Exam
Manual

LearnSmart

Microsoft SQL Server 2005

(70-431) Microsoft Certified IT Professional (MCITP)

This LearnSmart exam manual covers all the important concepts you must know in order to successfully complete the SQL Server 2005 exam (70-431). By studying this exam manual, you will become familiar with an array of exam-related content, including:

- Installing and Configuring SQL Server 2005
- Implementing High Availability and Disaster Recovery
- Supporting Data Consumers
- Maintaining Databases
- And more!

Give yourself the competitive edge necessary to further your career as an IT professional and purchase this exam manual today!

Microsoft SQL Server 2005 (70-431) LearnSmart Exam Manual

Product ID: 010458
Production Date: July 18, 2011

Warning and Disclaimer

Every effort has been made to make this document as complete and as accurate as possible, but no warranty or fitness is implied. The publisher and authors assume no responsibility for errors or omissions. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this document.

Volume, Corporate, and Educational Sales

Favorable discounts are offered on all products when ordered in quantity. For more information, please contact us directly:

1-800-418-6789
solutions@learnsmartsystems.com

International Contact Information

International: +1 (813) 769-0920

United Kingdom: (0) 20 8816 8036

Table of Contents

Abstract

This Exam Manual will help you prepare for the Microsoft 70-431 SQL Server 2005 Implementation and Maintenance exam. This exam covers installation, programming, administration, and use of databases in a SOHO, Medium, and Large business environment. It is considered to be a difficult exam and should not be taken lightly. Many people fail this exam on the first attempt; so, do not be frustrated if you find it difficult.

What to Know

Microsoft's 70-431 Exam is the only exam that is required to become a Microsoft Certified Technology Specialist (MCTS) on SQL Server 2005. This exam can be combined in several variations with the Microsoft 70-441, 70-442, 70-443, 70-444, 70-445, 70-446, 70-447 exams to attain the Microsoft Certified IT Professional (MCITP) certification with a concentration in either Database Development, Administration, or Business Intelligence Development.

The exam is available through either Pearson Vue or Thomson Prometric testing centers and vouchers for the exams can be purchased on our website.

Tips

It helps to have a great deal of experience with SQL Programming and Administrations before you take the exam. Additionally, it is recommended that you have installed, used, and tweaked SQL Server 2005 for a great period of time before you attempt the exam. SQL Server 2005 is a major update from SQL Server 2000 and has almost completely changed the field of Administration. Good luck!

Chapter 1: Install and Configure SQL Server 2005

1.1 Overview of Hardware and Software Requirements

Before installing SQL Server 2005 it is helpful to gain a grasp of :

- *SQL Server 2005 editions*
- *Installation requirements for different editions*

SQL Server 2005 Editions

SQL Server 2005 is available in a wide variety of editions to suit different environments and application development requirements, specifically:

- The **Enterprise edition** has the best performance and includes the widest possible scope of components and features, such as enhanced data warehousing, support for Web sites, business analytical capabilities, and high availability functionality. This edition can be installed on 32-bit as well as 64-bit computer.
- The **Standard edition** targets advanced database applications for small and mid-sized organizations. A very large subset of essential and interesting business database capabilities from Enterprise edition are in Standard edition, such as database mirroring, the Database Tuning Advisor, data warehousing support and native Web services. This edition can be installed on 32-bit as well as 64-bit computers.
- The **Workgroup edition** focuses on core data management capabilities with no limits for either the number of users or the size of databases. In addition to data management roles, this edition can be deployed for use along with a Web server or function as a local departmental or branch office database server. This edition can be installed exclusively on 32-bit computers.
- The **Express edition** is especially for those wishing to upgrade MSDE from either SQL Server 2000 or SQL Server 7 because Microsoft offers this edition, like MSDE, available without charge. In addition, by selecting the Advanced Services version of this edition, you can obtain support for full-text queries and Reporting Services as well as an integrated SQL Server Management Studio Express package.
- The **Mobile edition** is the only edition that offers relational database management capabilities for smart, mobile devices. In addition, you can synchronize through replication Mobile edition databases with other editions of either SQL Server 2005 or SQL Server 2000. It has a reduced feature set, for example no stored procedures. It will be renamed to SQL Server Everywhere in the future.
- The **Developer Edition** allows the use of all of the SQL Server 2005 Enterprise edition features, but it is licensed for development and testing purposes – not for production systems, like the Enterprise and Standard editions. You can upgrade a Developer edition to Enterprise, Standard, or Workgroup editions.
- The **Evaluation edition** is a 180-day trial version of the Enterprise edition, which you can upgrade during the trial period to Enterprise, Standard, Workgroup, Developer, or Express editions.

Installation Requirements

The System Configuration Checker automatically inspects your computer when you attempt to install a SQL Server 2005 edition. The minimum hardware and software requirements vary by SQL Server 2005 edition. An overview of the requirements are:

- Pentium III processor with 512 MB of RAM, except for the Express edition which requires only 192 MB of RAM.
- 1 GB of storage, except for the Express edition which has a 512 MB requirement.
- The 64-bit versions of SQL Server 2005 editions can run with processors, such as the AMD Opteron or Anthlon 64 as well as the Intel Xenon and Pentium IV with EM64T support. You can also run the Enterprise edition on an Itanium processor.
- Special requirements apply to the Mobile edition (for more details see http://msdn2.microsoft.com/en-us/library/ms172914.aspx).

The operating system requirements vary by edition.

- **Enterprise and Standard edition** - run on Windows 2003, including SBS, with SP1 and Windows 2000 with SP4. Neither the Enterprise nor Standard editions run on Professional editions of operating systems.
- **Workgroup, Evaluation, and Developer editions** - run on the above operating systems as well as Windows XP Professional, Windows XP Media Center Edition and Windows XP Tablet Edition with SP2.
- **Developer and Express editions** - run on all of the above operating systems, as well as Windows XP Home and Windows 2003 Server Web Edition.
- **64-bit versions of SQL Server 2005 -** install on comparable 64-bit operating systems to the 32-bit operating systems listed above.
- **32-bit versions of SQL Server 2005** - can be installed on computers with a 64-bit processor that is running Windows on Windows (WOW64).

The Internet Explorer requirements for 32-bit and 64-bit versions are the same.

- Internet Explorer 6.0 SP1 is required for the SQL Server Management Studio, Business Intelligence Management Studio, and the Report Designer component of SQL Server 2005 Reporting Services, Microsoft Management Console, HTML Help, and connecting to a server that requires encryption.
- When running SQL Server 2005 Reporting Services, you need both IIS 5 or greater and ASP.NET 2.0.

1.2 Installing Instances

Like SQL Server 2000, SQL Server 2005 allows multiple instances of some SQL Server components to run concurrently on a computer. These are referred to as instance aware components, and are:

- **Database Engine**
- **Analysis Services**
- **Reporting Services**

Each instance runs independently of other instances and are basically standalone servers that have unique settings for collations and all other options. Each instance on a computer has its own directory structure, registry structure settings, and data files. All instances share some base files in the system drive:\Program Files\Microsoft SQL Server\90 directory.

A large selection of how-to topics covering various installation scenarios is available from Microsoft's msdn2 site (http://msdn2.microsoft.com/en-us/library/ms143725(SQL.90).aspx). Use the how-to topics at the msdn2 site to perform an installation. Use this section in the exam manual to prepare for the installation before performing the installation steps and to interpret the outcome after you complete the installation steps.

SQL Server 2005 Instance Aware Components

There are two broad types of instances – default instances and named instances.

- ***The default instance is typically the initial instance of SQL Server on a computer.*** This default instance can be a SQL Server 6.5, 7, 2000, or 2005 version. There can be only one default instance per computer. The default instance is internally named MSSQLSERVER. When connecting to a default instance, a client can specify just the network name of the computer.
- ***A named instance is determined by a user during setup.*** When connecting to a named instance, a client specifies the name/address of the computer and the instance name. (Each instance can also be configured with it's own TCP/IP port as well.)

Instance aware components have a few special considerations

- An instance-aware component requires its own instance although it can be part of a single SQL Server 2005 installation.
- The maximum number of supported instances vary by the type of SQL Server 2005 edition
    - The Enterprise and Developer editions support up to 50 instances each of the Database Engine, Analysis Services, and Reporting Services
    - The Standard, Workgroup, and Express editions support up to 16 instances each of the Database Engine, Analysis Services, and Reporting Services
- Most SQL Server 2005 components do not have a distinct instance associated with their installation. Examples of these other component types include Integration Services, Notification Services, as well as the SQL Server Browser Service and WMI providers.

Instance Naming Conventions

You can assign an instance name when running the SQL Server 2005 Installation Wizard. The maximum number of characters allowed in an instance name is 16. The characters within a name are not case sensitive (myInstance is the same as MYinstance).

- The first character must be either a Unicode Standard 2.0 letter or an underscore (_).
- Subsequent characters after the first one for an instance name can be a Unicode Standard 2.0 letter, decimal number from Basic Latin or other national scripts, the dollar sign ($), and an underscore.
- Instance names cannot be reserved names, such as "MSSQLServer".
- Instance names cannot contain embedded spaces nor any of the following special characters: backslash (\), comma (,), colon (:), semicolon (;), single quote ('), at sign (@).

The approach you use to connect to a SQL Server instance can vary by the type of instance and the client application that you are using.

- To connect to the default instance on a computer with the SQL Server 2005, type "sqlcmd" at a command prompt and press the Enter key. You can then return the computer's name by running the following query: SELECT @@SERVERNAME. The sqlcmd utility is a native client for the SQL Server Database Engine.
- With SQL Server Management Studio (SSMS), you can connect to the default instance by representing the instance name with a period (.), (local) or localhost. Enter the period in the Server name box of the Connect to SQL Server dialog box.
- To connect to a named SQL Server instance with the sqlcmd utility, type: sqlcmd –S Computer_name\Instance_name.
- With SSMS, connect to a named instance by entering Computer_name\Instance_name in the Server name box of the Connect to SQL Server dialog box.

Key Instance Settings

An instance with its associated data is defined by a set of files stored on a computer's system drive, which is normally the C: drive. Default and named instances have specific path names in which they store files. By learning the conventions for the path names and how they relate to instance settings, you will know how to inspect the file structure of a computer to determine the SQL Server instances on a computer.

Recall that selected SQL Server 2005 components install within their own instance. Aside from the Database Engine component, the other two components with associated instances are Analysis Services and Reporting Services. Each installed instance has a unique instance ID assigned to it. This ID value is an ordinal number starting at 1 based on the order of installation. If an instance is removed, any subsequent instance installations fill in gaps in the ordinal instance ID sequence.

The folders for the instance-specific files reside in <system_drive>:\Program Files\Microsoft SQL Server. The files for the first, second, and third installed instances reside in the following three paths. The first

instance is normally the default instance.

- <system_drive>:\Program Files\Microsoft SQL Server\MSSQL.1
- <system_drive>:\Program Files\Microsoft SQL Server\MSSQL.2
- <system_drive>:\Program Files\Microsoft SQL Server\MSSQL.3

The three SQL Server components with associated instances save their files within a specially named folder within the path for an instance. The associated folders are extensions of the base paths mentioned above. The folder names for the components are:

- Database Engine: "..\MSSQL"
- Reporting Services: "..\Reporting Services"
- Analysis Services: "..\Analysis Services"

Therefore, if the first instance were the default instance for the Database Engine, its path would be as follows: <system_drive>:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL. The directories within the folder for an instance-aware component vary according to the type of component. For example, a Data directory within the MSSQL folder is not present in the Reporting Services folder. The Data directory is the default location for saving data files managed by a SQL Server instance.

A computer's registry also stores information associating instance ID numbers with instance names. The following three registry keys store information mapping instance ID numbers to instance names for Database Engine (SQL), Reporting Services (RS), and Analysis Services (OLAP).

- [HKEY_LOCAL_MACHINE\Software\Microsoft\Microsoft SQL Server\Instance Names\SQL]
- [HKEY_LOCAL_MACHINE\Software\Microsoft\Microsoft SQL Server\Instance Names\RS]
- [HKEY_LOCAL_MACHINE\Software\Microsoft\Microsoft SQL Server\Instance Names\OLAP]

Database Mail for SQL Server Instances

Database Mail is one of the many new features introduced with SQL Server 2005. This new feature deprecates SQL Mail, which is still available but shouldn't be used as it won't exist in the next version. Database Mail is more scalable and reliable than SQL Mail as a means of sending email messages from a SQL Server instance.

Database Mail is an instance-based feature that is *not* instance-aware. There is no need for a special SQL Server instance when you use Database Mail. Instead, you enable Database Mail within the Database Engine for a SQL Server instance. You can enable Database Mail for a SQL Server instance with either a graphical interface or with T-SQL code.

- To use the graphical method, invoke the SQL Server Surface Area Configuration tool. Use the

tool to select and enable the Database Mail feature for the Database Engine of the instance you are using.

- To use T-SQL code, open a new query for the server instance that you wish to enable Database Mail. Then, run the script at the following URL: http://msdn2.microsoft.com/en-us/library/ms191189.aspx. Notice the script's code reconfigures a SQL Server instance with the sp_configure system stored procedure and the RECONFIGURE statement.

After you enable Database Mail, you can setup the feature with the Database Mail Configuration Wizard, which guides you through the process. You can invoke the wizard from Object Explorer in SSMS. Merely open the Management folder for a connected SQL Server instance and double-click Database Mail.

1.3 Creating Databases

After you install an instance of SQL Server 2005 Database Engine component, you can add one or more databases to it. A database often holds data in one or more tables along with other database objects, such as views and stored procedures. There are two broad classes of databases: system databases and user-defined databases.

System Databases

System databases help with the administration and operation of a database instance. SQL Server 2005 has five system databases. These databases are generally created when you install SQL Server 2005.

- **Master** - Records system-level information for an instance of SQL Server.
- **Msdb** - SQL Server Agent uses this database to manage alerts, schedule jobs, implement database mail, keep history of backups, etc.
- **Resource** – special hidden database that contains the code to implement the sys schema objects that are used to query system information.
- **Model** - Serves as a template for new user-defined databases. Configure the model database so that it has the options and objects that you want in all new user-defined databases.
- **Tempdb** - created by SQL Server 2005 at the start of every session. Users can store temporary results for a session in this database, and a SQL Server instance can also use this database to help manage selected features, such as snapshot isolation transactions and sorting sets of data.

Creating Databases

You can create a new user-defined database with either the CREATE DATABASE or the New Database dialog box in SSMS. With either approach, all you have to do as a minimum is assign a name to your database. For example, the following statement adds a database named PrepLogic to the currentSQL Server instance.

```
CREATE DATABASE PrepLogic
```

Viewing Data and Log Files

You can learn more about the database after you create it with the sys.databases view and the sp_helpdb system stored procedure. The syntax for sp_helpdb optionally includes the name of a database. If you specify a database name, sp_helpdb returns two sets about the named database. If you do not specify a database, then the stored procedure returns one set with a separate row for each database within an instance.

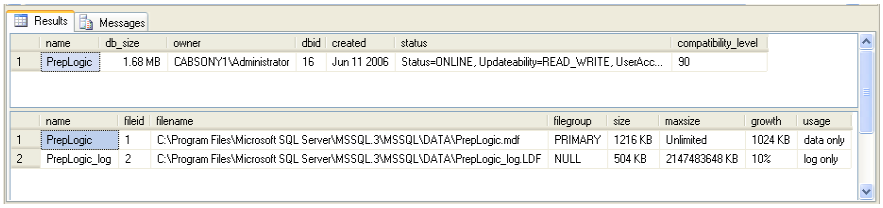
The first set returned by sp_helpdb describes the basics about the database:

- Name - The database name.
- db_size - the total database size.
- owner - the login of the database owner.
- dbid - the unique dbid value for the database in a SQL Server instance.
- created - the date that a database was created.
- status - comma-separated list of database options.
- compatibility_level - a value showing the database compatibility level: 90, 80, 70, and 65 are respectively for SQL Server 2005, SQL Server 2000, SQL Server 7, and SQL Server 6.5.

If you specify the database name, a second set returns information about the data and log files for the named database. Recall that this second result set only appears if you follow the sp_helpdb system stored procedure with the name of a specific database.

- name - the logical name of the file.
- fileid - the file's unique fileid for the database.
- filename - the operating system path and file names for the file.
- size - the file's size in megabytes.
- maxsize - the maximum size to which a file can grow. A value of UNLIMITED means the file can grow until the disk containing it is full.
- growth - designates the physical units or percentage by which a file grows when it needs new storage space.
- usage - designates the usage as "data only" or "log only".

Figure 1.1 shows the Results tab with the two result sets for the PrepLogic database generated by the preceding CREATE DATABASE statement. The database was created by the Administrator for the CABSONY1 computer. The data file has the name PrepLogic.mdf, and the log file has the name PrepLogic_log.ldf. Both files reside in the following path: c:\Program Files\Microsoft SQL Server\MSSQL.3\MSSQL\DATA\. This path indicates the files belong to the computer's third installed SQL Server 2005 instance (MSSQL.3).


Results Messages

	name	db_size	owner	dbid	created	status	compatibility_level
1	PrepLogic	1.68 MB	CABSONY1\Administrator	16	Jun 11 2006	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	90

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	PrepLogic	1	C:\Program Files\Microsoft SQL Server\MSSQL.3\MSSQL\DATA\PrepLogic.mdf	PRIMARY	1216 KB	Unlimited	1024 KB	data only
2	PrepLogic_log	2	C:\Program Files\Microsoft SQL Server\MSSQL.3\MSSQL\DATA\PrepLogic_log.LDF	NULL	504 KB	2147483648 KB	10%	log only

Figure 1.1. The Results tab for sp_helpdb PrepLogic.

Notice that the list of comma-separated values in the status column of Figure 1.1 is truncated. This is to conserve space (the full status column is very wide).

Over 180 system catalog views can be used to view metadata about the contents of a SQL Server instance.

- For example, you can easily examine database options with the sys.databases view. The sys.databases view has a separate column for each database option setting.
- The sys.database_files view is another system catalog view with extensive metadata on the files underlying a database. You can use this view to complement the second result set returned by the sp_helpdb system stored procedure.

Configuring Database Options and Files

After creating a database, you can modify a database's options and files with the ALTER DATABASE statement. See Books Online for more information about the types of database modifications and additions that you can make after creating a database. You can also obtain help for the ALTER DATABASE statement from Microsoft's msdn2 site (http://msdn2.microsoft.com/en-us/ms174269.aspx).

There are three types of database files.

- **primary data file** - the startup data file. If there is only one data file in a database, it is the primary file. This file can hold tables and other database objects, such as stored procedures and user-defined functions. The primary data file can point to other data files. The primary database file normally has a .mdf extension (the extensions on all files does not have any real significance.)
- **secondary data files** - A database can have one or more optional *secondary* data files. Use secondary data files to distribute data across multiple files. Secondary data files normally have a .ndf extension.
- **log files** - holds a log of the transactions on a database that can allow you to reconstruct the database or rollback transactions. A database can have multiple log files. The recommended file extension is .ldf.

Every database has one or more filegroups. You can use filegroups to organize sets of data files. Log files do not belong to a filegroup.

- The primary filegroup contains the primary data file. This group is automatically created when you create a database.
- You can create one or more user-defined filegroups that complement the primary filegroup. User-defined filegroups can organize secondary data files.

- The default filegroup is the one that data files belong to if you do not specify a filegroup when creating a data file. Unless you designate otherwise, the primary filegroup is the default filegroup.

You can use ALTER DATABASE statement for modifying the files for a database. For example, you can

- Set the file size or growth factors so that the files fit the needs of your application
- Create multiple data files and place them on different drives to improve database performance and reduce the chance that a file will expand beyond the limit of a disk drive
- Place your log file on a different drive than the drives for your data files to improve performance and recoverability
- Create multiple log files to reduce the chance of exceeding the capacity of a single file or drive
- Create filegroups to help organize your data files

You can also use the ALTER DATABASE statement to manage database options. The full list of options is enumerated and described in the Books Online Help topic for ALTER DATABASE (Transact-SQL). Selected options are mentioned below.

- db_user_access_option - allows you to limit database access so that it can be used by one user at a time (SINGLE_USER), just members of the db_owner fixed database role or the dbcreator and sysadmin fixed database roles (RESTRICTED_USER), or multiple concurrent users (MULTI_USER).
- db_update_option - allows you to determine if a database allows users to both read and write to it or just read from it.
- external_access_option - has two settings that determine cross-database access. DB_CHAINING setting determines whether the objects within a database can participate in cross-database ownership chains. The TRUSTWORTHY setting determines whether modules with impersonation can access resources such as tables or views outside the database.
- recovery_option - allows you to designate a recovery model for a database. The option settings are FULL, BULK_LOGGED, and SIMPLE. See the Recovery Models and Transaction Log Management topic in Books Online or navigate to http://msdn2.microsoft.com/en-us/library/ms366344.aspx

If you have modified a database in an undesired way or if you no longer need a database, you can delete the database with a DROP DATABASE statement. ***Data Definition Language for nearly all database objects in SQL Server 2005 have three basic types of statements – namely, CREATE, ALTER, and DROP.*** For example, there are CREATE TABLE, ALTER TABLE, and DROP TABLE statements for managing tables within a database.

1.4 Configuring SQL Server Security

SQL Server 2005 has three key elements to its security.

1) It manages authentication of users through tracking principals.

2) It manages the authorization to perform tasks through the assignment of permissions to securables for principals.

3) You can encrypt your data to thwart an unauthorized user who somehow slips through authentication and authorization security.

Managing and Tracking Principals

Principals are entities that can issue requests for resources, such as to view the data in a table or view. You can think of principals as existing at three levels – Windows, a SQL Server instance, a SQL Server database.

- Examples of Windows-level principals are Windows users and groups mapped to a login for a SQL Server instance.
- Examples of SQL Server instance-level principals are a SQL Server login and a fixed server role.
- A database-level principal can be a database user or a role defined for a database or application.

As you can see, principals are typically logins, users, and roles. You can add, modify and drop principals with three DDL statement types (CREATE, ALTER, and DROP).

- A login grants access to a SQL Server instance. There are two traditional types of logins – those authenticated by Windows (Windows users or groups) or SQL Server (a SQL Server login). SQL Server 2005 introduces a type of login based on a certificate or an asymmetric key that avoids the need to rely on passwords.
- Just as logins grant access to a SQL Server instance, a database user grants access to a specific database. A login can access a database when it maps to a database user.
- You can group users at the server or database level using roles. Servers and databases have a set of fixed roles that give a predefined set of rights. Databases also have:
    - User-defined roles – Allows you specify the permissions for a role. Roles can be members of other roles to provide complex rights.
    - Application roles - a special kind of role that suspends other SQL Server security and grants access to a database based on the submittal of a password. Any principal submitting the password can gain access to the database application.

Several system catalog views help you to keep track of principals.

- The sys.server_principals view has a row for every principal at the Windows or SQL Server instance level. The name column returns the principal's name, and the type column indicates the principal's status as SQL login, Windows login, Windows group, Server role, Login mapped to a certificate, or Login mapped to an asymmetric key.
- The sys.database_principals view has a row for every principal at the database level. The name column of this view returns the principal's name, and the type column designates the principal as one of the following types: SQL user, Windows user, Windows group, Application role, Database role (fixed or user-defined), User mapped to a certificate, User mapped to an asymmetric key.
- The sys.sql_logins view represents SQL Server standard logins. It inherits columns from the sys.server_principals views and adds special columns relating to the password for standard logins.
- The sys.database_role_members allows you to track which principals belong to fixed and user-defined database roles.

Managing and Tracking Securables

Principals operate within a SQL Server instance by making requests for securables, such as creating a database or reading the values in a table. If a principal has permission for a request, the SQL Server instance authorizes the request. Therefore, in order to understand the role of securables in SQL Server security, you have to learn about the types of securables as well as the permissions they expose. You also need to gain an understanding of how to manage the permissions available to a principal.

Securables exist at three different levels. You can manage securables with the three traditional DDL statement types (CREATE, ALTER, DROP), as well as some system stored procedures (columns cannot be renamed using ALTER, for example). The sys.objects view stores metadata for many securables, such as tables, views, stored procedures, and even CLR types, such as a CLR aggregate function. Many other system catalog views are also resources for tracking securables, for examples sys.tables, sys.triggers, and sys.schemas.

- The top level for securables is the SQL Server instance level. Within an instance, you can have databases, logins, endpoints. Yes, a principal, such as a login, can be a securable! Most applications require at least one principal with permission to create, alter, and drop a login. The endpoint object is a new one introduced with SQL Server 2005 to facilitate three new features: native XML Web services, Service Broker, and database mirroring.
- The next securable level is for objects that exist at the database level, such as database users and roles or database schemas. Keys and certificates used for encryption also exist at the database level.
- SQL Server 2005 introduces database schema objects. Schemas are containers that can be used to organize your objects logically, and to grant rights to contained objects.
    - Any one database can have multiple schemas with different database objects in each schema.
    - Individual database objects can belong to a schema without being owned by the schema.

    - A database schema can be owned by a database role so that no one SQL Server login or Windows account owns a schema.

- Within a database schema, you can have traditional database objects, such as tables, views, and stored procedures, as well as two new database objects, including a new CLR Type object and an XML Schema Collection.
    - The CLR Type object deprecates the traditional T-SQL user-defined type, also known as an alias type, by offering greater flexibility in the definition of user-defined types.
    - An XML Schema Collection allows the specification of typed xml values that conform to a collection of one or more XML schemas as well as XML syntax conventions.

There are two ways to provide principals with the permission to make requests for securables. *Use the sys.fn_builtin_permissions function to survey all the permissions that SQL Server 2005 makes available to manage requests made for securables. See books online for more information.*

- ***First, you can assign a principal to a fixed role or a user-defined role.*** As a member in a role, the principal inherits any permissions that belong to the role.
    - Fixed server roles convey server-level permissions, such as the ability to create a database or manage logins.
    - Fixed database roles convey permissions to perform actions within a database, such as create a table within a database.
    - User-defined roles allow you to specify permissions for securables within a database. You can define permissions more granularly for user-defined roles than SQL Server defines permissions for fixed database roles.

- ***Second, you can grant, deny, or revoke a permission for a securable to an individual principal or a user-defined role.*** Denying a permission prevents a user from using a securable, even if they are granted the permission through another role. Revoking a permission removes the explicit granting or denying of a permission.
    - Use GRANT, DENY, and REVOKE statements to assign specific permissions for securables to principals.
    - When you have a large number of principals for which to manage permissions, it is often simpler to setup clusters of permissions in user-defined roles and then add and drop principals from roles.
    - Use the sp_addrolemember and sp_droprolemember system stored procedures to add and drop principals from roles.

Managing Encryption

SQL Server 2005 introduces the availability of advanced encryption techniques that are relatively easy to manage and use. ***Encryption is useful for protecting selected data within a database, such as credit card numbers or other personal identity data. Encryption provides a last level of security for those who are able to pierce authentication and authorization security based on principals, securables, and permissions.***

The capability to encrypt data is based on certificates, asymmetric keys, and symmetric keys. Each type of entity for encrypting data has corresponding CREATE, ALTER, and DROP statements. You will often implement encryption in coordination with the Encryption Hierarchy.

