal can be a Windows user, a SQL Server user, a role, or an application role. With SQL Server 2000, directly owned by users database objects and the users themselves were found in the sys_users system table. That’s all changed with SQL Server 2005. Now database objects are owned by schemas. Users no longer directly own database objects; instead they own schemas. With SQL Server 2005, users and other security principals can be found in the new sys.database_principals view. SQL Server 2005’s list of schemas can be found in the new sys.schemas view.

A schema is a container of objects. The schema is identified by the third part of the four-part object naming syntax used by SQL Server. The following example illustrates SQL Server 2005’s naming syntax, where each part of the name gets increasingly more specific.

Server_name.Database_name.Schema_name.Object_name

In all of the previous releases of SQL Server, the schema name and the owner name were essentially the same thing. With SQL Server 2005, the owner has been separated from the schema. When SQL Server 2000 and earlier releases resolved object names, SQL Server first looked for Database_name.User_name.Object_name, and if that failed, it then looked for Database_name.dbo.Object_name.

The main reason for this separation of user and schema in SQL Server 2005 is to address the problem of needing to change the ownership of multiple database objects if a given user (aka the old object owner) leaves the organization. In addition, the
action of changing a database object’s ownership also resulted in a name change. For instance, if the owner of Table1 in database MyDB is changed from UserA to UserB, then the qualified name of Table1 will be changed from MyDB.UserA.Table1 to MyDB.UserB.Table1. To help prevent this problem, many organizations adopted the standard of having all database objects owned by dbo, but there was nothing in the server that forced this practice.

SQL Server 2005’s implementation of the concept of a database schema introduces a level of abstraction in the chain of database object ownership. You can see SQL Server 2005’s database object ownership chain in Figure 1-3.

With SQL Server 2005, database objects are contained in schemas, and the schemas are in turn owned by users. This new level of abstraction makes the problem of changing database object ownership much more manageable. Dropping a user that owns database objects in SQL Server 2005 means that the DBA now needs to change only the ownership of the schema and not all of the individual database objects. This vastly reduces the number of objects that the DBA needs to touch in order to change the ownership of the objects in a database. To change the ownership of all the objects in a SQL Server 2005 database, you simply change the owner of the schema, and then you can drop the old user. Changing the owner of database object doesn’t change an object’s name because the schema name doesn’t change, just its ownership.

As you might expect, the new schema object changes the way that SQL Server performs database object name resolution. Each user now has a default schema associated with that user, and SQL Server 2005 will first look for an unqualified object name using the user’s default schema. If that fails, SQL Server will look for the object using the schema name of dbo. For instance, if UserA has a default schema of MySchema1 and that user performs a query looking for Table1, then
the server will first attempt to resolve the name using MySchema1.Table1 and then fall back to dbo.Table1.

Just as SQL Server 2000 databases could contain multiple users and roles, SQL Server 2005 databases can contain multiple schemas. Each schema has an owning principal, which is typically either a user or a role. For name resolution purposes, each user has a default schema. The actual database objects are then contained in a schema. To create new database objects inside a schema, you must have CREATE permission for the database object itself and ALTER or CONTROL permission for the schema. Ownership chaining is still based on actual owners, not schemas.

SQL Server 2005 introduces several DDL changes for dealing with the new user and schema separation, including a CREATE/DROP/ALTER statement for USER, ROLE, and SCHEMA objects. The following listing demonstrates how a database schema is created and assigned:

```sql
/* Create a login */
CREATE LOGIN UserA WITH PASSWORD = 'ABC123#$'
GO
/* Create a user for that login - the schema doesn't need to exist*/
CREATE USER UserA FOR LOGIN UserA
    WITH DEFAULT_SCHEMA = MySchema
GO
/* Create the schema and assign its owner */
CREATE SCHEMA MySchema AUTHORIZATION UserA
GO
/* Create a Table in the new schema */
CREATE TABLE MySchema.Table1 (col1 char (20))
GO
```

The first line in this listing creates a new login named UserA and sets a password for that login. The next line creates a new user named UserA for the login and sets the default schema for UserA to MySchema. The actual schema does not need to exist at the time it is specified in the CREATE USER statement. If you don’t specify a default schema when you create a new user, then the default schema will be set to dbo. Next, the CREATE SCHEMA statement is used to create a new schema named MySchema. The AUTHORIZATION clause sets the owner of the schema to be UserA. Finally, a table named Table1 is created in the schema named MySchema. The owner for MySchema and its objects such as Table1 is UserA.

**Stored Procedure Execution Context**

While Microsoft has referred to one new feature as a stored procedure execution context, it really applies to modules rather than just stored procedures. A module
can be a stored procedure, a function, or an assembly. Setting the execution context for a module causes all statements that are contained in that module to be checked for permissions against the specified execution context. In other words, by setting the execution context of a given module, you cause all the statements that are contained in that module to be executed using the authority of the user that you specify rather than the actual caller of the module. This new feature enables you to get advantages similar to those realized through SQL Server 2000’s ownership chaining, but it is more flexible, as it doesn’t have the same limitations. For example, unlike SQL Server 2000’s ownership chaining, which didn’t allow you to alter the execution context for dynamic SQL, SQL Server 2005’s module execution context applies to dynamic SQL just as it does to static SQL. To better understand this, take a look at Figure 1-4, which demonstrates SQL Server 2000’s ownership chaining.

