Module 15: Appendix B - Installation Considerations

Contents:

Module Overview

Before you start to deploy Microsoft® SQL Server®, it is important to plan appropriately. As with almost any type of project, the installation work is always easier with the benefit of good planning.

To appreciate the requirements for SQL Server, you first need to gain an understanding of its architecture. In this module, you will see how SQL Server is structured and the requirements it places on the underlying server platform.

In larger enterprises, there is a common need to install many SQL Server instances while ensuring that the installations are performed consistently. You will see options for automating the installation of SQL Server. This can also be useful for silent installations of the product where SQL Server needs to be installed in the background to support a software application.

Objectives

After completing this module, you will be able to:

- Describe the SQL Server architecture.
- Describe how to automate SQL Server installations.

Lesson 1: Overview of SQL Server Architecture

Before you start looking at the resources that SQL Server requires from the underlying server platform, you need to gain some knowledge of how SQL Server functions, so that you can understand why each resource requirement exists. To be able to read SQL Server documentation, you also need to become familiar with some of the terminology used when describing how the product functions.

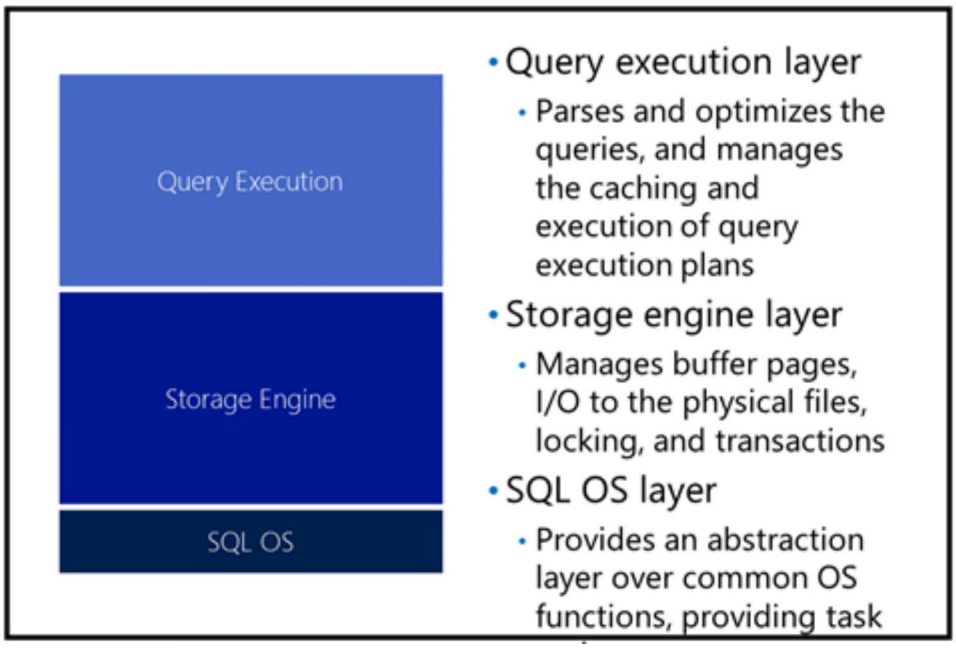
The most important resources that SQL Server utilizes from a server platform are CPU, memory, and I/O. In this lesson, you will see how SQL Server is structured and how it uses each of these.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe SQL Server architecture.
- Explain CPU usage by SQL Server.
- Describe the role of parallelism.
- Explain how 32-bit and 64-bit servers differ, with respect to SQL Server.
- Describe how SQL Server uses memory.
- Describe the difference between physical and logical I/O.

SQL Server Architecture



SQL Server is constructed from a series of many small components that work together. Three general categories of components exist within the database engine and are structured as layers – query execution (also referred to as the relational engine), storage engine, and SQL OS.

Query Execution Layer

As well as managing the query optimization process, the query execution layer manages connections and security. A series of sub-components help it to work out how to execute your queries:

- The parser checks that you have followed the rules of the Transact-SQL language and outputs a syntax tree, which is a simplified representation of the queries to be executed. The parser outputs what you want to achieve in your queries.
- The algebrizer converts the syntax tree into a relational algebra tree, where operations are represented by logic objects rather than words. The aim of this phase is to take the list of what you want to achieve, and convert it to a logical series of operations representing the work that needs to be performed.
- The query optimizer then considers the different ways that your queries could be executed and finds an acceptable plan. The costs are based on the required operation and the volume of data that needs to be processed, which is calculated taking into account the distribution statistics. For example, the query optimizer considers the data that needs to be retrieved from a table and the indexes available on the table, to decide how to access data in the table. It is important to realize that the query optimizer does not always look for the lowest

cost plan as, in some situations, this might take too long. Instead, the query optimizer finds a plan which it considers satisfactory. The query optimizer also manages a query plan cache to avoid the overhead of performing all this work, when another similar query is received for execution.