- This hierarchy starts with a master key for the SQL Server instance. This key is, in turn, encrypted by the Windows Data Protection API. The master key for the instance is often called the service master key. The SQL Server 2005 installation process automatically generates the service master key.
- Below the service master key is the database master key. This database key is a symmetric key. When you invoke the CREATE statement for a database master key, SQL Server automatically encrypts the password for the database master key with the service master key and stores the encrypted password in the master database. The process allows a SQL Server instance to open a database master key automatically. You can override this behavior if you prefer to open the database master key manually while supplying its symmetric key password.
- You can use a database master key to encrypt certificates and asymmetric keys.
    - Certificates can, in turn, secure symmetric keys and encrypt SQL Server data.
    - Asymmetric keys can also secure symmetric keys and encrypt SQL Server data.
    - One symmetric key can secure another symmetric key in a parallel fashion to the way a certificate or an asymmetric key secures a symmetric key.
- Instead of basing certificates and asymmetric keys on the database master key, you can base them on a password. You can also secure a symmetric key on a password instead of either a certificate or an asymmetric key.
    - Using the password approach unties the encryption process from the database master key and service master key so that you can encrypt data on one SQL Server instance and decrypt it on another instance.
    - On the other hand, your database application is responsible for managing the security of the password for the certificate, asymmetric key, or symmetric key.
- There are four pairs of functions for encrypting and decrypting SQL Server data.
    - EncryptByPassPhrase and DecryptByPassPhrase encrypt and decrypt data based on a pass phrase without any reliance on a certificate, asymmetric key, or symmetric key.
    - EncryptByKey and DecryptByKey encrypt and decrypt data based on a symmetric key. The same symmetric key performs both functions, but you must open the symmetric key before you can use it.
    - EncryptByAsymKey and DecryptByAsymKey encrypt and decrypt data with an asymmetric key.
    - EncryptByCert and DecryptByCert encrypt and decrypt data with a certificate.

Chapter 2: Implement High Availability and Disaster Recovery

2.1 Implement Log Shipping

Log shipping provides a form of high availability that can make a copy of a database available on multiple computers. The server instance for a primary database in a log shipping configuration performs regular backups of database changes recorded in the log file. The secondary computers in a log shipping configuration copy and restore the backup log files to make their own local copy of the primary database. Therefore, the same, or nearly the same, database can be accessed from multiple server instances by different communities of users.

- Inserts, updates, and deletes take place on the primary computer in a log shipping configuration.
- If your query load is very heavy, you can make multiple copies of a database available for queries on different computers.
- Of course, one copy of a database can serve as a warm standby in the event of a failure to the primary database on which database changes are made.
- A major caveat is that the copies will be read only, and as such must have everything the same, including security.

Overview of Log Shipping

It is common to implement log shipping with a minimum configuration of two or three computers.

- A primary server instance making the primary database available for inserts, updates, and deletes.
- One or more secondary server instances store a copy of the primary database derived from a restored copy of backed up log files from the primary database.
- An optional third server instance, the monitor server, can issue alerts for failures and stores information critical to the log shipping process, including the history of jobs for
    - creating log backup files on the primary server
    - copying and restoring those files to the secondary (destination) servers

Figure 2.1 shows a three-computer log shipping configuration. You can have log shipping with as few as two computers. In addition, you can ship logs to more than one secondary computer.

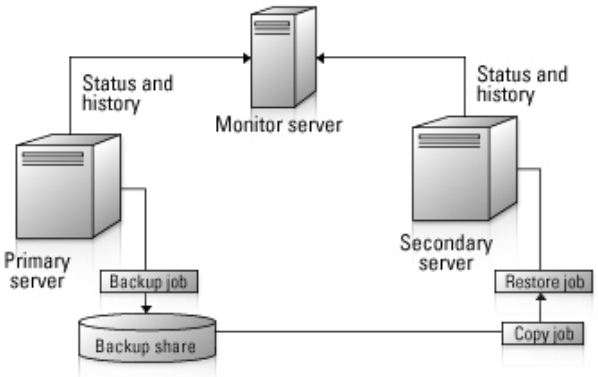


Figure 2.1. A diagram with a log shipping design and data flow.

Requirements for Configuring Log Shipping

Before you can configure log shipping, you must meet special requirements for SQL Server 2005 edition, recovery model, and security settings.

- All server instances in a log shipping configuration must be one of the following three editions:
  - Enterprise
  - Standard
  - Workgroup
- The database to be log shipped must have either a full or bulk-logged recovery model. Log shipping does not function with a simple recovery model.
  - You can display the current recovery model setting for a database with the recovery_model_desc column in the sys.databases view.
  - You can modify the recovery model for a database with the ALTER DATABASE statement.
- To configure log shipping, you must be a member of the sysadmin fixed server role on the involved instances. It is a best practice policy to restrict sysadmin role membership to database administrators who manage the servers. In addition:
  - The SQL Server service account for the primary server and the proxy account of the SQL Server Agent running the backup job must have read/write access to the backup directory.
  - SQL Server 2005 supports multiple proxy accounts that can each point at a different credential based on a name and password, such as a Windows account. By linking a proxy account to a login, you can specify permissions for resources outside of SQL Server, such as files and their folders in a Windows directory.
  - The SQL Server service account for each secondary server and the proxy account for the copy job must have read permission for the backup directory and write access to the copy directory.

    - The SQL Server service account for each secondary server and the proxy account for the restore job must have read/write access to the copy directory.

Configuring two or more SQL Server instances for log shipping requires the following kinds of steps:

- Choose the servers holding the primary and secondary databases.
    - Recall that you can have two or more instances storing independent copies of the primary database.
    - The use of a monitor computer is optional.
- Create a file share for transaction log backups. Microsoft recommends storing the backup files on a fault-tolerant computer that is not the primary server, one of the secondary servers, or the monitor server.
- Choose a backup schedule for the log of the primary database.
    - Schedule backups more frequently for primary databases with a heavy load of database changes.
    - Minimizing the time between backups reduces your exposure to potential loss of data.
- On each secondary computer designate a folder for storing copied log backup files.
    - Schedule copy and restore jobs for the first secondary server.
    - Schedule copy and restore jobs for all other secondary servers.
    - A single secondary server can participate in multiple log shipping configurations for different primary databases.
- Optionally, configure a monitor server.

How-to Log Shipping Topics

You can enable log shipping with either

- SQL Server Management Studio (SSMS)
- specialized system stored procedures that you run from T-SQL batches

If you are using SSMS to configure log shipping, you can start by opening the Database Properties dialog box for the database that will serve as the primary database. Next, select the Transaction Log Shipping page. This page has five sections or parts. It is mandatory that you complete the first three sections.

- In the top section, which has no section title, click a check box to enable a log shipping configuration for the primary database.

- In the second part, whose title is Transaction log backups, click the Backup Settings button to make the backup settings for the transaction logs on the primary server.
    - You need to setup the share folder for backup files before you can complete this section.
    - *Figure 2.2* shows the Database Properties dialog box for setting up log shipping for a database, such as the AdventureWorks database, after completing the settings for backing up the transaction logs of the primary database.
- In the third part, titled Secondary databases, you can add secondary databases running from secondary servers and specify the operation of copy and backup jobs. Clicking Add in the third section opens the Secondary Database Settings dialog box with three tabs.
    - On the first tab, you can initialize a secondary database based on the primary database.
    - On the second tab, you can specify the operation of the copy job.
    - On the third tab, you can specify the operation of the restore job.
- In the fourth section, labeled Monitor server instance, you can designate a monitor server instance and specify how it performs (for example, if it runs an alert job).
- The final section, which has no section title, lets you automate the scripting and saving of your settings for subsequent editing or re-use.

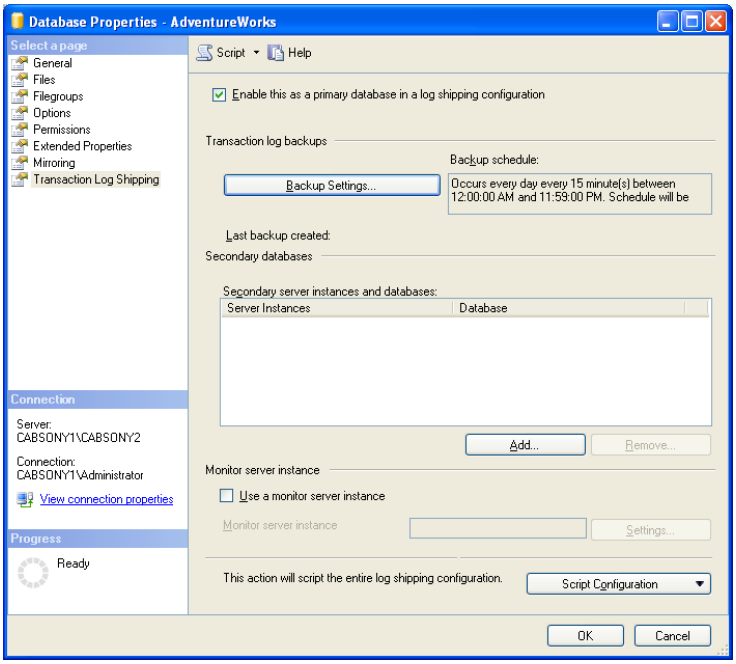


Figure 2.2. The AdventureWorks Database Properties dialog box for enabling log shipping.

Configuring and managing log shipping is an advanced topic with many detailed issues to master. You can drill down further into the SSMS and T-SQL approaches to these tasks from the msdn2 site.

- Use the following URL for additional details on conceptual issues:

  http://msdn2.microsoft.com/en-us/library/ms190016.aspx

- Drill down further on the implementation issues from the following URL:

  http://msdn2.microsoft.com/en-us/library/ms188625(SQL.90).aspx

2.2 Implement Database Mirroring

Database mirroring, introduced with SQL Server 2005, is a new way to implement high availability for a database. This high availability technique complements more traditional approaches, such as log shipping.

- ***One especially distinguishing feature of database mirroring in comparison to log shipping is that database mirroring readily supports automatic failover to the backup database copy, which is called a mirror database.***

  - Automatic failover is achieved with the aid of a witness server instance that continually assesses the availability of the principal server instance and the mirror server instance. When the witness can no longer connect to the database on the principal server, database mirroring automatically switches roles so the mirror database becomes the principal database.
  - You can also manually switch the roles of the principal and mirror databases.

- ***If you need to have multiple backup copies of a database available, log shipping is superior to database mirroring because database mirroring manages only one copy of a database.***

Three Database Mirroring Modes

Database mirroring operates in any of three modes. These modes are summarized in Table 2.1. There are two features that define the modes. The principal and mirror databases can work together to copy log records synchronously or asynchronously. When principal and mirror databases operate synchronously, the principal database does not commit a database change to a client application until after the mirror database confirms it copied the associated log record sent from the principal database. In asynchronous operation, the principal database confirms an update to a client application after merely sending the corresponding log record to the mirror database.

- The **high availability mode** operates synchronously with a witness server instance.
- The **high protection mode** operates synchronously but without a witness server instance.
- The **high performance mode** operates asynchronously. This mode can operate with or without a witness server instance.

Mode Name	Synchronous or Asynchronous?	Witness Present?	Comments
High availability	Synchronous	Yes	• Supports automatic failover • If the mirror database goes offline, the principal database can stay online provided it communicates with the witness server instance
High protection	Synchronous	No	• Supports manual role switching between principal and mirror databases • If the mirror database becomes unavailable, the principal database goes offline
High performance	Asynchronous	Yes/No	• Delivers fast performance by not forcing the principal database to wait for feedback from the mirror database • Do not use a witness with high performance mode because there can be no automatic failover • Best for applications that require high performance or have a narrow bandwidth between principal and mirror databases

Table 2.1 Database Mirroring Modes

Database Mirroring Requirements and Best Practices

Database mirroring has special requirements that must be met for its successful implementation. Some of the most prominent requirements are:

- The principal and mirror databases must run either Enterprise or Standard editions of SQL Server 2005. You cannot however mix editions and mirror from an Enterprise Edition instance to a Standard instance or vice versa.
- The witness server instance can run any of the following editions: Enterprise, Standard, Workgroup, or even Express (saving the need to pay for a license fee). Recall that the witness does not have to store a database.
- All server instances supporting the mirroring of a database should use the same master code page and collation.
- The principal database must have a full recovery model. Bulk-logged and simple recovery models cannot be used for database mirroring.
- All logins for connecting to the principal database must reside on server instances for both the mirror database and the principal database.

Several database mirroring best practices are not strictly required. Microsoft especially recommends the following best practices.

- Whenever possible, use a dedicated network interface card (NIC) for database mirroring applications.
- Have the principal, mirror, and optional witness server instances communicate with each other over a dedicated network or a wide-bandwidth network connection with plenty of unused capacity. This recommendation applies especially when using database mirroring in the high availability mode.
- If you wish to implement a database mirroring session in high availability mode:
    - Start first with a high performance mode to verify your network can support database mirroring.
    - Next, switch to synchronous operation by converting your configuration to a high protection mode without automatic failover.
    - Finally, implement automatic failover only after successfully testing the synchronous mode of operation with the high protection mode.

Creating Endpoints for Database Mirroring

Database server instances communicate with one another via database mirroring endpoints. You can think of an endpoint as a container for a computer port. The port allows traffic into and out of the computer. ***Each endpoint has an identifier that you specify with a CREATE ENDPOINT statement.*** Each port has a unique number for a computer. You can also manage endpoints with ALTER ENDPOINT and DROP ENDPOINT statements.

- ***The set of server instances (principal, mirror, and optionally witness) for a database comprise a database mirroring session.***
- Each server instance within a database mirroring session requires its own database mirroring endpoint. In fact, a server instance can have just one mirroring endpoint (there are other types of endpoints besides database mirroring endpoints).
- Any one server instance can participate in more than one database mirroring session, but all database mirroring sessions on a server instance must use the same database mirroring endpoint. A server instance can participate in different database mirroring sessions via different roles, such as a principal in one session, a mirror in another session, and a witness in a third session.
- The sys.database_mirroring_endpoints view contains one row for the database mirroring endpoint on a server instance. Its columns reveal
    - The endpoint identifier
    - The role description of the endpoint (PARTNER for principal or mirror, WITNESS for witness, and ALL for principal, mirror, or witness)
    - The type of connection authentication through the endpoint
    - The type of encryption for a connection through the endpoint

The CREATE ENDPOINT statement for database mirroring with Windows authentication can be defined with just three clauses.

- ***One clause indicates that the endpoint is started.*** This is the normal way to specify an endpoint, but other states include stopped and disabled.
- ***Another clause designates a protocol***, which always has to be TCP for database mirroring, and a port number (nnnn).
- ***The final clause specifies the role of the endpoint in database mirroring sessions*** (for example, ROLE=PARTNER).

The following sample CREATE ENDPOINT statement is suitable for specifying an endpoint on a server instance that participates in database mirroring sessions as either a principal database server or a mirror database server. The port number is 1234. The port number must be unique on any one computer system. However, you can re-use the same port number on another computer system. Use the sys. tcp_endpoints view to identify the currently used endpoints on a computer system. If you want to specify an endpoint that participates only as a witness, then re-write the role expression to ROLE=WITNESS. If you want an endpoint that can participate in database mirroring sessions as a principal, mirror, or witness, then re-write the role expression to ROLE=ALL.

```
CREATE ENDPOINT my_mirroring_endpoint
        STATE = STARTED
        AS TCP (LISTENER_PORT = 1234)
        FOR DATABASE_MIRRORING (ROLE=PARTNER)
```

For additional details on the CREATE ENDPOINT statement for database mirroring navigate to http://msdn2.microsoft.com/en-us/library/ms190456.aspx

Configuring Database Mirroring Sessions

You can implement a database mirroring session with five steps. Each step can have one or more elements that address one or more databases. Any one step can span multiple databases because database mirroring, and high availability in general, inherently involves multiple databases, which are typically on server instances running from different computers.

1. ***Create a database mirror if one does not exist already.***

    a. Make sure the principal database has a full recovery model.

    b. On the principal server instance, perform a full database backup and copy the backup to the mirror server instance.

    c. On the mirror server instance, create the mirror database by running the RESTORE DATABASE statement using a WITH NORECOVERY clause for the copied backup. Assign the same identifier to the mirror database and the principal database.

    d. Also, backup your transaction log on principal server instance, and restore the

resulting backup on the mirror server instance.

e. Repeat step d just before you implement a database mirroring session to synchronize the principal and mirror databases.

2. ***Create endpoints for all server instances participating in the database mirroring session.*** The text under the "Creating Endpoints for Database Mirroring" heading describes the process. You can use endpoint configuration to manage authentication and securing data transfers between servers. See the following URL for valuable details on database mirroring transport security: http://msdn2.microsoft.com/en-us/library/ms186360.aspx

3. ***Set the principal database on the principal server instance as a partner to the mirror database on the mirror server instance with the ALTER DATABASE statement..*** The syntax for the statement is:

```
ALTER DATABASE database_name
    SET PARTNER = server_network_address.
```

a. The database_name parameter is the name of the database to be mirrored. This database name must be the same on the principal and mirror server instances.

b. The server_network_address parameter precisely defines the principal server instance by specifying a server address and a port number.

i. A server address designates a computer system by its computer name, fully qualified domain name, or an IP address.

ii. The port number is the same one used to define an endpoint for the principal server instance.

iii. Therefore, you can designate a server_network_address for a principal server instance as TCP://PrincipalComputerName:instance_port_number.

iv. See the following URL for additional details on syntax and naming conventions for server network addresses: http://msdn2.microsoft.com/en-us/library/ms189921.aspx

4. ***Set the mirror database on the mirror server instance as a partner to the principal database on the principal server instance.***

a. You again use the ALTER DATABASE statement to achieve this task, except you initially connect to the principal database on the principal server instance.

b. In addition, your network server address should point at the port number for the endpoint on the mirror server instance in the database mirroring session.

5. ***By default, a database session starts in high protection mode (synchronous operation with no automatic failover). However, you can override the default session so that the database mirroring session runs in either high performance mode or high availability mode.***

a. To run in high performance mode, connect to the principal server instance and run the following statement: ALTER DATABASE database_name SET PARTNER SAFETY OFF.

   i. Database_name is the name of the principal and mirror databases; remember both databases must have the same name.

   ii. The SAFETY database option specifies whether a database mirroring session runs synchronously or asynchronously. The OFF setting is for asynchronous operation. The ON setting, which is the default SAFETY setting, causes a database mirroring session to run synchronously.

b. To run in high availability mode (with automatic failover), you need to perform two tasks.

   i. Add a database mirroring endpoint to the server instance that you want as your witness in the database mirroring session.

   ii. On the principal server instance, specify the server network address and port number for the witness server instance. For example, invoke: ALTER DATABASE database_name SET WITNESS = TCP:// WitnessComputerName: instance_port_number. The database_name parameter references the name of the principal and mirror databases.

   iii. Aside from specifying an endpoint on the witness server instance as described previously, there is nothing more to do on the witness server instance.

Use the following URL to drill down deeper into implementing a database mirroring session with T-SQL: http://msdn2.microsoft.com/en-us/library/ms190430.aspx. Instead of using T-SQL to configure a database mirroring session, you can implement database mirroring with SSMS. Learn more about the SSMS approach from the following URL: http://msdn2.microsoft.com/en-us/library/ms175134.aspx

2.3 Manage Database Snapshots

Database snapshots help you implement high availability by preserving the content at the time that you create a snapshot.

- Database snapshots are read-only.
- While you cannot add new data or revise and delete the existing data in a database snapshot, you can make changes to the source database for a database snapshot.
- Any one source database can have multiple snapshots each created at different points in times.
- Typical applications for database snapshots include
  - Database copies for queries, such as those in a quarterly report.
  - A resource for restoring a database modified incorrectly (for example, by inadvertently dropping a vital table).
  - Making a mirror database from a database mirroring session available for queries as well as backing up a principal database.

Database Snapshot Requirements and Operating Features

You can only create database snapshots for a production environment with the Enterprise edition of SQL Server 2005. In addition to this edition requirement, there are several other limitations for the source database of a database snapshot.

- Database snapshots must reside on the same server instance as their source database.
- As long as a source database has at least one database snapshot, an administrator cannot drop, detach, or restore the source database.
- Files cannot be dropped from a source database with one or more database snapshots.
- The source database for a database snapshot must be online unless the source database is a mirror database in a database mirroring session.
- Full-text catalogs in a database source cannot populate database snapshots.
- In a log shipping configuration, you can make database snapshots for the primary database, but not any of the secondary databases.

Database snapshots rely on sparse files and a copy-on-write operation. Sparse files are a feature of the NTFS file system.

- The sparse file(s) for a database snapshot initially contain no data and takes up very little space.
    - The source database will have one sparse file for each file that the database has.
    - This does not include log files or full text index files.
- As a database snapshot's source database changes, original data pages are copied to the database snapshot's sparse file(s). This process is called copy-on-write operation.
- A data page is the basic file storage unit within a data file for SQL Server databases. Each data page has a size of 8 KB.
- Subsequent changes after an initial change to the content within a source database do not affect the contents of the database snapshot's sparse file(s).

Because database snapshots rely on sparse files, they can save a lot of space versus a full copy of a database. Queries for a snapshot database read unchanged data pages from source database files and the original version of changed data pages from the database snapshot's sparse file(s). As changes to the source database affect more and more data pages, the size of a database snapshot's sparse file(s) grows. Therefore, ***database snapshots deliver their best space-saving value for scenarios where there are likely to be no or few changes to the source database***, such as the database for a product catalog that changes just once per quarter. In this case, you can generate one database snapshot for each quarterly catalog source database.

When searching for additional information about database snapshots, it is helpful to know that database snapshots are unrelated to snapshot isolation for transactions, snapshot replication, and even snapshot backups. Two useful URLs with good coverage of database snapshot conceptual and implementation

issues appear below:

http://msdn2.microsoft.com/en-us/library/ms175158.aspx

http://msdn2.microsoft.com/en-us/library/ms190467.aspx

Creating Database Snapshots

The only way to create a database snapshot is with the CREATE DATABASE statement in T-SQL as there is no user interface in SSMS.

- Microsoft recommends you assign an identifier for a database snapshot that is similar to the source database name with some extra information to indicate when the database snapshot was generated. For example, if the source database identifier is source_name, its database snapshot identifier might be source_name_2007_01_01.
- When using the CREATE DATABASE statement to generate a database snapshot, you must specify a special AS SNAPSHOT OF clause. Designate the source database identifier as the argument for the clause.
- In the ON clause for the CREATE DATABASE statement, you must specify each logical data file name from the source database. Log files are not specified as part of a database snapshot.

The following CREATE DATABASE statement demonstrates the syntax for creating a snapshot for the AdventureWorks database. The database snapshot identifier denotes a snapshot created on June 22, 2006. The AS SNAPSHOT OF clause indicates that the database snapshot is for the AdventureWorks sample database. The NAME argument specifies the logical data file name from the source database. The FILENAME argument denotes the file system name for a sparse file supporting the database snapshot. You can derive the logical and file system names for the data files in a source database with the sys.database_ files view. The ss file extension for the database snapshot data file is not required, but is a best practice so that they will stand out as to their purpose.

```
CREATE DATABASE AdventureWorks_2006_06_22
ON (
NAME = AdventureWorks_Data, FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\ (more)
Data\AdventureWorks_Data_20060622.ss')
AS SNAPSHOT OF AdventureWorks
```

Client applications can connect to database snapshots in the same way that they do to any other read-only database. Database snapshots inherit permissions, such as read permission for a table, from the source database. To explore a database snapshot from SSMS, connect to the database engine serving the database snapshot. Then, expose Database Snapshots within Databases. Finally, select the database snapshot that you wish to explore from Database Snapshots.

Reverting to a Database Snapshot

Reverting a source database to a database snapshot can be a fast and easy way to restore a database. Reverting overwrites source database data pages with the data pages in the database snapshot sparse file(s). This process can be faster and easier than restoring from database backup files. Instead of having to restore a whole database, you merely write the data pages that changed back to the source database from the database snapshot.

Several restrictions apply to reverting a source database to a database snapshot.

- The source database cannot contain any read-only or compressed data files.
- No files can be offline in the source database that were online when the database snapshot was created.
- A source database can have only one database snapshot when you revert a source database to a database snapshot.
- Reverting to a database snapshot breaks the log backup chain. Therefore, you should perform a full database backup after reverting to a database snapshot.

You revert a source database to a database snapshot with the RESTORE DATABASE statement. You need only two parameters for the statement.

- ***Identify the name of the source database you want to restore as the argument for the RESTORE DATABASE phrase.***
- ***Assign as a string value denoting the database snapshot name to the* DATABASE_SNAPSHOT parameter**.
- The following example illustrates the syntax for reverting the AdventureWorks database to the AdventureWorks_2006_06_22 database snapshot.

```
RESTORE DATABASE AdventureWorks from
     DATABASE_SNAPSHOT = 'AdventureWorks_2006_06_22'
```

Dropping a Database Snapshot

The more repeated changes made to column values in tables, the more obsolete a database snapshot becomes. In addition, database snapshots grow in size as you change more content in a source database. Even if you do prefer to retain selected database snapshots to maintain some history for a database, you will eventually need to drop other database snapshots to conserve storage space on the device holding the source database.

You can drop a database snapshot with the DROP DATABASE statement, just like dropping a regular database. Include in the statement the database snapshot identifier.

Chapter 3: Supporting Data Consumers

3.1 Retrieving Data with Queries

Many database administrators support communities of users who require data from one or more databases. As a database administrator, you may occasionally write a short query or a collection of queries to help clients use your databases or as a model for clients learning how to write their own custom queries. You can develop the code for a query with T-SQL in a query tab of SQL Server Management Studio (SSMS) or graphically using the query designer either directly in a query window or by launching a new view.

Opening and Saving Query Files

There are several ways to open a new query tab, including:

- **Right-click a server instance in Object Explorer** within SSMS to open a menu with a New Query item. Click the item to open a new query tab with a connection to the default database for the current user.
- Open the Databases item for a server instance in Object Explorer and right-click the New Query **item for a database**. This opens a new query tab with a connection to the database that you right-clicked.
- Open a new querytab by **clicking the New Query control on the Standard toolbar**. The connection for the query tab will be to a user's default database if the selected item is not for a specific database or an item within a database. Otherwise, the connection will be for the selected database.

The tab for a new query receives an arbitrary name based on its order of creation within a session, such as SQLQuery1.sql for the first query tab. After entering T-SQL code to specify a query, you can save the query with any file name you prefer, but it is common to use a .sql extension for query files. Later, the file can be reopened to be used again, modified, or used to start a new query tab that includes some code from the previously saved query tab. Management Studio also has a Solution Architecture that is inherited from Visual Studio that allows you to group together queries, database connection files, etc.

Designing a New Query

You can enter a SELECT statement in the new query tab to retrieve values from a database. In the following script, the USE statement changes the connection context to the AdventureWorks database. GO is a batch separator which makes previous statements execute in a different batch than subsequent ones. The SELECT statement

- Returns all rows from vSalesPerson view in the Sales schema
- Nests a string expression between the SalesPersonID and EmailAddress column values; the string expression and its alias
    - Computes the full name for sales persons by concatenating FirstName, MiddleName, and LastName column values and substituting a zero-length string for a null MiddleName column value
    - Assigns a column header of Full name to the computed column

```
USE AdventureWorks
GO

SELECT SalesPersonID,
  FirstName + ' ' +
  ISNULL(MiddleName + ' '. '') +
--add the space in the ISNULL to avoid extra space
  LastName AS [Full name],
  EmailAddress
FROM Sales.vSalesPerson
```

You can also generate a T-SQL statement for a query graphically by either using the query designer in the Query Design Query in Editor menu or by starting to create a new view. The graphical view designer makes it particularly easy to join multiple data sources. For those uncomfortable with T-SQL query statements, the graphical designer makes it easy to tweak a statement and quickly see the outcome. I will demonstrate the view editor, as it has extra functionality, but works much like the query designer.