For UserA to execute dbo.Proc1, UserA must have execute permission for that object. However, when dbo.Proc1 accesses dbo.Table1, no permissions are checked, because dbo is the owner of both objects. This is an example of an intact ownership chain. In the next scenario, for UserA to execute UserB.Proc2, UserA must have Execute permissions for that object. Then, when UserB.Proc2 attempts to access UserC.Table2, Select permissions from UserA must be checked. In this case, because UserB.Proc2 and UserC.Table2 have different owners, the ownership chain is broken.

SQL Server 2005’s execution simplifies this scenario, as shown in Figure 1-5. In this scenario, when UserA attempts to execute UserB.Proc2, SQL Server checks to ensure that UserA has Execute permissions for the UserB.Proc1. If the object UserB.Proc1 is created with an execution context that specifies that a stored procedure will be executed as UserZ, then when the UserB.Proc1 stored procedure attempts to access UserC.Table1, SQL Server will check for Select permissions only on the user specified in the execution context, which in this case is UserZ. No Select permissions are required for UserA, who is the actual caller.
The following listing shows how you change the execution context of a stored procedure named MyProc1:

```
ALTER PROC MySchema.Proc1 WITH EXECUTE AS USER UserB
```

This statement shows the new WITH EXECUTE clause. Here, the stored procedure named Proc1 that’s contained in MySchema is set to execute under the context of UserB. You must specify a user name for the execution content. You cannot specify the name of a role. Changes in execution context are stored in the new sys.sql_modules view.

**More Granular Permissions Control**

SQL Server 2005 also provides more granular control over permissions. With SQL Server 2005, Microsoft added more permissions at different scopes. The scopes to permissions that can be applied are: server, database, schema, object, and principal. The design idea behind SQL Server 2005’s enhanced permissions is the principle of least privileges, giving the DBA the ability to control exactly what permissions are assigned. The new granular permissions do not do away with SQL Server’s existing fixed roles. All of the older roles are still present and can coexist with no problems alongside the new permissions. One specific scenario that these new granular permissions are meant to address is the case of auditing. On SQL Server 2000, you needed to be a member of the sysadmins group in order to perform auditing. However, membership in this group also enables many other, more far-reaching capabilities. Some of the new permissions available in SQL Server 2005 enable auditing functions to be performed without requiring that the user be a part of the sysadmins group.

The same basic permission states of GRANT, DENY, and REVOKE that were used in previous versions of SQL Server still apply. One thing that’s different about the way that SQL Server 2005 uses permissions is that the same permission can be applied at multiple scopes. For example, if you apply a permission to the database scope, it applies to all objects in the database. If you apply a permission to the schema level, it applies to just those objects contained in the schema. With the exception of the DENY permission, the higher-scope permission is always used. However, a DENY at any level always takes precedence. Table 1-1 lists some of the
most important new SQL Server 2005 permissions. The server permissions are found in the sys.server_permissions view, while the database permissions are found in the sys.database_permissions view.

### Password Policy Enforcement

Another important new security feature found in SQL Server 2005 is support for password policies. This new policy enforcement feature follows local Windows password policies and enables you to implement a consistent enterprise-wide security policy, not just for your Windows server systems but also for your SQL Server database systems. SQL Server 2005 now has the capability to enforce password strength, password expiration, and account lockout policies. As you would expect, the password strength policy forces passwords to consist of a given complexity. The password expiration policy ensures that passwords expire and must be reset at a given interval. And the account lockout policy enables an account to be locked out after a given number of bad password attempts. All of these new password policies are supported on Windows Server 2003. However, only the password complexity policy is supported on Windows 2000 Server. Following Microsoft’s security push, all of the policies are enabled in the default installation, but they can also be reconfigured on a per-login basis. SQL Server 2005 stores the new password policies in the sys.sql_logins catalog view.

---

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>Grants the ability to change the properties of an object. Also grants the ability to execute CREATE/DROP/ALTER statements.</td>
</tr>
<tr>
<td>ALTER ANY ‘X’</td>
<td>Grants the ability to alter any object of type X. For instance, if you substituted TABLE for X, this would grant the ability to alter any table in the database.</td>
</tr>
<tr>
<td>ALTER TRACE</td>
<td>Grants the ability to perform auditing and run Profiler.</td>
</tr>
<tr>
<td>CONTROL</td>
<td>Grants a principal owner-like permissions.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Grants the ability to access an object. Now applies to the schema and database levels rather than just the object level.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Grants the ability to execute a procedure, assembly, or function. Now applies to the schema and database levels rather than just the object level.</td>
</tr>
<tr>
<td>IMPERSONATE</td>
<td>Grants a login or a user the ability to impersonate another user.</td>
</tr>
<tr>
<td>TAKE OWNERSHIP</td>
<td>Grants the ability to assume the ownership of an object.</td>
</tr>
<tr>
<td>VIEW DEFINITION</td>
<td>Grants the ability to view an object’s metadata.</td>
</tr>
</tbody>
</table>

Table 1-1 Some New Permissions in SQL Server 2005
The final security-related enhancement that I’ll cover in this section is the new catalog security provided by SQL Server 2005. The system tables that were used by SQL Server 2000 in the individual databases and in the master database are now implemented as catalog views in SQL Server 2005. The server’s metadata exposed in these views is secured by default, and there are minimal public permissions. SQL Server 2005’s catalog views employ row-level security, limiting access to the data contained in those views to only those objects that you own or have permissions to. Naturally, sa is an exception to this. The sa account still has access to all of the objects in the server.

To enable a user or role to access the metadata, the DBA needs to use the new VIEW DEFINITION permission. The VIEW DEFINITION permission can be used to grant a user who is not the owner and doesn’t have permissions to access an object the ability to view the object’s metadata. The VIEW DEFINITION permission can be applied at the server, database, schema, and object scopes.