Storage Engine Layer

The storage engine layer manages the data held within databases. The main responsibilities of the storage engine layer are to manage how data is stored, both on disk and in memory; to manage how the data is cached for reuse; and to manage the consistency of data through locking and transactions. The main sub-components of the storage engine layer are as follows:

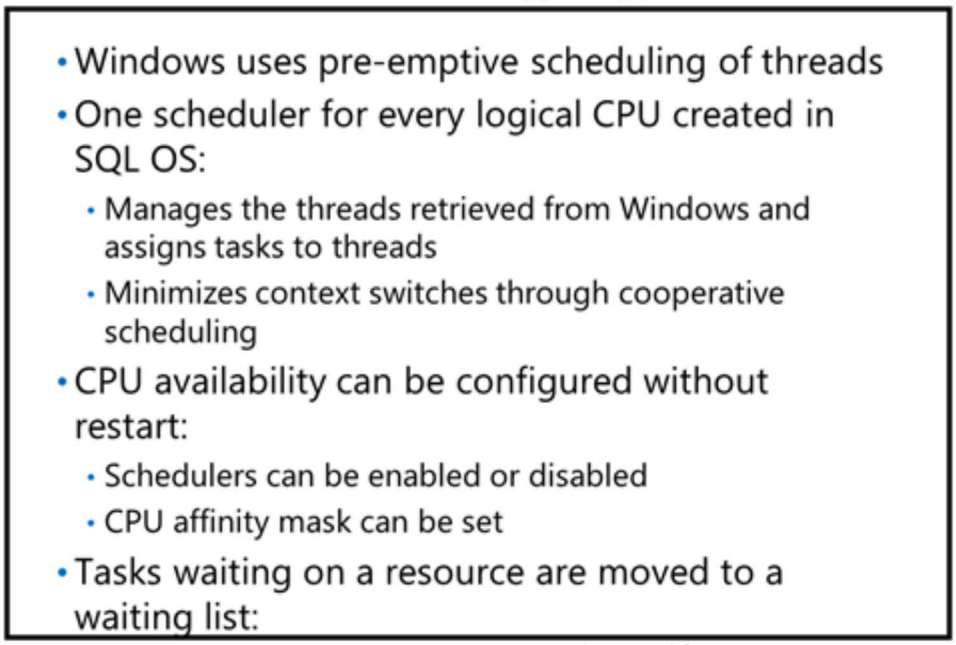
- The access methods component is used to manage how data is accessed. For example, the access methods component works with scans, seeks, and lookups.
- The page cache manages the storage of cached copies of data pages. Caching of data pages is used to minimize the time it takes to access them. The page cache places data pages into memory, so they are present when needed for query execution.
- The locking and transaction management components work together to maintain consistency of your data. This includes the maintenance of transactional integrity, with the help of the database log file.

SQL OS Layer

SQL OS is the layer of SQL Server that provides operating system functionality to the SQL Server components. All SQL Server components use programming interfaces provided by SQL OS to access memory, schedule tasks, or perform I/O.

The abstraction layer provided by SQL OS avoids the need for resource-related code to be present throughout the SQL Server database engine code. The most important functions provided by this layer are memory management and scheduling. These two aspects are discussed in more detail later in this lesson.

CPU Usage by SQL Server



All work performed in the Windows® operating system is based on the execution of threads. Windows uses pre-emptive scheduling of threads to maintain control. Rather than requiring threads to voluntarily give up control of the CPU, pre-emptive systems have a clock that is used to interrupt threads when their allocated share of CPU time has completed. Threads are considered to have encountered a context switch when they are pre-empted.

SQL Server Threads

SQL Server retrieves threads from Windows. A SQL Server configuration setting (max worker threads) determines how many threads will be retrieved. SQL Server has its own internal scheduling system, separate from the scheduling performed by the operating system. Instead of using Windows threads directly, SQL Server creates a pool of worker threads that are mapped to Windows threads whenever work needs to be performed.

When a SQL Server component needs to execute code, the component creates a task which represents the unit of work to be done. For example, if you send a batch of Transact-SQL commands to the server, it is likely that it will be executed within a task.