Figure 3.1 shows four panes that result from choosing the New View item from the right-click menu for the Views folder within a database. These menu choices open the graphical query designer:

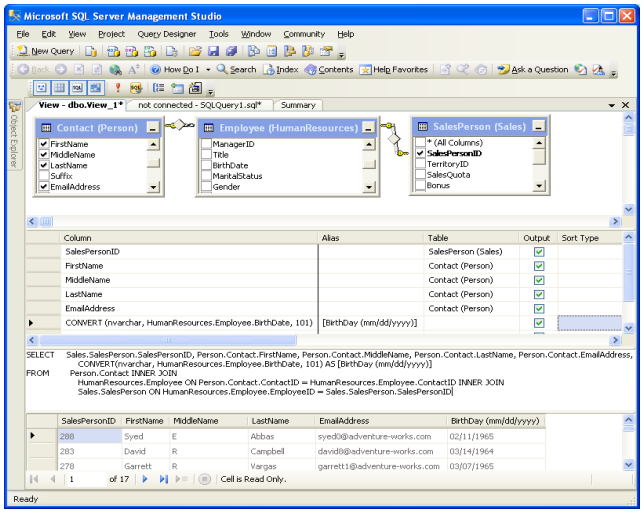


Figure 3.1. Use the New View menu item to graphically create a query statement.

- The top Design pane shows three tables that have been added to the pane (the choice to add tables will be the first thing you see when you start a new view).
- Notice the check marks next to selected column names. Selecting an item adds it to the Criteria pane below the Design pane.
- The final row in the Criteria pane, which is the second pane, includes an expression that uses the CONVERT function to represent the BirthDate datetime column value from the Employee table as an nvarchar type with dates in an mm/dd/yyyy format.

- The graphical manipulation of the top two panes automatically generates the SELECT statement that appears in the SQL pane, which appears below the Criteria pane.
- Clicking the Execute SQL control (!) populates the bottom Results pane.

After designing a new view, ***you can re-use the SELECT statement in one of two ways.*** You can ***save the view and re-use*** it like the vSalesPerson view. You can also ***cut and paste the graphically generated SELECT statement into a query tab***. Then, you can run, save, and modify the query from the file for the tab.

Formatting Retrieved Values

The view displayed in *Figure 3.1* demonstrates one way to reformat with the CONVERT function a datetime value as an nvarchar type. Without the CONVERT function, the Birthdate column value would display both date and time parts.

You can utilize the CONVERT function for reformatting date values. ***The CAST function is much like the CONVERT function, though CAST is ANSI (American National Standards Institute) compatible while CONVERT is not. CONVERT also lets you take advantage of special SQL Server date conversion features.*** You can learn more about the capabilities of the CONVERT and CAST functions as well as general conversion issues from the following URL: http://msdn2.microsoft.com/en-us/ms187928.aspx

Data can be formatted in a SELECT clause in most any way you imagine by using the concatenation operator (+) in conjunction with all of the string functions, math functions, and many other functions. For more reading on SQL Server functions the following URL will get you started: http://msdn2.microsoft.com/en-us/library/ms190642.aspx. Typically however, it is considered the presentation layers job to do most of the formatting of data for presenting to the user.

Managing Collation Issues

Collations govern how SQL Server represents and compares character values. Collations typically reflect language, culture, and locale. It is possible for two server instances with different collation settings to not decode binary values to characters consistently or to sort and compare character values differently across the two servers. SQL Server instances and client performance can be affected by inconsistent collations between a server instance and its clients. For these reasons, Microsoft recommends that organizations

- ***Adopt a standard collation setting for all server instances*** that need to communicate with one another.
- Unicode character data types use a common character set for all countries with modern languages, such as worldwide variants of English, non-English European, Far Eastern, and Arabic languages.
- Whenever deploying server instances that many need to store values from multiple langages, ***use Unicode (nchar, nvarchar, ntext) instead of non-Unicode (char, varchar, text) data types***. Note that Unicode characters are 16 bits rather than 8 so they take up twice the space as ANSI characters.
- ***Use Unicode character sets on a server and its clients***
    - If that is not possible then use the same code page on a server instance and its clients.
    - A code page is an ordered set of characters with a corresponding set of numeric or code point values.

SQL Server 2005 supports collation settings at the server, database, table, column, and expression levels. The SQL Server 2005 setup wizard automatically selects a collation that you can override. Unless you need to modify the default setting to match the collation of another server, or for some other critical reason, do not alter the default collation setting during setup. After installation of a server instance, you cannot readily modify the collation setting for a server instance, but you can easily modify the collation settings for databases within a server instance and tables within a database. Unless there is an overriding collation assignment for an object, the collation setting for a server instance filters down to all appropriate objects within it in the same way that a collation setting for a database or table apply to all appropriate objects within them. *(Note that once an object is created in one collation, changing the database collation will not change the object's collation. It has to be changed individually for all previously existing objects).*

There are three main types of collations that you can select.

- Windows collations are based on the Windows locale setting for the Windows operating system on a computer.
    - Windows collations specify such issues as the character set, the sort, and the code page for non-Unicode data.
    - Sort order can be affected by sensitivity to such issues as case, accent, and type of Kana characters for the Japanese language.
    - Windows collations are especially well suited for communicating between SQL Server 2005 instances.
- Binary collations sort data on a simplified set of rules relative to Windows collations that is more closely tied to the code point value for characters (characters will sort on their binary representation, not the value of their characters!) As a result, you can achieve faster performance with binary collations in comparison to Windows collations. There are two types of binary collations.
    - The classic binary collation type with a _BIN suffix in its name is suitable for a SQL Server 2005 server instance that needs to communicate with an instance based on an earlier version of SQL Server.
    - The new binary collation type with a _BIN2 suffix in its name is exclusively intended for use between SQL Server 2005 instances.
- SQL collations primarily target backward compatibility with versions prior to SQL Server 2005. You can use SQL collations instead of Windows collations, but you should choose a SQL collation that is compatible with the Windows operating system locale setting. You must use a SQL collation if:
    - You are performing replication with prior versions to SQL Server 2005
    - Your application depends on collations from previous versions of SQL Server

A detailed discussion of collation settings is available from the msdn2 site with collation names, functions, and syntax for setting collations. The following URL provides an overview with links for digging deeper into the topic: http://msd2.microsoft.com/en-us/library/ms143503.aspx

3.2 Manipulating Relational Data

There are multiple ways to modify the data in a relational database.

- ***The T-SQL statements for modifying the contents of a table include INSERT, UPDATE, and DELETE***. These statements modify the data and are always fully logged in the transaction log. As mentioned in the previous chapter, it is this transaction log that is used with database mirroring sessions and with log shipping configurations.
- Data modification statements can sometimes fail (for example, by trying to insert a row to a table that is missing a required column value). The failure can generate an error or an exception, which causes a script to fail. ***You can process these run-time errors and even recover in some cases by using Try and Catch blocks***.

INSERT, DELETE, and UPDATE Statements

The ***INSERT statement can add one or more rows to an existing data source***. The data source is typically a table or a view that permits the entry of new rows to its underlying data source. Before denoting a target data source, you can optionally use the INTO keyword. A clause of the SELECT statement,the INTO clause, permits you to create a new table with the results of the SELECT statement. *This variant of the SELECT statement is commonly referred to as SELECT INTO*

The INSERT statement has a rich format that enables you to use it many different ways. The full syntax specification for using the INSERT statement in all its variations is at http://msdn2.microsoft.com/en-us/library/ms174335.aspx. The following script shows a pair of INSERT statements adding new rows to an existing table. The script creates the table before inserting new rows to it.

- The script starts by referencing the PrepLogic database created in section 1.3 of Chapter 1; you can substitute any other database you prefer
- Next, the script
    - Drops any previously existing table named tab1 in the dbo schema.
    - Adds the tab1 table to the database with three columns – one of which is a primary key with an IDENTITY property (an automatic feature that automatically creates a new unique value for each new row.)
- Two subsequent INSERT statements each add a row to the tab1 table
    - The items in parentheses after the table name denote the order in which column values are specified
    - The VALUES keyword prefaces a list of column values for the new row
    - The values appear in parentheses at the end of the statement
    - ***There is no need to designate a value for a primary key column with an IDENTITY property because the SQL Server database engine automatically assigns this value***

- The final SELECT statement returns all the column values for all the rows

```
USE PrepLogic
GO

IF OBJECT_ID('dbo.tab1') IS NOT NULL
        DROP TABLE dbo.tab1
GO

CREATE TABLE tab1 (
pk_col int NOT NULL IDENTITY PRIMARY KEY,
FirstName nvarchar(10),
LastName nvarchar(12)
)

INSERT INTO dbo.tab1 (LastName, FirstName) VALUES ('Dobson', 'Rick')
INSERT INTO dbo.tab1 (LastName, FirstName) VALUES ('Hill', 'Glen')

SELECT * FROM dbo.tab1
```

The following listing shows the output from running the preceding script. Each INSERT statement generates a message about a single row being affected. The SELECT statement returns tab1's set of column values. I have turned off the messages about the number of rows affected using SET NOCOUNT ON, which is a common practice.

```
pk_col      FirstName  LastName
----------- ---------- ------------
1           Rick       Dobson
2           Glen       Hill
```

You can remove one or more rows from a table with a DELETE statement. It is common to use a WHERE clause in a DELETE statement to specify the rows that should be deleted. Specify a criterion expression that denotes one or more rows from a table. The following script drops the row with FirstName and LastName column values of Rick and Dobson and replaces them with a new row of column values (Rickie and Dobson).

```
DELETE dbo.tab1
WHERE FirstName = 'Rick' AND LastName = 'Dobson'

INSERT INTO dbo.tab1  (LastName, FirstName) VALUES ('Dobson', 'Rickie')

SELECT * FROM dbo.tab1
```

The concluding SELECT statement in the preceding script generates the following result set. Notice that Rickie Dobson has a different pk_col value than in the initial INSERT script for data modification statements. When you delete a row, it is no longer available. The only way to replace the deleted row is to add a new row *(your exact results may be different if you caused any errors. SQL Server does not reuse identity values).*

```
pk_col      FirstName  LastName
----------- ---------- ------------
2           Glen       Hill
3           Rickie     Dobson
```

The preceding sample demonstrates one approach to revising a column value – namely, delete the old row and replace it with a new row containing the revised column value. However, a better approach to this is to use ***the UPDATE statement***, which ***is designed explicitly to change one or more column values in one or more rows*** Updating the row:

```
UPDATE dbo.tab1
SET FirstName = 'Rickie', LastName = 'Bobson'
WHERE pk_col = 3
```

The following listing shows the result set for the tab1 table after the UPDATE statement runs. With the UPDATE statement, you can modify one or more column values in a row without deleting the old row and adding a new one with the revised column values. Execute the SELECT statement again, and you will see that the row with pk_col = 3 is still there with a new value for Last Name:

```
pk_col      FirstName  LastName
----------- ---------- ------------
2           Glen       Hill
3           Rickie     Bobson
```

Handle Exceptions

SQL Server 2005 introduces ***a Try…Catch construct that simplifies and enhances exception handling***. An exception is an error that occurs at run time. The range of possible errors is nearly infinite, for example, a user may input an incorrect value, or there might be some sort of hardware issue.

If you are familiar with error processing in any .NET language, you can leverage that knowledge to understand the SQL Server 2005 Try…Catch concept. Microsoft did not implement the construct exactly as it is implemented in Visual Basic or C#. Instead, the SQL Server 2005 Try…Catch construct reflects T-SQL coding conventions.

The Try…Catch construct is implemented with two blocks: a Try block of code followed by a Catch block of code. You can run the two blocks of code from a T-SQL batch, a stored procedure, or a trigger. Two keywords mark the beginning and end of each block of code:

```
BEGIN TRY
   <code to attempt>
      END TRY
      BEGIN CATCH
          <error handling>
      END CATCH
```

If an error occurs as the code within a Try block runs, control passes to the matching Catch block. A matching Catch block for a Try block should occur within the same unit of code and immediately behind its matching Try block. For example, you cannot place the Try block in one SQL batch and its matching Catch block in another SQL batch.

SQL Server 2005 offers a variety of functions for you to return information about the error in a Try block that transfers control to a Catch block. You may be able to use these functions to recover from an error. Some error numbers can occur for more than one reason. In these cases, the error state uniquely identifies the cause of an error.

- ERROR_NUMBER() returns the error number.
- ERROR_MESSAGE() returns the complete error message.
- ERROR_SEVERITY() returns the error severity.
- ERROR_STATE() returns the error state.
- ERROR_LINE() returns the line number within the SQL batch, stored procedure, or trigger that caused the error.
- ERROR_PROCEDURE() returns the name of the stored procedure or trigger from which the Try block ran.

The following code segment illustrates several coding conventions for using a Try…Catch construct. This code segment is designed to run immediately after the preceding sample that inserts and modifies two rows in the tab1 table in the dbo schema of the PrepLogic database. The code fills an initial gap in the pk_col values left by the deleted row.

- Within the Try block, an INSERT statement attempts to insert a new value for the pk_col column within the tab1 table.
    - However, since the pk_col has an IDENTITY property setting SQL Server cannot routinely perform this task.
    - As a consequence, the INSERT statement generates an error.
- The code between BEGIN CATCH and END CATCH illustrates two kinds of activities that you can perform within a Catch block.
    - First, a pair of SELECT statements returns values that describe the error. This information can be helpful in an error report for the database administrator responsible for fixing the problem. The second SELECT statement passes back the error message in a separate result set to make it easy to read.

- Second, the code dynamically recovers from an error with a number of 544. The code toggles the value of IDENTITY_INSERT. By setting this value to ON, T-SQL code can assign a value to a column with an IDENTITY property. The default IDENTITY_INSERT value is OFF so that SQL Server can exclusively manage the values in a column with an IDENTITY property.

```
BEGIN TRY
    INSERT INTO tab1 (more)
(pk_col, FirstName, LastName) VALUES (1, 'Joey', 'JoeJoe')
END TRY
BEGIN CATCH
        SELECT ERROR_NUMBER() 'ERROR_NUMBER',
               ERROR_SEVERITY() 'ERROR_SEVERITY',
               ERROR_STATE() 'ERROR_STATE'
  SELECT CAST (ERROR_MESSAGE() AS nvarchar(100)) 'ERROR_MESSAGE'
IF ERROR_NUMBER() = 544
  BEGIN
    SELECT 'FIXED UP 544 error'
    SET IDENTITY_INSERT dbo.tab1 ON
    INSERT INTO dbo.tab1 (more)
(pk_col, FirstName, LastName) VALUES (1, 'Joey', 'JoeJoe')
    SET IDENTITY_INSERT dbo.tab1 OFF
  END
END CATCH

SELECT * FROM dbo.tab1
```

The following listing provides an excerpt of the output from the preceding script.

- Notice that the first listing of column values from the tab1 table contains just two rows. This listing is from a SELECT statement that occurs after the first two rows are inserted into tab1.
- After the listing of tab1 column values, you see feedback describing the error. The 544 error number describes an error that occurs when a user tries to insert a value into a column with an IDENTITY for a table with its IDENTITY_INSERT value set to OFF, which is the default setting.
- The last result set shows the column values in tab1 after the Catch block code recovers from the error and inserts a new row with an explicit column value.

```
   pk_col      FirstName  LastName
----------- ---------- ------------
2           Glen       Hill
3           Rick       Dobson


ERROR_NUMBER ERROR_SEVERITY ERROR_STATE
```

```
------------ -------------- -----------
544          16             1

ERROR_MESSAGE
------------------------------------------------------------
Cannot insert explicit value for identity column in table …
------------------
FIXED UP 544 error
pk_col       FirstName  LastName
------------ ---------- ------------
1            Joey       JoeJoe
2            Glen       Hill
3            Rick       Dobson
```

3.3 Managing XML Data

XML functionality is greatly expanded in SQL Server 2005 versus SQL Server 2000. This section introduces you to the topic by discussing conceptual issues and presenting code samples for selected topics. The XML topic with SQL Server 2005 is exceptionally rich. You will therefore want to pay special attention to exploring the URLs offered as starting points for further study.

Structure of XML Data

SQL Server 2005 allows XML data to be either XML documents or XML fragments. XML data represents values with XML element and attribute values. Elements can nest hierarchically within an XML document, and attributes can be thought of as properties of an element instance. You can designate elements within a document by a matching pair of tags or a single tag denoting no need for an ending tag. The beginning root element tag occurs once before any other tag element, and the ending root element tag is always at the end of a document. You can include a declaration at the top of an XML document indicating the version to which the XML conforms and other pertinent information as attributes. An XML fragment is a subset of an XML document that contains no root element. The following is a simple XML document.

```
<?xml version = "1.0">
<myroot>
<mydataelement>My data element value. </mydataelement>
<dataelementwithattributes attr1="value1" attr2="value2" />
</myroot>
```

An xml data type, introduced with SQL Server 2005, stores values in a binary format that saves storage space and enables you to process xml data type values in ways similar to traditional SQL Server data types.

For example, you can

- ***declare a column in a table as an xml data type*** and populate the column with XML documents or fragments
- ***pass arguments to a stored procedure as an xml data type value***
- ***assign xml data type values to variables***

An xml data type can be either typed or untyped. Using typed XML data values allows you to specify the format of xml data type instances. An untyped XML data value allows more flexibility in the format of individual xml data type instances.

- ***A typed xml data type value has at least one XML schema associated with it***.
  - There are new CREATE, ALTER, and DROP statements for creating XML schemas.
  - Use the CREATE XML SCHEMA COLLECTION statement to specify a new schema.
  - An XML schema specifies the structure of an xml data type value in the same way that a relational schema defines the structure of a relational database.
- Typed and untyped xml data type values are different in their need to conform to a XML schema.
  - A typed xml data type value must be valid XML data and valid with respect to any XML schemas associated with the xml data type.
  - An untyped xml data type just must be valid XML data.

The xml data type is a very rich topic that is likely to grow in importance. Those seeking more extensive coverage of the xml data type can drill down on it with the help of the following resource:

http://msdn2.microsoft.com/en-us/library/ms189887.aspx

Retrieve XML Data Instances

One of the advantages of an xml data type is that you can query XML data along with relational data in a single SELECT statement. The sample AdventureWorks database provides an excellent resource for learning how to query xml data type values. It includes a collection of XML schemas used to specify column values in a variety of different tables. You can learn more scope of XML resources and examples in the AdventureWorks database from the following URL and by following the page's links:

http://msdn2.microsoft.com/en-us/library/ms191513.aspx

The following script demonstrates how to retrieve relational and XML column values with a single query statement. After establishing a connection to the AdventureWorks database, the script specifies a SELECT statement for the ProductModel table in the Production schema. The Production schema is a relational schema – not an XML schema. Within the ProductModel table, the Name column contains relational data, and the CatalogDescription column contains typed XML data. You can use Object Explorer in SSMS to derive the data types for XML as well as traditional relational columns. In addition, Object Explorer also denotes the XML schema for typed xml data type columns. For example, you can learn that the schema

name for the CatalogDescription column is ProductDescriptionSchemaCollection.

```
USE AdventureWorks
GO

SELECT Name, CatalogDescription
FROM Production.ProductModel
WHERE CatalogDescription IS NOT NULL
```

If you run the preceding script (sending the results to a grid,) SSMS populates the Results tab with six rows of data. The first column shows the Name column values from ProductModel rows with non-NULL values for CatalogDescription. The CatalogDescription column includes hyperlinks for xml data type column values. Clicking any hyperlink opens a window for viewing the content of the whole xml data type instance.

Figure 3.2 shows a slightly edited version of the window that opens after a click for the first row in the result set from the preceding query. The editing is to format the namespaces in the ProductDescription element as well as to make hierarchical elements for the Specifications element to be viewable without scrolling. Other content is clipped at the right border, but you can scroll right to view it from within SSMS.

- The ProductDescription element is the root element for the xml data type instance.
    - The element specifies several XML namespaces with an xmlns prefix. These namespaces define specific kinds of content that can appear in a CatalogDescription data type instance.
    - The ProductDescription element also includes a couple of assignments for ProductModelID and ProductModelName attributes.
- The Specifications element has numerous hierarchical elements below it, including one element for the material of a product. A subsequent query will demonstrate how to reference this specific element within a CatalogDescription XML data instance.

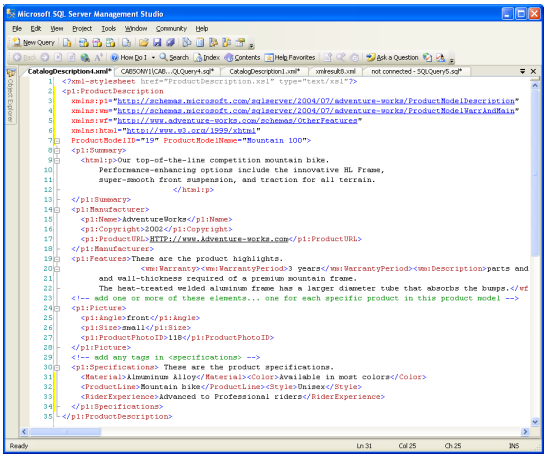


Figure 3.2. An xml data type instance from the CatalogDescription column in the ProductModel table within the Production schema of the AdventureWorks database.

Querying with XML Data Type Methods

The xml data type offers four methods to query the contents of a specific instance. With these four methods, you can return or process a subset of the contents within an XML data instance. ***These functions allow you to work with XML data instances more precisely than simply returning a whole XML data instance for viewing.***

Table 3.1 summarizes the four methods that you can use to query xml data type values. You can apply these methods to untyped as well as typed values, but the syntax changes slightly. Specifically, you have to designate the XML schema stored in an XML Collection object when you are querying a typed xml data type instance.

Method name	Comments
query()	Specifies an XQuery expression for a source xml data type to return an xml data type which is a subset of the original xml data type instance.
value()	Allows the extraction of a relational data type value from an xml data type value. You specify as arguments an XQuery expression that returns a value and a relational data type, such as int, that represents the returned value.
exist()	Returns 1, 0, or NULL based on an XQuery expression passed to the method. If the function finds XML content satisfying the expression, the return value is 1. If no XML content satisfies the expression the return value is 0 unless the original xml data type value contains a NULL.
nodes()	Facilitates the shredding of an xml data type value into relational column data. This method deprecates the OPENXML function from SQL Server 2000.

Table 3.1 Four Query Methods for the XML Data Type

The following SELECT statement illustrates the use of the query() and exist() methods with a typed XML instance.

- The WITH XMLNAMESPACES statement before the SELECT statement specifies an XML schema for use with the exist() method. The statement designates PD as a prefix for the schema.
- The SELECT list includes the Name column from the ProductModel table of the Production schema in the AdventureWorks database.
- The query() method in the SELECT list returns a Product element that has a ProductModelID attribute. This is the XML returned by the method. An expression within the query extracts the ProductModelID attribute for the ProductSpecification element within the XML instance for a row.
- The WHERE clause shows the syntax for an exist() method that examines the CatalogDescription XML instance on a row to assess if the instance contains a Specifications element. This method's expression in the WHERE clause also implicitly omits rows in the ProductModel table that have a

NULL CatalogDescription column value. The exist() expression uses the PD prefix denoted in the WITH XMLNAMESPACES statement.

```
WITH XMLNAMESPACES ('http://schemas.microsoft.com/sqlserver/
2004/07/adventure-works/ProductModelDescription' AS PD)
SELECT Name, CatalogDescription.query('
    <Product ProductModelID= "{ sql:column("ProductModelID") }" /> ')
AS [Product Model ID]
FROM Production.ProductModel
WHERE CatalogDescription.exist('/PD:ProductDescription[(PD:Specifications)]') = 1
```

The preceding script generates the following listing. Notice that it shows just the name and ID values for a product model. The Name column value derives from familiar relational query syntax. The query() method returns a much more concise xml data type value than the one appearing in *Figure 3.2*. Because the XML output generated by the query() method is so concise relative to returning a whole XML instance value, it is much easier to scan return values from multiple rows. For example, you do not need to click a hyperlink to see the XML return values in an SSMS query tab.

```
Name           Product Model ID
-------------- --------------------------------
Mountain-100   <Product ProductModelID="19" />
Mountain-500   <Product ProductModelID="23" />
Road-150       <Product ProductModelID="25" />
Road-450       <Product ProductModelID="28" />
Touring-1000   <Product ProductModelID="34" />
Touring-2000   <Product ProductModelID="35" />
```

The xml data type is much richer than relational data types. The ProductModel table contains a separate column each for ProductModelID and ProductModelName values although both these values also exist in the XML for the CatalogDescription value. In addition, there are numerous element node and attribute values in the CatalogDescription XML instances that do not appear as columns in the ProductModel table. As you can see from *Figure 3.2*, the Specifications element has below it several hierarchical elements, including one named Material.

The result set for the following SELECT statement returns one row for each row in the ProductModel table that has a non-NULL CatalogDescription column value. The syntax for using the value() method in the statement indicates how to extract the Material element node value that is nested within the Specifications element. In this application of an xml data type method, there is not a preceding WITH XMLNAMESPACES statement. Instead, the namespace declaration of the XML schema for the CatalogDescription column values appears as an argument for the value() method. A path expression queries the Material element within the Specifications element within the ProductDescription element. The number in brackets after each element name designates a sequential order for selecting an element. For example, if a schema allowed for multiple occurrences of an element instance within an XML instance,

you can denote which element instance you want by the number in brackets after an element name.

```
SELECT ProductModelID, Name,
                CatalogDescription.value('
declare namespace PD="http://schemas.microsoft.com\ (more)
sqlserver/2004/07/adventure-works/ProductModelDescription";
/PD:ProductDescription[1]/PD:Specifications[1]/ (more)
Material[1]', 'nvarchar(350)')

FROM Production.ProductModel
WHERE CatalogDescription IS NOT NULL
ORDER BY Material DESC
```

You can drill down further on the xml data type methods and the XQuery syntax with the following two URLs: http://msdn2.microsoft.com/en-us/library/ms190798.aspx and http://msdn2.microsoft.com/en-us/library/ms190262.aspx

Modifying XML Data

The modify() method for the xml data type facilitates fine-grained manipulation of XML data. The manipulation includes insertion, deletion, and replacement. Using the modify() method permits you to change the individual element node and attribute values. The three case-sensitive XQuery keywords that facilitate data manipulation of XML data values are:

- insert
- delete
- replace value of

You can use the XQuery keywords for data manipulation with variables and column values. Within these contexts, you can use the keywords with typed as well as untyped XML instances. The syntax is distinct for each keyword and in addition, you have to adapt the syntax for use with namespace references that denote XML schemas when you are modifying typed XML instances. The use of XQuery keywords within the modify() method is known as XML Data Manipulation Language (XML DML).

The next several scripts illustrate the setup for and use of the modify() method. The first script in the group creates a table named T in the AdventureWorks database. The table has two columns. One of these columns, x, has an xml data type. Because there is no explicit reference to an XML schema collection in the row being inserted, the xml data type is untyped. The x column value has a root element named ProductDescription and a hierarchical element named Specifications. While the ProductID and ProductName attributes describe the root element, there are no hierarchical elements nested within the Specifications element in the XML instance yet.

```
USE AdventureWorks;
GO
IF EXISTS(SELECT * from sys.tables WHERE name = 'T')
DROP TABLE T
GO
CREATE TABLE T (i int, x xml);
GO
INSERT INTO T VALUES(1,'
<ProductDescription ProductID="19" (more)
ProductName="Mountain 100">
<Specifications></Specifications>
</ProductDescription>
')
GO
```

When you invoke the modify() method, you must do it from a SET clause of an UPDATE statement or on an XML variable as SET @xmlVar.modify(). The modify() method applies to the column name, which is x in the example. In the following statement, the modify() method adds a nested element to the Specifications element that designates the type of material for the product.

```
UPDATE T
SET x.modify('insert <Material>Aluminum</Material>
                     into (/ProductDescription/Specifications)[1]')
```

Don't forget that XML commands are case sensitive. Change 'into' to INTO and the statement will fail!