When a task is created, it is assigned the next available worker thread that is not in use. If no worker threads are available, SQL Server will try to retrieve another Windows thread, up to the point that the max worker threads configuration limit is reached. At that point, the new task would need to wait to get a worker thread.

All tasks are arranged by the SQL Server scheduler until they are done.

Affinity Mask

Schedulers can be enabled and disabled by setting the CPU affinity mask on the instance. The affinity mask is a configurable bitmap that determines which CPUs from the host system should be used for SQL Server, and can be changed without the need for rebooting. By default, SQL Server will assume that it can use all CPUs on the host system. While it is possible to configure the affinity mask bitmap directly by using sp_configure, use the properties window for the server instance in SQL Server Management Studio (SSMS) to modify processor affinity.

> **Note:** Whenever the term CPU is used here in relation to SQL Server internal architecture, it refers to any logical CPU, regardless of whether core or hyper-threading CPUs are being used.
> SQL Server licensing is currently based on physical CPU sockets rather than on logical CPUs. If affinity settings are used to violate SQL Server licensing limits, the system detects this at startup and logs details about the violation in the event log.

Waiting for Resources

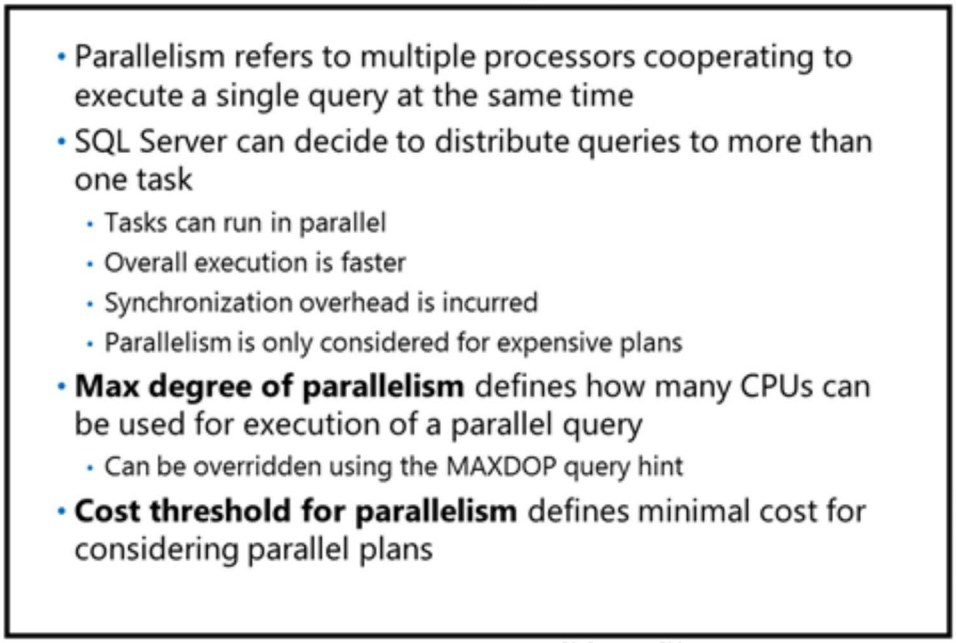
One concept that might be difficult to grasp at first is that most SQL Server tasks spend most of their time waiting for something external to happen. Most of the time, they are waiting for I/O or the release of locks, but this can involve other system resources.

When a task needs to wait for a resource, it is placed on a list until the resource is available. When the resource is available, the task is signaled that it can continue, though it still needs to then wait for another share of CPU time. This allocation of CPUs to resources is a function of the SQL OS.

SQL Server keeps detailed internal records of how long tasks spend waiting and of the types of resources they are waiting for. You can see these details by querying the following system views:

```
sys.dm_os_waiting_tasks;
sys.dm_os_wait_stats;
```

Parallelism



SQL Server normally executes queries using sequential plans on a single task. To reduce overall time spent on this, SQL Server can decide to distribute queries over several tasks so that it can execute them in parallel.

Parallel Execution Plans

Parallel execution involves the overhead of synchronizing and monitoring the tasks. Because of this, SQL Server only considers parallel plans for expensive queries, where the advantages outweigh the additional overhead.

The query optimizer decides whether a parallel plan should be used, based on two aspects of the query and of the system configuration:

- A value called the Maximum Degree of Parallelism (MAXDOP) determines a limit for how many CPUs can be used to execute a query.
- Another value, called the Cost Threshold for Parallelism, determines the cost that a query must meet before a parallel query plan will even be considered.

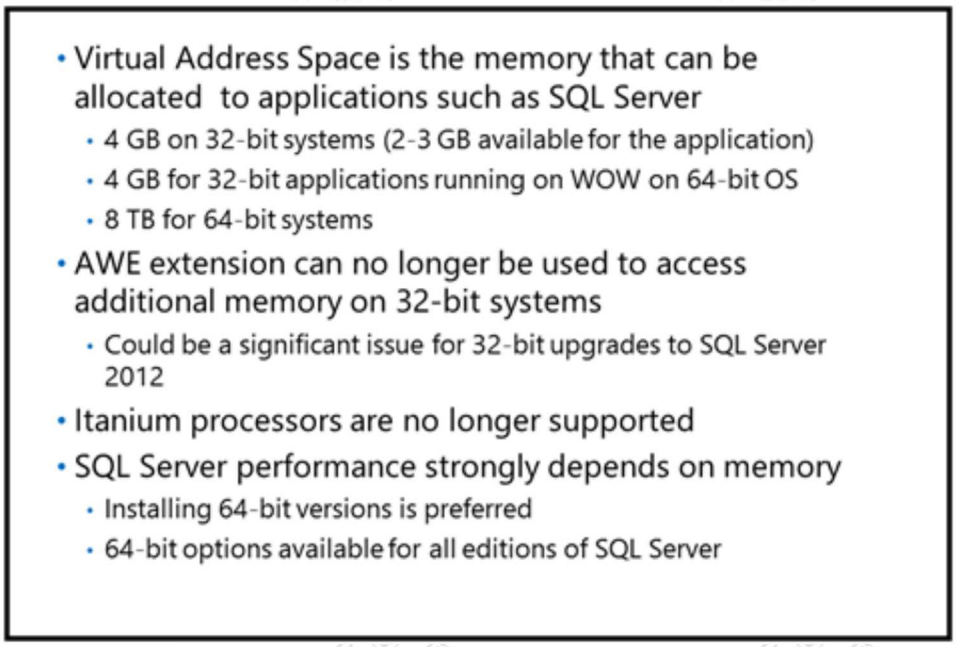
If a query is expensive enough to consider a parallel plan, SQL Server might still decide to use a sequential plan that is lower in overall cost.

Controlling Parallelism

The query optimizer only creates a parallel plan and is not involved in deciding the MAXDOP value. This value can be configured at the server level and overridden at the query level via a query hint. Even if the query optimizer creates a parallel query plan, the execution engine might decide to only use a single CPU, based on the resources available when it is time to execute the query.

In earlier versions of SQL Server, it was common to disable parallel queries on systems that were primarily used for transaction processing. This limitation was implemented by adjusting the server setting for MAXDOP to the value 1. In current versions of SQL Server, this is no longer generally considered a good practice.

32-bit vs. 64-bit Servers



Virtual Address Space (VAS) is the total amount of memory that an application like SQL Server could possibly refer to in Windows. The VAS depends upon the configuration of the server.

32-bit Systems

These systems have a VAS of 4 GB. By default, half the address space (2 GB) is reserved for the system and is known as kernel mode address space. The other half of the VAS (2 GB) is available for the application to use and is known as the user mode address space. It is possible to change this proportion by using a /3 GB switch in the boot.ini file of the operating system

(on earlier operating systems) or by using the bcdedit program to edit the Boot Configuration Datastore (on more recent operating systems). Once the /3 GB switch has been configured, 1 GB of the VAS is allocated for the kernel with the remaining 3 GB allocated for applications.

Note: More fine-grained control of the allocated space can be achieved by using the /USERVA switch instead of the /3 GB switch. The /USERVA switch allows the configuration of any value between 2 GB and 3 GB for user applications.

64-bit Systems

Database systems tend to work with large amounts of data and need substantial amounts of memory, so that the systems can serve significant numbers of concurrent users. SQL Server needs large amounts of memory for caching queries and data pages. A limit of 2 GB to 3 GB for VAS is a major constraint on most current systems. As is the case with most database engines, SQL Server is best installed on a 64-bit version of Windows, instead of a 32-bit version.