The following short script returns the first specification element within the XML instance (at this point, there is a single specification.) The listing below the script shows the arrangement of elements.

```
SELECT x.query(' /ProductDescription[1]/Specifications[1]')
FROM T

<Specifications>
  <Material>Aluminum</Material>
</Specifications>
```

The next script shows the code for adding a new element that appears before the Material element. The "as first" phrase denotes that the Color element should appear as the first nested element within the Specifications element. Therefore, if we ran the preceding script that shows the Specifications elements and its nested elements after the following UPDATE statement, the Color element appears before the Material element.

```
UPDATE T
SET x.modify('insert <Color>Most colors</Color> as first
  into   (/ProductDescription/Specifications)[1]')
GO
```

Let's say that you did not want the Color element appearing before the Material element, but you wanted the Color element to appear after the Material element. One way of altering the XML instance is to delete the Color element, and then add it again after the Material element. The following script with two UPDATE statements shows how to accomplish this. The first UPDATE statement uses the delete keyword, and the second UPDATE statement shows the use of the insert keyword with "after" instead of "as first" for its keyword phrase.

```
UPDATE T
SET x.modify (more)
('delete /ProductDescription/Specifications/Color[1]')
GO
UPDATE T
SET x.modify('insert <Color>Most colors</Color>
  after (/ProductDescription/Specifications/Material)[1]')
GO
```

You may encounter a situation where you need to alter an element's node value. Use the replace value of phrase within the modify() method to achieve this outcome. The following script shows the syntax to change the Color element node value to 'New colors' (notice from the preceding script that its initial value was 'Most colors').

```
UPDATE T
SET x.modify('
  replace value of (more)
(/ProductDescription/Specifications/Color[1]/text())[1]
  with     "New colors"
')
```

You may also develop a need to add an attribute to an existing element, such as the Color element. The following script illustrates how to add an attribute to an element and then show the whole XML instance from its root element. ***Notice that you specify an attribute's value, such as Brightness, within braces ({ }) and using double quotes.*** The argument for the query method specifies the XML instance's root element – ProductDescription. As the result set listing below the query listing shows this formulation returns the whole XML instance.

```
UPDATE T
SET x.modify(
'insert attribute Brightness {"High"}
 into   (/ProductDescription/Specifications/Color)[1]'
)
GO
SELECT x.query(' /ProductDescription[1]')
FROM T
GO
```

```
<ProductDescription ProductID="19" ProductName="Mountain 100">
  <Specifications>
    <Material>Aluminium</Material>
    <Color Brightness="High">New colors</Color>
  </Specifications>
</ProductDescription>
```

3.4 Importing and Exporting Data from a File

When copying large amounts of data into or out of a SQL Server instance, you can gain a substantial performance advantage by using bulk copy techniques. The performance advantage comes at the expense of the recoverability of changes to your database. Data copied to a database with a bulk technique does not support point in time recovery when a database has a bulk-logged recovery model instead of a full recovery model. Transactions are however minimally logged so that you can restore them using by restoring a log backup. With a full recovery model, bulk operations are fully logged. This section starts with a brief review of how to manage the recovery model setting for a database in connection with bulk copy techniques. The section features a tutorial presentation on three bulk copy techniques with in-depth coverage for the most flexible approach.

Setting the Bulk-logged Recovery Model

It can be beneficial to switch to a bulk-logged recovery model (bulk mode) when copying large amounts of data into a database. This is especially the case when you do not require point in time recovery (best practice is to do a back up after a bulk operation.) The bcp utility as well as the INSERT … FROM OPENROWSET(BULK …) and BULK INSERT statements can cause SQL Server to enter its bulk mode. ***The bulk-logged recovery model in comparison to the full recovery model offers two primary benefits:***

- ***Data importing can operate faster.***
- ***The storage requirement for the log files will not grow as large.***

You can switch between recovery models with the ALTER DATABASE statement, and you can determine the current recovery model for a database with either the sys.databases view or the DATABASEPROPERTYEX function.

- The syntax for switching to the bulk-logged recovery model requires two statements
  - ALTER DATABASE Database_name
  - SET RECOVERY Bulk_Logged
- The recovery_model and recovery_model_desc columns in the sys.databases view provide feedback about the current status of a database's recovery model setting
  - The recovery_model column denotes a bulk-logged recovery model with a tinyint value of 2
  - The recovery_model_desc column denotes a bulk-logged recovery model with a value of 'BULK_LOGGED'

- Using the DATABASEPROPERTYEX function with type of 'recovery' returns BULK_LOGGED when the denoted database has a bulk-logged recovery model: SELECT DATABASEPROPERTYEX('Adve ntureWorks','Recovery')

The bcp command-line utility

The bcp utility is a command-line program that facilitates transferring data between a table or a view in a SQL Server database and an external data file. The bcp utility works with views tied to a single table. Because the bcp utility runs in bulk mode, it is especially well suited for copying large amounts of data into or out of a SQL Server instance. You can use the bcp program with any amount of data. SQL Server 2005 enhances this traditional bulk copy utility.

The bcp program ships and installs with SQL Server. ***There are four distinct kinds of tasks that the bcp utility enables.*** These tasks are:

- ***Export data from a table or view to a file***
- ***Export data from a query to a file***
- ***Import data from a file into a table or view***
- ***Generate format files to facilitate the copying of data between a table or view and a file***

As is typical of command-line utilities, you control the operation of the bcp program with a collection of program settings some of which you specify with switches, such as –switch_name optionally followed by switch_argument_value. You use these program settings to specify the name of the table and the file to use in copying data. You also must specify login credentials and some information about the format of the data in both the table and the file.

The samples demonstrating the operation of the bcp utility (as well as the other two bulk copy techniques) work a table we will call Shippers in the PrepLogic database. Use the following script to create the the Shippers table in the PrepLogic database based on the table with the same name in the Northwind sample database. Because the criterion (1 = 2) is never true, the SELECT INTO statement creates the schema for the Shippers table, but never populates the table with column values.

```
USE PrepLogic
GO

If EXISTS(SELECT * FROM sys.tables
        WHERE name = 'Shippers' and schema_id = 1)
        DROP TABLE dbo.Shippers
GO

SELECT * INTO dbo.Shippers
FROM Northwind.dbo.Shippers
WHERE 1 = 2
```

The following command-line example copies rows from the Shippers table in the Northwind database to the Shippers.dat file.

```
bcp Northwind.dbo.Shippers out c:\Shippers.dat -T -c –Sserver_name\instance_name
```

- Notice that the first argument after the bcp program name is the three-part name of the SQL Server table providing the data that gets copied to Shippers.dat. The parts of the name are database_name.schema_name.table_name.
- The next argument, out, indicates that data flows from the table to the data file. Other keywords that can replace the argument are
    - in -- when data flows from the data file to the table
    - queryout -- when you want to specify a SELECT statement as the source for data to export to a data file
    - format -- when you want to create a format file to simplify how to define the way table column values precisely map to fields in a file; format files are for use with another bcp command-line statement or even other bulk copy statements
- The third argument specifies the name of the target file, which the bcp program can either create from scratch or overwrite (if it already exists.) In this case I am putting it in the root of C:\, but you should generally locate the files in some safer location.
- The last three arguments are switches
    - –T indicates that you are connecting with the Windows account for the current user; you can alternatively specify -U for a SQL Server login and let the utility prompt for a password or even designate the password with the –P switch
    - –c specifies the use of the char data type as the storage type for field values in the file; you can use other switches to specify alternative data types for the fields in a file
    - –S denotes the SQL Server instance; if you do not explicitly specify a particular server instance with the –S switch, the utility uses the default SQL Server instance

The next command-line sample copies the data copied to the Shippers.dat file in the preceding bcp statement to the Shippers table in the PrepLogic database.

```
bcp PrepLogic.dbo.Shippers in c:\Shippers.dat -T -c -E -Sserver_name\instance_name
```

- The first argument specifies the target table to receive data from the file
- The second setting, in, indicates that data flows into the table from the file
- The third setting is the file name
- There are four switch arguments in this use of the bcp program
    - The –T, –c, and -S arguments serve the same role as in the preceding example
    - –E indicates that the SQL Server instance should not automatically generate new identity column values, but it should instead accept values from the data file. If you

omit the –E switch, SQL Server automatically generates identity column values, which may be different from those in the source data file

There are numerous other ways to apply the bcp utility. For example, if you are going to use a format file, it is common to create the format file with one run of the utility. Then, in second and subsequent runs of the bcp utility, you can reference the previously created format file. The following bcp statement generates a format file for the Shippers table previously created:

```
bcp PrepLogic.dbo.Shippers format nul -T -c -f c:\Shippers.fmt
                                -Sserver_name\instance_name
```

- The format file is for the Shippers table in the dbo schema of the PrepLogic database.
- Notice that format setting appears in place of the in and out settings used with earlier bcp examples.
- The format file has the name Shippers.fmt, which is in the traditional style of prior SQL Server versions. SQL Server 2005 also enables the creation of a new XML-based format style with a slightly modified collection of settings.
- Other settings, such as those discussed for the two preceding bcp examples, allow you to refine how a format file operates in a bulk copy task.

Other Bulk Import Techniques

While the bcp utility has its own API for SQL Server, the BULK INSERT statement and the bulk rowset provider for the OPENROWSET function work from T-SQL. Both of the complementary bulk techniques to the bcp utility offer a subset of bcp functions. When you may not need the full functionality enabled by the bcp technique, these alternative bulk copy techniques may help simplify achieving your objective.

- The BULK INSERT statement is a T-SQL statement that is exclusively for copying values from a data file to a table.
- The OPENROWSET function bulk rowset provider allows you to query the field values in a data file with a SELECT statement, and you can also populate a table from a file by using the SELECT statement as part of an INSERT statement.

You can use both the BULK INSERT statement and the OPENROWSET function bulk rowset provider with format files. However, ***the only way to generate a format file is with the bcp utility.***

The following script starts with a DELETE statement to clear the Shippers table in the PrepLogic database. Then, the script invokes a BULK INSERT statement which illustrates how to use the format file created in the previous section by the bcp statement.

```
DELETE dbo.Shippers

BULK INSERT PrepLogic.dbo.Shippers
FROM 'c:\shippers.dat'
WITH (FORMATFILE = 'c:\shippers.fmt', KEEPIDENTITY)
```

- The identifier in the BULK INSERT clause specifies the name of the table or view into which you want to copy data.
- The FROM clause argument designates the path and name of the data file.
- The WITH clause accepts optional parameters that help you control how the BULK INSERT statement operates.
    - The FORMATFILE parameter specifies the path and file name for the format file; its use is optional in this very simple example.
    - The KEEPIDENTITY parameter allows the identity field values in a file to populate a column with an IDENTITY property (instead of the default behavior where the server instance automatically generates IDENTITY column values).

The next script shows the OPENROWSET function using its bulk provider to query the values in a file. The following SELECT statement returns a result set just like any other SELECT statement for a table or view in a database.

```
SELECT talias.*
FROM    OPENROWSET (BULK 'c:\Shippers.dat',
FORMATFILE = 'c:\shippers.fmt') AS talias
```

This bulk operation does not insert the data from a file into a server instance, but it instead queries the values within the data file directly. You must designate an alias for the FROM clause argument, which is talias in the script. The FORMATFILE parameter, which references the previously generated format file, allows the parsing of the field values on each row within the Shippers.dat file to column values in the result set. If you do not specify the FORMATFILE parameter, then you must designate a parameter for returning each row as a single string or binary value.

You can learn more about the operation and settings for the BULK INSERT statement and OPENROWSET bulk provider from the following two URLs:

http://msdn2.microsoft.com/en-us/library/ms188365.aspx, and http://msdn2.microsoft.com/en-us/library/ms190312.aspx

3.5 Other Related Topics with Resources

Two other major data consumer topics merit your attention as you prepare for the 70-431 certification examination.

- The first topic is native XML web services
    - Provides a means for exposing database objects, such as stored procedures, as XML web services
    - ***The SQL Server 2005 version implements a native XML web service with http endpoints instead of requiring Internet Information Services as was true of SQL Server 2000***
    - Use the following URL as a starting point for drilling down further on native XML web services: http://msdn2.microsoft.com/en-us/library/ms191274.aspx
- Service Broker is a new part of the Database Engine introduced by SQL Server 2005
    - Service Broker facilitates both single-instance and multi-instance SQL Server applications
        - ***With single-instance solutions, Service Broker provides an asynchronous programming model for faster interactive response times and better overall throughout***
        - ***With multi-instance solutions, Service Broker adds to asynchronous communication the exchange of encrypted messages over TCP/IP connections between pairs of server instances***
    - Service Broker is a very rich topic that can integrate administrator, developer, and architect activities
    - Use the following URL and the links it exposes to discover more about Service Broker: http://msdn2.microsoft.com/en-us/library/ms166043.aspx

Chapter 4: Maintaining Databases

4.1 Manage Database Indexes with T-SQL

An index for a table or view in a database is similar to the index for a book. Look up a term in the index to see all the pages on which the term appears. This is almost always a lot faster than going through each page in a book to scan for a term. Similarly, a table index can cut the search time to find matching values in a table.

Many database tables stay in a constant state of change in that new rows are being inserted and existing rows are being updated and deleted. As these changes are being made to the tables and corresponding indexes, the indexes will become fragmented, which is to say stored in a less than optimal state. Fragmentation reduces the search time savings for an index by causing SQL Server to work harder to use the index. You can quantify the amount of fragmentation with index fragmentation statistics and

then take actions to remedy index fragmentation. Reducing or eliminating index fragmentation can substantially improve performance.

Creating and Viewing Indexes

An index contains keys based on values from one or more columns in a table or view. There are two types of relational indexes: clustered and nonclustered. A ***clustered index stores the rows for the table or view sorted on its key values (commonly referred to as the clustering key). You can only have one clustered index on a table. Tables without a clustered index are stored in an unordered heap instead of a sorted order.*** Nonclustered indexes are completely separate structures from a table's data rows.

Both clustered and nonclustered indexes can be unique – that is point at a single row and all values for the key columns must be different for every row. A unique index is automatically created when you add a PRIMARY KEY or a UNIQUE constraint to a table. A table or a view can have up to 250 indexes.

SQL Server indexes are based on B-trees, which are common structures that database indexes are built upon. Each page for an index is a node in the B-tree. The top node is called the root node. The bottom nodes are called leaf nodes. Any levels between the root and leaf nodes are referred to as intermediate levels. For more reading on b-tree and SQL Server index structures, check out:

http://en.wikipedia.org/wiki/B-tree

http://msdn2.microsoft.com/en-us/library/ms177443.aspx

http://msdn2.microsoft.com/en-us/library/ms177484.aspx

You add a new index to a database with the CREATE INDEX statement. In fact, there are actually three variations of the CREATE INDEX statement.

- One version has a syntax that supports new features introduced with SQL Server 2005, such as indexes for very large databases that spread tables across multiple partitions.
- A second variant of the statement is suitable for creating keys for XML columns within a table.
- Another version has a syntax that is compatible with selected features and syntax for prior SQL Server versions.

As with other types of CREATE statements, there are matching ALTER and DROP statements for indexes when you need to change or remove indexes.

In the Enterprise (and Developer) editions, tables and indexes with multiple partitions are supported. Partitions are particularly useful for simplifying the maintenance and improving the availability of very large databases. A partition is a user-defined unit of data organization. What data goes on what partition is defined by a partition function that splits data by ranges All indexes in SQL Server 2005 reside on at least one partition, but you can explicitly assign an index to multiple partitions. By default, a database has a single partition associated with its single filegroup. For databases with just one partition, all indexes are stored in the single filegroup. In a multi-partition database, you are likely to have two or more partitions for a database with each partition having an associated filegroup. When a table is spread across multiple partitions, different groups of rows can be mapped to partitions that, in turn, map to separate filegroups. (More reading on partitions can be found here:

http://msdn2.microsoft.com/en-us/library/ms189051.aspx)

The main catalog view for getting index information is sys.indexes. This view contains one row for each index of a table (a heap or clustered table), indexed view, or table-valued function (with an index on the output) in a database You can query this view to retrieve such items as name, index_id, type, parent object_id for the indexes in a database.

- The object_id is the object id value of the object to which the index belongs.
- The type column indicates the type of index (HEAP, CLUSTERED, NONCLUSTERED)
- The index_id will be 0 for a heap, 1 for clustered (index_id = 1) and between 2 and 250 for the nonclustered indexes.
- The name is the index's identifier, such as the one you assign with the CREATE INDEX statement.
- Other system views that provide useful subsets of information about views include sys. partitions, sys.index_columns, and sys.key_constraints.

Assessing Index Fragmentation

Recall that for indexes to expedite searches, it is essential to minimize index fragmentation. After their creation, indexes become fragmented in the process of adding, deleting and moving around the content in a database. Index fragmentation occurs when the logical order of the index leaf values does not correspond to the order of the table pages on a physical storage unit.

The sys.dm_db_index_physical_stats function allows you to monitor the fragmentation of the indexes in a database. This function returns a row set with one row for each B-tree level in each partition of an index. You can reference the sys.dm_db_index_physical_stats function in the FROM clause of a SELECT statement. The function requires you to specify five arguements:

- database_id:
  - A database_id number for information about a specific database
  - A value of NULL, 0, or DEFAULT for information about all the databases on a server instance
  - DB_ID() for the database_id number of the current database context (set by a USE statement)
- object_id: of the table or view for which to monitor fragmentation:
  - An object_id number for information about a specific object within a database
  - A value of NULL, 0, or DEFAULT for information about all the tables and views in the database designated by the first argument
- index_id:
  - A value from 1 through 250 to correspond to the index_id value for a table or view
  - A value of NULL, -1, or DEFAULT for information about all indexes for the tables or views designated by the first two arguments

- partition_number: for use with multi-partition databases and lets you specify a particular partition for which to monitor fragmentation. You can designate the partition_number of a particular partition or NULL, which causes the computation of fragmentation statistics about all partitions for the indexes specified by the third argument. The sys.partitions view can return the partition_number values associated with the indexes in a database.
- mode: specifies the scan level mode that the function uses to monitor fragmentation. Use one of three modes that determine the level of scanning used to monitor index fragmentation.
    - LIMITED scans the fewest pages, and returns a result set most quickly.
    - SAMPLED scans just 1% of the index pages or 9,999 pages, whichever is greatest, to return a result set.
    - DETAILED scans all the pages for an index to return a result set.

The sys.dm_db_index_physical_stats function contains columns for identifying the indexes referred to as well as various fragmentation/index statistics. Columns denote the database, object, index, and partition about which the function returns fragmentation statistics. Three columns that help you assess the fragmentation of an index include avg_fragmentation_in_percent, fragment_count, and avg_fragment_size_in_pages.

- A fragment_count of 1 indicates no index fragmentation; the greater the value of fragment_count the more fragmentation there is.
- In general, you want the fragment_count as close to 1 and the avg_fragment_size_in_pages to be as large as possible.
- Microsoft issued rough guidelines for assessing index fragmentation based on the value of avg_fragmentation_in_percent. Whenever possible, you should supplement the rough guidelines with specific tests for your data to assess any improvement in query performances using indexes with different levels of index fragmentation.
    - Values below 5 or 10 percent do not typically require any action to reduce index fragmentation.
    - Values greater than 10 percent but less than 30 percent can benefit from index reorganization.
    - Values at or above 30 percent typically benefit from the rebuilding of an index.

The following script shows the syntax for returning the index_id, index name, and avg_fragmentation_in_percent for the indexes in the Product table of the Production schema of the AdventureWorks database. The FROM clause joins the sys.dm_db_index_physical_stats function with the sys.indexes view. The join is necessary to return the index names along with the fragmentation statistics.

```
USE AdventureWorks
GO

SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats
(DB_ID(),
OBJECT_ID(N'Production.Product'),
NULL,
NULL,
NULL) AS a
JOIN sys.indexes AS b ON a.object_id = b.object_id AND
a.index_id = b.index_id
```

The following listing shows the output from the preceding SELECT statement. According to the rough Microsoft guidelines, the only index that does not require the indexes to be rebuilt is the PK_Product_ ProductID index. Because the fragmentation percent for this index is between 10 and 30 percent, it would be sufficient to simply reorganize the index rather than rebuild.

```
index_id name                      avg_fragmentation_in_percent
-------- ------------------------- ----------------------------
    1         PK_Product_ProductID     23.0769230769231
    2         AK_Product_ProductNumber 50
    3         AK_Product_Name          66.6666666666667
    4         AK_Product_rowguid       50
```

Remedying Index Fragmentation

Two primary techniques let you remedy index fragmentation.

- ***Reorganizing one or more indexes is a less resource intensive and faster means of recovering from some of the effects of index fragmentation.*** Use reorganizing when attempting to recover from mild index fragmentation.
- ***Rebuilding is a more thorough means of recovering from index fragmentation, but it takes longer.*** Use rebuilding to recover from more severe cases of index fragmentation.
- You can implement either of these techniques with the ALTER INDEX statement. There is a clause (REORGANIZE or REBUILD) for each way to remedy index fragmentation.

The ALTER INDEX statement makes it possible for you to reorganize one or all indexes for a table or view. The ALTER INDEX with the REORGANIZE clause replaces the DBCC INDEXDEFRAG statement from prior versions of SQL Server. The reorganizing process

- reorders the physical leaf-level pages to match the logical order of the leaf nodes for an index
- compacts the index pages which can reduce the total number of pages for an index
- compacts large object data types (image, text, ntext, varchar(max), nvarchar(max), and varbinary(max)) in indexes and their underlying table

- is always done online so that the process does not interfere with querying a table or view whose indexes are being defragmented

There are two ways to rebuild an index: the ALTER INDEX statement with the REBUILD clause and the CREATE INDEX statement with the DROP_EXISTING clause. The rebuilding process actually drops the prior version of an index and replaces the old index version with a newly built one with the same name and settings (unless you override the prior index settings). Lookup the CREATE INDEX and ALTER INDEX statements at the msdn2 site for descriptions of index settings.

- The ALTER INDEX statement with a REBUILD clause
  - is especially designed for rebuilding an index from scratch without having to know the index structure
  - can also rebuild all the indexes for a table or view in a single statement
  - performs most index rebuild tasks online for minimal blocking of query tasks
  - replaces the DBCC DBREINDEX statement
- The CREATE INDEX statement with a DROP_EXISTING clause differs from the ALTER INDEX statement with a REBUILD clause in the following ways
  - Can specify new columns and drop prior columns from participating in an index
  - Can change an index from nonclustered to clustered (this process must take place offline)
  - Can process only one index in a single transaction
  - Cannot rebuild a single partition index

The following statement shows the syntax for reorganizing the PK_Product_ProductID index in the Product table within the Production schema of the AdventureWorks database. Replace the PK_Product_ProductID identifier with the ALL keyword to reorganize all the indexes for the Product table. Then, replace the REORGANIZE clause with a REBUILD clause to rebuild, instead of just reorganize, the indexes for the table.

```
USE AdventureWorks
GO

ALTER INDEX PK_Product_ProductID ON Production.Product
REORGANIZE
```

4.2 Recovery Models

SQL Server offers three recovery models to control database backup and restoration scope. The models trade off varying degrees of flexibility in the ability to backup and restore databases versus administration skill and resource consumption. For example, a database's recovery model can determine the ways in which the transaction log assists with backing up and restoring databases or whether bulk transfers to a database restore automatically. See section 3.5 in Chapter 3 for a discussion of bulk transfer techniques.

- ***The simple recovery model places the least demands on the system and the database administrator. This model allows you to restore a database to the most recent full or differential backup.*** After automatic checkpoints, which you can generate by performing a backup, inactive portions of the transaction log are made available for re-use. In the event of a database restoration forced by a database becoming unusable, you lose data entered into the database after the last backup.
- ***The full recovery model allows you to restore a database to any point in time, including the point of failure. So long as you can recover your transaction log to the point of failure, you can fully restore a database to that point in time.*** In order to perform a database recovery to a point in time, you need to restore the log. This, in turn, requires at least one full database backup and a complete series of transaction log backups subsequent to the full database backup. You can optionally trail a full database backup with a one or more differential backups and then a series of log backups. When recovering from a database failure, the last log series item should be a tail-log backup, which is the backup of the log from the last normal backup until the point of failure.
- ***The bulk-logged recovery model is a special case of the full recovery model. This special case explicitly targets recovering from a database failure where the database has one or more large bulk transfers into it.*** When running with a bulk-logged recovery model, bulk transfers cannot be recovered from the transaction log as they can with a full recovery model. In exchange for this benefit, the transaction log is much smaller when using a bulk-logged recovery model. On the other hand, the bulk-logged recovery model does not permit intermediate point-in-time recoveries – instead you can only recover a database to its point of failure.

You can think of a recovery model as a database option setting. You can retrieve the current recovery model setting for a database from the recovery_model and recovery_model_desc columns of the sys.databases view. The recovery_model and recovery_model_desc columns have different data types but corresponding values.

- The recovery_model data type is tinyint, and the recovery_model_desc data type is nvarchar(60).
- The corresponding values assign numeric and text values to denote the three recovery model types.
    - 1 and FULL represent the full recovery model.
    - 2 and BULK_LOGGED represent the bulk-logged recovery model.
    - 3 and SIMPLE represent the simple recovery model.

You can use the ALTER DATABASE statement with its SET RECOVERY clause to change the current recovery model setting for a database. As usual, immediately follow ALTER DATABASE with the database's identifier. Terminate the SET RECOVERY clause with the recovery_model_desc setting

corresponding to the recovery model that you want to assign to a database.

The sample AdventureWorks database ships with a simple recovery model. The following script shows the syntax for retrieving the recovery model for the database and then changing it to a full recovery model. If you run this script and you want to keep the AdventureWorks sample database consistent with its initial settings remember to use the ALTER DATABASE statement again to restore a simple recovery model setting (SET RECOVERY SIMPLE).

```
SELECT recovery_model, recovery_model_desc
FROM sys.databases
WHERE name = 'AdventureWorks'

ALTER DATABASE AdventureWorks
SET RECOVERY FULL
```

4.3 Backup a Database

There are two primary T-SQL statements for backing up your data. Each has corresponding statements for restoring data as well.

- ***BACKUP DATABASE allows you to backup up a database fully or in one of several partial ways.***
    - With this statement, you can create a backup file for all or a subset of data files/ filegroups supporting a database.
    - You will be able to use database backup files to restore all database files or a subset of database files.
- ***BACKUP LOG allows you to backup the transaction log for a database.***
    - This statement is only allowed when the recovery model is full or bulk-logged.
    - The set of transaction log backup files complement base database backup files.
    - You can use log backup files along with a base database backup file to restore a database to any point in time after the time you created the base database backup file.

BACKUP DATABASE and BACKUP LOG statements can operate concurrently with many T-SQL statements, such as SELECT, INSERT, UPDATE, and DELETE statements. Therefore, backing up a database does not interfere with many other typical database activities. However, you cannot manipulate the database files or transaction log files. For example, the following statements do not run while a backup is in process:

- ALTER DATABASE with either ADD FILE or REMOVE FILE clauses
- DBCC SHRINKDATABASE or DBCC SHRINKFILE

BACKUP DATABASE Statement

There are several ways that you can use the BACKUP DATABASE statement. Two very common ways are for a full database backup or for a differential backup that stores changes relative to a previous full or differential backup. Both of these backup types back up data from all the data filegroups, but you can also selectively backup a subset of data filegroups or files.

- Full database backups can backup all database files, including part of the transaction log.
- A partial backup can occur faster and require less storage than a full backup. A partial backup is for
  - the primary data filegroup, which contains the primary data file (typically having a .mdf file extension)
  - Every read-write filegroup
  - Any specified read-only file
- A file backup is a full backup for one or more files or filegroups.
  - This type of backup is only relevant for databases with multiple files.
  - With a simple recovery model, file and filegroup backups are basically limited to backing up read-only secondary database files.
  - With a full or bulk-logged recovery model, you can additionally use a file backup for read-write files so long as you make transaction log backups as well.
- You can perform differential backups subsequent to full, partial, or file backups. For example, if you take a full backup every Sunday, you can run a differential backup on all other days of the week.
  - The initial full, partial, or file backup for a database serves as the base for a subsequent differential restore.
  - Differential backup files contain the changed extents in a database since the differential base backup or the preceding differential backup, whichever is most recent. An extent is a collection of 8 physically contiguous pages, and a page is a fundamental data storage unit, which is 8 KB is size.
  - Full differential backups are typically smaller than their full base backups.
  - With a simple recovery model, you can restore a database to the time of any differential backup file.
- A copy-only backup is a backup that is managed separately from other types of backups. For example, a copy-only backup cannot serve as part of a differential backup set. You can create other types of backups with either SQL Server Management Studio (SSMS) or T-SQL, but SSMS cannot create a copy-only backup.

BACKUP LOG Statement

The BACKUP LOG statement backs up a database's transaction log in a comparable fashion to the way that the BACKUP DATABASE statement backs up a database's data files. You cannot use BACKUP LOG on a database in the SIMPLE recovery model. The transaction log is a log of the transactions for a database and the changes made to a database within those transactions. You can create a series of transaction log backup files just as you can create a series of differential backup database files.

Each successive log backup saves the changes made to the log since the previous log backup or the initial database backup file. You should generally perform log backups more often than you would perform full or differential backups. These files are generally smaller than full or differential backup files, though that can depend on logged database activity.

Transaction log backup sets complement database backup sets by being able to recover a database to any particular point in time, including the time of a disaster that makes a database inoperable. There are three types of transaction log backups.

- A pure log backup logs all transaction records during an interval, except for any bulk copy changes.
- A bulk log backup makes available the transaction log as well as the pages changed by bulk copy transfers to a database.
- A tail-log backup is a log backup taken after a database failure. This type of backup captures that portion of the log not copied before a database failure.

The sp_addumpdevice System Stored Procedure

You can simplify the syntax for BACKUP DATABASE and BACKUP LOG statements by designating a backup device for saving backup files. A backup device can store one or more database or log backup files. The sp_addumpdevice system stored procedure allows you to create a backup device for a SQL Server instance. You can survey the backup devices on a server instance through the sys.backup_devices view. It is common to designate disk and tape backup devices. See the following URL at the msdn2 site to drill down on the syntax for the sp_addumpdevice system stored procedure as well as get more detail about backup devices generally: http://msdn2.microsoft.com/en-us/library/ms188409.aspx

Implementing Database and Log Backups

You can implement backups with either SSMS or T-SQL code. Database backup concepts are the same whether you perform a backup with SSMS or T-SQL. An easy way to learn about the specific steps for implementing backups is with the "Backing Up and Restoring How-to Topics" page at the msdn2 site (http://msdn2.microsoft.com/en-us/library/ms189621.aspx). This page includes links for dozens of step-by-step instruction sets and T-SQL code samples for performing both database backups and restorations. A database restoration uses one or more backup files to recover a database. One typical reason for a restoration is that the database no longer operates. Section 4.4 discusses database restoration concepts in the same way that this section gives you the highlights of database backup issues.

4.4 Recover a Database

You can use RESTORE DATABASE with backup sets populated by BACKUP DATABASE and RESTORE LOG with backup sets populated with BACKUP LOG. Just as there are multiple types of backups, such as full, differential, and partial, there are corresponding database restore strategies. In addition, ***a database's recovery model places restrictions on the allowable uses of RESTORE DATABASE and RESTORE LOG.*** For example, the RESTORE LOG statement is exclusively for use with the full and bulk-logged recovery models. The bulk-logged database restoration options are generally a subset of those for the full recovery model. There are several common scenarios for restoring a database.

- The most common scenario is to perform a complete database restoration.
    - Under a simple recovery model, this technique involves restoring a full database backup or a full database backup supplemented by one from a set of differential backups.
    - Under full and bulk-logged recovery models, you can do a full or differential recovery as well as using log backup sets. ***The supplemental use of log backup sets enables the restoration of data until the time of a failure instead of just to the time of the last full or differential backup.***
    - When using a log backup set to restoration the database, you must have all intermediate backups from the first log backup through the last one. T-SQL has special syntax for recovering the tail-log backup as the last item in a backup set even when a database connection for normal queries is no longer possible.
    - A database is offline during a complete restoration.
- ***File restoration refers to a process where just selected data files are restored.*** If just a subset of the database files need repair, this approach is more appropriate than a complete database restoration.
    - Under the simple recovery model, this technique especially targets read-only files or filegroups within a database.
    - With full and bulk-logged recovery models, a file restoration can apply to secondary read-write files as well as read-only files. Log restorations apply only to read-write files (because there are no changes to restore for read-only files).
    - When performing a file restoration with any recovery model, the whole database is offline unless you are using the SQL Server 2005 Enterprise edition. For this edition, the overall database is online by default, but the filegroup containing the file being restored is offline.
- ***Piecemeal restoration, which is available in Enterprise Edition, allows you to restore a database in stages starting with the primary filegroup in the first stage. After the restoration of files in the first stage, the database can be brought online immediately instead of waiting for the restoration of the whole database.*** Files that are undamaged and consistent with restored filegroups do not need any restoration at all.
    - Under a simple recovery model, you can restore a filegroup to a partial database

backup or any of the partial backup's associated differential backups.

  - Under a full or bulk-logged recovery model, you can use the log backup set to recover to a point in time, including the time of failure for a database.

- Several other scenarios exist for restoring databases. One of these involves making a copy of a database by restoring the backup set. You can use the MOVE keyword for the WITH clause of RESTORE DATABASE to facilitate this goal. This approach is also convenient for upgrading SQL Server 7 and 2000 databases to SQL Server 2005.
- You can drill down further on restoration scenarios with the following two URLs and the links to additional resources that they offer: http://msdn2.microsoft.com/en-us/library/ms189323.aspx and http://msdn2.microsoft.com/en-us/library/ms190373.aspx

4.5 Backup and Restore T-SQL Examples

This section presents four examples to demonstrate selected T-SQL syntax conventions associated with backup and restore tasks. Each example includes at least one backup task with a corresponding restore task. The focus of all restoration examples is on completely restoring a database. The examples start by using BACKUP DATABASE and RESTORE DATABASE to make a copy of the sample AdventureWorks database. The other examples use the copied AdventureWorks database to demonstrate other more traditional backup and restore tasks.

Copying a Database

It is a common need to make a copy of a database and SQL Server offers multiple approaches to this task. The easiest is to make a backup the original database and then restore the database in a different location – even on another computer. ***The following script illustrates how to make a copy of the AdventureWorks database, and store the copy in a different location than the original AdventureWorks database. An adaptation of the process is appropriate for creating a complete backup of any database.*** The script consists of two statements.

- The BACKUP DATABASE statement creates a backup file (AdventureWorks.bak) for the AdventureWorks database in the PrepLogic folder of the C drive of a computer.
  - The TO clause with a DISK argument and a destination path and filename designates the backup file's destination. A subsequent example will illustrate the use of a backup device name instead of a destination path and filename. However, the destination path and filename acts like a device in that it can store one or more backup files.
  - The INIT argument in the WITH clause causes any existing backup files at the destination to be overwritten with the new backup file. The NOINIT argument, which is the default setting, results in the new backup file being appended to any existing files at the destination.
- The RESTORE DATABASE statement restores the AdventureWorks backup as DemoDB. You need to manually add the Restored subdirectory in the PrepLogic folder outside the script.
  - By specifying DemoDB as an identifier, the syntax designates a name of DemoDB for the copied database. You can use any legitimate identifier.

- The FROM clause specifies that the backup file resides on disk in the indicated path.
- The WITH clause uses the MOVE argument twice. In each case, the argument specifies a physical file destination for a logical file name. For example the AdventureWorks_Data logical filename in the original database is copied to the Demo.mdf file in the Restored subdirectory of the PrepLogic folder on the C drive of the local computer.

```
BACKUP DATABASE AdventureWorks
   TO DISK = 'C:\PrepLogic\AdventureWorks.bak'
   WITH INIT

RESTORE DATABASE DemoDB
   FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
   WITH MOVE 'AdventureWorks_Data' TO
        'C:\PrepLogic\Restored\DemoDB.mdf',
   MOVE 'AdventureWorks_Log' TO
        'C:\PrepLogic\Restored\DemoDB.ldf'
```

The completion of the above preceding script adds a new database named DemoDB to the current SQL Server instance. Once you create a backup file on any computer, you can restore it on any other computer. Just copy the backup file between computers. Then, use MOVE arguments in the WITH clause to specify new locations for the files underlying the copied database. There is no limitation about what edition a backup file can be restored on if it meets size requirements (such as for Express Edition) and features may not be used in the same way (such as indexed views.)

The preceding script also illustrates the general syntax for a full backup and a complete restore. Generally, you will not require the MOVE arguments in the WITH clause for this adaptation of the script. You don't need a MOVE argument because the RESTORE DATABASE statement copies restored database files to their original locations by default.

Creating and Restoring from Differential Backups

Differential backups are convenient because they allow you to restore to the time that a particular differential backup was taken. Recall that a differential backup is always with respect to a differential base backup. The backups for a differential base and the sequence of differential backups should reside in the same backup destination.

You restore a differential backup in two steps.

- ***The first step restores the differential base without making those values available for use.***
- ***The second step restores from a designated differential backup.***
- Refer to differential backups for a restoration by their file number in a backup set.
  - The first backup file in a differential backup set is the base backup.
  - The second and subsequent backup files in the backup set are for successive differential backups.

The following script reuses the backup set started in the preceding script – namely, the one at C:\ PrepLogic. By adding differential backups to the previous backup destination, you convert the initial full backup to the base backup for a differential backup set. The script makes two successive changes to the Bikes category name and performs a differential backup after each change. The script demonstrates two restorations of the DemoDB database to right after the first change and right after the second change. First, set this up by making a change in the database, and doing a differential backup:

```
UPDATE DemoDB.Production.ProductCategory
SET Name = Name + 'x'
WHERE Name = 'Bikes'

BACKUP DATABASE DemoDB
        TO DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH DIFFERENTIAL
```

- The BACKUP DATABASE statement after the first UPDATE statement adds the first differential backup to the backup set.
    - The TO clause designates the location of the backup set to use.
    - The DIFFERENTIAL argument in the WITH clause appends the current backup to any previously existing backup files in the destination specified by the TO clause.
    - The file number for this first differential backup in the backup set is 2 because SQL Server assigns a file number of 1 to the differential base backup and one more for each successive differential backup.

Next make another change and do another differential backup:

```
UPDATE DemoDB.Production.ProductCategory
SET Name = Name + 'y'
WHERE PATINDEX('%Bikes%', Name) > 0

BACKUP DATABASE DemoDB
        TO DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH DIFFERENTIAL
```

- The BACKUP DATABASE statement after the second UPDATE statement added another differential backup to the backup set in a third file. (You can see the files in the backup using the command: RESTORE HEADERONLY FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'

Next, restore the original backup and then the first differential backup:

```
RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH NORECOVERY

RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH FILE = 2, RECOVERY

SELECT * FROM DemoDB.Production.ProductCategory
```

- The first pair of RESTORE DATABASE statements after the second differential backup restores the database to the time of the first differential backup.
- The first RESTORE DATABASE statement restores the differential base. This statement does not need to specify a base file number, which is automatically the first file in a differential backup set. Notice the WITH clause includes a NORECOVERY argument, which instructs SQL Server to restore the data without making it available.
- The second RESTORE DATABASE statement specifies file 2 to restore the first differential backup and RECOVERY so that SQL Server will make the restored values available for use after rolling back any uncommitted transactions to the database.
- A SELECT statement after the RESTORE DATABASE statements shows the Bikes category name with a trailing x value:'Bikesx'. This shows the changes made by the first UPDATE statement.

The next set of restore statements restores the second differential backup:

```
RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH NORECOVERY

RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH FILE = 3, RECOVERY

SELECT * FROM DemoDB.Production.ProductCategory
```

- This pair of RESTORE DATABASE statements restores the database to the time of the second differential backup.
    - The first RESTORE DATABASE statement has the same syntax as the first RESTORE DATABASE statement in the preceding pair of statements because it performs the same task of restoring the differential base backup.
    - The second RESTORE DATABASE statement is identical to the second one in the preceding set, except that it designates the third file in the backup set to restore the

second differential backup. (Remember, the second backup you made is not used as part of the operation.)

- The final SELECT statement in this script retrieves the former Bikes category with a name of 'Bikesxy', which shows the changes made by the two UPDATE statements.

```
UPDATE DemoDB.Production.ProductCategory
SET Name = Name + 'y'
WHERE PATINDEX('%Bikes%', Name) > 0

BACKUP DATABASE DemoDB
        TO DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH DIFFERENTIAL
RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH NORECOVERY

RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH FILE = 2, RECOVERY

SELECT * FROM DemoDB.Production.ProductCategory
RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH NORECOVERY

RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH FILE = 3, RECOVERY

SELECT * FROM DemoDB.Production.ProductCategory
```

Creating and Applying Transaction Log Backups

The next and last final script in this chapter illustrate the creation and application of transaction log backups for a complete database restoration with a full recovery model. The script executes a RESTORE DATABASE statement to restore the DemoDB database to its initial state before any updates were made to the name for the Bikes category.

```
RESTORE DATABASE DemoDB
   FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
GO

ALTER DATABASE DemoDB SET RECOVERY FULL
GO

BACKUP DATABASE DemoDB
        TO DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH INIT
GO
```

- An ALTER DATABASE statement assigns a full recovery model to the database. ***Recall that this is necessary because using BACKUP LOG and RESTORE LOG statements is available only with the full or bulk-logged recovery model.*** *You must set it into FULL recovery mode before the database backup in order to do log backups later.*
- The BACKUP DATABASE statement containing a WITH clause that has an INIT argument reinitializes the backup set at C:\PrepLogic so that it contains just the backup restored by the preceding RESTORE DATABASE statement (recall that the destination previously contained a differential base backup and two differential backups).

Next, we will create a device to use for storing the transaction log backups. This is done using the sp_addumpdevice system stored procedure to add a new backup device named bklog_set.

```
IF EXISTS(SELECT* FROM sys.backup_devices
WHERE name = 'bklog_set')
        EXEC sp_dropdevice 'bklog_set'
GO
EXEC sp_addumpdevice 'disk', 'bklog_set',
'C:\PrepLogic\AdventureWorks_log.bak'
```

Next, two UPDATE statements followed by BACKUP LOG statements both modify the name of the Bikes category and save the changes in the log backup set within the bklog_set device.

```
UPDATE DemoDB.Production.ProductCategory
SET Name = Name + 'x'
WHERE Name = 'Bikes'

BACKUP LOG DemoDB
        TO bklog_set
        WITH INIT
GO

UPDATE DemoDB.Production.ProductCategory
SET Name = Name + 'y'
WHERE PATINDEX('%Bikes%', Name) > 0

BACKUP LOG DemoDB
        TO bklog_set
GO
```

Now, if you need to restore DemoDB from the backups you have made, you can follow the following steps. The same basic steps would be used to recover from a database issue, or You would do this if there was damage to the database, or if you just needed to restore the database.

- First, backup the tail of the log to cover any changes in the database.

```
BACKUP LOG DemoDB
        TO bklog_set
            WITH NORECOVERY
```

After invoking a third UPDATE statement, the script initiates a short pause with the WAITFOR statement and then invokes an extended stored procedure to erase the .mdf for the DemoDB database. The extended stored procedure simulates a disaster by destroying the primary data file. This disaster requires a database restoration to continue using the database. The pause before erasing the .mdf file waits for an interval so the preceding T-SQL statement can complete their execution. You may need to lengthen this interval if you are running with a slow computer or one which is burdened with many concurrent tasks.

- Immediately after the extended stored procedure, the script executes its third BACKUP LOG statement. ***The use of the NO_TRUNCATE argument in the WITH clause argument instructs SQL Server to attempt a log backup when the database is damaged. You can use this clause to add a tail-log backup to a backup set.***
- The steps for restoring DemoDB follows.
- Next, use RESTORE DATABASE to restore the backup before the first log backup. Like the restoring of a differential base backup, this RESTORE DATABASE statement for applying log backups requires a NORECOVERY argument in its WITH clause.

```
RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH NORECOVERY
```

- Next, three successive RESTORE LOG statements restore the first, second, and third log backup files from the bklog_set device. The third RESTORE LOG statement includes RECOVERY in its WITH clause to make the database available after rolling back any uncommitted transactions.

```
RESTORE LOG DemoDB
        FROM bklog_set
        WITH FILE = 1, NORECOVERY
GO

RESTORE LOG DemoDB
        FROM bklog_set
        WITH FILE = 2, NORECOVERY
GO

RESTORE LOG DemoDB
        FROM bklog_set
        WITH FILE = 3, RECOVERY
GO
```

- The final SELECT statement in the script demonstrates that the database has been restored through the tail-log backup so that the new name for the Bikes category is Bikesxyz. Recall that the script creates the tail-log backup after disabling the database by erasing its primary data file (with the .mdf extension).

```
RESTORE DATABASE DemoDB
   FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
GO

BACKUP DATABASE DemoDB
        TO DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH INIT
GO

ALTER DATABASE DemoDB SET RECOVERY FULL
GO

IF EXISTS(SELECT* FROM sys.backup_devices WHERE name = 'bklog_set')
        EXEC sp_dropdevice 'bklog_set'
GO
EXEC sp_addumpdevice 'disk', 'bklog_set', 'C:\PrepLogic\AdventureWorks_log.bak'
UPDATE DemoDB.Production.ProductCategory
SET Name = Name + 'x'
WHERE Name = 'Bikes'

BACKUP LOG DemoDB
        TO bklog_set
        WITH INIT
GO


UPDATE DemoDB.Production.ProductCategory
SET Name = Name + 'y'
WHERE PATINDEX('%Bikes%', Name) > 0


BACKUP LOG DemoDB
        TO bklog_set
GO
UPDATE DemoDB.Production.ProductCategory
SET Name = Name + 'z'
WHERE PATINDEX('%Bikes%', Name) > 0
GO
--Remove .MDF file for database
WAITFOR DELAY '00:00:05'
GO
EXEC xp_cmdshell 'erase C:\PrepLogic\Restored\DemoDB.mdf'
GO

BACKUP LOG DemoDB
        TO bklog_set
        WITH NO_TRUNCATE
GO
```

```
RESTORE DATABASE DemoDB
        FROM DISK = 'C:\PrepLogic\AdventureWorks.bak'
        WITH NORECOVERY
GO

RESTORE LOG DemoDB
        FROM bklog_set
        WITH FILE = 1, NORECOVERY
GO

RESTORE LOG DemoDB
        FROM bklog_set
        WITH FILE = 2, NORECOVERY
GO

RESTORE LOG DemoDB
        FROM bklog_set
        WITH FILE = 3, RECOVERY
GO


SELECT * FROM DemoDB.Production.ProductCategory
GO
```

4.6 SQL Server Agent Jobs

Many tasks associated with administering databases can benefit from automation. For example, running multiple differential base backups, differential backups, and log backups on a recurring database can help assure the recovery of the contents of a database. However, performing all these backups for numerous databases can be tedious and even error prone. Using SQL Server Agent, you can automate these kinds of tasks. By using SQL Server Agent in this way, database administrators free themselves from highly repetitive tasks so that they can devote their attention to more creative database planning and administrative activities. As such, it is in your best interest to learn about SQL Server Agent for the 70-431 certification examination.

The msdn2 site offers two key URLs with links for many related URLs to gain a thorough understanding on the concepts and practical steps for putting SQL Server Agent to use.

- Use the following URL to start studying the conceptual issues that allow you to understand the potential uses of SQL Server Agent and the background topics for the ways you can create and schedule jobs: http://msdn2.microsoft.com/en-us/library/ms187061.aspx
- Use the following URL for step-by-step instructions on how to use SQL Server Agent in a wide variety of different scenarios: http://msdn2.microsoft.com/en-us/library/ms189880.aspx

Chapter 5: Monitor and Troubleshoot SQL Server Performance

5.1 Using SQL Server Profiler

SQL Server Profiler is a graphical tool for generating a trace log. ***With SQL Server Profiler, you can trace all or a subset of Database Engine events within a SQL Server instance (you can also use Profiler to trace Analysis Services calls as well).*** A trace log records a sequence of events that occur on the server instance. The SQL Server Profiler tool enables you to

- review a trace log within the current SQL Server Profiler session
- save traces to SQL Server tables and files for reuse in a subsequent SQL Server Profiler session or for use with another tool, such as the Database Engine Tuning Advisor
- replay logs for benchmark testing

Within a SQL Server Profiler session, you start, stop, and pause tracing. Types of events are broken down into event classes, where are types of event that contains a set of data columns for reporting on event instances. Examples of data columns are the start time, end time, and duration for the running of a stored procedure or the error number, state and severity of an exception. A trace of all events is likely to track many more events than you are likely to be interested in monitoring. In addition, such a trace log is likely to require an excessive amount of storage space. Therefore, you can filter the events that you track, for example by collecting events just for a specific database or a specific login.

SQL Server Profiler Templates

Templates pre-define a set of event classes and data columns to use in a trace log. SQL Server Profiler ships with eight built-in templates. These templates track pre-specified event classes and data columns. You can build your own templates by dropping or adding event classes and data columns retained in the template for a trace. You can also filter events based on data column values to refine the range of events that you trace. The following are the standard templates:

- **Standard** - traces logins, stored procedures and T-SQL batches. This template is a general purpose one for monitoring server activity.
- **SP_Counts** - captures the start time for stored procedures. With this template, it is especially easy to see the flow of when stored procedures occur.
- **TSQL** - monitors all T-SQL statements submitted to a server instance by the client and the time issued. This template is particularly suited to debugging client applications.
- **TSQL_Duration** - lets you trace the execution time in milliseconds for all T-SQL statements. This template is especially handy for tracking slow-running queries that you may care to redesign.
- **TSQL_Grouped** - retrieves information about when all T-SQL statements run grouped by the client or user. This template is particularly appropriate when you need to filter information for a particular user or review the activity for several users.

- **TSQL_Replay** - targets scenarios when you want to replay a sequence of statements, such as for benchmark testing.
- **TSQL_SP** - tracks who ran what stored procedure when and how long it took to complete. You can also add the SP:Recompile event class to monitor if one or more stored procedures are being recompiled. This template is particularly suitable when you want to drill down on the usage of stored procedures.
- **Tuning** - tracks stored procedures and T-SQL statements, is tailored for generating trace output suitable for the Database Engine Tuning Advisor tool (see Section 5.2).

The Standard template is the default template that SQL Server Profiler uses for a trace log unless you explicitly specify another template. You can override the default setting so that any other built-in template serves as the default template. Whether or not you override the default template, it is easy to select any template when you start a trace.

Specifying a Trace Log

You can start SQL Server Profiler from either the Windows Start/Program Menu or from within SQL Server Management Studio (SSMS) from the Tools menu. Once SQL Server Profiler is open, you can start a trace log.

After opening SQL Server Profiler, you can initiate a new trace log by choosing File, New Trace. This lets you connect to a SQL Server instance. The trace log is for the connected server instance. After connecting to a server instance, SQL Server Profiler opens the Trace Properties dialog box. *Figure 5.1* shows the Trace Properties dialog box open with the array of built-in templates available for a selection. Use the Events Selection tab to modify the tracing that a template defines by

- adding or dropping event classes and data columns
- as well as designating filter values for data columns

The Trace Properties dialog box also offers additional fields for you to specify a file name or table to store the trace log for reuse later as well as a stop time to terminate tracing. Tracing can consume considerable resources – particularly when you elect to save a trace log to a table instead of a file. You do not need to save a trace log to review and examine the log from within the current SQL Server Profiler session.

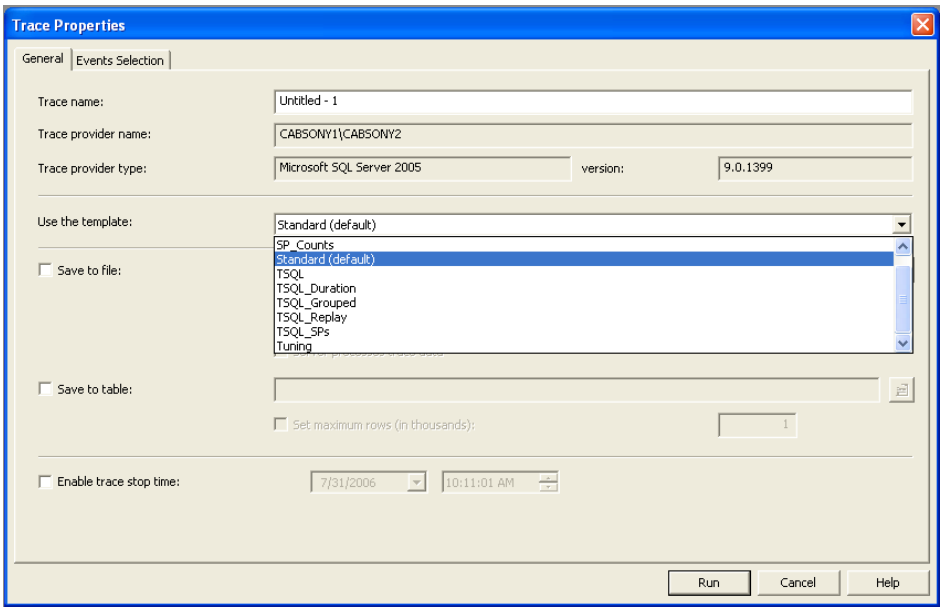


Figure 5.1. Use the Trace Properties dialog box to specify a trace log.

Viewing a Trace Log

Figure 5.2 shows a trace log for the following batches of statements. The first batch merely sets the database context to the PrepLogic database. You can use any other database context that you prefer for this example. The second batch inserts new rows into the #cmd_result temporary table based on the xp_cmdshell extended stored procedure. A SELECT statement after the extended stored procedure returns the directory names in the root of the C drive for the local workstation. In order to execute the xp_cmdshell stored procedure, you need to first enable its use with either the sp_configure system stored procedure and RECONFIGURE statement or the Surface Area Configuration utility. You can replace the INSERT statement with any other query statement you prefer to trace.

```
USE PrepLogic
GO

CREATE TABLE #cmd_result (output varchar(8000))
INSERT #cmd_result
EXEC master.dbo.xp_cmdshell 'DIR C:\*.*'
SELECT LTRIM(RIGHT(output,
        Len(output)-PATINDEX('%<DIR>%', output)-4))
FROM #cmd_result
WHERE PATINDEX('%<DIR>%', output)>0
DROP TABLE #cmd_result
GO
```

The trace log in *Figure 5.2* shows the completion of two SQL batches (using a modfied T-SQL template). The first batch, which assigns the PrepLogic database context, completed in 68 milliseconds. The second batch, which returns a list of directory names, ran in 149 milliseconds. Because the second batch is

selected in the top log pane, you can view the statements within the batch in the bottom log pane. If you were to run the batches multiple times, you would probably obtain different durations based on the state of the server instance during each run of the statements. For example, most statements run dramatically faster subsequent to SQL Server caching data after an initial execution.