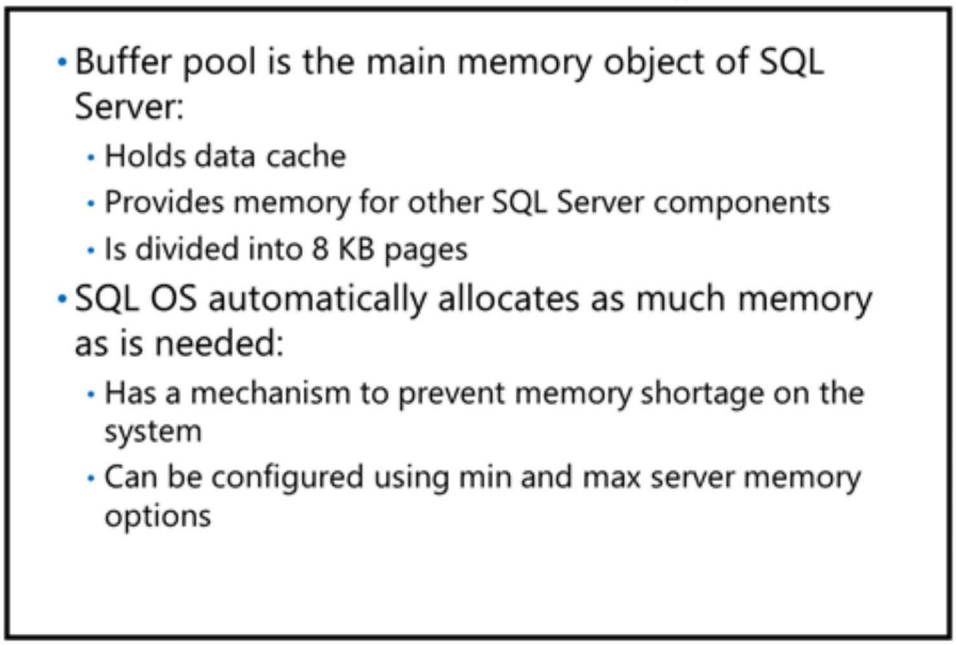
It is best to install a 64-bit version of SQL Server on a 64-bit version of Windows. Full 64-bit systems offer a VAS of 8 TB.

It is possible (but less desirable) to install a 32-bit version of SQL Server on a 64-bit version of Windows. This configuration provides a full 4 GB of VAS for the 32-bit applications and is based on the Windows on Windows (WOW) emulation technology.

64-bit System Limitations

Not all current systems can be implemented as 64-bit systems. The most common limitation is the availability of data providers. If special data providers are needed to connect SQL Server to other systems (particularly through the linked servers feature), it is important to check that the required data providers are available in 64-bit versions. For example, the Jet engine is currently only available as a 32-bit provider.

Overview of SQL Server Memory



The main memory object of SQL Server is known as the buffer pool. This is divided into 8-KB pages – the same size as data pages within a database. The buffer pool is comprised of three sections:

- Free Pages – pages that are not yet used and are kept to satisfy new memory requests.
- Stolen Pages – pages that are used (formally referred to as stolen) by other SQL Server components such as the query optimizer and storage engine.
- Data Cache: Pages that are used for caching database data pages. All data operations are performed in the data cache. If a query wants to select data from a specific data page, the data page is moved into the data cache first. Also, data modification is only ever performed in memory. Modifications are never performed directly on the data files. Changes that are made to pages result in them being considered dirty. The dirty pages are written to the database by a background process known as a checkpoint.

The data cache implements a least recently used (LRU) algorithm to determine candidate pages to be dropped from the cache as space is needed – after they have been flushed to disk (if necessary) by the checkpoint process. The process that performs the task of dropping pages is known as the lazy writer – this performs two core functions. By removing pages from the buffer cache, the lazy writer attempts to keep sufficient free space in the buffer cache for SQL Server to operate. The lazy writer also monitors the overall size of the buffer cache to avoid taking too much memory from the Windows operating system.

SQL Server Memory and Windows Memory

The memory manager within SQL OS allocates and controls the memory used by SQL Server. It does this by checking the available memory on the Windows system, calculating a target memory value – which is the maximum memory that SQL Server can use at that point in time – to avoid a memory shortage at the Windows operating system level. SQL Server is designed to respond to signals from the Windows operating system that indicate a change in memory needs.

As long as SQL Server stays within the target memory, it requests additional memory from Windows when needed. If the target memory value is reached, the memory manager answers requests from components by freeing up the memory of other components. This can involve evicting pages from caches. The target memory value can be controlled via the “min” and “max” server memory options.