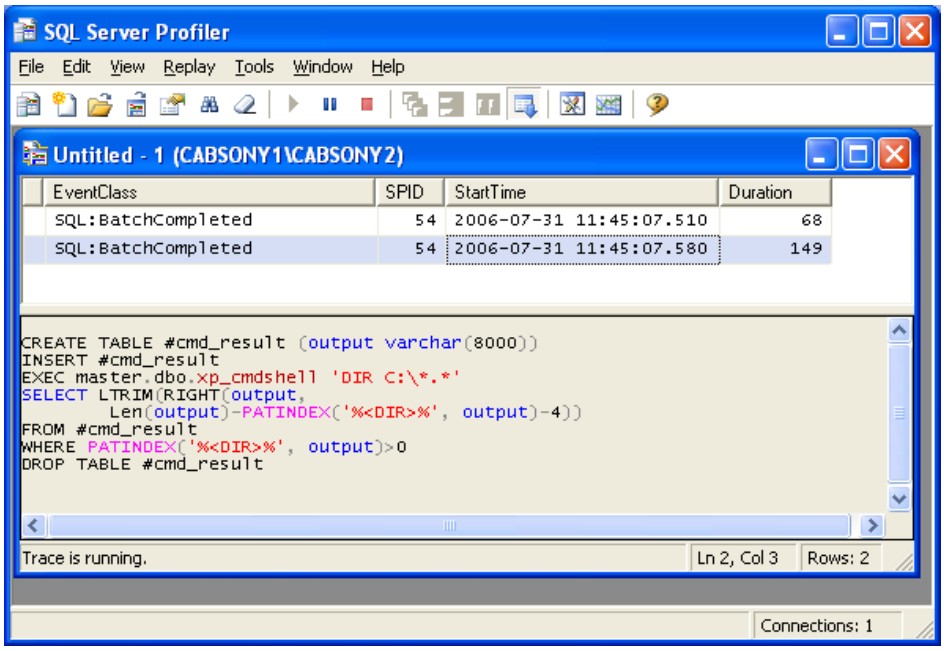


Figure 5.2. A trace log for a pair of batches.

You can drill down further on SQL Server Profiler from conceptual and implementation perspectives with the following pair of URLs: http://msdn2.microsoft.com/en-us/library/ms181091.aspx and http://msdn2.microsoft.com/en-us/library/ms181091.aspx. The second URL includes links for additional topics besides SQL Server Profiler, but the links for SQL Server Profiler are grouped together for easy reference.

5.2 Using Database Engine Tuning Advisor

The Database Engine Tuning Advisor can help you tune a database to improve the performance of one or more queries in a database context. This tool can analyze a workload of queries for a database. After the Database Engine Tuning Advisor completes its analysis, you are presented with a set of recommendations. You can also display, copy, paste, and run automatically generated code to implement the changes for modifying a database's physical design structure. For example, the tool can advise you to add or drop indexes, indexed views, or partitions.

Initializing and Running the Database Engine Tuning Advisor

On first use on a server instance, the Database Engine Tuning Advisor must be run by someone with sysadmin permissions. This initializes the tool for all subsequent uses on a server instance. The Database Engine Tuning Advisor prompts you to authenticate yourself prior to opening.

On subsequent uses, the Database Engine Tuning Advisor can be run by members of the sysadmin fixed server role or the db_owner fixed database role. Members of the sysadmin role can run the utility for any database on a server. Members of the db_owner role can run the utility for their database on a server instance. You can launch the Database Engine Tuning Advisor from the Windows ***/Program Menu*** or from within SSMS.

- From the Windows Start button, you can choose All Programs, Microsoft SQL Server 2005, Performance Tools, Database Engine Tuning Advisor, or in SSMS Choose Tools, Database Engine Tuning Advisor. You can then use it to analyze
  - a .sql file previously saved from SSMS (or other source)
  - a trace file or table saved from SQL Server Profiler
- From SSMS in a query tab, you can
  - Highlight the code for one or more queries and choose Query, Analyze Query in Database Engine Tuning Advisor

Analyzing a Selected Query

The easiest way to get started running the Database Tuning Engine Advisor is with a selected query from SSMS. Using this approach, the Database Tuning Engine Advisor already knows its workload when you open it. In addition, there is no need to independently create a workload file or table before running the Database Tuning Engine Advisor.

The following pair of batches of T-SQL statements starts by setting the database context to the AdventureWorks database. It is good practice to explicitly set the database context for a group of statements that you mean for analysis by the Database Engine Tuning Advisor. The second batch contains a query statement that counts the number of products within product categories. The query statement joins three tables, namely Product, ProductSubcategory, and ProductCategory in the Production schema of the AdventureWorks database. The ProductSubcategory serves as a junction table between the Product and ProductCategory tables.

```
USE AdventureWorks
GO

SELECT c.ProductCategoryID, COUNT(p.ProductID) '# of Products'
FROM Production.ProductCategory c
     JOIN Production.ProductSubcategory
        ON c.ProductCategoryID = s.ProductCategoryID
     JOIN Production.Product p
        ON p.ProductSubcategoryID = s.ProductSubcategoryID
GROUP BY c.ProductCategoryID
```

After selecting the preceding script from a query tab in SSMS and choosing Query, Analyze Query in Database Engine Tuning Advisor, you may be prompted to authenticate yourself. Once the Database Engine Tuning Advisor opens, it may appear as in *Figure 5.3*.

- The tool's session has an automatically defined name that you can optionally override. The automatically generated session name concatenates the login name with the date and time that a session starts.
- This view of the Database Engine Tuning Advisor in *Figure 5.3* is nearly ready to start. Pay particular attention to Workload option group.
    - Notice that the Query radio button is selected. This specifies a workload for analysis – namely, the selected statements from within SSMS.
    - You may need to use the drop-down box to specify a database for analysis.
    - For the example in *Figure 5.3*, you need to change the selected database for workload analysis from master to AdventureWorks. You can also designate specific tables that you want to analyze.
    - Use the Name and Selected Table columns in the grid below the Workload option group to specify a database and its tables that require analysis. (by default, when selecting a database all tables will be analyzed.)
- The Tuning Options tab gives you extended options limiting time to execute and what types of changes to suggest, like indexes, partitions, etc.
- After specifying the correct database context and tables, you can analyze the workload by choosing Actions, Start Analysis or simply clicking Start Analysis on the tool bar.

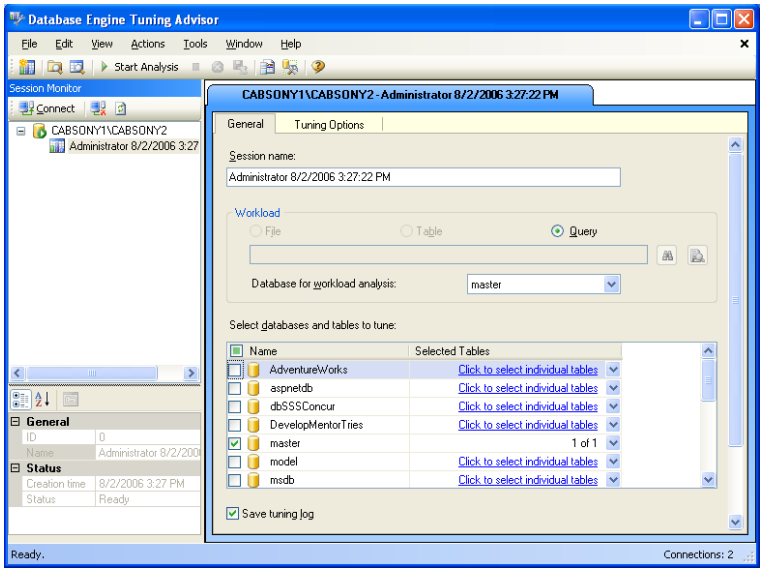


Figure 5.3. Initial Database Engine Tuning Advisor settings before customization for a workload based on the AdventureWorks database.

After the analysis completes, the Database Engine Tuning Advisor adds the Recommendations tab, which appears in *Figure 5.4*. For this example, the tool develops one tuning recommendation – namely, add a non-clustered index to the Product table. ***You can view the code to implement a recommendation by clicking the link in the Definition column. The display window for the code lets you copy the script to the Windows Clipboard. Then, you can paste the code into a query tab within SSMS. Finally, you can run the script for adding the index from within the AdventureWorks database context.***

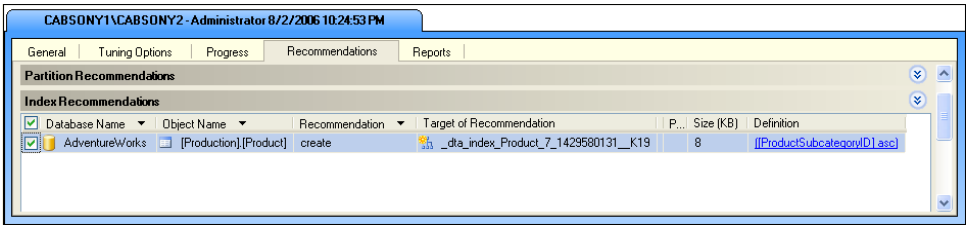


Figure 5.4. A Recommendations tab created by analyzing a workload for the preceding query.

One way to assess the worth of the recommendation for the new index is to run the preceding query with and without the recommended index applied to the Product table in the AdventureWorks database. You can quantify the performance gain by using SQL Server Profiler. After initially caching the query statement, the query ran in 3 milliseconds without the recommended index and in 1 millisecond with the recommended index. With larger tables or more queries, the savings could be much greater. However, even this outcome is advantageous for a query that is run many times by multiple users. This simple demonstration reveals the simplicity and power that you can gain by using the Database Engine Tuning Advisor to refine your database design.

Learning More about the Database Engine Tuning Advisor

To gain the full benefit from the Database Engine Tuning Advisor in your own projects and to properly prepare for the test, you will benefit from learning more about the tool. ***The msdn2 site offers three sections to help you grow your understanding of the Database Engine Tuning Advisor.***

- ***The Using Database Engine Tuning Advisor section provides an introduction to the tool.*** The pages in this section provide an overview of key concepts and techniques for applying the Database Engine Tuning Advisor. The URL for the first page in this section is: http://msdn2.microsoft.com/en-us/library/ms189303.aspx
- ***A tutorial section provides three lessons to give you guided instruction in how to use the Database Engine Tuning Advisor user interface as well as an associated command-line tool (dta).*** You can invoke the command-line tool for generating comparable results to the graphical tool without the need for manual intervention. The URL for the first page in this section is: http://msdn2.microsoft.com/en-us/library/ms166575.aspx
- ***The last major resource for the Database Engine Tuning Advisor is a reference section.*** This section drills down on the capabilities and techniques used to provide those functions that the Database Engine Tuning Advisor lets you achieve – especially for advanced topics. The URL for the first page in this section is: http://msdn2.microsoft.com/en-us/library/ms173494.aspx

5.3 Locks, Blocks, and Deadlocks

To allow multiple users to use database resources concurrently, the query requests require rules governing how users and processes interact with one another. A process is an application, such as the Database Engine Tuning Advisor, that can run code just like a typical database user. Without rules governing how multiple users can concurrently query a database resource, it is typical to experience four types of concurrency effects.

- Lost updates describe a scenario where two users concurrently read a database to edit some values, but the second user's change always overwrites the first user's change. In this scenario, the first user's updates are always lost.
- Uncommitted dependency refers to a situation in which a SELECT statement from one user reads an update that is started, but subsequently rescinded before it is committed, by another user. As a result, the SELECT statement's result set does not reflect the database's contents. (This is also known as a "dirty read")
- Inconsistent analysis applies to cases where one user reads a database two or more times while an update of one or more rows is in process. The results are described as inconsistent because successive reads return different result sets depending on the update status during successive reads. (This is also known as a non-repeatable read.)
- Phantom reads occur when a database is experiencing inserts and deletes concurrently along with successive reads. The result sets for successively executed identical SELECT statements can show "phantom" rows based on
    - previously existing rows that are deleted
    - recently inserted rows that did not exist at the time of a previously executed SELECT statement

Some of these effects are acceptable at time, while sometimes not. I will mention this again later in the chapter when I mention isolation levels.

Types of Locks

A locks is a piece of data that SQL Server uses to make sure that two users cannot do incompatible operations at the same time. For example, two users should not be able to make changes to the same data at the same time. SQL Server provides several different types of locks depending on the type of concurrency effect an application needs to manage.

- **Shared locks** – allows other users to read the same data, but prevents others from changing the data.
- **Update locks** - are used to prevent deadlocks (covered them later in this chapter) by marking rows that a statement will possibly update, rather than rows that are actively being updated).
- **Exclusive locks** - typically used when data is being modified by prohibiting multiple concurrent updates to the same resource.

- **Intent locks** - signals intention to place another kind of lock on a resource. There are six varieties of intent locks. Because intent locks apply only at the table level, they are more efficient than other locks which may require an application to check many rows before determining if it can lock a subset of rows within a data source
    - ***Intent shared locks*** - protect shared locks for some, but not all, rows within a data source
    - ***Intent exclusive locks*** - protect exclusive locks for some, but not all, rows within a data source. Intent exclusive locks are a superset of intent shared locks
    - ***Shared with intent exclusive locks*** - protect shared locks for some, but not all, and exclusive locks for all rows within a data source
    - ***Intent update locks*** - protect update locks on all rows within a data source
    - ***Shared intent update locks*** - are a combination of shared locks and intent update locks
    - ***Update with intent exclusive locks*** - are a combination of update locks and intent exclusive locks
- There are two varieties of schema locks.
    - ***Sch-M*** - blocks concurrent DDL statements for a database resource.
    - ***Sch-S*** - blocks concurrent DML statements, such as INSERT, UPDATE, and DELETE. However, Sch-S type locks permit some types of DML statements, including those with an X (exclusive) type lock.
- **Bulk update locks** - block other types of tasks while allowing on different threads two or more concurrent bulk copy operations for a table.
- **Key-range locks** - block phantom insertions and deletions from a record set.

To determine how long locks are held on resources, SQL Server defines five isolation levels:

- READ UNCOMMITTED – no shared locks taken, no locks honored
- READ COMMITTED (the default) – shared locks taken and released immediately. Allows phantoms and non-repeatable reads
- REPEATABLE READ – holds shared locks taken to prevent non-repeatable reads
- SERIALIZABLE – includes key-range locks to block phantoms
- SNAPSHOT – special case that allows viewing of data as it was when a transaction started

For more information on isolation levels, check here: http://msdn2.microsoft.com/en-us/library/ms172001.aspx and http://msdn2.microsoft.com/en-us/library/ms173763.aspx

Blocks and Deadlocks

The SQL Server Database Engine applies locks automatically based on query syntax, such as a shared lock for a SELECT statement or an exclusive or update lock for an UPDATE statement. The Database Engine can elect to apply locks at any of several levels, such as a table or one or more rows in a table. You can override the Database Engine lock assignments by using hints in your query statements that specify various locking scenarios, but generally you should refrain from doing this because it can adversely affect the concurrency of your database.

An active lock held by one user can block a request for a lock on the same resource by another user. Normally, the block is until the initial user releases its lock so that the second user can gain the lock to run a query statement. ***It is possible for two users to request locks for resources on which each other user already has a lock. In this situation, a deadlock results in which each user blocks the other user.*** Without some external action, the two deadlocked queries will block each other indefinitely.

Figure 5.5 illustrates a deadlock between two user sessions.

- The session for user 1 has a lock on table 1.
- The session for user 2 has a lock on table 2.
- The cross-user requests for locks on tables already locked in the other user's session generates a deadlock.

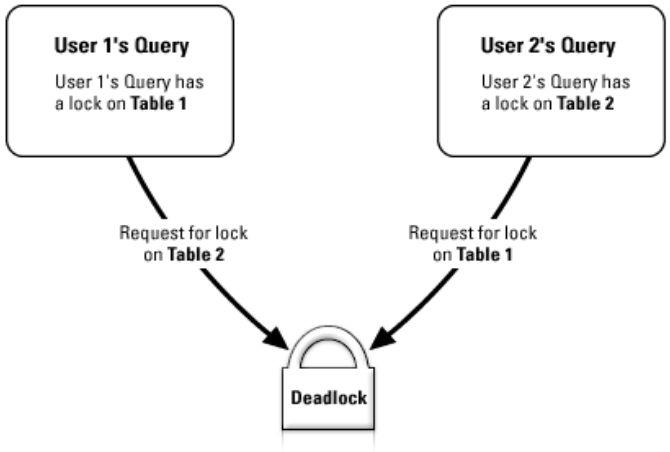


Figure 5.5. A diagram depicting a deadlock between two user sessions.

SQL Server automatically resolve deadlocks by *regularly checking for deadlocks within its running database applications. When it encounters a deadlock, the server instance terminates one session.* This allows the other session to complete because its request for a lock is no longer blocked by the terminated session. ***The server instance returns a 1205 error to the terminated session to signal that it was terminated because of a deadlock.*** In this case, the user can resubmit the query, which will normally run successfully if the other session participating in the deadlock finished.

It is generally a best practice to include error handlers for a 1205 error in your data access code. In the event of a 1205 error, you can simply resubmit the query. See the Handle Exceptions heading in section 3.2 of Chapter 3 for more detail on exception handlers.

There are several techniques to follow in order to minimize the chances for blocks and deadlocks between connections.

- Work with resources in the same order in all concurrent sessions.
    - The deadlock in *Figure 5.5* occurs because user 1's session requests a lock on table 2 after user 2's session already has a lock on table 2.
    - If both user sessions initially requested a lock on table 1 before a lock on table 2, then one session could just temporarily block, instead of deadlock, the other session.
- Enclose your queries in transactions and design each transaction so that it can contain as few locks as possible – preferably just one. This approach can eliminate or reduce sessions requesting locks on resources that are already locked in another session while holding locks on resources requested by other sessions.
- If you do include queries in transactions, avoid having code that waits for user input before completing the transaction. The reason for this guideline is because a user can leave a computer with an unfulfilled request for input within a transaction. As a result, the application generates a long-lasting block even if there is technically no deadlock.
- Use the lowest isolation level for a transaction to place the least restrictive locks on referenced resources. For example, you can use the read committed isolation level to avoid dirty reads holds shared locks for a shorter duration than a serializable isolation level to eliminate the possibility of phantom reads.

5.4 Other Monitoring and Troubleshooting Topics

This chapter introduced you to three important monitoring and troubleshooting topics. You can benefit from drilling down more on each of these topics as well as getting acquainted with selected other monitoring and troubleshooting techniques. This chapter closes with a several additional URLs to help you, but you should also review the URLs sited throughout the chapter.

- For deadlocks,
    - Analyzing deadlocks with SQL Server Profiler:

      http://msdn2.microsoft.com/en-us/library/ms188246.aspx
    - How to open, view, and print a deadlock file:

      http://msdn2.microsoft.com/en-us/library/ms187021.aspx
    - Assigning custom priorities to override the default selection of which session is eliminated during a deadlock:
        - http://msdn2.microsoft.com/en-us/library/ms186736.aspx

- Transactions:
    - Introduction to transactions:
        - http://msdn2.microsoft.com/en-us/library/ms172400.aspx
- Dynamic management views and functions are useful for monitoring the state of a SQL Server instance. The following URL provides an overview of these views and functions along with a series of links for drilling down on special sets of dynamic management views and functions: http://msdn2.microsoft.com/en-us/library/ms188754.aspx
- SQL Server copies and timestamps selected items in its own error log as well as the Windows application log. You can learn more about using these logs for troubleshooting SQL Server from the following URL: http://msdn2.microsoft.com/en-us/library/ms191202.aspx

Chapter 6: Creating and Implementing Database Options

6.1 Implement a Table

A table is the basic storage object for data inside SQL Server. A table consists of a set of rows each containing multiple columns. It is common in database modeling to use a table to represent an entity, such as an employee, sale, or product. For example, a Sale table might contain information for all sales within an organization. Each row in such a table would then represent an individual sale. The columns within a row can represent attributes of the sale, such as the sales date and clerk entering the sale.

When designing a table, you can optimize its query performance by assigning the most appropriate and smallest data types to columns. You can also specify other column properties besides data type, such as an IDENTITY property that improve the performance of a table. ***The IDENTITY property automatically assigns values to a column starting from a seed value with fixed increments.*** Many database applications use this type of column as a primary key for a table. ***The primary key is another type of property that restricts its contributing column***
values so that each row has a unique primary key value.

Table and Column Naming

Both the CREATE TABLE and ALTER TABLE statements reference a table by a name following the TABLE keyword. You can specify table names with one, two, three, or even four parts.

[Database_Identifier.][Schema_Identifier.]Table_Identifier

- **Table_identifier -** All database objects require identifiers. There are several rules for identifiers, like the must start with a letter and not contain special characters. See the complete rules for SQL Server 2005 identifiers at http://msdn2.microsoft.com/en-us/library/ms175874.aspx.
- **Schema_identifier -** Any database can have multiple relational schemas to which objects, such as tables, belong. Because tables and views occupy the same namespace within a relational

database schema, all object identifiers must be unique within a schema. When a table must reside in another schema besides the default schema for its creator or modifier, then you should qualify the table name's first part with a schema name, such as schema_name.table_identifier.

- **Database_Identifier -** If you are creating or modifying a table in a different database than the current database context, then you should explicitly qualify the schema name with a database name. For example, if the database context is set to the AdventureWorks database, but you want to create a SpecialShippers table in the Northwind database, then you could specify the table with the following name: Northwind.dbo.SpecialShippers.

Unless you name a table as a temporary table with the CREATE TABLE statement, the table is considered permanent and its scope extends until a DROP TABLE statement removes the table from the database. You can create a temporary tables in the following manner

- A local temporary table is available only from within the current client session or object where it is created. This type of temporary table has a name beginning with a pound/number sign (#). The table goes out of scope at the end of the client session.
- A global temporary table is available from within any client session within a server session. This type of table has a name beginning with a pair of number signs (##). A global temporary table goes out of scope when no client sessions reference it.

In addition to three part names, you can optionally specify a fourth part of the name before the database identifier to reference a linked server.

Specifying Table Columns

After specifying the name in the CREATE TABLE statement, you define the columns and constraints for a table within parentheses after the table name. Separate the column definitions from one another with commas.

- ***Columns are defined minimally by a column name and a data type.***
  - The name for a column serves as its identifier. A column name has just one part, but it must be unique within a table.
  - The data type designates how SQL Server stores the data within a column.
- ***You can designate computed columns that generate a value for a column based on an expression referencing one or more other columns.*** By default, computed column values are not stored but rather computed as requested. However, you can have the value stored in the database by marking it as PERSISTED.
- There are seven broad categories of built-in T-SQL data types. You can learn more about data types at http://msdn2.microsoft.com/en-us/library/ms187752.aspx. These categories are:
  - Exact numerics: bit, tinyint, smallint, int, bigint, decimal, numeric, money, smallmoney
  - Approximate numerics: float, real
  - Date and time: datetime, smalldatetime

  - Character strings: char, varchar, text (deprecated by varchar(max)
  - Unicode character strings: nchar, nvarchar, ntext (deprecated by nvarchar(max)
  - Binary strings: binary, varbinary, image (deprecated by varbinary(max))
  - Other data types: cursor, sql_variant, table, timestamp, uniqueidentifier, and xml *(you cannot use the cursor or table datatypes in a table declaration)*

- You can optionally use a user-defined alias or CLR (Common Language Runtime) type.
- Besides name and data type, other elements of a column definition can include whether the column

  - has an IDENTITY property (only one per table)
  - does or does not permit null values
  - contains the identifying ROWGUIDCOL (a globally unique identifier that can be used by the system to identify a row)
  - has constraints, such as a primary key, unique key, foreign key, default, or check constraints

The following script illustrates several design guidelines for the use of a CREATE TABLE statement.

```
USE PrepLogic
GO

IF EXISTS (SELECT * FROM sys.tables t
        JOIN sys.schemas s
        ON t.schema_id = s.schema_id
        WHERE t.name = 'Employee' AND
        s.name = 'dbo')
        DROP TABLE dbo.Employee
GO

CREATE TABLE dbo.Employee (
        EmpID int IDENTITY(2,2) CONSTRAINT PKemployee
        PRIMARY KEY CLUSTERED,
        Fname nvarchar (15) NOT NULL,
        Lname nvarchar (20) NOT NULL,
        Hiredate smalldatetime NULL
```

- The script starts with a USE statement to set the database context to the PrepLogic database. *Section 1.3 of Chapter 1 illustrates how to create the PrepLogic database. You can replace the PrepLogic database with any other database name that is readily available to you.*
- The IF statement starting the second batch checks for the existence of the Employee table in the dbo schema of the current database context (PrepLogic). When a table named Employee already exists in the dbo schema, the DROP TABLE statement removes the table from the database. ***If a CREATE TABLE statement attempts to create a table with a name that already exists within the relational database schema, the CREATE TABLE statement fails.*** You can modify the script to save the data before dropping the table or simply rename the table.

- The CREATE TABLE statement in the last part of the script specifies a new table named Employee in the dbo schema. The table contains four columns.
    - The first column, EmpID, has an int data type. The IDENTITY constraint indicates that SQL Server specifies values for the column starting with a value of 2 and increasing by a value of 2 for each new attempt to insert a row.
    - The CONSTRAINT PKemployee PRIMARY KEY phrase designates this column as the primary key for the table, and the CLUSTERED keyword specifies that rows are physically stored in the order of the column values. The name of the primary key object will be PKemployee. It is a best practice to name all constraints, or SQL Server will assign a name for you.
    - The Fname and Lname columns represent the first and last names of employees with a Unicode character set. The maximum number of characters for the first and last names is 15 and 20, respectively. SQL Server will not accept a row without values for the Fname and Lname columns because of the NOT NULL specification.
    - The Hiredate column is to store the date an employee is hired. The smalldatetime format, which represents dates from January 1, 1900 through June 6, 2079 with precision to the minute, is sufficient for many common business applications and requires half the space of the datetime data type. The NULL specification for the column indicates that a user can input a row for a new employee without designating a column value for Hiredate (in case it is not known at the time of data entry).

Specifying Filegroups

You add and remove filegroups and their files to a database with either CREATE DATABASE or ALTER DATABASE statements. We previously covered filegroups in Chapter 1, section 1.3, within the Configuring Database Options and Files heading. *Two popular uses for filegroups are to permit selective backup and restore of database contents, such as its tables, and to partition a single large table across multiple partitions.*

The ON clause in a CREATE TABLE statement allows you to specify a partition scheme, filegroup, or default filegroup. You can use the ON clause after the parentheses defining the columns of a table or in a PRIMARY KEY or UNIQUE constraint clause for a column definition within a table.

- The partition scheme is for working with partitioned tables, which is the focus of the next heading.
- You can also directly name a filegroup previously defined in a CREATE DATABASE or ALTER DATABASE statement. This will cause the data for the table to be stored within the filegroup.
- By using the DEFAULT keyword as the argument for an ON clause or omitting the ON clause, you can designate the storage of a table in the default filegroup. You can modify the default filegroup from the one containing the primary data file to any user-created filegroup with the ALTER DATABASE statement.

Specifying Partitioned Tables

Partitioned tables in SQL Server 2005 are available only in the Enterprise and Developer editions. This type of table supports traditional table design features while offering advantages for very large tables that naturally divide into segments or partitions. One partitioned table can be comprised of up to 1000 partitions.

The partitions within a partitioned table can be assigned to different filegroups so that you can administer the partitions separately for selected tasks, which can improve the performance of these tasks. You derive your best performance by aligning indexes for a partitioned table so that the indexes belong to the same partitions as the table partitions that they index. ***The primary reason for using partitioned tables is to achieve performance advantages with very large tables within a single server instance.*** If you have very large tables that are performing acceptably for your needs, then you can avoid the administrative planning that partitioned tables require.

There are three steps to creating a partitioned table, which are summarized in the following items.

- ***First, you must create a partition function with the CREATE PARTITION FUNCTION statement.*** This defines the boundaries for each partition in a set of partitions. The function maps the partition boundaries to ranges of data values, which you can, in turn, link to a column within a partitioned table. You can use any data type to define boundaries, except for timestamp, ntext, text, image, xml, varchar(max), nvarchar(max), and varbinary(max).
- ***Second, you must create a partition scheme with the CREATE PARTITION SCHEME statement.*** A partition scheme maps the partitions defined by a partition function to a set of filegroups. A single partition function can be used by multiple partition schemes, but a single partition scheme can use only one partition function. You can map partitions so that each partition belongs to a different filegroup, some partitions belong to different filegroups, all partitions belong to the same filegroup. You can also designate unassigned filegroups for use with partitions that you intend to add in the future.
- ***Third, you must specify a table with the CREATE TABLE statement that uses the ON clause to reference a partition scheme.*** The scheme, in turn, references a partition function. In addition, the ON clause designates a column with values to submit to the partition function for segmenting rows into partitions.

You can learn more about design concepts and implementation issues for partitioned tables from this URL and the links that it offers: http://msdn2.microsoft.com/en-us/library/ms188706.aspx

Assigning Permissions to Roles

There are several permissions that you can assign to users or user-defined roles for accessing a table after you create it. Use a GRANT statement to assign permission for accessing or creating tables. See the Managing and Tracking Securables heading in section 1.4 of Chapter 1 for more detail on using the GRANT statement with users and user-defined roles.

Explicit permissions for accessing a table include DELETE, INSERT, REFERENCES, SELECT, and UPDATE. With the exception of REFERENCES, the permission names convey permission to reference a table with a particular type of T-SQL statement, such as SELECT or UPDATE. The REFERENCES permission conveys permission to reference a table in a FOREIGN KEY constraint when the owner of the child table does not own the parent table pointed at by the foreign key.

There is a CREATE TABLE permission that can enable a user or role member to create tables if they do not already derive this permission from membership in a fixed-server or fixed-database role. When creating tables, users also require ALTER permission for the relational schema to which a table belongs. In addition, if a table includes a column with a CLR user-defined type or a typed xml column, the user creating the table needs to own the CLR user-defined type or xml schemas for the typed column or at least have REFERENCES permission for the type or schema.

6.2 Implement a View

A view is often referred to as a virtual table. There are two main reasons for this. First, you specify a view with a SELECT statement. Therefore, a view does not actually store data directly, but it instead returns data stored by one or more other tables. ***Second, you can use a view in the FROM clause of a SELECT statement, and you can manipulate the base table values for a view similarly, but not exactly like, a table.*** There are three classifications of views that are commonly referred to:

- **A standard view** – usually referred to simply as view - lets you shape data from one or more tables in a database. As a result, a view enables you to filter the rows and columns from its base tables.
- **An indexed view** distinguishes itself from a standard view by taking advantage of indexing and locking the base tables for a view. This can dramatically speed up the operation of a view and delivers performance benefits to other views and even SELECT statements in T-SQL batches with similar semantics.
- **A partitioned view** joins one or more tables across one or more server instances. When dealing with a single server instance, Microsoft recommends that you use partitioned tables instead of partitioned views. Partitioned views on multiple servers can concurrently make data sources locally available and provide a means for consolidating data across various organizational or geographic units within an enterprise.

Creating and Using Standard Views

A standard view is a database object that contains a single SELECT statement drawing on one or more base tables. The base tables can come from one or more databases. Multiple tables can be merged into a single set – typically by joining the base tables (you can also use UNION, INTERSECT, or DIFFERENCE to connect multiple statements into a single SELECT statement). Some major benefits of standard views include:

- Simplifying access to data that requires joining multiple tables (or other views). Highly normalized databases, such as the AdventureWorks database, frequently require joins to derive standard result sets, and standard views can save the joins and make accessing the result sets transparent. You can even use views to simplify access to data on othe servers.
- Accessing standard views instead of base tables simplifies making changes to the table design of a database. This is because updated standard views can return the same data to applications even when a database's relational schemas change.
- You can insert, update, and delete values for the base tables through a view. This feature can benefit data manipulation in the same way that it helps to retrieve data.

All type of views are created with a CREATE VIEW statement. In addition, you can modify existing views with ALTER VIEW or remove views from a database with the DROP VIEW statement. Just like tables, views reside in a relational schema, such as the dbo schema. The AdventureWorks database has a reasonably rich relational schema structure with views in multiple schemas. A view's name must be distinct from other views and tables within the same schema.

The core of a CREATE VIEW statement is a SELECT statement that defines the data made available through the view. The general syntax for a CREATE VIEW statement occurs below. The AS keyword introduces the SELECT statement for the view. You must always specify a view name. If the CREATE VIEW statement does not specify a schema name, the view belongs to the default schema for the user creating a view. In addition, the WITH clause for view attributes and the WITH CHECK OPTION clause are optional. After the initial creation of a view, you can modify any view element through an ALTER VIEW statement, including its SELECT statement, view attributes or CHECK OPTION setting.

```
CREATE VIEW schema_name.view_name
WITH view_attributes
AS
Select_statement_for_view
WITH CHECK OPTION
```

There are several restrictions on the general scope of a SELECT statement when it appears in a view. These restrictions vary by the type of view you are creating, but the restrictions for standard views apply to other types of views as well. Some of the more prominent restrictions appear below. Others are listed at http://msdn2.microsoft.com/en-us/library/ms189918.aspx

- An ORDER BY clause is not permitted unless the SELECT statement also includes a TOP clause. *(Note that even when a view has an ORDER BY the output of the view is not ordered. The ORDER BY is only used to determine values that will be returned based on the TOP clause.)*
- Column names in the SELECT statement must be unique within a view.
- The INTO keyword is not allowed.

You can specify any combination of three view attributes in the WITH clause immediately after a view's name. Separate two or more attribute names with commas in a single WITH clause.

- WITH ENCRYPTION encrypts the text for a view in SQL Server system views and prevents a view from being published during SQL Server replication.
- WITH SCHEMABINDING binds a view to the base tables specified in the FROM clause of its SELECT statement. This makes it impossible to change referenced objects in a way that alters the functionality provided through the view.
- WITH VIEW_METADATA facilitates the manipulation of base table data through client-side applications using DB-Library, ODBC, and OLE DB APIs.
- The WITH CHECK OPTION clause governs the updateability of the base data sources for a view. This clause is discussed in greater depth in the Updating Data heading within this section.

The following script creates a new view named product in the PrepLogic database.

```
USE PrepLogic
GO

IF OBJECT_ID(' dbo.product ') IS NOT NULL
        DROP VIEW dbo.product
GO

CREATE VIEW dbo.product
AS
SELECT c.ProductCategoryID, c.Name AS CategoryName,
          s.Name AS SubcategoryName , p.Name AS ProductName
FROM AdventureWorks.Production.ProductCategory c
    JOIN AdventureWorks.Production.ProductSubcategory s
      ON c.ProductCategoryID = s.ProductCategoryID
    JOIN AdventureWorks.Production.Product p
      ON s.ProductSubcategoryID = p.ProductSubcategoryID
```

- The script begins by setting the database context to the PrepLogic database.
- Next, a DROP VIEW statement removed the view named product if it already existed in the dbo schema. Searching for a database object within a schema is simpler using the OBJECT_ID function than joining the system view for a database object with the system view for relational schemas.
- The SELECT statement inside the CREATE VIEW statement
    - joins the ProductCategory, ProductSubcategory, and Product tables from the Production schema in the AdventureWorks database
    - assigns distinct names to the Name columns from the three joined tables to avoid having conflicting column names within a view

After creating view1, you can run the following script to put the view to use. Notice that the code for using the view1 view references the view like a table. However, view1 provides access to multiple base table column values in a single database object.

- The first SELECT statement simply returns all the rows for all the columns in the view. This makes it easy to browse the complete result set from the view.
- The second SELECT statement illustrates how to filter the returned values from the view. Notice that only rows with a subcategory name of Road Bikes within the Bikes category (ProductCategoryID = 1) are specified for inclusion in the SELECT statement's result set.
- The two UPDATE statements illustrate the syntax for modifying a column value from a view.
    - The first UPDATE statement appends an x to the end of a ProductName column value

for the row specified in the WHERE clause.

- The second UPDATE statement restores the converted column value to its original value.

```
SELECT *
FROM view1
ORDER BY ProductCategoryID

SELECT SubcategoryName, ProductName
FROM view1
WHERE ProductCategoryID = 1 AND
SubcategoryName = 'Road Bikes'

UPDATE view1
SET ProductName = ProductName + 'x'
WHERE ProductName = 'Road-150 Red, 62'

UPDATE view1
SET ProductName = 'Road-150 Red, 62'
WHERE ProductName = 'Road-150 Red, 62x'
```

Creating and Using Indexed Views

Indexed views specify a unique, clustered index for a view and bind the view to its base tables. Besides these two obvious features, indexed views differ from standard views in a variety of other ways – some of which are listed below. Other important distinctions for indexed views are summarized at http://msdn2.microsoft.com/en-us/library/ms191432.aspx

- All references to base tables in an indexed view must be from the same database as the one to which the view belongs.
- You can only reference base tables by two-part names denoting the relational schema name to which a view belongs.
- When working with aggregates, you will often find it essential to include a Count_Big function among your list of selected items.

For views with large base tables and especially views with aggregate functions, indexed views can deliver special performance benefits. Of course, there is a performance cost to maintaining data for the indexed view during data manipulation tasks, and this cost is greater for an indexed view than for a base table. Therefore, ***you should greatly limit the use of indexed views with base tables which it is common to frequently insert, update, or delete rows.***

The following script shows the syntax for creating an indexed view based on the ProductCategory, ProductSubcategory, and Product tables in the Production schema of the AdventureWorks database.

```
USE AdventureWorks
GO

IF OBJECT_ID('dbo.product') IS NOT NULL
DROP VIEW dbo.product
GO

CREATE VIEW dbo.product
WITH SCHEMABINDING
AS
SELECT c.ProductCategoryID, c.Name AS CategoryName,
   s.Name AS SubcategoryName, COUNT_BIG(*) AS [# of Products]
FROM Production.ProductCategory c
    JOIN Production.ProductSubcategory s
     ON c.ProductCategoryID = s.ProductCategoryID
    JOIN Production.Product p
     ON s.ProductSubcategoryID = p.ProductSubcategoryID
GROUP BY c.ProductCategoryID, c.Name, s.Name
GO

CREATE UNIQUE CLUSTERED INDEX ind_product
ON dbo.product(ProductCategoryID, SubcategoryName)
GO
```

- Because an indexed view must refer to tables in the same database as it resides, the script starts by changing the database context to AdventureWorks.
- Notice that the CREATE VIEW statement included a WITH SCHEMABINDING clause, which binds the view to its base tables. This clause is mandatory for an indexed view.
- Also notice that all references to tables in the FROM clause of the SELECT statement within the CREATE VIEW statement use two-part names, such as Production.ProductCategory.
- In addition, the list for the SELECT statement includes a COUNT_BIG function with an asterisk as its argument. The use of this expression is mandatory when an indexed view returns any aggregate function values.
- A CREATE INDEX statement after the CREATE VIEW specifies the index for the indexed view.
    - Notice especially the use of the UNIQUE and CLUSTERED keywords in the CREATE INDEX statement. The first index for an indexed view must be a unique clustered index.
    - The ON clause in the CREATE INDEX statement makes the index conditional on ProductCategoryID and SubcategoryName column values from the indexed view. In a production application, you might prefer to alter the SELECT statement to permit the use of SubcategoryID instead of SubcategoryName.

Despite the fact that you have to create an indexed view in the same database as the base tables that it references, you can refer to an indexed view from any database. The following script shows two references to the indexed view named view1 in the AdventureWorks database.

- The initial SELECT statement assumes the AdventureWorks database. Because the column name (# of Products) includes non-standard identifier characters, it is contained within brackets.
    - This SELECT statement reports the total number of products per category.
    - The ORDER BY clause is necessary to ensure that the rows display in ascending order by ProductCategoryID.

```
USE PrepLogic
GO

SELECT * FROM AdventureWorks.dbo.view1
```

- The next SELECT statement references view1 in the dbo schema of the AdventureWorks database from the PrepLogic database context.
    - You need a three-part name from the PrepLogic database context that specifies the AdventureWorks database to reference view1 in the dbo schema.
    - The SELECT statement illustrates the basic syntax of returning all rows for all columns. However, you can use any SELECT statement syntax to refer to an indexed view in another database so long as you specify the view with a three-part name that denotes the database in which the indexed view resides.

```
SELECT ProductCategoryID, CategoryName, SUM([# of Products])
FROM dbo.view1
GROUP BY ProductCategoryID, CategoryName
ORDER BY ProductCategoryID
```

Updating Views

T-SQL code samples for a standard view within this section's Creating and Using Standard Views heading illustrate the use of the UPDATE statement to change a base table value through a view. You can also use INSERT and DELETE statements. You can additionally use the bcp utility as well as the BULK INSERT statement.

You can modify the values of base tables through a view, but several restrictions apply. When restrictions inhibit your ability to perform essential work for a database application, consider using an INSTEAD OF trigger. A trigger is a code module like a stored procedure, which is the focus of the next section. You may be able to use custom T-SQL code in an INSTEAD OF trigger to work around a general limitation for updating data through a view. Some of the most common restrictions for modifying data through a view include the following items.

- You can only reference columns from one table when attempting to change the values in a view's base tables.

- You cannot modify any aggregated values. Instead, you can modify individual column values across rows that affect aggregate values.
- You cannot alter column values that result from expressions based on other columns in the same row as well as columns from SELECT statements that include any of these terms: UNION, UNION ALL, CROSS JOIN, EXCEPT, and INTERSECT.
- A TOP clause cannot appear in a SELECT statement for a view used to modify its underlying base table values.

You can generally update values through a view within the limits listed in the preceding set of bullets. However, the WITH CHECK OPTION clause can further limit how you can update values. In particular, ***including the WITH CHECK OPTION clause in the definition for a view restricts changes so that modified values cannot cause a row to leave a view's result set. Omitting the WITH CHECK OPTION clause allows you to update values through a view so that rows can exit the current result set for a view.***

6.3 Implement a Stored Procedure

Stored procedures are code modules that are basically a batch of SQL commands. The module for the code makes the code easier to reuse than say a batch of T-SQL code in a query tab within SQL Server Management Studio (SSMS). There are several types of stored procedures:.

- ***System stored procedures -*** *are built-in stored procedures for administrative and informational activities that database administrators commonly perform.* System stored procedure names generally begin with sp_ or with an xp_ prefix for the general extended system stored procedures. Therefore, you should avoid sp_ and xp_ prefixes when naming your own user-defined stored procedures.
- ***Extended stored procedures -****.dll files, which you can create in a language, such as C, and implement by using the SQL Server Extended Stored Procedure API. An extended stored procedure typically runs in the same address space as SQL Server.* Extended stored procedure are a potential security weakness and will be removed in an upcoming version of SQL Server.
- ***User-defined stored procedures -*** *custom code modules crafted by professional programmers, administrators, or other sophisticated users.* You can code a user-defined stored procedure with either T-SQL or a .NET language, such as Visual Basic or C#. T-SQL code is optimized for data-oriented tasks, and .NET language's code is optimized for computation and string manipulation tasks.

A user-defined stored procedure facilitates creating easily reusable custom code. ***Stored procedures can optionally accept one or more parameters. In addition, they can optionally return tabular datasets, scalar values via the parameters, and status values indicating the status of a stored procedure when it terminates.***

- ***T-SQL is a set-oriented language especially designed for manipulating as well as querying data, and SQL Server is optimized to run T-SQL code. It is this combination of factors that makes T-SQL stored procedures ideal for querying database objects.***
- SQL Server always runs interpreted T-SQL code while SQL Server runs .NET language stored procedures as compiled code. This is not usually an issue because set-based procedures usually require very few statements. However, .NET language objects are compiled and execute

statements very quickly, but there is overhead working with data as you still need to use T-SQL.

- ***Stored procedures implemented with a .NET language can take advantage of the massive code libraries available through the hosted CLR within SQL Server 2005. The hosted CLR's tight integration with the SQL Server Database Engine and the enhanced security of managed code, makes writing and using CLR types more flexible and secure than extended stored procedures.***

The process for designing and running compiled code is not identical to designing and running interpreted T-SQL code. For example, you are most likely to create and test .NET language stored procedures from the Visual Studio IDE (Integrated Development Environment) instead of SSMS. During normal use, you will run .NET language stored procedures with the same EXECUTE statement that you use for T-SQL stored procedures, but you save the compiled .NET language code separately from the module holding the CREATE PROCEDURE statement that defines a stored procedure. For these reasons and the fact that T-SQL stored procedures are likely to be used commonly for query and administrative tasks, this section devotes its attention almost exclusively to T-SQL stored procedures.

SQL Server's hosted CLR facilitates the creation of multiple CLR types – not just stored procedures. Other CLR types include user-defined functions, triggers, user-defined types, and user-defined aggregates. While a CLR type uses standard CREATE statements to create a module for reference in a database, the module references the contents of a saved .dll file, which is typically a compiled version of code developed within Visual Studio. The Microsoft SQL Server Developer Center offers in-depth coverage of CLR programming techniques in its Programmability section (http://msdn.microsoft.com/sql/learning/prog/default.aspx). Use the resources in the Programmability section to learn about CLR programming skills.

Overview of System Stored Procedures

System stored procedures are very similar to T-SQL functions. However, you invoke system stored procedures with an EXECUTE (or EXEC, for short), statement. If you are just executing a single system stored procedure in a batch by itself or as the first statement of a batch, then you do not need to embed the system stored procedure within an EXEC statement. However, it is common to integrate system stored procedures in the middle of a batch of one or more other system stored procedures or T-SQL statements. Therefore, ***it is good practice to always embed a system stored procedure reference within an EXEC statement.***

There are literally dozens of system stored procedures. Because of their extensive scope, it is best to become acquainted with broad categories of system stored procedures. Then, you can drill down in those categories that look most interesting and valuable to the tasks that you have to perform. As your scope of responsibility expands, you can explore additional categories.

Quite often, system stored procedures overlap with graphical techniques or T-SQL statements. When there is overlap with a T-SQL statement, you should generally use the statement instead of the system stored procedure. This is because the T-SQL statement typically offers a more current implementation of how to best perform a task and may expose new features *(The MSDN2 library warns you when a system stored procedure is being deprecated in favor of a T-SQL statement. For an example, the library indicates at http://msdn2.microsoft.com/en-us/library/ms181422.aspx that sp_adduser is being deprecated by CREATE USER.)*. A listing of the system stored procedure category names with brief descriptions follows. Selected system stored procedure names appear in parentheses after category summaries to suggest the kinds of actions you can implement with a category.

Using System Stored Procedures

The following script shows three system stored procedure calls that return partially overlapping results for the AdventureWorks database. However, each system stored procedure serves a different kind of purpose.

```
USE AdventureWorks
GO

EXEC sp_databases
EXEC sp_helpdb N'AdventureWorks'
EXEC sp_spaceused
```

- sp_databases returns a row with three columns for each database on a SQL Server instance and any other databases through a database gateway. This particular stored procedure takes no parameters. ***If you want to easily obtain a list of the database names on the current server instance, sp_databases is an especially easy way to get it.*** The three columns for each database provide the following information
    - database name as a sysname data type, which is equivalent to nvarchar(128) in SQL Server 2005
    - database size in kilobytes
    - NULL for a REMARKS column value; sp_databases does not populate this column
- sp_helpdb can be used with and without parameters.
    - ***When used without parameters, sp_helpdb returns a single result set with a row for each database with columns for the database name and size as well as selected other properties.***
    - ***When used with a single parameter for a database's name, the system stored procedure returns two sets for the named database. The first set has the same columns whether or not you specify a parameter, except that the set has just one row for the named database. The second result includes a row for each data and log file underlying a database. The column values for these rows designate a file's logical name, identification number, operating system file name, and size along with selected other properties.***
- ***sp_spaceused can return the space used by a database divided several different ways, including an overall number for a database.*** This system stored procedure can also report the space used by a table, indexed view, or SQL Server 2005 Broker queue in the current database.
    - To obtain the space used by the current database, the system stored procedure does not require a parameter. However, you must set the database context whose size you want to learn.
    - Both sp_helpdb and sp_spaceused return just the total reserved storage for a

database, but sp_spaceused augments this value with a database's allocated and unallocated storage.

The following script illustrates the syntax for the sp_tables, sp_columns, and sp_table_privileges system stored procedures. Each of these system stored procedures drill down into the table schema for the current database context. This code sample uses named parameters to assign parameters instead of assigning values based on position as in the preceding code sample. Named parameters can help to self-document your code. (Note that you will need to have Northwind installed for this script to execute and return results.)

```
USE Northwind
GO

EXEC sp_tables @table_name = 'S%',
        @table_owner = 'dbo',
        @table_qualifier = 'Northwind'
EXEC sp_columns @table_name = 'Shippers'
EXEC sp_table_privileges @table_name = 'Shippers'

USE AdventureWorks
GO

EXEC sp_table_privileges @table_name = 'SalesPerson'
```

- ***sp_tables returns the name (along with selected other properties) for tables and views matching its @table and @table_owner parameters.*** The @table_owner parameter denotes a table's schema in SQL Server 2005. The table_qualifier is optional, but if you use the parameter, it must point at the current database.
- ***sp_columns returns column names (along with selected other column properties) for a table or view within the current database context.*** If you do not specify a parameter with a table or view name, the system stored procedure returns an error message.
- ***sp_table_privileges returns information denoting the grantee for data access permissions (INSERT, UPDATE, DELETE, REFERENCES) on a table or view specified by the @table_name parameter.*** The preceding script performs a change of the database context to control which database the stored procedure searches for a table or view.

Overview of User-defined Stored Procedures

User-defined stored procedures are excellent general purpose tools for making T-SQL (and even CLR) code re-usable. You can embed within stored procedures most any SQL statements to query and maintain data, create and manage database objects, as well as administer servers and databases. Stored procedures can also contain statements that allow branching and iterative processing. ***While other database objects make data access code reusable, such as views and user-defined functions, stored procedures can provide much of the functionality of other more specialized database objects. Furthermore, stored procedures provide richer functionality than is possible with other more specialized database objects.***

Four issues recommend stored procedures as a valuable tool for making code reusable.

- ***You can use input parameters to set variable valuesfor selected elements in T-SQL statements.*** This approach is particularly useful for criterion values in the WHERE clause of a SELECT or DELETE statement or for new or updated values in an INSERT, UPDATE statements.
- ***You can return result sets for multiple SELECT statements from a single invocation of a stored procedure.*** Views and table-valued functions on the other hand can only return the result set of a single SELECT statement.
- ***You can return one or more scalar values from a stored procedure by declaring a parameter as OUTPUT along with one or more result sets from SELECT statements.*** A user-defined function can also return a scalar value, but not two or more scalar values with a single invocation. Although a table-valued user-defined function that can return a result set for a SELECT statement, this same type of user-defined function cannot also return a scalar value.
- Finally, ***a stored procedure can return a status value, which indicates how it concludes.*** A client application can use return status values to help interpret other output from a stored procedure or why a procedure execution failed.

While input parameters allow you to reuse code by varying the values for selected elements in a static T-SQL statement, they do not allow you to alter the value for any element in a T-SQL statement. For example, you cannot replace the name of a table or a view with an input parameter and neither can you modify with an input parameter the items in a SELECT statement's list.

When you encounter situations in which an input parameter does not serve your needs for making code reusable, you can consider dynamic SQL. A dynamic SQL statement represents a T-SQL statement as a local variable in a character data type. By writing an expression for the local variable's value, you can make any part of a T-SQL variable at run time. You can embed dynamic SQL within a stored procedure, and then use input parameters to pass values to the dynamic SQL. It is common to invoke either an EXEC statement or the sp_executesql system stored procedure to run dynamic SQL within a stored procedure. Dynamic SQL does have some major downsides and should only be considered when a static SQL statement will not suffice. Dynamic SQL has some security and performance downsides that should be understood. URLs that can help you learn more dynamic SQL include these: http://www.sqlteam.com/item.asp?ItemID=4599, http://www.sqlteam.com/item.asp?ItemID=4619, http://msdn2.microsoft.com/en-us/library/ms188001.aspx

A couple of new features help to make user-defined stored procedures more powerful than in earlier versions.

- The WITH RECOMPILE clause within a CREATE PROCEDURE statement applies to all the statements within a stored procedure and tells the optimizer to re-optimize the procedure on each execution. Re-optimization causes the object to have a new query plan built to tell the query processor what steps to take to execute the query. A few additional capabilities in SQL Server 2005 enhance recompile capabilities.
    - ***The RECOMPILE query hint specifies recompilation for a single query within a stored procedure*** *(new in SQL Server 2005)*
    - ***The sp_recompile system stored procedure, which SQL Server 2005 also introduces, causes triggers and stored procedures to be re-optimized the next time they run. The sp_recompile system stored procedure can designate a table or view so that any stored procedure or trigger dependent on the table or view is re-optimized on its next use of the table or view.***

- ***The new EXECUTE AS clause for stored procedures, and other user-defined code modules, allows a code module to run in one of four different user contexts.*** The main benefit of this clause is that you can more precisely restrict permissions than with prior versions of SQL Server. For example, ***users can run a code module under a higher permission level designated by its WITH EXECUTE AS clause although neither the user nor even the code module author have permissions for objects referenced by a code module.***

Using User-defined Stored Procedures

In many cases, specifying a user-defined stored procedure with one or more input parameters will provide enough flexibility to make a stored procedure readily reusable. The following script demonstrates the use of an input parameter in a stored procedure.

```
USE PrepLogic
GO

IF OBJECT_ID('dbo.TwoWithInputParam') IS NOT NULL
        DROP PROC dbo.TwoWithInputParam
GO

CREATE PROC dbo.TwoWithInputParam
(
     @CtryName nvarchar(50) = 'United States'
)
AS
SELECT TOP 5 FirstName, LastName,
        CONVERT(decimal(12,2),SalesYTD) AS [Sales YTD],
        CountryRegionName
FROM AdventureWorks.Sales.vSalesPerson

SELECT TOP 5 FirstName, LastName,
        CONVERT(dec(12,2),SalesYTD) AS [Sales YTD],
        CountryRegionName
FROM AdventureWorks.Sales.vSalesPerson
WHERE CountryRegionName = @CtryName
GO
```

- The TwoWithInputParam stored procedure is created in the dbo schema of the PrepLogic database, references two tables in the AdventureWorks database by using three-part names.
- The second SELECT statement in the stored procedure uses the value of the input parameter named @CtryName to limit the return to only matching values in CountryRegionName.
    - ***The declaration of the input parameter appears between the stored procedure identifier (TwoWithInputParam) and the AS keyword in the CREATE PROC statement.*** You can specify multiple parameters for a stored procedure by separating them with commas.

- ***The equal sign (=) in the parameter declaration assigns a default value (United States) to the input parameter.*** If a user does not designate a parameter value at run time, the procedure runs with its default value for a parameter.
- ***The second SELECT statement shows the syntax for referencing the parameter as the criterion value for a criterion expression in its WHERE clause.***

The following two EXEC statements illustrate two ways of invoking the TwoWithInputParam stored procedure.

EXEC dbo.TwoWithInputParam

EXEC dbo.TwoWithInputParam 'Canada'

- The first EXEC statement runs the stored procedure with the default value for the input parameter. Notice there is no value for the input parameter.
- The second EXEC statement assigns a value of Canada to the @CtryName input parameter. ***You can also use an expression that specifically assigns a value to a parameter by name (for example, @CtryName = N'Canada'). As noted previously for system stored procedures, using an expression to assign a parameter's value is most helpful when you have multiple input parameters and you want your code to document itself.***

The final code sample for this chapter demonstrates the syntax for using output parameters and return status values with a stored procedure. Both types of return values are scalar values.