Physical vs. Logical I/O

I/O Type	Description
Physical I/O	Physical I/O occurs when the requested page is not available in buffer cache and must be read from the data file into the buffer cache before the requested page can be supplied or when a changed page is written to the data file
Logical I/O	Logical I/O occurs when the requested page is available in the buffer cache

A logical I/O occurs when a task can retrieve a data page from the buffer cache. When the requested page is not present in the buffer cache, it must first be read into the buffer cache from the database. This database read operation is known as a physical I/O.

Physical I/O Minimization

From an overall system performance point of view, two I/O aspects must be minimized:

- You need to minimize the number of physical I/O operations.
- You need to minimize the time taken by each I/O operation that is still required.

Minimizing the number of physical I/O operations is accomplished by:

- Providing enough memory for the data cache.
- Optimizing the physical and logical database layout, including indexes.
- Optimizing queries to request as few I/O operations as possible.

One of the major goals of query optimization is to reduce the number of logical I/O operations. The side effect of this is a reduction in the number of physical I/O operations.

Note: Logical I/O counts can be difficult to interpret as certain operations can cause the counts to be artificially inflated, due to multiple accesses to the same page. However, in general, lower counts are better than higher counts.

Monitoring Query I/O Operations

Both logical and physical I/O operation counts can be seen for each query by setting the following connection option:

```
SET STATISTICS IO ON;
```

The overall physical I/O operations occurring on the system can be seen by querying the sys.dm_io_virtual_file_stats system function. The values returned by this function are cumulative from the point that the system was last restarted.

Demonstration: CPU and Memory Configurations in SSMS

In this demonstration, you will see how to review and configure SQL Server CPU and memory.

Demonstration Steps

1. Ensure that the MSL-TMG, MIA-DC, and MIA-SQL virtual machines are running and log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. On the taskbar, click the **SQL Server Management Studio** shortcut.
3. In the **Connect to Server** dialog box, click **Connect**.
4. On the **File** menu, point to **Open**, and then click **File**.
5. In the **Open File** dialog box, navigate to **D:\Demofiles\AppendixB**, click **CPUAndMemory.sql** and then click **Open**.
6. Follow the instructions contained in the comments of the script file.
7. Close SQL Server Management Studio without saving changes.

Lesson 2: Upgrading and Automating Installation

Rather than installing SQL Server, you sometimes need to upgrade from an earlier version. In this lesson, you will see the benefits and limitations of the available methods for performing upgrades.

Not every installation of SQL Server is performed individually by an administrator. In larger enterprises, there is a need to be able to install many instances of SQL Server – and in a very consistent manner. Unattended installation options are provided for SQL Server and can be used to satisfy the requirements of this scenario. They can also be performed in a silent mode. This enables the installation of SQL Server to be performed during the installation of other applications.

Lesson Objectives

After completing this lesson, you will be able to:

- Upgrade SQL Server.

- Apply SQL Server hotfixes, cumulative updates, and service packs.
- Perform unattended installations.

Upgrading SQL Server

In-place Upgrade	Side-by-side Upgrade
Easy, mostly automated	More granular control over process
System data upgraded	Use to perform test migration
No additional hardware	Relatively straightforward rollback
Apps pointing to same names	Can leverage failover/switchover

- Side-by-side installs of some SQL Server versions have additional considerations:
  - Shared components are upgraded to the latest version if the major build of the two installations is the same (that is, SQL Server 2008 and SQL Server 2008 R2)

There are two basic ways that SQL Server upgrades can be performed – there is no single preferred way to do this. Each method has benefits and limitations, and is appropriate in certain circumstances.

In-place Upgrades

In-place upgrades occur when the installed version of SQL Server is directly replaced by a new version. This is an easier and highly automated, though riskier, method of upgrading. If an upgrade fails, it is much harder to return to the previous operating state. For most customers, the risk of this cannot be ignored.

When you are weighing up risk, you need to consider that it may not be the SQL Server upgrade that fails. Even if the SQL Server upgrade works as expected, but then the application fails to operate as anticipated on the new version of SQL Server, the need to recover the situation quickly will be just as important.

In-place upgrades have the added advantage of minimizing the need for additional hardware resources and avoid the need to redirect client applications that are configured to work with

the existing server.

Side-by-side Upgrades

Side-by-side upgrades are a safer alternative, as the original system is left in place and can be quickly returned to production should an upgrade issue arise. However, side-by-side upgrades involve extra work and more hardware resources.

To perform a side-by-side upgrade, you will need enough hardware resources to provide for both the original and the new systems. Two common risks associated with side-by-side upgrades relate to the time taken to copy all the user databases to a new location and the space required to hold these copies.