- ***You can explicitly declare an output parameter to be a designated data type (just like an input parameter), but a return status value is always an integer value.***
- You specify output parameters in the same area of a stored procedure that you specify input parameters, but each output parameter must end with the keyword OUTPUT.
- ***A stored procedure can have just one return status value each time it runs. However, the return status value can change from one run to the next one depending on the last line that the stored procedure executes on each run.***
    - You assign a value to a stored procedure's status value with the RETURN statement within a CREATE PROCEDURE statement.
    - The RETURN statement unconditionally terminates a stored procedure.
    - The RETURN statement can pass back a constant or the value of an expression.
    - A stored procedure's design can have multiple RETURN statements each passing back a different value to reveal to a calling routine or T-SQL batch where the called routine concluded.
    - When a stored procedure encounters a RETURN statement at run time, it is always the last line that a stored procedure executes.

The following script creates a stored procedure named OutParamStatus in the dbo schema of the PrepLogic database. If the input parameter (@int1) is valid, the procedure computes a value for the output parameter (@plusone.)

```
USE PrepLogic
GO
IF OBJECT_ID('dbo.OutParamStatus') IS NOT NULL
        DROP PROC dbo.OutParamStatus
GO

CREATE PROC dbo.OutParamStatus
(
          @int1 tinyint,
          @plusone tinyint OUTPUT
)
AS
     IF UNICODE(@int1) IS NULL
 -- UNICODE returns the Unicode character
 -- corresponding to an integer
       BEGIN
        PRINT 'NULL input value.'
        RETURN 0
       END
     IF @int1 < 0 OR @int1 > 9
        BEGIN
        PRINT 'Input not ok.'
        RETURN 1
        END
     ELSE
        BEGIN
        SET @plusone = @int1+1
        PRINT 'Input ok.'
        RETURN 2
        END
GO
```

- ***Input and output parameters are declared in the same area of the CREATE PROC statement. The key difference is that the output parameter declaration ends with the OUTPUT keyword.***
- The code within the stored procedure consists of a pair of IF statements to determine if @int1 is valid (the code sample defines @int1 values between 0 and 9 as valid) and compute the value of the output parameter if it is valid.
    - The first IF statement checks if the input parameter (@int1) is NULL. If yes, the procedure prints a message to that effect to the Messages pane in SSMS and assigns a value of 0 to the return status value before ending. A BEGIN…END block is necessary because the IF statement has to execute more than one statement when its condition is true.
    - The second IF statement checks if the input parameter has a value from 0 through

9. If the value is not in this range, the IF statement returns a message and exits with a return status value of 1. If the input parameter value is 0 through 9, then the second IF statement's ELSE clause assigns a value to the output parameter (@plusone) of one greater than the input parameter value. In addition, the ELSE clause assigns a value of 2 to the return status value.

The following script shows the code for invoking the OutParamStatus stored procedure with three different input parameter values. A DECLARE statement defines local variables for the script to store the return status value (@rs) and the output parameter value (@inplusone). The @in local variable represents the value for the input parameter in the script before it is sent to the stored procedure.

- The EXEC statement recovers the return status value from the stored procedure by assigning the stored procedure to the local variable for the return status value (@rs).
- The EXEC statement designates the output parameter by trailing the local variable for the output parameter (@inplusone) with the OUT keyword. You can replace OUT with OUTPUT.

The first, second, and third stored procedure invocations assign values of NULL, 99, and 9 to the input parameter. The output listing, which follows the script for each invocation, shows the outcome from each invocation. The use of the SET NOCOUNT setting in the script suppresses information about the number of rows affected in the output listing.

```
SET NOCOUNT ON
DECLARE @rs AS int, @in tinyint, @inplusone tinyint

SET @in = NULL
EXEC     @rs = dbo.OutParamStatus @in, @inplusone OUT
SELECT @rs AS [Return Status value],
            @in AS [Input parameter value],
            @inplusone  AS [Output parameter value]
```

Returns:

```
NULL input value.

Return Status value Input parameter value Output parameter value
------------------- ---------------------- ----------------------
0                   NULL                   NULL
```

```
SET @in = 99
EXEC @rs = dbo.OutParamStatus @in, @inplusone OUT
SELECT @rs AS [Return Status value],
             @in AS [Input parameter value],
             @inplusone  AS [Output parameter value]
```

Returns:

```
Input not ok.

Return Status value Input parameter value Output parameter value

------------------- --------------------- ----------------------

1                   99                    NULL
```

```
SET @in = 9
EXEC @rs = dbo.OutParamStatus @in, @inplusone OUT
SELECT @rs AS [Return Status value],
            @in AS [Input parameter value],
            @inplusone  AS [Output parameter value]
```

```
Returns:


Input ok.

Return Status value Input parameter value Output parameter value

------------------- --------------------- ----------------------

2                   9                     10
```

```
SET NOCOUNT OFF
```

6.4 Other Database Object Topics

This chapter drilled down on techniques associated with creating and using tables, views, and user-defined stored procedures. In addition, the chapter provides in-depth coverage of system stored procedures. You also received references to learn more about CLR types, such as CLR-based stored procedures.

After reviewing the content in this chapter, you should be comfortable about creating and using database objects generally. You'll use similar development techniques when designing other kinds of database objects beyond those explicitly covered in this chapter. In your preparation for the 70-431 examination, you should use the following URLs to drill down on other kinds of database objects and related topics besides tables, views, and stored procedures.

- User-defined functions: http://msdn2.microsoft.com/en-us/library/ms189593.aspx
- DML Triggers: http://msdn2.microsoft.com/en-us/library/ms191524.aspx
- DDL Triggers: http://msdn2.microsoft.com/en-us/library/ms190989.aspx
- Alias user-defined types: http://msdn2.microsoft.com/en-us/library/ms189283.aspx

Full-text queries: http://msdn2.microsoft.com/en-us/library/ms142571.aspx

Practice Questions

Chapter 1 Installing and Configuring SQL Server 2005

1. Which of the following system stored procedures and system catalog views let you list all the databases on a server?

   Choose all that apply.

   - ❑ A. sys.all_databases
   - ❑ B. sp_helpdb
   - ❑ C. sp_helpdb database_name
   - ❑ D. sp_databases
   - ❑ E. sys.databases

2. You can upgrade a prior SQL Server instance to SQL Server 2005 by installing SQL Server 2005 over the prior SQL Server installation. You can also upgrade individual databases from prior versions for use with SQL Server 2005 through of any of several contrasting techniques. Which of the following statements correctly describe how to upgrade instances and databases from prior SQL Server versions to SQL Server 2005?

   Choose all that apply.

   - ❑ A. You can run SQL Server 2005 setup to perform most upgrade operations for migrating a SQL Server 7 instance or a SQL Server 2000 instance to a SQL Server 2005 instance.
   - ❑ B. SQL Server 2005 does not support the following network protocols that were enabled by prior versions of SQL Server Banyan VINES Sequenced Packet Protocol (SPP), Multiprotocol, AppleTalk, NWLINK IPX/SPX network protocols, or shared memory protocol.
   - ❑ C. You can use the Copy Database Wizard to upgrade a database from a prior SQL Server version to one that works with SQL Server 2005. After completing the copying of the database between version instances, run the sp_updatestats system stored procedure to optimize performance of the copied database.
   - ❑ D. You can backup a SQL Server 7 or SQL Server 2000 database and restore the backup set on a SQL Server 2005 instance. Because SQL Server 2005 has different file locations for databases in default and named instances, you will likely need to specify the MOVE option with RESTORE when restoring a backup to SQL Server 2005 from a backup created on an instance for a prior version of SQL Server.
   - ❑ E. When attempting to upgrade a database on a SQL Server 6.5 instance for use on a SQL Server 2005 instance, you cannot restore the database to SQL Server 2005 from a backup media set created in SQL Server 6.5. However, you can use either SQL Server Integration Services or the bcp utility.

3. SQL Server 2005 introduces a new means of sending mail from a SQL Server instance. Database Mail deprecates the SQL Mail, which was introduced with SQL Server 7. Which of the following statements correctly describe Database Mail architecture and how to configure Database Mail? Choose all that apply.

   - ❑ A. One of the main advantages of Database Mail versus SQL Mail is that you no longer require Microsoft Outlook or another Extended Messaging Applications Programming Interface client.
   - ❑ B. Before you can configure Database Mail, you must enable it for a SQL Server instance. There are just two ways to do this. First, you can use a Transact-SQL script to configure the Database Mail extended stored procedures. Second, you can choose Enable Database Mail stored procedures from within the Surface Area Configuration tool.
   - ❑ C. Database Mail permits any user within a SQL Server instance to send mail provided the user has Execute permission for the sp_send_dbmail system stored procedure. Part of what makes Database Mail so powerful is that no other restrictions apply.
   - ❑ D. When configuring database mail, you can create one or more main accounts. Each main account can have one or more sub-accounts. Users connect to mail servers via sub-accounts that belong to main accounts.
   - ❑ E. Any Database Mail profile can have one or more accounts that belong to it. However, the same account can belong to just one profile.

Chapter 2 Implementing High Availability and Disaster Recovery

1. One of the most important reasons for implementing log shipping is so that you can failover from the primary database to a secondary database. Two common reasons mandating a failover is planned maintenance on the primary server or damage to the primary database. Which of the following statements accurately describe steps for failing over from a primary database to a secondary database for planned maintenance on the primary database?

   Choose all that apply.

   - ❑ A. Copy any log backups from the backup file share to the secondary server to which you are failing over. Then, restore any unapplied log backups on the secondary server with NORECOVERY.
   - ❑ B. Perform a log backup to capture the active log since the last scheduled log backup. Then, backup the log again to get the tail after the active log. Apply these log backups to the secondary database.
   - ❑ C. When you perform the final log backup on the primary server use a RECOVERY setting. This special use of the RECOVERY setting for a primary server, in a log shipping configuration, permits you to restore the primary server from the secondary server to capture any changes to the secondary database after it is switched to a primary database.
   - ❑ D. The first time that you switch a secondary server to a primary server, and vice versa, requires two types of tasks. First, configure the secondary server being switched to a primary server for log shipping. Designate the original primary database as the secondary database for the new log shipping configuration. Second, disable the old backup, copy, and restore jobs and create new jobs based on the new server roles.
   - ❑ E. After you have initially switched roles between the primary and secondary servers, you can switch again just by making a selection in SQL Server Management Studio to switch the primary and secondary servers.

Chapter 3 Supporting Data Consumers

1. Manipulating relational data allows you to insert, modify, and delete data from database objects, such as tables, views, and table-valued user-defined functions. Select the following statements which correctly describe how you can accomplish these tasks with Transact-SQL statements.
Choose all that apply.

   - ❑ A. To remove a table's data and definition from a database, you can use either a TRUNCATE TABLE statement or a DROP TABLE statement, as opposed to a DELETE statement.
   - ❑ B. The only way to remove a table and its data from a database that is referenced by a foreign key in another table is to first drop the table with the foreign key. You can rebuild the table with the foreign key after removing the table to which the foreign key refers.
   - ❑ C. You can create a permanent table based on the items in a SELECT list with the INTO clause of a SELECT statement. In order for the INTO clause for a SELECT statement to operate successfully in SQL Server 2005, the select into/bulkcopy database option must be set.
   - ❑ D. SQL Server 2005 introduces a new OUTPUT clause for data modification statements that lets you directly access the INSERTED and DELETED tables without using a trigger. This new feature simplifies tasks, such as custom auditing features.
   - ❑ E. When the SET TRANSACTION ISOLATION LEVEL statement specifies READ COMMITTED, SQL Server 2005 blocks attempts to read rows that are in the process of being modified by another transaction so long as the READ_COMMITTED_SNAPSHOT database option is set to ON.

2. You can use Transact-SQL code and collation naming conventions to manipulate collation settings. Which items below correctly describe how to apply collations to server instances, databases, and database objects?
Choose all that apply.

   - ❑ A. The following statement returns all the collations that a SQL Server instance is set to process. SELECT * FROM fn_helpcollations ().
   - ❑ B. The Latin_CI_AI collation is insensitive to width and Kana type characters just like Latin_CS_AS.
   - ❑ C. The _BIN suffix in a collation name specifies sorting based on Unicode code point values for Unicode data and character bit pattern representations for non-Unicode data type values.
   - ❑ D. The COLLATE clause allows you to assign a collation to an object, such as a database, a schema, a table, or a column. You can specify a Windows collation name or a SQL collation name as an argument for the COLLATE clause.
   - ❑ E. You can use the ALTER TABLE statement to modify the collation for a column unless the column: (1) is a computed column, (2) has query statistics, (3) is used in an index, (4) is part of a CHECK constraint or a foreign key constraint.

Chapter 4 Maintaining Databases

1. You want to set the recovery model for a database so that it is simple to recover all logged changes, to a database, from the database log. You decide to do this by modifying an existing database and setting a database option so that the database operates as you wish. You use a T-SQL statement with the following format.

   ALTER DATABASE database_name

   SET option_name option_value

   Select the SET statement that achieves your objective.

   - ❍ A. SET BACKUP FULL
   - ❍ B. SET SIMPLE FULL
   - ❍ C. SET RECOVERY SIMPLE
   - ❍ D. SET FULL RECOVERY
   - ❍ E. SET RECOVERY FULL

Chapter 5 Creating and Implementing Database Objects

1. Triggers help you extend the application of your database applications by permitting the specification of constraints across multiple tables or even allowing referential integrity across multiple databases. In order to take proper advantage of triggers, you must follow the rules for properly constructing triggers. Which statements below reflect valid rules for creating and using triggers?

   Choose all that apply.

   - ❑ A. You can use the CREATE TRIGGER statement to create both AFTER and INSTEAD OF DML triggers. However, you must use the CREATE DDL_TRIGGER statement to create DDL triggers.
   - ❑ B. A private trigger identifier can start with #. However, a global temporary trigger identifier starts with ##.
   - ❑ C. INSTEAD OF triggers can only be specified for an updatable view when the view has a WITH CHECK OPTION clause.
   - ❑ D. DML triggers support nvarchar(max), varchar(max), and varbinary(max) columns in inserted and deleted tables.
   - ❑ E. When one trigger performs an action that initiates another trigger, the second trigger is nested within the first trigger. Your applications can nest triggers within one another up to 32 levels.

2. Binding a view to its data sources provides a variety of advantages. Which of the following advantages become available, and which special requirements apply, when you bind a view to its base data sources?

   Choose all that apply.

   - ❑ A. You must specify the SCHEMA BINDING view attribute in the CREATE VIEW or ALTER VIEW statement.
   - ❑ B. You must reference all base data sources within the view with two-part or three-part names, such as schema_name.table_name or database_name.schema_name.table_name.
   - ❑ C. All the data sources for the view can come from two or more databases so long as all databases have the same name. For example, one database named database_name can reside on server1, and a second database named database_name can reside on server2.
   - ❑ D. Any indexed view cannot include SCHEMA BINDING as a view attribute as part of its definition.
   - ❑ E. SCHEMA BINDING can be specified whether or not the view contains alias data type columns.

3. When you create user-defined function with a .NET language for the embedded SQL Server 2005 CLR which of the following rules are valid?

   Choose all that apply.

   - ❑ A. The ability of SQL Server 2005 to execute CLR code is off by default. However, SQL Server administrators can turn the capability on and off with the SQL Server Configuration Manager.
   - ❑ B. There are three fundamental steps to creating a CLR user-defined function. First, author and compile a static (shared) class method in a .NET language (Visual Basic 2005 or C#). Second, register in a SQL Server database the library file for the compiled class with the CREATE ASSEMBLY statement. Third, use the CREATE FUNCTION statement to reference the registered assembly.
   - ❑ C. Two typically used clauses for a CREATE ASSEMBLY statement are the FROM clause and the WITH PERMISSION_SET clause. You can use the FROM clause to reference a file that contains just the .NET source code. You can assign a permission scope with the WITH PERMISSION_SET clause.
   - ❑ D. A .NET static (shared) class method for a user-defined function must be decorated with a SqlFunction attribute. This attribute allows you to specify parameters that determine how the SQL Server Database Engine processes the compiled code. For example, set the DataAccess parameter to DataAccessKind.Read if your user-defined function reads from a SQL Server database.
   - ❑ E. The SQL Server Database Engine can automatically infer selected Transact-SQL user-defined function attributes, such as whether a function is deterministic or whether a function returns a precise value (one not based on floating point data types). One of the advantages of Service Pack 1 for SQL Server 2005 is that it enables the Database Engine to infer these function attributes for CLR attributes.

Answers and Explanations

Chapter 1

1. Answers: B, D, E

Explanation A. Incorrect. There is no system catalog view named sys.all_databases.

Explanation B. Correct. When used without an argument for a specific database name, sp_helpdb returns a result set with the name, and selected other information, for all databases on a SQL Server instance.

Explanation C. Incorrect. When used with an argument for a specific database name, sp_helpdb returns two result sets for the database name argument. An argument should not be used, because the result set returns information for just one database, the argument's value, instead of all the databases on a server.

Explanation D. Correct. The sp_databases system stored procedure returns the name and size of databases in kilobytes of databases on the current server instance, as well as those on other server instances that are accessible through a database gateway.

Explanation E. Correct. The sys.databases view returns a result set with one row per database on the current server instance.

2. Answers: A, C, D, E

Explanation A. Correct. SQL Server 2005 setup can upgrade SQL Server 2000 and SQL Server 7 instances, but setup cannot upgrade instances based on a version prior to SQL Server 7. Additional restrictions apply to editions. For example, you can upgrade a SQL Server 7 MSDE instance to a SQL Server Express instance, but not to an instance for any other SQL Server 2005 edition.

Explanation B. Incorrect. Shared memory is one of the protocols supported by SQL Server 2005, but all other named protocols are not supported. If you have an application connecting with an obsolete protocol, you must switch to one of the protocols supported by SQL Server 2005. In addition to shared memory, these protocols include: TCP/IP, named pipes, and Virtural Interface Adapter software (VIA) with VIA hardware.

Explanation C. Correct. The Copy Database Wizard has a number of special requirements that may make its use inappropriate for some installations. For example, you must have SQL Server 2005 Integration Services installed and configured for use with the Copy Database Wizard. If either of these two requirements is not met, you can use other database migration techniques.

Explanation D. Correct. Learn the proper default location for data and log files for an instance and use the MOVE option in RESTORE to specify either the default location or an alternate preferred location. In any event, if you try to restore a data or log file to a non-existent drive, the RESTORE statement fails.

Explanation E. Correct. When using the bcp utility, use bcp out on the SQL Server 6.5 instance to export data in a character format. Then, use bcp in with the -V option (or switch) to import the data into SQL Server 2005.

3. Answers: A, D

Explanation A. Correct. Database Mail is a native SQL Server 2005 subsystem that can use any Simple Mail Transport Protocol (SMTP) instance to send mail. In fact, Database Mail supports failover accounts. If an account for one SMTP server mail is unresponsive, Database Mail will try any other SMTP servers that you specify.

Explanation B. Incorrect. There is a third way to enable Database Mail. You can launch the Database Mail Configuration Wizard from within SQL Server Management Studio. If necessary, the wizard will enable Database Mail the first time that you start it for a SQL Server instance.

Explanation C. Incorrect. To send mail via Database Mail, a user must be a database user in msdb. In addition, the user must belong to the DatabaseMailUserUser database role. These permissions enable the use of all public profiles. Database Mail also offers private profiles that you can specify for the exclusive use of designated msdb security principals.

Explanation D. Incorrect. Database Mail uses accounts to specify the information for connecting to an SMTP server. There are no sub-accounts. However, any one account can belong to multiple profiles. Explanation E. Incorrect. Any one account can belong to more than one profile. In other words, there is a many-to-many relationship between accounts and profiles.

Chapter 2

1. Answers: A, B, D

Explanation A. Correct. Using the NORECOVERY setting leaves the database in restoring state so that you can apply additional backups, if necessary. Using a RECOVERY setting makes the database available for immediate use but makes it impossible to apply additional backups (because it is not in a restoring state).

Explanation B. Correct. Applying these log backups fully synchronizes the secondary database with the primary database. Of course, you will loose any uncommitted transactions from the primary database.

Explanation C. Incorrect. The last log backup on the primary database should be with NORECOVERY instead of with RECOVERY. The NORECOVERY setting leaves the database in a restoring state so that you can apply log backups from the secondary database converted to a primary database.

Explanation D. Correct. Of course, you also need to point client applications at the new database acting as a primary database. This change permits all changes to occur on the new primary database.

Explanation E. Incorrect. There is no graphical option for switching servers, but you can employ just three simple manual steps. First, perform backup and restore operations. Second, disable the backup, copy, and restore jobs for the log shipping configuration with the former primary database. Third, enable replacement backup, copy, and restore jobs for the configuration with the secondary database acting as a primary database.

Chapter 3

1. Answer: D

Explanation A. Incorrect. The DROP TABLE statement alone removes a table's definition and data as well as its triggers, indexes, and user permissions. The TRUNCATE TABLE statement merely removes all the rows of data from a table without recording the changes in the transaction log. The DELETE statement can remove one or more rows from a data source, but it records changes in the transaction log.

Explanation B. Incorrect. A much simpler approach is to just drop the foreign key in the second table. Then, you can drop the table to which the foreign key refers. This eliminates the need to drop and rebuild the table with the foreign key.

Explanation C. Incorrect. The select into/bulkcopy database option only applies to the operation of the INTO clause for a SELECT statement for SQL Sever versions prior to SQL Server 2000. The select into/bulkcopy database option does not affect the operation of the INTO clause in a SELECT statement for either SQL Server 2000 or SQL Server 2005.

Explanation D. Correct. The OUTPUT clause applies to INSERT, UPDATE, and DELETE statements. The OUTPUT clause does not apply when you are using partitioned views, remote tables, or an EXECUTE statement within an INSERT statement.

Explanation E. Incorrect. The READ_COMMITTED_SNAPSHOT database option must be set to OFF for the READ COMMITTED transaction isolation level to block reading rows in the process of being modified. If the database option is ON, then SQL Server 2005 uses row versioning to provide each transaction a consistent dataset from the start of its transaction.

2. Answers: B, E

Explanation A. Incorrect. The result set for the function is not constrained by the collation settings for a particular SQL Server instance. The fn_helpcollations function returns the name and description of all the collations supported by SQL Server 2005.

Explanation B. Correct. Unless kana and width sensitivity are explicitly specified by a suffix in a collation name, the collation is insensitive to both features. The Latin1 collation designator is applicable to locale variations of English, such as United States and Great Britain, as well as many European languages, such as Dutch and German. CI/CS designates case sensitivity, and AI/AS denotes accent sensitivity.

Explanation C. Incorrect. The description is for the _BIN2 suffix in a collation name. SQL Server 2005 introduces the _BIN2 suffix. The _BIN suffix sorts both Unicode and non-Unicode characters based on character bit pattern representations. Without the specification of either of these suffixes in a collation name, sorting is based on the dictionary for a language.

Explanation D. Incorrect. In the CREATE TABLE and ALTER TABLE statements, the COLLATE clause can apply to an individual column -but not a whole table. In addition, you cannot use a COLLATE clause in CREATE SCHEMA or ALTER SCHEMA statements. The COLLATE clause can be used with CREATE DATABASE and ALTER DATABASE statements.

Explanation E. Correct. Use the COLLATE clause in the ALTER TABLE statement to assign a collation to a column. Include the COLLATE clause in the column specification. You can also initially specify a collation for a table column in a CREATE TABLE statement with the COLLATE clause.

Chapter 4

1. Answer: E

Explanation A. Incorrect. There is no database option named BACKUP . FULL is the correct option value for the RECOVERY option.

Explanation B. Incorrect. There is no database option named SIMPLE. FULL is the correct option value for the RECOVERY option.

Explanation C. Incorrect. There is a database option named RECOVERY. However, the SIMPLE option value does not enable log backups and their recovery.

Explanation D. Incorrect. There is no database option named FULL. In addition, there is no option value named RECOVERY.

Explanation E. Correct. The FULL option value setting for the RECOVERY option name causes all logged changes to be saved in a database's log. Invoke this SET RECOVERY FULL clause from within an ALTER DATABASE statement.

Chapter 5

1. Answers: D, E

Explanation A. Incorrect. Use the CREATE TRIGGER statement to generate both DML and DDL triggers. The keywords and some of the clause names for a DML trigger are different than for a DDL trigger. For example, the ON clause with a DML trigger can take the name of a table or view, but the same clause in a DDL trigger takes one of two keywords (ALL SERVER or DATABASE).

Explanation B. Incorrect. SQL Server does not allow temporary triggers. When you no longer need a trigger, you can invoke the DROP TRIGGER statement to remove it from a database. Alternatively, you can invoke the DISABLE TRIGGER statement to temporarily disable a trigger that you can invoke again later with an ENABLE TRIGGER statement.

Explanation C. Incorrect. INSTEAD OF triggers cannot be specified for an updatable view when the view has a WITH CHECK OPTION clause. This option disallows database changes that make a row disappear from a view.

Explanation D. Correct. The nvarchar(max), varchar(max), and varbinary(max) data types introduced with SQL Server 2005 deprecate, respectively, the ntext, text, and image data types. DML triggers do not support ntext, text, and image in inserted and deleted tables.

Explanation E. Correct. You can nest both DML and DDL triggers. If two or more triggers construct an infinite loop for their invocation, the sequence automatically stops at the 32nd level of the invocation.

2. Answer: A

Explanation A. Correct. The correct name for the view attribute that binds a view to its data sources is SCHEMA BINDING. Specifying this attribute for a view restricts changes to the base table that would affect the view definition. For example, you cannot drop a table if a view definition with a SCHEMA BINDING view attribute references the table.

Explanation B. Incorrect. You only require two-part names when specifying the data sources for a view with schema binding.

Explanation C. Incorrect. All the data sources for a view with schema binding must come from just one database on a single server instance. A view with base data sources from two or more different databases on the same or different servers cannot be used for schema binding.

Explanation D. Incorrect. Any indexed view must include SCHEMA BINDING as a view attribute as part of its definition. In addition, an indexed view must have a unique clustered index.

Explanation E. Incorrect. A view with alias data type columns cannot have a SCHEMA BINDING view attribute in its definition. An alias data type is a special specification of a system data type. You can use alias data types to ensure that the same data is stored identically in multiple tables within a database.

3. Answers: B, D

Explanation A. Incorrect. It is true that the ability of SQL Server 2005 to execute CLR code is off by default, but you turn the capability on and off with the SQL Server Surface Area Configuration tool -not SQL Server Configuration Manager. You can also use Transact-SQL for the same purpose by setting the clr_enabled option with sp_configure.

Explanation B. Correct. Notice that the CREATE FUNCTION statement does not directly process .NET source code. Instead, the statement references the registered assembly for a compiled .NET class.

Explanation C. Incorrect. The FROM clause normally points at the compiled library file based on the source code -not the original source code. In addition, the WITH PERMISSION_SET clause can specify one of three permissions: SAFE, EXTERNAL_ACCESS, or UNSAFE.

Explanation D. Correct. If your user-defined function attempts to query a database without setting the DataAccess parameter to DataAccessKind.Read, it will fail. Another especially useful SqlFunction parameter is the TableDefinition parameter. This parameter allows you to specify the output table format for a table-value user-defined function.

Explanation E. Incorrect. The Database Engine does not automatically infer whether a CLR user-defined function is deterministic or precise. Instead, the author of the CLR user-defined function must declare these attributes with SqlFunction parameters. The parameter names are IsDeterministic and IsPrecise. You assign true or false to each parameter.