While most side-by-side upgrades are performed on separate servers, it is possible to install both versions of SQL Server on the same server during a side-by-side upgrade. However, side-by-side upgrades of versions with the same major build number (that is, SQL Server 2008 R2 and SQL Server 2008) on the same server are a special case. Because the major version number is identical, separate versions of the shared components cannot co-exist on the same server. Shared components will be upgraded.

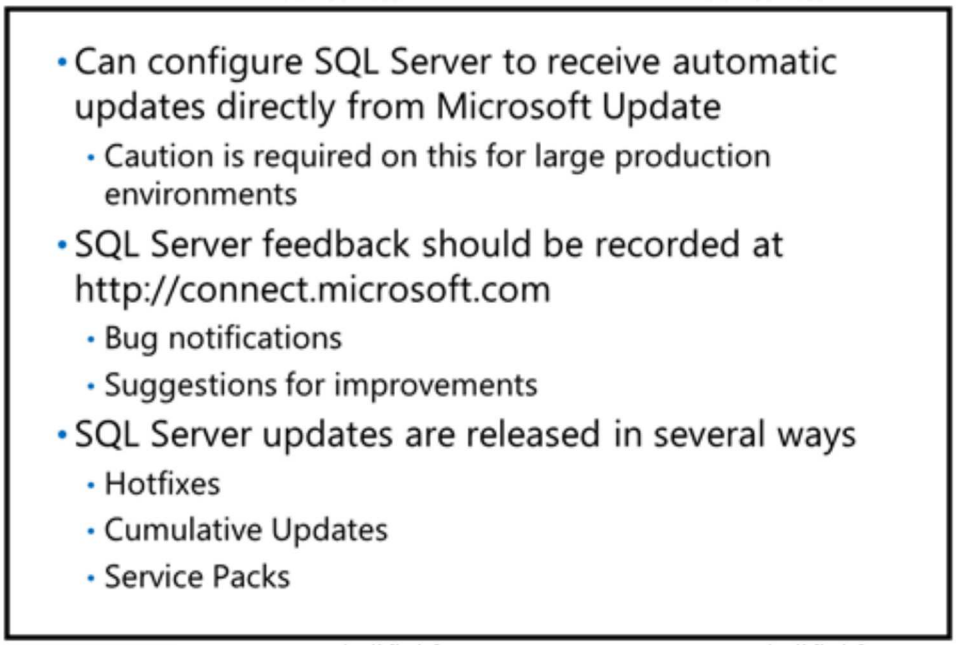
Not all versions of SQL Server are supported when installed side-by-side. Consult SQL ServerBooks Online for a matrix of versions that are supported when installed together.

Hybrid Options

It is also possible to use some elements of an in-place upgrade and a side-by-side upgrade together. For example, rather than copying all the user databases, after installing the new version of SQL Server beside the old version, and migrating all the server objects such as logins, you could detach user databases from the old server instance and reattach them to the new one.

Once user databases have been attached to a newer version of SQL Server, they cannot be reattached to an older version again, even if the database compatibility settings have not been upgraded. This is a risk that needs to be considered when using a hybrid approach.

SQL Server Servicing



As with all software products over time, issues can be encountered with SQL Server. The product group is very responsive in fixing any identified issues. If there are issues that you wish to notify Microsoft about, or suggestions for how the product might be improved, please visit **http://connect.microsoft.com**.

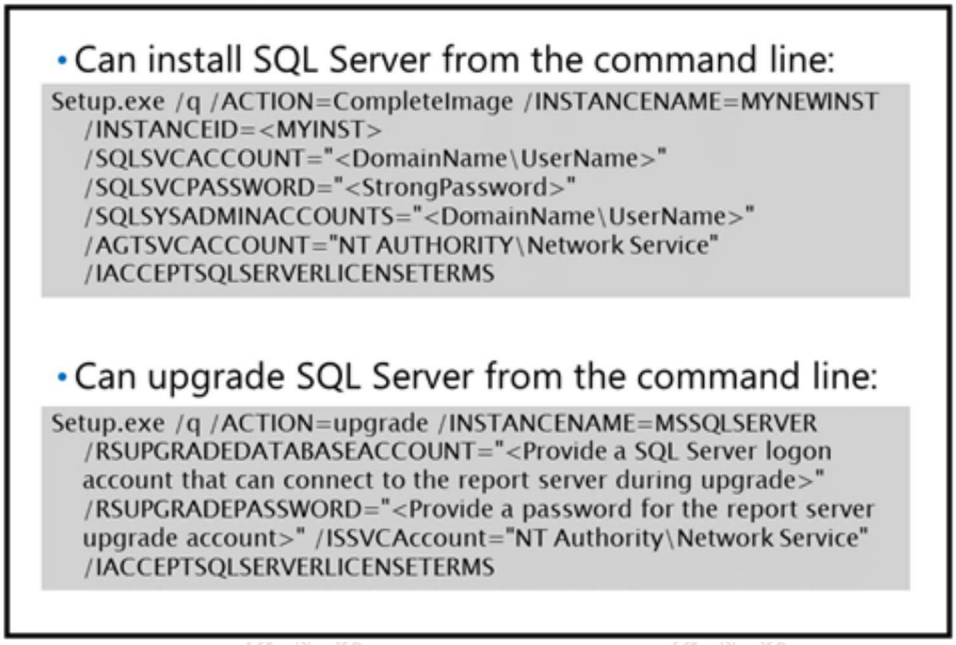
The simplest way to keep SQL Server up to date is to enable automatic updates from the Microsoft Update service. Larger organizations or those with strong change processes should exert caution in applying automatic updates. It is likely that the updates should be applied to test or staging environments before being applied to production environments.

SQL Server updates are released in several ways:

- Hotfixes (also known as QFE or Quick Fix Engineering) are released to address urgent customer concerns. Due to the tight time constraints, only limited testing can be performed on these fixes, so they should only be applied to systems that are known to be experiencing the issues that they address.
- Cumulative Updates (CUs) are periodic roll-up releases of hotfixes that have received further testing as a group.
- Service Packs (SPs) are periodic releases where full regression testing has been performed. Microsoft recommend applying SPs to all systems after appropriate levels of organizational testing.

SQL Server 2008 R2 and later can also have product SPs slipstreamed into the installation process to avoid the need to apply them after installation.

Unattended Installation



In many organizations, script files for standard builds of software installations are created by senior IT administrators and used to ensure consistency throughout the organization. Unattended installations can help with the deployment of multiple identical installations of SQL Server across an enterprise. Unattended installations can also provide for the delegation of the installation to another person.

Unattended Installation Methods

One option for performing an unattended installation of SQL Server is to create an .ini file containing the required setup information and passing it as a parameter to setup.exe at a command prompt. A second alternative is to pass all the required SQL Server setup details as parameters to the setup.exe program, rather than placing the parameters into an .ini file. You can choose to use the .ini file or the separate parameters, but a combination of both is not permitted.

In both examples on the slide, the second method has been used. The first example shows a typical installation command and the second shows how an upgrade could be performed using the same method.

/q Switch

The "/q" switch shown in the examples specifies "quiet mode" – no user interface is provided. An alternative switch "/qs" specifies "quiet simple" mode. In the quiet simple mode, the installation runs and shows progress in the UI but does not accept any input.

Creating an .ini File

An .ini file for unattended installation can be created by using any text editor, such as Notepad. The SQL Server installation program creates a file called ConfigurationFile.ini in a folder that is named based upon the installation date and time, under the folder C:\Program Files\Microsoft SQL Server\110\Setup Bootstrap\Log. You can use this as a starting point for creating your own .ini file. The .ini file is composed of a single [Options] section containing multiple parameters, each relating to a different feature or configuration setting.

Demonstration: Reviewing an Unattended Installation File

In this demonstration, you will see an unattended installation file that can be used to configure SQL Server.

Demonstration Steps

1. Ensure that the MSL-TMG, MIA-DC, and MIA-SQL virtual machines are running, and log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. On the taskbar, click the **SQL Server Management Studio** shortcut.
3. In the **Connect to Server** dialog box, click **Connect**.
4. On the **File** menu, point to **Open**, and then click **File**.
5. In the **Open File** dialog box, navigate to **D:\Demofiles\AppendixB**, click **ConfigurationFile.ini**, and then click **Open**.
6. Review the contents of the configuration file, in particular noting the values of the following properties:

    - INSTANCEID

- ACTION
- FEATURES
- QUIET
- QUIETSIMPLE
- INSTALLSHAREDDIR
- INSTANCEDIR
- INSTANCENAME
- AGTSVCSTARTUPTYPE
- SQLCOLLATION
- SQLSVCACCOUNT
- SQLSYSADMINACCOUNTS
- TCPENABLED

7. Close SQL Server Management Studio without saving changes.

Module Review and Takeaways

Best Practice:

1. Understand the architecture of SQL Server.

Review Question(s)